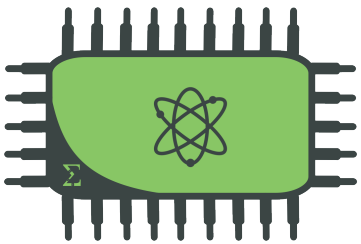
Curso de Modelagem de Circuitos Digitais em VHDL - Aula 1

Alceu Bernardes Castanheira de Farias[1]

[1]Semana Universitária
Universidade de Brasília, campus Gama - FGA

Códigos e slides do curso disponíveis aqui:
https://github.com/alceu-castanheira/Curso-VHDL-Eletronjun-2020

22 de setembro de 2020



eletronjun
Engenharia Eletrônica Júnior

Sumário

Sumário

O que é VHDL?

- VHDL é uma sigla derivada de *Very High Speed Integrated Circuits* (VHSIC) *Hardware Description Language*;

O que é VHDL?

- VHDL é uma sigla derivada de *Very High Speed Integrated Circuits* (VHSIC) *Hardware Description Language*;
- Linguagem de descrição de *hardware*.

O que é VHDL?

- VHDL é uma sigla derivada de *Very High Speed Integrated Circuits* (VHSIC) *Hardware Description Language*;
- Linguagem de descrição de *hardware*.

Definição de linguagem de descrição de hardware

É uma linguagem que permite descrever o comportamento de um circuito/sistema eletrônico, de forma que seja possível implementar o circuito posteriormente.

VHDL

- VHDL permite descrever o comportamento de um circuito eletrônico e pode ser utilizado tanto para síntese como para simulação;

VHDL

- VHDL permite descrever o comportamento de um circuito eletrônico e pode ser utilizado tanto para síntese como para simulação;
- É uma dentre várias linguagens de descrição de *hardware* existentes, como:

VHDL

- VHDL permite descrever o comportamento de um circuito eletrônico e pode ser utilizado tanto para síntese como para simulação;
- É uma dentre várias linguagens de descrição de *hardware* existentes, como:
  - Verilog;
  - Handel-C;
  - SystemVerilog;
  - SystemC;
  - AHDL;
  - Verilog-AMS.

VHDL - Vantagens

- Facilidade para descrever funcionalidades complexas de *hardware*;

VHDL - Vantagens

- Facilidade para descrever funcionalidades complexas de *hardware*;
- Funções que implementam portas lógicas e circuitos combinacionais de maneira muito simples;

VHDL - Vantagens

- Facilidade para descrever funcionalidades complexas de *hardware*;
- Funções que implementam portas lógicas e circuitos combinacionais de maneira muito simples;
- O detalhamento da lógica de controle de sistemas sequenciais é feito de forma automática;

VHDL - Vantagens

- Facilidade para descrever funcionalidades complexas de *hardware*;
- Funções que implementam portas lógicas e circuitos combinacionais de maneira muito simples;
- O detalhamento da lógica de controle de sistemas sequenciais é feito de forma automática;
- Portabilidade:

VHDL - Vantagens

- Facilidade para descrever funcionalidades complexas de *hardware*;
- Funções que implementam portas lógicas e circuitos combinacionais de maneira muito simples;
- O detalhamento da lógica de controle de sistemas sequenciais é feito de forma automática;
- Portabilidade:
  - » É comum, na indústria, o uso de FPGAs para protótipos iniciais em projetos que posteriormente possam ser implementados em ASICs (*Application Specific Integrated Circuits*).

Linguagem de descrição de *hardware* X Linguagem de programação

Linguagem de descrição de *hardware:*

- Descreve um circuito eletrônico;

Linguagem de programação:

Linguagem de descrição de *hardware* X Linguagem de programação

Linguagem de descrição de *hardware:*

- Descreve um circuito eletrônico;

Linguagem de programação:

- Descreve um *software*;

Linguagem de descrição de *hardware* X Linguagem de programação

Linguagem de descrição de *hardware:*

- Descreve um circuito eletrônico;
- Instruções naturalmente paralelas;

Linguagem de programação:

- Descreve um *software*;

Linguagem de descrição de *hardware* X Linguagem de programação

Linguagem de descrição de *hardware:*

- Descreve um circuito eletrônico;
- Instruções naturalmente paralelas;

Linguagem de programação:

- Descreve um *software*;
- Instruções naturalmente sequenciais;

Linguagem de descrição de *hardware* X Linguagem de programação

Linguagem de descrição de *hardware:*

- Descreve um circuito eletrônico;
- Instruções naturalmente paralelas;
- Passa pelas etapas de síntese e implementação;

Linguagem de programação:

- Descreve um *software*;
- Instruções naturalmente sequenciais;

Linguagem de descrição de *hardware* X Linguagem de programação

Linguagem de descrição de *hardware:*

- Descreve um circuito eletrônico;
- Instruções naturalmente paralelas;
- Passa pelas etapas de síntese e implementação;

Linguagem de programação:

- Descreve um *software*;
- Instruções naturalmente sequenciais;
- Passa por etapa de compilação;

Linguagem de descrição de *hardware* X Linguagem de programação

Linguagem de descrição de *hardware:*

- Descreve um circuito eletrônico;
- Instruções naturalmente paralelas;
- Passa pelas etapas de síntese e implementação;
- Gera um arquivo *bitstream*, que mapeia o circuito no dispositivo.

Linguagem de programação:

- Descreve um *software*;
- Instruções naturalmente sequenciais;
- Passa por etapa de compilação;

Linguagem de descrição de *hardware* X Linguagem de programação

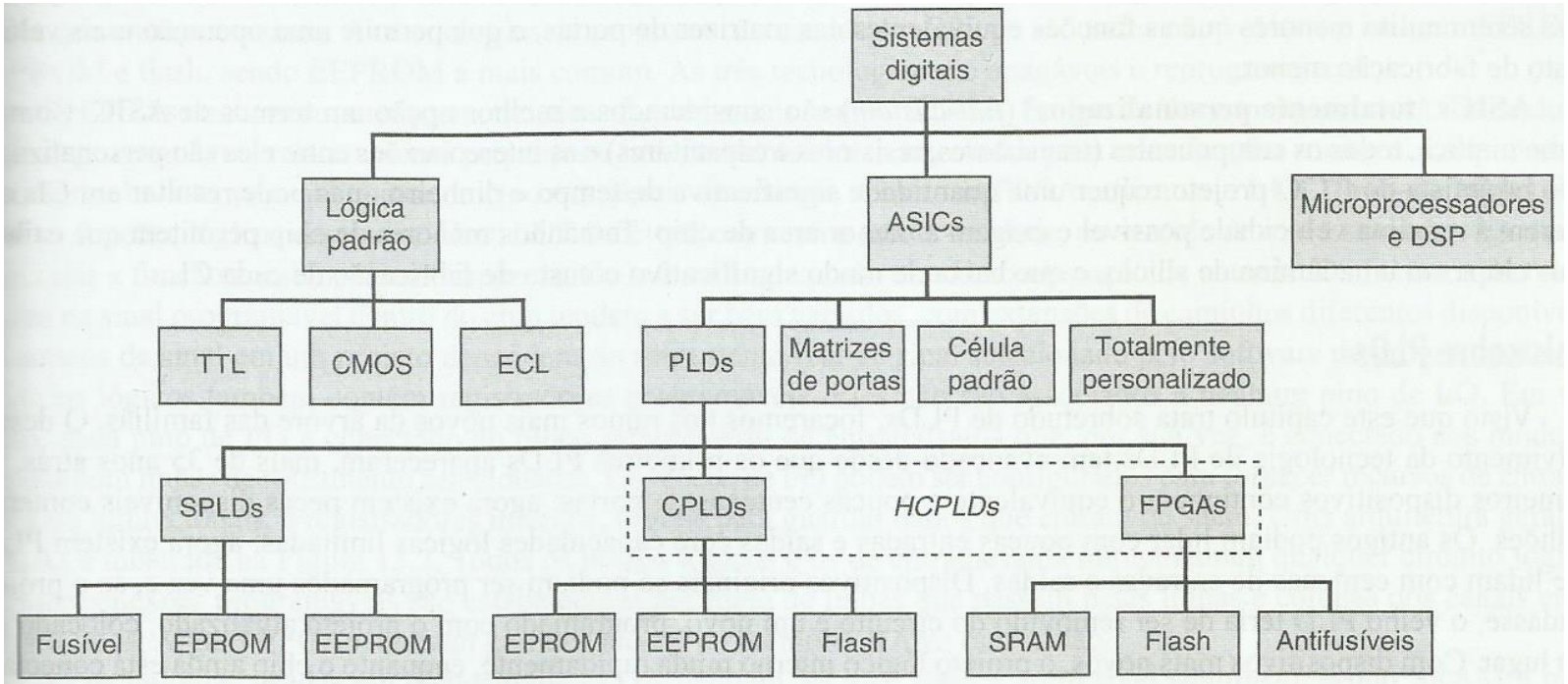
Linguagem de descrição de *hardware:*

- Descreve um circuito eletrônico;
- Instruções naturalmente paralelas;
- Passa pelas etapas de síntese e implementação;
- Gera um arquivo *bitstream*, que mapeia o circuito no dispositivo.

