

Universidade Tecnológica Federal do Paraná – Toledo
Engenharia da Computação – COENC

Lógica Reconfigurável

BLOCK

Tiago Piovesan Vendruscolo



Esta licença permite que outros remixem, adaptem e criem a partir do trabalho para fins não comerciais, desde que atribuam o devido crédito aos autores originais. [4.0 international](https://creativecommons.org/licenses/by-nc-nd/4.0/)

- Há dois tipos de BLOCK: Simples e GUARDED.
- BLOCK simples: Usado para particionar o código gerando variáveis e sinais locais. Segue a sintaxe:

```
Label: BLOCK
    [sinais visíveis (locais) no bloco]
BEGIN
    [comandos]
END BLOCK label;
```

- BLOCK/GUARDED: As operações GUARDED são executadas apenas se a expressão de guarda for verdadeira. Segue a sintaxe:

```
Label: BLOCK expressao_guarda
    [sinais visíveis no bloco]
BEGIN
    sinal_a <= GUARDED expressao_1;
    [comandos guarded e unguarded]
END BLOCK label;
```


Sinal de sensibilidade

BLOCK

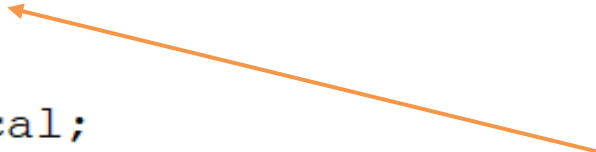
```
ENTITY mux_com_block IS
    PORT(i0, i1, i2: IN BIT;
         sel: IN INTEGER RANGE 3 DOWNT0 0;
         ot: OUT BIT);
END mux_com_block;
```

```
ARCHITECTURE comportamento OF mux_com_block IS
    SIGNAL global: BIT_VECTOR (0 TO 1);
BEGIN
    bloco1: BLOCK
    BEGIN
        (CODIGO...)
    END BLOCK bloco1;

    bloco2: BLOCK
        SIGNAL local: BIT_VECTOR (0 TO 1);
    BEGIN
        (CODIGO...)
        global <= local;
    END BLOCK BLOCO2;
END comportamento;
```



Sinal pode ser utilizado em toda a architecture.



Sinal local pode ser utilizado apenas dentro do bloco2.

- Exemplo: Latch usando BLOCK/GUARDED

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

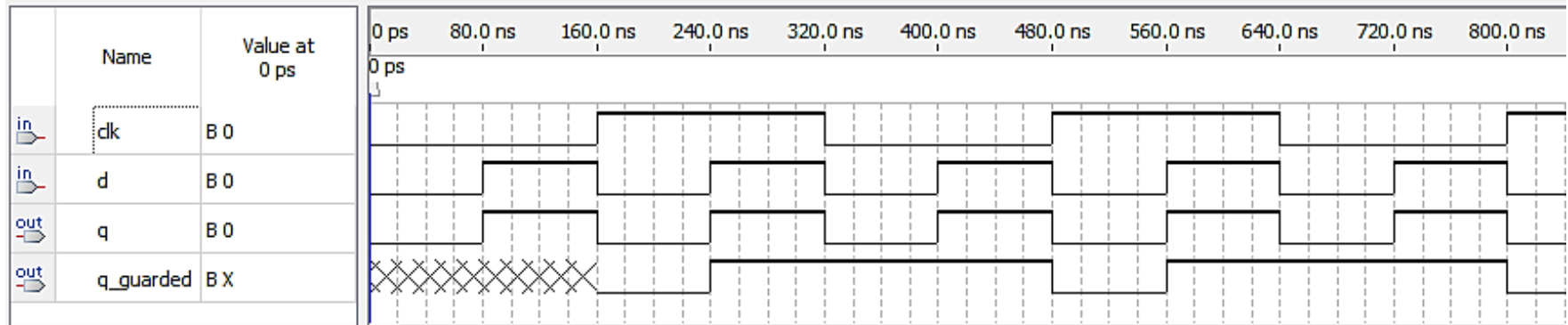
ENTITY latch_guarded IS
    PORT(d, clk: IN STD_LOGIC;
         q_guarded, q: OUT STD_LOGIC);
END latch_guarded;

