

# DESCRIÇÃO DE CIRCUITOS DIGITAIS EM VHDL

*Slides by: Pedro Tomás*

# Outline

2

Sistemas Digitais, 2013

- Linguagens de descrição de Hardware [\[LINK\]](#)
- Introdução a VHDL [\[LINK\]](#)
  - ▣ Descrição de estruturas básicas em VHDL [\[LINK\]](#)
  - ▣ Exemplos:
    - Cadeado digital [\[LINK\]](#)
    - Cadeado digital (V2) [\[LINK\]](#)
    - Unidade aritmética [\[LINK\]](#)
- Simulação de circuitos em VHDL [\[LINK\]](#)
- Xilinx ISE [\[LINK\]](#)

## Etapas

3

Sistemas Digitais, 2013

### 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

# Projecto de um circuito digital

## Implementação física

4

Sistemas Digitais, 2013

### □ 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

# Hardware Description Language (HDL)

## VHDL, para que serve?

5

Sistemas Digitais, 2013

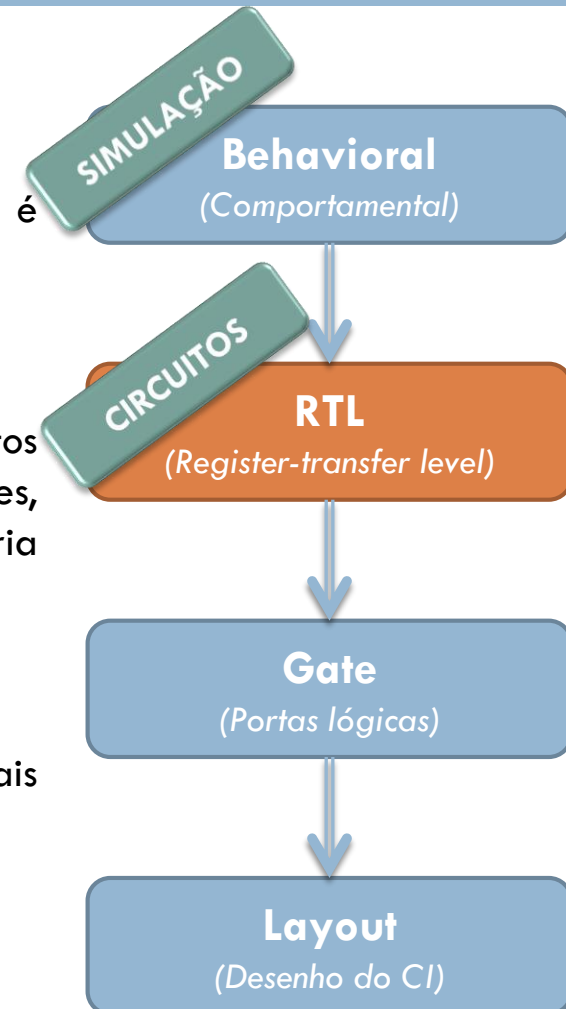
- VHDL (e *Verilog*) serve para:
  - ▣ Descrever circuitos digitais
  - ▣ Simular circuitos digitais
    - Verificar a funcionalidade, testar e corrigir os erros
  
- VHDL (e *Verilog*) não é uma linguagem de programação
  - ▣ Os circuitos digitais não se programam... descrevem-se!
  - ▣ Escrever código 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

6

Sistemas Digitais, 2013

- **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, decodificadores, 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

**V**ery High Speed Integrated Circuit (VHSIC)

**H**ardware

**D**escription

**L**anguage

# Descrição de circuitos em VHDL

8

Sistemas Digitais, 2013

- 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



## Descrição de circuitos em VHDL

# Estrutura típica de um ficheiro VHDL

9

Sistemas Digitais, 2013

```
-- 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

10

Sistemas Digitais, 2013

- 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 atribuídos automaticamente pela ferramenta (Xilinx ISE) quando um sinal (fio) não tiver valor atribuído ou não for possível a sua determinação (ex: quando são atribuídos 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

12

Sistemas Digitais, 2013

- 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)
```

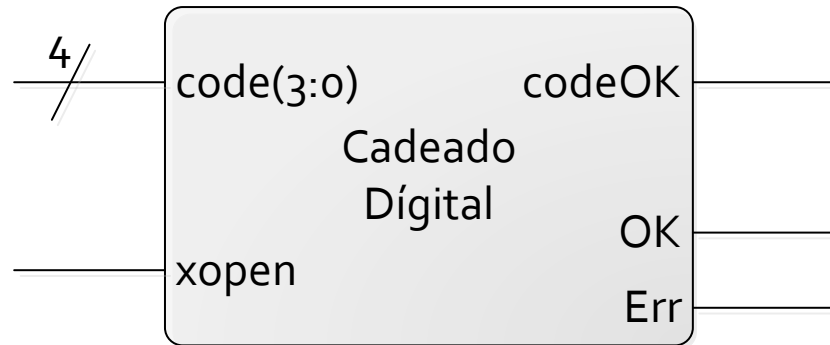
Cada um do tipo **std\_logic**

## Descri  o de circuitos em VHDL

# Exemplo: cadeado digital

13

Sistemas Digitais, 2013



O cadeado abre  
com o c digo  
 $0111_2 = 7_{16}$

```
-- Defini  o do nome da entidade e
-- dos sinais (fios) de entrada/s  da
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 n o  
deve ser terminada  
por “;”

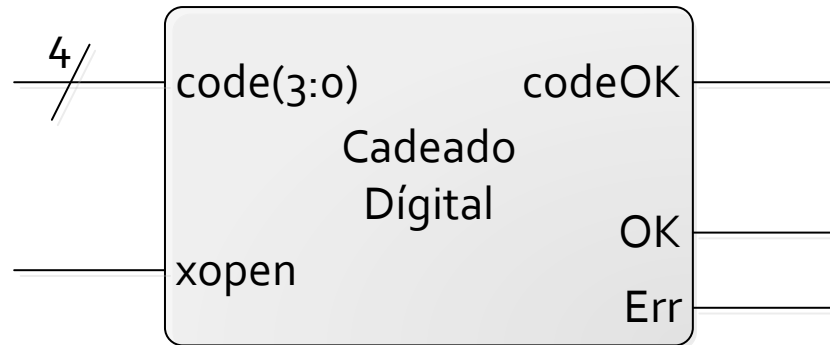
Nota: J  que a palavra `open` est  reservada em VHDL, n    poss vel representar sinais com este nome. Assim, o sinal de comando para abertura do cadeado foi renomeado para “`xopen`”

## Descrição de circuitos em VHDL

# Exemplo: cadeado digital

14

Sistemas Digitais, 2013



O cadeado abre  
com o código  
 $0111_2 = 7_{10}$ .

```
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 “;”

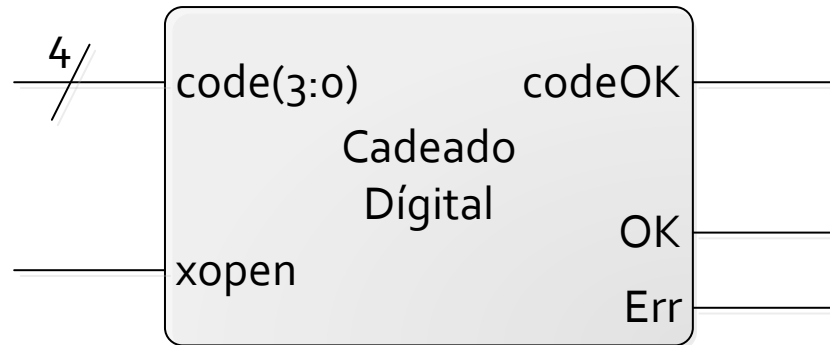
A ultima linha não  
deve ser terminada  
por “;”

## Descri  o de circuitos em VHDL

# Exemplo: cadeado digital

15

Sistemas Digitais, 2013



O cadeado abre  
com o c digo  
 $0111_2 = 7_{16}$ .

-   Descri  o da implementa  o do circuito  
"cadeado\_digital"

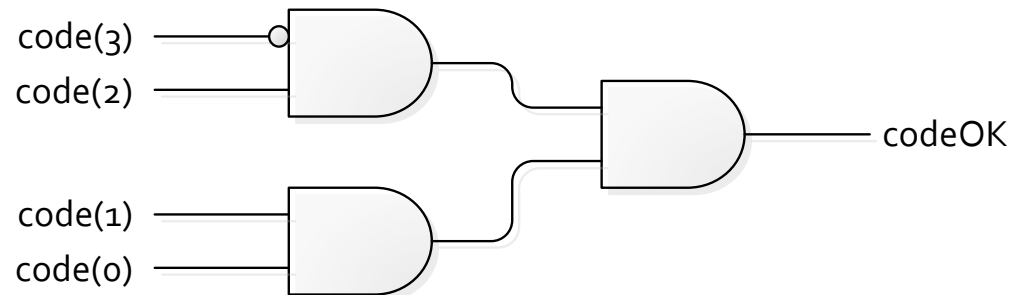
## Descrição de circuitos em VHDL

# Exemplo: cadeado digital

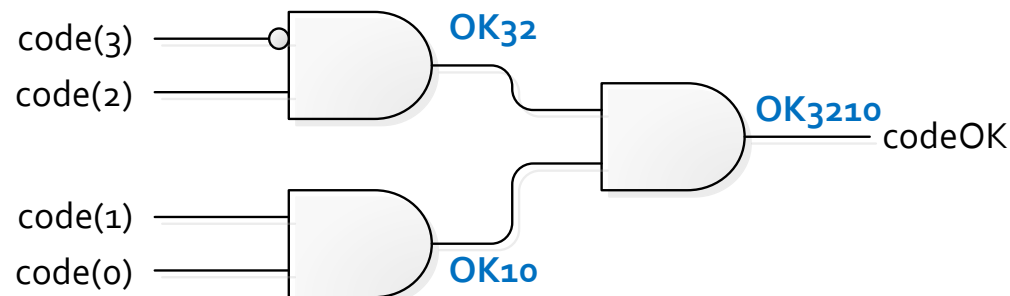
16

Sistemas Digitais, 2013

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



- Em VHDL deverá ser dado um nome a cada um dos fios intermédios:

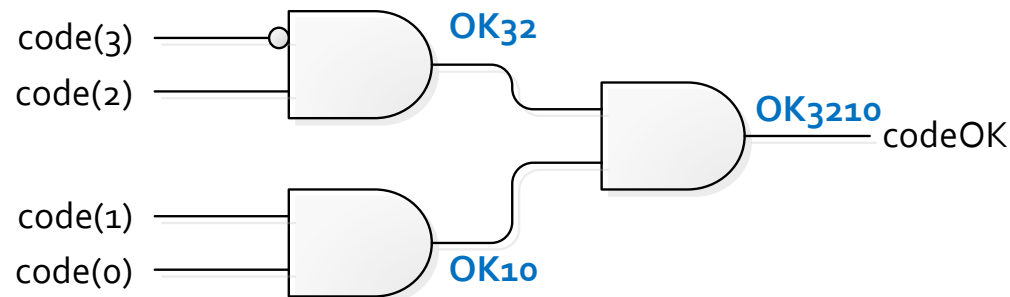




## Exemplo: cadeado digital

17

Sistemas Digitais, 2013



□ Assim   necess rio declarar os sinais OK32, OK10 e OK3210

□ A declara  o de sinais   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   poss vel ter sinais (fios) de diferentes tipos

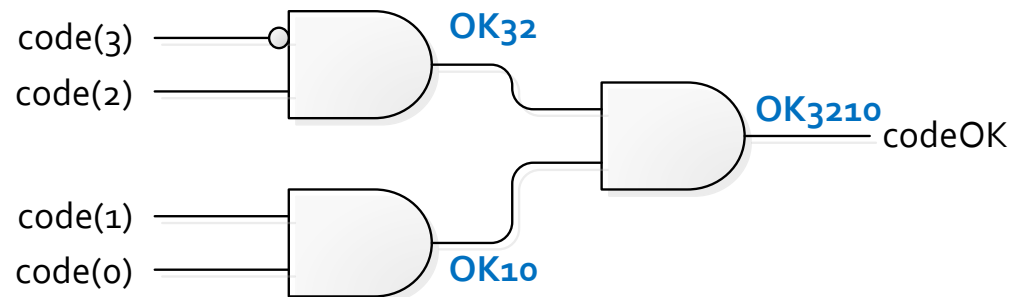
▣ No entanto a declara  o anterior obriga a que os sinais 1 e 2 tenham o mesmo tipo

## Descrição de circuitos em VHDL

# Exemplo: cadeado digital

18

Sistemas Digitais, 2013



- 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...

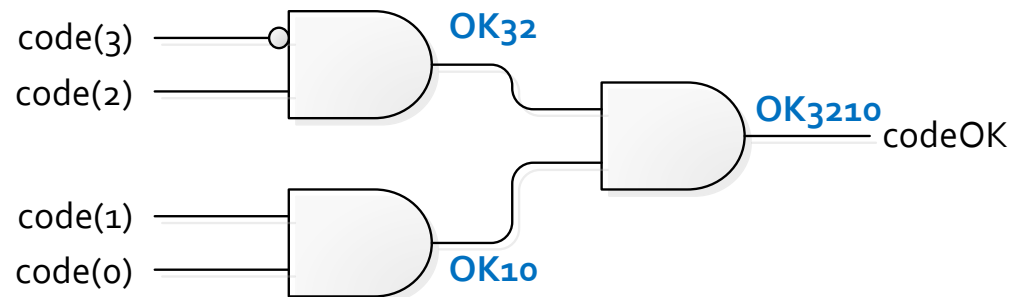
O Xilinx ISE normalmente usa a designação behavior

## Descrição de circuitos em VHDL

# Exemplo: cadeado digital

19

Sistemas Digitais, 2013



- 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

20

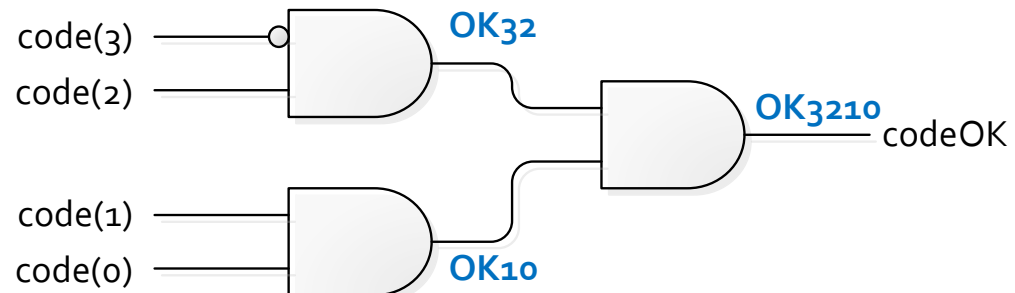
Sistemas Digitais, 2013

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

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

  -- Atribui  o do valor de sa  da
  codeOK <= OK3210;

end behavior;
```



## Descri  o de circuitos em VHDL

# Exemplo: cadeado digital

21

Sistemas Digitais, 2013

```

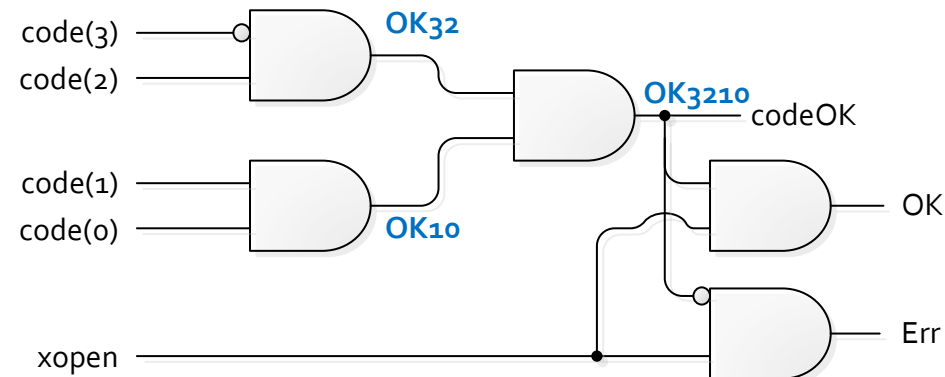
...
architecture behavior of cadeado_digital is
  -- declara  o dos sinais (fios) internos ao componente
  signal OK32, OK10, OK3210 : std_logic;
begin

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

  -- Atribui  o do valor de sa  da
  codeOK  <= OK3210;
  OK      <= OK3210 and xopen;
  Err     <= (not OK3210) and xopen;

end behavior;

```



# Descrição de circuitos em VHDL

## Exemplo: cadeado digital

22

Sistemas Digitais, 2013

```
...
architecture behavior of cadeado_digital
-- declaração dos sinais (fios) internos
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;
```

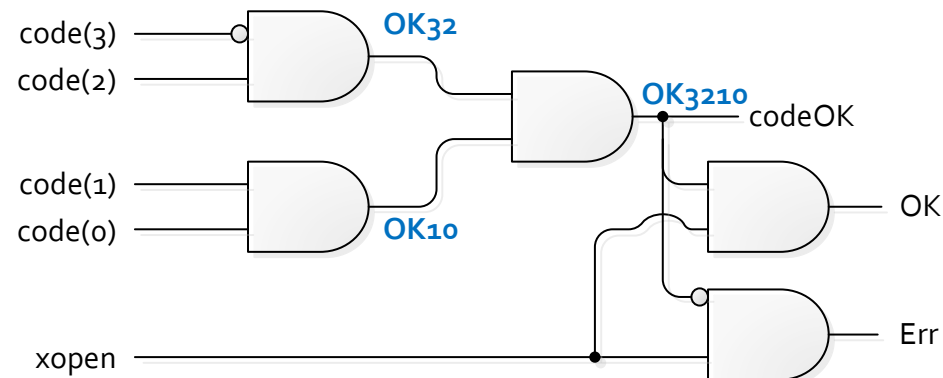
```
end behavior;
```

### ERRADO:

Este código tem a mesma função, mas o mapeamento do circuito para VHDL está errado,



Perde visibilidade dos sinais intermédios.



# Descrição de circuitos em VHDL

## cadeado\_digital.vhd

23

Sistemas Digitais, 2013