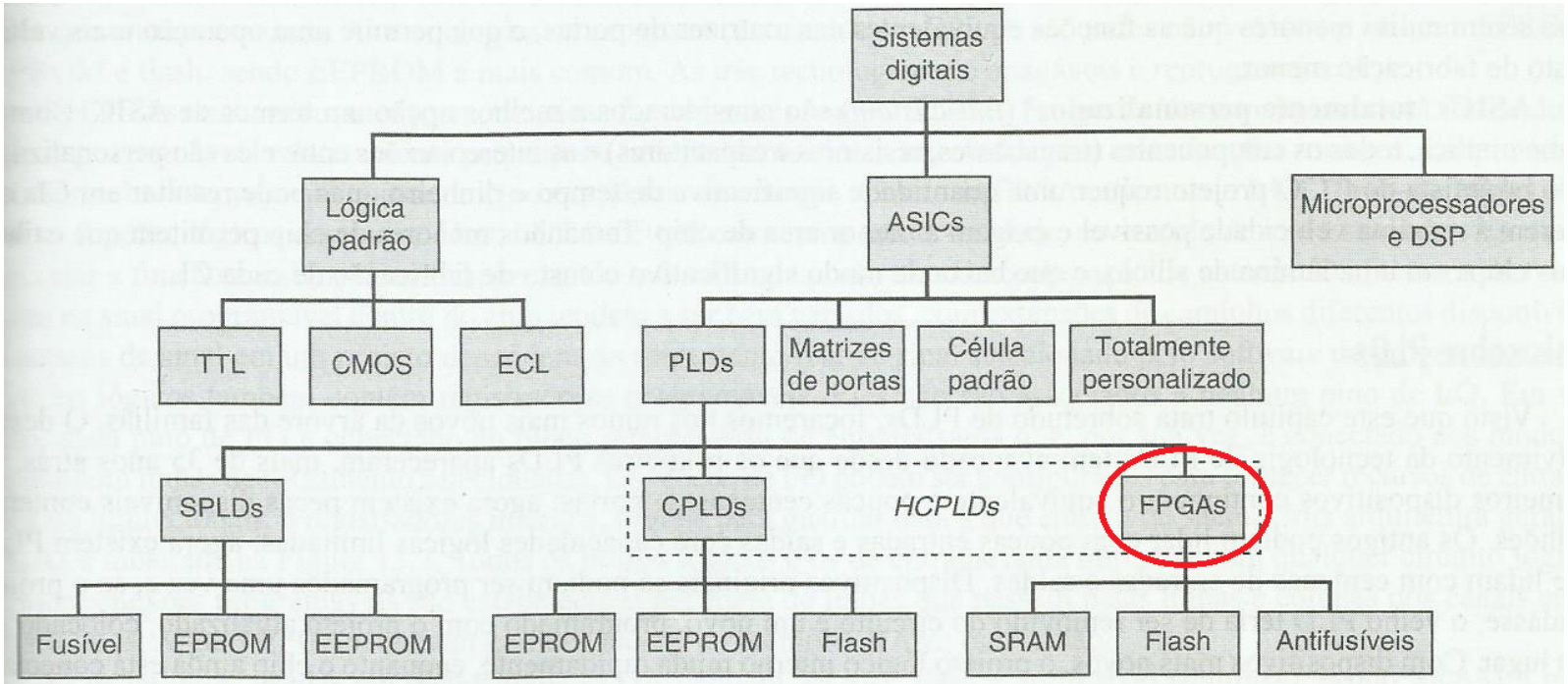
Linguagem de programação:

- Descreve um *software*;
- Instruções naturalmente sequenciais;
- Passa por etapa de compilação;
- Gera um arquivo executável, que permite que o *software* "rode"no dispositivo.

Dispositivos Lógicos Programáveis



Dispositivos Lógicos Programáveis



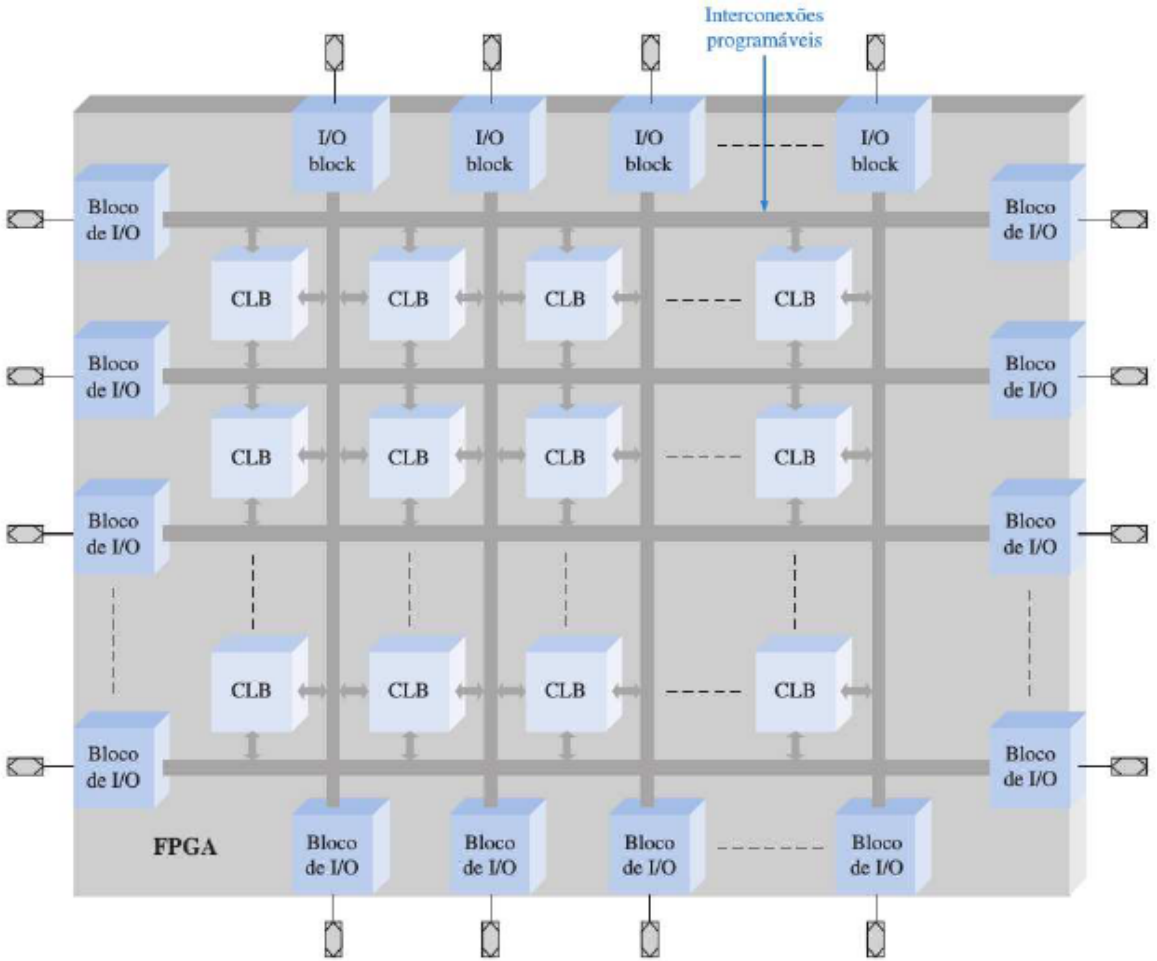
O que é um FPGA?

O que é um FPGA?

Definição de FPGA

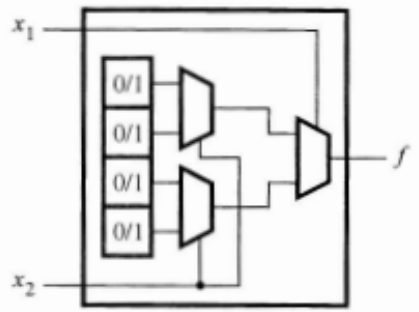
FPGA (*Field Programmable Gate Array*) é um dispositivo lógico programável que permite a implementação de circuitos lógicos relativamente grandes, implementando as funções desejadas com o uso de blocos lógicos.

FPGA - Estrutura Interna



FPGA - Estrutura Interna

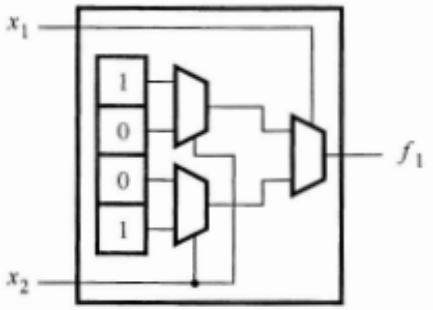
- Os blocos lógicos são compostos por:
  - LUTs (*Look Up Tables*);
  - Multiplexadores;
  - *Flip-flops*.
- Tamanhos e quantidades de elementos podem variar de acordo com a arquitetura e modelo de FPGA.



(a) Circuit for a two-input LUT

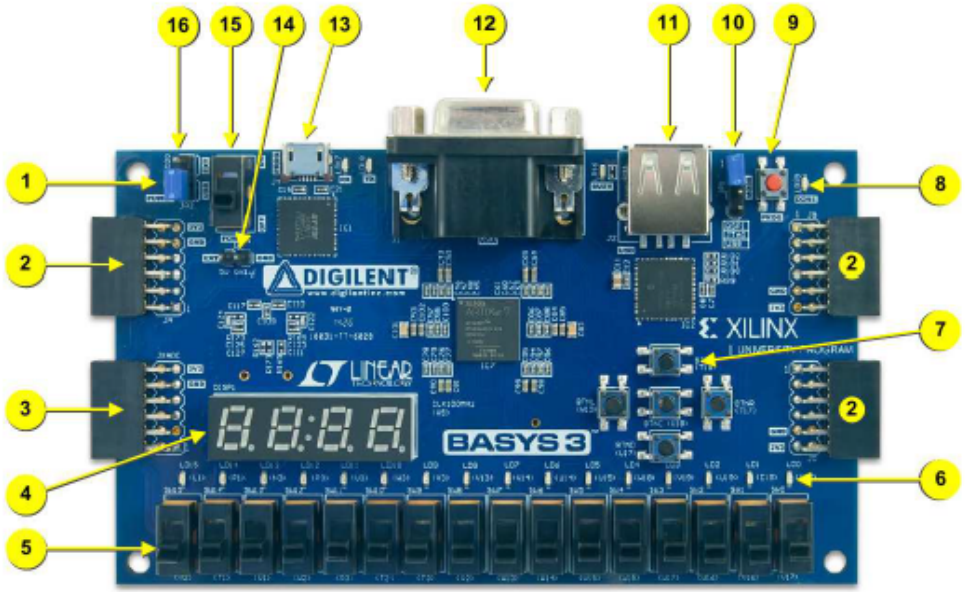
x_1	x_2	f_1
0	0	1
0	1	0
1	0	0
1	1	1

(b) $f_1 = \bar{x}_1\bar{x}_2 + x_1x_2$



(c) Storage cell contents in the LUT

Basys 3: Recursos



	Descrição do componente		*Descrição do componente*
1	LED da alimentação	9	Botão de reset da FPGA
2	Conectores Pmod	10	Jumper de modo de programação
3	Conector Pmod sinal analógico (XADC)	11	Conector host USB
4	Displays 7-segmentos (4)	12	Conector VGA
5	Chaves (16)	13	Porta partilhada UART/JTAG
6	LEDs (16)	14	Conector para alimentação externa
7	*Pushbuttons* (5)	15	Chave de alimentação
8	LED de programação completa	16	Jumper de seleção de alimentação

