Sistemas Reconfiguráveis
Eng. de Computação

Profs. Francisco Garcia e Antônio Hamilton Magalhães

Aula 8 - Código sequencial em VHDL

rev. D – 18/10/23

Códigos em VHDL

Vimos anteriormente que existem dois tipos de código em VHDL:

- Código concorrente (paralelo)
- Código sequencial

Até aqui estudamos o código concorrente.

A partir dessa aula estudaremos o **código sequencial**

Código sequencial

- A linguagem VHDL é inerentemente concorrente (paralela).
- Dessa forma, o código sequencial deve estar dentro de uma estrutura chamada PROCESS.
- O PROCESS é uma estrutura do código concorrente. Assim, se em uma ARCHITECTURE existirem mais de um PROCESS, eles serão concorrentes entre si.
- O código sequencial pode ser usado para descrever tanto circuitos de lógica combinacional quanto de lógica sequencial.

Obs. O código sequencial é usado também dentro de FUNCTION e de PROCEDURE.
Essas estruturas poderão ser estudadas posteriormente.

PROCESS

- Sintaxe:

```
[label:] PROCESS (sensitivity_list)
    [process_declarative_part]
BEGIN
    sequential_statement_part
END PROCESS [label];
```

VARIABLES são declaradas aqui

O PROCESS será “executado” toda vez que algum desses sinais mudar de valor.

Deverão estar listados todos os sinais que, ao mudarem de valor, possam alterar alguma saída do PROCESS.

Para lógica combinacional, isso quer dizer que **todos** os sinais que são entrada no PROCESS deverão ser listados.

VARIABLE

- Sintaxe:

```
[label:] PROCESS (sensitivity_list)
    [VARIABLE variable1_name: type [:= default_value];]
    [VARIABLE variable2_name: type [:= default_value];]
    . . .
BEGIN
    . . . . .
```

Uso direto dos operadores

- Assim como no código concorrente, no código sequencial é possível expressões com o uso direto dos operadores.
- Apesar da sintaxe ser a mesma, a semântica é diferente:

Código concorrente

```
....
    x <= a OR b;
    x <= a AND b;
....
```

Erro por múltipla atribuição

Código sequencial (dentro de um PROCESS)

```
....
    x <= a OR b;
    x <= a AND b;
....
```

O código é **avaliado** sequencialmente, de cima para baixo. Assim, permanece apenas a última atribuição.

Construção IF/THEN/ELSE

- Sintaxe:

```
[label:] IF condition1 THEN
        expressions;
ELSIF condition2 THEN
        expressions;
ELSE
        expressions;
END IF;
```

verdadeiro / falso

É possível aninhamento com outras construções

Pode-se ter inúmeros ELSIF, inclusive nenhum

Pode-se ter ou não ter ELSE

Obs.: Não existe WHEN/ELSE no código sequencial

Construção IF/THEN/ELSE

Exemplos:

```
.....
    PROCESS (a, b, c, d, e)
    BEGIN
       IF a = "00" THEN
          x <= c AND d;
          y <= 9;
       ELSIF b = "11" THEN
          x <= c OR d;
          y <= e;
       ELSE
          x <= '0';
          y <= 1;
       END IF;
    END PROCESS;
.....
```

```
.....
    PROCESS (a, b, c, d, e)
    BEGIN
       IF a = "00" THEN
          x <= c AND d;
          y <= 9;
       ELSE
          IF b = "11" THEN
               x <= c OR d;
               y <= e;
          ELSE
               x <= '0';
               y <= 1;
          END IF;
       END IF;
    END PROCESS;
.....
```

Os códigos são equivalentes

Construção IF/THEN/ELSE

Exemplos:

```
.....
    PROCESS (a, b, c, d, e)
    BEGIN
       IF a = "00" THEN
          x <= c AND d;
          y <= 9;
       ELSIF b = "11" THEN
          x <= c OR d;
          y <= e;
       ELSE
          x <= '0';
          y <= 1;
       END IF;
    END PROCESS;
.....
```

```
.....
    PROCESS (a, b, c, d, e)
    BEGIN
       x <= '0';
       y <= 1;
       IF a = "00" THEN
          x <= c AND d;
          y <= 9;
       ELSIF b = "11" THEN
          x <= c OR d;
          y <= e;
       END IF;
    END PROCESS;
.....
```

Os códigos são equivalentes. A segunda construção deve ser usada com cuidado

Construção CASE/WHEN

- Sintaxe:

```
[label:] CASE identifier IS
      WHEN value1 =>
          expressions;
      WHEN value2 | value3 =>
          expressions;
      WHEN value4 TO value7 =>
          expressions;
      WHEN OTHERS =>
          expressions;
END CASE;
```

Todos os valores possíveis do ***identifier*** devem ser testados

A "|" significa OR

Faixa de valores

Quando usado, deve vir por último

Obs.: Não existe WITH/SELECT/WHEN no código sequencial

Construção CASE/WHEN

Exemplos:

```
.....
    PROCESS (a, b, c, d)
    BEGIN
       CASE a IS
          WHEN "000" =>
               x <= b AND c;
               y <= 9;
          WHEN "001" TO "101" =>
               x <= b OR c;
               y <= d;
          WHEN OTHERS =>
               x <= '0';
               y <= 1;
       END CASE;
    END PROCESS;
.....
```

Construção CASE/WHEN

Exemplos:

```
.....
    PROCESS (a, b, c, d)
    BEGIN
       CASE a IS
          WHEN "000" =>
              IF b = '0' THEN
                   x <= c AND d;
              ELSE
                   x <= '1';
              END IF;
          WHEN "001" TO "101" =>
              x <= c OR d;
          WHEN OTHERS =>
              x <= '0';
       END CASE;
    END PROCESS;
.....
```

