DESCRIÇÃO DE CIRCUITOS DIGITAIS EM VHDL (SIMPLIFICADA)

Slides by: Pedro Tomás

TÉCNICO LISBOA

SISTEMAS DIGITAIS

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Outline

- Linguagens de descrição de Hardware [LINK]
- Introdução a VHDL [LINK]
  - Descrição de estruturas básicas em VHDL [LINK]
  - Exemplos:
    - Cadeado digital [LINK]
- Simulação de circuitos em VHDL [LINK]

Etapas

1. **Descrição do sistema a projectar**
   - A descrição do sistema é tipicamente feita sob uma forma verbal, não totalmente especificada (tal como aparece nos enunciados de laboratório)
2. **Especificação do sistema**
   - Divisão do problema em partes (funções lógicas) e especificação de cada uma das partes (funções), geralmente sob a forma de tabelas de verdade
3. **Derivação das expressões lógicas**
   - Obtenção e minimização das funções lógicas, através da expressão booleana ou mapas de Karnaugh
4. **Desenho do circuito digital**
   - Desenho do circuito digital usando os elementos básicos de lógica (portas NOT, AND, OR, NAND, NOR, XOR, MUX, DECODER, ...)
5. **Implementação física**
   - Actualmente feita quase exclusivamente em FPGAs (lógica programável) ou em circuitos integrados

Implementação física

- **Implementação física**
  - Actualmente é raro a implementação de circuitos digitais com componentes discretos (CIs).
    - Os circuitos são tipicamente muito complexos e não permitem tais implementações
  - Recorre-se tipicamente a linguagens de descrição de *hardware*, tais como VHDL ou Verilog
    - Para descrever correctamente o circuito digital em VHDL ou Verilog é necessário conceber primeiro o diagrama lógico

- **Linguagens de descrição de hardware**
  (HDL – Hardware Description Languages)
  - **VHDL**

    Very High Speed Integrated Circuits (VHSIC) Hardware Description Language
  - **Verilog**

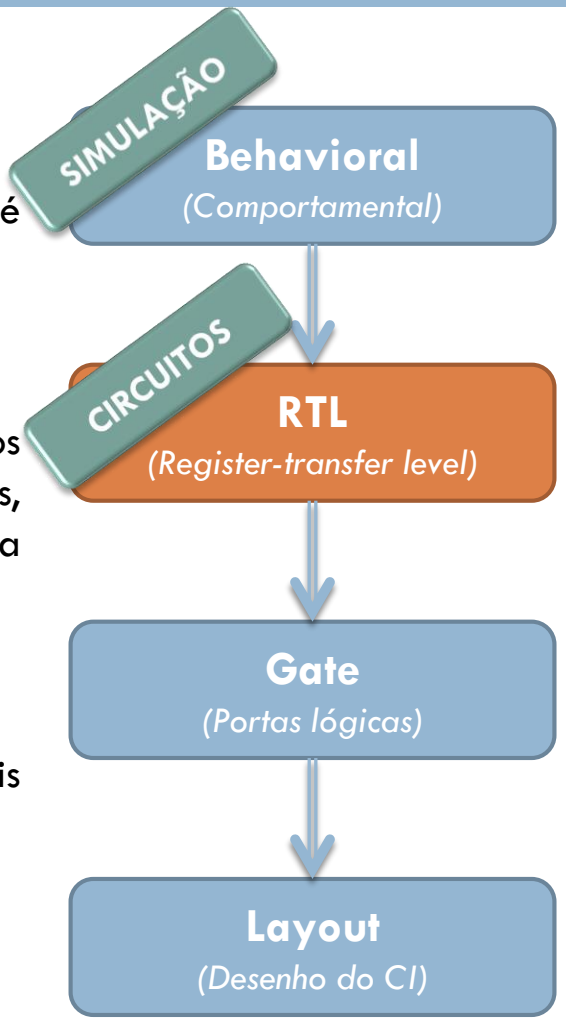
VHDL, para que serve?

- VHDL (e *Verilog*) serve para:
  - Descrever circuitos digitais
  - Simular circuitos digitais
    - Verificar a funcionalidade, testar e corrigir os erros

- VHDL (e *Verilog*) não é uma linguagem de programação
  - Os circuitos digitais não se programam... descrevem-se!
  - Escrever código VHDL não é mais do que desenhar o esquema lógico do circuito digital!
    - Na Unidade Curricular (UC) de Sistemas Digitais (SD) será sempre OBRIGATÓRIA a apresentação do esquema lógico
    - Nem todas as funcionalidade de VHDL serão permitidas!
    - Quaisquer diferenças entre o esquema apresentado e o código VHDL apresentado levarão a penalizações na nota final!
    - O código VHDL deverá ser sempre apresentado em anexo, devidamente comentado

Níveis de abstração

- Comportamental (*Behavioral*)
  - Descrição funcional do circuito digital
  - Geralmente usada para simular circuitos, mas nem sempre é sintetizável para portas lógicas
- RTL (*Register-Transfer Level*)
  - Descrição do circuito através da divisão entre os elementos combinatórios (AND, OR, NOT, multiplexers, descodificadores, somadores/subtractores, ...) e os elementos de memória (latches, flip-flops, registos, ...)
- Gate (portas lógicas AND, OR, ...)
  - Descrição do circuito através dos componentes principais existentes numa dada biblioteca lógica
- Layout
  - Desenho do circuito integrado



Descrição de circuitos em VHDL

Very High Speed Integrated Circuit (VHSIC)
Hardware
Description
Language

Descrição de circuitos em VHDL

- Um ficheiro VHDL (extensão .vhd) descreve o funcionamento de um circuito digital e pode ser decomposto em duas partes:
  - Entidade
    - Definição do componente (circuito digital), nomeadamente nome e sinais (fios) de entrada e de saída
  - Arquitectura:
    - Descrição da forma como o componente está implementado

TÉCNICO LISBOA

Estrutura típica de um ficheiro VHDL


```
-- COMENTÁRIOS

-- Declaração de bibliotecas com pré-definições
library IEEE;
use IEEE.std_logic_1164.all;

-- Definição do nome da entidade e dos sinais (fios) de entrada/saída
entity <NOME_DO_COMPONENTE> is
   port (
     ...
   );
end <NOME_DO_COMPONENTE>;

-- Descrição da arquitectura (implementação) do componente
architecture <TIPO_DE_ARQUITECTURA> of <NOME_DO_COMPONENTE> is
   -- declaração dos sinais (fios) internos ao componente
Begin
   -- descrição do circuito digital que implementa o componente
   ...
end <TIPO_DE_ARQUITECTURA>;
```

