



TÉCNICO LISBOA

SISTEMAS DIGITAIS (SD)

MEEC

Acetatos das Aulas Teóricas

Versão 4.0 - Português

Aula Nº 14:

Título: Circuitos Sequenciais Básicos: Flip-Flops

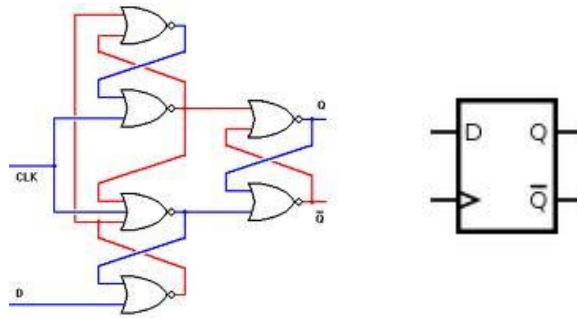
Sumário: Flip-Flops (Flip-flop master-slave, Flip-flop JK, Flip-flop edge-triggered); Simbologia; Descrição e Simulação de Circuitos Sequenciais em VHDL.

2015/2016

Nuno.Roma@tecnico.ulisboa.pt

Sistemas Digitais (SD)

Circuitos Sequenciais Básicos: Flip-Flops



Aula Anterior

■ Na aula anterior:

- ▶ Elementos básicos de memória
- ▶ Latches
 - Latch RS
 - Latch RS sincronizado
 - Latch D
- ▶ Flip-Flops



SEMANA	TEÓRICA 1	TEÓRICA 2	PROBLEMAS/LABORATÓRIO
14/Set a 19/Set	Introdução	Sistemas de Numeração e Códigos	
21/Set a 26/Set	Álgebra de Boole	Elementos de Tecnologia	P0
28/Set a 3/Out	Funções Lógicas	Minimização de Funções Booleanas (I)	L0
5/Out a 10/Out	Minimização de Funções Booleanas (II)	Def. Circuito Combinatório; Análise Temporal	P1
12/Out a 17/Out	Circuitos Combinatórios (I) – Codif., MUXs, etc.	Circuitos Combinatórios (II) – Som., Comp., etc.	L1
19/Out a 24/Out	Circuitos Combinatórios (III) - ALUs	Linguagens de Descrição e Simulação de Circuitos Digitais	P2
26/Out a 31/Out	Circuitos Sequenciais: Latches	Circuitos Sequenciais: Flip-Flops	L2
2/Nov a 7/Nov	Caracterização Temporal	Registos	P3
9/Nov a 14/Nov	Revisões Teste 1	Contadores	L3
16/Nov a 21/Nov	Síntese de Circuitos Sequenciais: Definições	Síntese de Circuitos Sequenciais: Minimização do número de estados	P4
23/Nov a 28/Nov	Síntese de Circuitos Sequenciais: Síntese com Contadores	Memórias	L4
30/Nov a 5/Dez	Máq. Estado Microprogramadas: Circuito de Dados e Circuito de Controlo	Máq. Estado Microprogramadas: Endereçamento Explícito/Implícito	P5
7/Dez a 12/Dez	Circuitos de Controlo, Transferência e Processamento de Dados de um Processador	Lógica Programável	L5
14/Dez a 18/Dez	P6	P6	L6



■ Tema da aula de hoje:

- ▶ Flip-Flops
 - Flip-flop master-slave
 - Flip-flop JK
 - Flip-flop edge-triggered
- ▶ 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

■ Latches vs. Flip-Flops

- Os circuitos básicos de memória podem ser classificados em **latches** e **flip-flops**.



Latches:

- Se a entrada de activação (enable) de um latch sincronizado estiver ligada ao sinal de relógio, o seu estado está continuamente a ser actualizado enquanto o relógio estiver a 1.
- Como não é possível garantir que o estado dos latches se mantem estável durante a fase em que o sinal de relógio estiver a 1, não é também possível garantir que todos os latches mudem sincronamente num circuito complexo.
- Os latches têm aplicações muito específicas (menos complexos, mais rápidos), nomeadamente em circuitos assíncronos.

