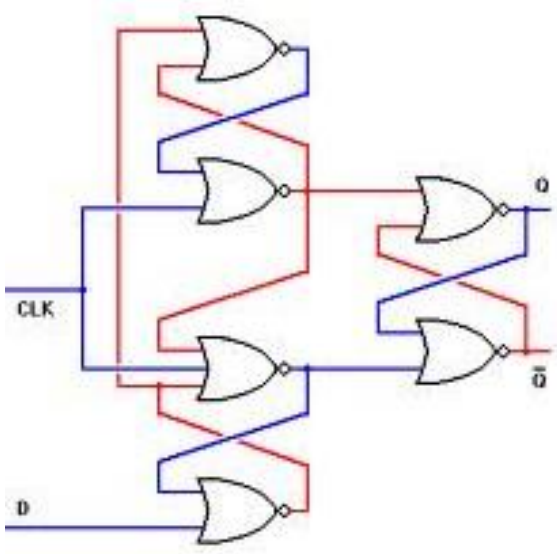
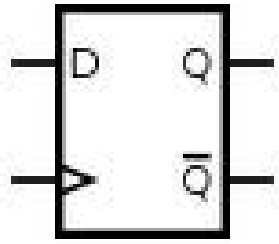
TÉCNICO LISBOA

Sistemas Digitais (SD)

Circuitos Sequenciais Básicos: Simbologia e Descrição em VHDL





TÉCNICO LISBOA

Aula Anterior

- **Na aula anterior:**
  - Elementos básicos de memória
  - Latches
    - Latch RS
    - Latch RS sincronizado
    - Latch D
  - Flip-Flops
    - Flip-flop master-slave
    - Flip-flop JK
    - Flip-flop edge-triggered

TÉCNICO LISBOA

Planeamento

SEMANA	TEÓRICA 1	TEÓRICA 2	PROBLEMAS/LABORATÓRIO
19/Set a 23/Set	Introdução	Sistemas de Numeração e Códigos	
26/Set a 20/Set	Álgebra de Boole	Elementos de Tecnologia	P0
3/Out a 7/Out	Funções Lógicas	Minimização de Funções Booleanas (I)	L0
10/Out a 14/Out	Minimização de Funções Booleanas (II)	Def. Circuito Combinatório; Análise Temporal	P1
17/Out a 21/Out	Circuitos Combinatórios (I) – Codif., MUXs, etc.	Circuitos Combinatórios (II) – Som., Comp., etc.	L1
24/Out a 28/Out	Circuitos Combinatórios (III) - ALUs	Linguagens de Descrição e Simulação de Circuitos Digitais	P2
31/Out a 4/Nov	**FERIADO (1/Nov)**	Circuitos Sequenciais: Latches e Flip-Flops	L2
7/Nov a 11/Nov	Circuitos Sequenciais: Simbologia e Descrição em VHDL	Caracterização Temporal	P3
14/Nov a 18/Nov	Registos	Contadores **Teste 1**	L3
21/Nov a 25/Nov	Síntese de Circuitos Sequenciais: Definições	Síntese de Circuitos Sequenciais: Minimização do número de estados	P4
28/Nov a 2/Dez	Síntese de Circuitos Sequenciais: Síntese com Contadores	**FERIADO (1/Dez)**	L4
5/Dez a 9/Dez	Memórias	**FERIADO (8/Dez)**	P5
12/Dez a 16/Dez	Máq. Estado Microprogramadas: Circuito de Dados e Circuito de Controlo	Máq. Estado Microprogramadas: Endereçamento Explícito/Implícito	L5
19/Dez a 23/Dez	**Férias Natal**	**Férias Natal**	**Férias Natal**

TÉCNICO LISBOA

Sumário

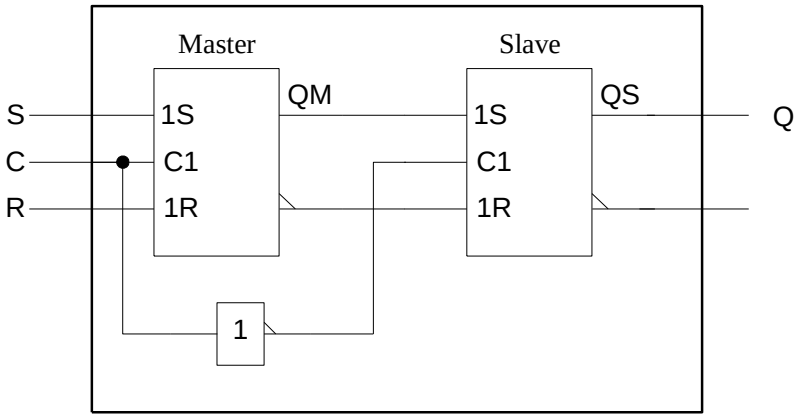
- **Tema da aula de hoje:**
  - Flip-Flops
    - Flip-flop master-slave
    - Flip-flop edge-triggered
    - Entradas assíncronas
  - Simbologia
  - Descrição e Simulação de Circuitos Sequenciais em VHDL

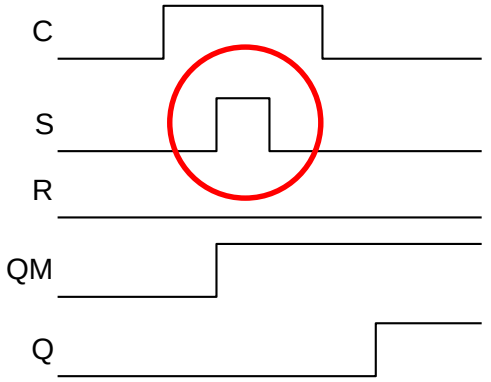
- **Bibliografia:**
  - **M. Mano, C. Kime**: Secções 5.3 e 5.6
  - **G. Arroz, J. Monteiro, A. Oliveira**: Secção 6.4

Flip-Flops

Flip-Flops Master-Slave Pulse-Triggered

- Os flip-flops master-slave respondem aos valores na entrada que existam durante o semi-período em que C = 1. Por isso, são também chamados de **pulse-triggered**.