```
-- 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;
```

# Cadeado digital (v2)

## Considere-se agora os seguintes sinais:

- Sinal de entrada “code” – código inserido.
- Sinais de entrada “xopen”/“xclose” – permite abrir/fechar o cadeado se o código inserido estiver correcto e o estado do cadeado for fechado/aberto, respectivamente.
- Sinais de saída “OK”/“Err” – indicação de que o cadeado foi aberto/fechado (conforme o caso) com sucesso/erro.
- Sinal de saída “lock\_state” – indica o estado do cadeado, nomeadamente, 0 – fechado, 1 – aberto.
- Sinais de controlo (entrada) “clk” e “reset” – sinais de relógio e de re-inicialização do cadeado.

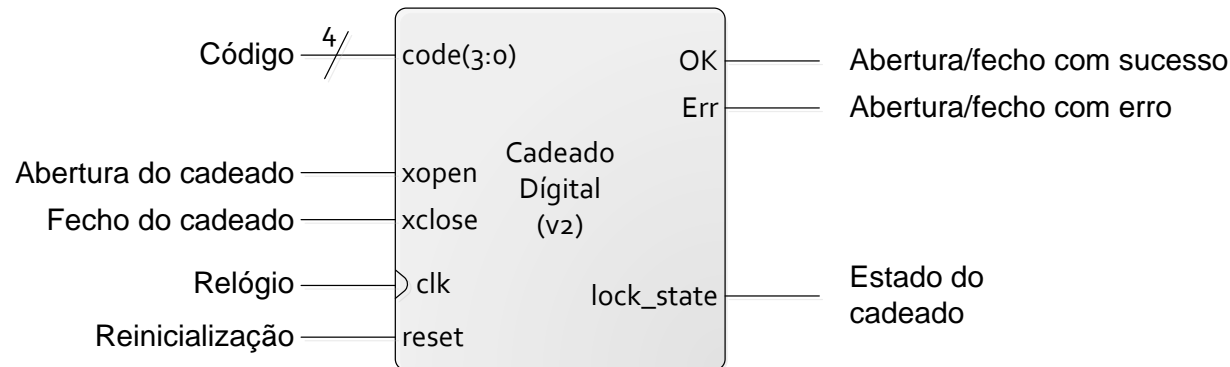


## Descrição da entidade

25

Sistemas Digitais, 2013

### 1. Identificação do componente:



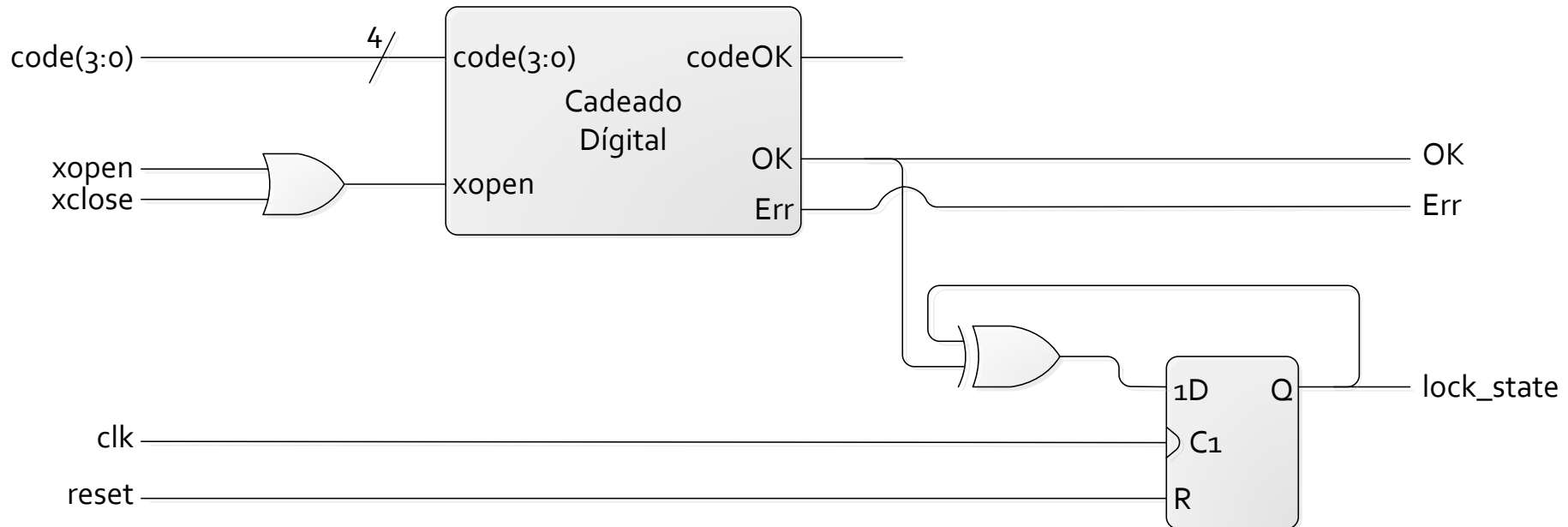
```
entity cadeado_digital_v2 is
  port (
    code      : in  std_logic_vector(3 downto 0);
    xopen     : in  std_logic;
    xclose    : in  std_logic;
    clk       : in  std_logic;
    reset     : in  std_logic;
    OK        : out std_logic;
    Err       : out std_logic;
    lock_state : out std_logic
  );
end cadeado_digital_v2;
```

# Logigrama do circuito

26

Sistemas Digitais, 2013

## 2. Desenho do logigrama do cadeado digital (v2)



ps: o objectivo destes slides não é perceber como se chega a este circuito, apenas como o descrever em VHDL. Assim, assume-se que o circuito está correctamente projectado.

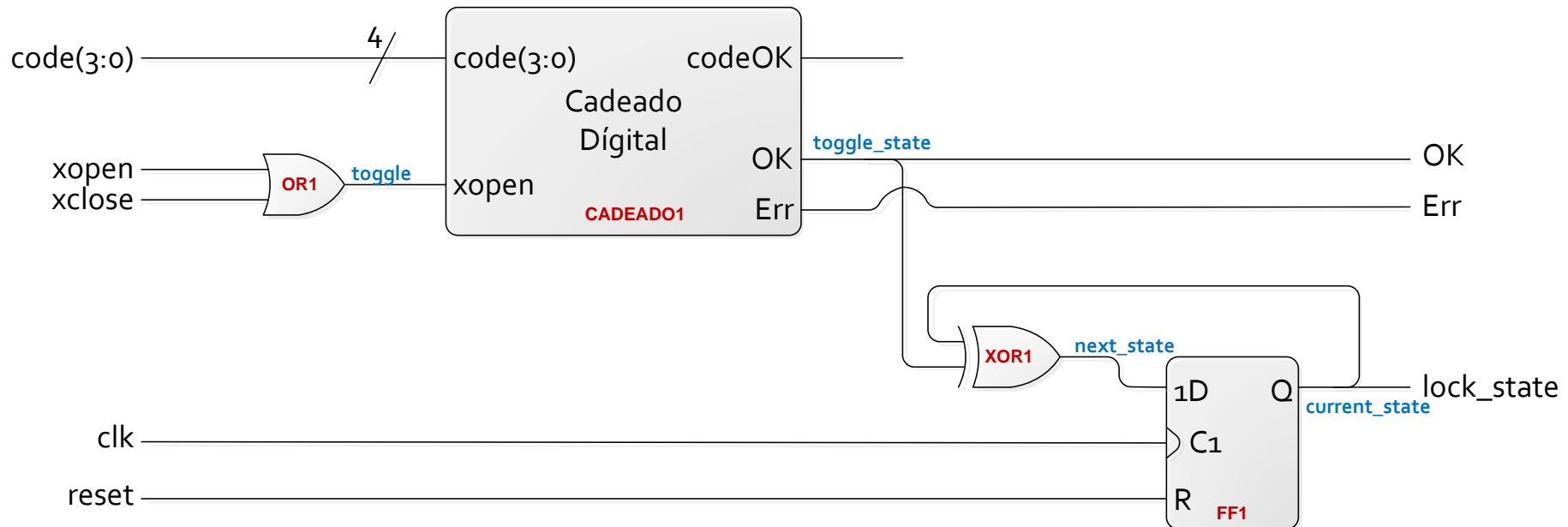
# Logigrama do circuito

27

Sistemas Digitais, 2013

## 2. Desenho do logigrama do cadeado d gital (v2)

- a) Nomea  o dos sinais utilizados internamente (azul) e do nome dos circuitos (vermelho)



ps: o objectivo destes slides n o   perceber como se chega a este circuito, apenas como o descrever em VHDL. Assim, assume-se que o circuito est  correctamente projectado.

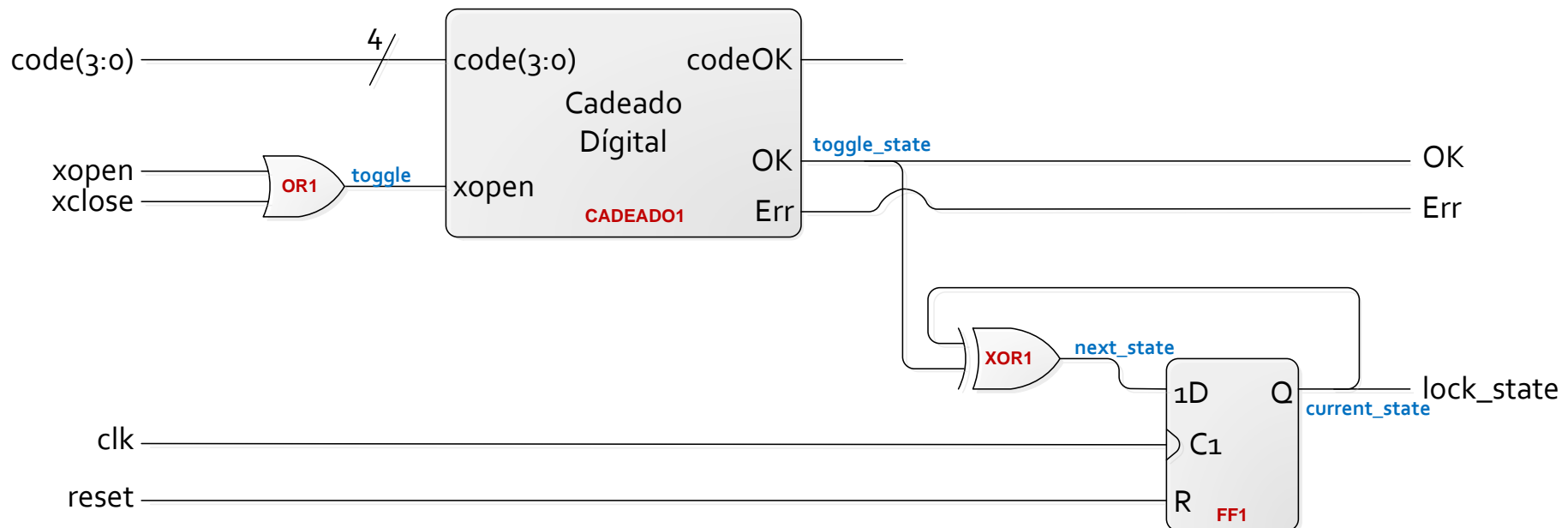
# Logigrama do circuito

28

Sistemas Digitais, 2013

## 3. Implementação da arquitectura

- Para utilizar o componente “cadeado\_digital”, descrito no ficheiro “cadeado\_digital.vhd”, devem ser tomados os seguintes passos:
  - A. Declarar o componente
  - B. Utilizar uma instância do componente



## Logigrama do circuito

29

Sistemas Digitais, 2013

### 3. Implementação da arquitectura

- Para utilizar o componente “cadeado\_digital”, descrito no ficheiro “cadeado\_digital.vhd”, devem ser tomados os seguintes passos:

**A. Declarar o componente**

**B. Utilizar uma instância do componente**

-- Declaração da entidade, colocada entre  
-- “architecture” e “begin”

```
component <NOME_DO_COMPONENTE>
  port (
    <SINAL1> : <IN/OUT> <TIPO_DE_SINAL>;
    ...
    <SINALN> : <IN/OUT> <TIPO_DE_SINAL>
  );
end component;
```

*Definição da entidade a utilizar, tal como definido no topo do  
ficheiro vhd correspondente*

```
entity NOME_DO_COMPONENTE is
  port (
    <SINAL1> : <IN/OUT> <TIPO_DE_SINAL>;
    ...
    <SINALN> : <IN/OUT> <TIPO_DE_SINAL>
  );
end NOME_DO_COMPONENTE;
```

- A declaração de um componente é uma cópia quase perfeita da descrição da entidade. As únicas diferenças residem na primeira e ultima linhas

## Logigrama do circuito

30

Sistemas Digitais, 2013

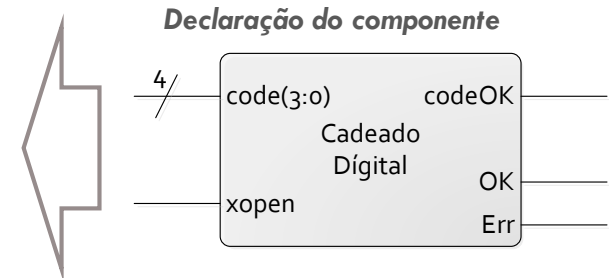
### 3. Implementação da arquitectura

- Para utilizar o componente “cadeado\_digital”, descrito no ficheiro “cadeado\_digital.vhd”, devem ser tomados os seguintes passos:

#### A. Declarar o componente

...

```
architecture behavior of cadeado_digital_v2 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 (fios) internos ao componente
signal toggle, toggle_state, next_state, current_state : std_logic;
begin
```



## Logigrama do circuito

31

Sistemas Digitais, 2013

### 3. Implementação da arquitectura

- Para utilizar o componente “cadeado\_digital”, descrito no ficheiro “cadeado\_digital.vhd”, devem ser tomados os seguintes passos:

#### B. Utilizar uma instância do componente

```
-- Utilização de uma instancia do componente
<NOME_DA_INSTANCIA>: <NOME_DO_COMPONENTE>
    port map (
        <SINAL_DO_COMPONENTE> => <NOME_DO_SINAL_NO_CIRCUITO>,
        <SINAL_DO_COMPONENTE> => <NOME_DO_SINAL_NO_CIRCUITO>,
        ...
        <SINAL_DO_COMPONENTE> => <NOME_DO_SINAL_NO_CIRCUITO>
    );
```

- Deverão ser ligados todos os sinais do componente, com eventual excepção dos sinais com direcção “out”, os quais podem ser desligado através da atribuição:

```
<SINAL_DO_COMPONENTE> => open
```

## Logigrama do circuito

32

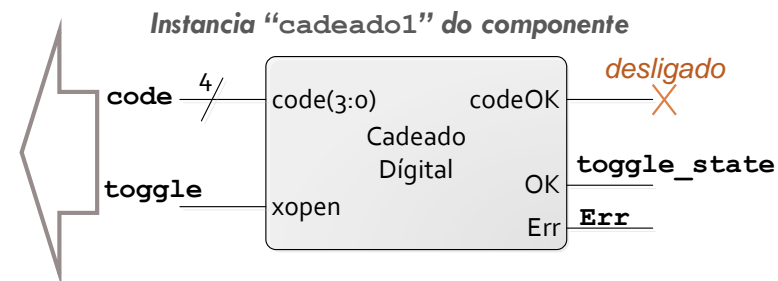
Sistemas Digitais, 2013

### 3. Implementa  o da arquitectura

- Para utilizar o componente “cadeado\_digital”, descrito no ficheiro “cadeado\_digital.vhd”, devem ser tomados os seguintes passos:

#### B. Utilizar uma inst ncia do componente

```
...
architecture behavior of cadeado_digital_v2 is
-- Declara  o do componente cadeado_digital original
...
-- declara  o dos sinais (fios) internos ao componente
signal toggle, toggle_state, next_state, current_state : std_logic;
begin
-- Utiliza  o de 1 instancia do componente "cadeado_digital"
cadeado1: cadeado_digital port map(
    code => code,
    xopen => toggle,
    codeOK => open,
    OK => toggle_state,
    Err => Err
);
...
```





## Logigrama do circuito

33

Sistemas Digitais, 2013

### 3. Implementação da arquitectura (continuação)

**begin**

*-- Utilização de 1 instancia do componente "cadeado\_digital"*

```
cadeado1: cadeado_digital port map(
    code => code,
    xopen => toggle,
    codeOK => open,
    OK => toggle_state,
    Err => Err
);
```

*-- porta OR*

```
toggle <= xopen or xclose;
```

*-- porta XOR: funciona como um inversor controlado pelo sinal toggle\_state*

```
next_state <= current_state xor toggle_state;
```

*-- FLIP-FLOP tipo D com reset*

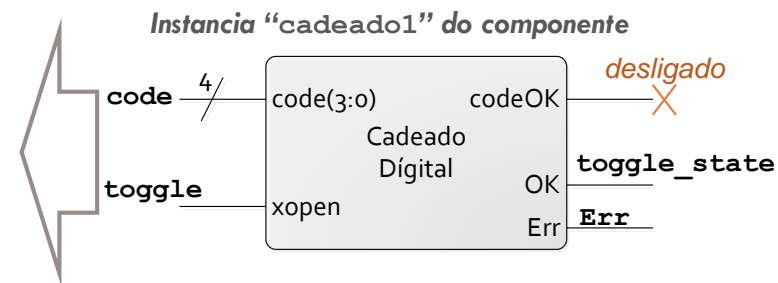
```
current_state <= '0' when reset='1' else next_state when rising_edge(clk);
```

*-- Atribuição do resultado*

```
lock_state <= current_state;
```

```
OK <= toggle_state;
```

```
end behavior;
```



# Cadeado digital (versão 2)

## cadeado\_digital\_v2.vhd

34

Sistemas Digitais, 2013

```
-- FICHEIRO cadeado_digital_v2.vhd
-- Declaração de bibliotecas
library IEEE;
use IEEE.std_logic_1164.all;
-- Declaração da entidade
entity cadeado_digital_v2 is
    port (
        code    : in  std_logic_vector(3 downto 0);
        xopen   : in  std_logic;
        xclose  : in  std_logic;
        clk     : in  std_logic;
        reset   : in  std_logic;
        OK      : out std_logic;
        Err     : out std_logic;
        lock_state : out std_logic
    );
end cadeado_digital_v2;

architecture behavior of cadeado_digital_v2 is
-- Declaração do componente cadeado_digital
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 internos
signal toggle, toggle_state : std_logic;
signal next_state, current_state : std_logic;

begin

-- Utilização de 1 instancia do "cadeado_digital"
cadeado1: cadeado_digital port map(
    code => code,
    xopen => toggle,
    codeOK => open,
    OK => toggle_state,
    Err => Err
);

-- porta OR
toggle <= xopen or xclose;

-- porta XOR
next_state <= current_state xor toggle_state;

-- FLIP-FLOP tipo D com reset
current_state <= '0' when reset='1' else
    next_state when rising_edge(clk);

-- Atribuição das restantes saidas
lock_state <= current_state;
OK <= toggle_state;

end behavior;
```

# Descrição de circuitos em VHDL

**Portas lógicas simples**

**Flip-flops**

**Multiplexers**

**Decodificadores**

**Registos**

## Atribuições simples

36

Sistemas Digitais, 2013

```
...
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 logograma original!

# Portas lógicas simples

37

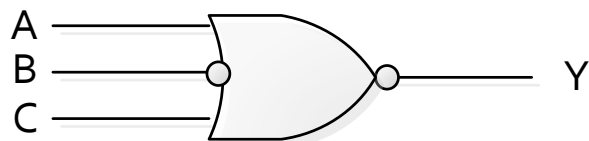
Sistemas Digitais, 2013



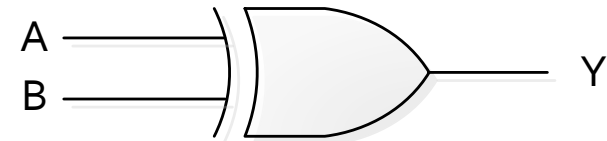
```
Y <= A and B and C;
```



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



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



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

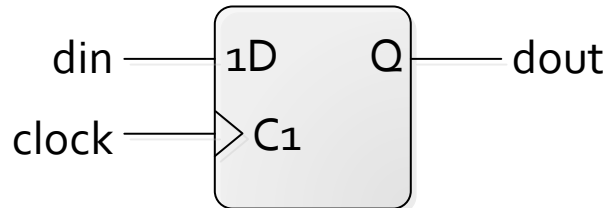
## CONSTRUÇÕES POSSÍVEIS:

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

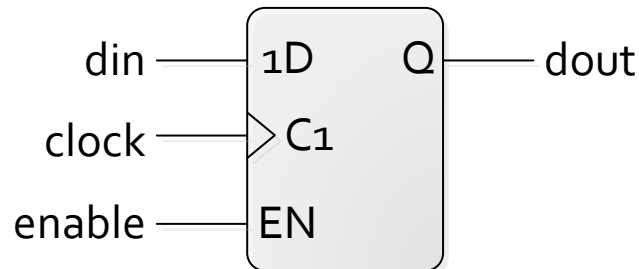
# Flip-flops

38

Sistemas Digitais, 2013



```
dout <= din when rising_edge(clock);
```



```
dout <= din when rising_edge(clock) and enable='1';
```

## CONSTRUÇÕES POSSÍVEIS:

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

**NOTA:** Esta é a única situação onde se aceitam os operadores **and/or/xor/...** após o operador **when**.

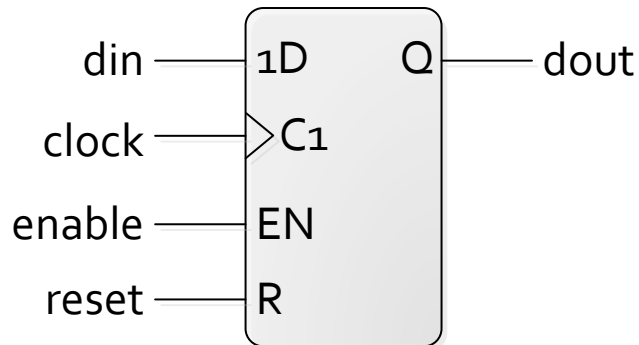
**Embora seja possível a utilização em outros casos, tal não será permitido no contexto da UC de SD.**

## Descri  o de circuitos em VHDL

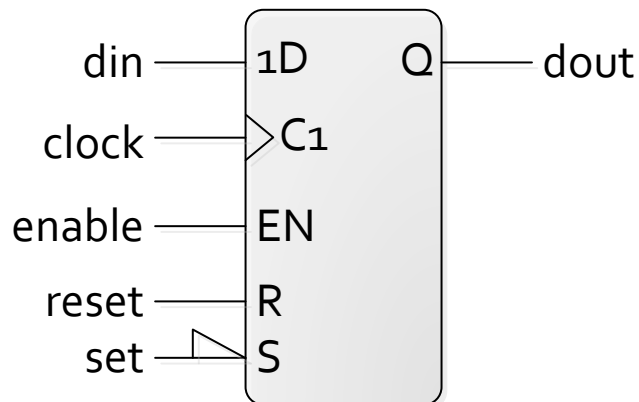
# Flip-flops com set/reset assincronos

39

Sistemas Digitais, 2013



```
dout <= '0' when reset='1' else
    din when rising_edge(clock) and enable='1';
```



```
dout <= '0' when reset='1' else
    '1' when set='0' else
    din when rising_edge(clock) and enable='1';
```

### CONSTRU  ES POSS  VEIS:

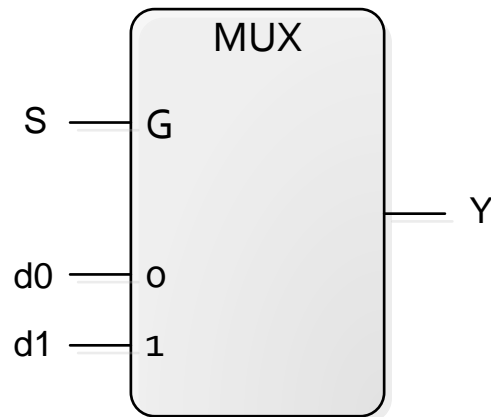
Poder  o ser usadas quaisquer constru  es que tenham um mapeamento directo no logograma original!

