

# VHDL

Técnicas Digitais II

Prof. Jorge Amaral

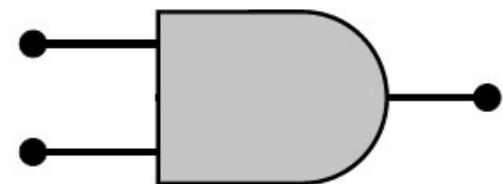
[jamaral@eng.uerj.br](mailto:jamaral@eng.uerj.br)

# Linguagem de Descrição de Hardware

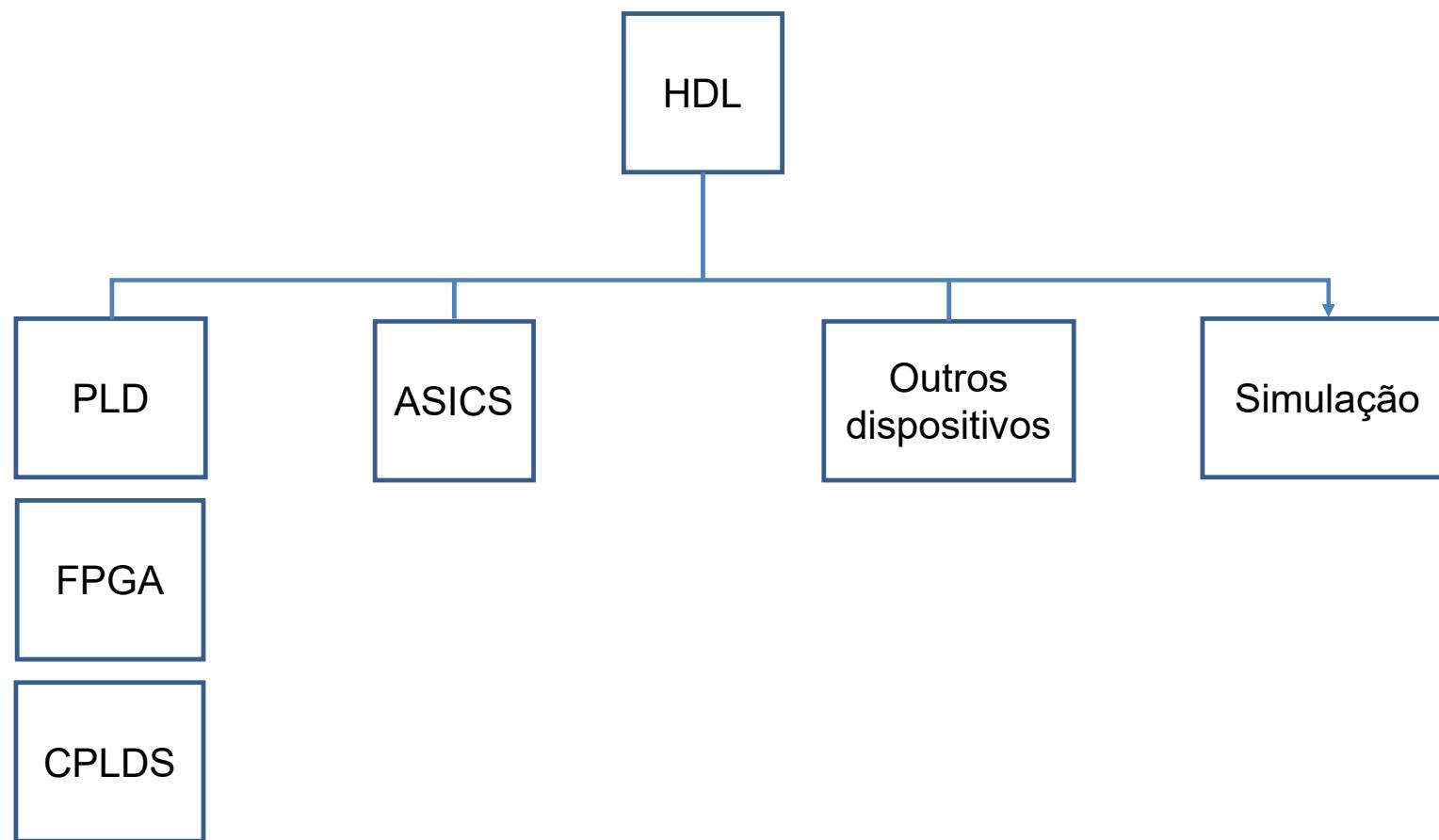
- HDL – Linguagem de Descrição de Hardware. Permite que linguagens rigidamente definidas representem circuitos lógicos.

```
entity and_gate is
  port (
    a, b  :  in  bit;
    z      :  out bit
  );
end and_gate;

architecture logic of and_gate is
begin
  z <= a and b;
end architecture ; -- logic
```



# Linguagem de Descrição de Hardware



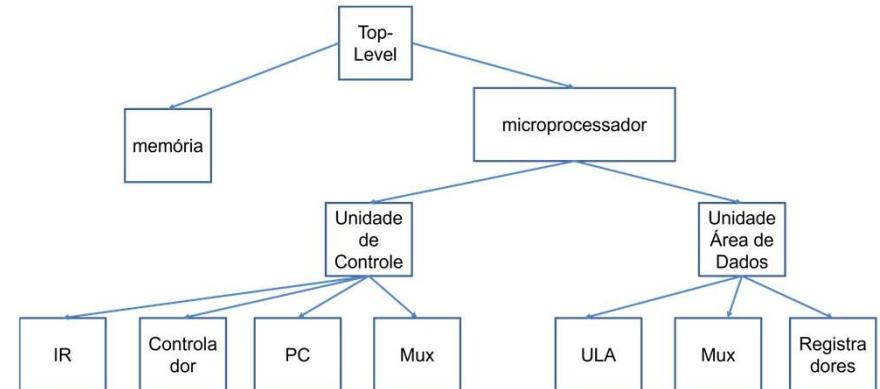
# VHDL é HDL

- VHDL – Linguagem de Descrição de Hardware de Velocidade Muito Alta. Desenvolvido pelo Departamento de Defesa Norte-Americano (DoD), padronizado pelo IEEE e amplamente utilizado para descrever projetos para dispositivos reais.
  - VHSIC (very High Speed Integrated Circuit)
  - Hardware
  - Description
  - Language

IEEE 1076-1987, 1076-1993, 1076 2000, 1076-2002, 1076-2008

# Aspectos gerais da Linguagem

- Top down
- Estilos de descrição
  - Comportamental
  - Expressões Lógicas
  - Redes de Ligações
  - Misto
- Ferramentas de síntese:
  - suportam diferentes estilos de descrição
  - normalmente: modos preferenciais devem ser empregados
- Concorrente
  - ordem dos comandos: não importa
  - -comandos sequenciais: somente em regiões específicas

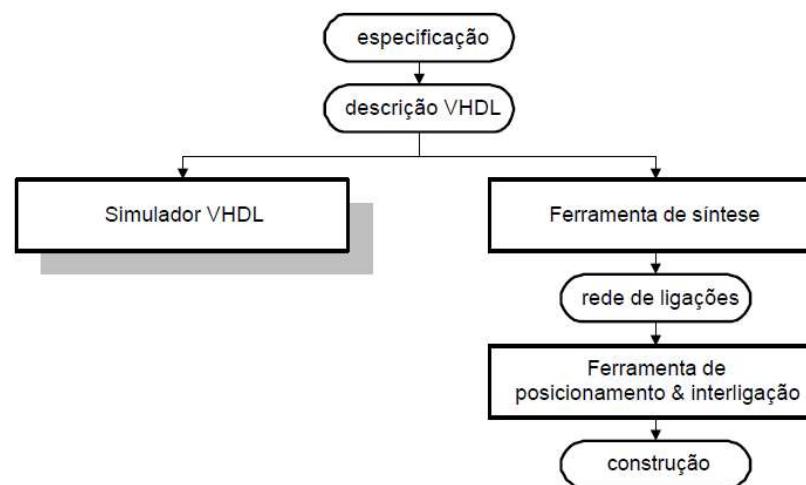


# Síntese de Circuitos

- VHDL: não foi concebida para síntese de circuitos
  - consequência: nem todas construções são suportadas
- Motivos da limitação:
  - falta de correspondência da construção / descrição com um circuito
    - exemplo: flip flop com dois terminais de clk
- impossibilidade da síntese direta
  - exemplo: multiplicação de dois números reais

# Etapas gerais do processo de Síntese

- Simulador VHDL
  - Compilação e/simulação do código



Fonte: VHDL Descrição e Síntese de Circuitos Digitais Roberto d'Amore ISBN 85-216-1452-7

Ferramenta de  
síntese

descrição VHDL

nível RTL

nível portas

otimização velocidade / área

rede de ligações

dados da  
tecnologia

opções de projeto

teste do circuito  
sintetizado

estímulos

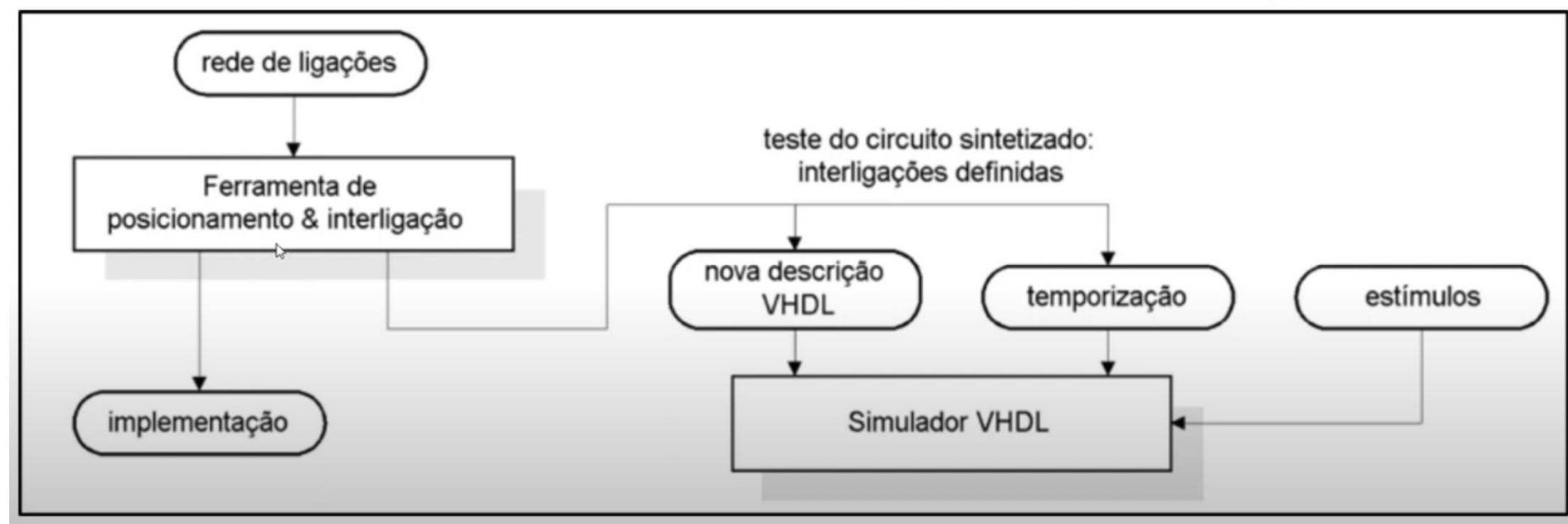
nova descrição  
VHDL

temporização

Simulador VHDL

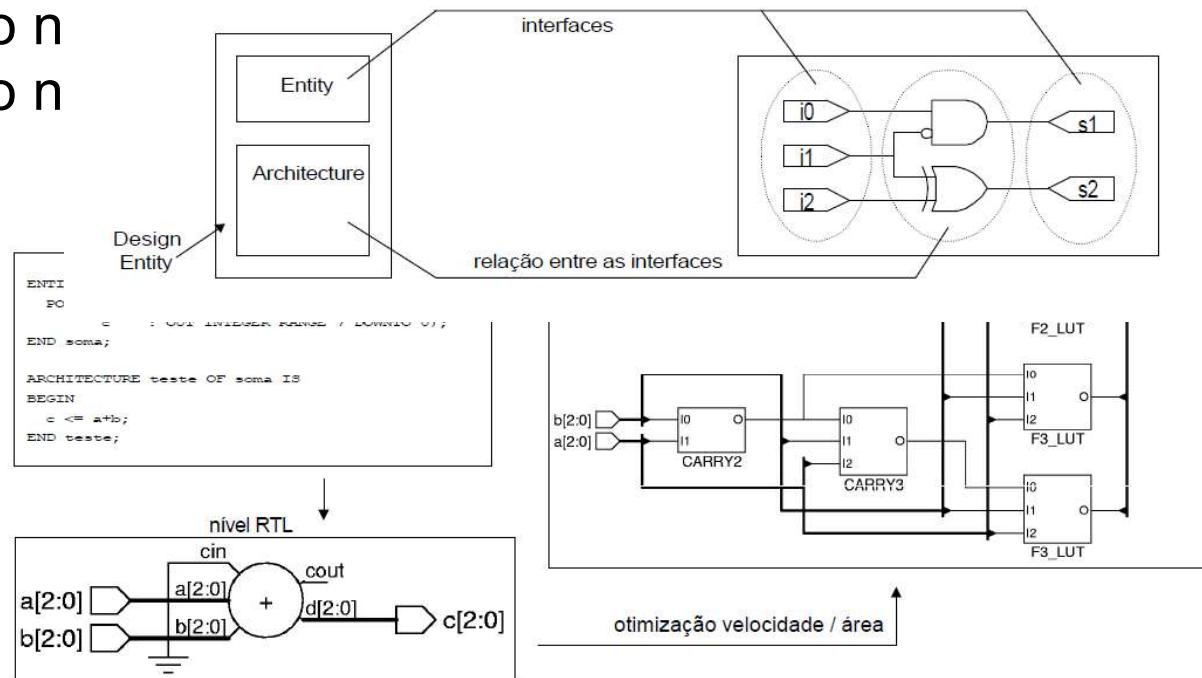


# Place and Route



# Etapas gerais do processos de síntese

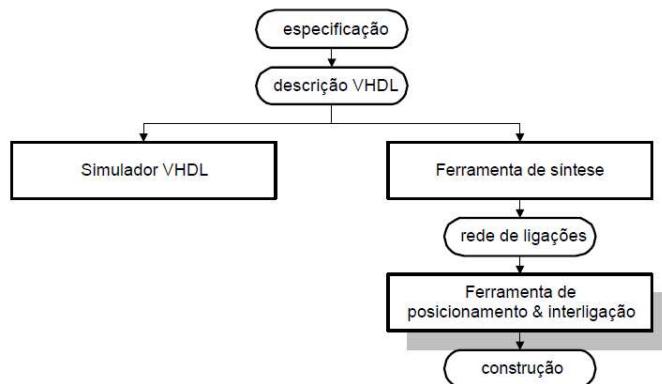
- Síntese de descrição
  - Descrição do circuito
  - Circuito n
  - Circuito n



Fonte: VHDL Descrição e Síntese de Circuitos Digitais Roberto d'Amore ISBN 85-216-1452-7

# Etapas gerais do processos de síntese

- Ferramenta de posicionamento e interligação
  - Posiciona e interliga as primitivas / componentes
  - dispositivo empregado na implementação:
    - FPGA - dispositivo lógico programável
    - ASIC - circuitos integrados de aplicação específica



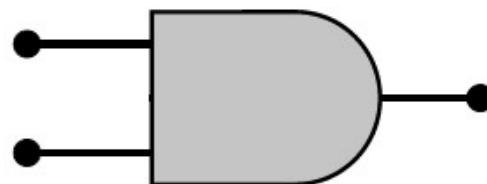
Fonte: VHDL Descrição e Síntese de Circuitos Digitais Roberto d'Amore ISBN 85-216-1452-7

# Linguagem de Descrição versus Linguagem de Programação

- É importante distinguir entre as linguagens de descrição de hardware e as linguagens de programação.
  - Em ambas, a linguagem é usada para programar um dispositivo.
- Os computadores funcionam seguindo uma lista de tarefas, que devem ser realizadas em ordem sequencial. A velocidade de operação é determinada pela rapidez do computador para executar cada instrução.
- Um circuito de lógica digital é limitado em velocidade apenas pela rapidez com que o circuito pode mudar as saídas em resposta a mudanças nas entradas. É possível monitorar todas as entradas simultaneamente e responder a quaisquer alterações.

# Linguagem de Descrição versus Linguagem de Programação

Comparação entre a operação de um computador  
e um circuito de lógica na execução da operação lógica  $y = AB$ .