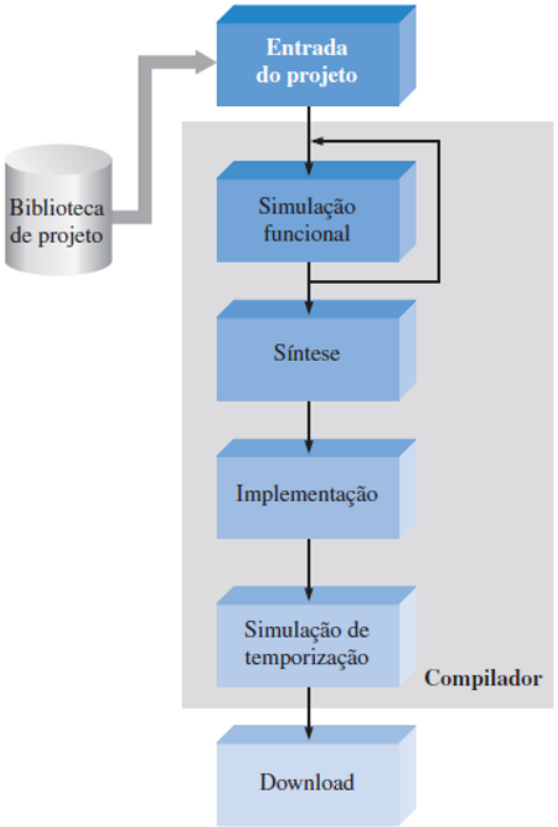
Basys 3: Recursos

- 33280 células lógicas - CLBs (com 4 LUTs de 6 entradas e 8 *flip-flops* cada);
- 1800 KB de memória RAM;
- 90 blocos de DSP (*Digital Signal Processors*);
- Clock interno de 100 MHz, com capacidade de frequência de até 450 MHz.

Ferramentas de desenvolvimento

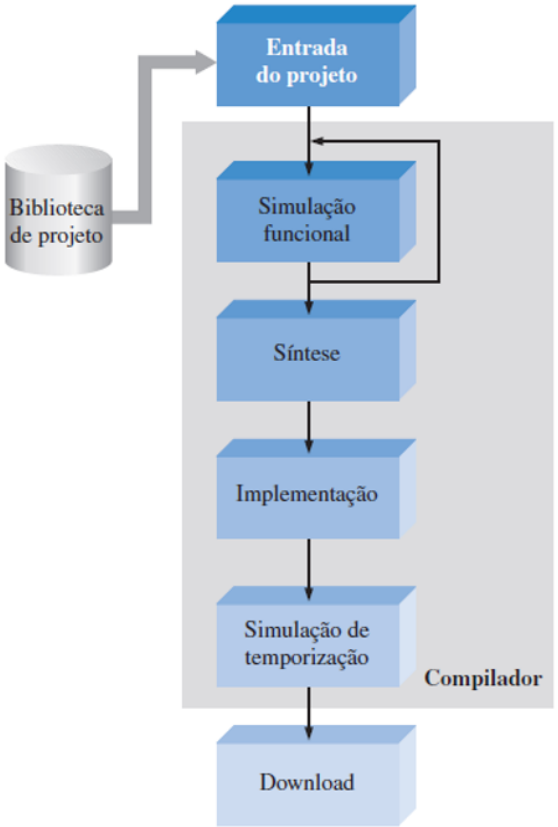
- Altera (Quartus II);
- Cadence (NCLaunch e NC-Sim);
- Mentor Graphics (Leonardo Spectrum e ModelSim);
- Synplicity (Synplify);
- Xilinx (ISE Design Suite e Vivado).

Fluxo de desenvolvimento



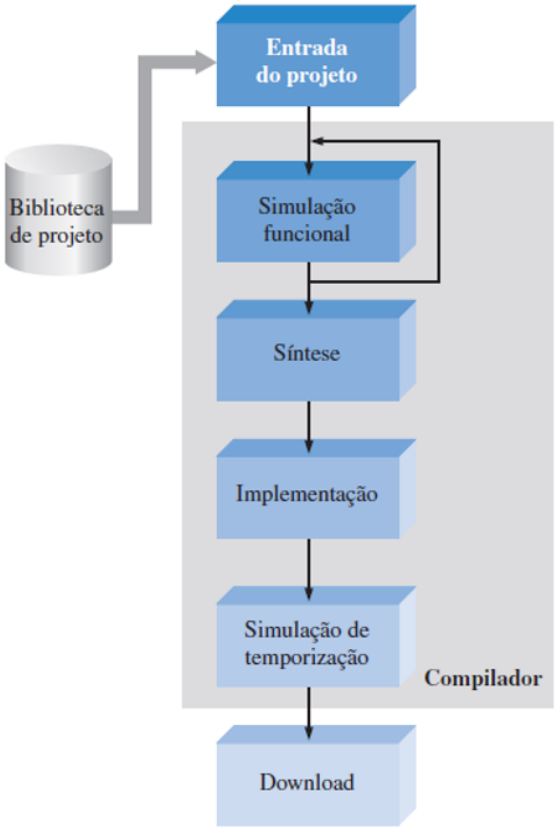
- **Entrada do projeto:** Descrição de um circuito por meio de um código VHDL;

Fluxo de desenvolvimento



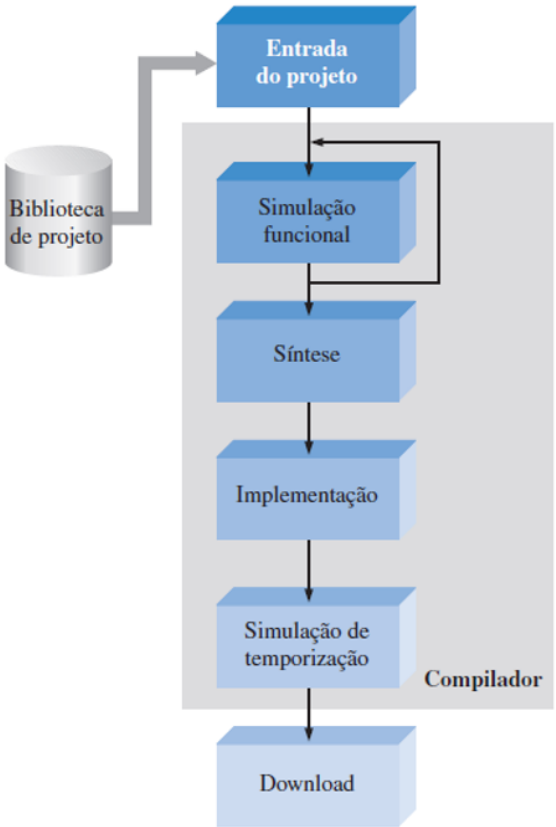
- **Entrada do projeto:** Descrição de um circuito por meio de um código VHDL;
- **Simulação funcional:** Teste do funcionamento do circuito sem levar em consideração condições de temporização;

Fluxo de desenvolvimento



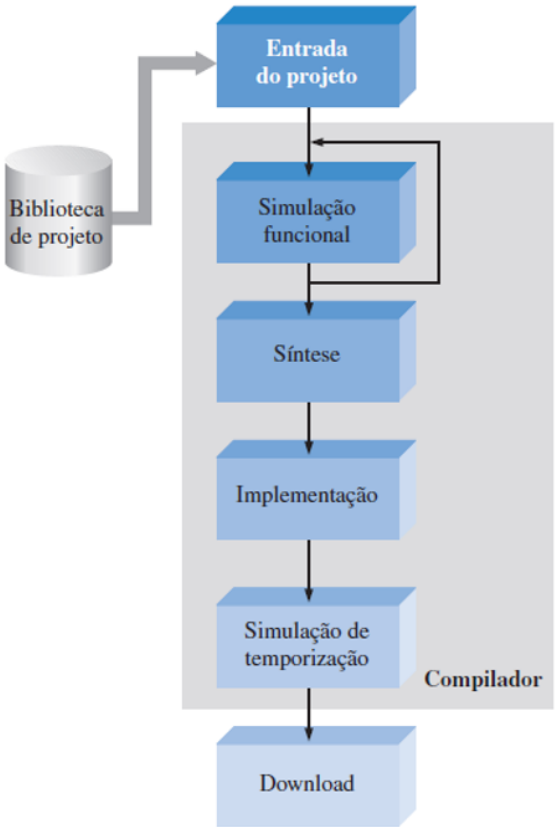
- **Entrada do projeto:** Descrição de um circuito por meio de um código VHDL;
- **Simulação funcional:** Teste do funcionamento do circuito sem levar em consideração condições de temporização;
- **Síntese:** Geração de um *netlist*, uma lista de componentes necessários para implementar o circuito.