■ Latches vs. Flip-Flops

- Os circuitos básicos de memória podem ser classificados em **latches** e **flip-flops**.

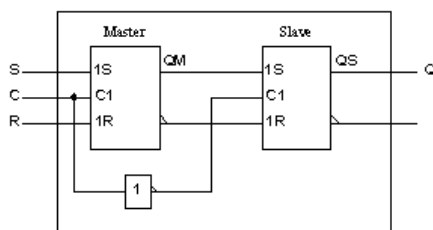


Flip-Flops:

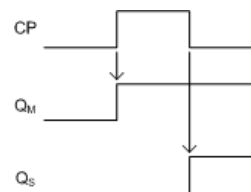
- Os flip-flops mudam as saídas apenas quando há uma variação do relógio (diz-se que são sensíveis ao flanco).
- Este modo de funcionamento garante que o seu estado só é alterado uma única vez em cada período de relógio.
- Esta característica permite que se utilize quase todo o período de relógio para geração de novos valores nas entradas.
- Os circuitos síncronos utilizam, na grande maioria dos casos, flip-flops (sensíveis ao flanco).

■ Flip-Flop Master-Slave

- ▶ O flip-flop Master-Slave consiste na ligação em cascata de 2 latches sincronizados, com sinais de controlo complementares.
- ▶ Funcionamento: o *mestre* “aceita” ordens de Set ou Reset enquanto $C = 1$, mas só “passa” a ordem ao *escravo* quando $C = 0$;
- ▶ Do ponto de vista das saídas externas o estado apenas muda após a transição de $1 \rightarrow 0$ do relógio.

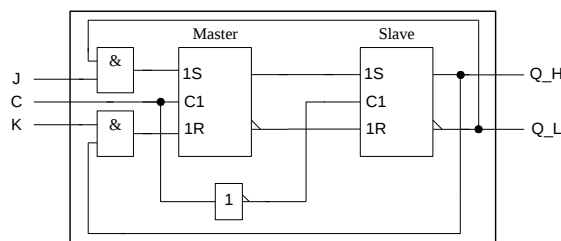


Exemplo: $S=1$ $R=0$



■ Caso Particular: Flip-Flop JK Master-Slave

- ▶ O flip-flop JK permite eliminar o estado indefinido, mantendo 2 entradas e, portanto, 4 funcionalidades distintas.

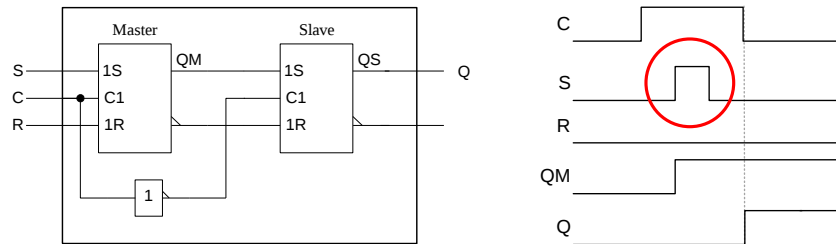


J	K	Q_{n+1}	
0	0	Q_n	HOLD
0	1	0	RESET
1	0	1	SET
1	1	$\overline{Q_n}$	TOGGLE

Nota: continua a só existir mudança de estado (variação nas saídas) após a transição de relógio de $1 \rightarrow 0$.

■ Flip-Flops Master-Slave

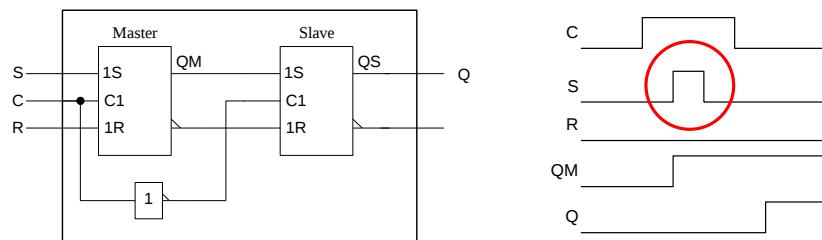
- Os flip-flops master-slave respondem aos valores na entrada que existam durante o semi-período em que $C = 1$. Por isso, são também chamados de **pulse-triggered**.