O circuito lógico é uma porta AND. A saída  $y$  será ALTO a partir de 10 nanossegundos do momento em que A e B são ALTO simultaneamente.  
Dentro de aproximadamente 10 nanossegundos após uma entrada se tornar BAIXO, a saída  $y$  será BAIXO.

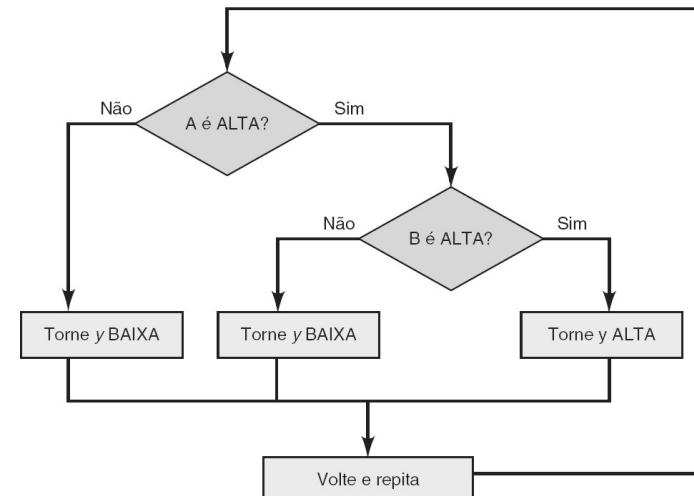
# Linguagem de Descrição versus Linguagem de Programação

Comparação entre a operação de um computador e um circuito de lógica na execução da operação lógica  $y = AB$ .

O computador deve executar um programa de instruções que toma decisões.

Cada forma no fluxograma representa uma instrução.

Se cada uma leva 20 ns, estima-se que irá demorar de duas a três instruções (40 - 60 ns) para responder às mudanças nas entradas.



# Implementação de Circuitos Lógicos com PLDs

- Dispositivos lógicos programáveis (PLDs) podem ser configurados de várias maneiras para executar funções lógicas.
- Conexões internas são feitas eletronicamente para dispositivos de programação.

# Implementação de Circuitos Lógicos com PLDs

- A linguagem de descrição de hardware define as conexões a serem feitas.
- É carregada no dispositivo após a tradução por um compilador.
- A linguagem de alto-nível de descrição de hardware torna a programação de PLDs muito mais fácil, se comparada à álgebra booleana, aos desenhos esquemáticos ou às tabelas-verdade.

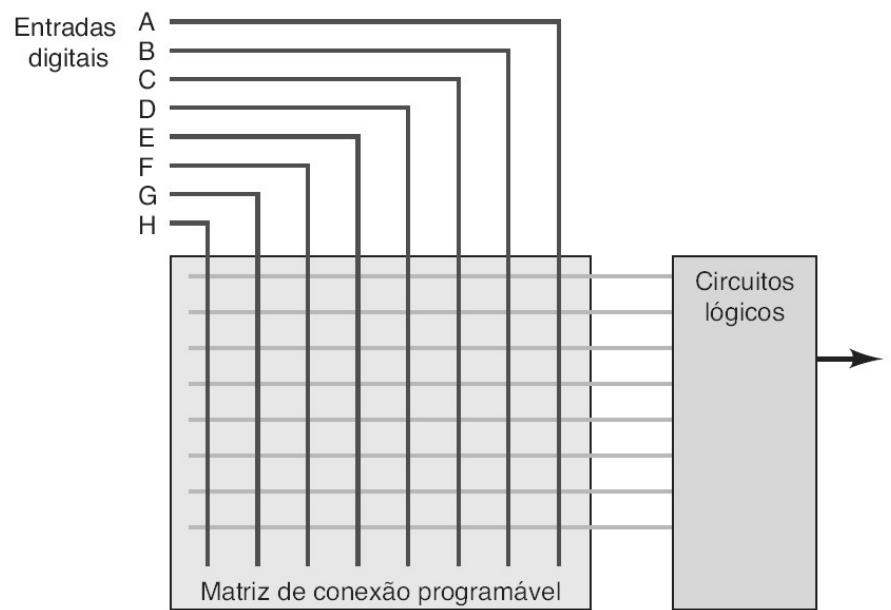
# Implementação de Circuitos Lógicos com PLDs

PLDs são configurados eletronicamente.

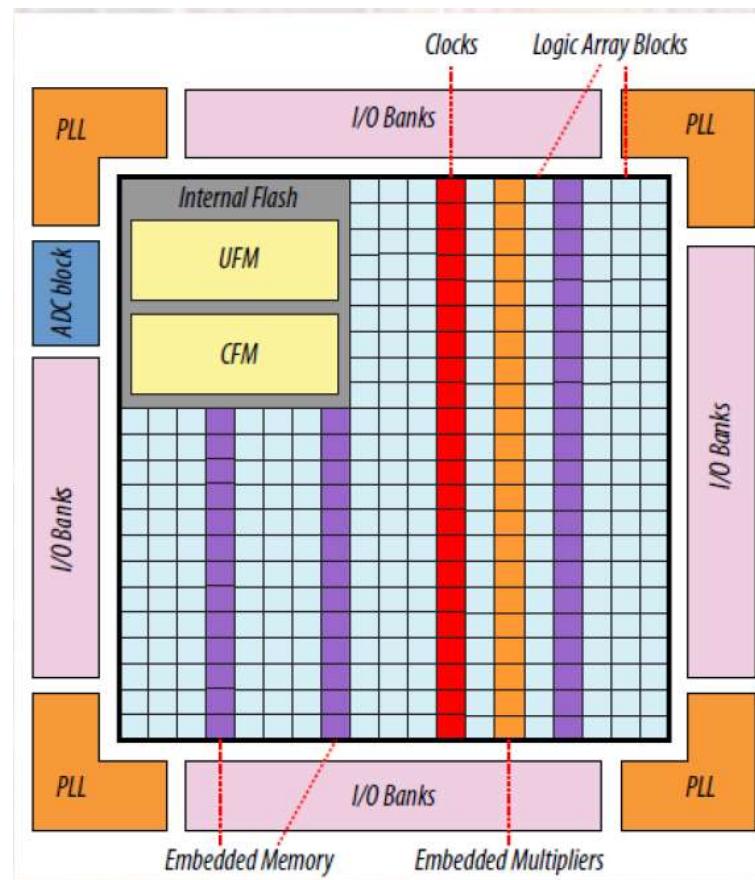
Seus circuitos internos são conectados eletronicamente para formar um circuito lógico.

Essa fiação programável pode ser pensada como milhares de conexões, conectadas (1) ou não (0).

Cada interseção de uma linha (fio horizontal) e coluna (fio vertical) é uma conexão programável.

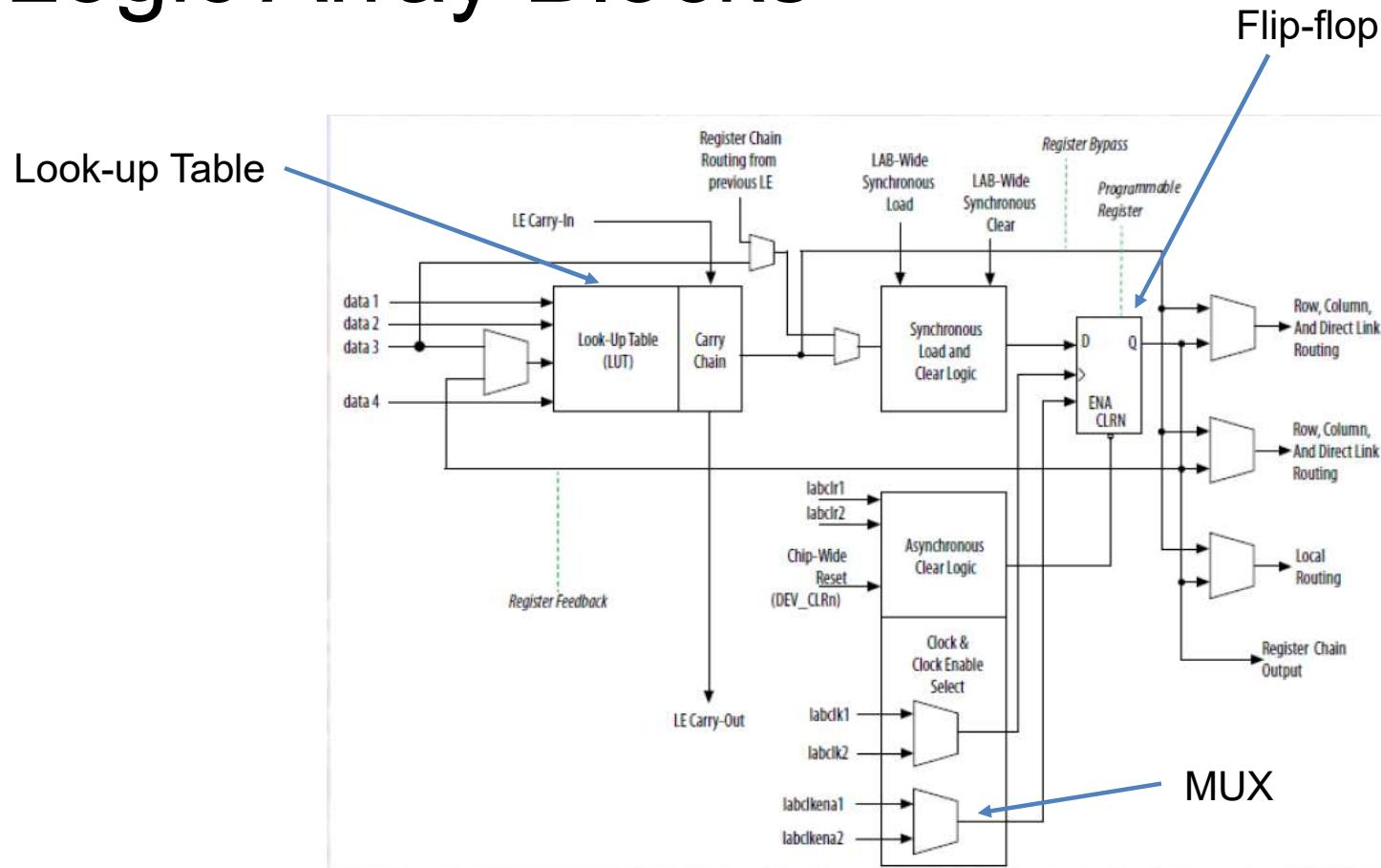


# Floorplan FPGA MAX10

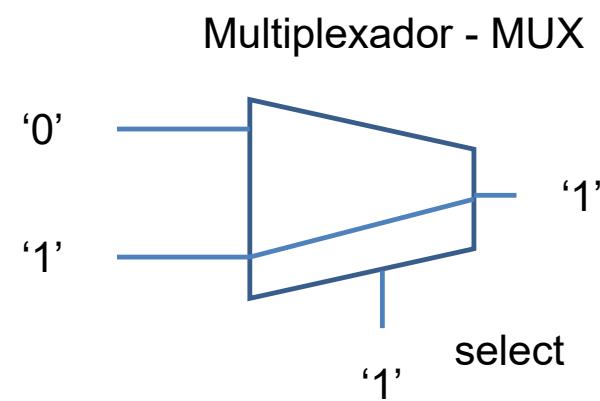
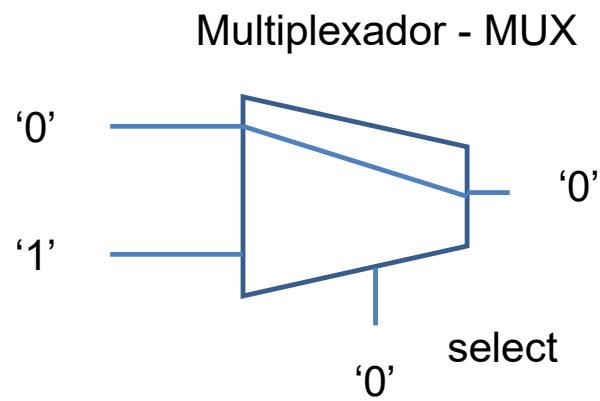


Fonte : MAX10 Device Handbook

# Logic Array Blocks

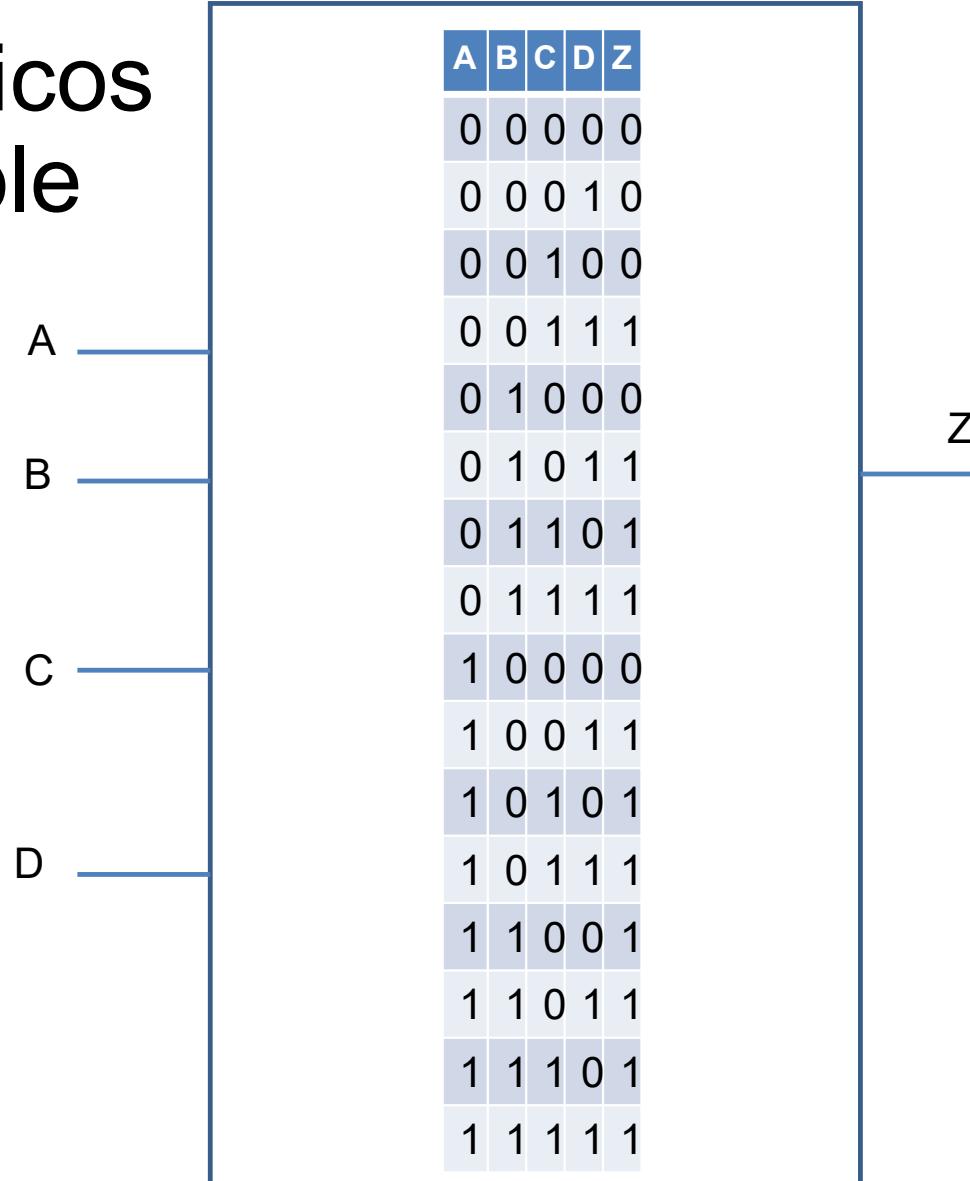


# Blocos básicos



# Blocos Básicos

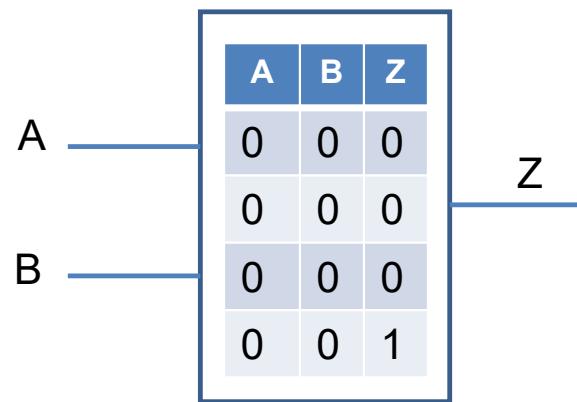
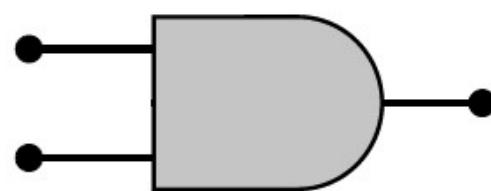
## Look up table