Fluxo de desenvolvimento



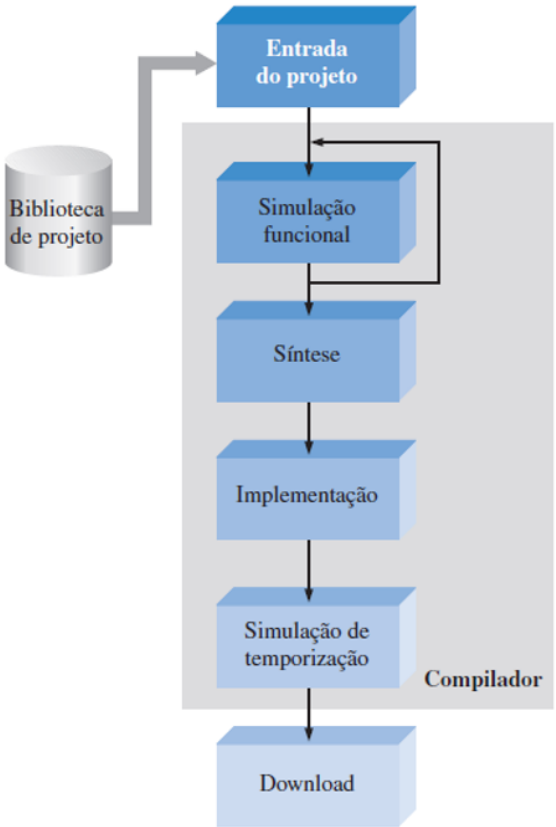
- **Implementação:** mapeamento de toda a estrutura lógica definida na etapa anterior para que o circuito possa ser implementado no FPGA;

Fluxo de desenvolvimento



- **Implementação:** mapeamento de toda a estrutura lógica definida na etapa anterior para que o circuito possa ser implementado no FPGA;
- **Simulação de temporização:** testa o funcionamento do circuito mediante condições de temporização;

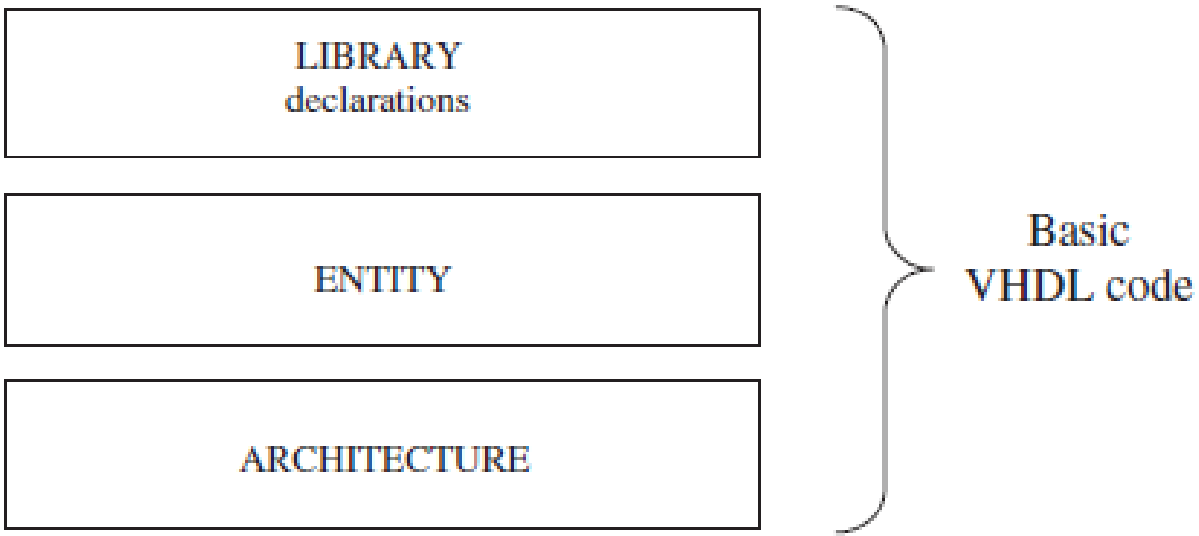
Fluxo de desenvolvimento



- **Implementação:** mapeamento de toda a estrutura lógica definida na etapa anterior para que o circuito possa ser implementado no FPGA;
- **Simulação de temporização:** testa o funcionamento do circuito mediante condições de temporização;
- ***Download***: gera um *bits-tream*, que é baixado no FPGA para implementar o circuito desenvolvido.

Sumário

Estrutura de um código VHDL



Libraries

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

- Coleção de trechos de código amplamente utilizados.

Libraries

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

- Coleção de trechos de código amplamente utilizados.
- Podem ser reutilizados e aproveitadas em vários códigos VHDL;

Libraries

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

- Coleção de trechos de código amplamente utilizados.
- Podem ser reutilizados e aproveitadas em vários códigos VHDL;
- Declarada no início do código.

Library IEEE

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

- Duas das principais bibliotecas utilizadas em VHDL são:

Library IEEE

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

- Duas das principais bibliotecas utilizadas em VHDL são:
  - **std_logic_1164:** Especifica os tipos *std_logic* e *std_logic_vector*.

Library IEEE

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

- Duas das principais bibliotecas utilizadas em VHDL são:
  - **std_logic_1164:** Especifica os tipos *std_logic* e *std_logic_vector*.
  - **numeric_std:** Especifica os tipos *signed* e *unsigned*, além de funções para realizar operações aritméticas e de comparação. Contém várias vários de conversão de tipos em VHDL.

Entities

```
ENTITY entity_name IS
    PORT (
        port_name : signal_mode signal_type;
        port_name : signal_mode signal_type;
        ...);
END entity_name;
```

- Especificação com todas as entradas e saídas do circuito;.

Entities

```
ENTITY entity_name IS
    PORT (
        port_name : signal_mode signal_type;
        port_name : signal_mode signal_type;
        ...);
END entity_name;
```

- Especificação com todas as entradas e saídas do circuito;.
- Declaração após a seção *library*.

Entities

```
ENTITY entity_name IS
   PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END entity_name;
```

- **signal_mode:** *in*, *out*;

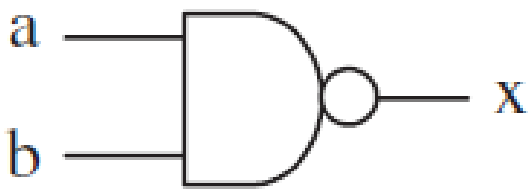
Entities

```
ENTITY entity_name IS
    PORT (
        port_name : signal_mode signal_type;
        port_name : signal_mode signal_type;
        ...);
END entity_name;
```

- **signal_mode:** *in*, *out*;
- **signal_type:** std_logic/std_logic_vector, signed/unsigned, integer.

Entity: exemplo

```
ENTITY nand_gate IS
    PORT (a, b : IN STD_LOGIC;
          x : OUT STD_LOGIC);
END nand_gate;
```



Architectures

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

- Descrição de como o circuito deve funcionar;

Architectures

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

- Descrição de como o circuito deve funcionar;
- Composto por duas partes:

Architectures

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

- Descrição de como o circuito deve funcionar;
- Composto por duas partes:
  - **Declarativa:** Opcional. Declaração de *signals* e *constants* para auxiliar na implementação de lógica do circuito;

Architectures

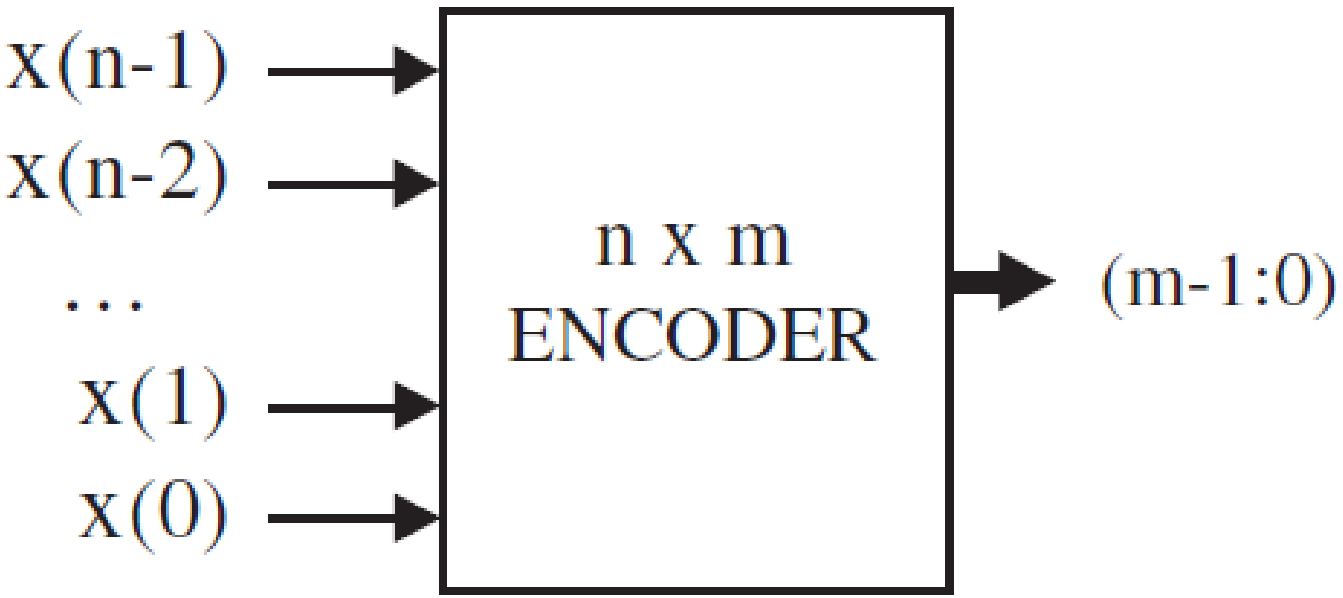
```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

- Descrição de como o circuito deve funcionar;
- Composto por duas partes:
  - **Declarativa:** Opcional. Declaração de *signals* e *constants* para auxiliar na implementação de lógica do circuito;
  - **Código:** Código que representa a descrição do comportamento do circuito em VHDL.

Architecture: exemplo

```
ARCHITECTURE myarch OF nand_gate IS
BEGIN
    x <= a NAND b;
END myarch;
```

Exemplo: *encoder*



Exemplo: *encoder*

```
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY encoder IS
6     PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
8  END encoder;
9  ---------------------------------------------
10 ARCHITECTURE encoder2 OF encoder IS
11 BEGIN
12    WITH x SELECT
13       y <=   "000" WHEN "00000001",
14              "001" WHEN "00000010",
15              "010" WHEN "00000100",
16              "011" WHEN "00001000",
17              "100" WHEN "00010000",
18              "101" WHEN "00100000",
19              "110" WHEN "01000000",
20              "111" WHEN "10000000",
21              "ZZZ" WHEN OTHERS;
22 END encoder2;
23 ---------------------------------------------
```

Exemplo: *encoder*

```
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY encoder IS
6     PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
8  END encoder;
9  ---------------------------------------------
10 ARCHITECTURE encoder2 OF encoder IS
11 BEGIN
12    WITH x SELECT
13       y <=   "000" WHEN "00000001",
14              "001" WHEN "00000010",
15              "010" WHEN "00000100",
16              "011" WHEN "00001000",
17              "100" WHEN "00010000",
18              "101" WHEN "00100000",
19              "110" WHEN "01000000",
20              "111" WHEN "10000000",
21              "ZZZ" WHEN OTHERS;
22 END encoder2;
23 ---------------------------------------------
```

Exemplo: *encoder*