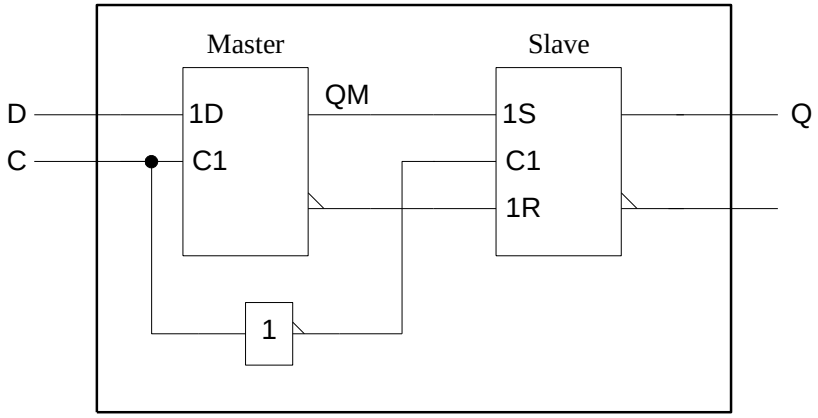


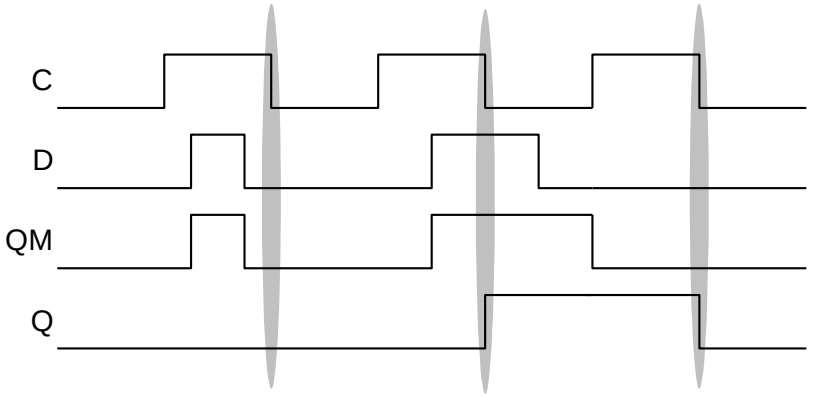


- **Problema:** se durante o pulso de relógio R = 0 e S = 0 → 1 → 0, esperar-se-ia que o flip-flop mantivesse o estado, pois a última ordem é de HOLD. No entanto, o Mestre respondeu à ordem de SET e é essa ordem que é passada ao Escravo.

Flip-Flops

Flip-Flops Master-Slave Edge-Triggered

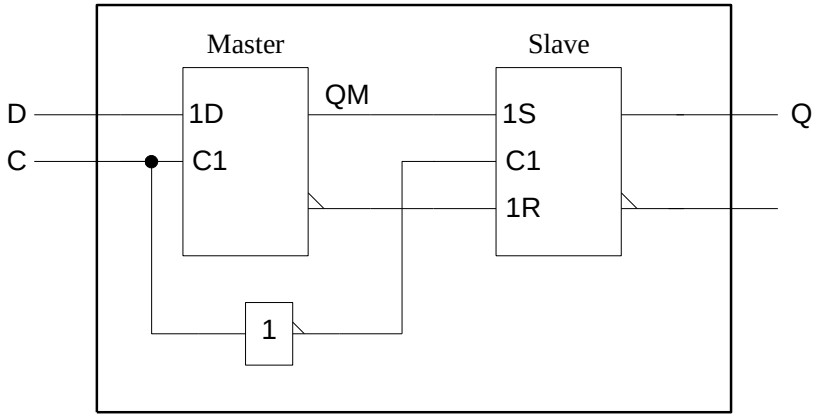


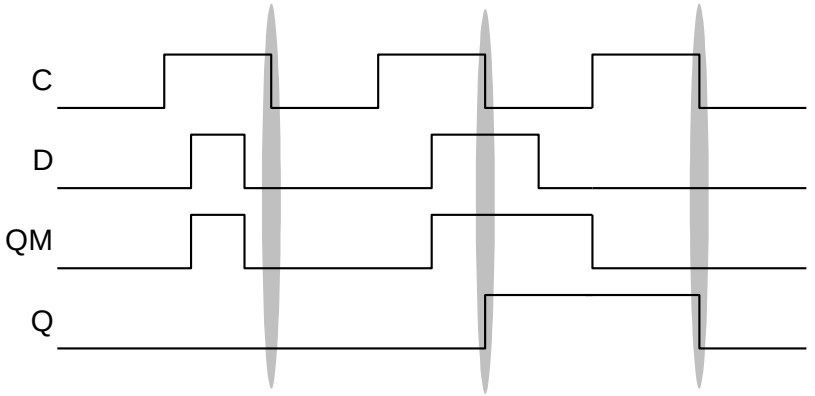


- Os flip-flops **edge-triggered** ignoram o pulso enquanto este se mantém num valor constante, e apenas reagem à transição de relógio.
- Uma estrutura tipo master-slave em que o Mestre é um flip-flop D funciona como edge-triggered (e não como pulse-triggered): o estado que é passado do Mestre para o Escravo é sempre o estado definido pelas entradas na transição de relógio.