Exemplo de IF dentro de CASE

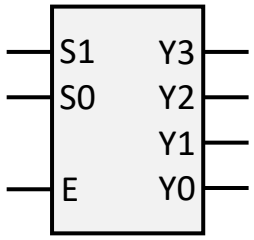
Exemplos
lógica combinacional e *latch*

A seguir vamos ver alguns exemplos de circuito de lógica combinacional e um *latch* descritos com código sequencial de VHDL:

- Decodificador
- Codificador com prioridade
- Multiplexador
- *Latch*

Exemplo dec_2x4

Exemplo dec_2x4: decodificador duas entradas de seleção, quatro saídas ativas em nível alto e enable ativo em nível alto



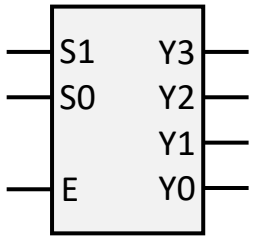
Entradas			Saídas			
E	**S1**	**S0**	**Y3**	**Y2**	**Y1**	**Y0**
0	X	X	0	0	0	0
1	0	0	0	0	0	**1**
1	0	1	0	0	**1**	0
1	1	0	0	**1**	0	0
1	1	1	**1**	0	0	0

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dec_2x4 IS
    PORT (
        s: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        e: IN STD_LOGIC;
        y: OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );
END ENTITY;
```

Continua na próxima página

Exemplo dec_2x4

Exemplo dec_2x4: decodificador duas entradas de seleção, quatro saídas ativas em nível alto e enable ativo em nível alto

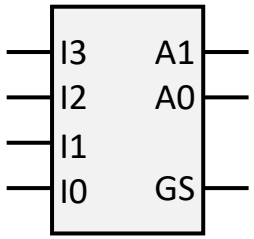


Entradas			Saídas			
E	**S1**	**S0**	**Y3**	**Y2**	**Y1**	**Y0**
0	X	X	0	0	0	0
1	0	0	0	0	0	**1**
1	0	1	0	0	**1**	0
1	1	0	0	**1**	0	0
1	1	1	**1**	0	0	0

```
ARCHITECTURE arch1 OF dec_2x4 IS
BEGIN
    PROCESS(e, s)
    BEGIN
        IF e = '0' THEN
            y <= "0000";
        ELSE
            CASE s IS
                WHEN "00" => y <= "0001";
                WHEN "01" => y <= "0010";
                WHEN "10" => y <= "0100";
                WHEN "11" => y <= "1000";
            END CASE;
        END IF;
    END PROCESS;
END arch1;
```

Exemplo enc_4x2

Exemplo enc_4x2: codificador com quatro entradas ativas em nível alto e duas saídas de código



+ prior. -

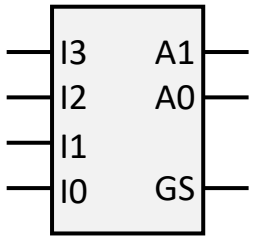
Entradas				Saídas		
I3	**I2**	**I1**	**I0**	**A1**	**A0**	**GS**
0	0	0	0	0	0	0
0	0	0	**1**	0	0	1
0	0	**1**	X	0	1	1
0	**1**	X	X	1	0	1
1	X	X	X	1	1	1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY enc_4x2 IS
    PORT (
        i: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        gs: OUT STD_LOGIC;
        a: OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
    );
END ENTITY;
.......
```

Continua na próxima página

Exemplo enc_4x2

Exemplo enc_4x2: codificador com quatro entradas ativas em nível alto e duas saídas de código



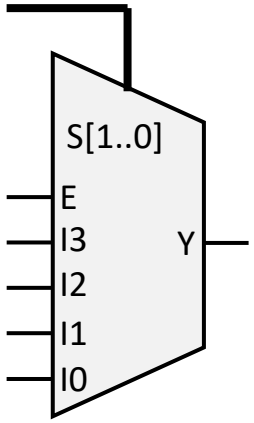
+ prior. -

Entradas				Saídas		
I3	I2	I1	I0	A1	A0	GS
0	0	0	0	0	0	0
0	0	0	**1**	0	0	1
0	0	**1**	X	0	1	1
0	**1**	X	X	1	0	1
1	X	X	X	1	1	1

```
ARCHITECTURE arch OF enc_4x2 IS
BEGIN
     PROCESS(i)
     BEGIN
          IF i(3) = '1' THEN    -- i(3) tem mais prior.
               a <=  "11";
          ELSIF i(2) = '1' THEN
               a <= "10";
          ELSIF i(1) = '1' THEN
               a <= "01";
          ELSE
               a <= "00";
          END IF;

          IF i = "0000" THEN   -- nenhuma entrada ativada
               gs <='0'
          ELSE                 -- alguma entrada ativada
               gs <= '1';
          END IF;
     END PROCESS;
END arch;
```

Exemplo mux_4x1e

Exemplo mux_4x1e: multiplexador com quatro entradas de dados para uma saída, com *enable* ativo alto