Tipos de sinais

- Um sinal em VHDL corresponde a um fio num circuito físico
- Num circuito digital o tipo de fio deverá ser:
  - `bit`, o qual pode ter os valores lógicos 0 e 1
- No entanto é comum usar-se outro sinal, o qual é particularmente útil durante o passo de simulação do circuito:
  - `std_logic`, o qual pode tomar os valores lógicos:
    - '0' (zero) e '1' (um)
    - 'Z' alta impedância
    - '-' *don't care*
    - 'U' *undefined* (o valor do fio não foi definido)
    - 'X' *unknown* (não é possível determinar o valor do fio)

Nota: na UC de SD apenas é permitido a atribuição dos valores 0 e 1 a um sinal! Os valores 'U' e 'X' serão atribuidos automaticamente pela ferramenta (Xilinx ISE) quando um sinal (fio) não tiver valor atribuído ou não for possível a sua determinação (ex: quando são atribuidos simultaneamente os valores '0' e '1').

Tipos de sinais

- Por vezes são necessários vários fios para representar um único valor.

- Exemplo:

  Representação do número de um aluno do IST

  - Considerando que o maior número de aluno é o 80 000
  - São necessários pelo menos $\log_2(80\,000)$ bits = 16,3 bits

  - Portanto o sinal `num_aluno` precisa de pelo menos 17 bits

Tipos de sinais

- Por vezes são necessários vários fios para representar um único valor.

- Exemplo:

  Representação do número dos alunos do IST

  - Para evitar a declaração (especificação) de 17 sinais (fios) individualmente, utiliza-se o sinal de barramento (*bus*):

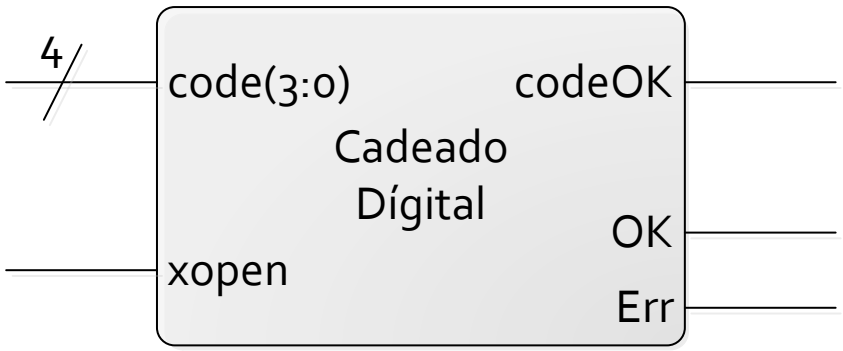
```
signal num_aluno : std_logic_vector(16 downto 0);
```

  Permitindo assim definir os fios

  `num_aluno(16), num_aluno(15), …, num_aluno(1), num_aluno(0)`

  Cada um do tipo `std_logic`

Exemplo: cadeado digital



O cadeado abre com o código $0111_2=7_{16}$

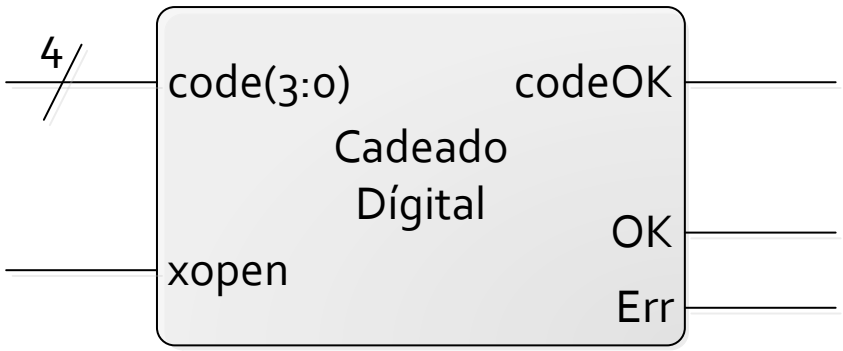
```
-- Definição do nome da entidade e
-- dos sinais (fios) de entrada/saída
entity <NOME_DO_COMPONENTE> is
    port (
          <NOME_DO_SINAL> : <IN/OUT> <TIPO_DE_SINAL>;
           . . .
          <NOME_DO_SINAL> : <IN/OUT> <TIPO_DE_SINAL>
    );
end <NOME_DO_COMPONENTE>;
```

Lista de sinais separados por ";"

A ultima linha não deve ser terminada por ";"

Nota: Já que a palavra `open` está reservada em VHDL, não é possível representar sinais com este nome. Assim, o sinal de comando para abertura do cadeado foi renomeado para "`xopen`"

Exemplo: cadeado digital



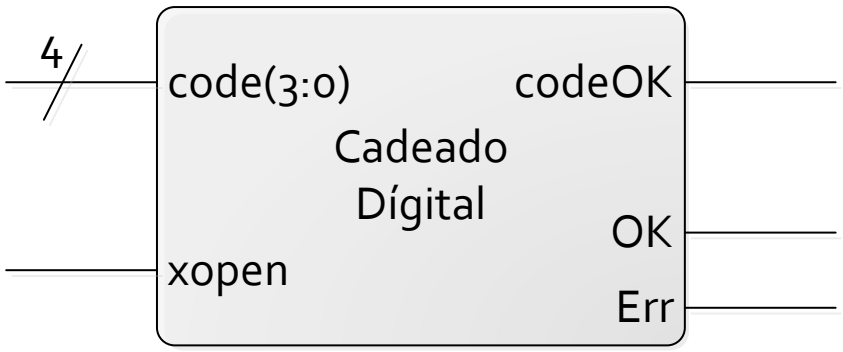
O cadeado abre com o código $0111_2=7_{16}$.

```
entity cadeado_digital is
   port (
         code    : in  std_logic_vector(3 downto 0);
         xopen   : in  std_logic;
         codeOK  : out std_logic;
         OK      : out std_logic;
         Err     : out std_logic
         );
end cadeado_digital;
```