Flip-Flops

Flip-Flops Master-Slave Edge-Triggered





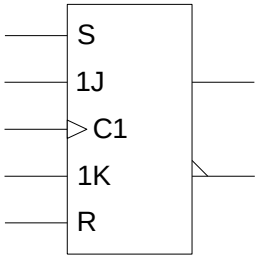
- Os flip-flops dizem-se **positive-edge-triggered** se reagem à transição de relógio 0 → 1.
- Os flip-flops dizem-se **negative-edge-triggered** se reagem à transição de relógio 1 → 0.

Flip-Flops

Entradas Assíncronas

- Alguns flip-flops incluem entradas adicionais que permitem fazer o SET ou o RESET assíncronamente, i.e., independentemente do relógio.
- A entrada de **set assíncrono** é também às vezes designada por “direct set” ou “preset”, e a entrada de **reset assíncrono** é também às vezes designada por “direct reset” ou “clear”.

Exemplo:

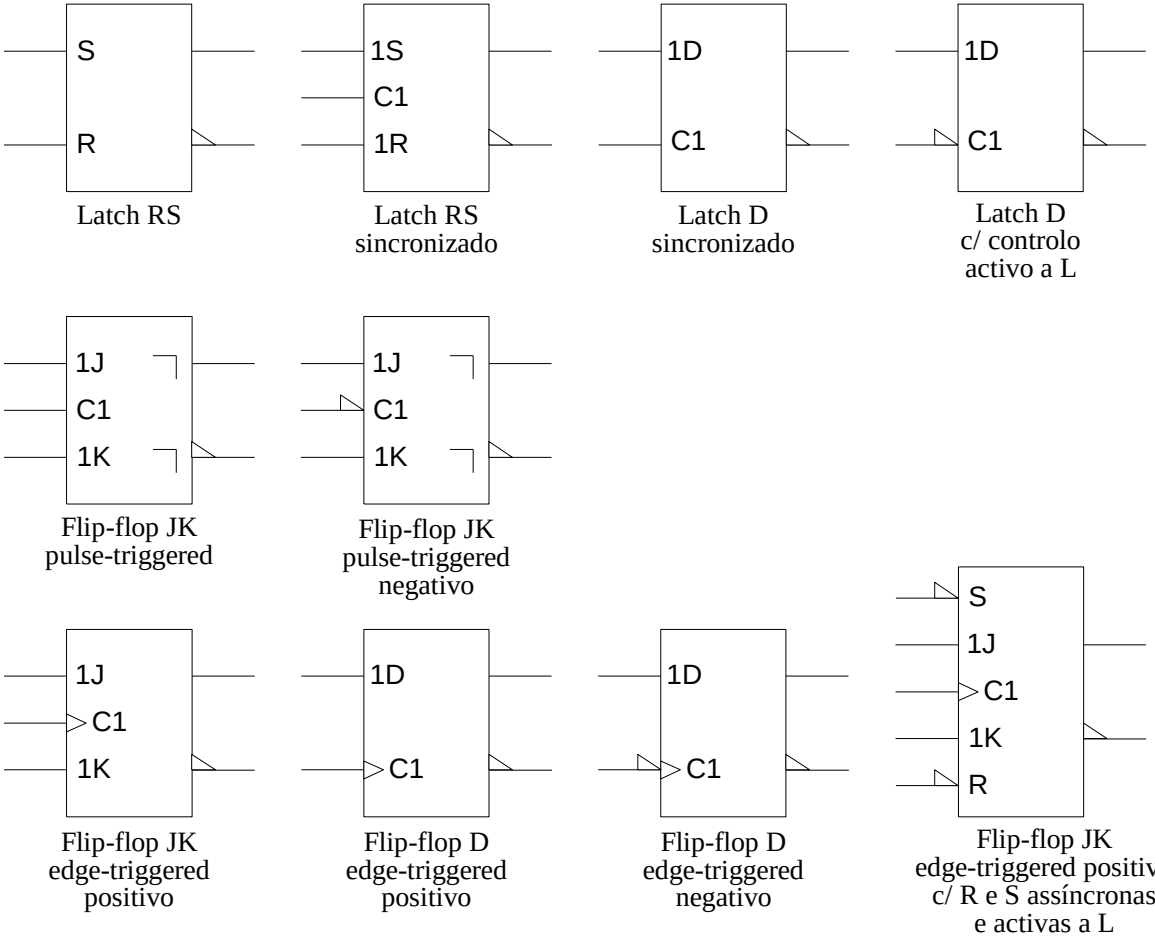


Flip-flop JK com R e S assíncronos

S	R	C	J	K	Q_{n+1}	
0	0	↑	0	0	Q_n	HOLD
0	0	↑	0	1	0	RESET
0	0	↑	1	0	1	SET
0	0	↑	1	1	$\overline{Q}_n$	TOGGLE
1	0	X	X	X	1	SET
0	1	X	X	X	0	RESET
1	1	X	X	X	U	Indefinido