# Blocos Básicos

## Look up table

$Z \leq A \text{ and } B$

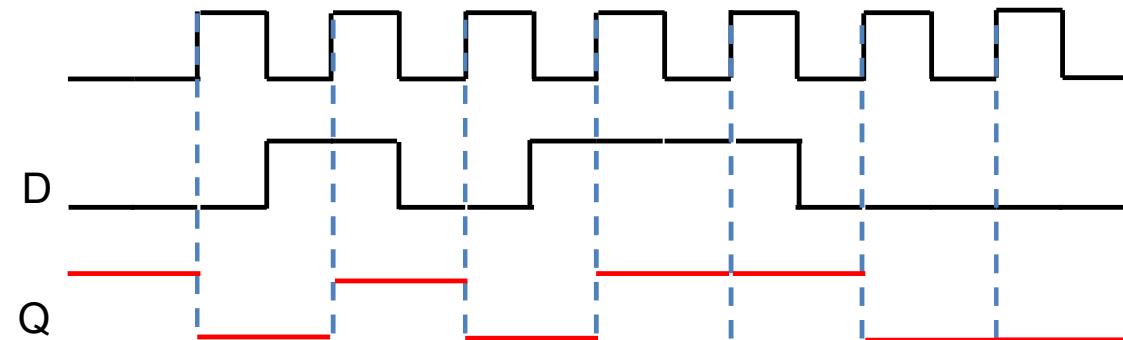
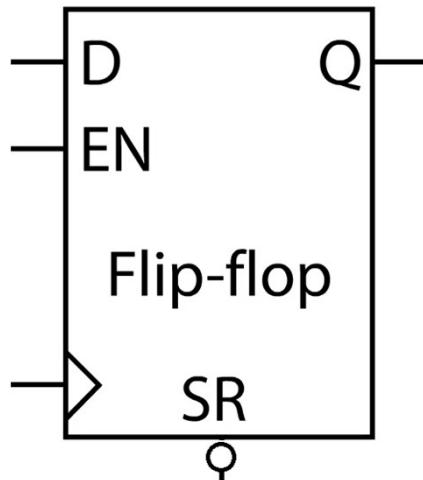


A large truth table enclosed in a blue border. The columns are labeled A, B, C, D, and Z. The rows show the following values:

A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Inputs A, B, C, and D are shown as blue lines on the left, with A at the top. The output Z is shown as a blue line on the right, connected to the bottom-right cell of the table.

# Blocos básicos flip-flop

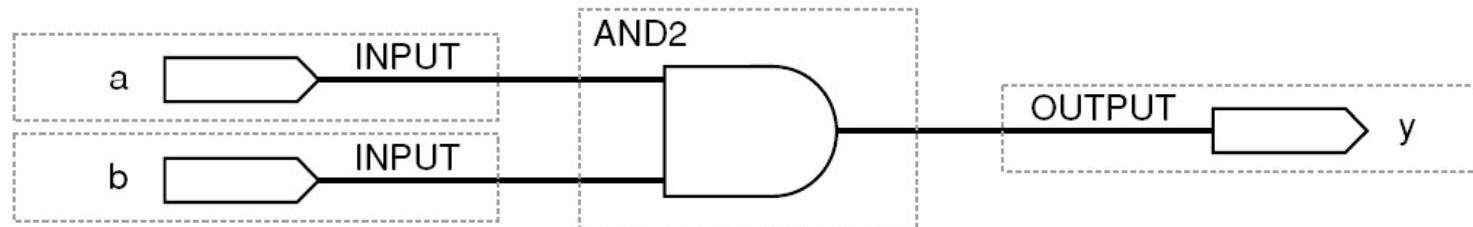


EN = ENABLE : habilita ou não o Flip-Flop

SR = Set/Reset do flip-flop

# Formato e Sintaxe do VHDL

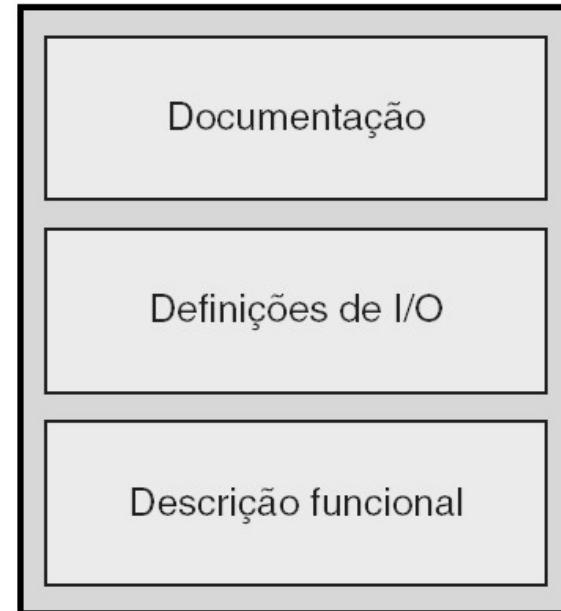
- Linguagens interpretadas por computadores devem seguir regras rígidas de sintaxe.
- A sintaxe refere-se a ordem dos elementos.
- No lado esquerdo do diagrama está o conjunto de entradas e à direita está o conjunto de saídas.



# Formato e Sintaxe do HDL

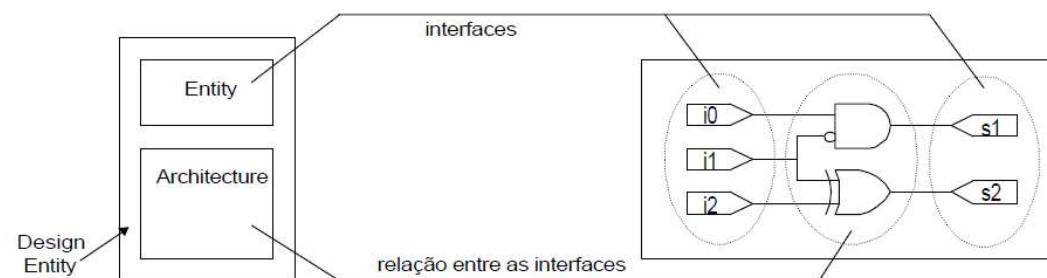
- O formato se refere a uma definição de entradas, saídas e de como as saídas respondem as entradas (operação)

Formato dos arquivos HDLs.



# Formato e Sintaxe do VHDL

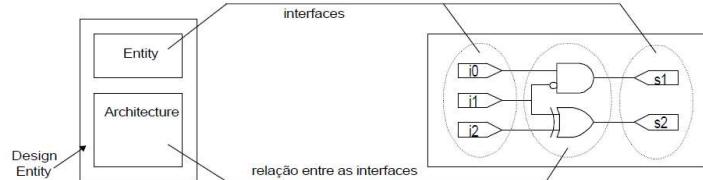
- Entidade de projeto
  - Pode representar de uma porta lógica até um sistema completo
  - Composta por:
    - Declaração da entidade
      - define portas de entrada e saída
    - Arquitetura
      - Relações entre as portas



Fonte: VHDL Descrição e Síntese de Circuitos Digitais Roberto d'Amore ISBN 85-216-1452-7

# Formato e Sintaxe do HDL

- Declaração de entidade
  - ENTITY: inicia a declaração
  - PORT: define o modo e o tipo das portas
    - IN
    - OUT
    - BUFFER: saída pode ser referenciada internamente
    - INOUT: bidirecional
  - END: termina a declaração



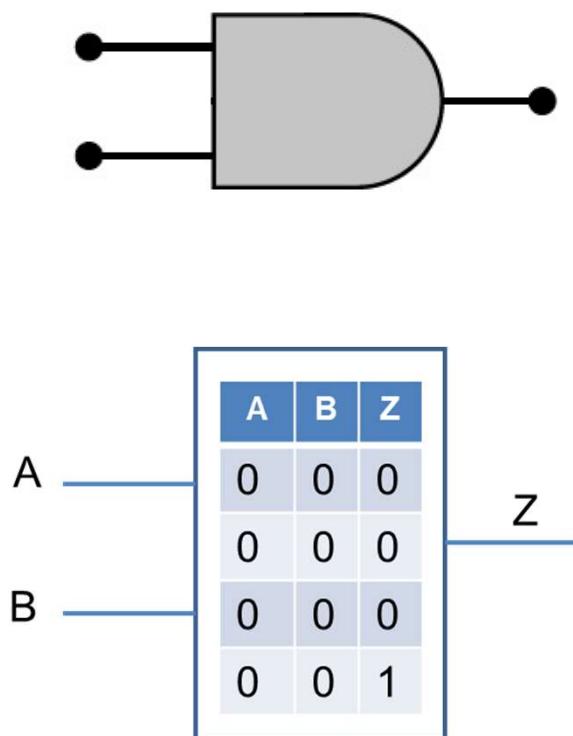
```
ENTITY entidade_abc IS
  PORT (x0, x1      : IN      tipo_a;    -- entradas
        y0, y1      : OUT     tipo_b;    -- saídas
        y2          : BUFFER  tipo_c;    -- saída
        z0, z1      : INOUT   tipo_d);  -- entrada / saída
END entidade_abc;
```

Fonte: VHDL Descrição e Síntese de Circuitos Digitais Roberto d'Amore ISBN 85-216-1452-7

- Declaração de arquitetura
- **ARCHITECTURE**: inicia a declaração
  - Linhas seguintes
    - declaração de sinais e constantes
    - declaração de componentes referenciados
    - descrição de subprogramas locais
    - definição de novos tipos
    - **BEGIN**: inicia a descrição
    - **END**: termina a descrição

```
: ARCHITECTURE estilo_abc OF entidade_abc IS
  -- declaracoes de sinais e constantes
  -- declaracoes de componentes referenciados
  -- descricao de sub-programas locais
  -- definicao de novos tipos de dados locais
  --
BEGIN
  --
  -- declaracoes concorrentes
  --
END estilo_abc;
```

# Formato e Sintaxe do VHDL



```
ENTITY and_gate IS
PORT (    a, b      :IN BIT;
          y         :OUT BIT);
END and_gate;
ARCHITECTURE ckt OF and_gate IS
BEGIN
          y <= a AND b;
END ckt;
```

# Formato e Sintaxe do VHDL

- Exemplos de classes de objetos : constante e sinal
  - Exemplo:
    - CONSTANT: valor estático
    - SIGNAL: valor imposto pode ser alterado em regiões de código concorrente e sequencial

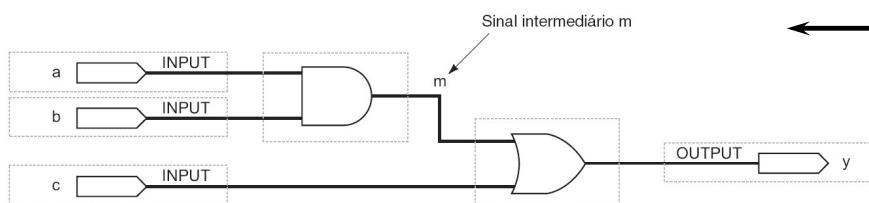
```
sinal_2    <= sinal_1;      -- transferencia entre sinais
sinal_4    <= constante_1;  -- transferencia de constante para sinal
```

# Formato e Sintaxe do HDL

- Constante – Valor fixo
  - Constant nome\_constant: tipo := valor;
- Signal – meio físico onde os dados podem transitar
  - Signal nome\_signal: tipo:= valor inicial;
- Variable – usado somente em código sequencial;
  - Variable nome\_Variable:= tipo:= valor\_inicial;
- FILE – criação e leitura de arquivos;
  - File nome\_file:tipo;

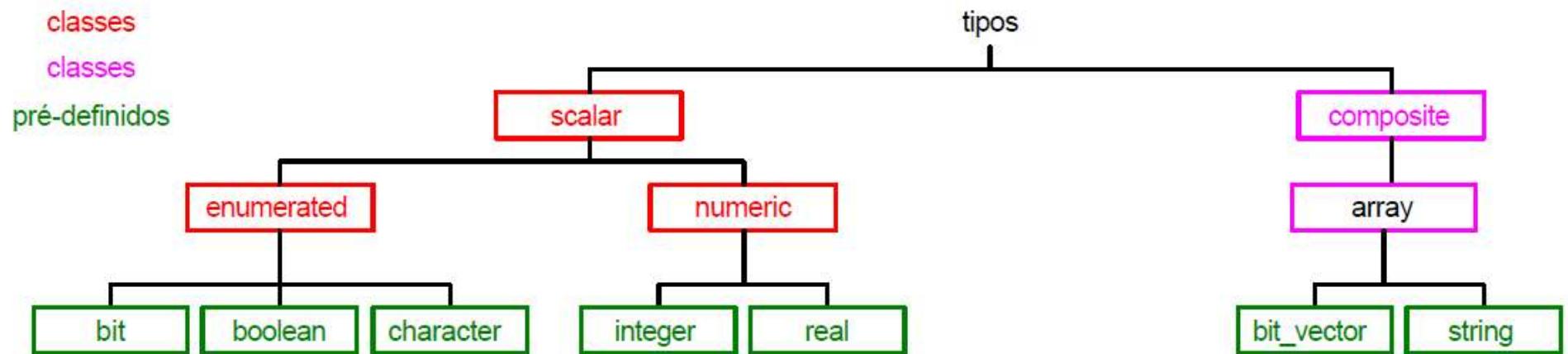
# Sinais Intermediários em VHDL

```
1  -- Variáveis intermediárias em VHDL (Figura 3.49)
2  -- Sistemas digitais 11a. ed
3  -- NS Widmer
4  -- MAY 24, 2010
5
6  ENTITY fig3_51 IS
7  PORT( a, b, c  :IN BIT;      -- define entradas no bloco
8        y         :OUT BIT);   -- define a saída do bloco
9  END fig3_51;
10
11 ARCHITECTURE ckt OF fig3_51 IS
12
13     SIGNAL m      :BIT;      -- nomeia um sinal intermediário
14
15 BEGIN
16     m <= a AND b;          -- gera um termo de produto interno
17     y <= m OR c;          -- gera soma na saída
18 END ckt;
```



# Formato e Sintaxe do VHDL

- Tipos
  - Objetos : devem ser declarados segundo uma especificação de tipo
  - Objetos de tipos diferentes:
    - Não é permitida a transferência de valores



Fonte: VHDL Descrição e Síntese de Circuitos Digitais Roberto d'Amore ISBN 85-216-1452-7

- Tipos escalares

classe	tipo	valor	exemplos
enumerado	BIT	Um, zero	1,0
	BOOLEAN	Verdadeiro, falso	TRUE, FALSE
	CHARACTER	Caracteres ASCII	a,b,c, A, B, C, ?
numérico	INTEGER  Representado em 32 bits	$-2^{31} \leq x \leq 2^{31} - 1$	123, 65565
	REAL  Não suportado pela ferramentas de síntese	$-3.65 \times 10^{47} \leq x \leq 3.65 \times 10^{47}$	1.23, 12E+2

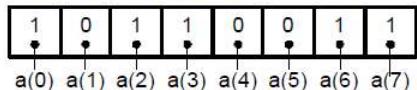
- Tipos compostos

- Classe vetor

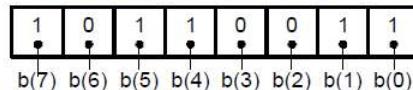
- **BIT\_VECTOR**: vetor contendo elementos tipo bit
- **STRING**: vetor contendo elementos tipo character

Classe	tipo	valor	exemplos
vetor	BIT_VECTOR	"1", "0"	"1010", B"10_10", O"12", X"A"
	STRING	tipo "character"	"texto", ""incluso_aspas""

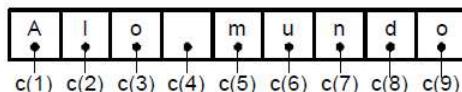
CONSTANT a: BIT\_VECTOR(0 TO 7) := "10110011"



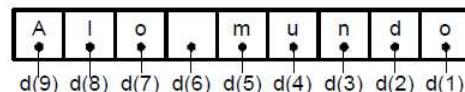
CONSTANT b: BIT\_VECTOR(7 DOWNTO 0) := "10110011"