A saída vai para '0' quando desabilitado

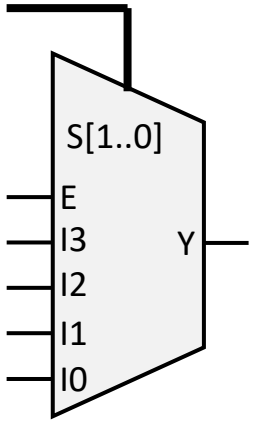
Entradas							Saída
E	S1	S0	I3	I2	I1	I0	Y
0	X	X	X	X	X	X	**0**
1	0	0	X	X	X	**0**	0
1	0	0	X	X	X	**1**	1
1	0	1	X	X	**0**	X	0
1	0	1	X	X	**1**	X	1
1	1	0	X	**0**	X	X	0
1	1	0	X	**1**	X	X	1
1	1	1	**0**	X	X	X	0
1	1	1	**1**	X	X	X	1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux_4x1e IS
    PORT (
        i: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        s: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        e: IN STD_LOGIC;
        y: OUT STD_LOGIC
    );
END ENTITY;
.......
```

Continua na próxima página

Exemplo mux_4x1e

Exemplo mux_4x1e: multiplexador com quatro entradas de dados para uma saída, com *enable* ativo alto



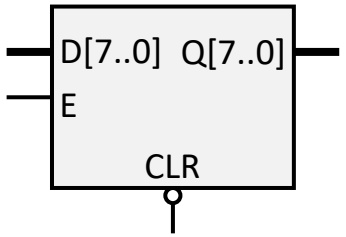
A saída vai para '0' quando desabilitado

Entradas							Saída
E	S1	S0	I3	I2	I1	I0	Y
0	X	X	X	X	X	X	**0**
1	0	0	X	X	X	**0**	0
1	0	0	X	X	X	**1**	1
1	0	1	X	X	**0**	0	0
1	0	1	X	X	**1**	1	1
1	1	0	X	**0**	X	0	0
1	1	0	X	**1**	X	1	1
1	1	1	**0**	X	X	0	0
1	1	1	**1**	X	X	1	1

```
ARCHITECTURE arch OF mux_4x1e IS
BEGIN
    PROCESS(i, e, s)
    BEGIN
        IF e = '0' THEN      -- disabled
            y <= '0';
        ELSE                 -- enabled
            CASE s IS
                WHEN "00" => y <= i(0);
                WHEN "01" => y <= i(1);
                WHEN "10" => y <= i(2);
                WHEN "11" => y <= i(3);
            END CASE;
        END IF;
    END PROCESS;
END arch;
```

Exemplo lat_d8

Exemplo lat_d: *latch* D com 8 bits de dados com entrada de *enable* ativa em nível lógico alto e entrada de *clear* ativa em nível lógico baixo. A entrada de *clear* tem preferência sobre as demais.



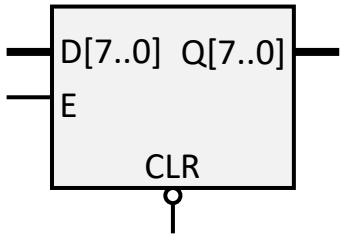
Entradas			Saída
CLR	E	Dn	Qn
0	X	X	0
1	0	X	Qa
1	1	0	0
1	1	1	1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY lat_d8 IS
    PORT (
        d: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        e, nclr: IN STD_LOGIC;
        q: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END ENTITY;
.......
```

Continua na próxima página

Exemplo lat_d8

Exemplo lat_d: *latch* D com 8 bits de dados com entrada de *enable* ativa em nível lógico alto e entrada de *clear* ativa em nível lógico baixo. A entrada de *clear* tem preferência sobre as demais.



Entradas			Saída
CLR	E	Dn	Qn
0	X	X	0
1	0	X	Qa
1	1	0	0
1	1	1	1

```
ARCHITECTURE arch OF lat_d8 IS
BEGIN
     PROCESS(e, d, nclr)
     BEGIN
          IF nclr = '0' THEN
               q <= (OTHERS => '0');
          ELSIF e = '1' THEN
               q <= d;
          END IF;
     END PROCESS;
END arch;
```

Maior prioridade, testa primeiro

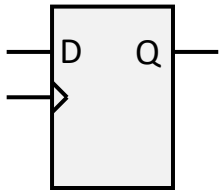
Não tem ELSE, permanece como está

Lógica sequencial
Flip-flops

- Um **flip-flop** é uma unidade de memória
- A diferença entre um ***latch*** e um **flip-flop** é que, nesse último, o sinal de controle, normalmente chamado de ***clock***, atua apenas quando sofre uma transição de estado.
- Existem flip-flops que são disparados pela transição de ‘0’ para ‘1’ do *clock* (chamada de **borda de subida**) enquanto outros são disparados pela transição de ‘1’ para ‘0’, (chamada **borda de descida**).
- Enquanto a entrada *clock* permanece em ‘0’ ou em ‘1’, ou nas transições ao contrário daquela ao qual o flip-flop é sensível, não há mudança na saída do flip-flop.
- Um flip-flop pode ter entradas que mudam o estado da saída independentemente da entrada *clock*. São chamadas **entradas assíncronas**:
  - *Clear*: leva a saída para ‘0’
  - *Preset*: leva a saída para ‘1’

Exemplo d_ff

Exemplo d_ff: flip-flop tipo D disparado pela borda de **subida** de *clock*



Entradas		Saída
D	CLK	Q
0	↑	0
1	↑	1
X	0	Qa
X	1	Qa
X	↓	Qa

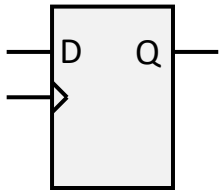
Após a transição de subida de **CLK**, a saída passa a ser o valor da entrada **D** no instante da transição de **CLK**

Enquando **CLK** permanece em ‘0’, em ‘1’ ou na transição de descida, a saída permanece como está.