```
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY encoder IS
6     PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
8  END encoder;
9  ---------------------------------------------
10 ARCHITECTURE encoder2 OF encoder IS
11 BEGIN
12    WITH x SELECT
13       y <=   "000" WHEN "00000001",
14              "001" WHEN "00000010",
15              "010" WHEN "00000100",
16              "011" WHEN "00001000",
17              "100" WHEN "00010000",
18              "101" WHEN "00100000",
19              "110" WHEN "01000000",
20              "111" WHEN "10000000",
21              "ZZZ" WHEN OTHERS;
22 END encoder2;
23 ---------------------------------------------
```

Signals em VHDL

- *Signals* são estruturas utilizadas para alguns fins específicos em VHDL:

Signals em VHDL

- *Signals* são estruturas utilizadas para alguns fins específicos em VHDL:
  - » Armazenar informações temporárias do código;

Signals em VHDL

- *Signals* são estruturas utilizadas para alguns fins específicos em VHDL:
  - » Armazenar informações temporárias do código;
  - » Armazenar dados de saídas para que os mesmos possam ser lidos e utilizados na lógica do código;

Signals em VHDL

- *Signals* são estruturas utilizadas para alguns fins específicos em VHDL:
  - Armazenar informações temporárias do código;
  - Armazenar dados de saídas para que os mesmos possam ser lidos e utilizados na lógica do código;
  - Conectar entradas e saídas de diferentes circuitos;

Signals em VHDL

- *Signals* são estruturas utilizadas para alguns fins específicos em VHDL:
  - Armazenar informações temporárias do código;
  - Armazenar dados de saídas para que os mesmos possam ser lidos e utilizados na lógica do código;
  - Conectar entradas e saídas de diferentes circuitos;
  - Conectar entradas e saídas de um circuito aos sinais de teste do mesmo em um *testbench*.

Signals em VHDL

```
SIGNAL x: STD_LOGIC;
-- x is declared as a one-digit (scalar) signal of type STD_LOGIC.

SIGNAL y: STD_LOGIC_VECTOR (3 DOWNTO 0) := "0001";
-- y is declared as a 4-bit vector, with the leftmost bit being
-- the MSB. The initial value (optional) of y is "0001". Notice
-- that the ":=" operator is used to establish the initial value.
```

Atribuição de valores em VHDL

- *Bits* são retratados com aspas simples.
  - » Ex: '0' ou '1'.
- Vetores de *bits* entre aspas duplas.
  - » Ex: "0100", "110100"
- Atribuição de valores a saídas e sinais é feita através de '<=':
  - » out <= "1000";
  - » **Saída/sinal e valor devem ser do mesmo tamanho e do mesmo tipo**.

Atribuição de valores em VHDL

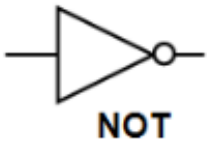
```
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;

BEGIN

c <= d(5);     -- legal (same scalar type: STD_LOGIC)
d(0) <= c;     -- legal (same scalar type: STD_LOGIC)



e <= d;        -- illegal (type mismatch: INTEGER x
               -- STD_LOGIC_VECTOR)
```

Sumário

Portas lógicas

- Blocos fundamentais de sistemas digitais.
- Dispositivos que implementam funções *booleanas* básicas.
  - AND;
  - OR;
  - NOT;
  - NAND;
  - NOR;
  - XOR;
  - XNOR

Portas Lógicas



NOT

Input	Output
I	F
0	1
1	0

AND

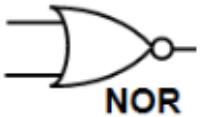
Inputs		Output
A	B	F
0	0	0
1	0	0
0	1	0
1	1	1

NAND

Inputs		Output
A	B	F
0	0	1
1	0	1
0	1	1
1	1	0

OR

Inputs		Output
A	B	F
0	0	0
1	0	1
0	1	1
1	1	1



NOR

Inputs		Output
A	B	F
0	0	1
1	0	0
0	1	0
1	1	0

EXCLUSIVE OR

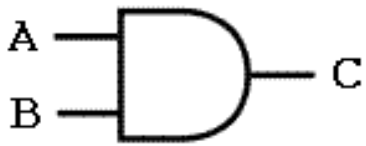
Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE NOR

Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Portas Lógicas em VHDL

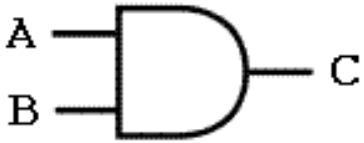
AND



Inputs		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Portas Lógicas em VHDL

AND

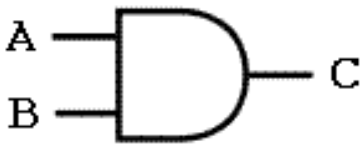


Inputs		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

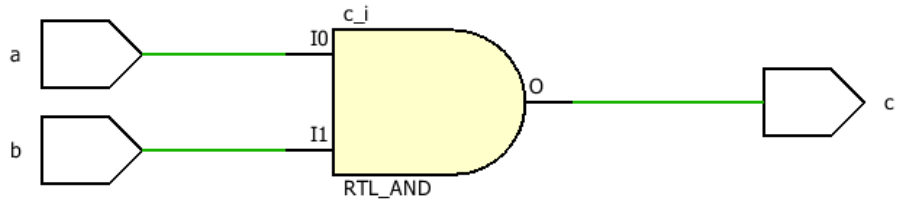
```
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23
24  entity and_gate is
25      Port ( a : in STD_LOGIC;
26             b : in STD_LOGIC;
27             c : out STD_LOGIC);
28  end and_gate;
29
30  architecture Behavioral of and_gate is
31  begin
32
33      c <= a and b;
34
35  end Behavioral;
```

Portas Lógicas em VHDL

AND



Inputs		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

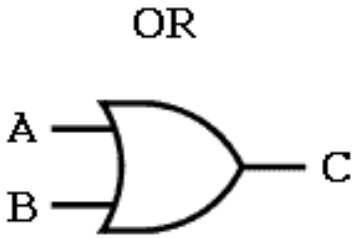


Portas Lógicas em VHDL

OR

Inputs		Output
A	**B**	**C**
0	0	0
0	1	1
1	0	1
1	1	1

Portas Lógicas em VHDL



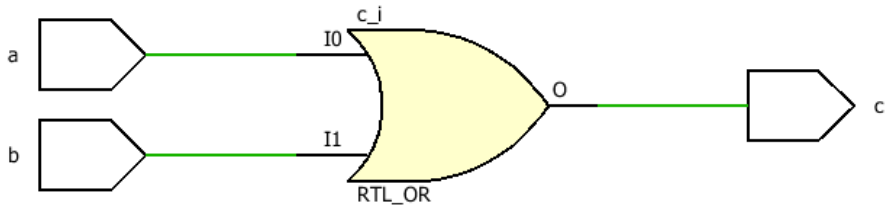
Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  entity gate is
26      Port ( a : in STD_LOGIC;
27             b : in STD_LOGIC;
28             c : out STD_LOGIC);
29  end gate;
30
31  architecture Behavioral of gate is
32  begin
33
34      c <= a or b;
35
36  end Behavioral;
```

Portas Lógicas em VHDL

OR

Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1



Portas Lógicas em VHDL

NOT

Input	Output
A	C
0	1
1	0

Portas Lógicas em VHDL

NOT

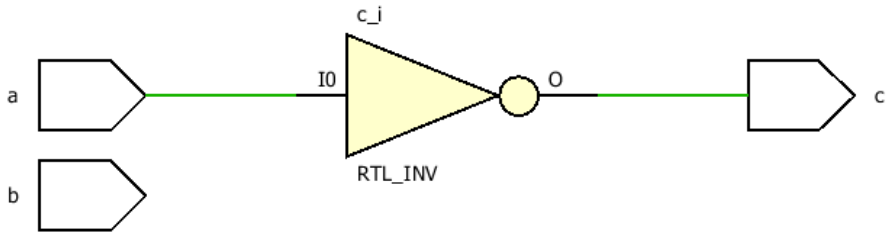
Input	Output
A	C
0	1
1	0

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  entity gate is
26      Port ( a : in STD_LOGIC;
27             b : in STD_LOGIC;
28             c : out STD_LOGIC);
29  end gate;
30
31  architecture Behavioral of gate is
32  begin
33
34      c <= not a;
35
36  end Behavioral;
```

Portas Lógicas em VHDL

NOT

Input	Output
A	C
0	1
1	0



Portas Lógicas em VHDL

NAND

Inputs		Output
A	B	F
0	0	1
1	0	1
0	1	1
1	1	0

Portas Lógicas em VHDL

NAND

Inputs		Output
A	B	F
0	0	1
1	0	1
0	1	1
1	1	0

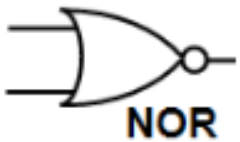
```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  entity gate is
26      Port ( a : in STD_LOGIC;
27             b : in STD_LOGIC;
28             f : out STD_LOGIC);
29  end gate;
30
31  architecture Behavioral of gate is
32  begin
33
34      f <= a nand b;
35
36  end Behavioral;
```

Portas Lógicas em VHDL

NAND

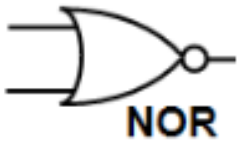
Inputs		Output
A	B	F
0	0	1
1	0	1
0	1	1
1	1	0

Portas Lógicas em VHDL



Inputs		Output
A	B	F
0	0	1
1	0	0
0	1	0
1	1	0

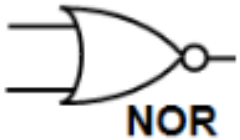
Portas Lógicas em VHDL



Inputs		Output
A	B	F
0	0	1
1	0	0
0	1	0
1	1	0

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  entity gate is
26      Port ( a : in STD_LOGIC;
27             b : in STD_LOGIC;
28             f : out STD_LOGIC);
29  end gate;
30
31  architecture Behavioral of gate is
32  begin
33
34      f <= a nor b;
35
36  end Behavioral;
```