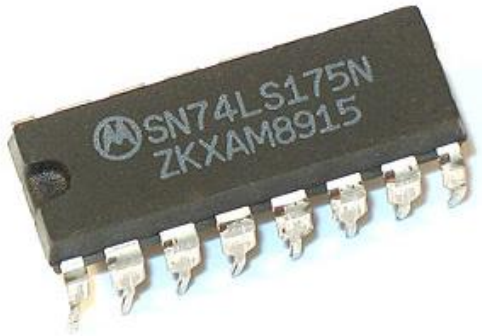
Latches e Flip-Flops

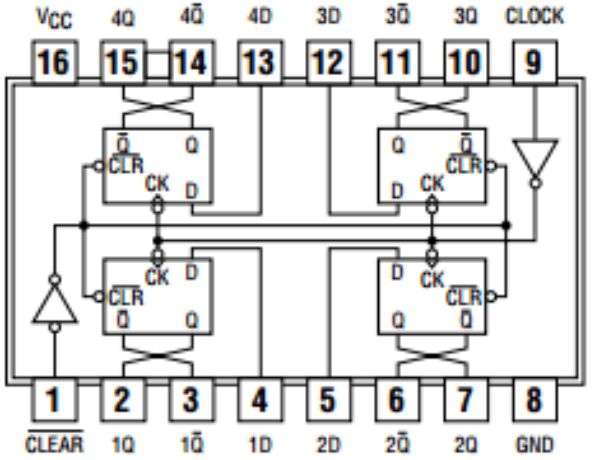
Simbologia

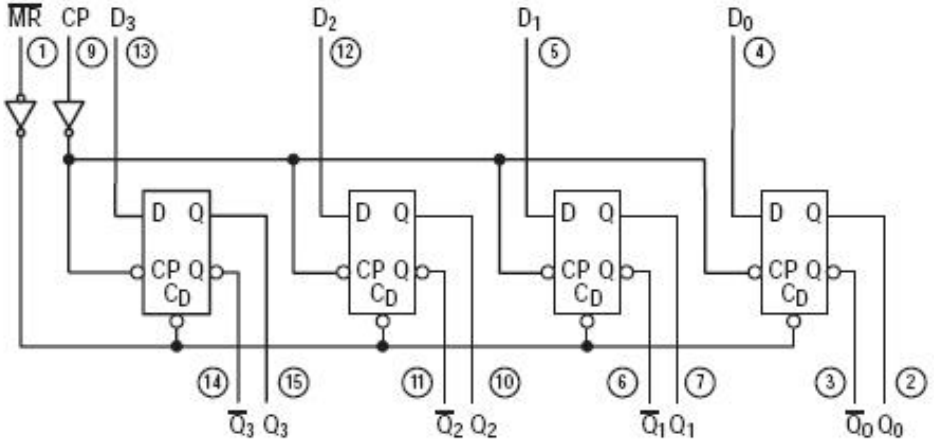


Latches e Flip-Flops

- **Exemplo: 74LS175**



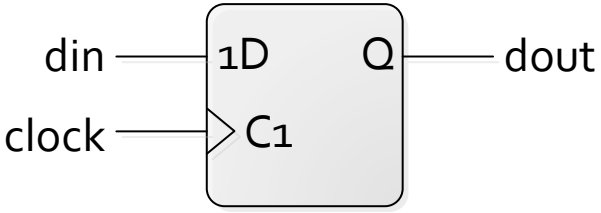




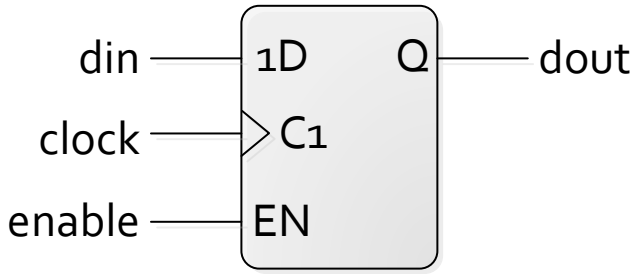
DESCRIÇÃO E SIMULAÇÃO DE CIRCUITOS SEQUENCIAIS EM VHDL

Flip-Flops

Flip-Flops




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

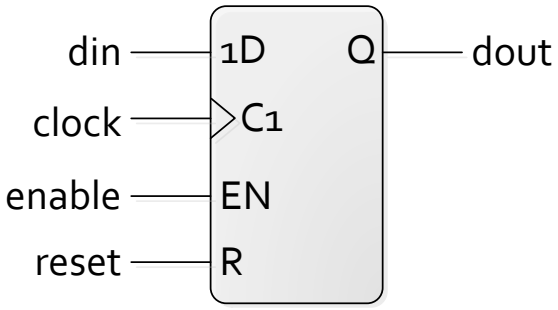



```
dout <= din when rising_edge(clock) and enable='1';
```

NOTA IMPORTANTE: 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 em SD.

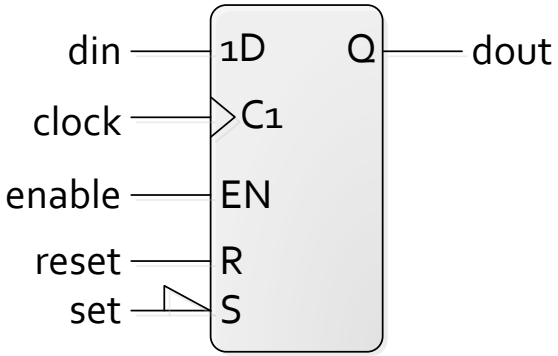
Flip-Flops com SET/RESET assíncronos



CONSTRUÇÕES POSSÍVEIS:

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

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

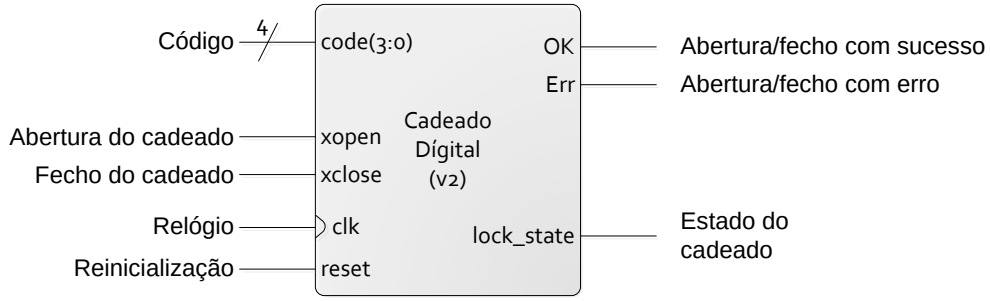


Única situação onde se aceitam os operadores `and`/`or`/`xor`/... após o operador `when`...