Muitas vezes essas três linhas não são mostradas na tabela

Exemplo d_ff

Exemplo d_ff: flip-flop tipo D disparado pela borda de subida de *clock*



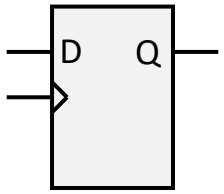
Entradas		Saída
D	CLK	Q
0	↑	0
1	↑	1
X	0	Qa
X	1	Qa
X	↓	Qa

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_ff IS
    PORT (
        clk, d : IN STD_LOGIC;
        q : OUT STD_LOGIC
    );
END ENTITY;
.......
```

Continua na próxima página

Exemplo d_ff

Exemplo d_ff: flip-flop tipo D disparado pela borda de subida de *clock*



Entradas		Saída
D	CLK	Q
0	↑	0
1	↑	1
X	0	Qa
X	1	Qa
X	↓	Qa

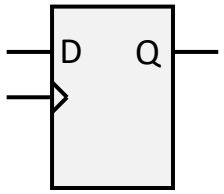
```
ARCHITECTURE arch1 OF d_ff IS
BEGIN

    PROCESS(clk)
    BEGIN
        IF clk'EVENT AND clk = '1' THEN
            q <= d;
        END IF;
    END PROCESS;
END arch1;
```

Não é necessário incluir a entrada **d** na lista de sensibilidade

O apóstrofo indica um **atributo**. O atributo EVENT torna-se TRUE quando há uma variação (‘0’ → ‘1’ ou ‘1’ → ‘0’) no sinal associado. Pode ser usado com BIT ou STD_LOGIC

Exemplo d_ff

Exemplo d_ff: flip-flop tipo D disparado pela borda de subida de *clock*



Entradas		Saída
D	CLK	Q
0	↑	0
1	↑	1
X	0	Qa
X	1	Qa
X	↓	Qa

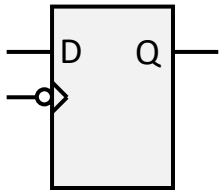
```
ARCHITECTURE arch2 OF d_ff IS
BEGIN

    PROCESS(clk)
    BEGIN
        IF RISING_EDGE(clk) THEN
            q <= d;
        END IF;
    END PROCESS;
END arch1;
```

Não é necessário incluir a entrada **d** na lista de sensibilidade

Essa função necessita da biblioteca std_logic_1164
Não funciona com o tipo BIT

Exemplo d_ff
Borda de descida

Exemplo d_ff: flip-flop tipo D disparado pela borda de **descida** de *clock*



Entradas		Saída
D	CLK	Q
0	↓	0
1	↓	1
X	0	Qa
X	1	Qa
X	↑	Qa

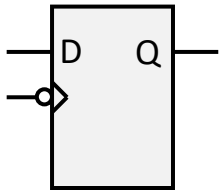
Após a transição de descida de **CLK**, a saída passa a ser o valor da entrada **D** no instante da transição de **CLK**

Enquando **CLK** permanece em ‘0’, em ‘1’ ou nas transições de subida, a saída permanece como está.

Muitas vezes essas três linhas não são mostradas na tabela

Exemplo d_ff
Borda de descida

Exemplo d_ff: flip-flop tipo D disparado pela borda de descida de *clock*



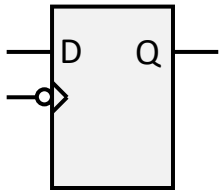
Entradas		Saída
D	CLK	Q
0	↓	0
1	↓	1
X	0	Qa
X	1	Qa
X	↑	Qa

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_ff IS
    PORT (
        clk, d : IN STD_LOGIC;
        q : OUT STD_LOGIC
    );
END ENTITY;
.......
```

Continua na próxima página

Exemplo d_ff
Borda de descida

Exemplo d_ff: flip-flop tipo D disparado pela borda de descida de *clock*



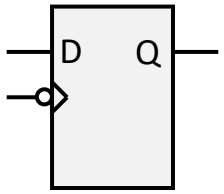
Entradas		Saída
D	CLK	Q
0	↓	0
1	↓	1
X	0	Qa
X	1	Qa
X	↑	Qa

```
ARCHITECTURE arch1 OF d_ff IS
BEGIN

     PROCESS(clk)
     BEGIN
          IF clk'EVENT AND clk = '0' THEN
               q <= d;
          END IF;
     END PROCESS;
END arch1;
```

Não é necessário incluir a entrada **d** na lista de sensibilidade

Exemplo d_ff
Borda de descida

Exemplo d_ff: flip-flop tipo D disparado pela borda de descida de *clock*



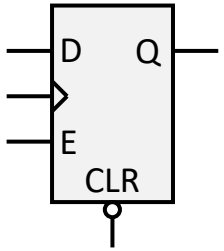
Entradas		Saída
D	CLK	Q
0	↓	0
1	↓	1
X	0	Qa
X	1	Qa
X	↑	Qa

```
ARCHITECTURE arch2 OF d_ff IS
BEGIN

    PROCESS(clk)
    BEGIN
        IF FALLING_EDGE(clk) THEN
            q <= d;
        END IF;
    END PROCESS;
END arch1;
```

Não é necessário incluir a entrada **d** na lista de sensibilidade

Exemplo d_ffe

Exemplo d_ffe: flip-flop tipo D disparado pela borda de subida de *clock*, com *enable* ativo em nível lógico alto e *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



Entradas				Saída
$\overline{CLR}$	E	D	CLK	Q
0	X	X	X	0
1	0	X	↑	Qa
1	1	0	↑	0
1	1	1	↑	1

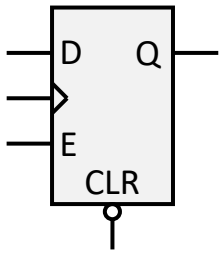
Clear assíncrono. Não depende do *clock*

E = '0' ⇒ desabilitado. A saída não muda, mesmo que haja transição de *clock*

Habilitado.