ARCHITECTURE funcao OF latch_guarded IS
BEGIN
    bloco1: BLOCK (clk='1')
    BEGIN
        q_guarded <= GUARDED d;
        q <= d;
    END BLOCK bloco1;
END funcao;
```

Sinal de sensibilidade, o código irá rodar a partir da primeira mudança e o GUARDED funcionará de acordo com o que for definido.

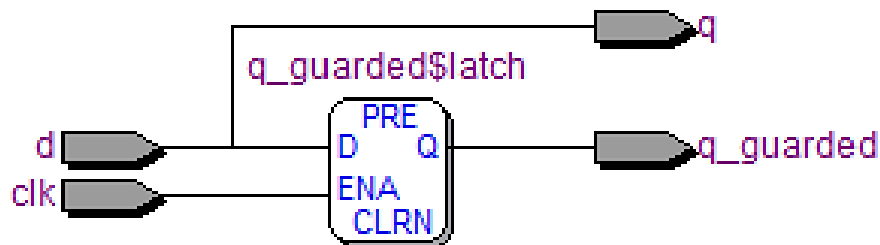
As variáveis "GUARDED" são executadas apenas após o primeiro `clk='1'`, e então passam a responder de acordo com o sinal de sensibilidade. As variáveis "não guarded" são executadas normalmente, independentes do sinal de sensibilidade.

- Exemplo: Latch usando BLOCK/GUARDED



Obs: como a variável de sensibilidade não possui "clk'event", as variáveis guarded respondem de acordo com o nível do clk, e não pela borda.

- RTL VIEWER






- Exercício 1: Faça um flip-flop tipo síncrono usando BLOCK/GUARDED com sinal rst de acordo com a tabela. Também faça uma saída assíncrona (não depende do guarded). (saídas LED [0 e 1], entrada d DIP switch[0], entrada rst DIP switch[1] e clk KEY[0]). Dica: Pode ser utilizado “When-else” com GUARDED.

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

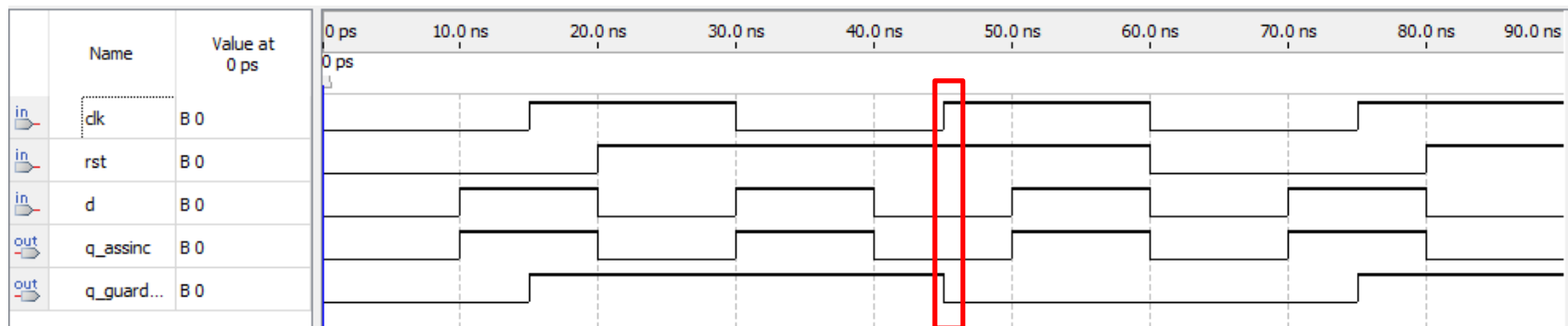
ENTITY latch_guarded_borda IS
    PORT(d, clk, rst: IN STD_LOGIC;
         q_guarded_sinc, q_assinc: OUT STD_LOGIC);
END latch_guarded_borda;

ARCHITECTURE funcao OF latch_guarded_borda IS
BEGIN
    blocol: BLOCK (clk'EVENT AND clk='1')
    BEGIN
        q_guarded_sinc <= GUARDED '0' WHEN rst='1' ELSE d;
        q_assinc <= d;
    END BLOCK blocol;
END funcao;
```