CONSTANT c: STRING(1 TO 9) := "Alo mundo"



CONSTANT d: STRING(1 DOWNTO 9) := "Alo mundo"



- Operadores

- Divididos em classes:
  - Classes definem a precedência dos operadores
  - operadores de uma mesma classe: igual precedência
- Maior precedência: classe diversos
- Menor precedência: classe lógicos
- Operador “not”: operador lógico, está na classe diversos devido a precedência

classe	operadores					
lógicos	and or nand nor xor xnor					
relacionais	= /= < <= > >=					
adição	+ - &					
diversos	not					

- Operadores relacionais
  - igualdade e desigualdade (`=` `/=`): qualquer tipo
  - -  $a=b$  para escalares: a mesmo valor de b
  - -  $a=b$  para compostos: cada elemento de mesma posição igual
  - ordenação (`>` `<` `>=` `<=`): tipos escalares (bit, boolean, character, integer, real, time)

operadores	operando “L”	operando “R”	retorna
<code>=</code> <code>/=</code>	qualquer tipo	mesmo tipo de “L”	boolean
<code>&gt;</code> <code>&lt;</code> <code>&gt;=</code> <code>&lt;=</code>	qualquer tipo escalar	mesmo tipo de “L”	boolean

## • Operadores de adição

- adição e subtração (+ -): tipo numérico
- concatenação (&): vetor unidimensional e elementos (mesmo tipo)

operadores	operando "L"	operando "R"	retorna
+ -	tipo numérico	o mesmo tipo de "L"	mesmo tipo
&	vetor unidimensional	vetor unidimensional	vetor unidimensional
	vetor unidimensional	elemento	vetor unidimensional
	elemento	vetor unidimensional	vetor unidimensional
	elemento	elemento	vetor unidimensional

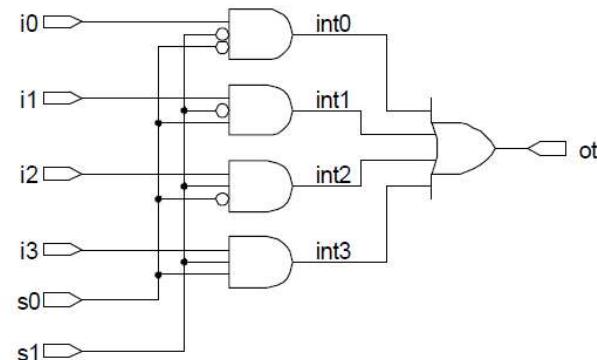
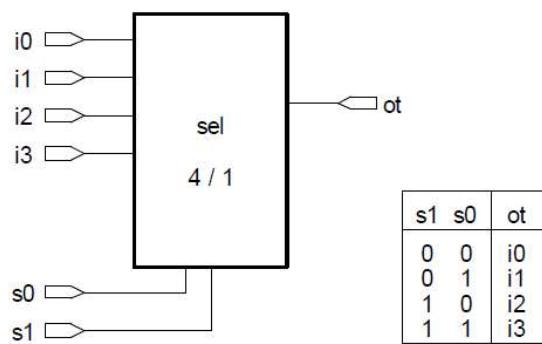
# Comandos concorrentes básicos

- WHEN ELSE
- WITH SELECT

# Estrutura When else

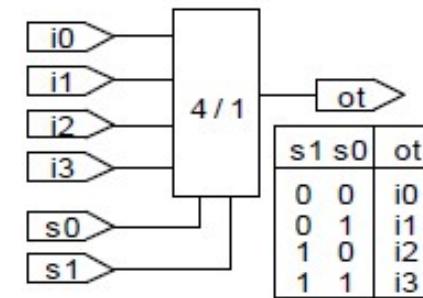
- **Descrição de um circuito de seleção**

- entradas: i0, i1, i2 e i3
- saída: ot
- controle da seleção: s0 e s1



# Estrutura When else

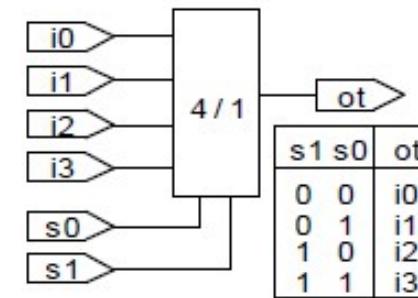
- Descrição de um circuito de seleção
  - entradas: i0, i1, i2 e i3
  - saída: ot
  - controle da seleção: s0 e s1



```
1 ENTITY mux_0 IS
2   PORT (i0, i1, i2, i3      : IN BIT;    -- entradas
3         s0, s1           : IN BIT;    -- selecao
4         ot                : OUT BIT); -- saida
5 END mux_0;
6
7 ARCHITECTURE nivel_logico OF mux_0 IS
8 BEGIN
9   ot <= (i0 AND NOT s1 AND NOT s0) OR
10     (i1 AND NOT s1 AND     s0) OR
11     (i2 AND     s1 AND NOT s0) OR
12     (i3 AND     s1 AND     s0);
13 END nivel_logico;
```

# Estrutura When else

- Descrição de um circuito de seleção
  - entradas: i0, i1, i2 e i3
  - saída: ot
  - controle da seleção: s0 e s1

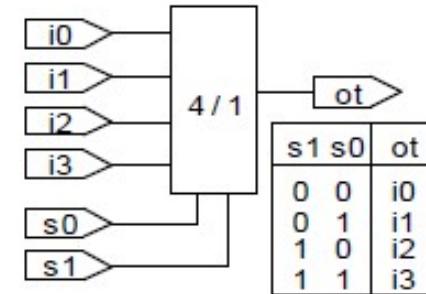


```
1 ENTITY mux_00 IS
2   PORT (i0, i1, i2, i3      : IN  BIT;    -- entradas
3           s0, s1          : IN  BIT;    -- selecao
4           ot              : OUT BIT); -- saida
5 END mux_00;
6
7 ARCHITECTURE teste OF mux_00 IS
8   SIGNAL int0, int1, int2, int3 : BIT; -- sinais internos
9 BEGIN
10   ot  <= int0 OR int1 OR int2 OR int3;
11   int0 <= i0 AND NOT s1 AND NOT s0;
12   int1 <= i1 AND NOT s1 AND     s0;
13   int2 <= i2 AND      s1 AND NOT s0;
14   int3 <= i3 AND      s1 AND     s0;
15 END teste;
```

A ordem das instruções está correta?

# Estrutura When else

- nível de abstração mais elevado
- descrição mais próxima do comportamento do circuito
- opção de escolha:



```
1 ENTITY mux_1 IS
2   PORT (i0, i1, i2, i3      : IN  BIT;
3         s0, s1           : IN  BIT;
4         ot                : OUT BIT);
5 END mux_1;
6
7 ARCHITECTURE teste OF mux_1 IS
8   SIGNAL s1_s0 : BIT_VECTOR(1 DOWNTO 0);
9 BEGIN
10   ot <= i0 WHEN s1= '0' AND s0='0' ELSE
11     i1 WHEN s1= '0' AND s0='1' ELSE
12     i2 WHEN s1_s0="10"          ELSE
13     i3;
14 END teste;
```

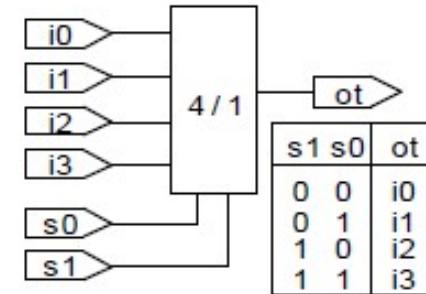
# WITH SELECT

- Transferência condicional de um sinal
- Contém: uma lista de opções e expressões
- Todas condições da expressão de escolha devem ser consideradas
  - não existe uma prioridade como na construção WHEN ELSE
- Opções pode ser agrupadas:
  - caractere | equivale a “ou”
- - TO e DOWNTO delimitam faixas de opções
- Opções restantes: palavra reservada OTHERS
- Formato da construção:

```
WITH expressao_escolha SELECT          -- expressao_escolha =
    sinal_destino <= expressao_a WHEN condicao_1,      -- condicao_1
                                    expressao_b WHEN condicao_2,      -- condicao_2
                                    expressao_c WHEN condicao_3 | condicao_4,  -- condicao_3 ou condicao_4
                                    expressao_d WHEN condicao_5 TO condicao_7, -- condicao_5 ate condicao_7
                                    expressao_e WHEN OTHERS;           -- condicoes restantes
```

# Estrutura With SELECT

- nível de abstração mais elevado
- descrição mais próxima do comportamento do circuito
- opção de escolha: sinal sel = s1 e s0 concatenados



```
1 ENTITY mux_9 IS
2   PORT (i0, i1, i2, i3 : IN BIT;
3         s0, s1 : IN BIT;
4         ot : OUT BIT);
5 END mux_9;
6
7 ARCHITECTURE teste OF mux_9 IS
8   SIGNAL sel : BIT_VECTOR (1 DOWNTO 0);
9 BEGIN
10   sel <= s1 & s0;
11   WITH sel SELECT
12     ot <= i0 WHEN "00",
13           i1 WHEN "01",
14           i2 WHEN "10",
15           i3 WHEN "11";
16 END teste;
```

# Tabelas-verdade - VHDL

$$x = BC + AC + AB$$

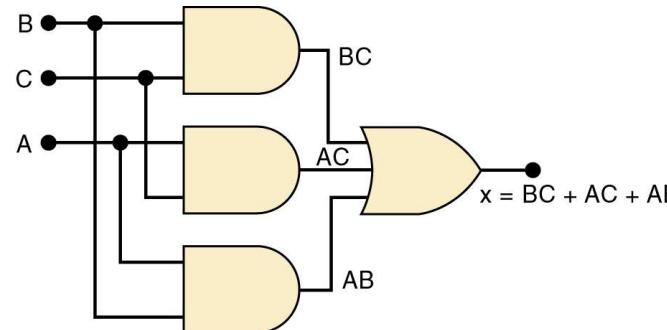
A	B	C	x
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$\rightarrow \bar{A}\bar{B}C$

$\rightarrow A\bar{B}\bar{C}$

$\rightarrow A\bar{B}\bar{C}$

$\rightarrow ABC$



```

ENTITY fig4_51 IS
PORT(
    a,b,c :IN BIT;           --a é mais significativo
    y      :OUT BIT);
END fig4_51;

ARCHITECTURE truth OF fig4_51 IS
    SIGNAL in_bits :BIT_VECTOR(2 DOWNTO 0);
BEGIN
    in_bits <= a & b & c;   --concatena bits de entrada em bit_vector
    WITH in_bits SELECT
        y  <=  '0' WHEN "000",  --tabela-verdade
              '0' WHEN "001",
              '0' WHEN "010",
              '1' WHEN "011",
              '0' WHEN "100",
              '1' WHEN "101",
              '1' WHEN "110",
              '1' WHEN "111";
END truth;
    
```

# Comandos Sequenciais Básicos

- Processos
- Lista de sensibilidade
- Construção IF ELSE
- Construção CASE WHEN

# Comandos Sequenciais

- Comandos seqüenciais podem ocorrer em:
  - processos
  - subprogramas
- Processo
  - é um comando concorrente
  - delimita uma região contendo código sequencial
- Uma descrição: composta de comandos concorrentes
- todas são executadas concorrentemente



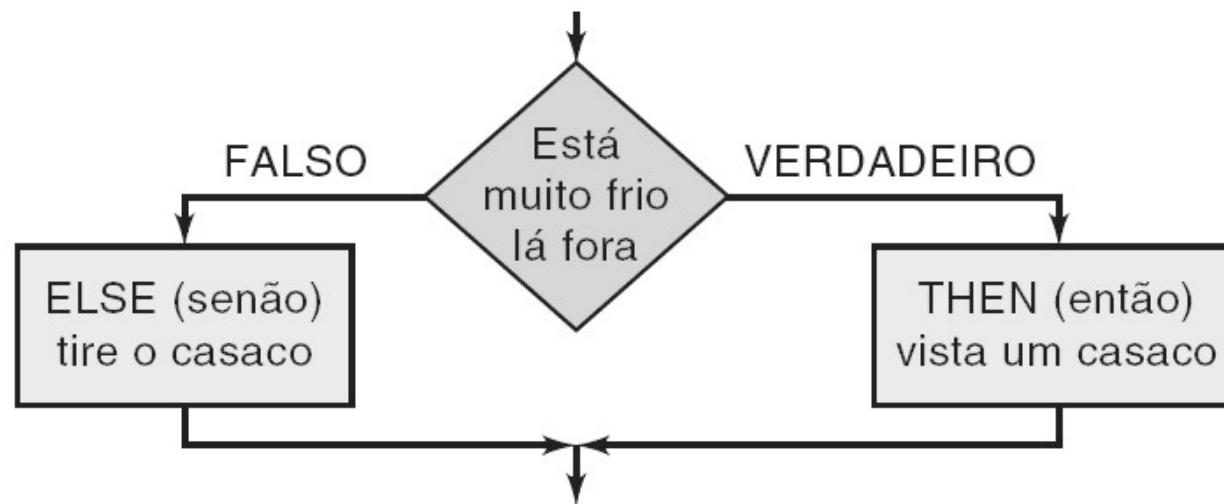
# Lista de sensibilidade em processos

- Após a palavra reservada PROCESS:
  - possível declarar a lista de sensibilidade
- Lista de sensibilidade:
  - define quais sinais causam a execução do processo
- Execução do processo ocorre se:
  - um sinal da lista tem valor alterado
- Iniciada a execução:
  - comandos são avaliados na seqüência
  - ao término da avaliação do último comando:
  - processo é suspenso (aguarda uma nova alteração de valor - sinais da lista)

```
abc: PROCESS(sinal_a, sinal_b)      -- (lista de sensibilidade)
  BEGIN
    comando_1;
    comando_2;
    ..
    comando_n;
  END PROCESS abc;
```

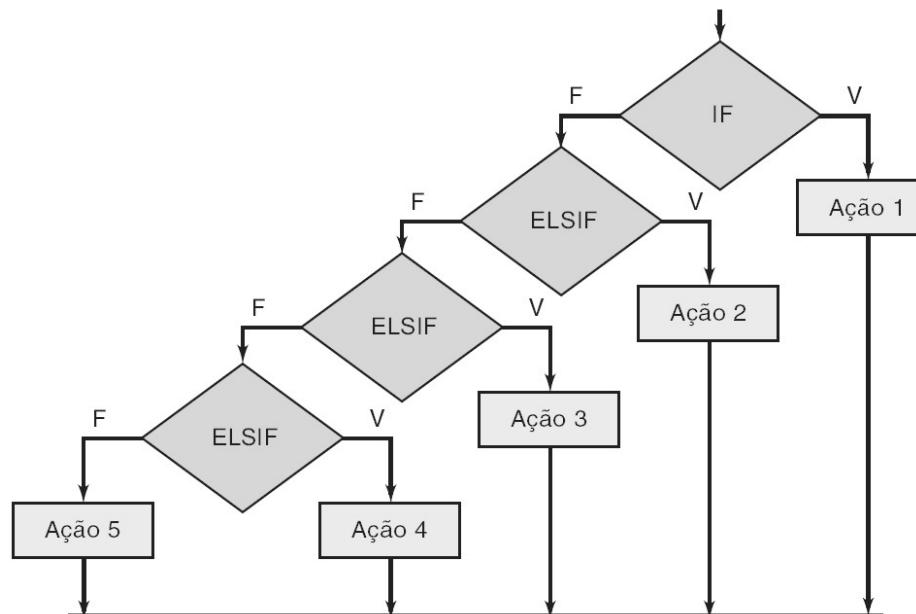
# Estruturas de Controle de Decisão em HDL – IF/THEN/ELSE

- Declarações IF/THEN/ELSE fornecem um quadro para a tomada de decisões lógicas em um sistema.
- IF/THEN/ELSE é usada quando existe uma escolha entre duas ações possíveis.



# Estruturas de Controle de Decisão em HDL – IF/THEN/ELSE

- Ao combinar decisões IF e ELSE, podemos criar uma estrutura de controle chamada ELSIF, a qual escolhe um dos muitos resultados possíveis.