Exemplo d_ffe

Exemplo d_ffe: flip-flop tipo D disparado pela borda de subida de *clock*, com *enable* ativo em nível lógico alto e *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



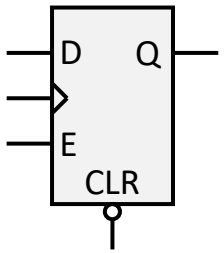
Entradas				Saída
$\overline{CLR}$	E	D	CLK	Q
0	X	X	X	0
1	0	X	↑	Qa
1	1	0	↑	0
1	1	1	↑	1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_ffe IS
    PORT (
        nclr, clk, d, e : IN STD_LOGIC;
        q : OUT STD_LOGIC
    );
END ENTITY;
.......
```

Continua na próxima página

Exemplo d_ffe

Exemplo d_ffe: flip-flop tipo D disparado pela borda de subida de *clock*, com *enable* ativo em nível lógico alto e *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



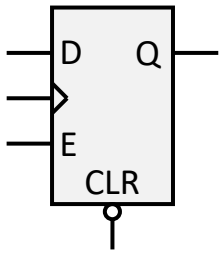
Entradas				Saída
$\overline{CLR}$	E	D	CLK	Q
0	X	X	X	0
1	0	X	↑	Qa
1	1	0	↑	0
1	1	1	↑	1

```
ARCHITECTURE arch1 OF d_ffe IS
BEGIN
    PROCESS(nclr, clk)
    BEGIN
        IF nclr = '0' THEN
            q <= '0';
        ELSIF RISING_EDGE(clk) THEN
            IF e = '1' THEN
                q <= d;
            END IF;
        END IF;
    END PROCESS;
END arch1;
```

É necessário incluir a entrada **nclr**

Exemplo d_ffe

Exemplo d_ffe: flip-flop tipo D disparado pela borda de subida de *clock*, com *enable* ativo em nível lógico alto e *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



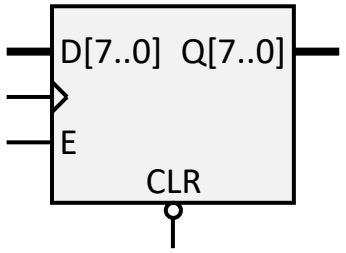
Entradas				Saída
$\overline{CLR}$	E	D	CLK	Q
0	X	X	X	0
1	0	X	↑	Qa
1	1	0	↑	0
1	1	1	↑	1

```
ARCHITECTURE arch2 OF d_ffe IS
BEGIN
    PROCESS(nclr, clk, e)
    BEGIN
        IF nclr = '0' THEN
            q <= '0';
        ELSIF RISING_EDGE(clk) AND e = '1' THEN
            q <= d;
        END IF;
    END PROCESS;
END arch2;
```

Nesse caso, a entrada **e** tem de estar na lista de sensibilidade

Exemplo reg8

Exemplo reg8: 8 flip-flops tipo D disparados pela borda de subida de *clock*, formando um **registrador de 8 bits**, com *enable* ativo em nível lógico alto e *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



Entradas				Saída
$\overline{CLR}$	E	Dn	CLK	Qn
0	X	X	X	0
1	0	X	↑	Qna
1	1	0	↑	0
1	1	1	↑	1

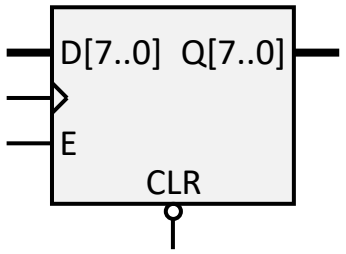
Clear assíncrono. Não depende do *clock*

E = '0' ⍰ desabilitado. A saída não muda

Habilitado.

Exemplo reg8

Exemplo reg8: 8 flip-flop tipo D bits disparados pela borda de subida de *clock*, formando um **registrador de 8 bits**, com *enable* ativo em nível lógico alto e *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



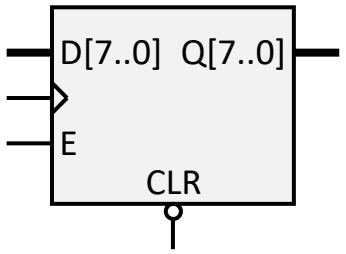
Entradas				Saída
$\overline{CLR}$	E	Dn	CLK	Qn
0	X	X	X	0
1	0	X	↑	Qna
1	1	0	↑	0
1	1	1	↑	1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reg8 IS
    PORT (
        nclr, clk, e : IN STD_LOGIC;
        d: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END ENTITY;
.......
```

Continua na próxima página

Exemplo reg8

Exemplo reg8: 8 flip-flop tipo D bits disparados pela borda de subida de *clock*, formando um **registrador de 8 bits**, com *enable* ativo em nível lógico alto e *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



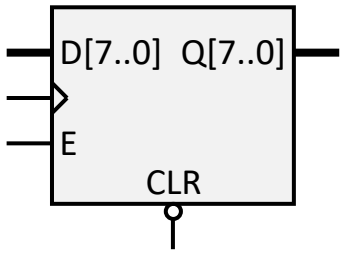
Entradas				Saída
$\overline{CLR}$	E	Dn	CLK	Qn
0	X	X	X	0
1	0	X	↑	Qna
1	1	0	↑	0
1	1	1	↑	1

```
ARCHITECTURE arch1 OF reg8 IS
BEGIN
    PROCESS(nclr, clk)
    BEGIN
        IF nclr = '0' THEN
            q <= (OTHERS => '0');
        ELSIF RISING_EDGE(clk) THEN
            IF e = '1' THEN
                q <= d;
            END IF;
        END IF;
    END PROCESS;
END arch1;
```

Todos os bits do registrador são zerados

Exemplo reg8

Exemplo reg8: 8 flip-flop tipo D bits disparados pela borda de subida de *clock*, formando um **registrador de 8 bits**, com *enable* ativo em nível lógico alto e *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



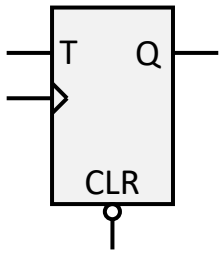
Entradas				Saída
$\overline{CLR}$	E	Dn	CLK	Qn
0	X	X	X	0
1	0	X	↑	Qna
1	1	0	↑	0
1	1	1	↑	1

```
ARCHITECTURE arch2 OF reg8 IS
BEGIN
    PROCESS(nclr, clk, e)
    BEGIN
        IF nclr = '0' THEN
            q <= (OTHERS => '0');
        ELSIF RISING_EDGE(clk) AND e = '1' THEN
            q <= d;
        END IF;
    END PROCESS;
END arch2;
```