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

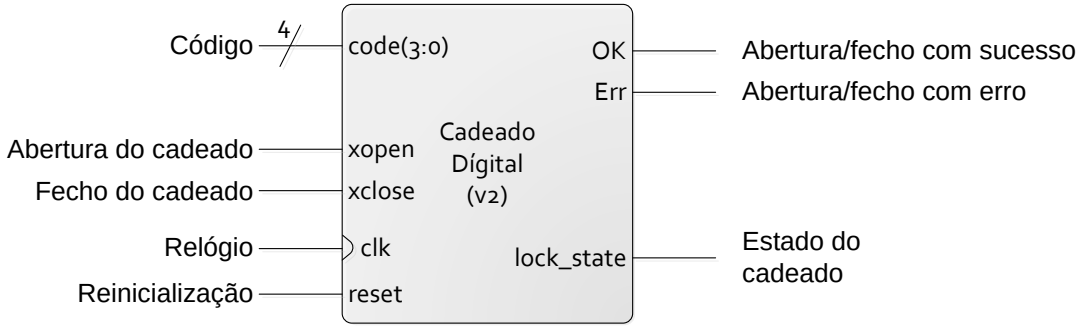
Exemplo: Cadeado Digital (v2)

- **Considere-se 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: 0 = fechado, 1 = aberto.
  - Sinais de controlo (entrada) “clk” e “reset” – sinais de relógio e de reinicialização do cadeado.



Exemplo: Cadeado Digital (v2)