**NOTA:** Esta   a  nica situa  o onde se aceitam os operadores **and/or/xor** ap s o operador **when**.

## Multiplexer 2:1 de 1 bit

40

Sistemas Digitais, 2013



### CONSTRU  ES POSS  VEIS:

Poder  o ser usadas quaisquer constru  es que tenham um mapeamento directo no logigrama original!

```
Y <= d0 when S='0' else d1;
```

**ou**

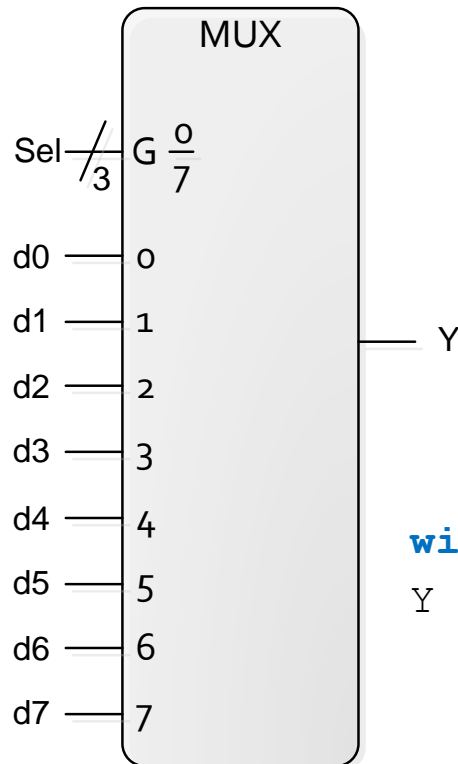
```
With S select
  Y <= d0 when '0',
      d1 when others;
```



## Multiplexer 8:1 de 1 bit

41

Sistemas Digitais, 2013



### CONSTRUÇÕES POSSÍVEIS:

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

**with** Sel **select**

```
Y <= d0 when "000",
      d1 when "001",
      d2 when "010",
      d3 when "011",
      d4 when "100",
      d5 when "101",
      d6 when "110",
      d7 when others;
```

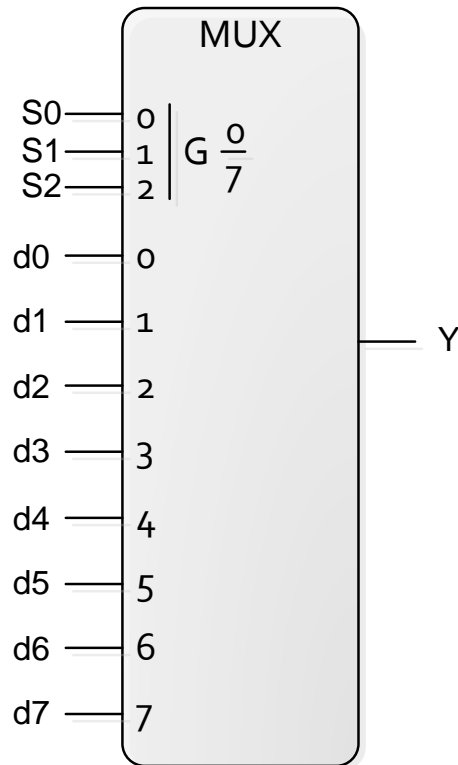
**ou**

```
Y <= d0 when Sel="000" else
      d1 when Sel="001" else
      d2 when Sel="010" else
      d3 when Sel="011" else
      d4 when Sel="100" else
      d5 when Sel="101" else
      d6 when Sel="110" else
      d7;
```

## Multiplexer 8:1 de 1 bit

42

Sistemas Digitais, 2013



### CONSTRUÇÕES POSSÍVEIS:

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

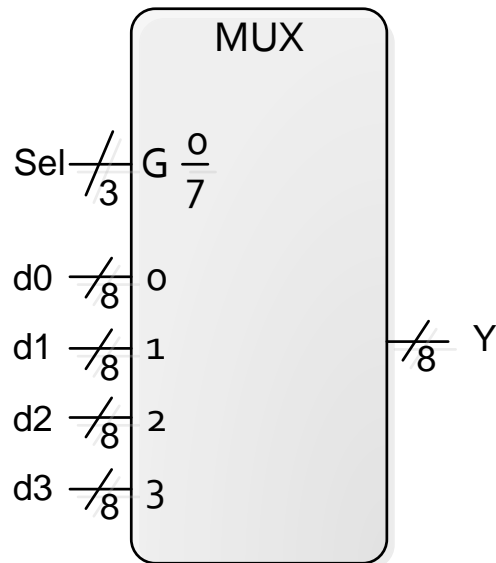
```
...
architecture behavior of meu_circuito is
  -- declaração do sinal de selecção
  signal sel : std_logic_vector(2 downto 0);
  ...
begin
  ...
  -- cálculo (concatenação) do sinal de selecção
  Sel <= S2 & S1 & S0;
  -- multiplexer 8:1... Ver código do slide anterior
  ...

end behavior;
```

## Multiplexer 4:1 de 8 bits

43

Sistemas Digitais, 2013



...

```
architecture behavior of meu_circuito is
```

```
-- declaração do sinal de selecção
```

```
signal sel : std_logic_vector(2 downto 0);
```

```
signal d0,d1,d2,d3,y: std_logic_vector(7 downto 0);
```

...

```
begin
```

...

```
-- Descrição de 8 multiplexers 4:1, 1 por cada bit.
```

```
-- Requer que o número de bits de y, d0, d1, d2 e
```

```
-- d3 seja a mesma (neste caso 8 bits)
```

```
y <= d0 when Sel="00" else
```

```
    d1 when Sel="01" else
```

```
    d2 when Sel="10" else
```

```
    d3;
```

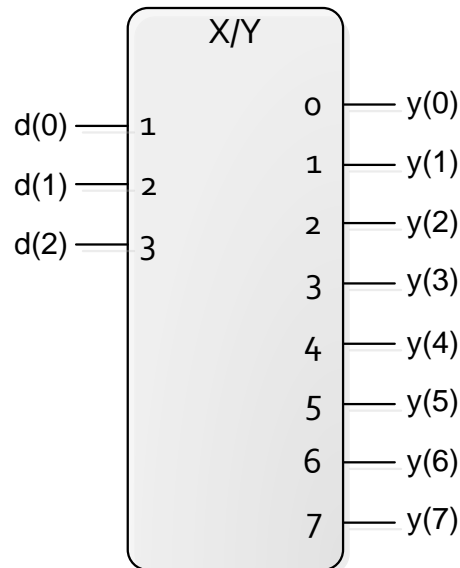
```
end behavior;
```

## Descri  o de circuitos em VHDL

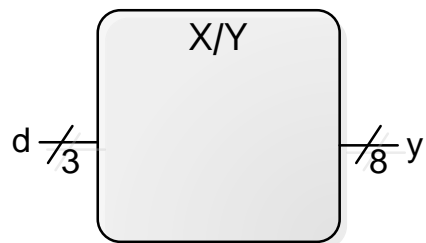
# Decodificador 3:8

44

Sistemas Digitais, 2013



**Equivalente a:**



```
architecture behavior of meu_circuito is
  -- declara  o do sinal de selec  o
  signal d : std_logic_vector(2 downto 0);
  signal y : std_logic_vector(7 downto 0);
  ...
begin
  ...
  -- Descri  o de 1 decodificador 3:8
  y <= "00000001" when d="000" else
        "00000010" when d="001" else
        "00000100" when d="010" else
        "00001000" when d="011" else
        "00010000" when d="100" else
        "00100000" when d="101" else
        "01000000" when d="110" else
        "10000000";

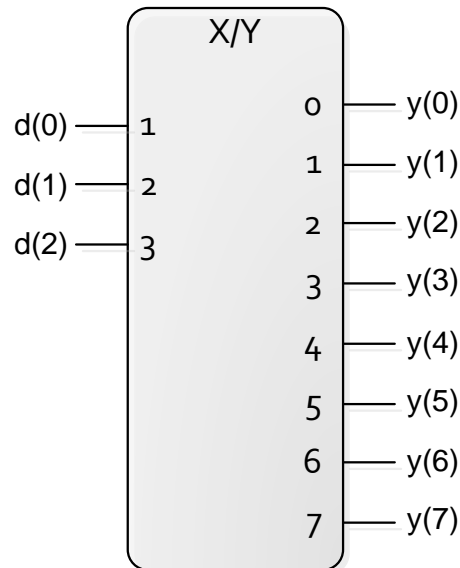
end behavior;
```

# Descrição de circuitos em VHDL

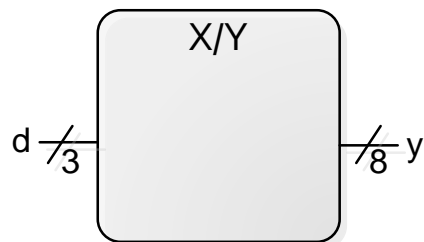
## Decodificador 3:8

45

Sistemas Digitais, 2013



Equivalente a:



### SOLUÇÃO ALTERNATIVA

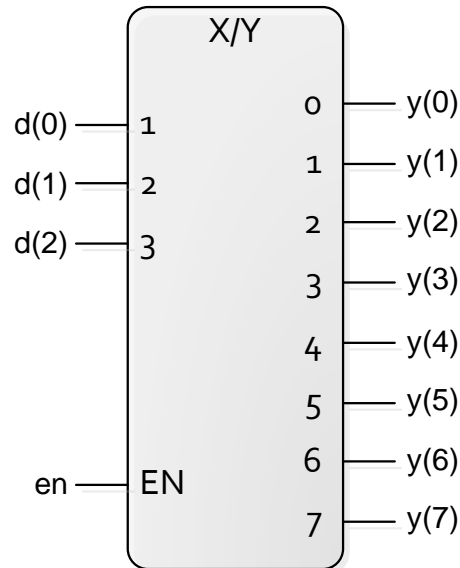
```
architecture behavior of meu_c
-- declaração do sinal de seleção
signal d : std_logic_vector (2 downto 0);
signal y : std_logic_vector (7 downto 0);
...
begin
...
-- Descrição de 1 decodificador 3:8
With d select
    y <= "00000001" when "000",
        "00000010" when "001",
        "00000100" when "010",
        "00001000" when "011",
        "00010000" when "100",
        "00100000" when "101",
        "01000000" when "110",
        "10000000" when others;

end behavior;
```

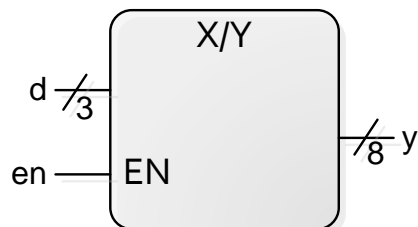
# Decodificador 3:8 com enable

46

Sistemas Digitais, 2013



**Equivalente a:**



```
architecture behavior of
-- declara  o do sinal de sele   o
signal d : std_logic_vector (2 downto 0);
signal y : std_logic_vector (7 downto 0);
...
begin
...
-- Descri  o de 1 decodificador 3:8
y <= "00000000" when en='0' else
      "00000001" when d="000" else
      "00000010" when d="001" else
      "00000100" when d="010" else
      "00001000" when d="011" else
      "00010000" when d="100" else
      "00100000" when d="101" else
      "01000000" when d="110" else
      "10000000";

end behavior;
```

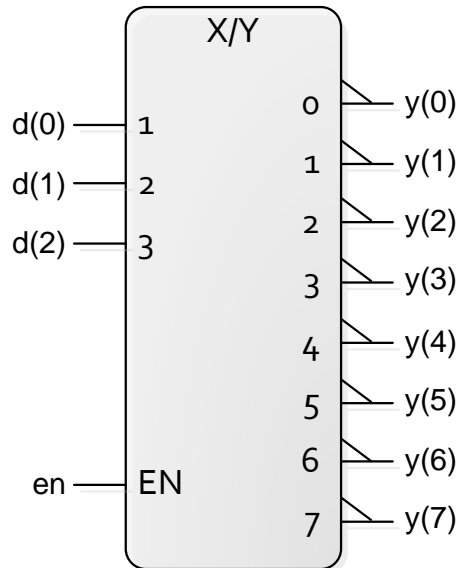
**DESCODIFICADOR COM ENABLE**

# Descrição de circuitos em VHDL

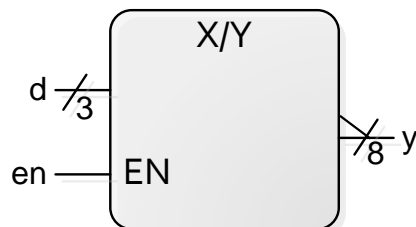
## Decodificador 3:8 (saídas negadas)

47

Sistemas Digitais, 2013



Equivalente a:



```
architecture behavior of
-- declaração do sinal de seleção
signal d : std_logic_vector (2 downto 0);
signal y : std_logic_vector (7 downto 0);
...
begin
...
-- Descrição de 1 decodificador 3:8
y <= "11111111" when en='0' else
    "11111110" when d="000" else
    "11111101" when d="001" else
    "11111011" when d="010" else
    "11110111" when d="011" else
    "11101111" when d="100" else
    "11011111" when d="101" else
    "10111111" when d="110" else
    "01111111";

end behavior;
```

DESCODIFICADOR COM ENABLE

## Circuito somador/subtractor

Descrição de uma unidade de cálculo básica capaz de realizar somas e subtrações

Utilização de uma abordagem *bottom-up*



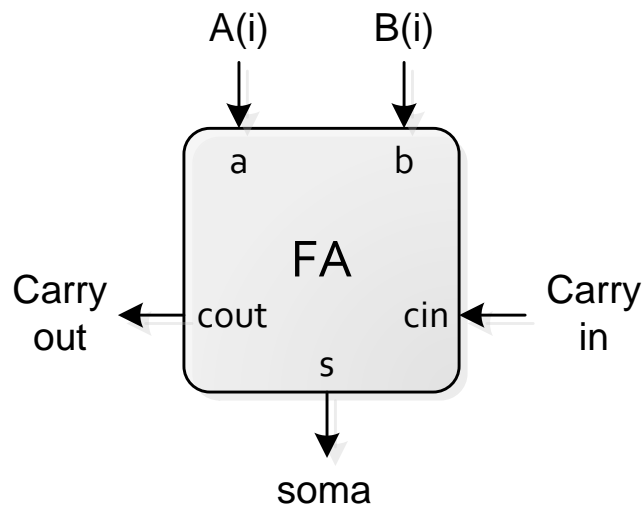
# Full-Adder

49

Sistemas Digitais, 2013

- O elemento básico de um circuito somador/subtrator é:

- ▣ *Full-Adder*

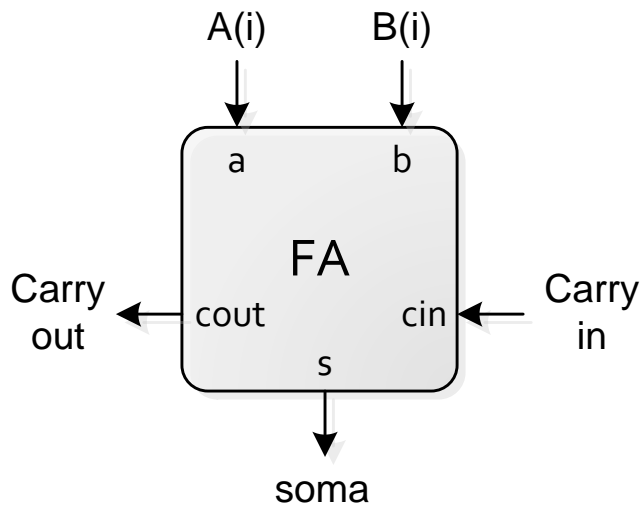


# Elementos básicos

50

Sistemas Digitais, 2013

## 1. Identificação do componente:



```
-- FICHEIRO full_adder.vhd

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

-- Definição do nome da entidade e
-- dos sinais (fios) de entrada/saída
entity full_adder is
    port (
        a      : in  std_logic;
        b      : in  std_logic;
        cin    : in  std_logic;
        s      : out std_logic;
        cout   : out std_logic
    );
end full_adder;

...
```

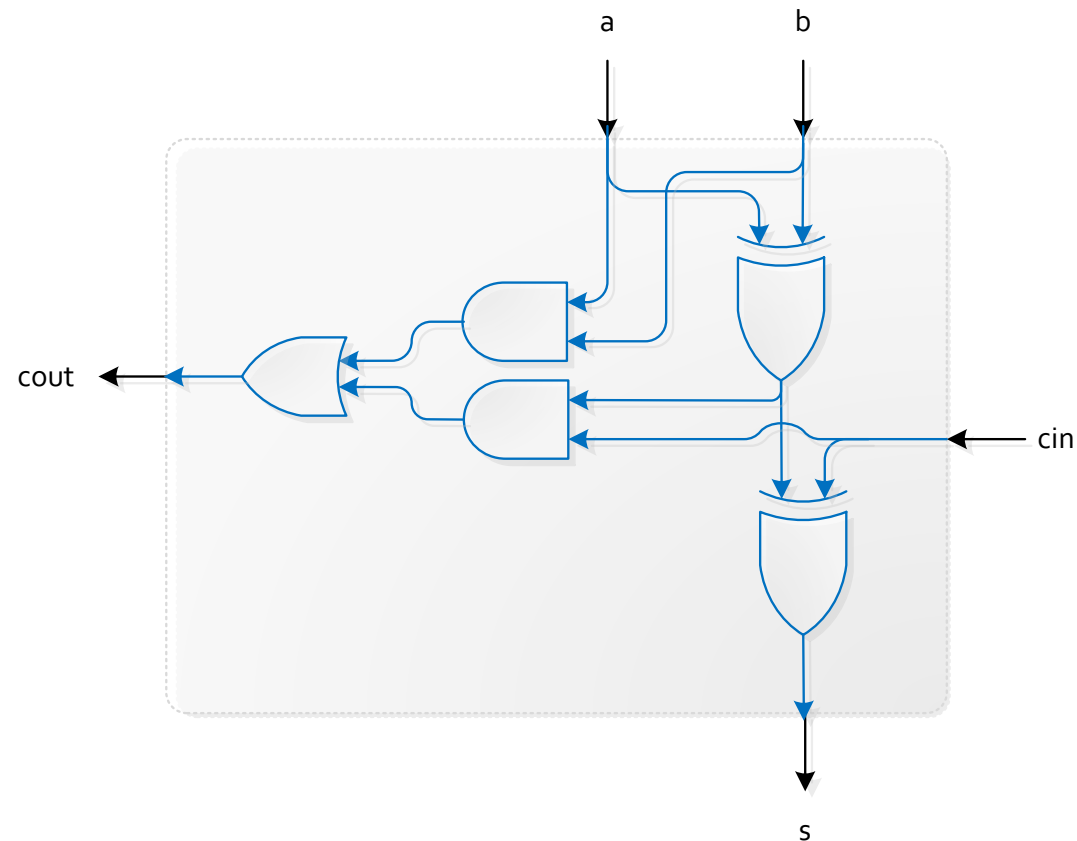
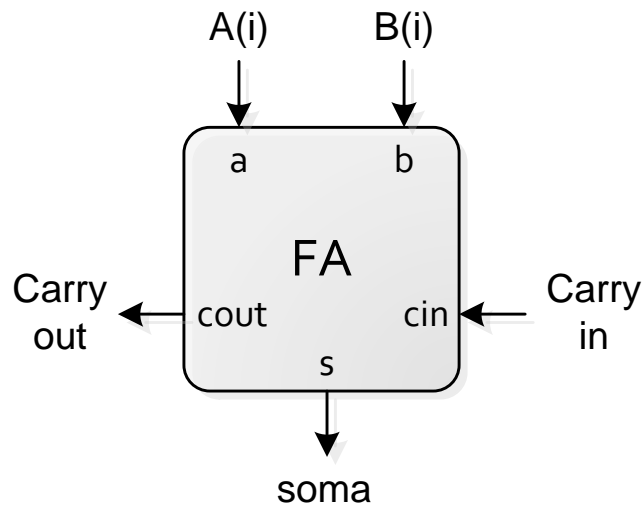
# Elementos básicos

51

Sistemas Digitais, 2013

## 2. Desenho do logigrama interno do *full-adder*:

Nota: o logigrama podia ser alterado de forma a usar só portas NAND e XOR



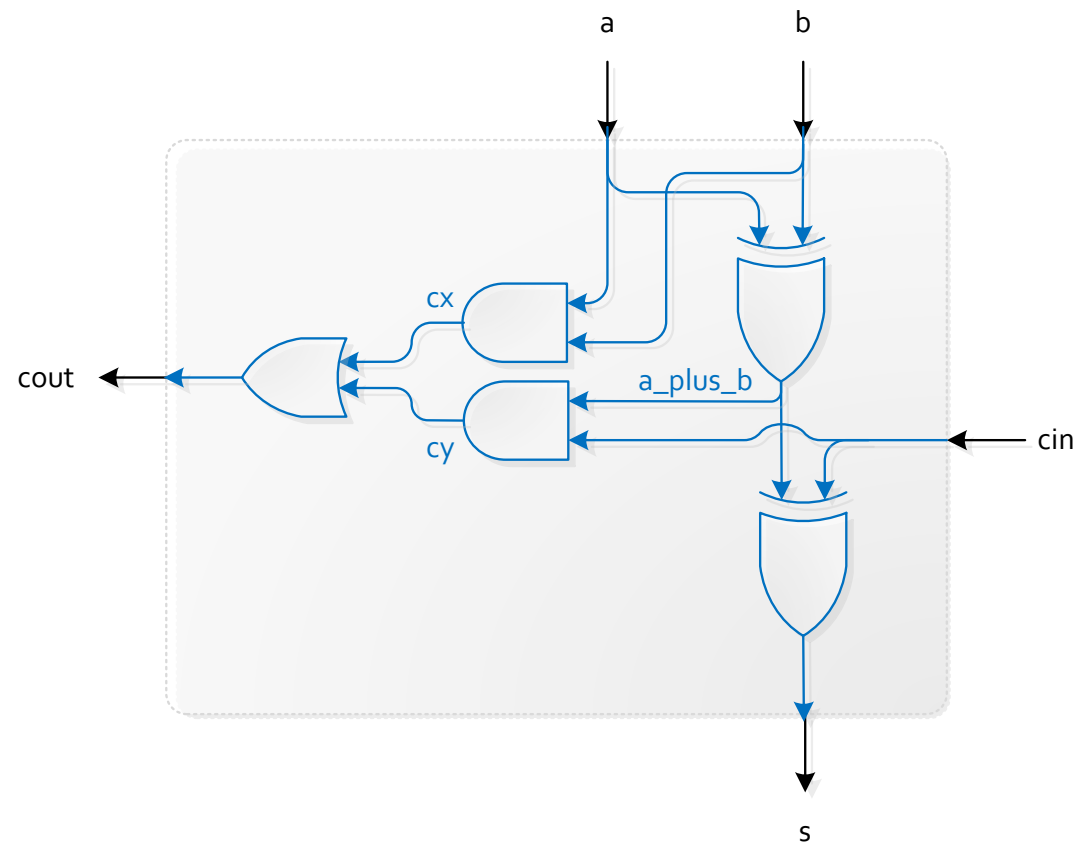
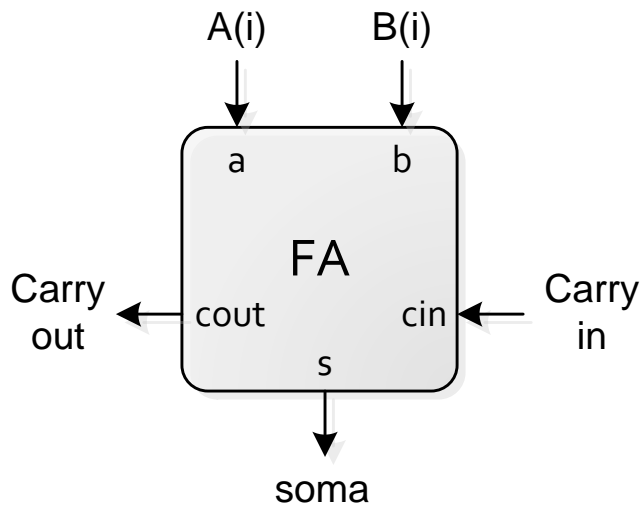
# Elementos básicos