- No entanto, para o seu funcionamento correcto, não devem ser permitidas variações nas entradas durante o pulso de relógio.

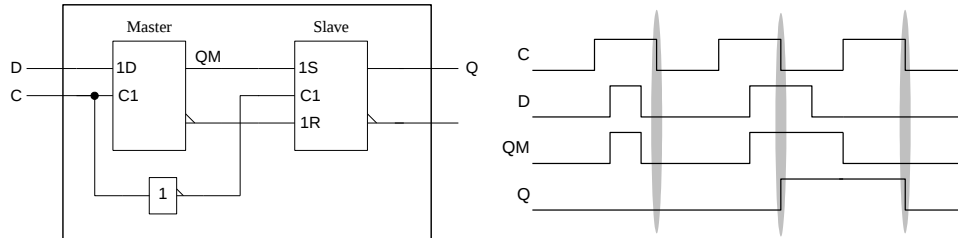
■ Flip-Flops Master-Slave

- 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**.



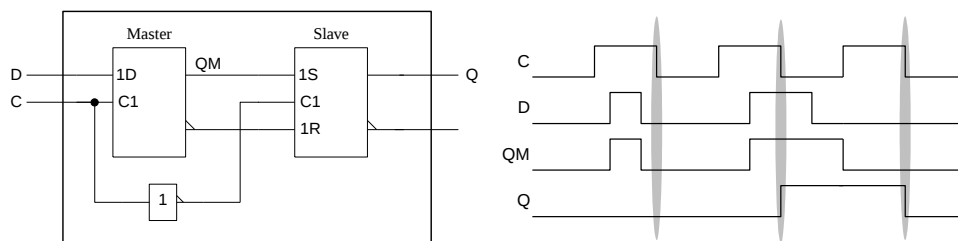
- Problema:** se durante o pulso de relógio $R = 0$ e $S = 0 \rightarrow 1 \rightarrow 0$, esperar-se-ia que o flip-flop mantivesse o estado, pois a última ordem é de HOLD. No entanto, o Mestre respondeu à ordem de SET e é essa ordem que é passada ao Escravo.

■ Flip-Flops 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 Edge-Triggered

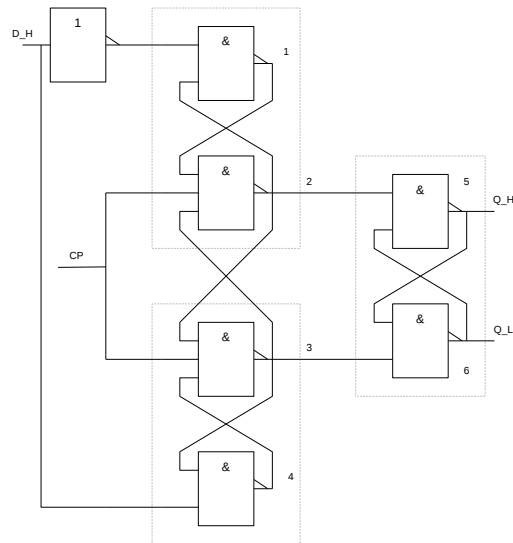


- ▶ Os flip-flops dizem-se **positive-edge-triggered** se reagem à transição de relógio $0 \rightarrow 1$.
- ▶ Os flip-flops dizem-se **negative-edge-triggered** se reagem à transição de relógio $1 \rightarrow 0$.

Flip-Flop D Edge-Triggered

- Os flip-flops D positive-edge-triggered são habitualmente realizados com o circuito ao lado.

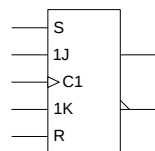
CP	D_H	Q_{n+1}
	L	L
	H	H
	-	Q_n
L	-	Q_n
H	-	Q_n



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”.