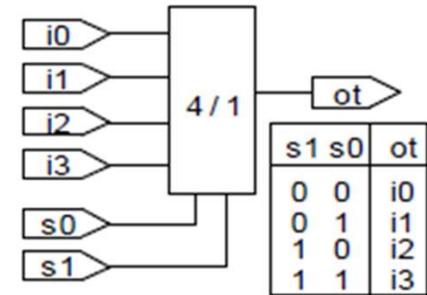
# IF ELSE

- Execução condicional de um ou mais comandos seqüenciais
- Teste: definido por uma lista de condições
- Primeira condição verdadeira: define as comandos executados
- Condição de teste: qualquer expressão que retorne BOOLEAN
- Início da construção: comandos IF
- Cláusulas ELSIF e ELSE: opcionais

```
IF condicao_1 THEN
    comando_sequencial;
    comando_sequencial;
ELSIF condicao_2 THEN      -- clausula ELSIF opcional
    comando_sequencial;
    comando_sequencial;
ELSIF condicao_3 THEN
    comando_sequencial;
ELSE                      -- clausula ELSE opcional
    comando_sequencial;
END IF;
```

# IF ELSE

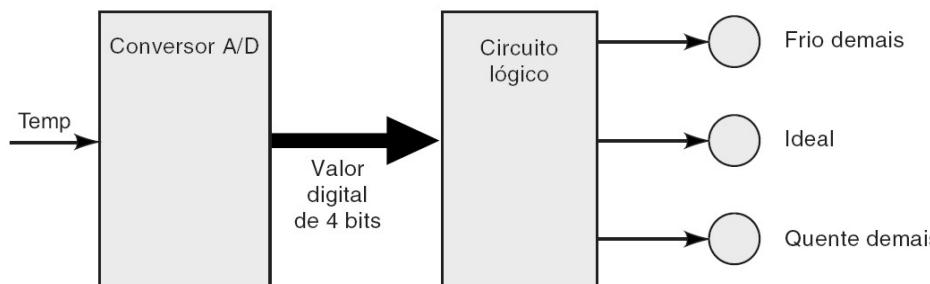
- Comando sequencial: necessário definir um processo
- Lista de sensibilidade: sinais **i0 i1 i2 i3 sel**
- - remoção de um destes sinais: consequência?



```
1 ENTITY mux_4aa IS
2   PORT (i0, i1, i2, i3 : IN BIT; -- entradas
3         s0, s1 : IN BIT; -- selecao
4         ot      : OUT BIT); -- saida
5 END mux_4aa;
6
7 ARCHITECTURE teste OF mux_4aa IS
8   SIGNAL sel : BIT_VECTOR(1 DOWNTO 0);
9 BEGIN
10   sel <= s1 & s0;
11   abc: PROCESS (i0, i1, i2, i3, sel)
12   BEGIN
13     IF sel = "00" THEN ot <= i0;
14     ELSIF sel = "01" THEN ot <= i1;
15     ELSIF sel = "10" THEN ot <= i2;
16     ELSE                  ot <= i3;
17     END IF;
18   END PROCESS abc;
19 END teste;
```

# Estruturas de Controle de Decisão em HDL – IF/THEN/ELSE

Um sistema de medição de temperatura usando um conversor A /D.



<i>Valores Digitais</i>	<i>Categoria</i>
0000–1000	Frio demais
1001–1010	Ideal
1011–1111	Quente demais

IF (se) o valor digital é menor ou igual a 8...

THEN (então) acenda apenas o indicador **Muito Frio**.

ELSE/IF (senão, se) o valor digital for maior que 8 AND  
(e) menor que 11...

THEN (então) acenda apenas o indicador Na **medida certa**.

ELSE (senão) acenda apenas o indicador **Muito Quente**.

# Estruturas de Controle de Decisão em HDL – IF/THEN/ELSE

```
ENTITY fig4_59 IS
PORT(valor_digital:IN INTEGER RANGE 0 TO 15; -- declara entrada de 4 bits
      frio_demais, ideal, quente_demais :OUT BIT);
END fig4_59 ;

ARCHITECTURE howhot OF fig4_59 IS
SIGNAL status :BIT_VECTOR (2 downto 0);
BEGIN
    PROCESS (valor_digital)
    BEGIN
        IF (valor_digital <= 8) THEN status <= "100";
        ELSIF (valor_digital > 8 AND valor_digital < 11) THEN
            status <= "010";
        ELSE status <= "001";
        END IF;
    END PROCESS ;
    frio_demais     <= status(2);      -- atribui a status bits para a saída
    ideal          <= status(1);
    quente_demais <= status(0);
END howhot;
```

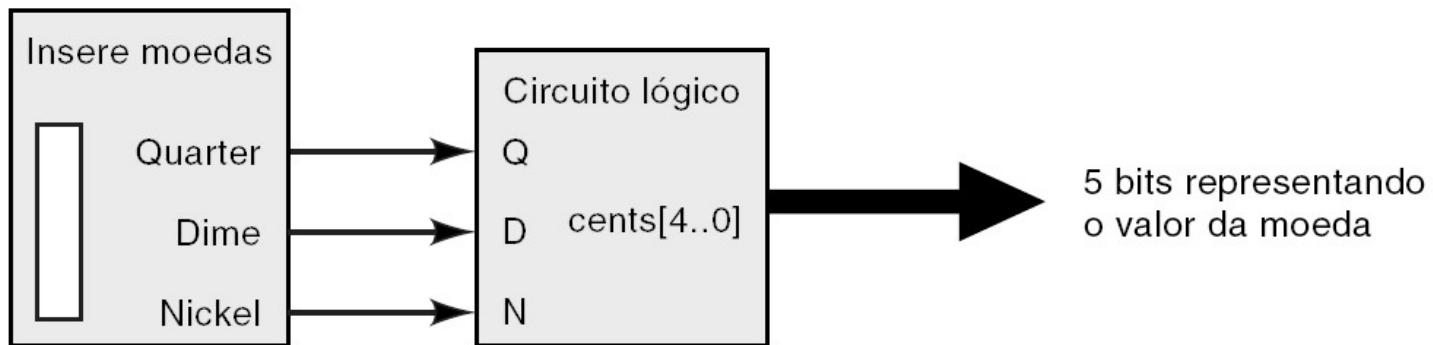
# CASE WHEN

- Execução condicional de um ou mais comando seqüenciais
- A execução dos comando: controlada pelo valor de uma expressão
- Todas condições da expressão de escolha devem ser consideradas
  - não existe prioridade (como na construção WHEN ELSE)
- Opções podem ser agrupadas: - caracter | equivale a “ou”
  - TO e DOWNTO delimitam faixas de opções
  - Opções restantes: palavra reservada OTHERS

```
CASE expressao_escolha IS
    WHEN condicao_1          => comando_a;           -- expressao_escolha =
    WHEN condicao_2          => comando_b; comando_c;   -- condicao_1
    WHEN condicao_3 | condicao_4 => comando_d;           -- condicao_2
    WHEN condicao_5 TO condicao_9 => comando_d;           -- condicao_3 ou condicao_4
    WHEN OTHERS               => comando_e; comando_f;  -- condicao_5 ate condicao_9
END CASE;
```

# Estruturas de Controle de Decisão em HDL - CASE

Um detector de moedas de uma máquina de vendas aceita moedas de 25 centavos, moedas de 10 centavos e moedas de 5 centavos, ativando o sinal digital correspondente (25, 10, 5) apenas quando a moeda correta está presente



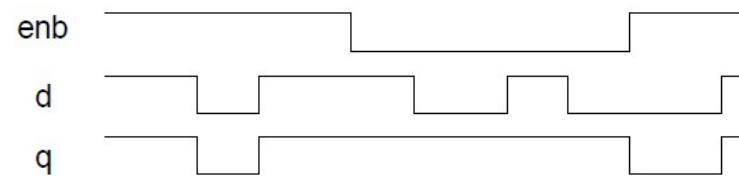
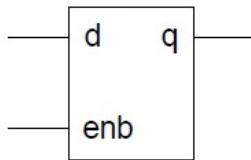
# Estruturas de Controle de Decisão em HDL - CASE

```
ENTITY fig4_64 IS
PORT( q, d, n:IN BIT;                                -- quarter, dime, nickel
      cents :OUT INTEGER RANGE 0 TO 25);-- valor binário das moedas
END fig4_64;
ARCHITECTURE detector of fig4_64 IS
  SIGNAL coins:BIT_VECTOR(2 DOWNTO 0);-- grupo de sensores de moedas
BEGIN
  coins <= (q & d & n);                      -- atribui sensores para o grupo
  PROCESS (coins)
  BEGIN
    CASE (coins) IS
      WHEN "001" => cents <= 5;
      WHEN "010" => cents <= 10;
      WHEN "100" => cents <= 25;
      WHEN others => cents <= 0;
    END CASE;
  END PROCESS;
END detector;
```

# Estratégias de descrição de circuitos síncronos

- Registrador sensível a nível
- Registrador sensível a borda - inicialização síncrona
- Registrador sensível a borda - inicialização assíncrona
- Contadores
- Máquinas de estado finito

# Registrador Sensível a nível



```
PROCESS (ena, d)
BEGIN
  IF (ena = '1') THEN  d <= q ;
  END IF ;
END PROCESS;
```

# Registrador Sensível a nível

- Set e Reset assíncronos (lista de sensibilidade)

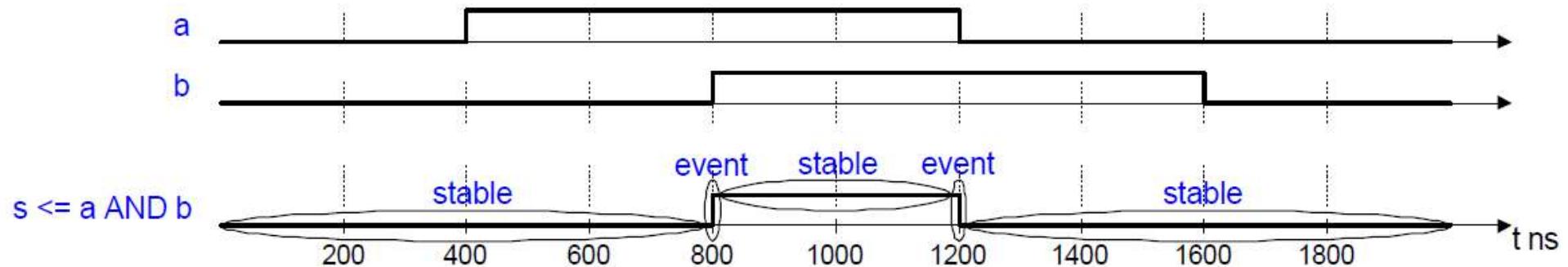
```
1 ENTITY latch3_1 IS
2   PORT (en    : IN  BIT;           -- habilita
3         rst   : IN  BIT;           -- rst=1 leva q=000
4         set   : IN  BIT;           -- set=1 leva q=111
5         d     : IN  BIT_VECTOR(2 DOWNTO 0);
6         q     : OUT BIT_VECTOR(2 DOWNTO 0));
7 END latch3_1;
8
9 ARCHITECTURE teste OF latch3_1 IS
10 BEGIN
11   PROCESS (en, d, rst, set)
12   BEGIN
13     IF      (rst ='1') THEN q <="000";  -- q=000 independente de en
14     ELSIF (set ='1') THEN q <="111";  -- q=111 independente de en
15     ELSIF (en  ='1') THEN q <=d;      -- condicao do sinal para habilitar
16     END IF;
17   END PROCESS;
18 END teste;
```

# Atributos

- Informações adicionais
  - associadas: tipos objetos e unidades de projeto
  - um objeto: (num dado instante de tempo)
  - pode conter um único valor
  - pode possuir vários atributos
- Atributo pode ser referenciado na forma: **prefixo'nome\_atributo**
  - prefixo: corresponde ao item (por exemplo um sinal)
  - nome\_atributo : atributo desejado do item
  - ' : separa o prefixo e o nome\_atributo

# Atributos

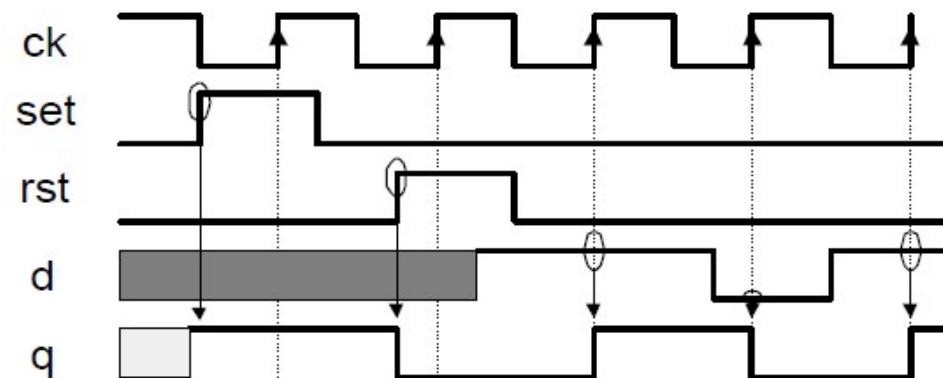
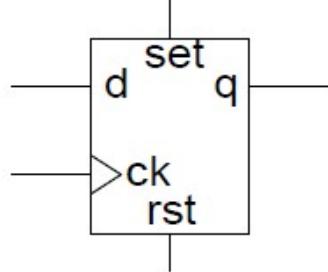
- `s'STABLE` - verdadeiro: não ocorreu uma troca de valor
  - falso caso contrario
- `s'EVENT` - verdadeiro: ocorreu uma troca de valor
- - falso: caso contrario



›

```
(ck'EVENT AND ck ='1')          -- borda de subida  
(ck'EVENT AND ck ='0')          -- borda de descida  
(NOT ck'STABLE AND ck ='1')     -- borda de subida  
(NOT ck'STABLE AND ck ='0')     -- borda de descida
```

# Registrador sensível a borda - inicialização assíncrona

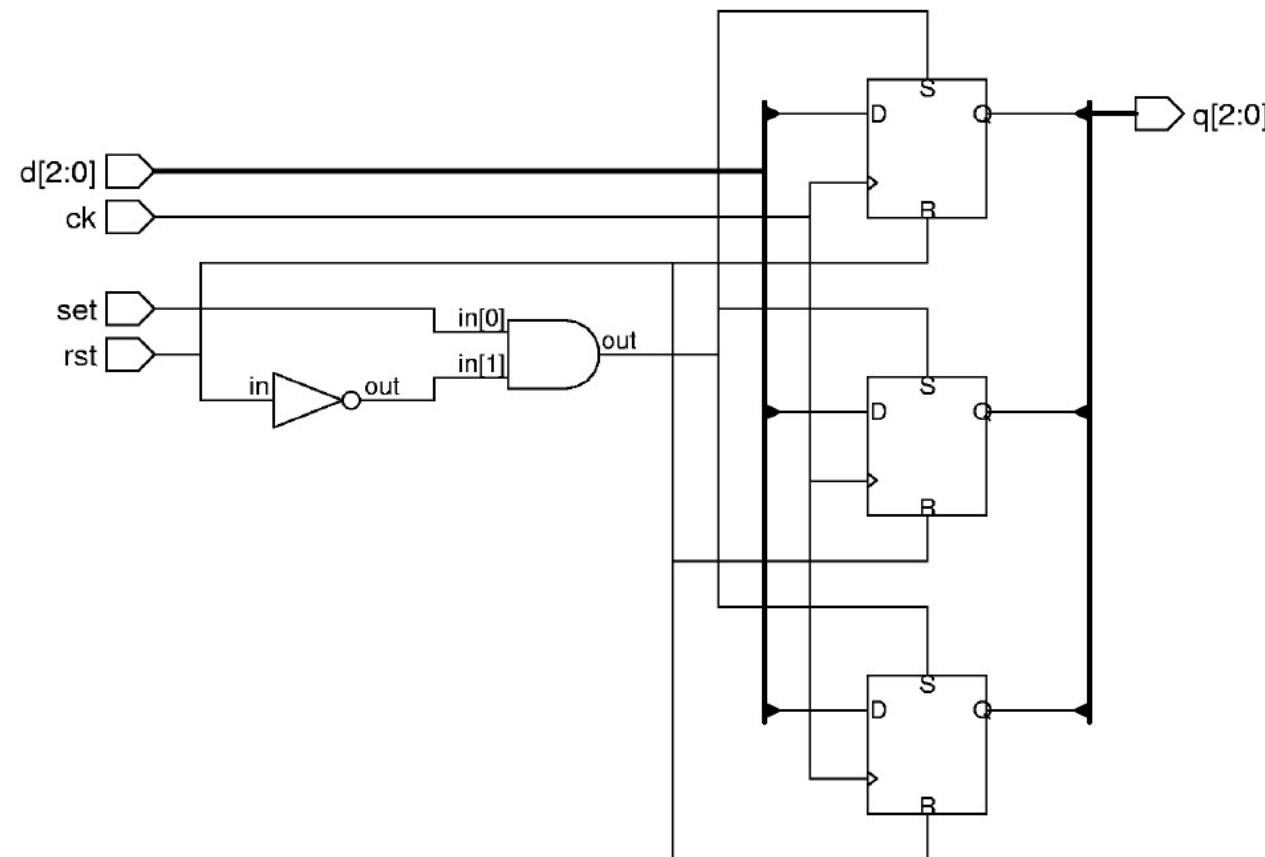


```
PROCESS (ck, rst, set)
BEGIN
  IF      (rst = '1')          THEN q <= '0'; -- eventos assincronos
  ELSIF   (set = '1')          THEN q <= '1'; -- .
  -- .
  -- .
  ELSIF (ck'EVENT AND ck ='1') THEN q <=d;    -- detecta borda de subida do relogio
  END IF;
END PROCESS;
```