Lista de sinais separados por ";"

A ultima linha não deve ser terminada por ";"

Exemplo: cadeado digital

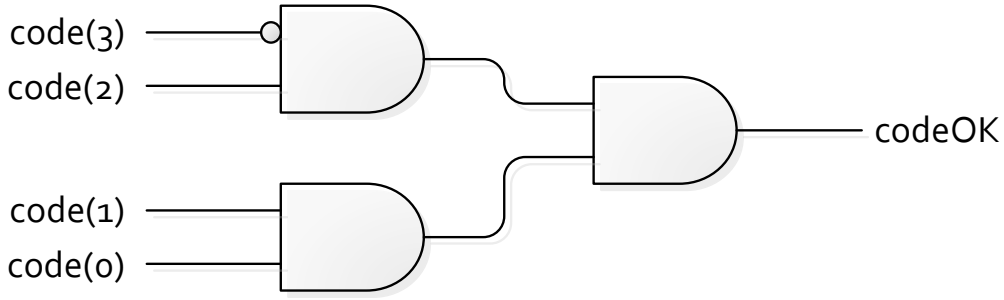


O cadeado abre com o código $0111_2=7_{16}$.

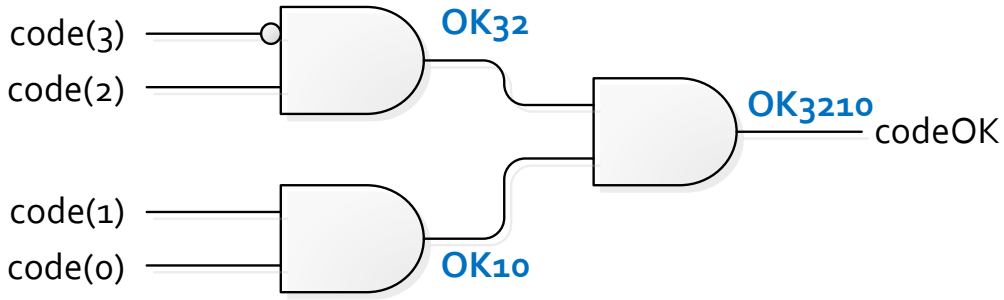
- Descrição da implementação do circuito "`cadeado_digital`"

Exemplo: cadeado digital

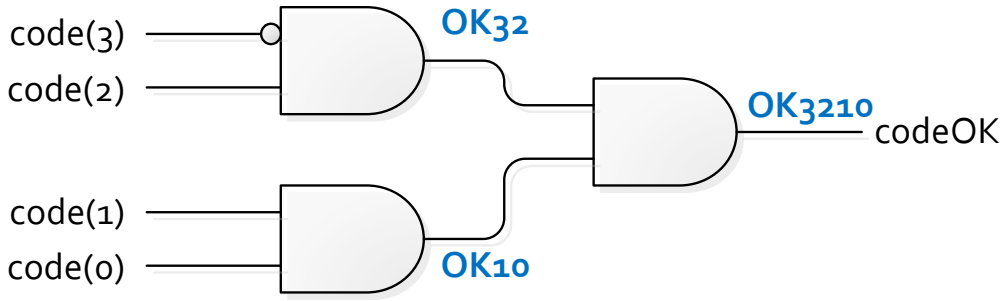
- Após projecto e desenho do logigrama correspondente, obtem-se:



- Em VHDL poderia ser dado um nome a cada um dos fios intermédios, para ter visibilidade relativamente a esses sinais internos:



Exemplo: cadeado digital



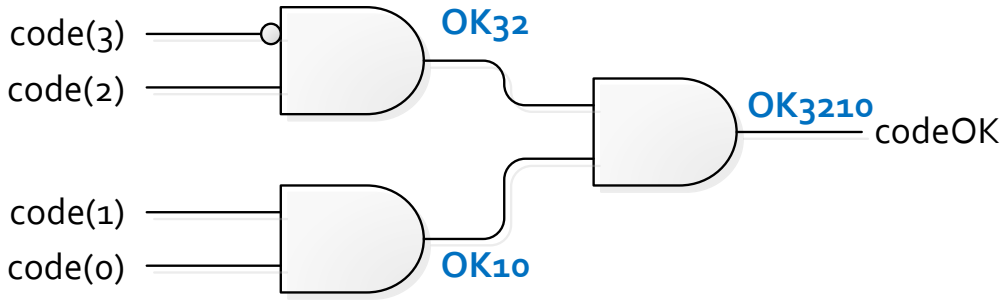
- Assim, declaram-se os sinais OK32, OK10 e OK3210

- A declaração de sinais é realizada sob a forma:

```
signal <NOME_DO_SINAL_1>, <NOME_DO_SINAL_2> : TIPO_DE_SINAL>;
signal <NOME_DO_SINAL_3> : <TIPO_DE_SINAL>;
signal <NOME_DO_SINAL_4> : TIPO_DE_SINAL>;
```

- Naturalmente é possivel ter sinais (fios) de diferentes tipos
  - No entanto a declaração anterior obriga a que os sinais 1 e 2 tenham o mesmo tipo

Exemplo: cadeado digital



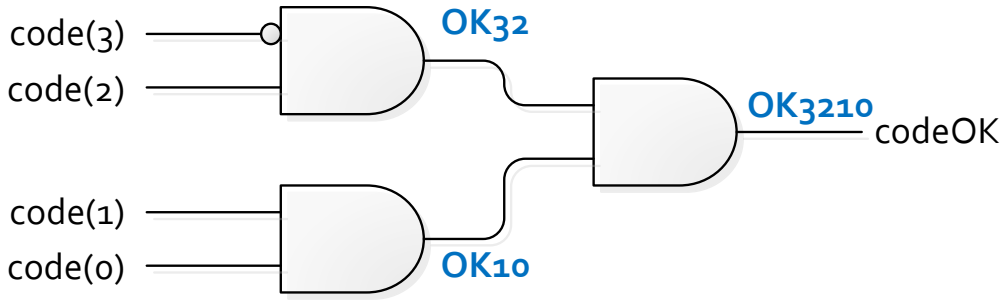
- Assim é necessário declarar os sinais OK32, OK10 e OK3210

```
entity cadeado_digital is
   port (
     ...
   );
end cadeado_digital;

architecture behavior of cadeado_digital is
-- declaração dos sinais (fios) internos ao componente
signal OK32, OK10, OK3210 : std_logic;

Begin
...
end behavior;
```

O nome dado à arquitectura não é relevante...