52

Sistemas Digitais, 2013

## 2. Desenho do logigrama interno do *full-adder*:

a) *É necessário identificar no esquema o nome dos sinais (fios) internos*



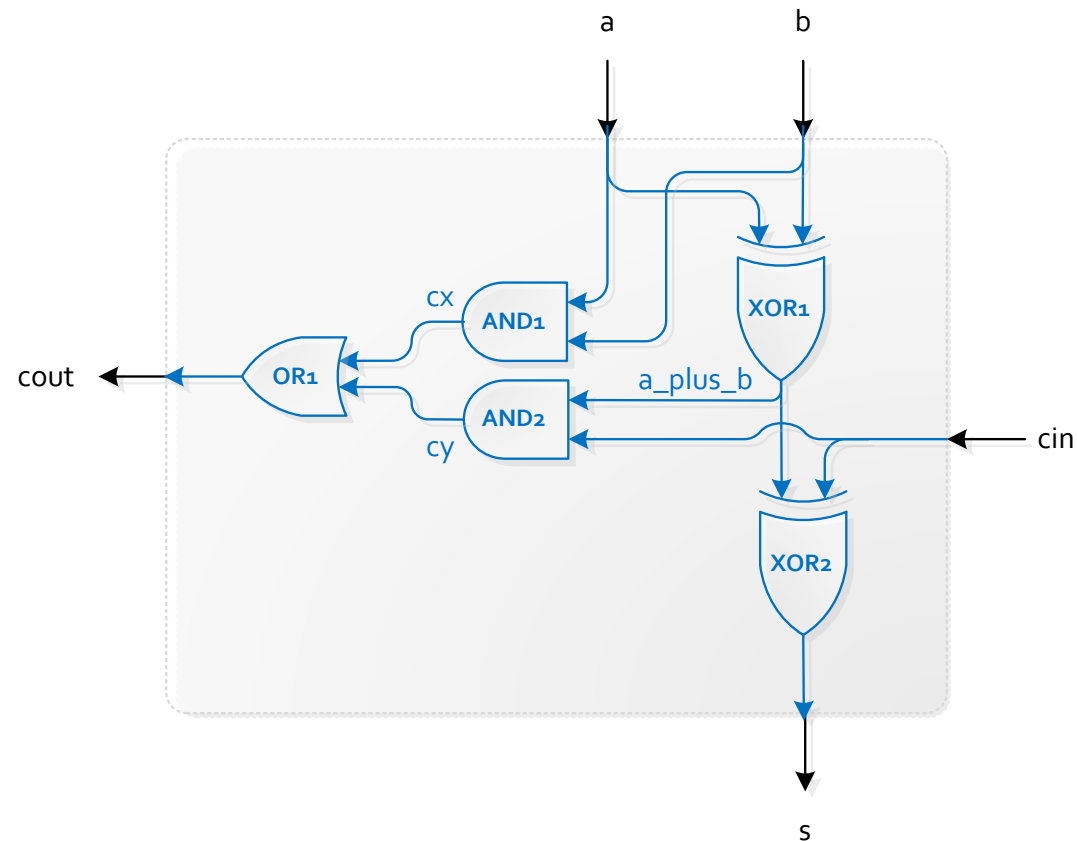
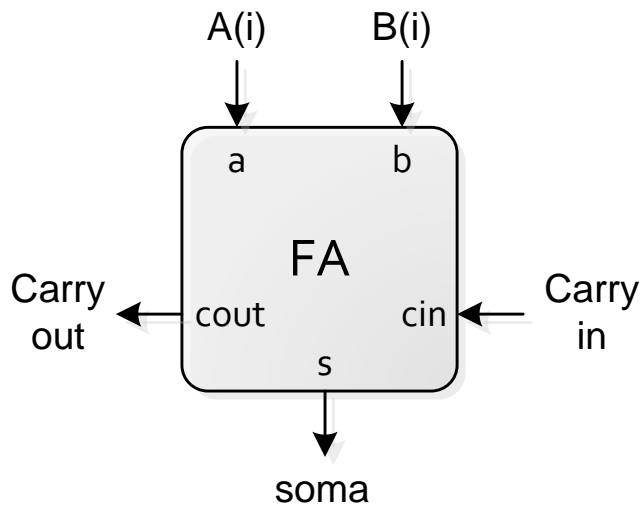
# Elementos básicos

53

Sistemas Digitais, 2013

## 2. Desenho do logigrama interno do *full-adder*:

b) *Vamos ainda identificar cada uma das portas lógicas*



### 3. Descrição da arquitectura do circuito `full_adder`

```
architecture behavior of full_adder is
```

```
-- declaração dos sinais internos
```

```
signal a_plus_b, cx, cy : std_logic;
```

```
begin
```

```
-- XOR1
```

```
a_plus_b <= a xor b;
```

```
-- XOR2
```

```
s <= cin xor a_plus_b;
```

```
-- AND1
```

```
cx <= a and b;
```

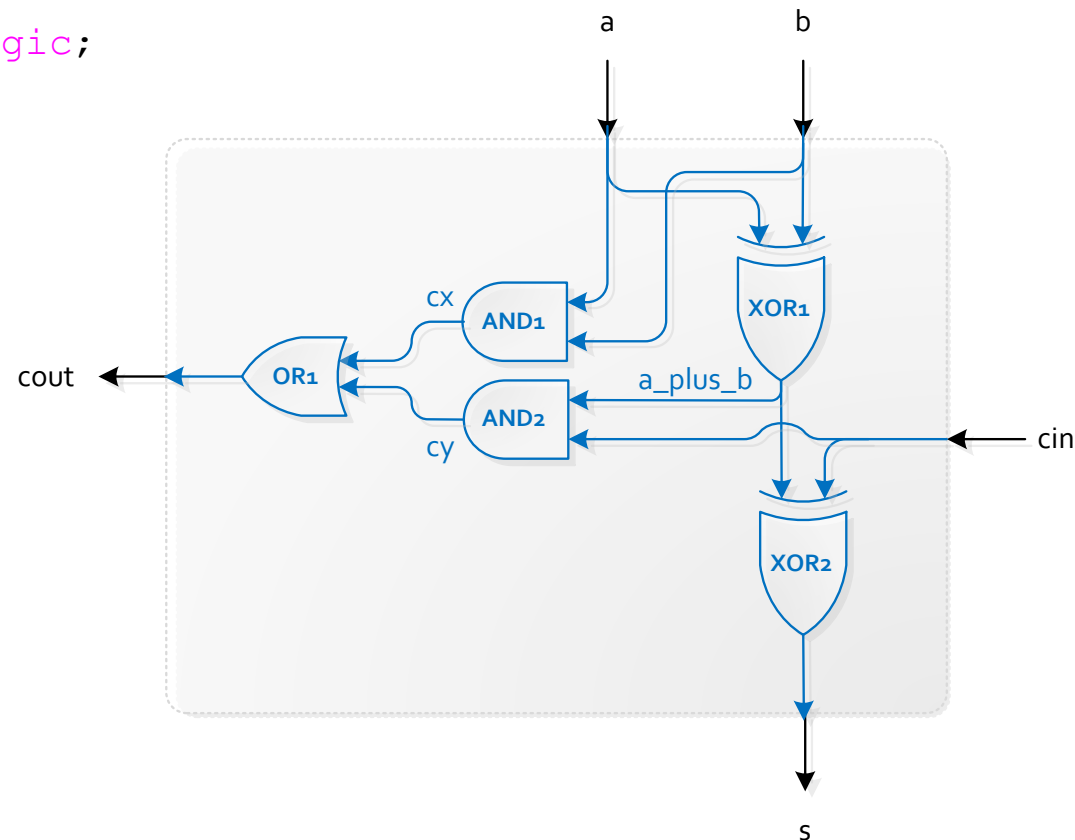
```
-- AND2
```

```
cy <= a_plus_b and cin;
```

```
-- OR1
```

```
cout <= cx or cy;
```

```
end behavior;
```



# Circuito somador/subtrator

## full\_adder.vhd

55

Sistemas Digitais, 2013

```
-- FICHEIRO full_adder.vhd

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

-- Definição do nome da entidade e
-- dos sinais (fios) de entrada/saída
entity full_adder is
    port (
        a      : in  std_logic;
        b      : in  std_logic;
        cin    : in  std_logic;
        s      : out std_logic;
        cout   : out std_logic
    );
end full_adder;
```

```
architecture behavior of full_adder is
-- declaração dos sinais internos
signal a_plus_b, cx, cy : std_logic;

begin
    -- XOR1
    a_plus_b <= a xor b;
    -- XOR2
    s <= cin xor a_plus_b;
    -- AND1
    cx <= a and b;
    -- AND2
    cy <= a_plus_b and cin;
    -- OR1
    cout <= cx or cy;

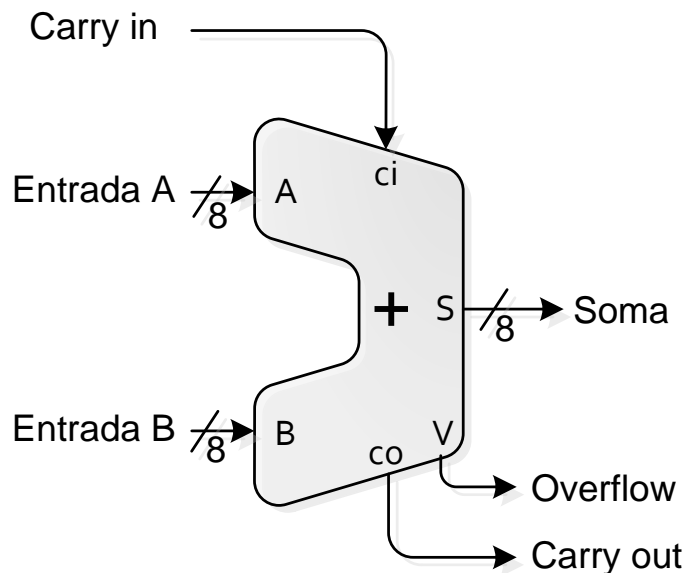
end behavior;
```

## Somador de 8-bits com carry in

56

Sistemas Digitais, 2013

- Usando o circuito full-adder (`full_adder.vhd`)   poss vel construir um somador de n-bits com carry in



```
-- FICHEIRO somador8.vhd

-- Declara  o de bibliotecas
library IEEE;
use IEEE.std_logic_1164.all;

-- Defini  o do nome da entidade e
-- dos sinais (fios) de entrada/sa  da
entity somador8 is
    port (
        A  : in  std_logic_vector (7 downto 0);
        B  : in  std_logic_vector (7 downto 0);
        ci : in  std_logic;
        S  : out std_logic_vector (7 downto 0);
        co : out std_logic;
        V  : out std_logic
    );
end somador8;

...
```



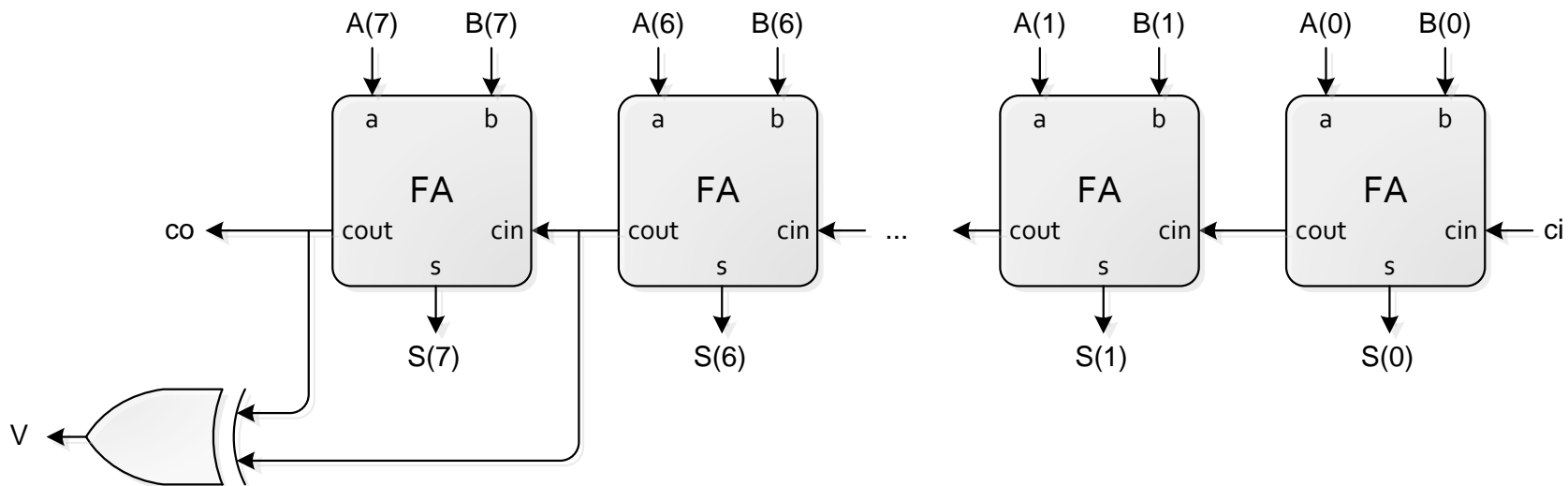
## Circuito somador/subtrator

# Somador de 8-bits com *carry in*

57

Sistemas Digitais, 2013

### □ Logigrama do circuito somador de 8-bits



## Circuito somador/subtractor

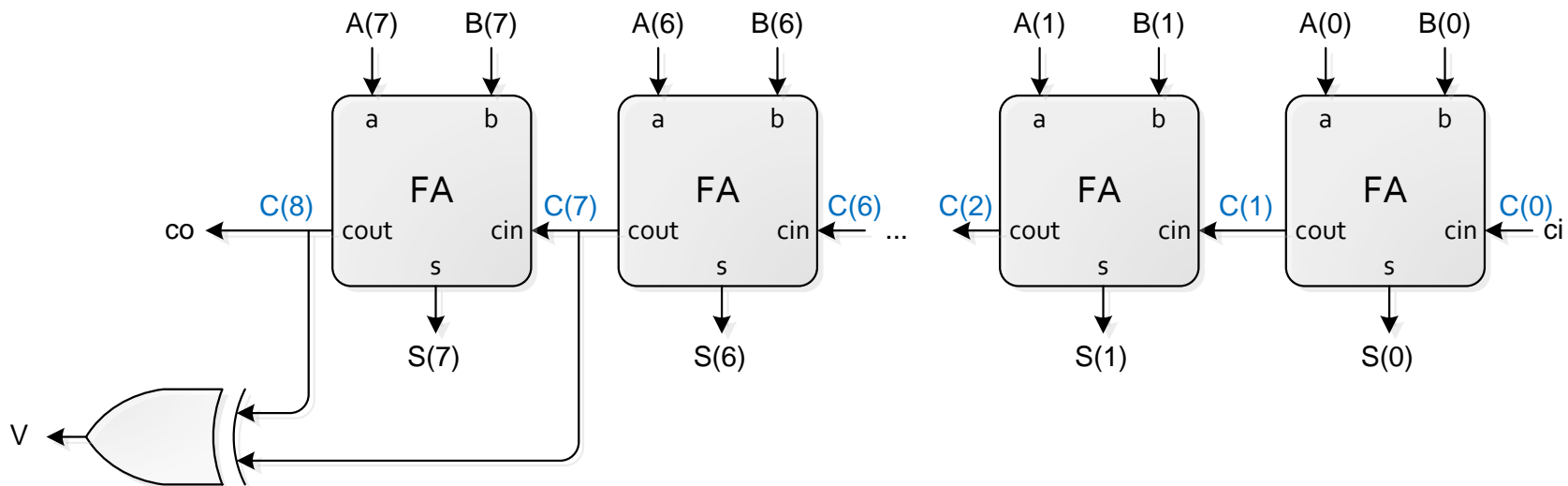
# Somador de 8-bits com carry in

58

Sistemas Digitais, 2013

### Logigrama do circuito somador de 8-bits

#### Sinais internos (azul)



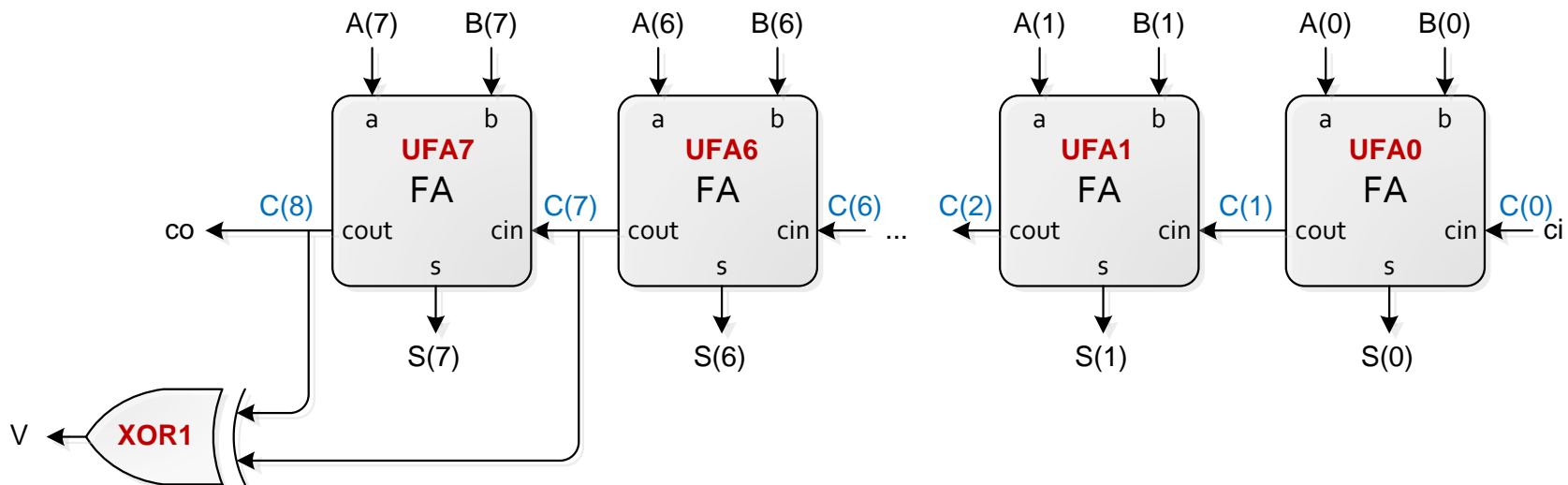
## Circuito somador/subtractor

# Somador de 8-bits com carry in

59

Sistemas Digitais, 2013

- Logigrama do circuito somador de 8-bits
  - ▣ *Sinais internos (azul)*
  - ▣ *Nome das instancias (vermelho)*



Nota: os sinais A, B e S são barramentos de 8 bits, enquanto o sinal C (a azul) é um barramento de 9 bits, C(8), C(7), ..., C(0)

## Somador de 8-bits com carry in

60

Sistemas Digitais, 2013

### □ Logigrama do circuito somador de 8-bits

#### ▣ Declaração do componente *full\_adder*

Para utilizar um componente descrito num ficheiro .vhd é necessário declarar a sua existência. A declaração de um componente é a seguinte:

```
-- Declaração da entidade, colocada entre
-- "architecture" e "begin"
component <NOME_DO_COMPONENTE>
    port (
        <SINAL1> : <IN/OUT> <TIPO_DE_SINAL>;
        ...
        <SINALN> : <IN/OUT> <TIPO_DE_SINAL>
    );
end component;
```

```
-- Definição da entidade, colocada
-- no topo do ficheiro .vhd
entity NOME_DO_COMPONENTE is
    port (
        <SINAL1> : <IN/OUT> <TIPO_DE_SINAL>;
        ...
        <SINALN> : <IN/OUT> <TIPO_DE_SINAL>
    );
end NOME_DO_COMPONENTE;
```

- A declaração de um componente é uma cópia quase perfeita da descrição da entidade. As únicas diferenças residem na primeira e ultima linhas

## Circuito somador/subtrator

# Somador de 8-bits com carry in

61

Sistemas Digitais, 2013

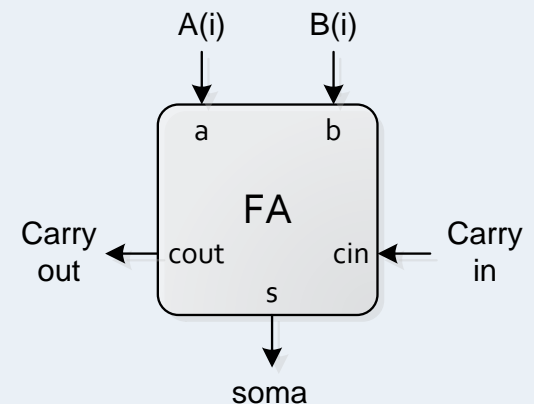
## □ Logigrama do circuito somador de 8-bits

### ▣ Declaração do componente *full\_adder*

```
architecture behavior of somador8 is
-- declaração do componente full_adder
component full_adder
    port (
        a      : in  std_logic;
        b      : in  std_logic;
        cin    : in  std_logic;
        s      : out std_logic;
        cout   : out std_logic
    );
end component;
-- declaração dos sinais internos
signal C : std_logic_vector(8 downto 0);

begin
...
end;
```

Componente *full\_adder*  
descrito no ficheiro  
*full\_adder.vhd*



Nota: a declaração de um componente é uma cópia quase perfeita da descrição da entidade. As únicas diferenças residem em:  
1) A primeira linha é dada por “”

## Somador de 8-bits com carry in

62

Sistemas Digitais, 2013

### □ Arquitectura do circuito somador de 8-bits

```

architecture behavior of somador8 is
-- declaração do componente full_adder
...
-- declaração dos sinais internos
signal C : std_logic_vector(8 downto 0);

begin
-- Utilização de 8 instancias do componente full_adder
UFA0: full_adder port map( a=>A(0) , b=>B(0) , cin=>C(0) , s=>S(0) , cout=>C(1) );
UFA1: full_adder port map( a=>A(1) , b=>B(1) , cin=>C(1) , s=>S(1) , cout=>C(2) );
UFA2: full_adder port map( a=>A(2) , b=>B(2) , cin=>C(2) , s=>S(2) , cout=>C(3) );
...
UFA7: full_adder port map( a=>A(7) , b=>B(7) , cin=>C(7) , s=>S(7) , cout=>C(8) );

-- sinais adicionais de saída
co <= C(8);
V <= C(7) xor C(8);
end behavior;