rst	d	clk	q
1	–		0
0	1		1
0	0		0

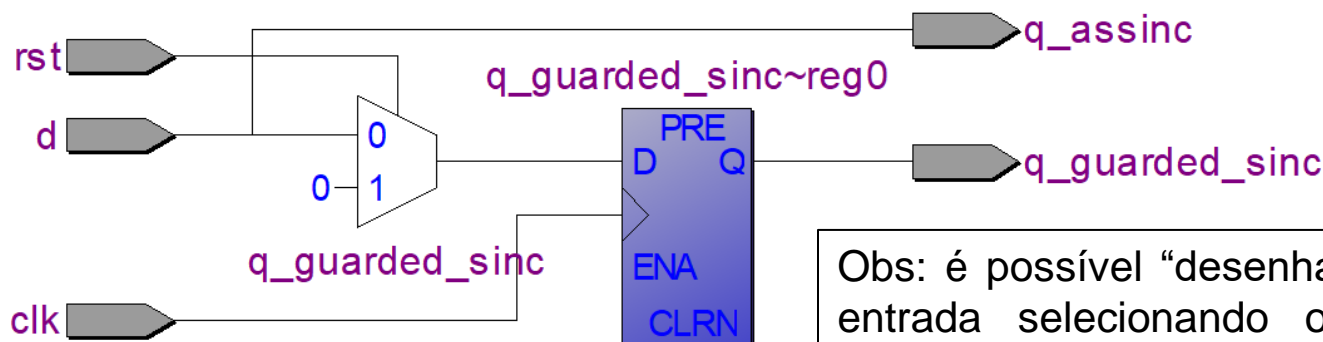
Função de sensibilidade
com detecção de borda

- Exercício 1 : Faça um flip-flop tipo síncrono usando BLOCK/GUARDED com sinal rst de acordo com a tabela. Também faça uma saída assíncrona (não depende do guarded).



Agora a mudança acontece apenas na mudança de borda

- RTL VIEWER



Obs: é possível “desenhar” a onda de entrada selecionando o intervalo e escolhendo ‘1’ ou ‘0’.

- Exercício 2: Adicione o pino rst no primeiro exemplo (LATCH) e compare o comportamento do sinal.

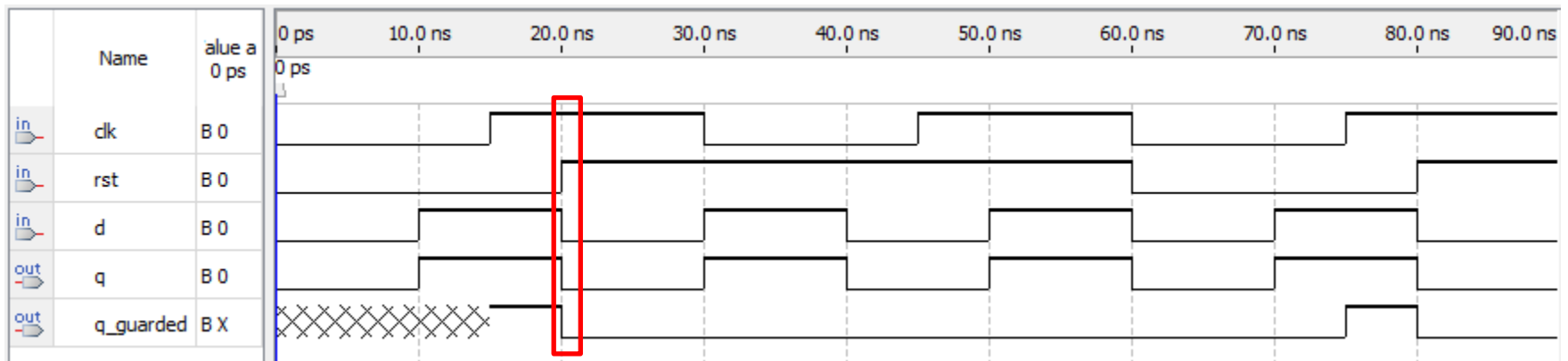
- Exercício 2: Adicione o pino rst no primeiro exemplo (LATCH) e compare o comportamento do sinal.

```

ENTITY latch_guarded_rst IS
    PORT(d, clk, rst: IN STD_LOGIC;
          q_guarded, q: OUT STD_LOGIC);
END latch_guarded_rst;

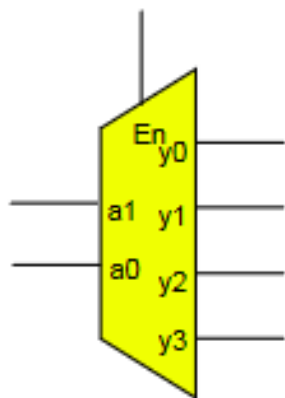
ARCHITECTURE funcao OF latch_guarded_rst IS
BEGIN
    bloco1: BLOCK (clk='1')
    BEGIN
        q_guarded <= GUARDED '0' WHEN rst='1' ELSE d;
        q <= d;
    END BLOCK bloco1;
END funcao;