O Vivado normalmente usa a designação behavior

Exemplo: cadeado digital



- Resta-nos agora descrever o circuito projectado

- A atribuição de valores para os sinais é feita da seguinte forma:

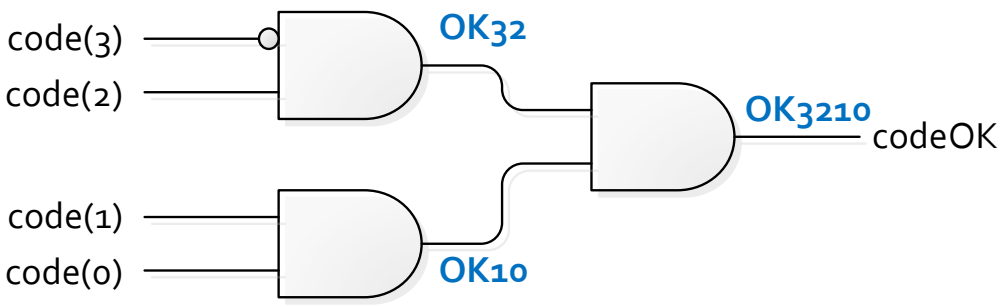
```
NOME_SINAL_DESTINO <= OPERAÇÃO_LÓGICA_SOBRE_OPERANDOS;
```

- Existem várias operações típicas:

NOT **AND** **OR** **NAND** **NOR** **XOR** ...

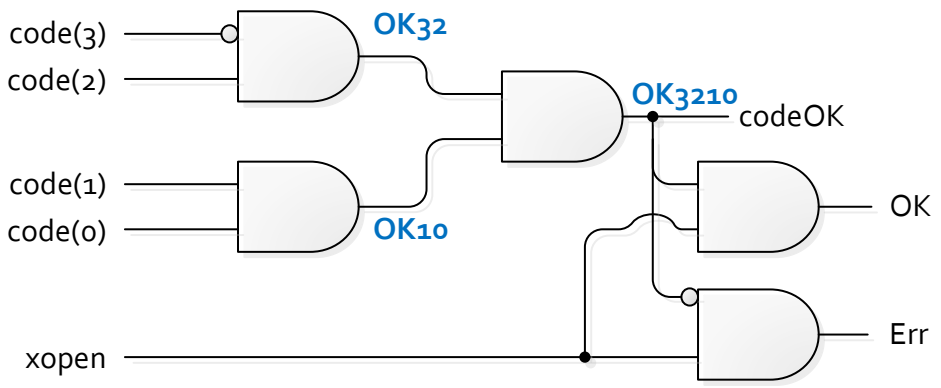
Exemplo: cadeado digital


```
...
architecture behavior of cadeado_digital is
-- declaração dos sinais (fios) internos ao componente
signal OK32, OK10, OK3210 : std_logic;
begin

-- Cálculo do resultado
OK32   <= (not code(3)) and code(2);
OK10   <= code(1) and code(0);
OK3210 <= OK32 and OK10;

-- Atribuição do valor de saída
codeOK <= OK3210;

end behavior;
```



Exemplo: cadeado digital

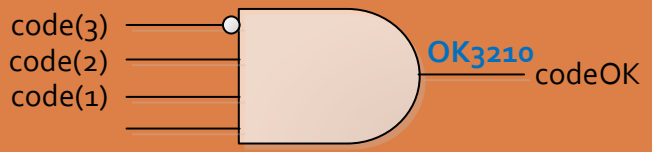

```
...
architecture behavior of cadeado_digital is
-- declaração dos sinais (fios) internos ao componente
signal OK32, OK10, OK3210 : std_logic;
begin

-- Cálculo do resultado
OK32   <= (not code(3)) and code(2);
OK10   <= code(1) and code(0);
OK3210 <= OK32 and OK10;

-- Atribuição do valor de saída
codeOK <= OK3210;
OK  <= OK3210 and xopen;
Err <= (not OK3210) and xopen;

end behavior;
```



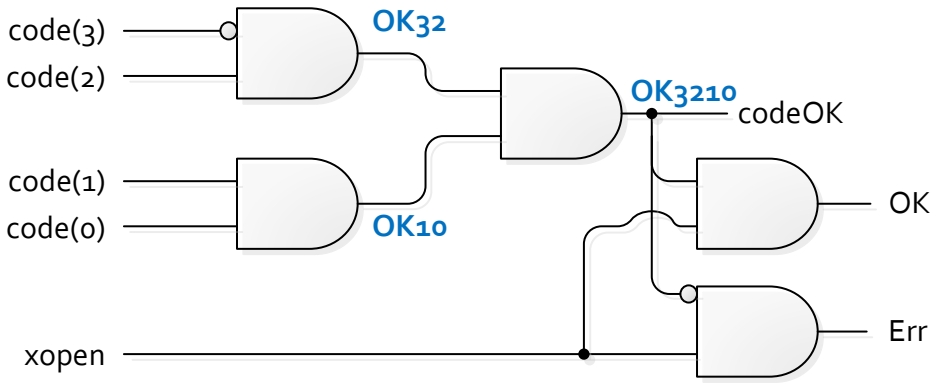
Exemplo: cadeado digital


```
...
architecture behavior of cadeado_digita
-- declaração dos sinais (fios) interno
signal OK32, OK10, OK3210 : std_logic;
begin

-- Cálculo do resultado
OK3210 <= (not code(3)) and code(2) and code(1) and code(0);

-- Atribuição do valor de saída
codeOK <= OK3210;
OK  <= OK3210 and xopen;
Err <= (not OK3210) and xopen;

end behavior;
```

Alternativa:
Este código tem a mesma função, mas resulta num mapeamento para VHDL diferente.



Perde visibilidade dos sinais intermédios.