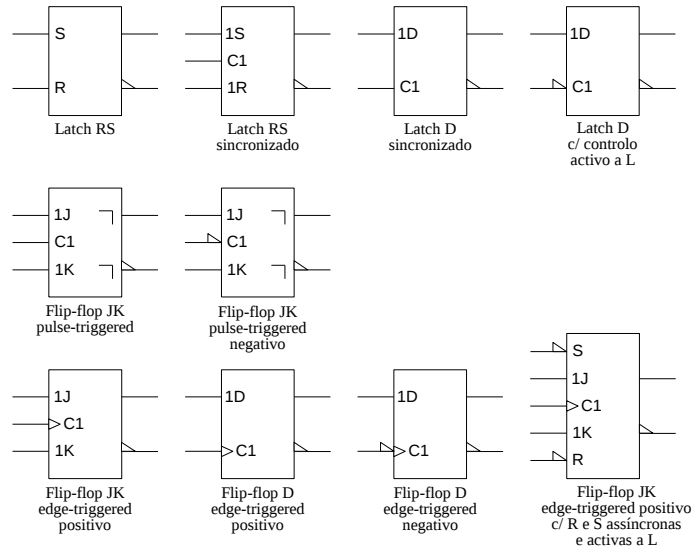
Exemplo:



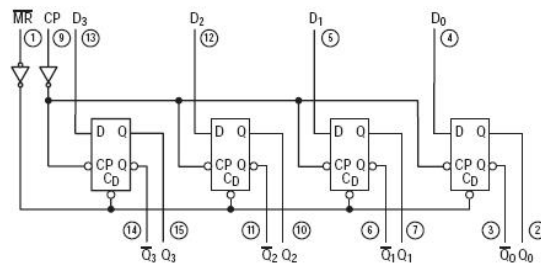
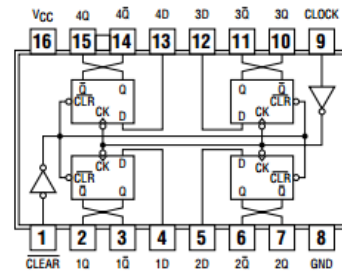
Flip-flop JK com R e S assíncronos

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

■ Simbologia



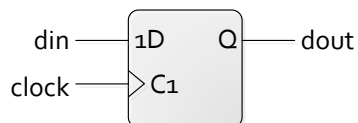
■ Exemplo: 74LS175



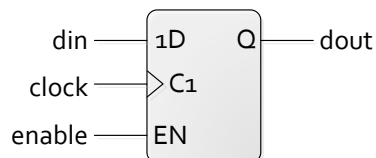
DESCRIÇÃO E SIMULAÇÃO DE CIRCUITOS SEQUENCIAIS EM VHDL

Flip-Flops

■ Flip-Flops



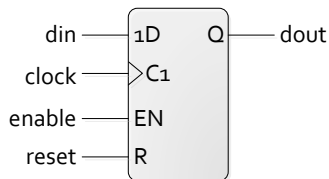
```
dout <= din when rising_edge(clock);
```



```
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**.

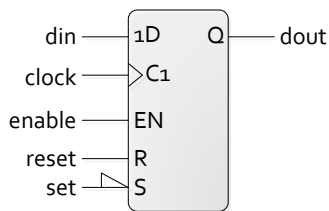
Embora seja possível a utilização em outros casos, tal não será permitido em SD.



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

CONSTRUÇÕES POSSÍVEIS:

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

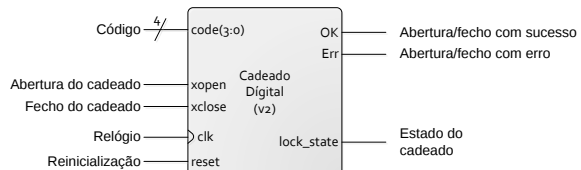