1. Descrição da entidade

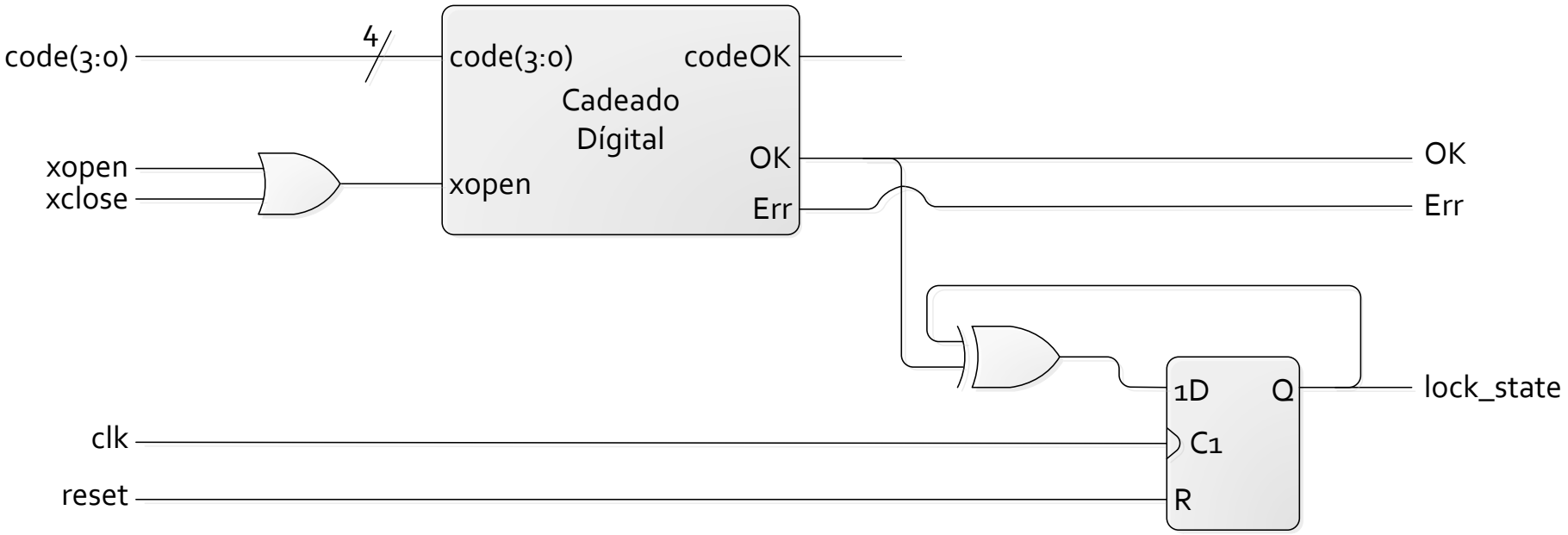



```
entity cadeado_digital_v2 is
   port (
         code       : in  std_logic_vector(3 downto 0);
         xopen      : in  std_logic;
         xclose     : in  std_logic;
         clk        : in  std_logic;
         reset      : in  std_logic;
         OK         : out std_logic;
         Err    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

Exemplo: Cadeado Digital (v2)

2. Desenho do logigrama do cadeado digital

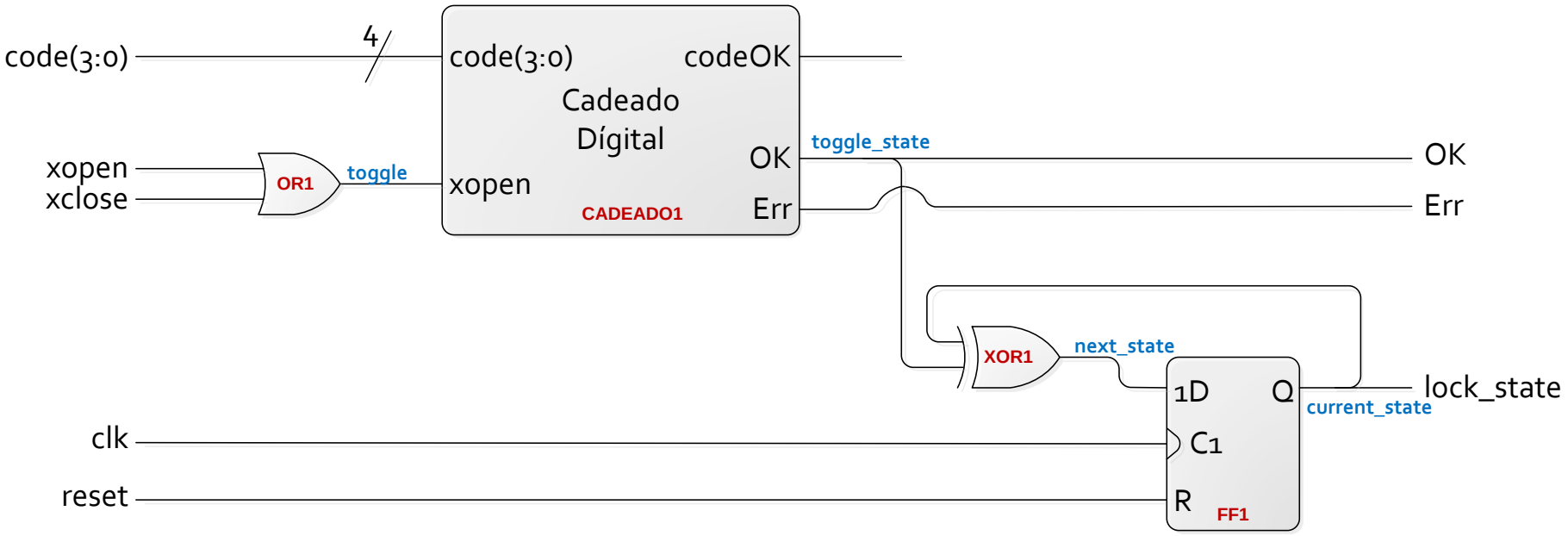
NOTA: o objectivo destes slides não é perceber como se chega a este circuito, apenas como o descrever em VHDL. Assim, assume-se que o circuito está correctamente projectado.



Exemplo: Cadeado Digital (v2)

2. Desenho do logigrama do cadeado digital

Nomeação dos sinais utilizados internamente (azul) e do nome dos circuitos (vermelho)



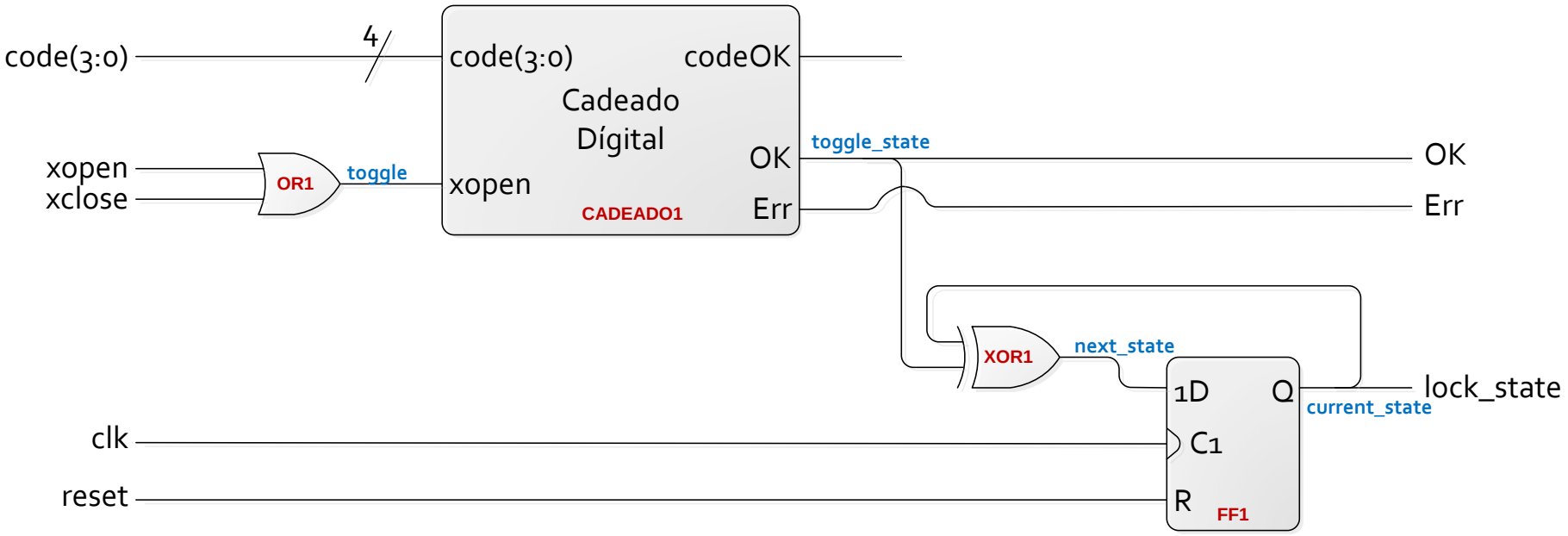
Exemplo: Cadeado Digital (v2)

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



Exemplo: Cadeado Digital (v2)

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 vhdl correspondente

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

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

Exemplo: Cadeado Digital (v2)

3. Implementação da arquitectura

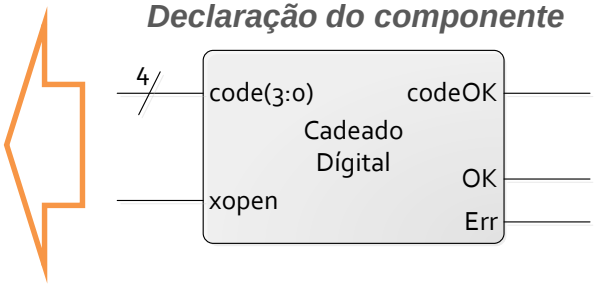
Para utilizar o componente "cadeado_digital", descrito no ficheiro "cadeado_digital.vhd", devem ser tomados os seguintes passos:

A. Declarar o componente

B. Utilizar uma instância do componente