```



Exercício 3

- Projete e simule o decodificador 2:4 (parecido com o da aula 6) com pino de enable abaixo: (saídas LED [0-3], entradas ax: DIP switch [0-1], entrada en: DIP Switch 2).
- Pode ser feito de 2 formas:
 - *Todo o código dentro da estrutura block.*
 - *Utilizar o block apenas para habilitar a saída.*
 - *Faça das duas formas.*

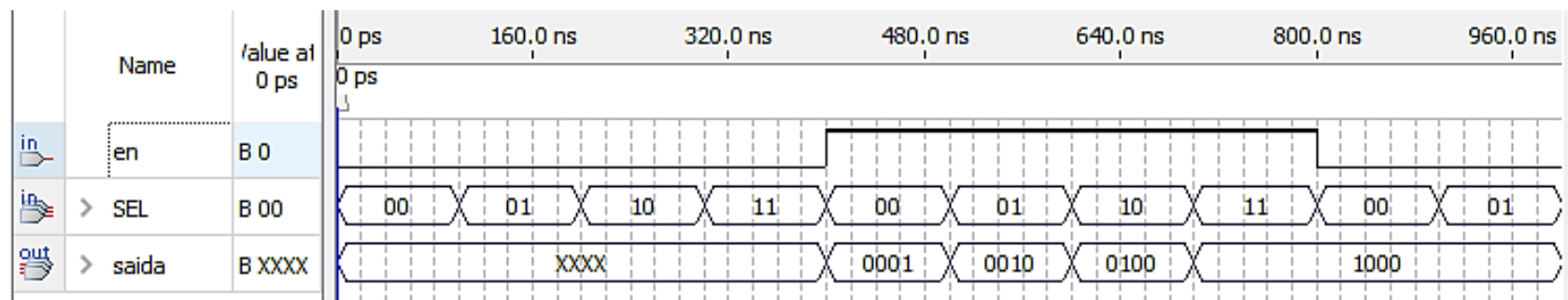


En	a1	a0	y0	y1	y2	y3
0	X	X	X	X	X	X
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

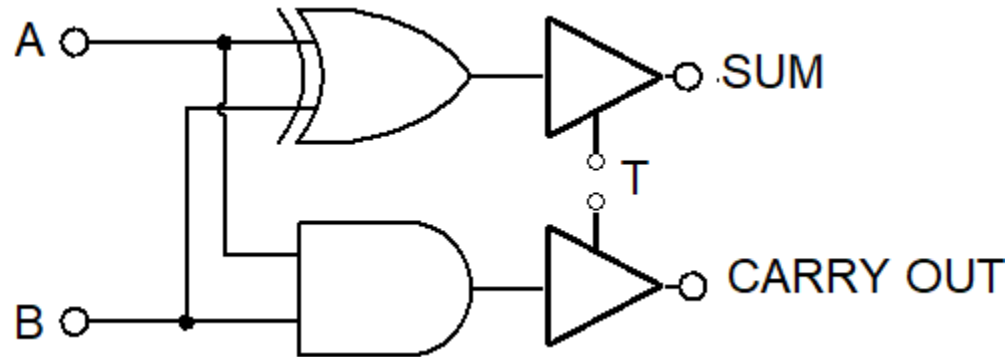
Exercício 3

```
architecture funcao of decod is
signal x: std_logic_vector(3 downto 0);
begin
    block1: BLOCK (en='1')
    BEGIN
        with SEL select
            x <= "0001" when "00",
                "0010" when "01",
                "0100" when "10",
                "1000" when "11";
        saida <= guarded x;
    END BLOCK block1;
end funcao;
```

```
architecture funcao of decod is
signal x: std_logic_vector(3 downto 0);
begin
    with SEL select
        x <= "0001" when "00",
            "0010" when "01",
            "0100" when "10",
            "1000" when "11";
    block1: BLOCK (en='1')
    BEGIN
        saida <= guarded x;
    END BLOCK block1;
end funcao;
```