# Registrador sensível a borda - inicialização assíncrona

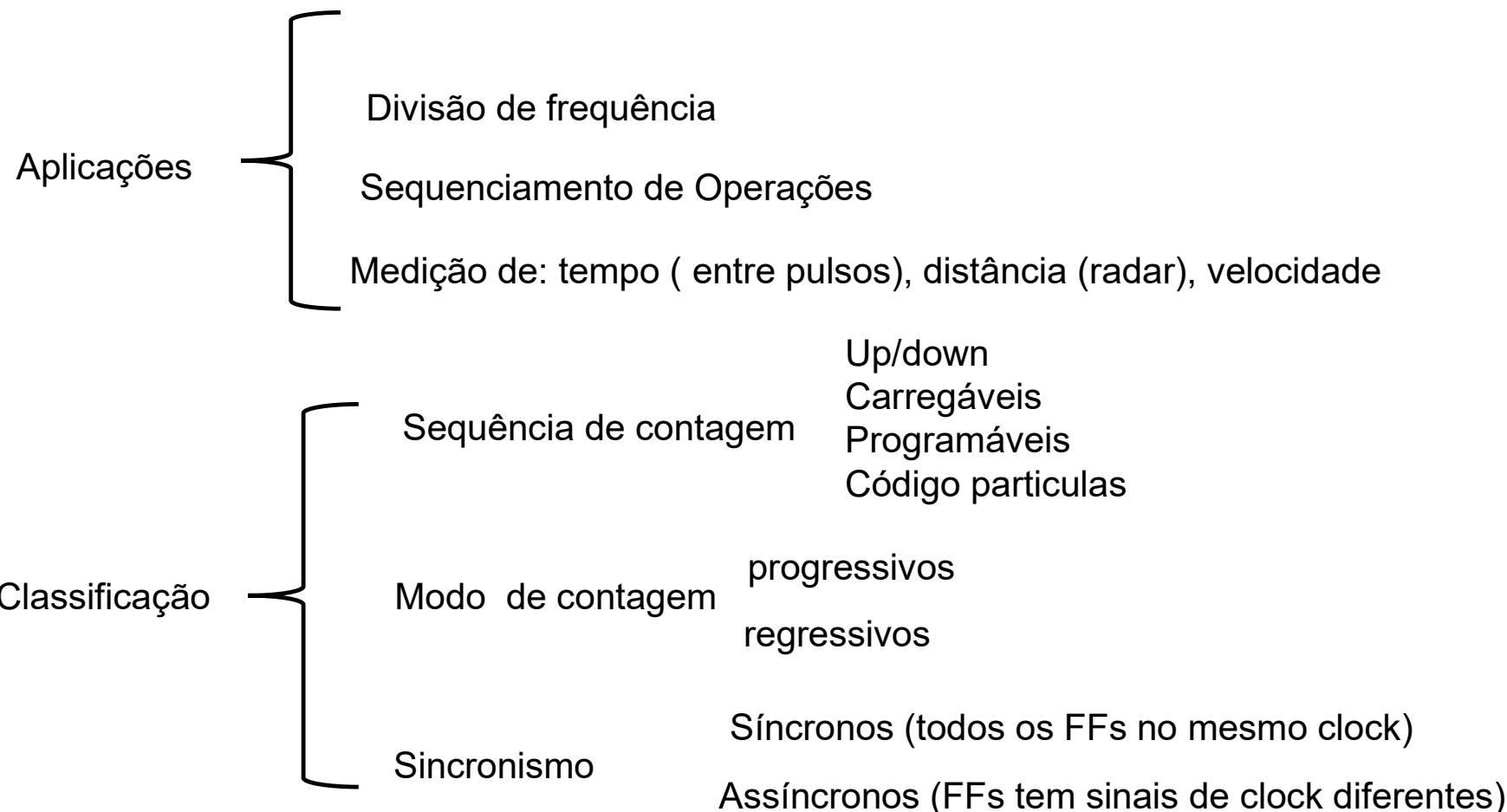
```
1 ENTITY flip3_3 IS
2     PORT (ck    : IN  BIT;                      -- relogio
3             rst   : IN  BIT;                      -- rst=1 leva q=000 assincrono
4             set   : IN  BIT;                      -- set=1 leva q=111 assincrono
5             d     : IN  BIT_VECTOR(2 DOWNTO 0);
6             q     : OUT BIT_VECTOR(2 DOWNTO 0));
7 END flip3_3;
8
9 ARCHITECTURE teste OF flip3_3 IS
10 BEGIN
11     PROCESS (ck, rst, set)
12     BEGIN
13         IF          (rst = '1')      THEN q <="000"; -- q=000 independente de ck
14         ELSIF      (set = '1')      THEN q <="111"; -- q=111 independente de ck
15         ELSIF (ck'EVENT AND ck ='1') THEN q <=d;      -- condicao do sinal relogio
16     END IF;
17     END PROCESS;
18 END teste;
```

# Registrador sensível a borda - inicialização assíncrona



# Contadores

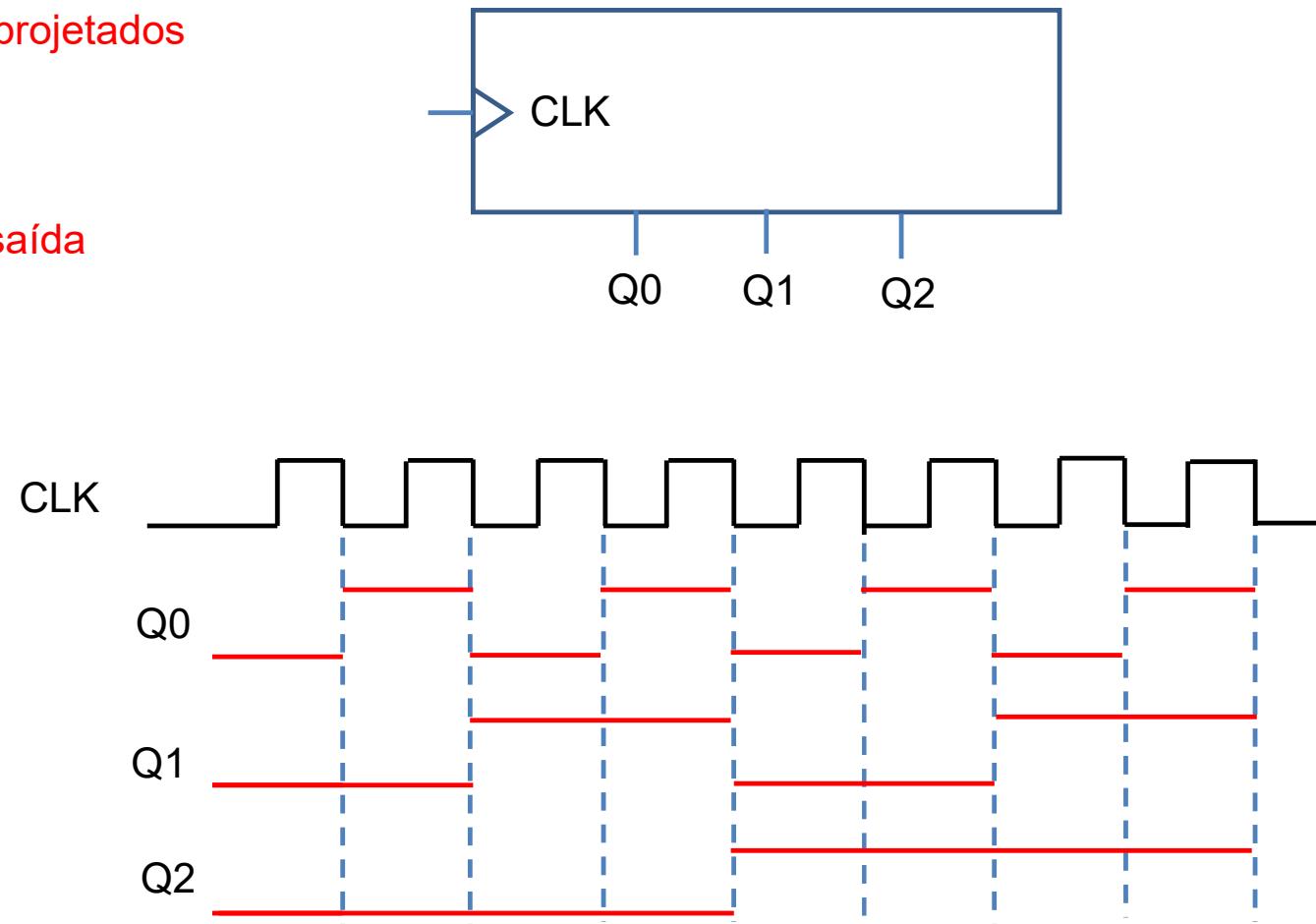
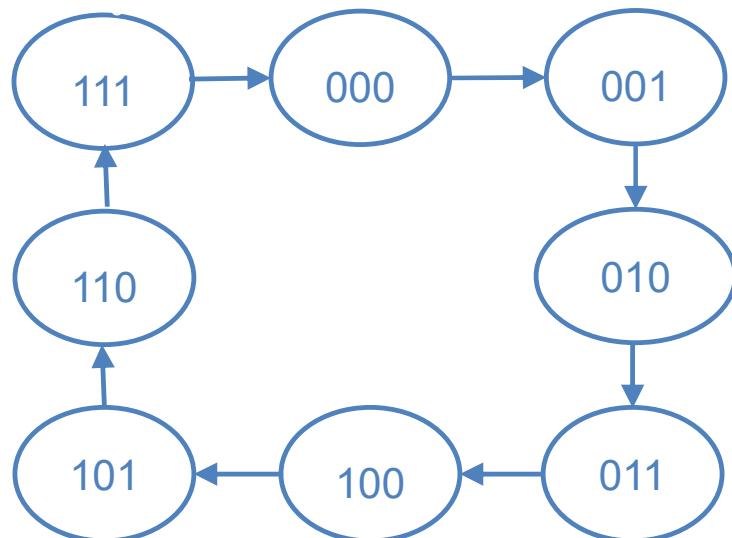
Um contador binário é um circuito capaz de contar, segundo uma determinada sequência, o número de pulsos de clock que recebe na entrada



# Contador síncrono de módulo 8.

Contadores síncronos podem ser facilmente projetados pelo método dos comportamentos

Máquinas de Moore onde o estado é igual a saída



# Contadores

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity simple_counter is
    generic(
        n: natural :=4
    );
    port(
        i_clk:      in std_logic;
        i_rstb:     in std_logic;
        o_cnt :  out std_logic_vector(n-1 downto 0)
    );
end entity;

architecture behavioral of simple_counter is

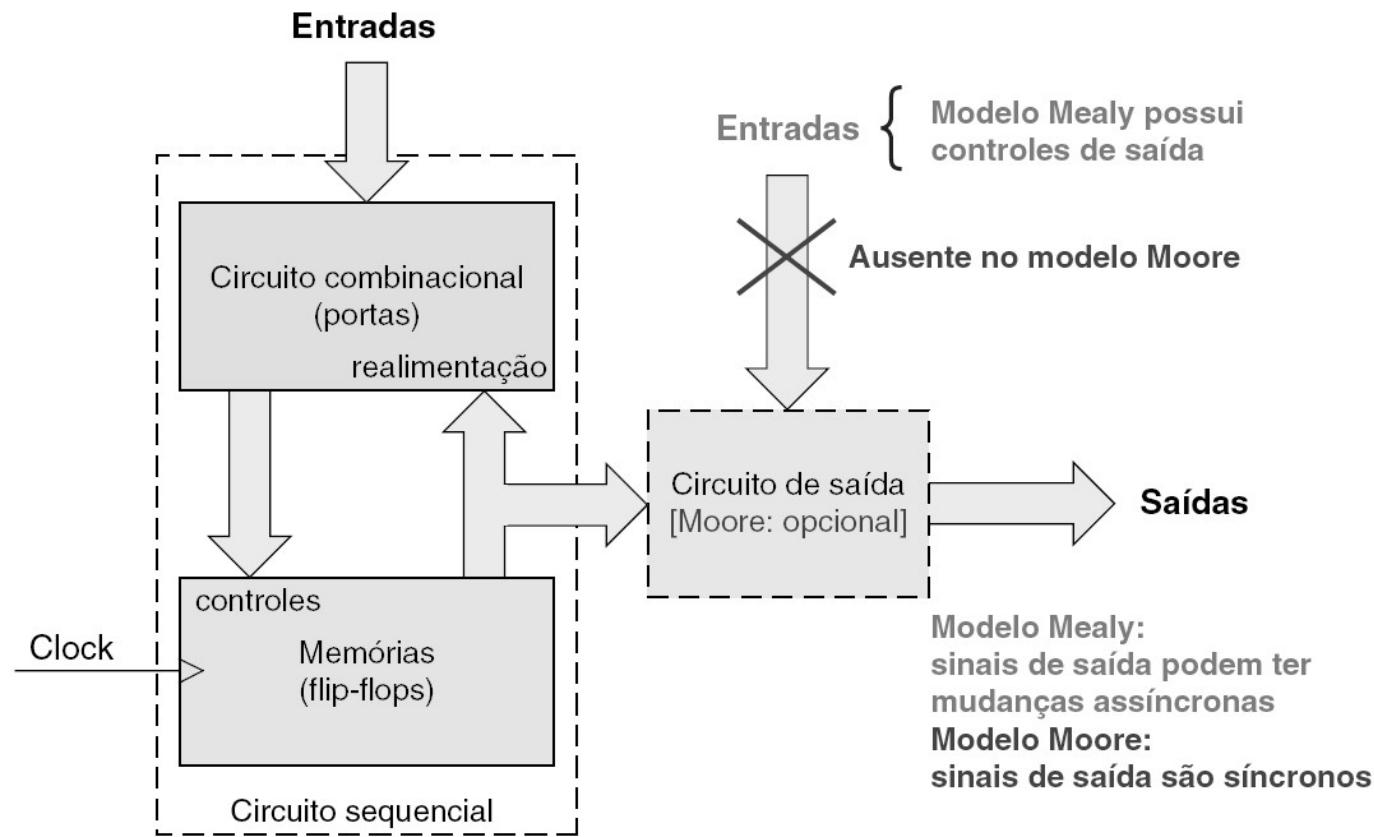
    -- internal signal
    signal cnt_sig: unsigned(n-1 downto 0);
begin
    process(i_clk, i_rstb)
    begin
        if (i_rstb = '0') then
            cnt_sig <= ( others => '0');
        elsif ( rising_edge(i_clk) ) then
            cnt_sig <= cnt_sig + 1;
        end if;
    end process;
    o_cnt <= std_logic_vector(cnt_sig);
end behavioral;
```

# Máquinas de Estado

- O termo **máquina de estado** se refere a um circuito que tem uma sequência através de um conjunto de estados predeterminados controlados por um clock e outros sinais de entrada.
- O termo contador é usado para circuitos sequenciais que têm uma sequência de contagens numéricas regulares.
- A distinção geral entre os dois termos:  
Um *contador* é comumente usado para *contar* eventos.  
Uma *máquina de estado* é comumente usada para *controlar* os acontecimentos.

# Máquinas de estado

Diagrama em blocos de contadores e máquinas de estado.



# Máquinas de Estado

Máquinas de estado podem estar em apenas um estado por vez no tempo

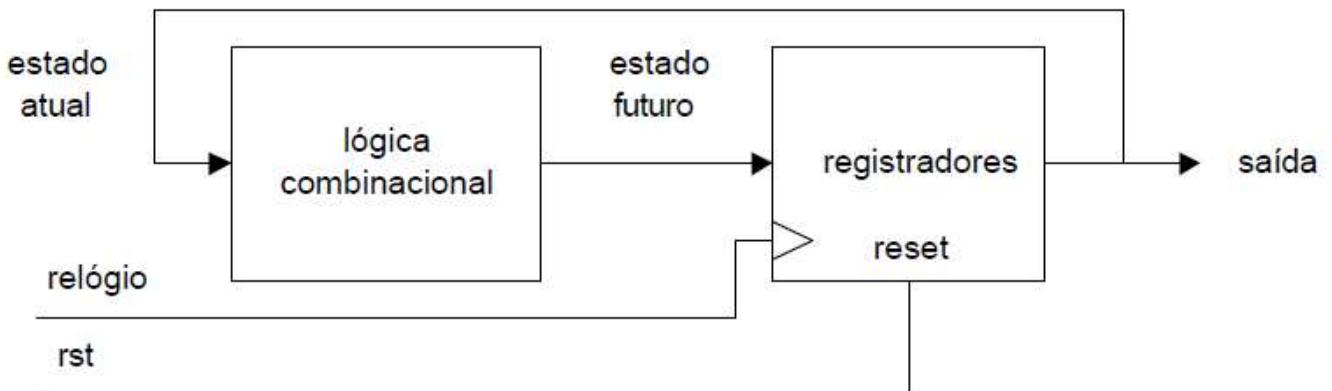
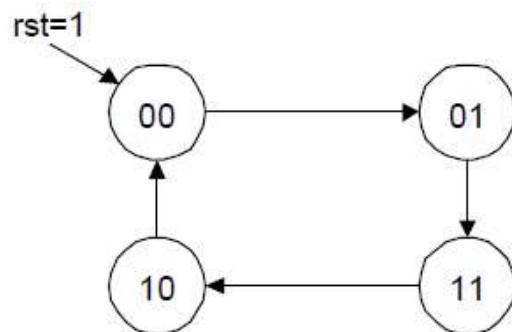
Transição de estados são permitidas apenas na transição de disparo  
do elemento de armazenamento de estado (transição positiva ou negativa)

Máquinas de Mealy, as entradas e saídas são definidas  
nos arcos (transições entre estados).

Máquina de Moore, as entradas são definidas nos arcos  
(transições entre estados) e a saída é definida no estado  
(dentro)

# Máquina de estado finito

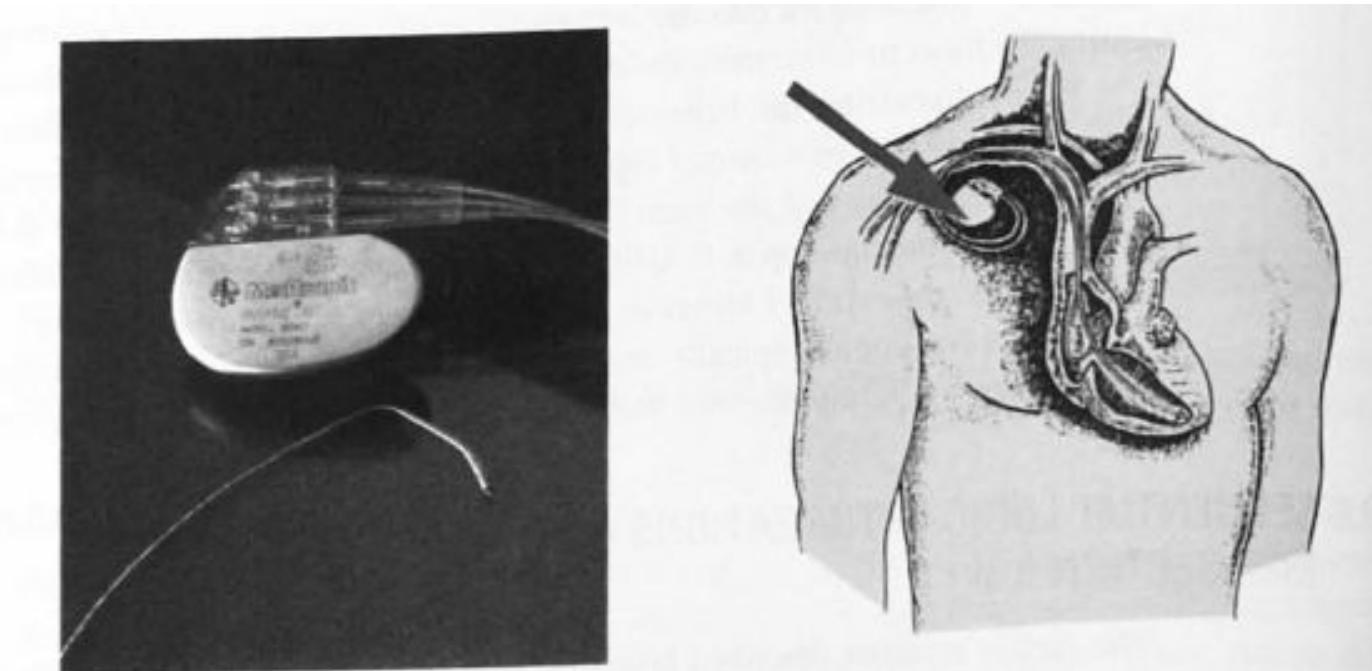
- Contador é um caso particular de uma máquina de estados
- Saída dos registradores corresponde ao valor de saída
- código do estado = valor de saída



# Máquina de estado finito