Descrição de circuitos em VHDL

cadeado_digital.vhd


```
-- Declaração de bibliotecas com pré-definições
library IEEE;
use IEEE.std_logic_1164.all;

entity cadeado_digital is
   port (
         code   : in  std_logic_vector(3 downto 0);
         xopen  : in  std_logic;
         codeOK : out std_logic;
         OK     : out std_logic;
         Err    : out std_logic
         );
end cadeado_digital;

architecture behavior of cadeado_digital is
-- declaração dos sinais (fios) internos
-- ao componente
signal OK32, OK10, OK3210 : std_logic;
begin

-- Cálculo do resultado
OK32   <= (not code(3)) and code(2);
OK10   <= code(1) and code(0);
OK3210 <= OK32 and OK10;

-- Atribuição do valor de saída
codeOK <= OK3210;
OK  <= OK3210 and xopen;
Err <= (not OK3210) and xopen;

end behavior;
```

Descrição de circuitos em VHDL

Portas lógicas simples

Atribuições simples


```
...
architecture behavior of meu_circuito is
-- declaração do sinal de selecção
signal A, B, C : std_logic;
signal vec1 : std_logic_vector(2 downto 0);
signal vec2 : std_logic_vector(2 downto 0);
...
begin
...
A <= '0'; -- atribuição do valor lógico zero
B <= '1'; -- atribuição do valor lógico um
C <= A;   -- atribuição do valor lógico dado em A

vec1 <= "011"; -- atribuição do número 3
vec2 <= A & B & C; -- atribuição do resultante da concatenação de
                   -- A, B e C. Assim vec2 tomará o valor 2.
...
end behavior;
```

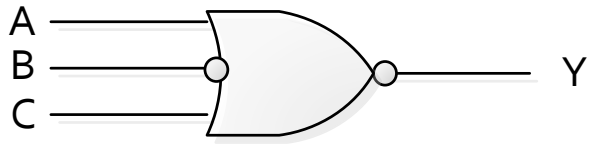
CONSTRUÇÕES POSSÍVEIS:

Poderão ser usadas quaisquer construções que tenham um mapeamento directo no logigrama original!

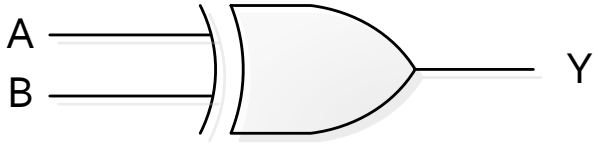
Portas lógicas simples


```
Y <= A and B and C;
```


```
Y <= not (A and B and C);
```




```
Y <= not (A or (not B) or C);
```




```
Y <= A xor B;
```

CONSTRUÇÕES POSSÍVEIS:

Poderão ser usadas quaisquer construções que tenham um mapeamento directo no logigrama original!

Simulação de circuitos

Criação de um ficheiro VHDL para simular e validar o funcionamento dos circuitos anteriormente descritos

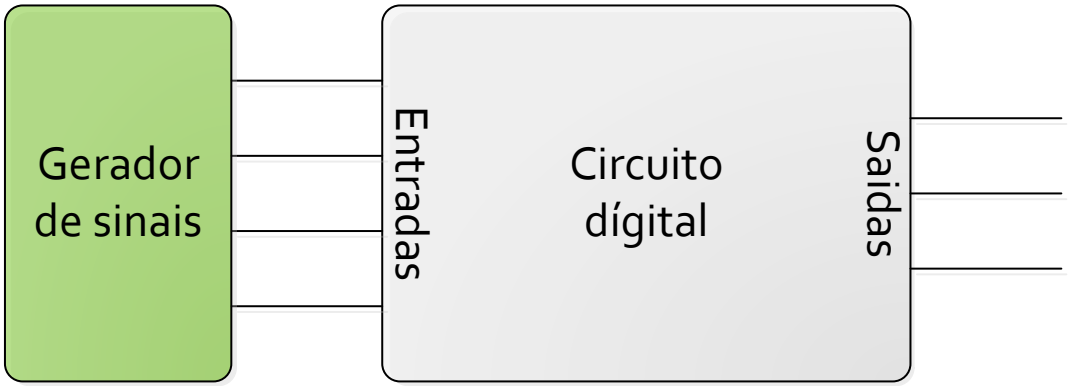
Simulação e teste de circuitos

- Para validar correctamente o funcionamento de um circuito digital é necessário verificar o valor das saídas do circuito para todas as combinações de:
  - Circuitos combinatórios: sinais de entrada
  - Circuitos sequenciais: sinais de entrada e estado do sistema

    Nota: o estado do sistema digital é dado pelo valor à saída dos elementos de memória, i.e., dos latches e dos flip-flops.

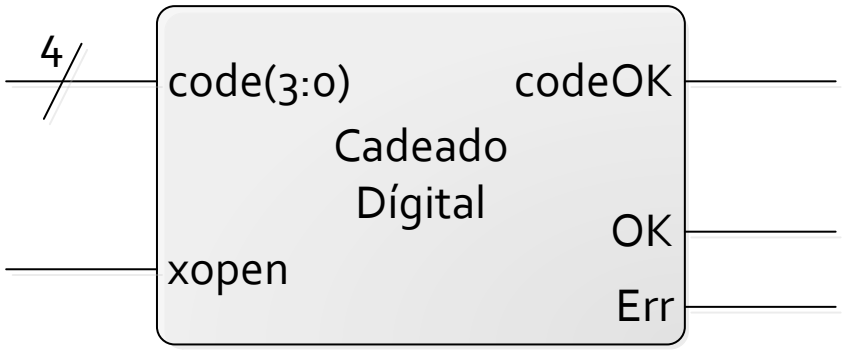
Simulação e teste de circuitos

- Para validar correctamente o funcionamento de um circuito digital é necessário verificar o valor das saídas do circuito.
  - É preciso desenvolver um módulo capaz de gerar os sinais de entrada do circuito digital



  - O gerador de sinais deve gerar todas as combinações de sinais de entrada

Tabela de verdade



O cadeado abre com o código $0111_2=7_{16}$.