```
...
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
```



Exemplo: Cadeado Digital (v2)

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

```
-- 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
```

Exemplo: Cadeado Digital (v2)

3. Implementação da arquitectura

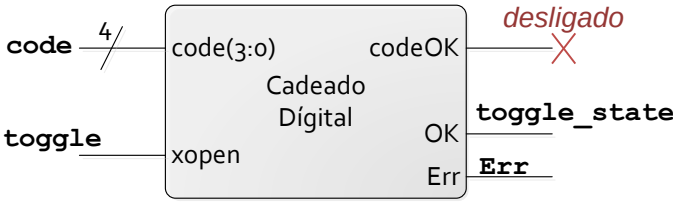
Para utilizar o componente "cadeado_digital", descrito no ficheiro "cadeado_digital.vhd", devem ser tomados os seguintes passos:

A. Declarar o componente

B. Utilizar uma instância do componente

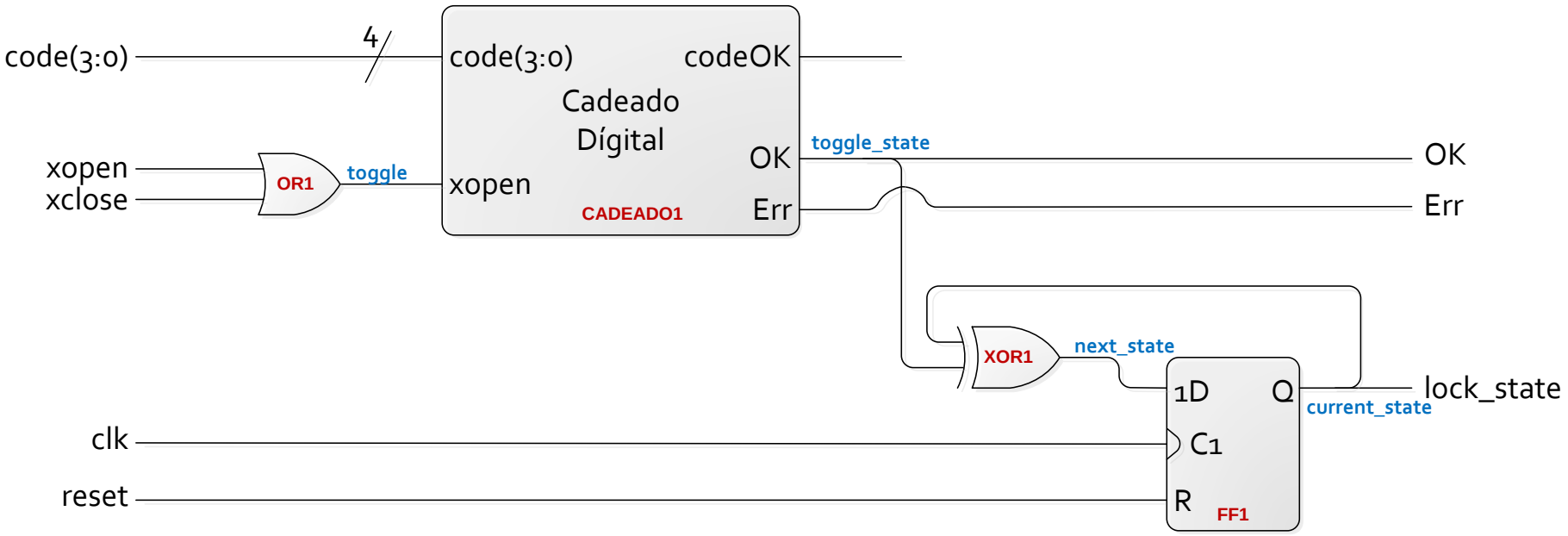
```
...
architecture behavior of cadeado_digital_v2 is
-- Declaração do componente cadeado_digital original
...
-- declaração dos sinais (fios) internos ao componente
signal toggle, toggle_state, next_state, current_state : std_logic;
begin
-- Utilização de 1 instancia do componente "cadeado_digital"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

Instancia "cadeado1" do componente



Exemplo: Cadeado Digital (v2)

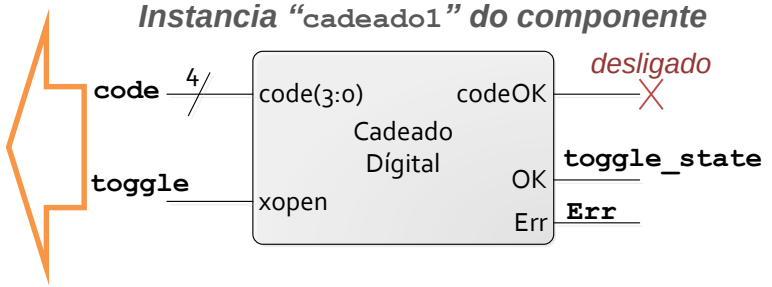
3. Implementação da arquitectura



Exemplo: Cadeado Digital (v2)

3. Implementação da arquitectura

```
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;
```



Exemplo: Cadeado Digital (v2)

```
-- 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;
```