É necessário acrescentar a entrada **e**

Exemplo t_ff

Exemplo t_ff: flip-flop tipo T (*toggle*) disparado pela borda de subida de *clock* com *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



Entradas			Saída
$\overline{CLR}$	T	CLK	Q
0	X	X	0
1	0	↑	Qa
1	1	↑	$\overline{Qa}$

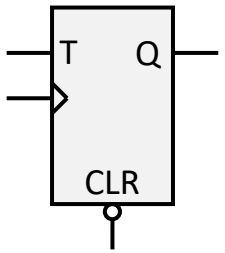
Clear assíncrono.
Não depende do *clock*

T = ‘0’.
A saída não muda

T = ‘1’.
A saída inverte (*toggle*)

Exemplo t_ff

Exemplo t_ff: flip-flop tipo T (*toggle*) disparado pela borda de subida de *clock* com *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



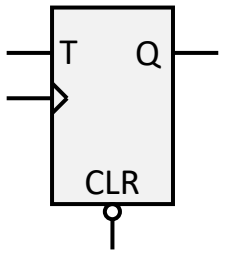
Entradas			Saída
$\overline{CLR}$	T	CLK	Q
0	X	X	0
1	0	↑	Qa
1	1	↑	$\overline{Qa}$

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY t_ff IS
    PORT (
        nclr, clk, t : IN STD_LOGIC;
        q : OUT STD_LOGIC
    );
END ENTITY;
.......
```

Continua na próxima página

Exemplo t_ff

Exemplo t_ff: flip-flop tipo T (*toggle*) disparado pela borda de subida de *clock* com *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



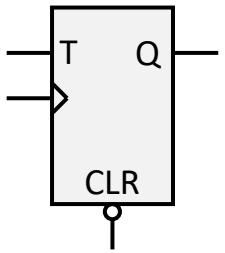
Entradas			Saída
$\overline{CLR}$	T	CLK	Q
0	X	X	0
1	0	↑	Qa
1	1	↑	$\overline{Qa}$

```
ARCHITECTURE arch1 OF t_ff IS
BEGIN
    PROCESS(nclr, clk)
        VARIABLE ff : STD_LOGIC;
    BEGIN
        IF nclr = '0' THEN
            ff := '0';
        ELSIF RISING_EDGE(clk) THEN
            IF t = '1' THEN
                ff := NOT ff;
            END IF;
        END IF;
        q <= ff;
    END PROCESS;
END arch1;
```

Como o flip-flop precisa ser lido, não pode ser do modo OUT

Exemplo t_ff

Exemplo t_ff: flip-flop tipo T (*toggle*) disparado pela borda de subida de *clock* com *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



Entradas			Saída
$\overline{CLR}$	T	CLK	Q
0	X	X	0
1	0	↑	Qa
1	1	↑	$\overline{Qa}$

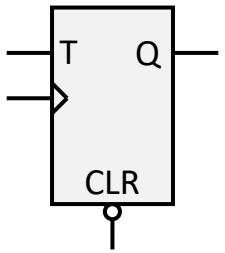
```
ARCHITECTURE arch1 OF t_ff IS
BEGIN
    PROCESS(nclr, clk)
        VARIABLE ff : STD_LOGIC;
    BEGIN
        IF nclr = '0' THEN
            ff := '0';
        ELSIF RISING_EDGE(clk) THEN
            IF t = '1' THEN
                ff := NOT ff;
            END IF;
        END IF;
        q <= ff;
    END PROCESS;
END arch1;
```

Então precisa ser declarado como VARIABLE ou SIGNAL

VARIABLE recebe valor com :=

Exemplo t_ff

Exemplo t_ff: flip-flop tipo T (*toggle*) disparado pela borda de subida de *clock* com *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.



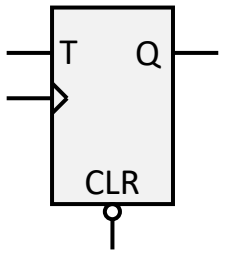
Entradas			Saída
$\overline{CLR}$	T	CLK	Q
0	X	X	0
1	0	↑	Qa
1	1	↑	$\overline{Qa}$