Portas Lógicas em VHDL



Inputs		Output
A	B	F
0	0	1
1	0	0
0	1	0
1	1	0

Portas Lógicas em VHDL

Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE OR

Portas Lógicas em VHDL

Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

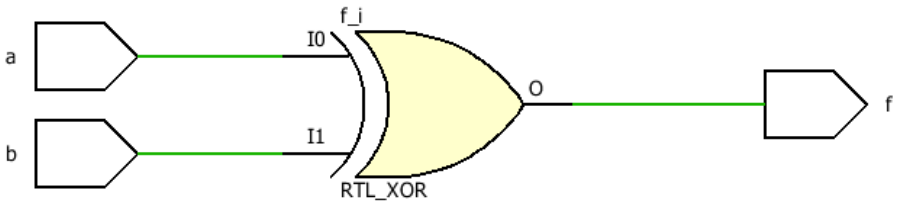
EXCLUSIVE OR

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  entity gate is
26      Port ( a : in STD_LOGIC;
27             b : in STD_LOGIC;
28             f : out STD_LOGIC);
29  end gate;
30
31  architecture Behavioral of gate is
32  begin
33
34      f <= a xor b;
35
36  end Behavioral;
```

Portas Lógicas em VHDL

Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE OR



Portas Lógicas em VHDL

EXCLUSIVE NOR

Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Portas Lógicas em VHDL

EXCLUSIVE NOR

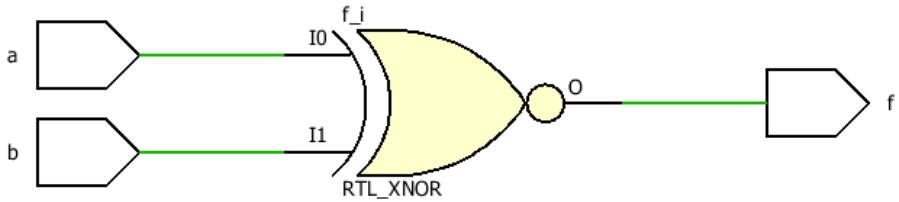
Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  entity gate is
26      Port ( a : in STD_LOGIC;
27             b : in STD_LOGIC;
28             f : out STD_LOGIC);
29  end gate;
30
31  architecture Behavioral of gate is
32  begin
33
34      f <= a xnor b;
35
36  end Behavioral;
```

Portas Lógicas em VHDL

EXCLUSIVE NOR

Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1



Circuitos combinacionais

Circuitos combinacionais

Definição de circuitos combinacionais

Circuitos cujo valor de(s) saída(s) depende exclusivamente da combinação dos valores presentes nas entradas.

Circuitos combinacionais

- Circuitos puramente combinacionais podem ser facilmente descritos em VHDL usando operações lógicas.
- Entretanto, nem sempre é simples obter a(s) expressão(ões) lógica(s) que representa(m) a(s) saída(s) do circuito. Para isso, existem técnicas de obtenção e simplificação de circuitos combinacionais, como:
  - Levantamento da tabela-verdade do circuito;
  - Álgebra de *Boole*;
  - Mapa de *Karnaugh*.

Exemplo 1: Circuitos combinacionais

- Vamos considerar um circuito de votação por maioria: temos 3 votos de entrada ('0' = não, '1' = sim) e a saída (decisão) recebe o resultado votado pela maioria.

Voto 1	Voto 2	Voto 3	Decisão
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

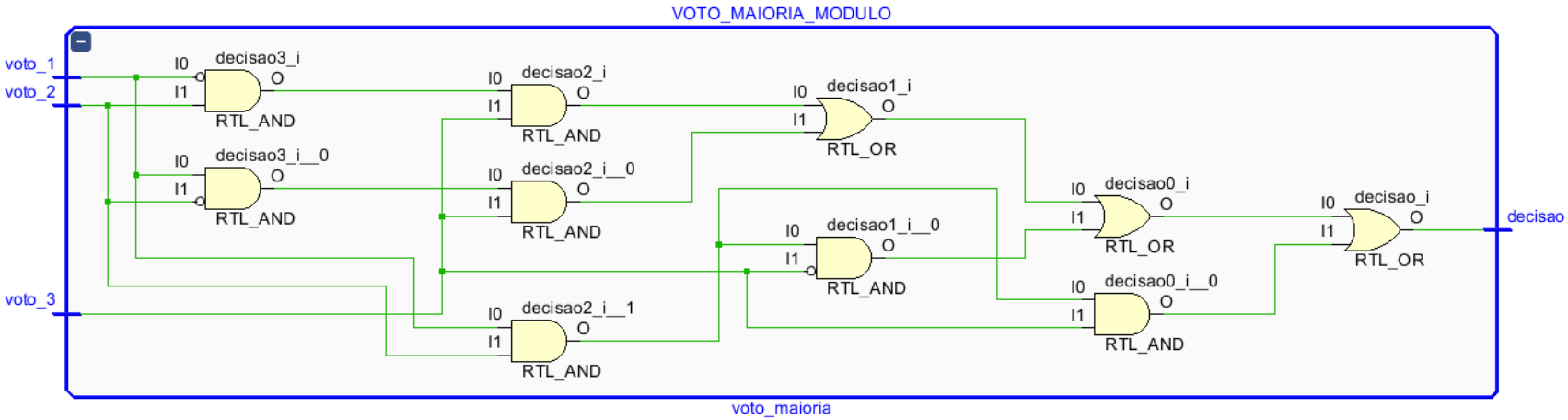
Exemplo 1: Circuitos combinacionais

- Temos a princípio a seguinte expressão booleana de saída:
  $\overline{v1}v2v3 + v1\overline{v2}v3 + v1v2\overline{v3} + v1v2v3$

v1	v2	v3	Decisão
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Exemplo 1: Circuitos combinacionais

- Podemos verificar o esquemático gerado pelo Vivado na opção **RTL Analysis -> Schematic**.



Exemplo 1: Circuitos combinacionais

- Podemos verificar a tabela verdade gerada pelo Vivado na opção **Synthesized Design -> Schematic -> Selecionar a LUT no esquemático -> Cell Properties -> Truth Table**

Cell Properties

decisao_INST_0

I2	I1	I0	O=I0 & I1 + I0 & I2 + I1 & I2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- O próprio Vivado otimizou a expressão lógica da saída!

Exemplo 1: Circuitos Combinacionais

- E para testar o circuito? Usamos um arquivo .vhd de *testbench* (cuja entidade é vazia) que é composto por 4 etapas:
  1. **Declarar o módulo a ser testado como componente;**
  2. Declarar os sinais de teste;
  3. Conectar o módulo a ser testado aos sinais de teste;
  4. Criar valores de testes para os sinais de teste.

```
-- Seção Library: declarando as bibliotecas necessárias para o nosso módulo.
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Seção Entity: declarando as entradas e saídas do nosso módulo, bem como o tipo
-- e a quantidade de bits de cada um. Para testbenches, a entidade geralmente é vazia.
--
entity tb_voto_maioria is
end tb_voto_maioria;

-- Seção Architecture: descrição do teste comportamental.
--
architecture Behavioral of tb_voto_maioria is

    -- Instanciação do componente voto_maioria para teste
    --
    component voto_maioria is
        Port (
            voto_1 : in std_logic;
            voto_2 : in std_logic;
            voto_3 : in std_logic;
            decisao : out std_logic);
    end component voto_maioria;
```

Exemplo 1: Circuitos Combinacionais

- E para testar o circuito? Usamos um arquivo .vhd de *testbench* (cuja entidade é vazia) que é composto por 4 etapas:
  1. Declarar o módulo a ser testado como componente;
  2. **Declarar os sinais de teste;**
  3. Conectar o módulo a ser testado aos sinais de teste;
  4. Criar valores de testes para os sinais de teste.

```
-- Sinal de teste s_voto_1, inicializando com '0'.
signal s_voto_1 : std_logic := '0';

-- Sinal de teste s_voto_2, inicializando com '0'.
signal s_voto_2 : std_logic := '0';

-- Sinal de teste s_voto_3, inicializando com '0'.
signal s_voto_3 : std_logic := '0';

-- Sinal de teste s_decisao, inicializando com '0'.
signal s_decisao : std_logic := '0';
```

Exemplo 1: Circuitos Combinacionais

- E para testar o circuito? Usamos um arquivo .vhd de *testbench* (cuja entidade é vazia) que é composto por 4 etapas:
  1. Declarar o módulo a ser testado como componente;
  2. Declarar os sinais de teste;
  3. **Conectar o módulo a ser testado aos sinais de teste;**
  4. Criar valores de testes para os sinais de teste.

```
-- Realizando a conexão dos sinais de teste às suas entradas e saídas
-- respectivas
uut: voto_maioria port map
(
    voto_1 => s_voto_1,
    voto_2 => s_voto_2,
    voto_3 => s_voto_3,
    decisao => s_decisao
);
```

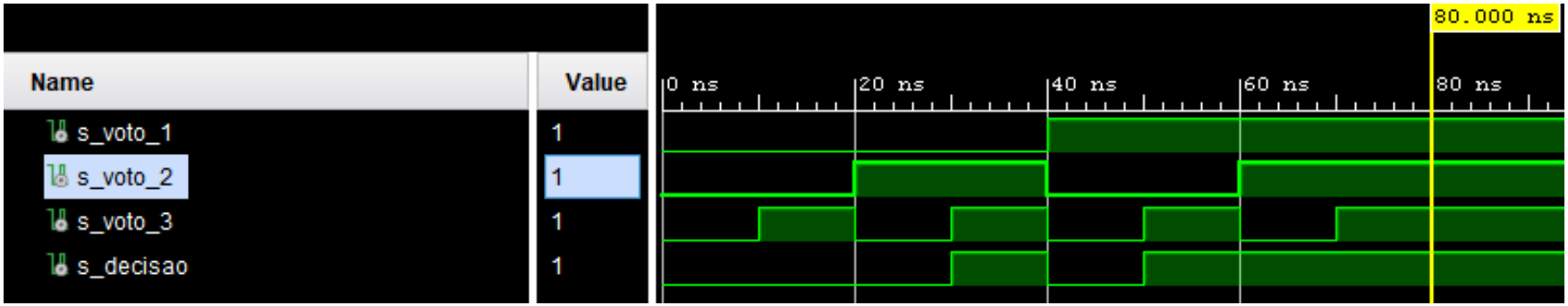
Exemplo 1: Circuitos Combinacionais

- E para testar o circuito? Usamos um arquivo .vhd de *testbench* (cuja entidade é vazia) que é composto por 4 etapas:
  1. Declarar o módulo a ser testado como componente;
  2. Declarar os sinais de teste;
  3. Conectar o módulo a ser testado aos sinais de teste;
  4. **Criar valores de testes para os sinais de teste.**

```
-- A seção após a conexão dos sinais de teste corresponde aos estímulos, ou seja,
-- fazer com que os sinais de teste assumam determinados valores durante a
-- simulação comportamental. Associamos valores de teste às entradas e verificamos
-- nas saídas o comportamento do circuito.
--
-- Os estímulos abaixo buscam emular a tabela-verdade de um circuito combinacional
-- de 3 entradas:
--
-- Estímulo do sinal s_voto_1: atribuir um novo valor a cada 10 ns
s_voto_1 <= '0', '0' after 10 ns, '0' after 20 ns, '0' after 30 ns,
            '1' after 40 ns, '1' after 50 ns, '1' after 60 ns, '1' after 70 ns;

-- Estímulo do sinal s_voto_2: atribuit um novo valor a cada 10 ns
s_voto_2 <= '0', '0' after 10 ns, '1' after 20 ns, '1' after 30 ns,
            '0' after 40 ns, '0' after 50 ns, '1' after 60 ns, '1' after 70 ns;
-- Estímulo do sinal s_voto_3: atribuir um novo valor a cada 10 ns
s_voto_3 <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 30 ns,
            '0' after 40 ns, '1' after 50 ns, '0' after 60 ns, '1' after 70 ns;
```