```
dout <= '0' when reset='1' else
      '1' when set='0' else
      din when rising_edge(clock) and enable='1';
```

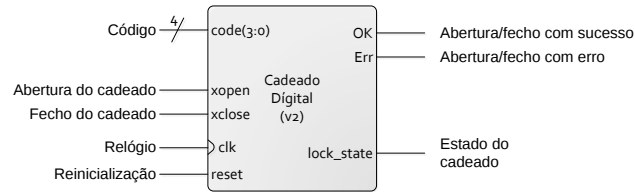
Única situação onde se aceitam os operadores **and/or/xor/...** após o operador **when...**

■ 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.



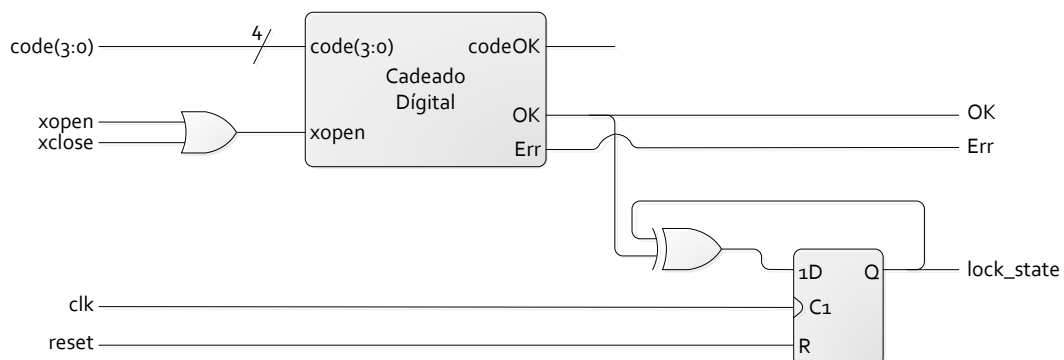
1. Descrição da entidade



```
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;
```

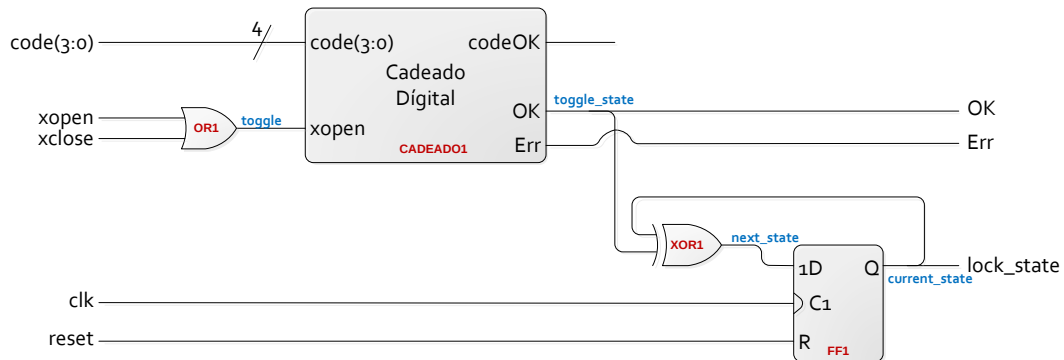
2. Desenho do logograma do cadeado digital

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.



2. Desenho do logograma do cadeado digital

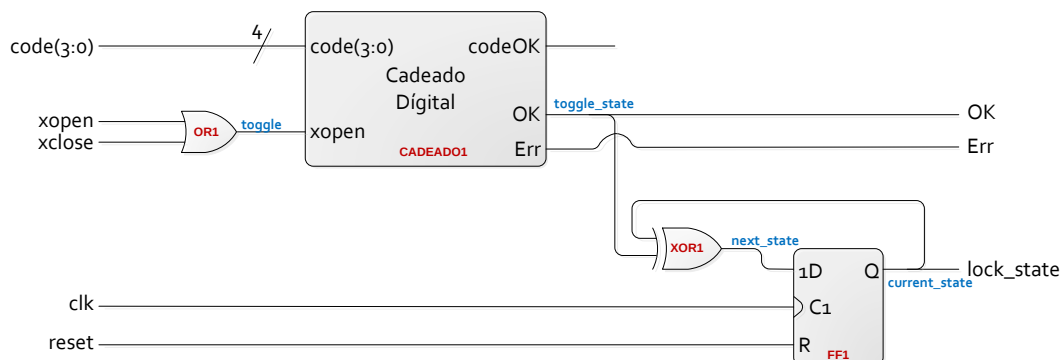
Nomeação dos sinais utilizados internamente (**azul**) e do nome dos circuitos (**vermelho**)



3. Implementação da arquitectura

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

- Declarar o componente
- Utilizar uma instância do componente



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;
```