```

## Somador de 8-bits com carry in

63

Sistemas Digitais, 2013

### □ Arquitectura do circuito somador de 8 bits

SOLUÇÃO ALTERNATIVA

```
architecture behavior of somador8 is
-- declaração do componente full_adder
...
-- declaração dos sinais internos
signal C : std_logic_vector(8 downto 0);
begin
-- atribuição do valor de C(0)
C(0) <= ci;
-- Instanciação automática de 8 componentes (0 ... 7)
uGen1 : for i in 0 to 7 generate
    UFA0: full_adder port map(
        a=>A(i) , b=>B(i) , cin=>C(i) ,
        s=>S(i) , cout=>C(i+1)
    );
end generate;
-- sinais adicionais de saída
co <= C(8);
V <= C(7) xor C(8);
end behavior;
```

# Circuito somador/subtractor

## somador8.vhd

64

Sistemas Digitais, 2013

```
-- FICHEIRO somador8.vhd
-- Declaração de bibliotecas
library IEEE;
use IEEE.std_logic_1164.all;
-- Definição da entidade/sinais de entrada/saída
entity somador8 is
    port (
        A  : in  std_logic_vector(7 downto 0);
        B  : in  std_logic_vector(7 downto 0);
        ci : in  std_logic;
        S  : out std_logic_vector(7 downto 0);
        co : out std_logic;
        V  : out std_logic
    );
end somador8;

architecture behavior of somador8 is
-- declaração do componente full_adder
component full_adder
    port (
        a      : in  std_logic;
        b      : in  std_logic;
        cin    : in  std_logic;
        s      : out std_logic;
        cout   : out std_logic
    );
end component;
```

```
-- declaração dos sinais internos
-- C(i) corresponde ao carry-out do full-adder
-- i-1, e serve como carry-in do
-- full-adder i
signal C : std_logic_vector(8 downto 0);

begin

-- atribuição do valor de C(0)
C(0) <= ci;

-- simplificação (opcional) da utilização das
-- várias instancias
uGen1 : for i in 0 to 7 generate
    UFA0: full_adder port map(
        a=>A(i) , b=>B(i) , cin=>C(i) ,
        s=>S(i) , cout=>C(i+1)
    );
end generate;

-- sinais adicionais de saída
co <= C(8);
V <= C(7) xor C(8);

end behavior;
```

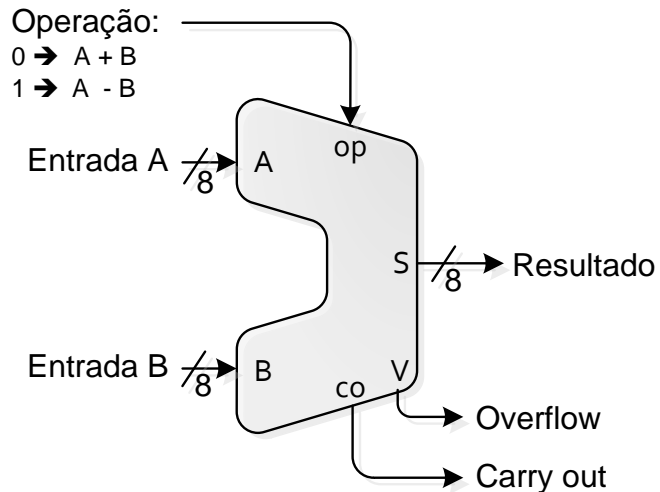


## Circuito final

65

Sistemas Digitais, 2013

- Usando o somador de 8 bits (`somador8.vhd`)   poss vel construir o circuito somador/subtrator



```
-- FICHEIRO arithmetic_unit.vhd

-- Declara  o de bibliotecas
library IEEE;
use IEEE.std_logic_1164.all;

-- Defini  o do nome da entidade e
-- dos sinais (fios) de entrada/s  da
entity arithmetic_unit is
    port (
        A  : in  std_logic_vector (7 downto 0);
        B  : in  std_logic_vector (7 downto 0);
        Op : in  std_logic;
        S  : out std_logic_vector (7 downto 0);
        co : out std_logic;
        V  : out std_logic
    );
end arithmetic_unit;

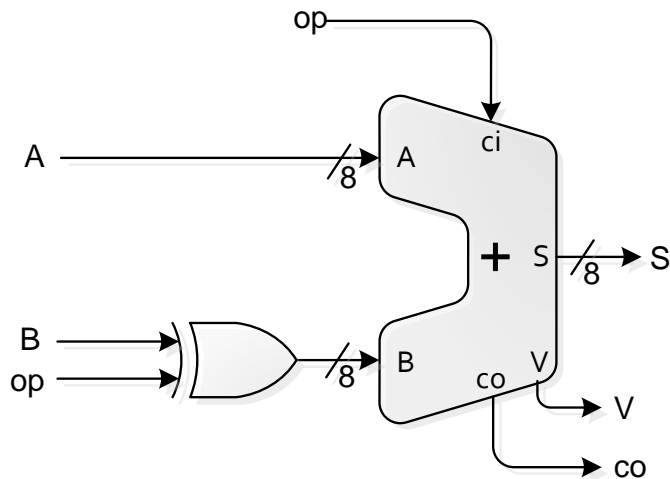
...
```

# Circuito final

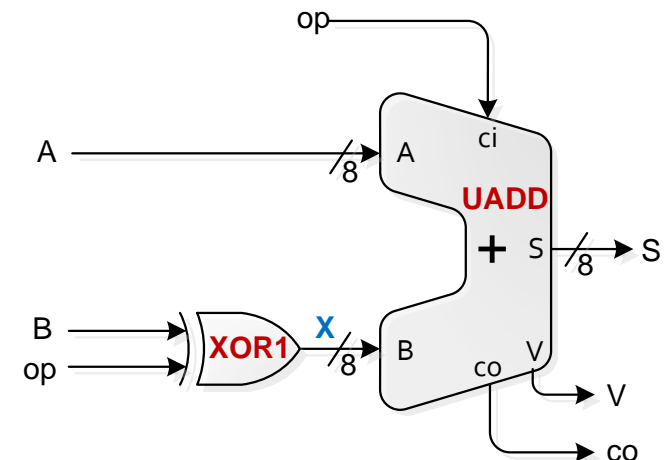
66

Sistemas Digitais, 2013

- Logigrama do circuito somador/subtractor



## Após nomeação dos sinais e circuitos internos



## Circuito final

67

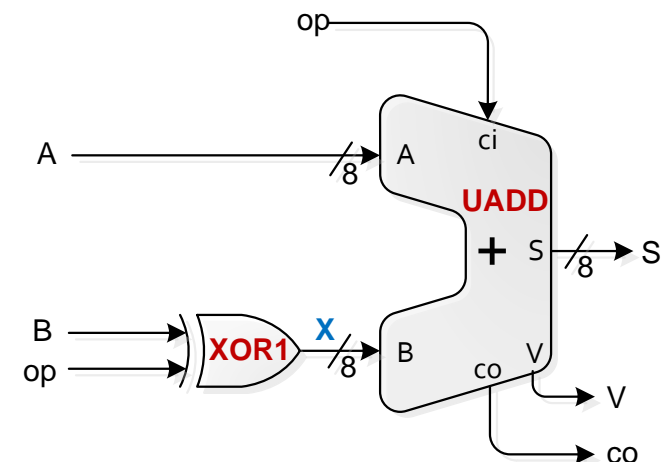
Sistemas Digitais, 2013

### □ Arquitectura (VHDL) do circuito

#### ▣ Declaração do componente somador8

```
architecture behavior of arithmetic_unit is
-- declaração do componente somador8
component somador8
  port (
    A  : in  std_logic_vector(7 downto 0);
    B  : in  std_logic_vector(7 downto 0);
    ci : in  std_logic;
    S  : out std_logic_vector(7 downto 0);
    co : out std_logic;
    V  : out std_logic
  );
end component;
-- declaração dos sinais internos
signal X : std_logic_vector(7 downto 0);

begin
```



## Circuito final

68

Sistemas Digitais, 2013

### □ Arquitectura (VHDL) do circuito

```
architecture behavior of arithmetic_unit is
-- declaração do componente somador8
...
-- declaração dos sinais internos
signal X : std_logic_vector(7 downto 0);
```

**begin**

```
-- 8 portas XOR, uma por cada bit, onde cada porta i
-- tem como entradas os bits B(i) e op
X <= B xor (7 downto 0 => op);
-- utilização de uma instancia do componente somador8
```

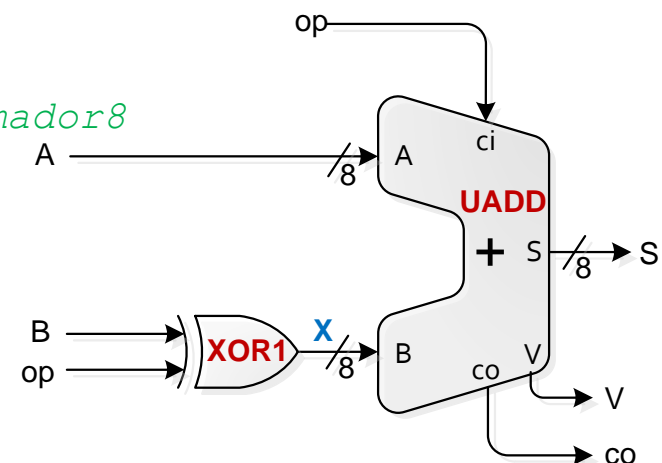
```
UADD: somador8 port map(
    A => A, B => X, ci => op,
    S => S, V => V, co => co
);
```

**end** behavior;

Nota: a declaração:

(n-1 downto 0 => X)

cria um sinal de n bits, onde todos os bits têm o mesmo valor que o sinal X.



# Circuito somador/subtrator

## arithmetic\_unit.vhd

69

Sistemas Digitais, 2013

```
-- FICHEIRO arithmetic_unit.vhd

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

-- Definição do nome da entidade e
-- dos sinais (fios) de entrada/saída
entity arithmetic_unit is
    port (
        A : in  std_logic_vector(7 downto 0);
        B : in  std_logic_vector(7 downto 0);
        Op : in  std_logic;
        S : out std_logic_vector(7 downto 0);
        co : out std_logic;
        V : out std_logic
    );
end arithmetic_unit;
```

```
architecture behavior of arithmetic_unit is
    -- declaração do componente somador8
    component somador8
        port (
            A : in  std_logic_vector(7 downto 0);
            B : in  std_logic_vector(7 downto 0);
            ci : in  std_logic;
            S : out std_logic_vector(7 downto 0);
            co : out std_logic;
            V : out std_logic
        );
    end component;
    -- declaração dos sinais internos
    signal X : std_logic_vector(7 downto 0);

    begin
        -- 8 portas XOR, uma por cada bit
        X <= B xor (7 downto 0 => op);
        -- instancia do componente somador8
        UADD: somador8 port map(
            A => A, B => X, ci => op,
            S => S, V => V, co => co
        );
    end behavior;
```

## Simulação de circuitos

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

# Simulação e teste de circuitos

71

Sistemas Digitais, 2013

- 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

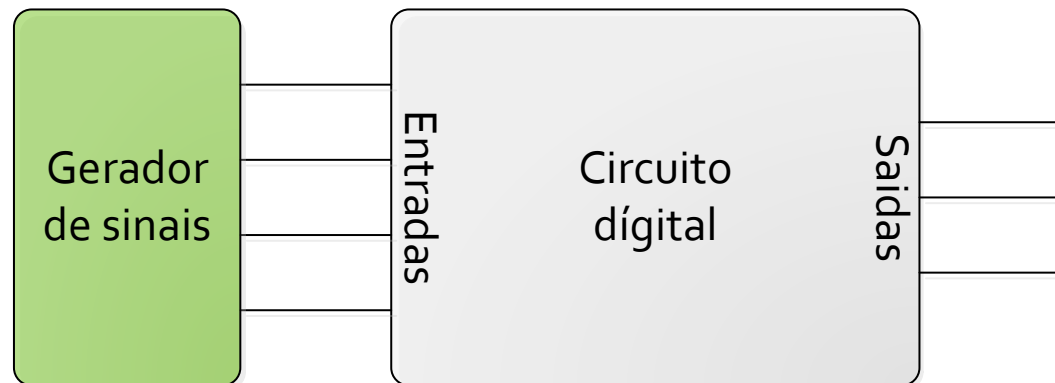
Nota: o estado do sistema digital é dado pelo valor à saída dos elementos de memória, i.e., dos latches e dos flip-flops.

# Simulação e teste de circuitos

72

Sistemas Digitais, 2013

- 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

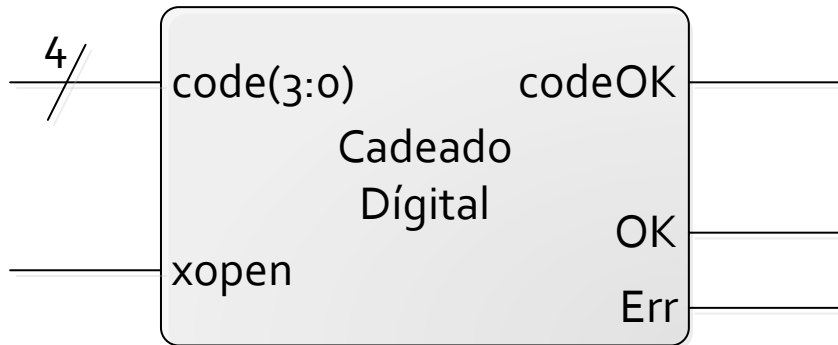


# Simulação e teste do circuito “Cadeado Digital”

## Tabela de verdade

73

Sistemas Digitais, 2013



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:

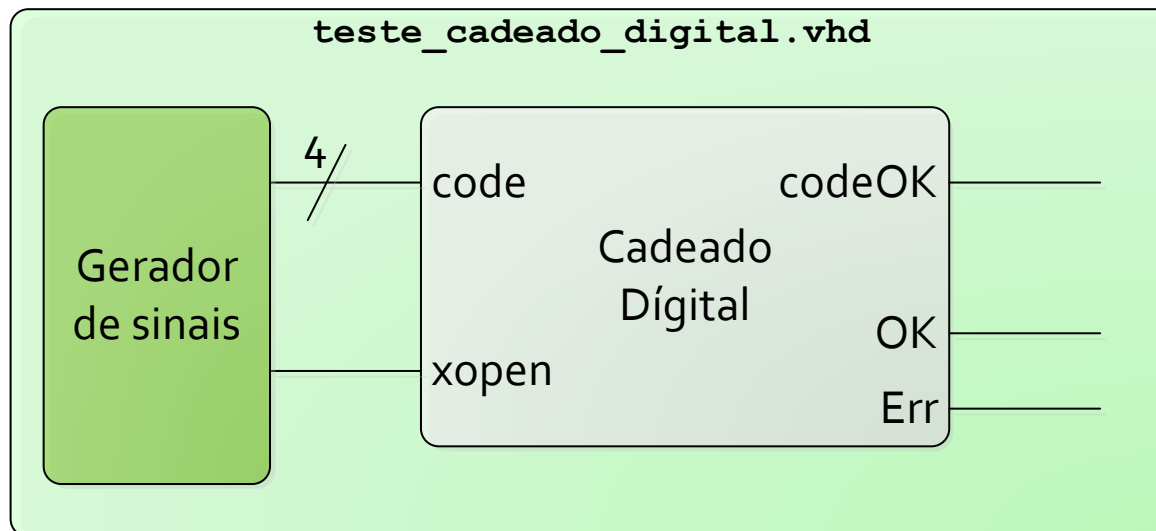
ENTRADAS		SAIDAS ESPERADAS		
code	xopen	codeOK	OK	Err
0000	0	0	0	0
0001	0	0	0	0
...	...	...	...	...
0110	0	0	0	0
0111	0	1	0	0
1000	0	0	0	0
...	...	...	...	...
0110	1	0	0	1
0111	1	1	1	0
1000	1	0	0	1
...	...	...	...	...

# Estrutura do ficheiro de simulação

74

Sistemas Digitais, 2013

- 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

75

Sistemas Digitais, 2013

## □ 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 saída
entity cadeado_digital_testbench is
end cadeado_digital_testbench;

architecture behavior of cadeado_digital_testbench is
...

```

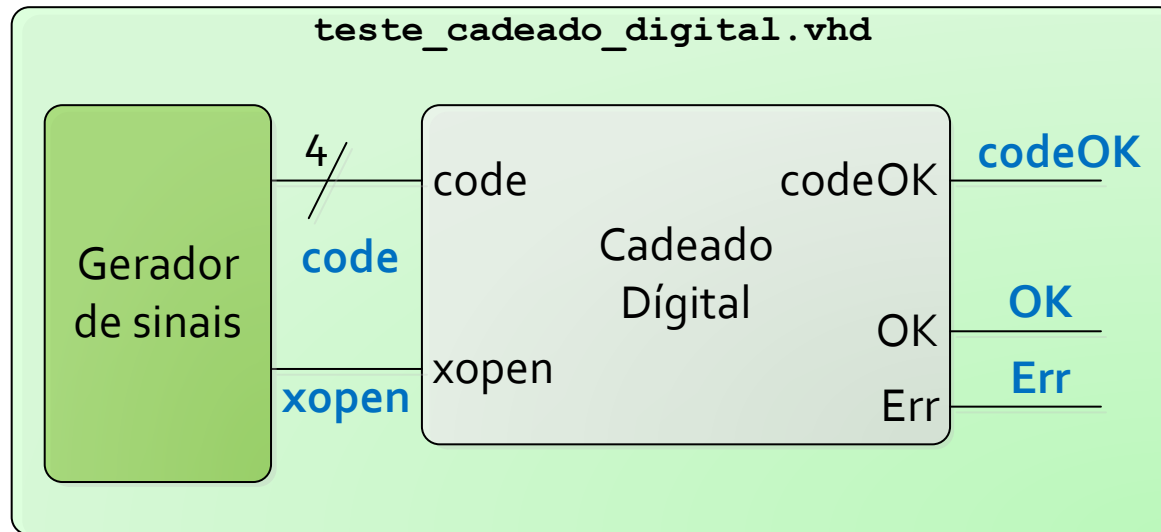
# Simulação e teste do circuito “Cadeado Digital”

## Descrição da arquitectura

76

Sistemas Digitais, 2013

- Descrição da arquitectura
  - ▣ Declaração de componentes e sinais



# Simula  o e teste do circuito “Cadeado Digital”

## Descri   o da arquitectura

77

Sistemas Digitais, 2013

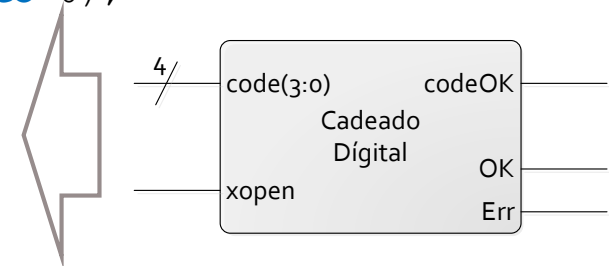
- Descri   o da arquitectura
  - ▣ Declara   o de componentes e sinais

```

...
architecture behavior of cadeado_digital_testbench is
-- Declara   o 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;
-- Declara   o dos sinais para o testbench
signal code : std_logic_vector(3 downto 0);
signal xopen, codeOK, OK, Err : std_logic;

begin

```



# Simulação e teste do circuito “Cadeado Digital”

## Descrição da arquitectura

78

Sistemas Digitais, 2013

### □ 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
  -- casa (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

79

Sistemas Digitais, 2013

## □ 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;
```