```
1 ENTITY maq_est1 IS
2     PORT (ck      : IN      BIT;                      -- relogio borda subida
3            rst      : IN      BIT;                      -- rst=1, q=00
4            q       : BUFFER BIT_VECTOR (1 DOWNTO 0)); -- saida codigo Gray
5 END maq_est1;
6
7 ARCHITECTURE teste OF maq_est1 IS
8
9 BEGIN
10    abc: PROCESS (ck, rst)
11    BEGIN
12        IF rst = '1' THEN                                -- estado inicial
13            q <= "00";
14        ELSIF (ck'EVENT and ck ='1') THEN               -- ciclo de estados
15            CASE q IS
16                WHEN "00" => q <= "01";
17                WHEN "01" => q <= "11";
18                WHEN "11" => q <= "10";
19                WHEN "10" => q <= "00";
20            END CASE;
21        END IF;
22    END PROCESS abc;
23 END teste;
```

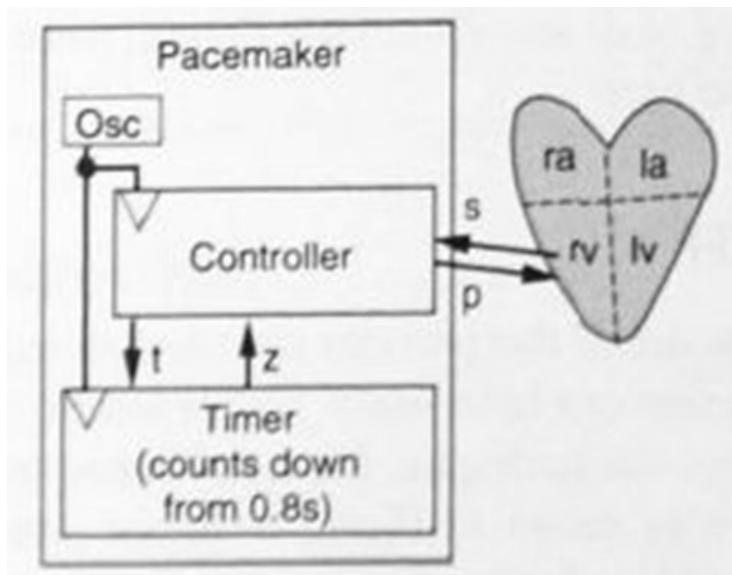
# Marca-passo Simples



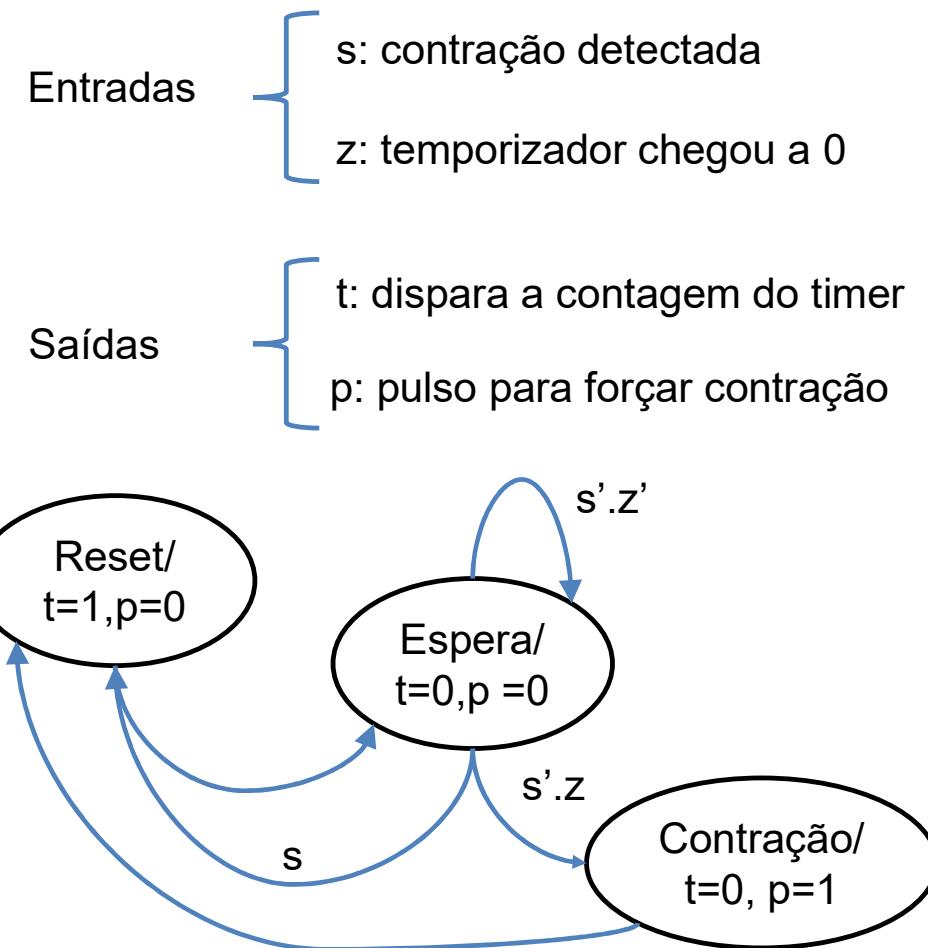
- Um marca-passo é um dispositivo eletrônico que fornece estímulos elétricos ao coração para ajudar a regular o seu batimento.

# Marca-Passo Simples

Vamos projetar a máquina de estados para o controlador do marca-passo



[Vahid ,2007]

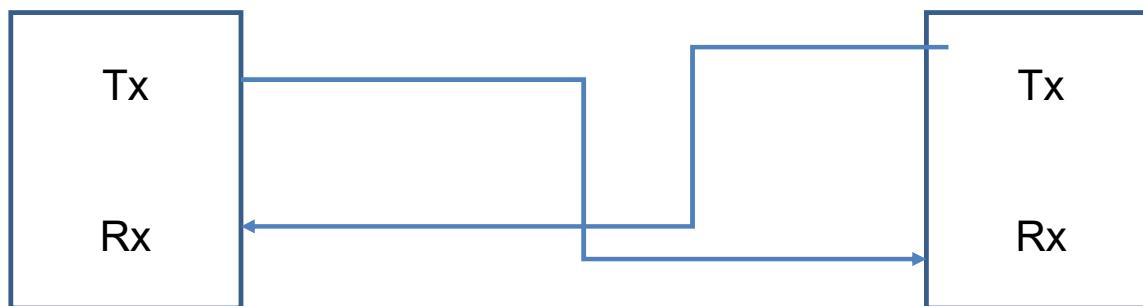


# UART

- Protocolo de comunicação serial
  - Microcontroladores
  - Computadores /Modems
  - FPGA
  - Universal Asynchronous Receiver/Transmitter

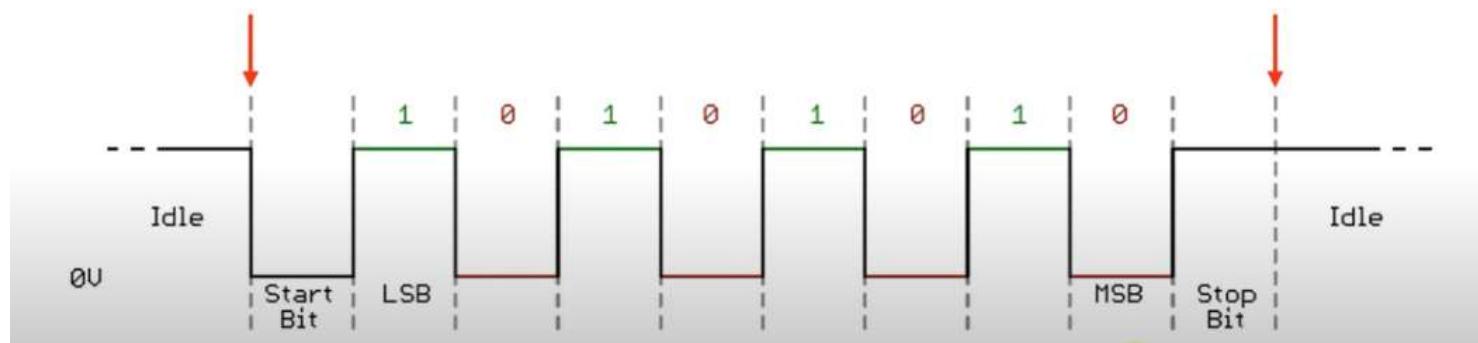
## Parâmetros

- Baud rate – 9600, 19200, 115200
- Tamanho da palavra : 5 a 9 bits
- Bit de paridade (opcional)
- Stop bits : 1 ou 2 bits
- Controle de Fluxo: Nenhum,Xon/Xoff, hardware

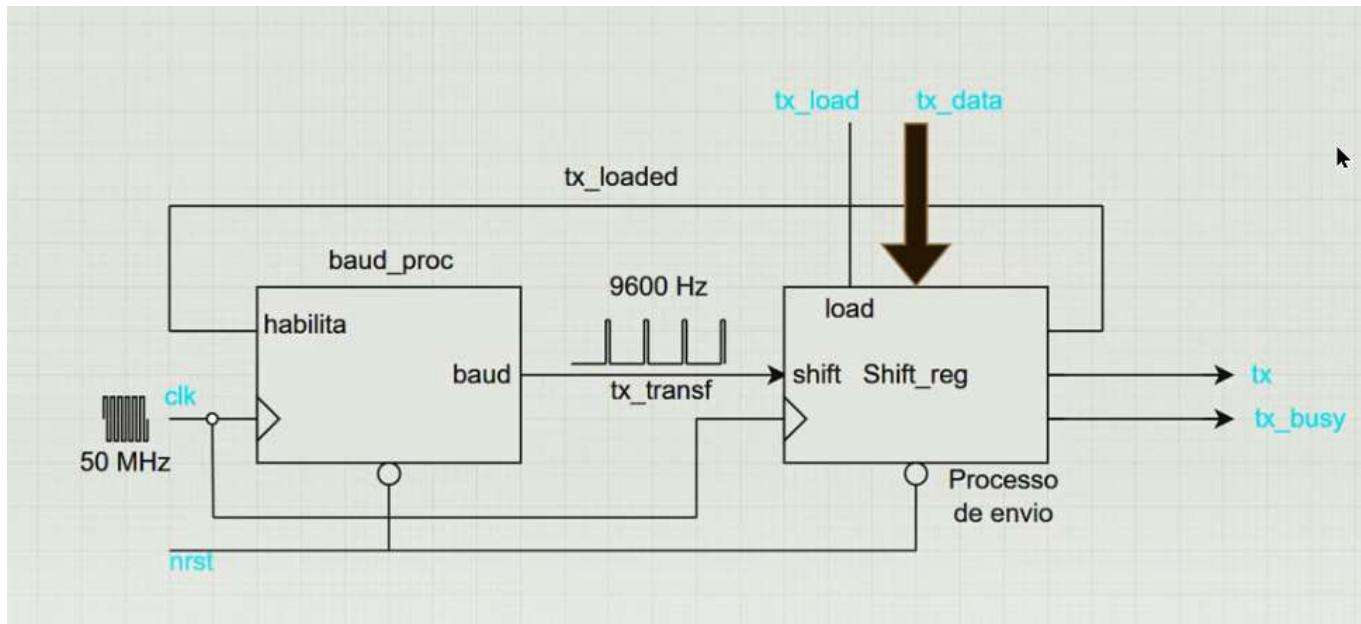


# UART Estrutura de um dado

<b>Start bit (1bit)</b>	<b>Datra Frame ( 5 a 9 bits)</b>	<b>Parity Bit (par ou ímpar)</b>	<b>Stop Bits (1 ou 2 bits)</b>
-----------------------------	----------------------------------	--------------------------------------	------------------------------------



# UART



# UART Implementação em VHDL

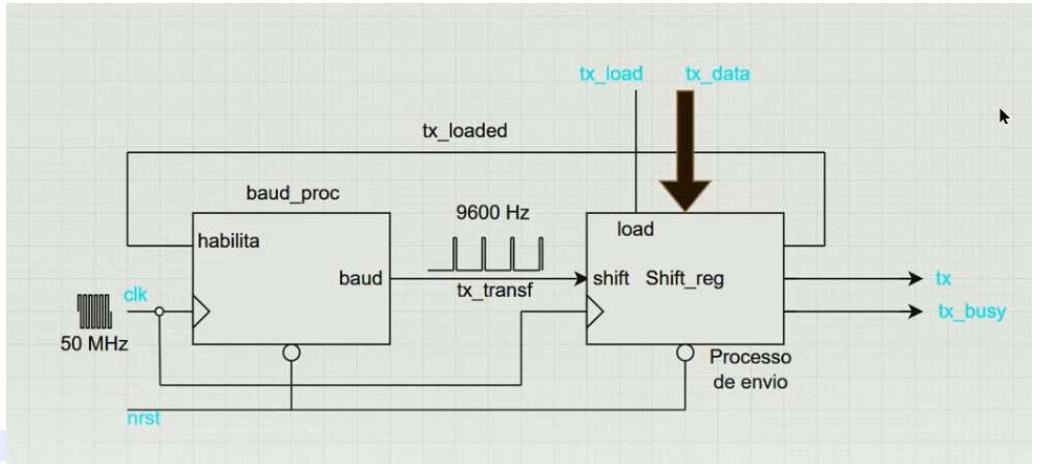
```
entity uart_tx is
  generic(
    baud           integer:= 9600;
    input_freq     integer:= 50e6
  );
  port (
    clk, nrst      : in      std_logic;
    tx             : out    std_logic; -- tx pin
    tx_data        : in      std_logic_vector(7 downto 0); -- data to be transferred
    tx_load        : in      std_logic; -- send data
    tx_busy        : out    std_logic
  );
end entity uart_tx;
```

# UART Implementação em VHDL