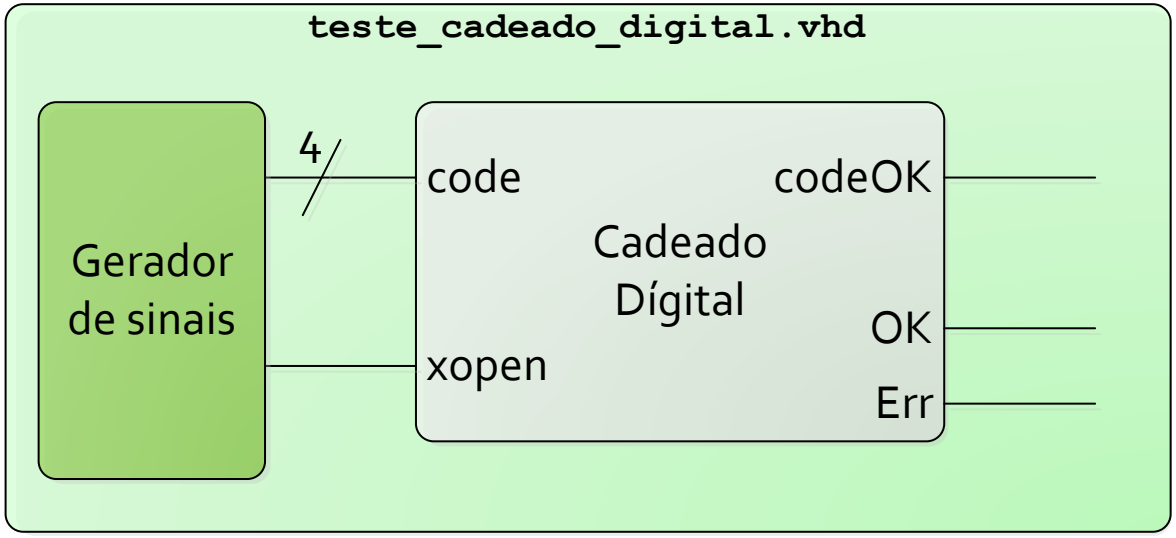
O gerador de sinais a projectar deve ser capaz de gerar todas as combinações de entradas da tabela de verdade de forma a ser possível verificar o valor da saída

- Tabela de verdade:

ENTRADAS		SAIDAS ESPERADAS		
code	xopen	codeOK	OK	Err
0000	0	0	0	0
0001	0	0	0	0
…	…	…	…	…
0110	0	0	0	0
0111	0	1	0	0
1000	0	0	0	0
…	…	…	…	…
0110	1	0	0	1
0111	1	1	1	0
1000	1	0	0	1
…	…	…	…	…

Estrutura do ficheiro de simulação

- Como se pode ver no diagrama abaixo, o ficheiro VHDL para simulação (e teste) de circuitos:
  - Necessita uma instancia do circuito a testar
  - Necessita de gerar os sinais de dados e/ou controlo do circuito
  - Não necessita de entradas ou saídas



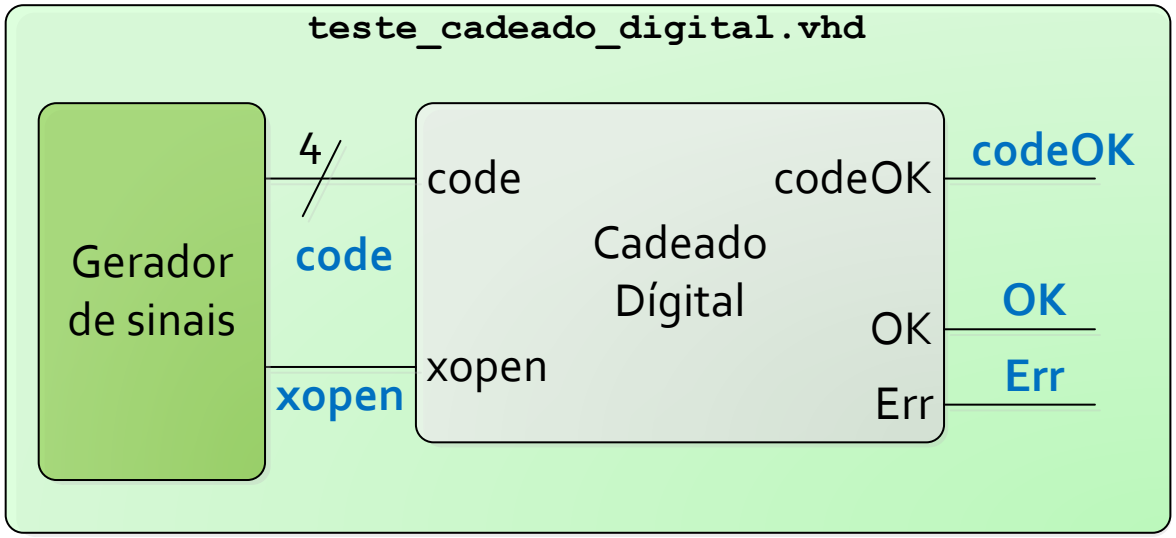
Descrição da entidade

- Descrição da entidade
  - Sem entradas/saídas

```
-- FICHEIRO cadeado_digital_testbench.vhd

-- Declaração de bibliotecas
library IEEE;
use IEEE.std_logic_1164.all;

-- Definição do nome da entidade, sem qualquer entrada ou saida
entity cadeado_digital_testbench is
end cadeado_digital_testbench;

architecture behavior of cadeado_digital_testbench is
...
```

Descrição da arquitectura

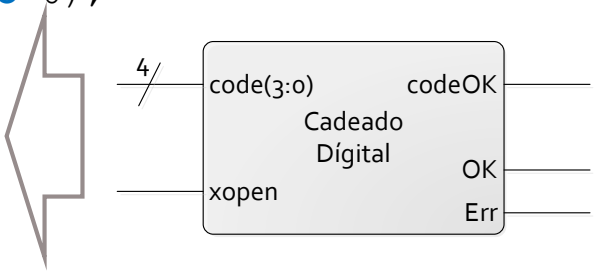
- Descrição da arquitectura
  - Declaração de componentes e sinais



Descrição da arquitectura

- Descrição da arquitectura
  - Declaração de componentes e sinais

```
...
architecture behavior of cadeado_digital_testbench is
-- Declaração do componente cadeado_digital original
component cadeado_digital
   port (
         code    : in  std_logic_vector(3 downto 0);
         xopen   : in  std_logic;
         codeOK  : out std_logic;
         OK      : out std_logic;
         Err     : out std_logic
         );
end component;
-- Declaração dos sinais para o testbench
signal code : std_logic_vector(3 downto 0);
signal xopen, codeOK, OK, Err : std_logic;

begin
```