# Simulação e teste do circuito “Cadeado Digital”

## Descrição da arquitectura

80

Sistemas Digitais, 2013

### □ 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;
```





# Simulação e teste do circuito “Cadeado Digital”

## Descrição da arquitectura

81

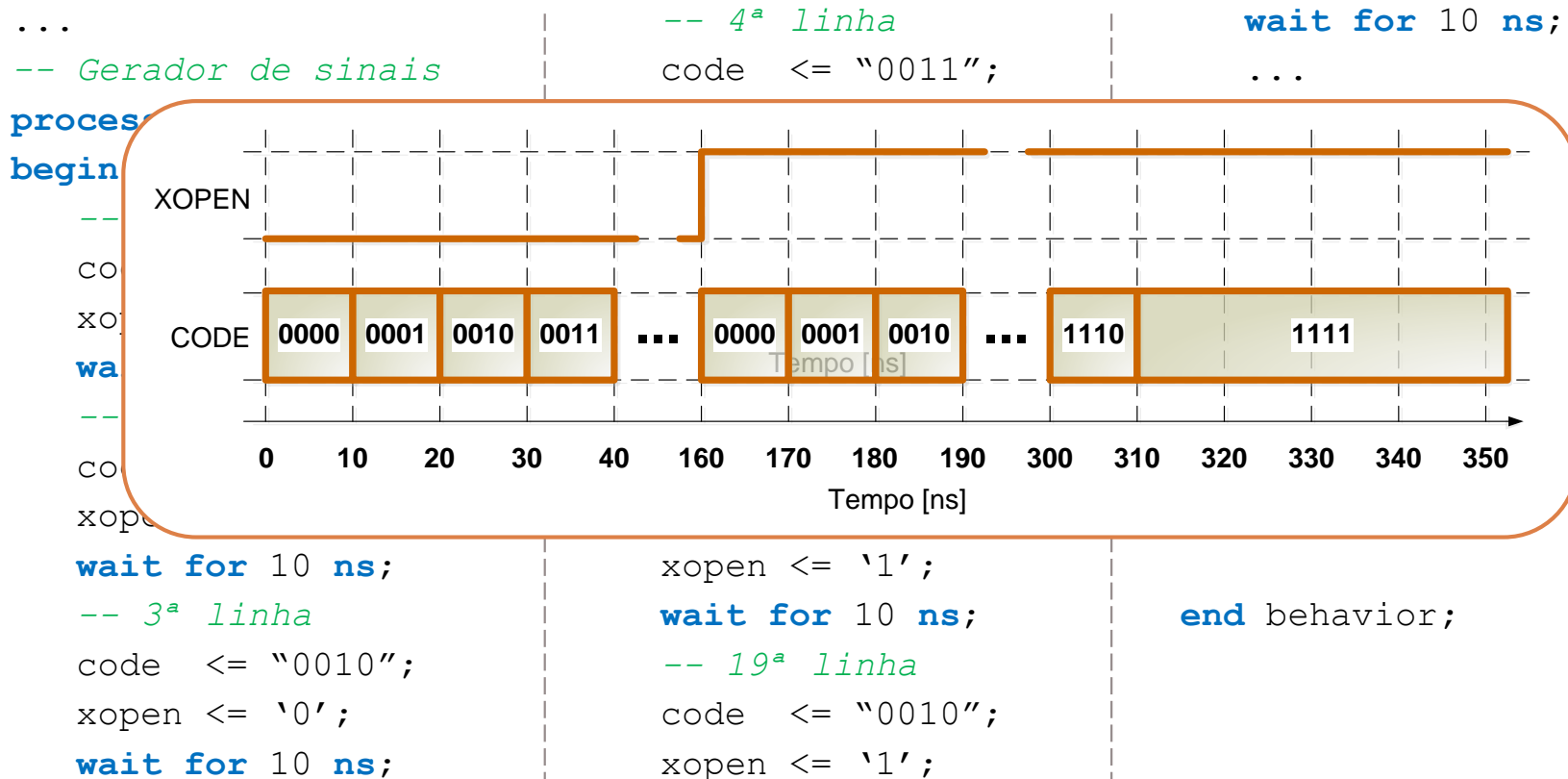
Sistemas Digitais, 2013

### 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



# Simulação e teste do circuito “Cadeado Digital”

## Descrição da arquitectura

82

Sistemas Digitais, 2013

### □ Implementação

#### ▣ Descrição do gerador de sinais

- Simulação de 1 linha da tabela de verdade a cada 10 ns

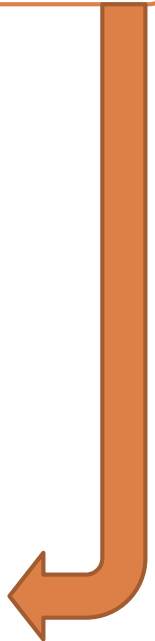
**O gerador de sinais repete após a ultima linha da tabela de verdade**

```
...
-- 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;
```



# Simulação e teste do circuito “Cadeado Digital”

## Descrição da arquitectura

83

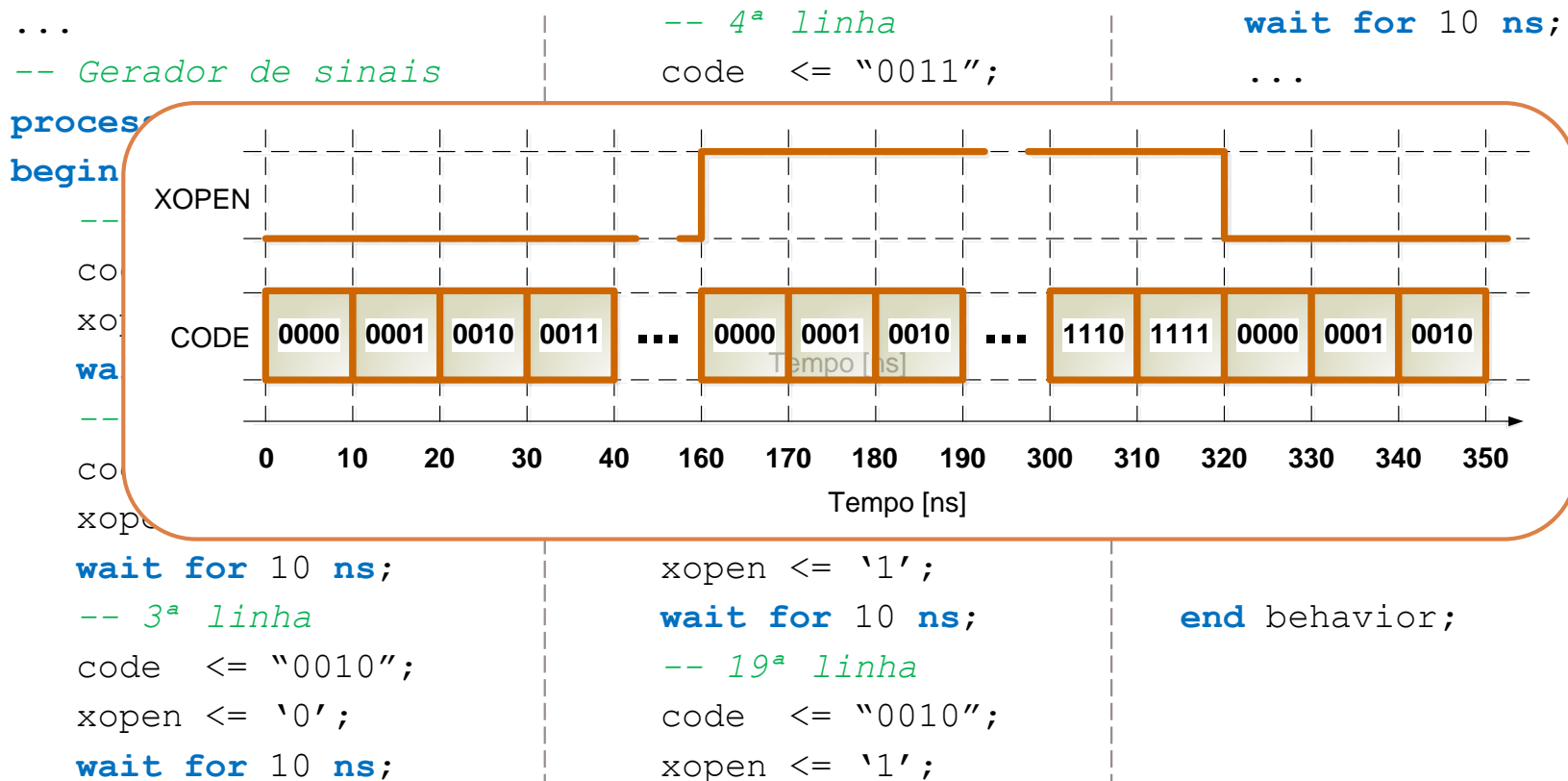
Sistemas Digitais, 2013

### □ 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



## Descrição da arquitectura

84

Sistemas Digitais, 2013

SOLUÇÃO ALTERNATIVA

### □ Implementação

#### ▣ 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;**

# Simula  o e teste do circuito “Cadeado Digital”

## Descri   o da arquitectura

85

Sistemas Digitais, 2013

SOLU   O ALTERNATIVA

###   Implementa   o

-   Descri   o do gerador de sinais
  -   Utiliza   o de macros (contador e inversor)

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

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

end behavior;
```

Requer:

1. a inicializa   o dos sinais
2. A utiliza   o da biblioteca `ieee.std_logic_unsigned`

# Simulação e teste do circuito “Cadeado Digital”

## cadeado\_digital\_testbench\_C.vhd

86

Sistemas Digitais, 2013

```
-- 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;
```

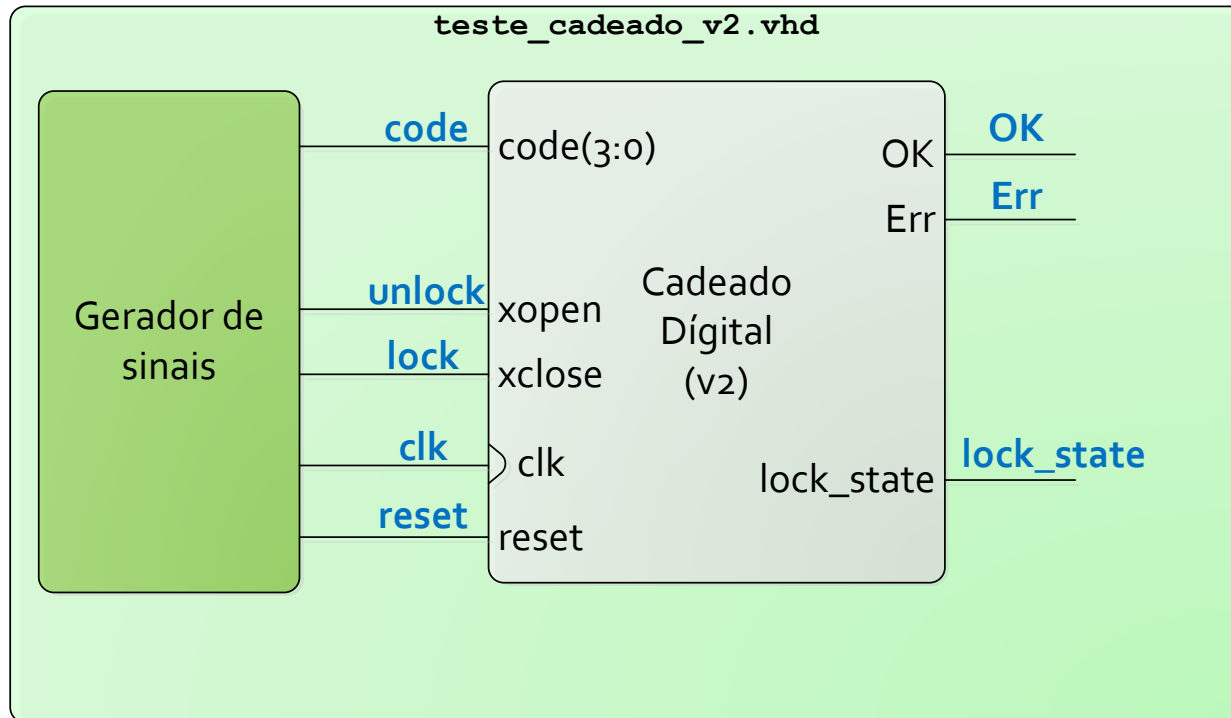
# Simulação e teste do circuito “Cadeado Digital (v2)”

## Descrição da arquitectura

87

Sistemas Digitais, 2013

### Componente para teste:



## Descrição da entidade

88

Sistemas Digitais, 2013

### □ Descrição da entidade

#### ▣ Sem entradas/saídas

```
-- FICHEIRO teste_cadeado_v2.vhd

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

-- Definição do nome da entidade, sem qualquer entrada ou saída
entity teste_cadeado_v2 is
end teste_cadeado_v2;

architecture behavior of teste_cadeado_v2 is
...
```



## Simulação e teste do circuito “Cadeado Digital (v2)”

# Declaração de componentes e sinais

89

Sistemas Digitais, 2013

```
...
architecture behavior of teste_cadeado_v2 is
  -- Declaração do componente cadeado_digital (v2)
component cadeado_digital_v2
  port (
    code      : in  std_logic_vector(3 downto 0);
    xopen     : in  std_logic;
    xclose    : in  std_logic;
    clk       : in  std_logic;
    reset     : in  std_logic;
    OK        : out std_logic;
    Err       : out std_logic;
    lock_state : out std_logic
  );
end component;
  -- Declaração dos sinais para o testbench
signal lock, unlock, clk, reset : std_logic := '0'; -- inicializados a 0
signal code : std_logic_vector(3 downto 0) := "1111"; -- inicializado a "1111"
signal OK, Err, lock_state: std_logic; -- sem inicialização (i.e., sem valor
                                         -- previamente definido para t=0s )

begin
```

## Simulação e teste do circuito “Cadeado Digital (v2)”

# Descrição da unidade para teste

90

Sistemas Digitais, 2013

```
...  
architecture behavior of teste_cadeado_v2 is  
-- Declaração do componente cadeado_digital (v2)  
...  
-- Declaração dos sinais para o testbench  
...  
begin  
-- declaração da instancia para teste  
test_unit: cadeado_digital_v2 port map (  
    code    => code,  
    xopen   => unlock,  
    xclose  => lock,  
    clk     => clk,  
    reset   => reset,  
    OK      => OK,  
    Err     => Err,  
    lock_state => lock_state  
);  
...  
end behavior;
```

# Simulação e teste do circuito “Cadeado Digital (v2)”

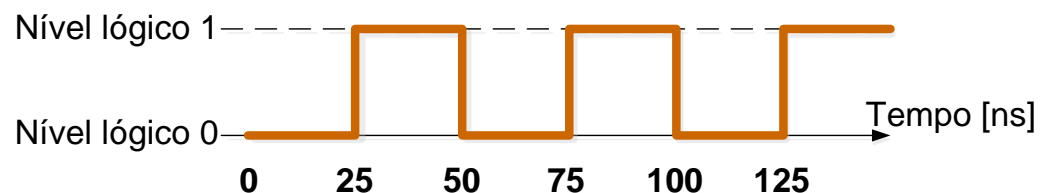
## Geração do sinal de relógio (clk)

91

Sistemas Digitais, 2013

```
...
begin
  -- declaração da instancia para teste
  ...
  -- gerador de sinal para o relógio
process
begin
  clk <= '0';
  wait for 25 ns; -- o periodo do relógio é 50 ns
  clk <= '1';
  wait for 25 ns; -- o periodo do relógio é 50 ns
end process;
...
end behavior;
```

**Diagrama temporal do sinal clk resultante**



## Simulação e teste do circuito “Cadeado Digital (v2)”

# Geração dos sinais lock e unlock

92

Sistemas Digitais, 2013

```
-- gerador de sinal para o relógio
process
begin
    clk <= '0';
    wait for 25 ns; -- o periodo do relógio é 50 ns
    clk <= '1';
    wait for 25 ns; -- o periodo do relógio é 50 ns
end process;

-- gerador dos sinais lock e unlock
process
begin
    lock <= '0'; unlock <= '0';
    wait for 2*25 ns; -- espera 1 periodo de relógio
    lock <= '0'; unlock <= '1';
    wait for 2*25 ns; -- espera 1 periodo de relógio
    lock <= '1'; unlock <= '0';
    wait for 2*25 ns; -- espera 1 periodo de relógio
    lock <= '1'; unlock <= '1';
    wait for 2*25 ns; -- espera 1 periodo de relógio
end process;
```

...

# Simulação e teste do circuito “Cadeado Digital (v2)”

## Geração do código

93

Sistemas Digitais, 2013

```
...  
-- gerador dos sinais lock e unlock  
process  
begin  
    lock <= '0'; unlock <= '0';  
    wait for 2*25 ns; -- espera 1 periodo de relógio  
    lock <= '0'; unlock <= '1';  
    wait for 2*25 ns; -- espera 1 periodo de relógio  
    lock <= '1'; unlock <= '0';  
    wait for 2*25 ns; -- espera 1 periodo de relógio  
    lock <= '1'; unlock <= '1';  
    wait for 2*25 ns; -- espera 1 periodo de relógio  
end process;  
-- gerador do sinal code  
process  
begin  
    code <= code + 1;  
    wait for 4*2*25 ns;  
end process;  
...
```

## Inicialização da máquina de estados

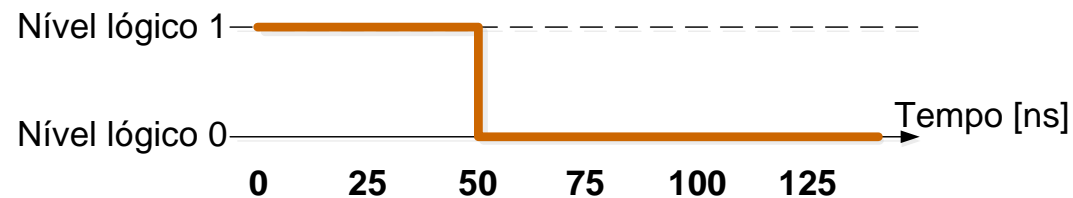
94

Sistemas Digitais, 2013

```
...
-- gerador do sinal code
process
begin
    code <= code + 1;
    wait for 4*2*25 ns;
end process;
-- gerador do sinal de inicialização
process
begin
    reset <= '1';
    wait for 50 ns;
    reset <= '0';
    wait; -- este gerador de sinal fica aqui parado
end process;

end behavior;
```

**Diagrama temporal do sinal reset resultante**



# Simulação e teste do circuito “Cadeado Digital (v2)”

## tb\_cadeado\_v2.vhd

95

Sistemas Digitais, 2013

```
-- FICHEIRO teste_cadeado_v2.vhd
-- Declaração de bibliotecas
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- Entidade de teste
entity tb_cadeado_v2 is
end tb_cadeado_v2;

architecture behav of tb_cadeado_v2 is
-- declaração do componente para teste
component cadeado_digital_v2
port (
    code    : in  std_logic_vector(3 downto 0);
    xopen   : in  std_logic;
    xclose  : in  std_logic;
    clk     : in  std_logic;
    reset   : in  std_logic;
    OK      : out std_logic;
    Err     : out std_logic;
    lock_state : out std_logic
);
end component;

-- Sinais para o testbench
signal lock, unlock, clk, reset : std_logic := '0';
signal code:std_logic_vector(3 downto 0) := "1111";
signal OK, Err, lock_state: std_logic;

begin
```

```
-- Instancia para teste
test_unit: cadeado_digital_v2
    port map ( clk=>clk,
        reset=>reset, code=>code,
        xclose=>lock, xopen=>unlock,
        OK=>OK, Err=>Err,
        lock_state=>lock_state );
```

```
-- Sinal de relógio
process
begin
    clk <= '0';
    wait for 25 ns;
    clk <= '1';
    wait for 25 ns;
end process;
```

```
-- Sinais lock e unlock
process
begin
    lock <= '0'; unlock <= '0';
    wait for 2*25 ns;
    lock <= '0'; unlock <= '1';
    wait for 2*25 ns;
    lock <= '1'; unlock <= '0';
    wait for 2*25 ns;
    lock <= '1'; unlock <= '1';
    wait for 2*25 ns;
end process;
```

```
-- Sinal code
process
begin
    code <= code + 1;
    wait for 4*2*25 ns;
end process;

-- Sinal de inicialização
process
begin
    reset <= '1';
    wait for 50 ns;
    reset <= '0';
    wait;
end process;

end behav;
```

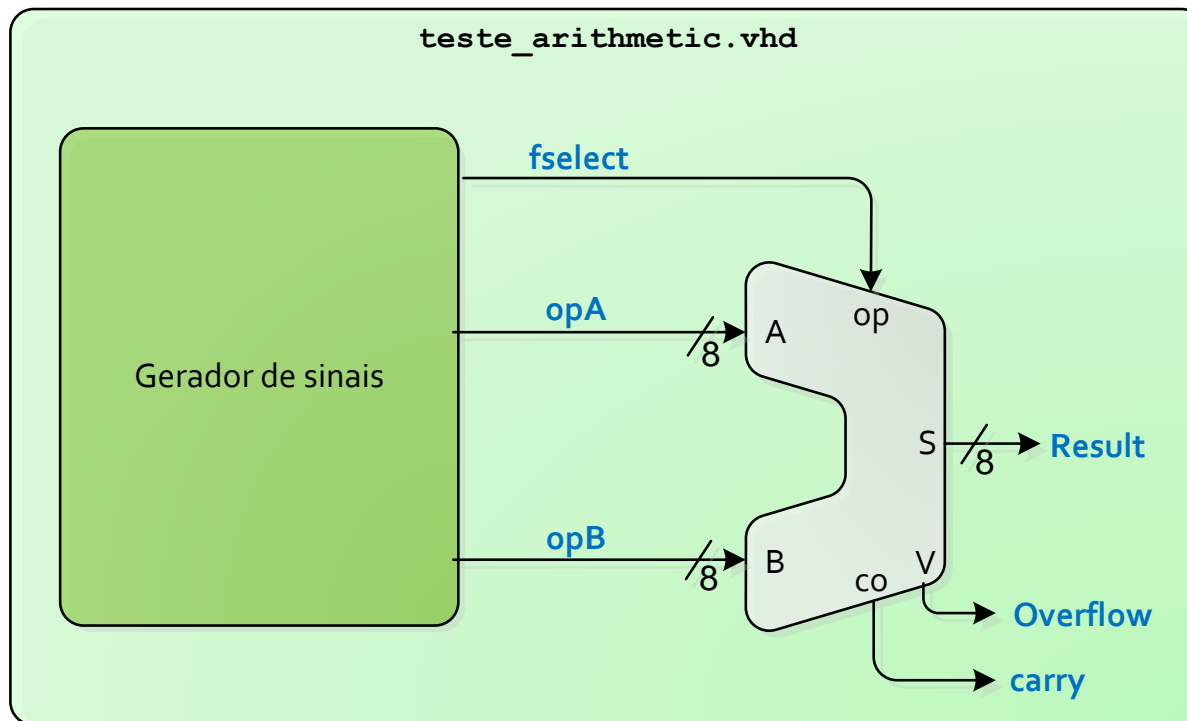
# Simulação e teste da “Unidade Aritmética”

## Descrição da arquitectura

96

Sistemas Digitais, 2013

### Componente para teste:





# Simulação e teste da “Unidade Aritmética”

## Descrição da entidade

97

Sistemas Digitais, 2013

### □ Descrição da entidade

#### ▣ Sem entradas/saídas

```
-- FICHEIRO tb_arithmetic.vhd

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

-- Definição do nome da entidade, sem qualquer entrada ou saída
entity tb_arithmetic is
end tb_arithmetic;

architecture behavior of tb_arithmetic is
...
```

## Simulação e teste da “Unidade Aritmética”

# Declaração de componentes e sinais

98

Sistemas Digitais, 2013

```
...  
architecture behavior of tb_arithmetic is  
-- Declaração do componente arithmetic_unit  
component arithmetic_unit  
    port (  
        A  : in   std_logic_vector(7 downto 0);  
        B  : in   std_logic_vector(7 downto 0);  
        Op : in   std_logic;  
        S  : out  std_logic_vector(7 downto 0);  
        co : out  std_logic;  
        V  : out  std_logic  
    );  
end component;  
  
-- Declaração dos sinais para o testbench  
signal fselect : std_logic := '0';  
signal opA, opB : std_logic_vector(7 downto 0) := "00000000";  
signal result : std_logic_vector(7 downto 0);  
signal overflow, carry: std_logic;  
  
begin
```

## Simulação e teste da “Unidade Aritmética”

# Descrição da unidade para teste

99

Sistemas Digitais, 2013

```
...
architecture behavior of teste_cadeado_v2 is
  -- Declaração do componente arithmetic_unit
  ...
  -- Declaração dos sinais para o testbench
  ...
begin
  -- declaração da instancia para teste
  test_unit: arithmetic_unit port map (
    A  => opA, B => opB,
    op => fselect,
    S  => result,
    V  => overflow,
    Co => carry
  );
  ...
end behavior;
```

## Simulação e teste da “Unidade Aritmética”

# Geração dos sinais de dados/controlo

100

Sistemas Digitais, 2013

```
...
begin
  -- declaração da instancia para teste
  ...
  -- gerador dos sinais de dados/controlo
process
begin
  fselect <= '0';  -- operação de soma
  opA <= x"08";  -- inicialização em hexadecimal
  opB <= x"FE";  -- inicialização em hexadecimal
  wait for 20 ns;
  fselect <= '1';  -- operação de subtração
  wait for 20 ns;
  fselect <= '0';  -- operação de soma
  opA <= x"A8";  -- inicialização em hexadecimal
  opB <= x"FE";  -- inicialização em hexadecimal
  wait for 20 ns;
  ...
end process;
...
end behavior;
```

Por vezes não é possível fazer testar todos os valores possíveis para as entradas.