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

3. Implementação da arquitectura

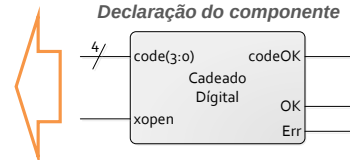
Para utilizar o componente “cadeado_digital”, descrito no ficheiro “cadeado_digital.vhd”, devem ser tomados os seguintes passos:

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
```

Declaração do componente



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
```

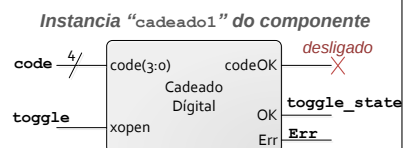
3. Implementação da arquitectura

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

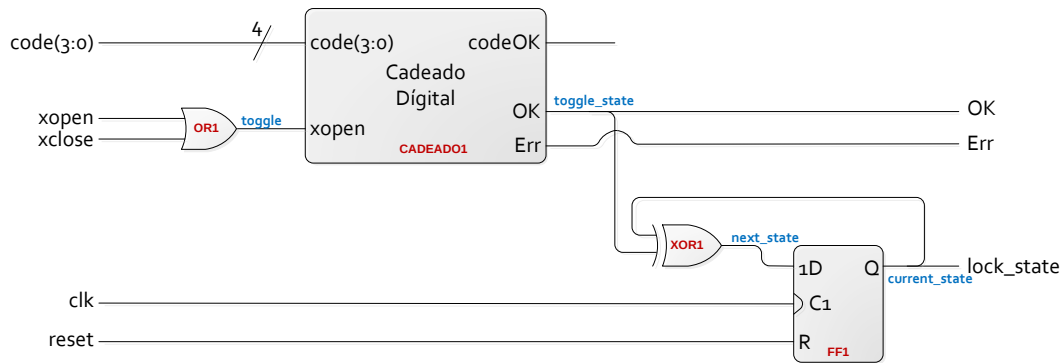
A. Declarar o componente

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
);
...
```



3. Implementação da arquitectura

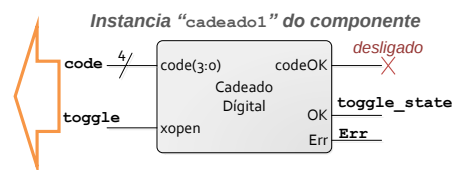


3. Implementação da arquitectura

```

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;

```



```
-- 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 saídas
    lock_state <= current_state;
    OK <= toggle_state;

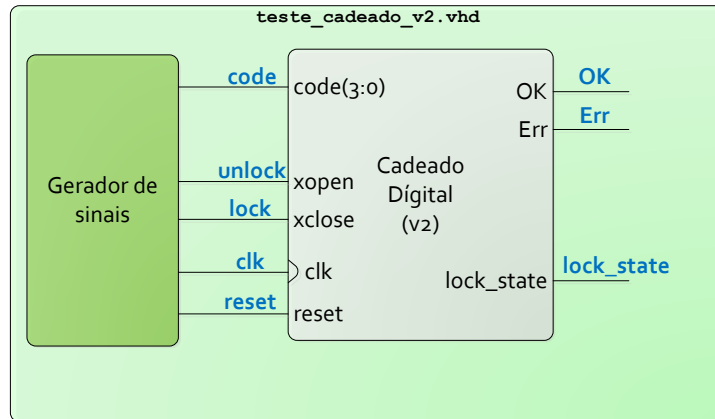
end behavior;
```

- 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.

■ Componente para teste:



■ 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
...
```

■ 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
```

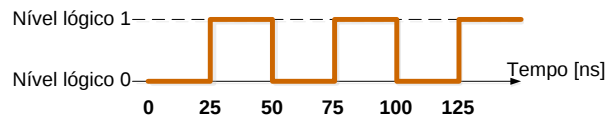
■ Descrição da unidade para teste

```
...
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;
```

■ 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



■ 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;
...
```

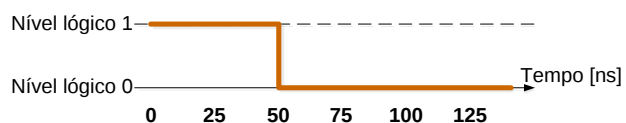
■ Geração do sinal code

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

■ 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





```
-- 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*25 ns;
end process;

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

end behav;
```



■ 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