Descrição da arquitectura

- Implementação
  - Descrição da unidade para teste

```
...
architecture behavior of cadeado_digital_testbench is
-- Declaração do componente cadeado_digital original
...
-- Declaração dos sinais para o testbench
...
begin

-- Declaração da unidade de teste... O nome dos sinais no circuito é neste
-- caso (não obrigatório) o mesmo que o nome dos sinais no componente
Utest: cadeado_digital port map (
         code => code, xopen => xopen,
         codeOK => codeOK, OK => OK, Err => Err);

-- Descrição do gerador de sinais
...
end behavior;
```

Descrição da arquitectura

- Implementação
  - Descrição do gerador de sinais
    - Simulação de 1 linha da tabela de verdade a cada 10 ns

```
...
-- Descrição do gerador de sinais
process
begin
   -- valor dos sinais para a 1ª linha da tabela de verdade
   ...
   wait for 10 ns;
   -- valor dos sinais para a 2ª linha da tabela de verdade
   ...
   wait for 10 ns;
   -- valor dos sinais para a n-ésima linha da tabela de verdade
   ...
   wait; -- end of signal generation
end process;

end behavior;
```

Descrição da arquitectura

- Implementação
  - Descrição do gerador de sinais
    - Simulação de 1 linha da tabela de verdade a cada 10 ns

O gerador de sinais acaba na ultima linha da tabela de verdade

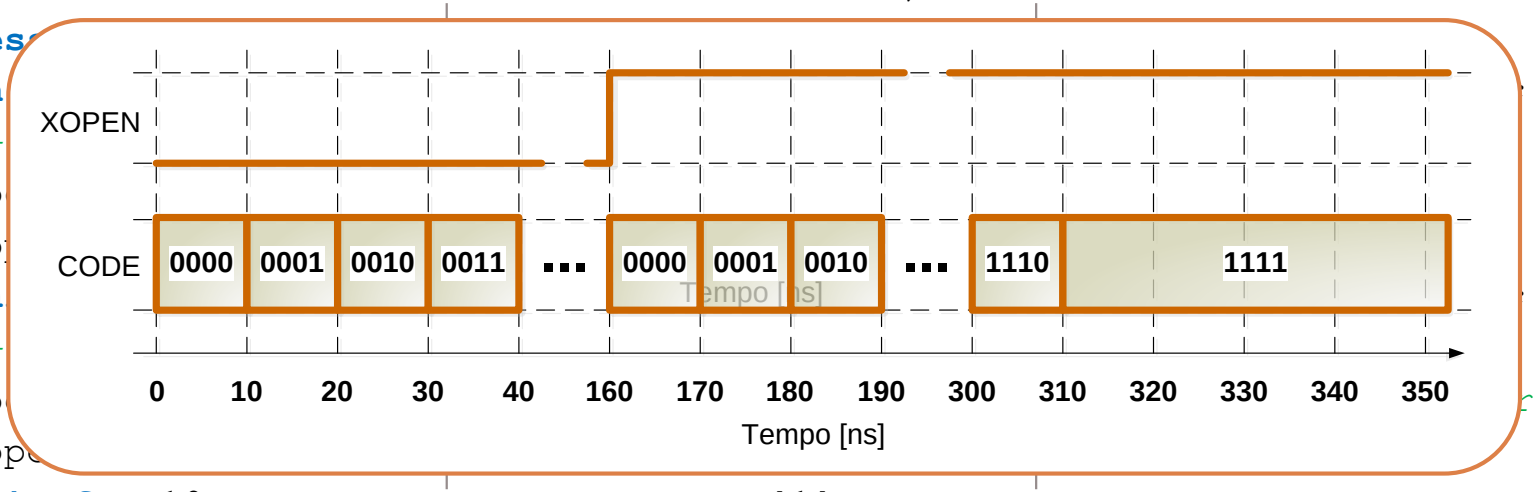
```
...
-- Gerador de sinais
process
begin
    -- 1ª linha
    code  <= "0000";
    xopen <= '0';
    wait for 10 ns;
    -- 2ª linha
    code  <= "0001";
    xopen <= '0';
    wait for 10 ns;
    -- 3ª linha
    code  <= "0010";
    xopen <= '0';
    wait for 10 ns;
    -- 4ª linha
    code  <= "0011";
    xopen <= '0';
    wait for 10 ns;
    ...
    -- 17ª linha
    code  <= "0000";
    xopen <= '1';
    wait for 10 ns;
    -- 18ª linha
    code  <= "0001";
    xopen <= '1';
    wait for 10 ns;
    -- 19ª linha
    code  <= "0010";
    xopen <= '1';
    wait for 10 ns;
    ...
    -- 31ª linha
    code  <= "1110";
    xopen <= '1';
    wait for 10 ns;
    -- 32ª linha
    code  <= "1111";
    xopen <= '1';
    wait; -- forever
end process;

end behavior;
```

Descrição da arquitectura

- Implementação
  - Descrição do gerador de sinais
    - Simulação de 1 linha da tabela de verdade a cada 10 ns

O gerador de sinais acaba na ultima linha da tabela de verdade

```
...
-- Gerador de sinais
proces
begin
    --
    co
    xo
    wa
    --
    co
    xope
    wait for 10 ns;
    -- 3ª linha
    code  <= "0010";
    xopen <= '0';
    wait for 10 ns;
    -- 4ª linha
    code  <= "0011";
    xopen <= '1';
    wait for 10 ns;
    -- 19ª linha
    code  <= "0010";
    xopen <= '1';
    wait for 10 ns;
    ...
end behavior;
```



Descrição da arquitectura

- Implementação
  - Descrição do gerador de sinais
    - Simulação de 1 linha da tabela de verdade a cada 10 ns