- Exercício 4: Faça o meio somador com saídas Tristate abaixo, sendo que, quando $T = '1'$, a saída do tristate é igual à entrada, e quando $T = '0'$ a saída é de alta impedância. O bloco “meio somador” é ativado por um pino de enable.



```

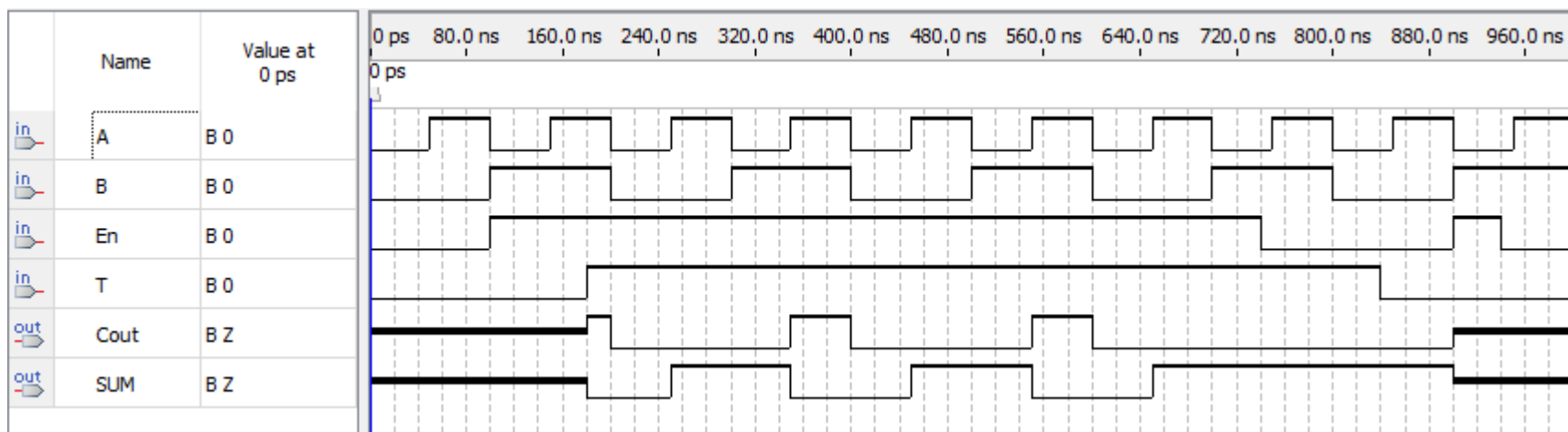
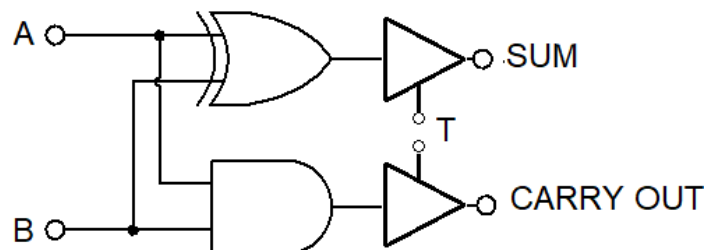
ENTITY meio_somador IS
  PORT(A, B, T, En: IN STD_LOGIC;
        SUM, Cout: OUT STD_LOGIC);
END meio_somador;

```

```

ARCHITECTURE funcao OF meio_somador IS
BEGIN
  bloc01: BLOCK(En='1')
  BEGIN
    SUM <= GUARDED (A XOR B) WHEN T='1' ELSE 'Z';
    Cout <= GUARDED (A AND B) WHEN T='1' ELSE 'Z';
  END BLOCK bloc01;
END funcao;

```



- Códigos sequenciais - IF-THEN-ELSE

- PEDRONI, Volnei A. Eletrônica Digital Moderna e VHDL. 1. ed. Campus. 2010, 648 p. ISBN 8535234659