```
ARCHITECTURE arch2 OF t_ff IS
BEGIN
    PROCESS(nclr, clk, t)
        VARIABLE ff : STD_LOGIC;
    BEGIN
        IF nclr = '0' THEN
            ff := '0';
        ELSIF RISING_EDGE(clk) AND t = '1' THEN
            ff := NOT ff;
        END IF;
        q <= ff;
    END PROCESS;
END arch2;
```

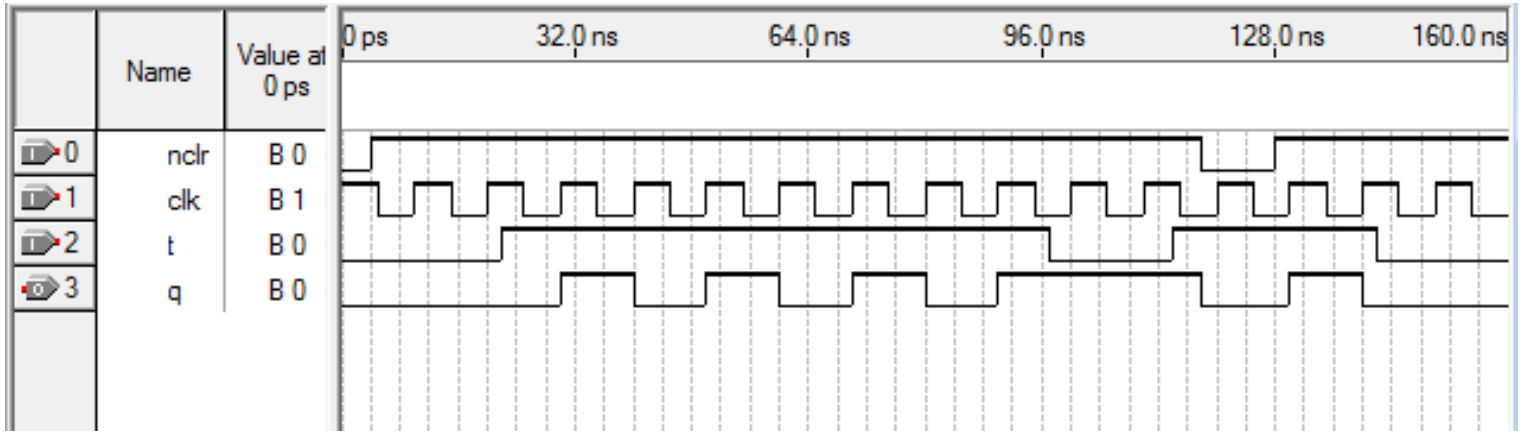
É necessário acrescentar a entrada **t**

Exemplo t_ff

Exemplo t_ff: flip-flop tipo T (*toggle*) disparado pela borda de subida de *clock* com *clear* assíncrono ativo em nível lógico baixo. A entrada de *clear* tem prioridade sobre todas as outras.

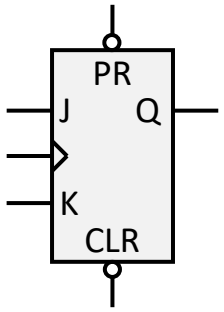


Entradas			Saída
$\overline{CLR}$	T	CLK	Q
0	X	X	0
1	0	↑	Qa
1	1	↑	$\overline{Qa}$



Exemplo jk_ff

Exemplo jk_ff: flip-flop tipo JK disparado pela borda de subida de *clock* com *clear* e *preset* assíncronos ativos em nível lógico baixo. A entrada de *clear* tem prioridade sobre *preset*, que, por sua vez, tem prioridade sobre as demais.



Entradas					Saída
$\overline{CLR}$	$\overline{PR}$	J	K	CLK	Q
0	X	X	X	X	0
1	0	X	X	X	1
1	1	0	0	↑	Qa
1	1	0	1	↑	0
1	1	1	0	↑	1
1	1	1	1	↑	$\overline{Qa}$

Clear assíncrono.

Preset assíncrono.

J = **K** = ‘0’.
A saída não muda (memória)

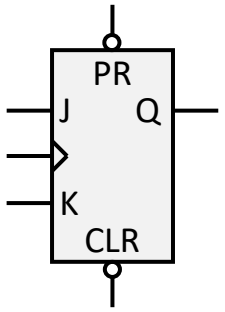
J = ‘0’ e **K** = ‘1’.
Reset síncrono

J = ‘1’ e **K** = ‘0’.
Set síncrono

J = **K** = ‘1’.
A saída inverte (*toggle*)

Exemplo jk_ff

Exemplo jk_ff: flip-flop tipo JK disparado pela borda de subida de *clock* com *clear* e *preset* assíncronos ativos em nível lógico baixo. A entrada de *clear* tem prioridade sobre *preset*, que, por sua vez, tem prioridade sobre as demais.



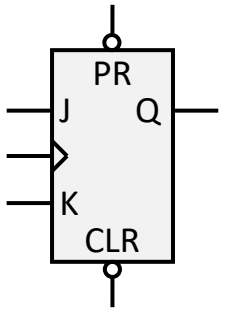
Entradas					Saída
$\overline{CLR}$	$\overline{PR}$	J	K	CLK	Q
0	X	X	X	X	0
1	0	X	X	X	1
1	1	0	0	↑	Qa
1	1	0	1	↑	0
1	1	1	0	↑	1
1	1	1	1	↑	$\overline{Qa}$

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY jk_ff IS
    PORT (
        nclr, npr, clk, j, k : IN STD_LOGIC;
        q : OUT STD_LOGIC
    );
END ENTITY;
.......
```

Continua na próxima página

Exemplo jk_ff

Exemplo jk_ff: flip-flop tipo JK



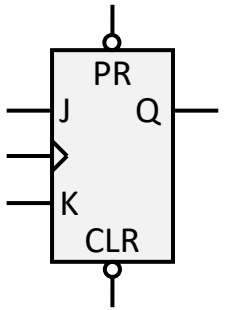
Entradas					Saída
$\overline{CLR}$	$\overline{PR}$	J	K	CLK	Q
0	X	X	X	X	0
1	0	X	X	X	1
1	1	0	0	↑	Qa
1	1	0	1	↑	0
1	1	1	0	↑	1
1	1	1	1	↑	$\overline{Qa}$

```
ARCHITECTURE arch1 OF jk_ff IS
BEGIN
    PROCESS(nclr, npr, clk, j, k)
        VARIABLE ff : STD_LOGIC;
        VARIABLE jk : STD_LOGIC_VECTOR(1 DOWNTO 0);
    BEGIN
        jk := j & k;
        IF nclr = '0' THEN
            ff := '0';
        ELSIF npr = '0' THEN
            ff := '1';
        ELSIF RISING_EDGE(clk) THEN
            CASE jk IS
                WHEN "00" => ff := ff;
                WHEN "01" => ff := '0';
                WHEN "10" => ff := '1';
                WHEN "11" => ff := NOT ff;
            END CASE;
        END IF;
        q <= ff;
    END PROCESS;
END arch1;
```