O gerador de sinais repete após a ultima linha da tabela de verdade

```
...
-- Gerador de sinais
process
begin
    -- 1ª linha
    code  <= "0000";
    xopen <= '0';
    wait for 10 ns;
    -- 2ª linha
    code  <= "0001";
    xopen <= '0';
    wait for 10 ns;
    -- 3ª linha
    code  <= "0010";
    xopen <= '0';
    wait for 10 ns;
    -- 4ª linha
    code  <= "0011";
    xopen <= '0';
    wait for 10 ns;
    ...
    -- 17ª linha
    code  <= "0000";
    xopen <= '1';
    wait for 10 ns;
    -- 18ª linha
    code  <= "0001";
    xopen <= '1';
    wait for 10 ns;
    -- 19ª linha
    code  <= "0010";
    xopen <= '1';
    wait for 10 ns;
    ...
    -- 31ª linha
    code  <= "1110";
    xopen <= '1';
    wait for 10 ns;
    -- 32ª linha
    code  <= "1111";
    xopen <= '1';
    wait for 10 ns;
end process;

end behavior;
```

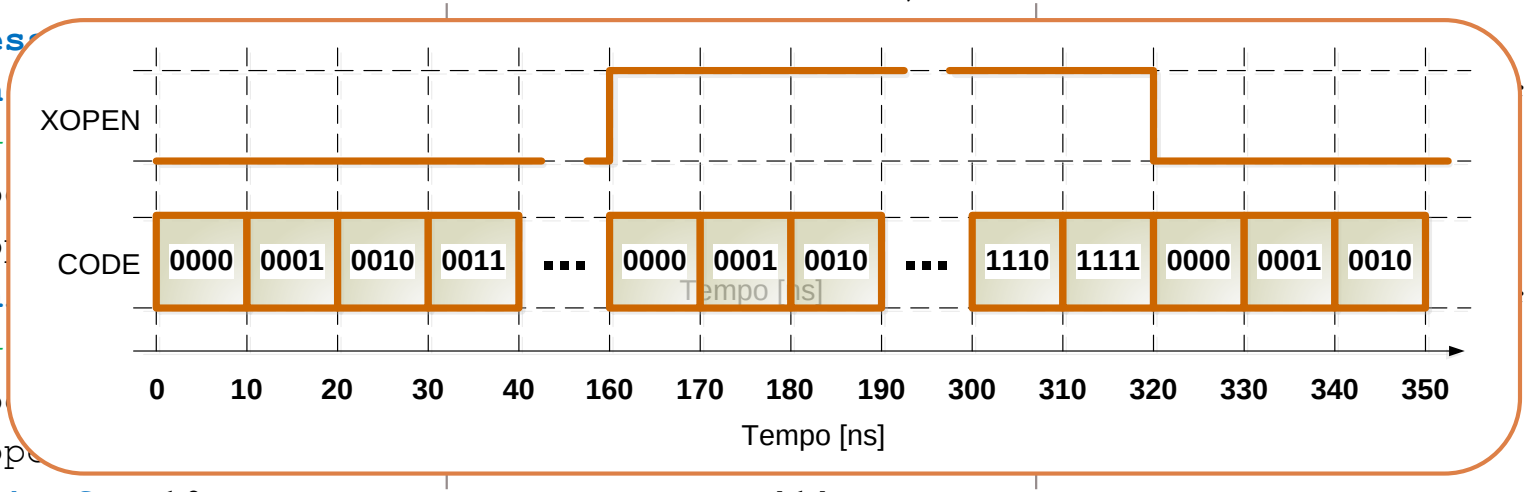
Descrição da arquitectura

- Implementação
  - Descrição do gerador de sinais
    - Simulação de 1 linha da tabela de verdade a cada 10 ns

O gerador de sinais repete após a ultima linha da tabela de verdade

```
...
-- Gerador de sinais
proces
begin
    --
    co
    xo
    wa
    --
    co
    xope
    wait for 10 ns;
    -- 3ª linha
    code  <= "0010";
    xopen <= '0';
    wait for 10 ns;
```

```
    -- 4ª linha
    code  <= "0011";
```




```
    xopen <= '1';
    wait for 10 ns;
    -- 19ª linha
    code  <= "0010";
    xopen <= '1';
```

```
    wait for 10 ns;
    ...
end behavior;
```

Descrição da arquitectura

- Implementação

SOLUÇÃO ALTERNATIVA

  - Descrição do gerador de sinais
    - Separação da tabela de verdade em duas partes: sinais `code` e `xopen`

```
...
-- Geração do sinal code
process
begin
   code <= "0000";
   wait for 10 ns;
   code <= "0001";
   wait for 10 ns;
   ...
   code <= "1110";
   wait for 10 ns;
   code <= "1111";
   wait for 10 ns;
end process;
```

```
-- Geração do sinal xopen
process
begin
   xopen <= '0';
   wait for 16*10 ns;
   xopen <= '1';
   wait for 16*10 ns;
end process;

end behavior;
```

Descrição da arquitectura

- Implementação
  - Descrição do gerador de sinais
    - Utilização de macros (contador e inversor)

SOLUÇÃO ALTERNATIVA

```
process
begin
   code <= code + 1;
   wait for 10 ns;
end process;

-- Geração do sinal xopen
process
begin
   xopen <= not xopen;
   wait for 16*10 ns;
end process;

end behavior;
```

Requer:
1. **a inicialização dos sinais**
2. **A utilização da biblioteca `ieee.std_logic_unsigned`**

Simulação e teste do circuito "*Cadeado Digital*"

cadeado_digital_testbench_C.vhd


```
-- FICHEIRO cadeado_digital_testbench_C.vhd

-- Declaração de bibliotecas
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- Definição da entidade
entity testbench_C is
end testbench_C;

architecture behavior of testbench_C is

-- Declaração do componente
-- cadeado_digital original
component cadeado_digital
 port (
    code    : in  std_logic_vector(3 downto 0);
    xopen   : in  std_logic;
    codeOK  : out std_logic;
    OK      : out std_logic;
    Err     : out std_logic
 );
end component;
```

```
-- Declaração dos sinais para o testbench
signal code : std_logic_vector(3 downto 0) := "0000";
signal xopen, codeOK, OK, Err : std_logic := '0';

begin

-- Declaração da unidade de teste
utest: cadeado_digital port map (
           code => code, xopen => xopen,
           codeOK => codeOK, OK => OK, Err => Err);

-- descrição do gerador para o sinal code
gen_code: process
begin
   code <= code + 1;
   wait for 10 ns;
end process;

-- descrição do gerador para o sinal xopen
gen_open: process
begin
   xopen <= not xopen;
   wait for 16*10 ns;
end process;

end behavior;
```