Por exemplo, neste caso o número de combinações possíveis para as entradas (i.e., o número de linhas da tabela de verdade) é de:

$$2 \times 2^8 \times 2^8 = 2^{17}.$$

Assim deve ser escolhido um conjunto representativo de valores da tabela de verdade de forma a testar todos o maior número de casos possíveis.

Criação e síntese de circuitos no Xilinx ISE

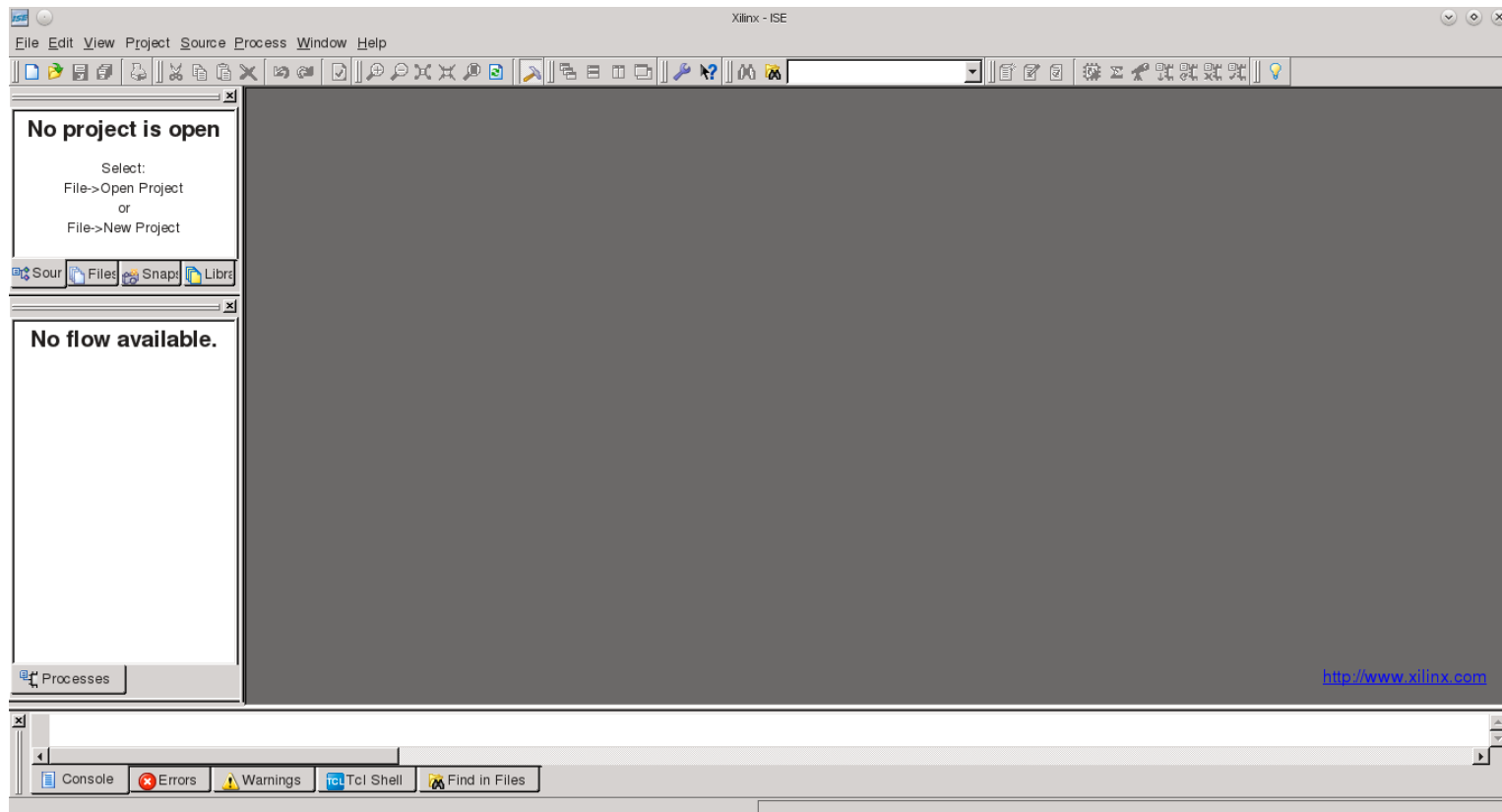
Simulação de circuitos

## Xilinx ISE no Boole

102

Sistemas Digitais, 2013

- Após arrancar a máquina virtual, fazer:
  - Ice WM (start menu) → Xilinx → ISE

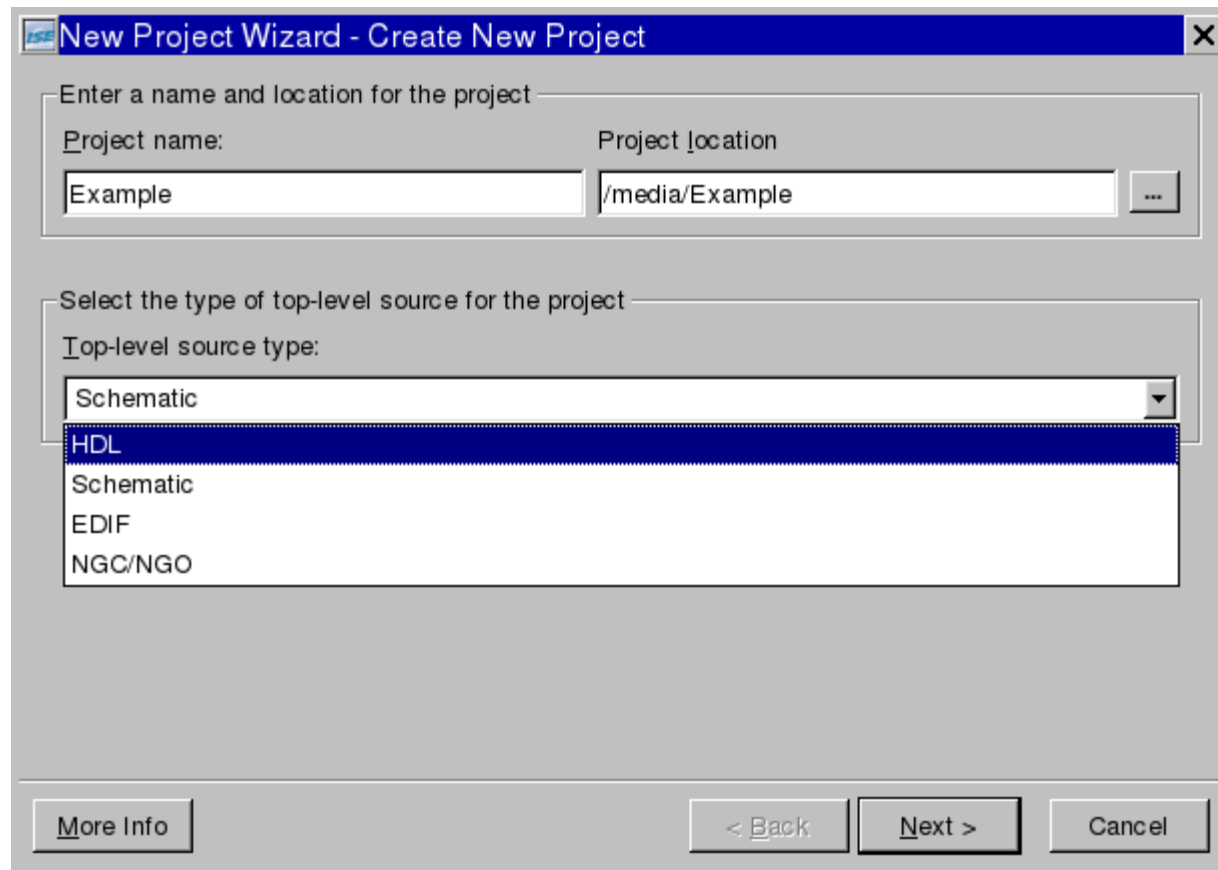


# Criação de um projecto

103

Sistemas Digitais, 2013

- File → New Project



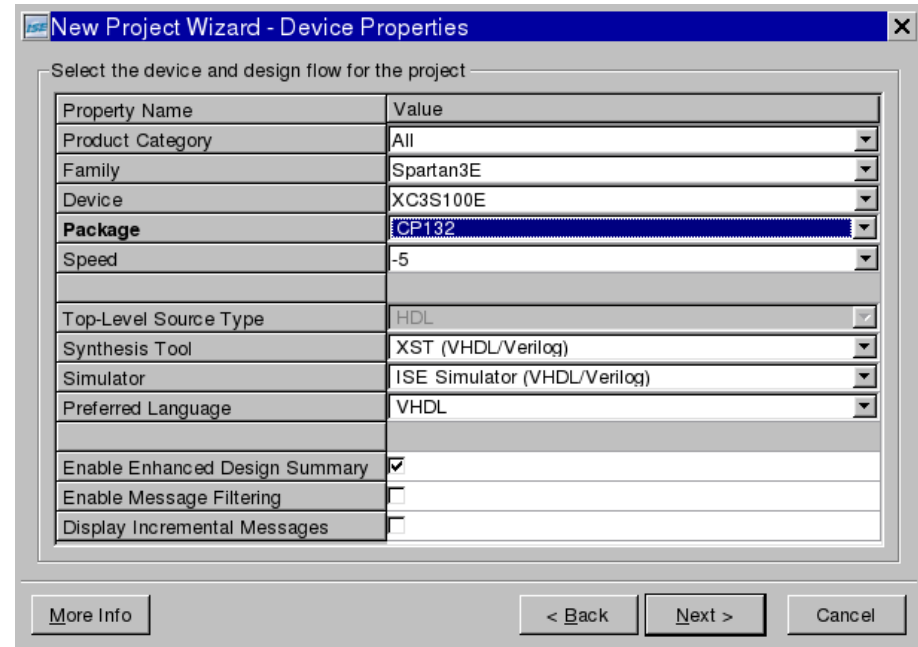
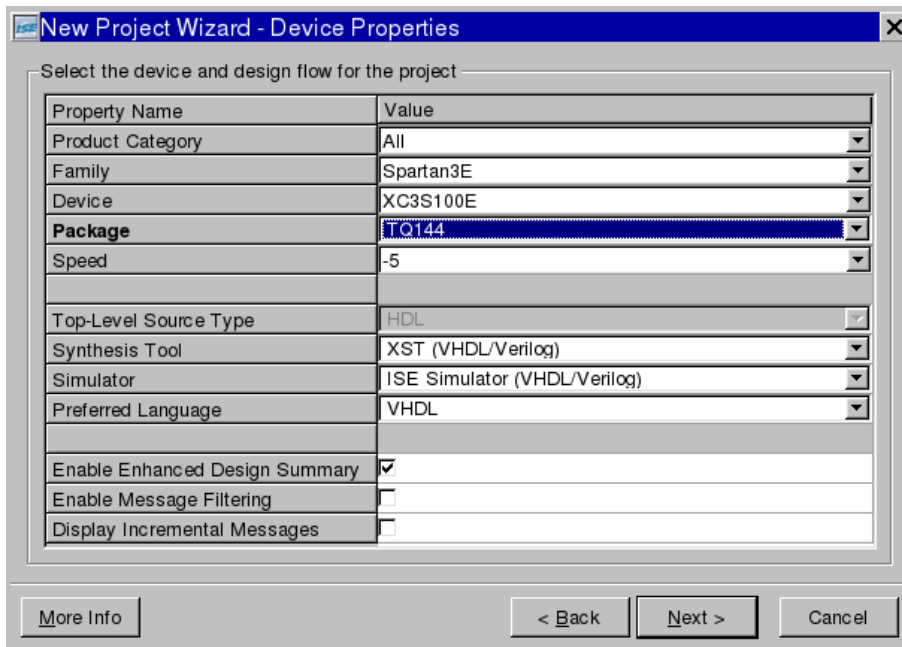
Identificar o nome do projecto, a pasta e o tipo de projecto (HDL)

## Criação de um projecto

104

Sistemas Digitais, 2013

LE3



LSD1



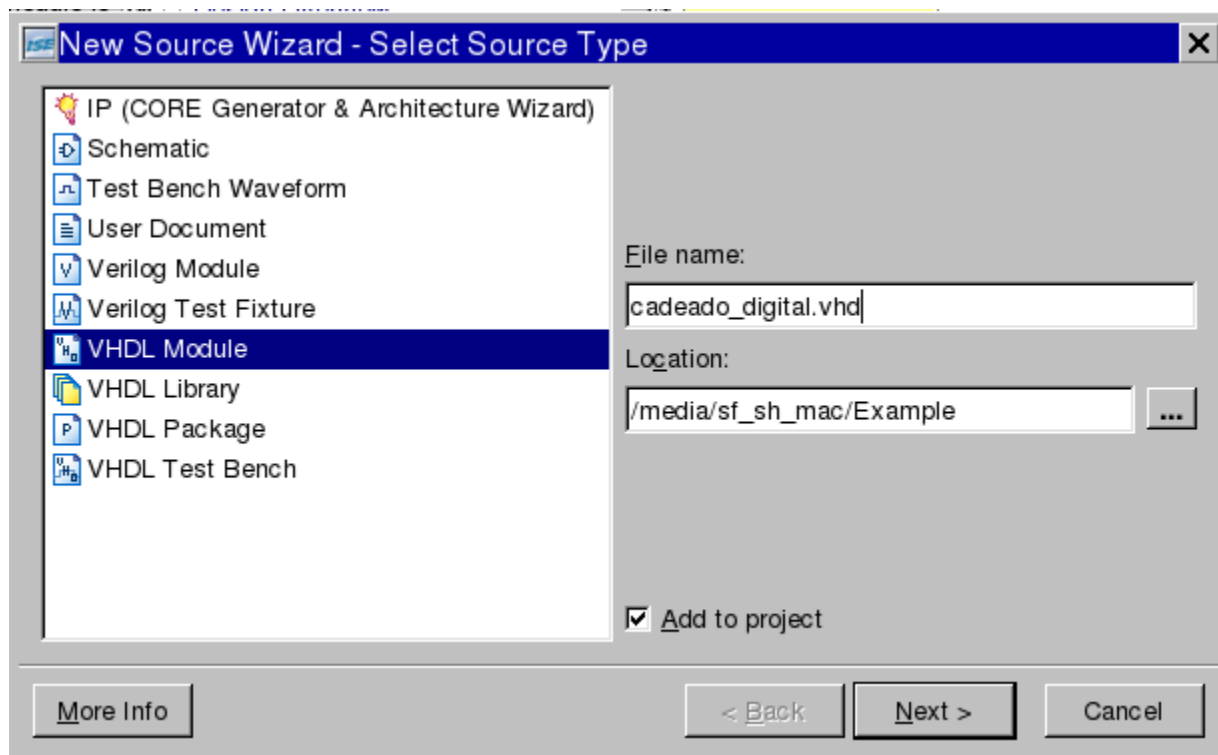
## Criação de um ficheiro VHDL para...

# Descrição de um circuito VHDL

105

Sistemas Digitais, 2013

□ File → New Source



□ Fazer *next* nos menus seguintes...

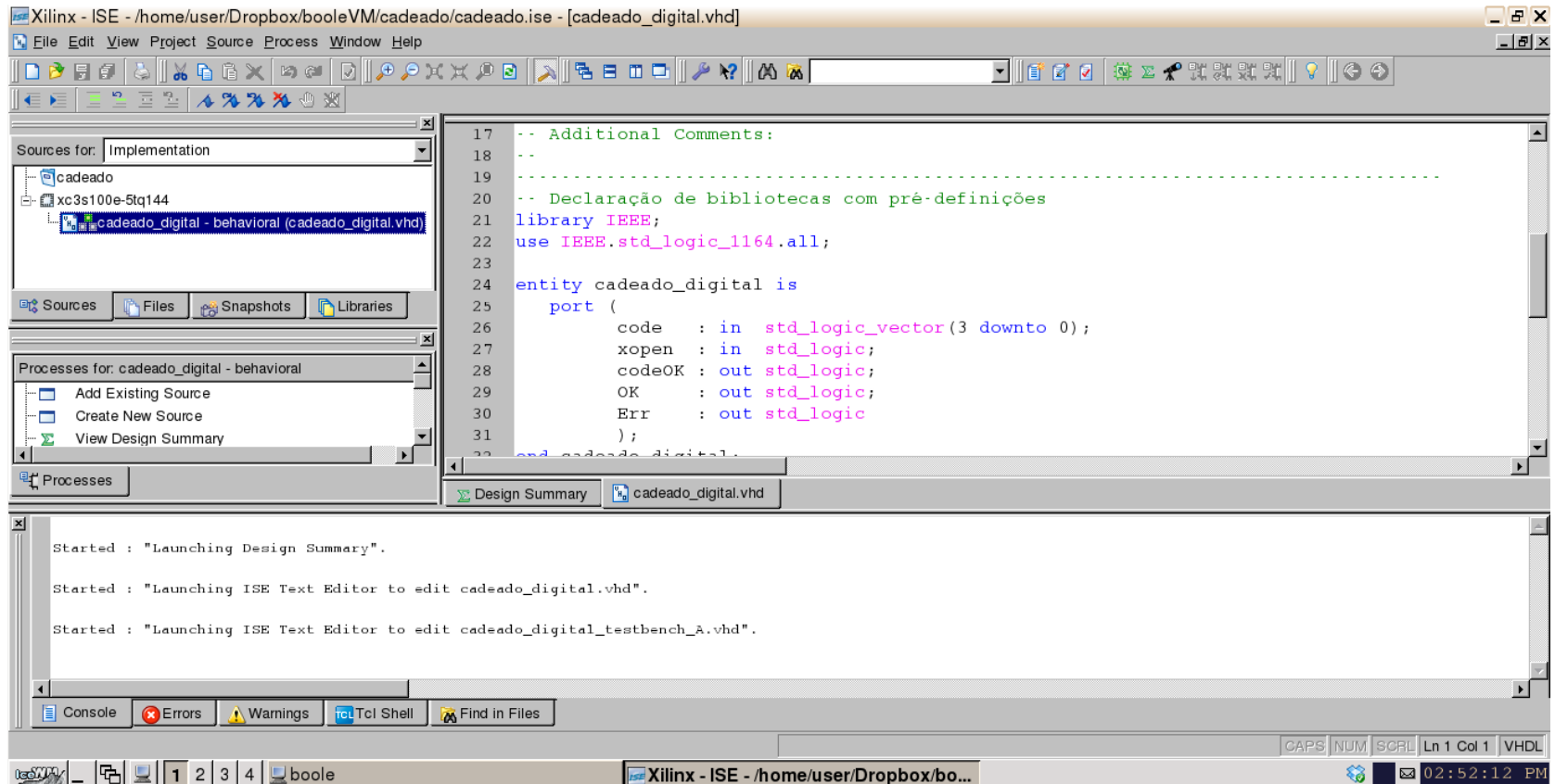
## Criação de um ficheiro VHDL para...