Exemplo 1: Circuitos Combinacionais

- Para realizar a simulação compportamental: **Run Simulation**



Sumário

Processos

```
optional_label: process (optional sensitivity list)
        declarations
begin
        sequential statements
end process optional_label;
```

- Implementam instruções sequenciais;

Processos

```
optional_label: process (optional sensitivity list)
        declarations
begin
        sequential statements
end process optional_label;
```

- Implementam instruções sequenciais;
- Declarados dentro da arquitetura de um sistema;

Processos

```
optional_label: process (optional sensitivity list)
        declarations
begin
        sequential statements
end process optional_label;
```

- Implementam instruções sequenciais;
- Declarados dentro da arquitetura de um sistema;
- Possuem uma lista de sensibilidade, que possui todos os sinais que ao mudar, iniciam as instruções dentro de um processo.

Processos

```
process (ALARM_TIME, CURRENT_TIME)
begin
  if (ALARM_TIME = CURRENT_TIME) then
    SOUND_ALARM <= '1';
  else
    SOUND_ALARM <= '0';
  end if;
end process;
```

- **MUITO IMPORTANTE:**
  1. Processos ocorrem em paralelo em relação a outros processos.

Processos

```
process (ALARM_TIME, CURRENT_TIME)
begin
  if (ALARM_TIME = CURRENT_TIME) then
    SOUND_ALARM <= '1';
  else
    SOUND_ALARM <= '0';
  end if;
end process;
```

- **MUITO IMPORTANTE:**
  1. Processos ocorrem em paralelo em relação a outros processos.
  2. Processos são atualizados somente ao final de todas as instruções, **com exceção de variáveis**.

Processos

```
process (ALARM_TIME, CURRENT_TIME)
begin
  if (ALARM_TIME = CURRENT_TIME) then
    SOUND_ALARM <= '1';
  else
    SOUND_ALARM <= '0';
  end if;
end process;
```

- **MUITO IMPORTANTE:**
  1. Processos ocorrem em paralelo em relação a outros processos.
  2. Processos são atualizados somente ao final de todas as instruções, **com exceção de variáveis**.
  3. Tomar muito cuidado ao acessar saídas e sinais em dois processos ao mesmo tempo. Processos ocorrem em paralelo, mas as instruções não.

Processos

Mandamentos de processos

1. Não escreverás nos mesmos sinais, variáveis e saídas em processos diferentes.
2. Não se esquecerás da lista de sensibilidade.
3. Não esquecerás do *begin*.
4. Não nomearás dois processos com o mesmo nome.
5. Não utilizarás variáveis em vão.

Estrutura condicional - *when/case*

```
case expression is
   when choice =>
     sequential statements
   when choice =>
      sequential statements
end case;
```

- Dado valor de uma variável, realiza a instrução correspondente ao valor lido;
- Declarados dentro de um processo;
- Todas as opções devem ser declaradas, a não ser quando utilizada a opção *when others*.
- As opções não podem conflitar entre si.

Estrutura condicional - *when/case*

```
case SEL is
   when "01" =>    Z <= A;
   when "10" =>    Z <= B;
   when others => Z <= 'X';
end case;
```

Estrutura condicional - *if*

```
if condition_1 then
        sequential statements
elsif condition2 then
        sequential statements
else
        sequential statements
end if;
```

- Dentre um determinado números de condições, executa a que for verdadeira;
- Declarados dentro de um processo;
- O uso de *elsif* é opcional.
- **Se existirem condições que não forem especificadas, um *latch* pode ser criado e problemas de temporização do circuito podem acontecer.**

Estrutura condicional - *if*

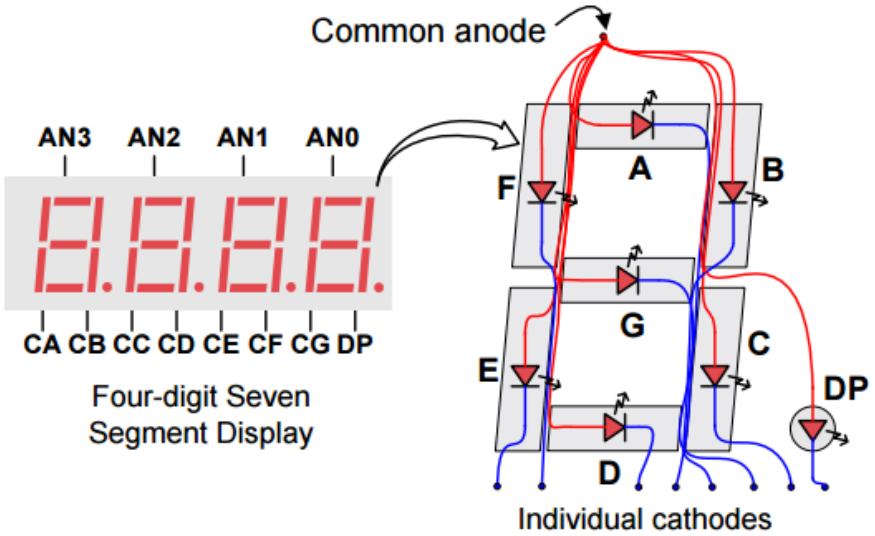
```
if (X = 5) and (Y = 9) then
        Z <= A;
elsif (X >= 5) then
        Z <= B;
else
        Z < C;
end if;
```

Estrutura condicional - *if*

```
process (EN, D)
begin
        if (EN = '1') then Q <= D;
        end if;
end process;
```

Exercício 2 - Processos e estruturas condicionais

- Projetar em VHDL um decodificador para *display* de 7 segmentos;
- O sistema recebe uma entrada pelas chaves e exibe o valor correspondente em um *display*;
- Cada segmento do display corresponde é ativo em nível lógico baixo;
- Cada display é acionado por um ânodo ativo em nível lógico baixo.



Exercício 2 - Processos e estrututas condicionais

HEX	dp	A	B	C	D	E	F	G
0	1	0	0	0	0	0	0	1
1	1	1	0	0	1	1	1	1
2	1	0	0	1	0	0	1	0
3	1	0	0	0	0	1	1	0
4	1	1	0	0	1	1	0	0
5	1	0	1	0	0	1	0	0
6	1	0	1	0	0	0	0	0
7	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0
9	1	0	0	0	0	1	0	0
A	1	0	0	0	1	0	0	0
B	1	1	1	0	0	0	0	0

Exercício 2 - Processos e estrututas condicionais

HEX	dp	A	B	C	D	E	F	G
C	1	0	1	1	0	0	0	1
D	1	1	0	0	0	0	1	0
E	1	0	1	1	0	0	0	0
F	1	1	1	1	0	0	0	0

Exercício 2 - Processos e estruturas condicionais

- E para testar no laboratório remoto?

Exercício 2 - Processos e estruturas condicionais

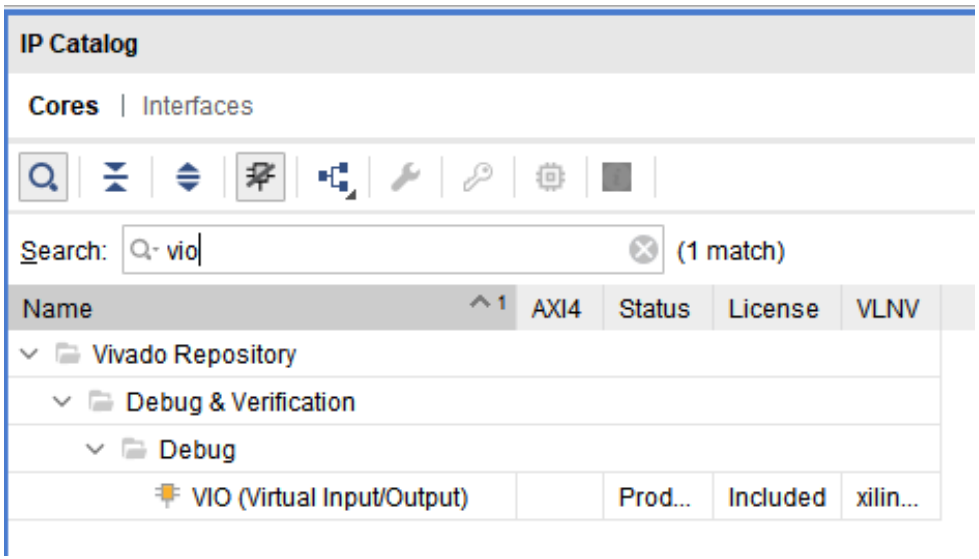
- E para testar no laboratório remoto?
- Precisamos combinar o nosso circuito com um IP da Xilinx: o VIO IP Core

Exercício 2 - Processos e estruturas condicionais

- E para testar no laboratório remoto?
- Precisamos combinar o nosso circuito com um IP da Xilinx: o VIO IP Core
- O VIO irá gerar entradas controladas virtualmente por nós e enviá-las para o FPGA para que possamos testar o circuito no FPGA.