Todos os valores de **jk** têm de ser testados

Exemplo jk_ff

Exemplo jk_ff: flip-flop tipo JK



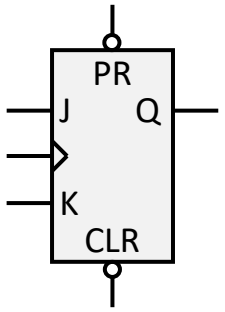
Entradas					Saída
$\overline{CLR}$	$\overline{PR}$	J	K	CLK	Q
0	X	X	X	X	0
1	0	X	X	X	1
1	1	0	0	↑	Qa
1	1	0	1	↑	0
1	1	1	0	↑	1
1	1	1	1	↑	$\overline{Qa}$

```
ARCHITECTURE arch2 OF jk_ff IS
BEGIN
    PROCESS(nclr, npr, clk, j, k)
       VARIABLE ff : STD_LOGIC;
    BEGIN
       IF nclr = '0' THEN
          ff := '0';
       ELSIF npr = '0' THEN
          ff := '1';
       ELSIF RISING_EDGE(clk) THEN
          IF j = '0'  AND k = '1' THEN ff := '0';
          ELSIF j = '1'  AND k = '0' THEN ff := '1';
          ELSIF j = '1'  AND k = '1' THEN ff := NOT ff;
          END IF;
       END IF;
       q <= ff;
    END PROCESS;
END arch2;
```

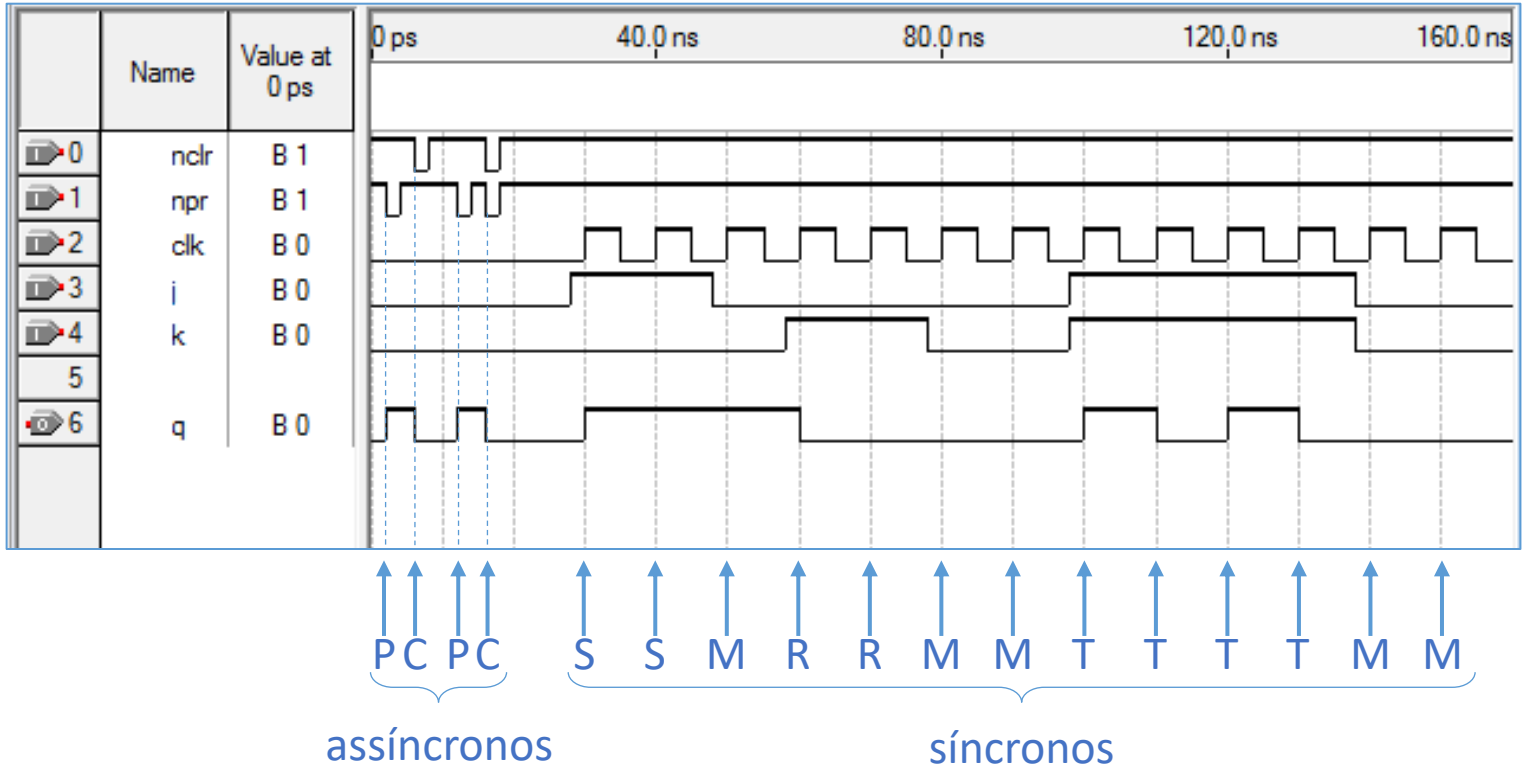
Quando **j** = **k** = '0' (não testado), **ff** permanece como está

Exemplo jk_ff

Exemplo jk_ff: flip-flop tipo JK disparado pela borda de subida de *clock* com *clear* e *preset* assíncronos ativos em nível lógico baixo. A entrada de *clear* tem prioridade sobre *preset*, que, por sua vez, tem prioridade sobre as demais.



Entradas					Saída	
$\overline{CLR}$	$\overline{PR}$	J	K	CLK	Q	
0	X	X	X	X	0	← C
1	0	X	X	X	1	← P
1	1	0	0	↑	Qa	← M
1	1	0	1	↑	0	← R
1	1	1	0	↑	1	← S
1	1	1	1	↑	$\overline{Qa}$	← T



Fim