# Descrição de um circuito VHDL

106

Sistemas Digitais, 2013

## □ Ambiente de desenvolvimento



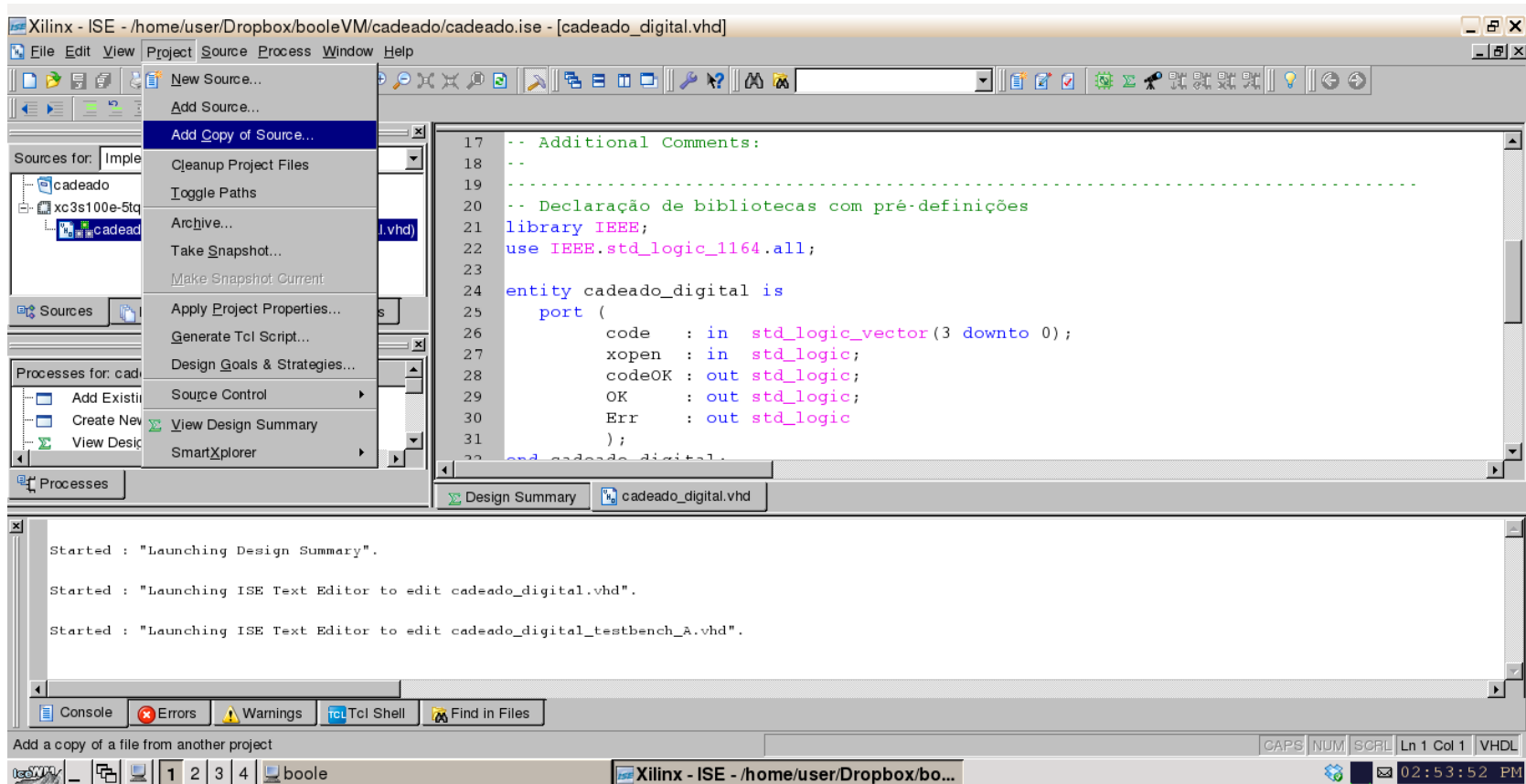
# Adicionar ao projecto um ficheiro VHDL existente

## Add Copy of Source

107

Sistemas Digitais, 2013

□ File → Add Copy of Source



## Criação de um ficheiro VHDL para...

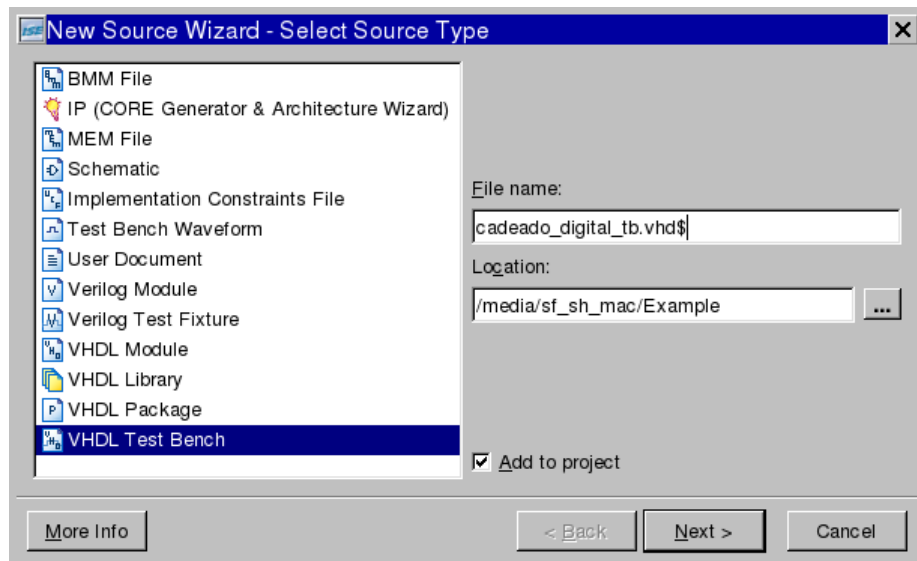
# Simulação de um circuito em VHDL

108

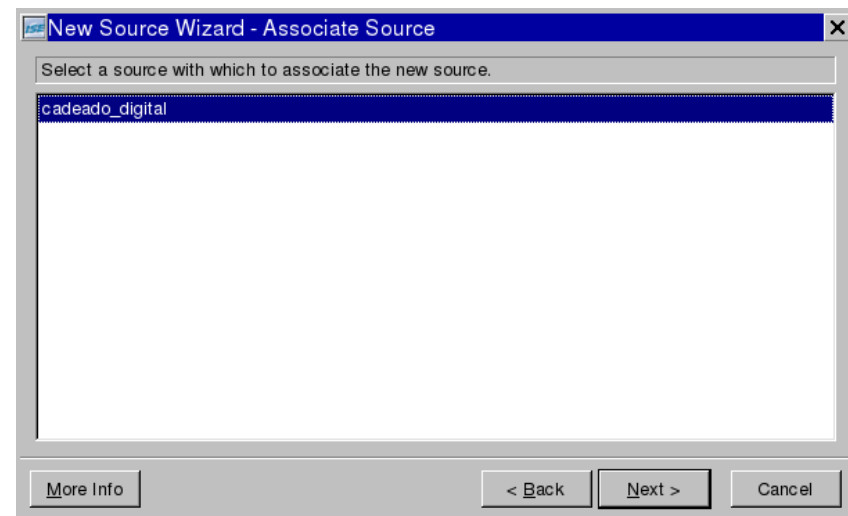
Sistemas Digitais, 2013

□ File → New Source

## 1. Selecionar “VHDL Test Bench”



## 2. Selecionar o circuito a simular



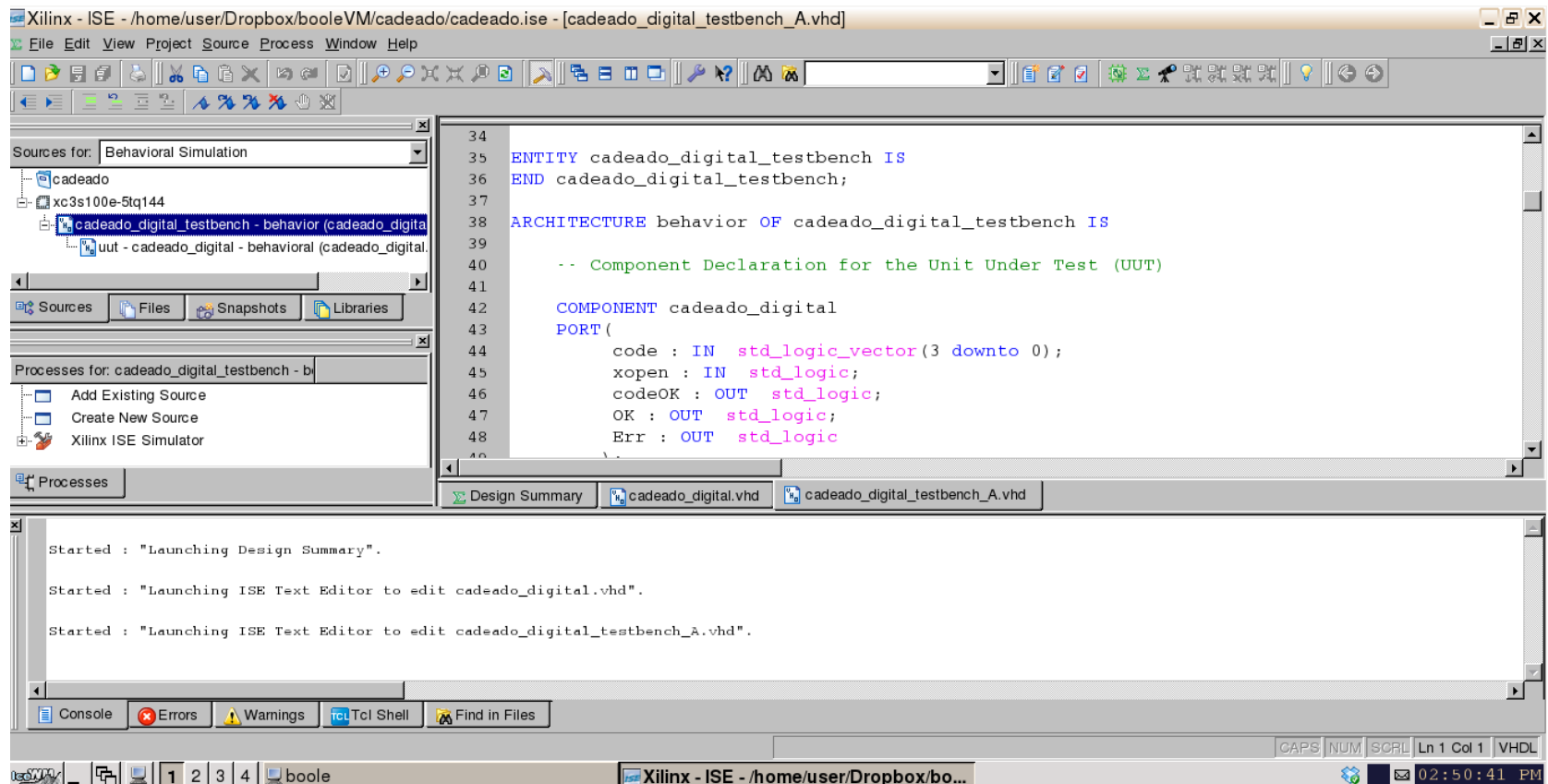
## Criação de um ficheiro VHDL para...

# Simulação de um circuito em VHDL

109

Sistemas Digitais, 2013

## □ Ambiente de desenvolvimento



## Criação de um ficheiro VHDL para...

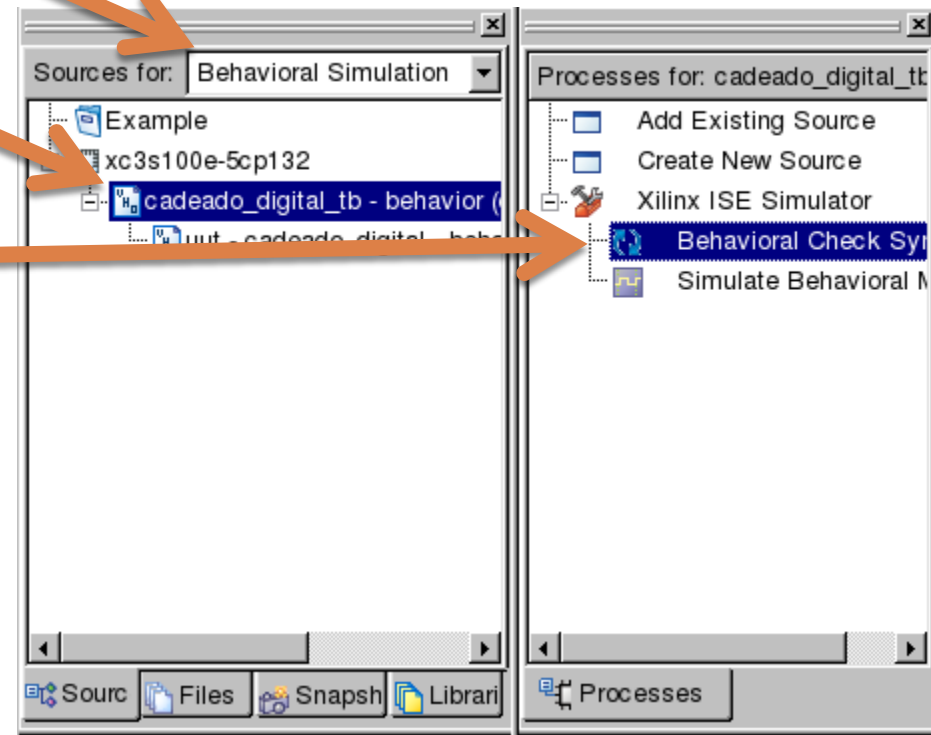
# Simulação de um circuito em VHDL

110

Sistemas Digitais, 2013

### □ Simular o circuito:

1. Mudar para modo simulação
2. Selecionar o ficheiro de simulação
3. Simular



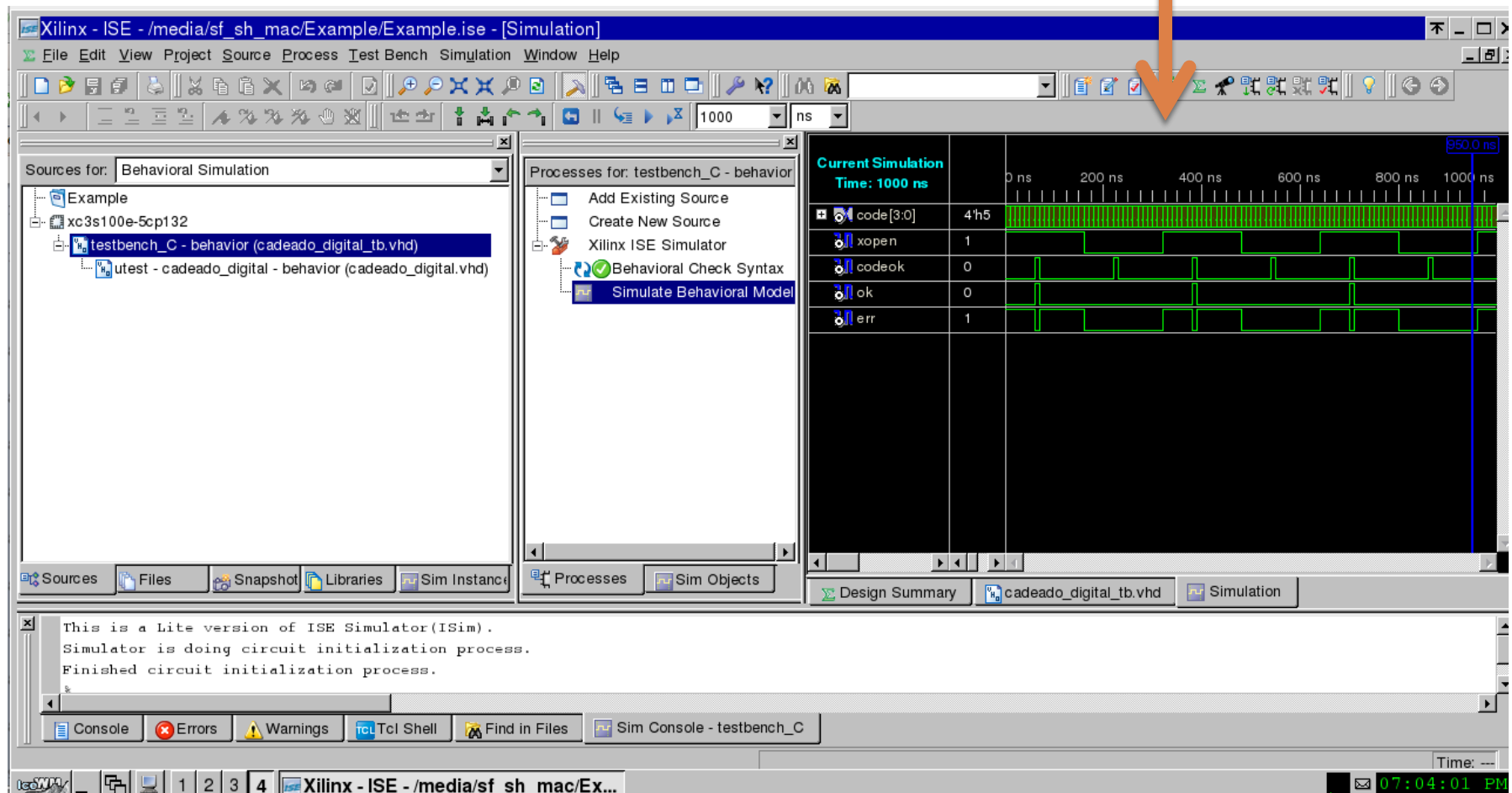
Criação de um ficheiro VHDL para...

# Simulação de um circuito em VHDL

111

Sistemas Digitais, 2013

## Simulação:



The screenshot shows the Xilinx ISE simulation environment. The title bar indicates the file path: `Xilinx - ISE - /media/sf_sh_mac/Example/Example.ise - [Simulation]`. The menu bar includes `File`, `Edit`, `View`, `Project`, `Source`, `Process`, `Test Bench`, `Simulation`, `Window`, and `Help`. The toolbar contains various icons for file operations, simulation, and debugging.

The **Sources** panel on the left shows the project structure for `Behavioral Simulation`. It includes a tree view with the following items:

- Example
  - xc3s100e-5cp132
    - testbench\_C - behavior (cadeado\_digital\_tb.vhd)
    - utest - cadeado\_digital - behavior (cadeado\_digital.vhd)

The **Processes** panel in the center shows the simulation process for `testbench_C - behavior`. The process list includes:

- Add Existing Source
- Create New Source
- Xilinx ISE Simulator
- Behavioral Check Syntax
- Simulate Behavioral Model

The **Simulation** window on the right displays the current simulation results. The title bar indicates `Current Simulation Time: 1000 ns`. The window shows a timing diagram with a time axis from 0 ns to 1000 ns. The signal `code[3:0]` is shown as a green waveform. The signal `xopen` is shown as a green waveform. The signal `codeok` is shown as a green waveform. The signal `ok` is shown as a green waveform. The signal `err` is shown as a green waveform.

The **Console** window at the bottom shows the simulation output:

```
This is a Lite version of ISE Simulator(ISim).
Simulator is doing circuit initialization process.
Finished circuit initialization process.
```

The status bar at the bottom shows the file path: `Xilinx - ISE - /media/sf_sh_mac/Ex...` and the time: `07:04:01 PM`.

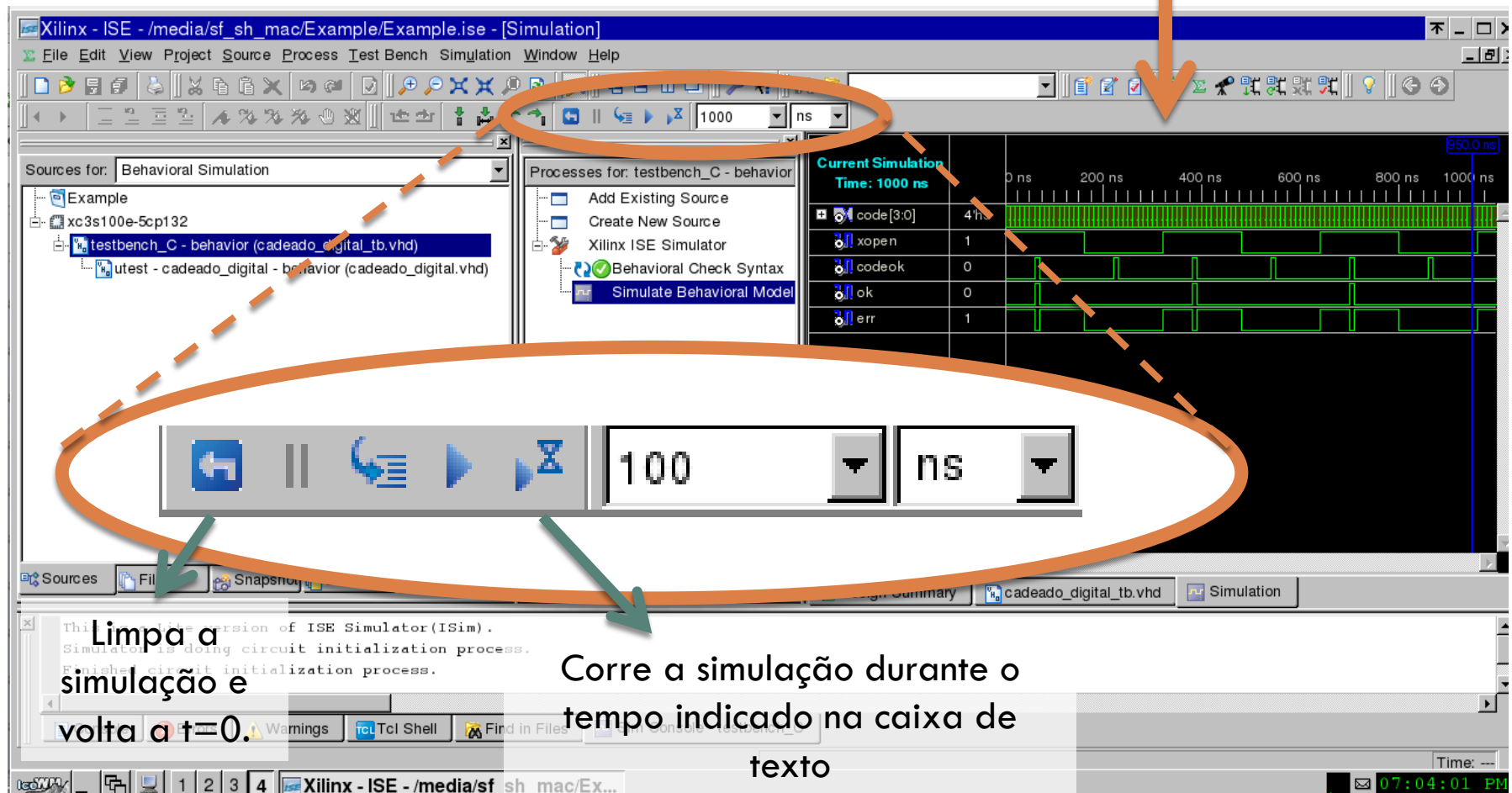
Criação de um ficheiro VHDL para...

# Simulação de um circuito em VHDL

112

Sistemas Digitais, 2013

## Simulação:



The screenshot shows the Xilinx ISE simulation environment. The top toolbar contains various simulation controls. A large orange oval highlights the simulation controls, including the 'Run' button (a green play icon) and a time input field set to '100 ns'. An orange arrow points from the 'Run' button to the 'Current Simulation Time: 1000 ns' label in the waveform window. Another orange arrow points from the 'Run' button to the '100 ns' input field. Below the waveform window, a text box contains the message: 'This is the first version of ISE Simulator (ISim). Simulator is doing circuit initialization process. Finish circuit initialization process.' At the bottom, a status bar shows the time '07:04:01 PM'.

Limpa a simulação e volta a  $t=0$ .

Corre a simulação durante o tempo indicado na caixa de texto