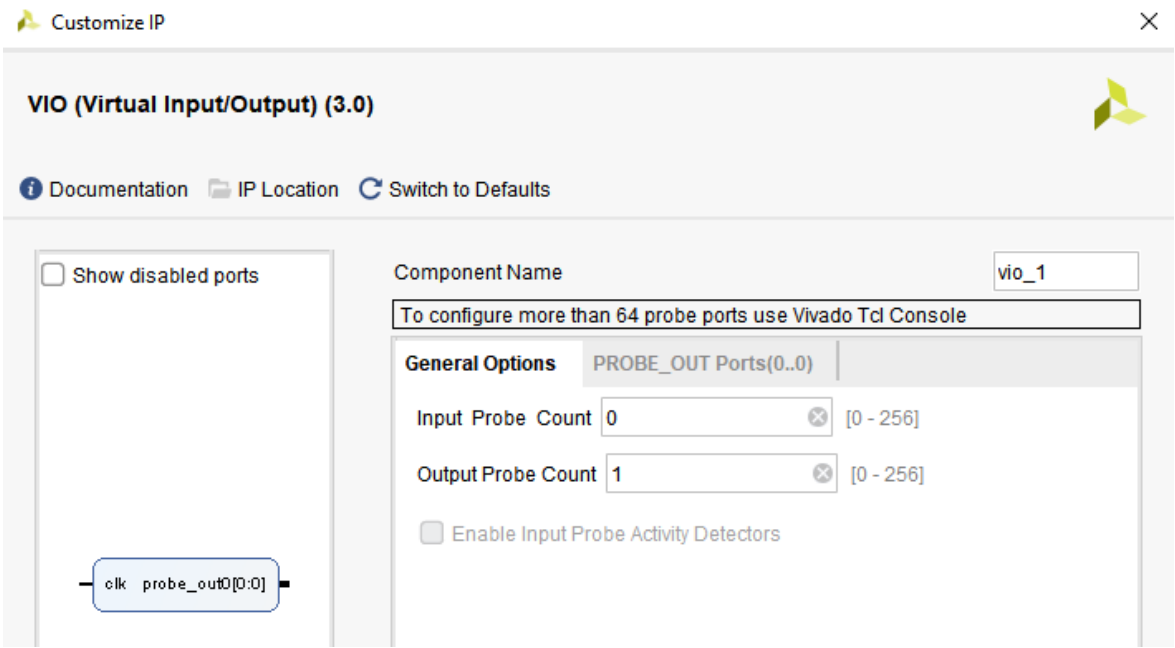
Exercício 2 - Processos e estruturas condicionais

- O VIO pode ser gerado em **IP Catalog -> VIO**.



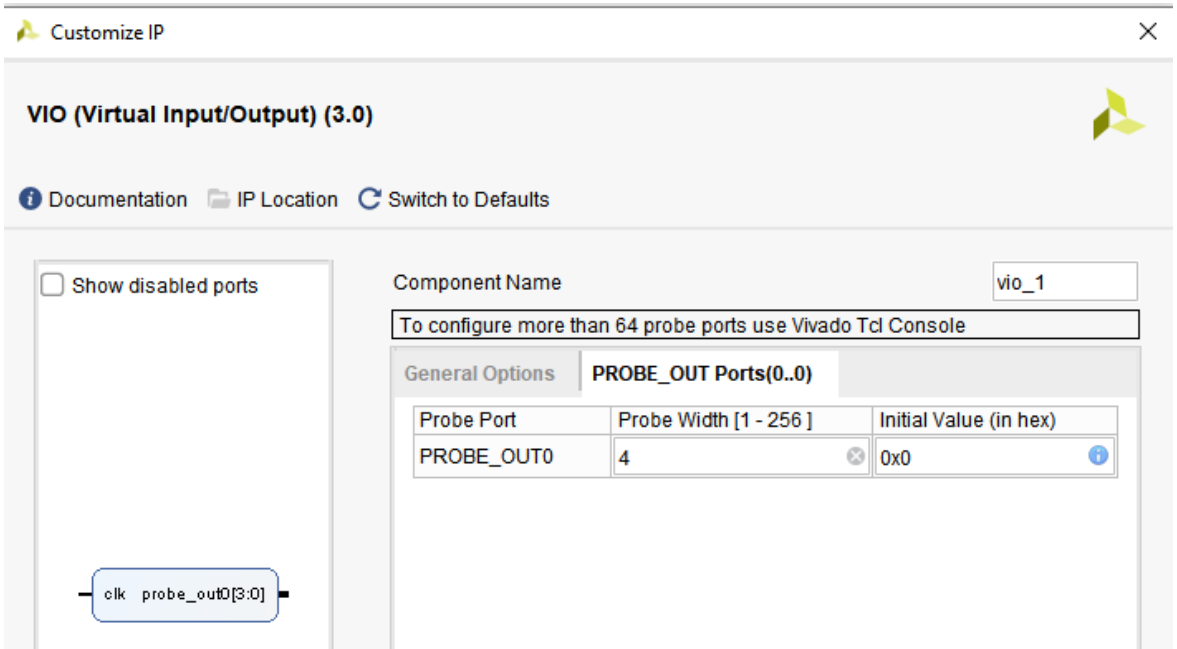
Exercício 2 - Processos e estruturas condicionais

- Temos que definir o número de entradas e saídas do VIO: não teremos entradas e o número de saídas do VIO será igual ao número de entradas do circuito a ser testado no laboratório remoto.



Exercício 2 - Processos e estruturas condicionais

- Temos que definir o número de entradas e saídas do VIO: não teremos entradas e o número de saídas do VIO será igual ao número de entradas do circuito a ser testado no laboratório remoto.



Exercício 2 - Processos e estruturas condicionais

- Criamos um arquivo que conecta nosso circuito ao VIO (*top module*);
- OBS: O VIO precisa de um sinal de clk, que dita a frequência de funcionamento do mesmo.

```
entity top_module_display_7seg is
    Port (
            -- Entrada
            --
            -- Entrada de clock de 1-bit.
            -- Descrição: Entrada de clock que implementa a frequência de operação
            -- do VIO Core. O circuito 'display_7seg' é puramente combinacional,
            -- não necessita do clock, mas o VIO IP core sim.
            --
            clk : in STD_LOGIC;

            -- Saídas
            --
            -- Saída an de 4-bits.
            -- Descrição: Cada bit dessa saída corresponde ao anôdo de um display
            -- da Basys 3: o bit menos significativo controla o anôdo do display
            -- mais à direita no kit e o bit mais significativo controla o anôdo
            -- mais à esquerda dno kit. Os anôdos habilitam seu respectivo display
            -- em '0', e desabilitam o mesmo em '1'.
            --
            an : out STD_LOGIC_VECTOR (3 downto 0);

            -- Pino de saída referente aos 7 segmentos dos displays da Basys 3: 7-bits
            -- Descrição: Cada bit dessa saída corresponde a um dos 7 segmentos dos
            -- displays da Basys 3 na ordem 'gfedcba'. Os segmentos são ligados em
            -- '0' e desligados em '1'.
            --
            seg : out STD_LOGIC_VECTOR (6 downto 0));
end top_module_display_7seg;
```

Exercício 2 - Processos e estruturas condicionais

- Criamos um arquivo que conecta nosso circuito ao VIO (*top module*);
- OBS: O VIO precisa de um sinal de clk, que dita a frequência de funcionamento do mesmo.

```
architecture Behavioral of top_module_display_7seg is

    -- Instanciação do componente display_7seg
    --
    component display_7seg is
        Port(
                data_in : in STD_LOGIC_VECTOR(3 DOWNTO 0);
                an : out STD_LOGIC_VECTOR(3 DOWNTO 0);
                seg : out STD_LOGIC_VECTOR(6 DOWNTO 0));
    end component;

    -- Instanciação do componente vio_0
    component vio_0 is
        Port(
                clk : in STD_LOGIC;
                probe_out0 : out STD_LOGIC_VECTOR(3 DOWNTO 0));
    end component;

    -- Sinal de conexão entre a saída 'probe_out0' do VIO core e
    -- a entrada 'data_in' do módulo display_7seg.
    --
    signal s_data_in : std_logic_vector(3 downto 0) := (others => '0');
```

Exercício 2 - Processos e estruturas condicionais

- Criamos um arquivo que conecta nosso circuito ao VIO (*top module*);
- OBS: O VIO precisa de um sinal de clk, que dita a frequência de funcionamento do mesmo.

```
    -- Conexão dos pinos do módulo display_7seg a seus respectivos sinais, entradas e saídas
    --
    MODULO_DISPLAY_7SEG: display_7seg port map
    (
        data_in => s_data_in,
        an => an,
        seg => seg
    );

    -- Conexão dos pinos do módulo vio_0 a seus respectivos sinais, entradas e saídas.
    --
    VIO_CORE: vio_0 port map
    (
        clk => clk,
        probe_out0 => s_data_in
    );

end Behavioral;
```

Exercício 2 - Processos e estruturas condicionais

- Por fim, precisamos do arquivo de *constraints*, que realiza o mapeamento das entradas e saídas do circuito para os recursos disponíveis no kit da Basys 3.
- Há um arquivo .xdc da Basys3 disponível na internet que facilita esse processo: descomentamos os recursos que serão utilizados e colocamos o nome das nossas entradas e saídas.

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
    set_property IOSTANDARD LVCMOS33 [get_ports clk]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
```

Exercício 2 - Processos e estruturas condicionais

- Por fim, precisamos do arquivo de *constraints*, que realiza o mapeamento das entradas e saídas do circuito para os recursos disponíveis no kit da Basys 3.
- Há um arquivo .xdc da Basys3 disponível na internet que facilita esse processo: descomentamos os recursos que serão utilizados e colocamos o nome das nossas entradas e saídas.

```
#7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
```

Exercício 2 - Processos e estruturas condicionais

- Por fim, precisamos do arquivo de *constraints*, que realiza o mapeamento das entradas e saídas do circuito para os recursos disponíveis no kit da Basys 3.
- Há um arquivo .xdc da Basys3 disponível na internet que facilita esse processo: descomentamos os recursos que serão utilizados e colocamos o nome das nossas entradas e saídas.

```
set_property PACKAGE_PIN U2 [get_ports {an[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
```

Fim da 1 ªaula.

Muito obrigado pela atenção de todos.

Códigos e apresentação disponíveis aqui: https://github.com/alceu-castanheira/Curso-VHDL-Eletronjun-2020