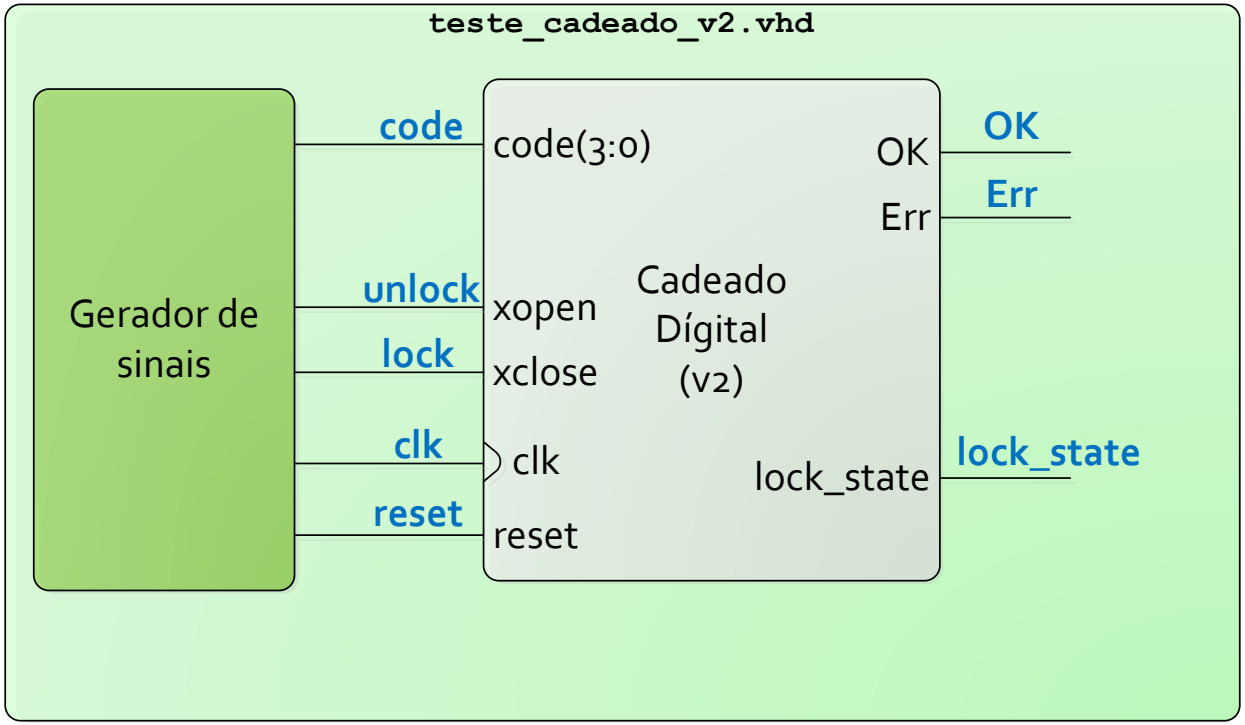
Simulação e teste de circuitos sequenciais

- **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.

Exemplo: Cadeado Digital (v2)

- **Componente para teste:**



Exemplo: Cadeado Digital (v2)

- **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 saida
entity teste_cadeado_v2 is
end teste_cadeado_v2;

architecture behavior of teste_cadeado_v2 is
...
```

Exemplo: Cadeado Digital (v2)

Declaração de componentes e sinais

```
...
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
```

Exemplo: Cadeado Digital (v2)

Descrição da unidade para teste

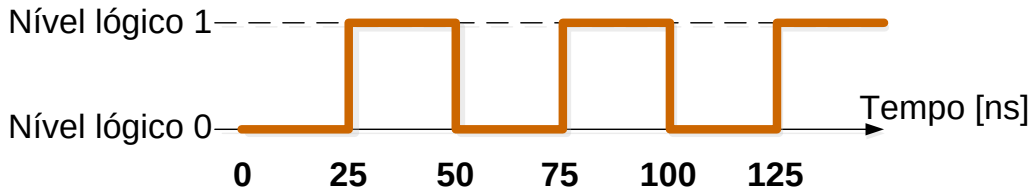
```
...
architecture behavior of teste_cadeado_v2 is
-- Declaração do componente cadeado_digital (v2)
...
-- Declaração dos sinais para o testbench
...
begin
-- declaração da instancia para teste
test_unit: cadeado_digital_v2 port map (
          code    => code,
          xopen   => unlock,
       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

Exemplo: Cadeado Digital (v2)

Geração do sinal de relógio (`clk`)

```
...
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



Exemplo: Cadeado Digital (v2)

- Geração dos sinais `lock` e `unlock`

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

Exemplo: Cadeado Digital (v2)

- **Geração do sinal code**

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

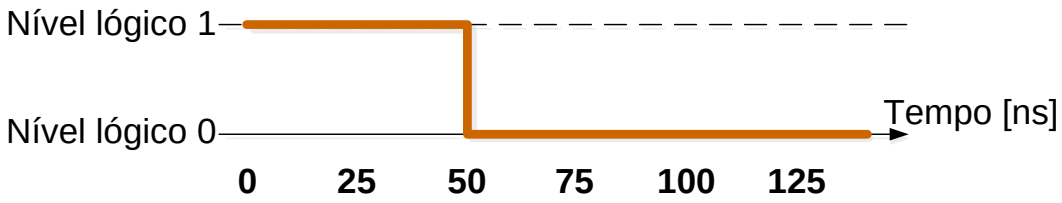
Exemplo: Cadeado Digital (v2)

Inicialização da máquina de estados (`reset`)

```
...
-- 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



Exemplo: Cadeado Digital (v2)

```
-- 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 tese
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;
```

Próxima Aula

- **Tema da Próxima Aula:**
  - Caracterização temporal
  - Metodologia de sincronização temporal

Agradecimentos

Algumas páginas desta apresentação resultam da compilação de várias contribuições produzidas por:

- Guilherme Arroz
- Horácio Neto
- Nuno Horta
- Pedro Tomás