```
architecture behavoiral of uart_tx is
    constant      max_cycles : integer := input_freq/baud;
    signal shift_register : std_logic_vector(9 downto 0);
    signal tx_loaded      : boolean;
    signal tx_transf      : boolean;
```

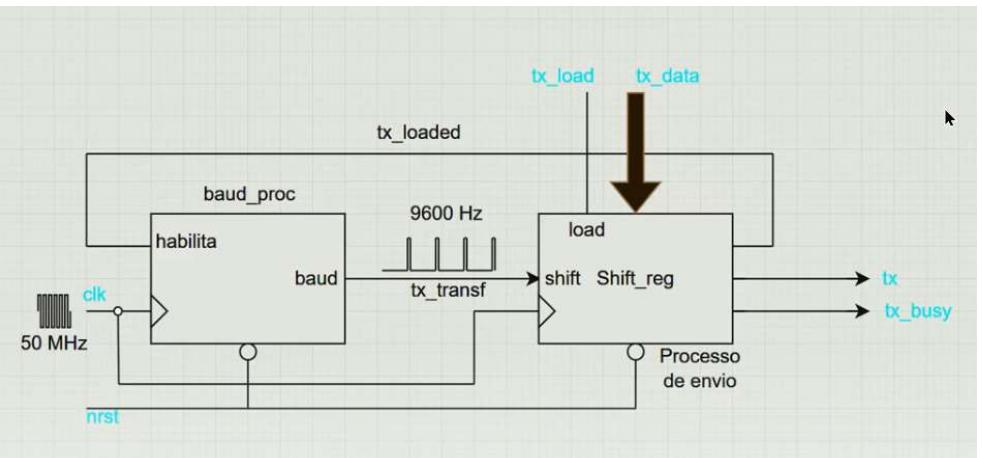
# UART Implementação em VHDL

```
tx_proc: process(clk,nrst)
  variable i : integer range 0 to 9;
begin
  if nrst = '0' then
    shift_register <= (others => '1');
    tx_loaded <= False;
    tx_busy <= '0';
    tx <= '1';
    i:=0;
  elsif rising_edge(clk) then
    if tx_load ='0' then
      shift_register <= '1' & tx_data & '0';
      tx_loaded <= True;
      tx_busy <= '1';
    elsif tx_loaded then
      if tx_transf then
        tx<= shift_register(0);
        shift_register <= '1' & shift_register(9 downto 1)
        if i = 9 then
          tx_loaded <= False;
          tx_busy <='0';
          tx<= '1';
          i:=0;
        else
          i := i+1;
        end if;
      end if;
    end if;
  end process tx_proc;
```



# UART Implementação em VHDL

```
[baud_proc: process (clk, nrst)
begin
    variable i : integer range 0 to max_cycles;
    if nrst = '0' then
        i := 0;
        tx_transf <= False;
    elsif rising_edge(clk) then
        if i = max_cycles -1 then
            i:=0;
            tx_transf <= True;
        elsif tx_loaded then
            i:= i + 1;
            tx_transf <= False;
        end if;
    end if;
end process baud_proc;
end behavoiral;
```



# Exemplos do Livro “Sistemas Digitais: Princípios e Aplicações”

# Contadores em VHDL

- Contador de módulo 5 em VHDL.

```
ENTITY fig7_41 IS
PORT (
    clock  :IN BIT;
    q      :OUT BIT_VECTOR(2 DOWNTO 0)
);
END fig7_41 ;

ARCHITECTURE a OF fig7_41 IS
BEGIN
    PROCESS (clock)                      -- responde à entrada clk
    VARIABLE count: BIT_VECTOR(2 DOWNTO 0); -- cria registrador de 3 bits
    BEGIN
        IF (clock = '1' AND clock'EVENT) THEN -- dispara borda de subida
            CASE count IS
                -- Present       Next
                -----
                WHEN "000"  => count := "001";
                WHEN "001"  => count := "010";
                WHEN "010"  => count := "011";
                WHEN "011"  => count := "100";
                WHEN "100"  => count := "000";
                WHEN OTHERS => count := "000";
            END CASE;
        END IF;
        q <= count;                         -- conecta registrador com os pinos de saída
    END PROCESS;
END a;
```

# Contadores em VHDL

```
1 ENTITY fig7_44 IS
2 PORT( clock  :IN BIT;
3       q      :OUT INTEGER RANGE 0 TO 7 );
4 END fig7_44;
5
6 ARCHITECTURE a OF fig7_44 IS
7 BEGIN
8     PROCESS (clock)
9     VARIABLE count: INTEGER RANGE 0 to 7;--define uma VARIABLE numérica
10    BEGIN
11        IF (clock = '1' AND clock'EVENT) THEN      -- borda de subida?
12            IF count < 4 THEN                  -- inferior ao máximo?
13                count := count + 1;      -- incrementa valor
14            ELSE                            -- deve ser igual ou maior que o máximo
15                count := 0;                  -- recicla para zero
16            END IF;
17        END IF;
18        q <= count; -- transfere conteúdo do registrador para as saídas
19    END PROCESS;
20 END a;
```

]

**FIGURA 7.44** Descrição comportamental de um contador em VHDL.

# Contadores em VHDL

```
1 ENTITY fig7_47 IS
2 PORT( clock, clear, load, cntenabl, down      :IN BIT;
3       din                                :IN INTEGER RANGE 0 TO 15;
4       q                                 :OUT INTEGER RANGE 0 TO 15;
5       term_ct                           :OUT BIT);
6 END fig7_47;
7
8 ARCHITECTURE a OF fig7_47 IS
9 BEGIN
10    PROCESS (clock, clear, down)
11    VARIABLE count :INTEGER RANGE[ 0 to 15; -- define sinal numérico
12    BEGIN
13       IF clear = '1' THEN      count := 0;      -- clear assíncrono
14       ELSIF (clock = '1' AND clock'EVENT) THEN-- borda de subida?
15          IF load = '1' THEN      count := din;   -- Carga paralela
16          ELSIF cntenabl = '1' THEN           -- habilitada?
17             IF down = '0' THEN count := count + 1; -- incrementa
18             ELSE                  count := count - 1; -- decrementa
19             END IF;
20          END IF;
21       END IF;
22       IF (((count = 0) AND (down = '1')) OR
23           ((count = 15) AND (down = '0'))) AND cntenabl = '1'
24          THEN term_ct <= '1';
25          ELSE   term_ct <= '0';
26       END IF;
27       q <= count; --transfere conteúdo do registrador para saídas
28    END PROCESS;
29 END a;
```

FIGURA 7.47 Contador com recursos completos em VHDL.

# Contadores em VHDL

```
1 ENTITY decode5 IS
2 PORT (
3     c, b, a : IN BIT;
4     state    : OUT BIT_VECTOR (0 TO 4)
5 );
6 END decode5;
7
8 ARCHITECTURE a OF decode5 IS
9 SIGNAL input : BIT_VECTOR (2 DOWNTO 0);
10 BEGIN
11     input <= (c & b & a); -- combina entradas em vetor de bits
12     PROCESS (c, b, a)
13     BEGIN
14         CASE input IS
15             WHEN "000" => state <= "10000";
16             WHEN "001" => state <= "01000";
17             WHEN "010" => state <= "00100";
18             WHEN "011" => state <= "00010";
19             WHEN "100" => state <= "00001";
20             WHEN OTHERS => state <= "00000";
21         END CASE;
22     END PROCESS;
23 END a;
```

FIGURA 7.52 Módulo de decodificador de contador de módulo 5 em VHDL.

# Contadores em VHDL

```
1  ENTITY mod100 IS
2      PORT (
3          clk, en, clr           :IN BIT;
4          ones                  :OUT INTEGER RANGE 0 TO 15;
5          tens                  :OUT INTEGER RANGE 0 TO 15;
6          max                   :OUT BIT
7      );
8  END mod100;
9  ARCHITECTURE toplevel OF mod100 IS
10     COMPONENT mod10
11         PORT (
12             clock, enable, clear :IN BIT;
13             q                  :OUT INTEGER RANGE 0 TO 15;
14             tc                 :OUT BIT
15         );
16     END COMPONENT;
17     SIGNAL rco               :BIT;
18 BEGIN
19     digit1: mod10 PORT MAP (clock => clk, enable => en,
20                             clear => clr, q => ones, tc => rco);
21     digit2: mod10 PORT MAP (clock => clk, enable => rco,
22                             clear => clr, q => tens, tc => max);
23 END toplevel;
24
25
26 ENTITY mod10 IS
27     PORT (
28         clock, enable, clear   :IN BIT;
29         q                      :OUT INTEGER RANGE 0 TO 15;
30         tc                     :OUT BIT
31     );
32 END mod10;
33 ARCHITECTURE lowerblk OF mod10 IS
34 BEGIN
35     PROCESS (clock, enable)
36         VARIABLE counter       :INTEGER RANGE 0 TO 15;
37     BEGIN
38         IF ((counter = 9) AND (enable = '1')) THEN tc <= '1';
39         ELSE tc <= '0';
40         END IF;
41         IF (clock'EVENT AND clock = '1') THEN
42             IF (clear = '1') THEN counter := 0;
43             ELSIF (enable = '1') THEN
44                 IF (counter = 9) THEN counter := 0;
45                 ELSE counter := counter + 1;
46                 END IF;
47             END IF;
48         END IF;
49         q <= counter;
50     END PROCESS;
51 END lowerblk;
```

FIGURA 7.58 Contador BCD de módulo 100 em VHDL.

# Contadores em VHDL

```
1  ENTITY mod5decoded1 IS
2    PORT (
3      clk          : IN BIT;
4      q           : BUFFER BIT_VECTOR (2 DOWNTO 0);
5      cntr_state : OUT BIT_VECTOR (0 TO 4)
6    );
7  END mod5decoded1;
8
9  ARCHITECTURE toplevel OF mod5decoded1 IS
10   COMPONENT mod5
11     PORT (
12       clock      : IN BIT;
13       q         : OUT BIT_VECTOR (2 DOWNTO 0)
14     );
15   END COMPONENT;
16   COMPONENT decode5
17     PORT (
18       c, b, a : IN BIT;
19       state   : OUT BIT_VECTOR (0 TO 4)
20     );
21   END COMPONENT;
22   BEGIN
23     counter: mod5    PORT MAP (clock => clk, q => q);
24     decoder: decode5 PORT MAP
25       (c => q(2), b => q(1), a => q(0), state => cntr_state);
26   END toplevel;
```

FIGURA 7.53 Arquivo em VHDL de nível mais alto para conectar os módulos mod5 e decode5.

# Máquina de Estado

```
1 ENTITY traffic IS
2 PORT ( clock, car, reset      :IN BIT;
3          tmaingrn, tsidegrn   :IN INTEGER RANGE 0 TO 31;
4          lite                 :BUFFER INTEGER RANGE 0 TO 3;
5          change               :BUFFER BIT;
6          mainred, mainyelo, maingrn :OUT BIT;
7          sidered, sideyelo, sidegrn :OUT BIT);
8 END traffic;
9 ARCHITECTURE toplevel OF traffic IS
10 COMPONENT delay
11    PORT ( clock, car, reset      :IN BIT;
12          lite                 :IN INTEGER RANGE 0 TO 3;
13          tmaingrn, tsidegrn   :IN INTEGER RANGE 0 TO 31;
14          change               :OUT BIT);
15 END COMPONENT;
16 COMPONENT control
17    PORT ( clock, enable, reset   :IN BIT;
18          lite                 :OUT INTEGER RANGE 0 TO 3);
19 END COMPONENT;
20 COMPONENT lite_ctrl
21    PORT ( lite                :IN INTEGER RANGE 0 TO 3;
22          mainred, mainyelo, maingrn :OUT BIT;
23          sidered, sideyelo, sidegrn :OUT BIT);
24 END COMPONENT;
25 BEGIN
26 module1: delay    PORT MAP (clock => clock, car => car, reset => reset,
27                             lite => lite, tmaingrn => tmaingrn,
28                             tsidegrn => tsidegrn, change => change);
29 module2: control   PORT MAP (clock => clock, enable => change,
30                             reset => reset, lite => lite);
31 module3: lite_ctrl PORT MAP (lite => lite, mainred => mainred, mainyelo => mainyelo,
32                             maingrn => maingrn, sidered => sidered,
33                             sideyelo => sideyelo, sidegrn => sidegrn);
34 END toplevel;
35 -----
36 ENTITY delay IS
37 PORT ( clock, car, reset      :IN BIT;
38          lite                 :IN BIT_VECTOR (1 DOWNTO 0);
39          tmaingrn, tsidegrn   :IN INTEGER RANGE 0 TO 31;
40          change               :OUT BIT);
41 END delay;
```

# Máquina de Estado

```
42 ARCHITECTURE time OF delay IS
43 BEGIN
44 PROCESS (clock, reset)
45 VARIABLE mach :INTEGER RANGE 0 TO 31;
46 BEGIN
47 IF reset = '0' THEN mach := 0;
48 ELSIF (clock = '1' AND clock'EVENT) THEN -- com clock de 1 Hz, tempos em segundos
49 CASE lite IS
50 WHEN "00"
51     IF car = '0' THEN mach := 0;      -- espera carro na via secundária
52     ELSE mach := tmaingrn - 1;      -- fixa tempo para verde na principal
53     END IF;
54 WHEN "01" => mach := 5 - 1;        -- fixa tempo para amarelo na principal
55 WHEN "10" => mach := tsidegrn - 1; -- fixa tempo para verde na secundária
56 WHEN "11" => mach := 5 - 1;        -- fixa tempo para amarelo na secundária
57 END CASE;
58 ELSE mach := mach - 1;            -- decrementa temporizador
59 END IF;
60 END IF;
61 IF mach = 1 THEN change <= '1';    -- muda luzes em control
62 ELSE change <= '0';
63 END IF;
64 END PROCESS;
65 END time;
66 -----
67 ENTITY control IS
68 PORT ( clock, enable, reset :IN BIT;
69         lite :OUT BIT_VECTOR (1 DOWNTO 0));
70 END control;
71 ARCHITECTURE a OF control IS
72 TYPE enumerated IS (mgnr, myel, sgrn, syel); -- necessita de 4 estados para combinações de luz
73 BEGIN
74 PROCESS (clock, reset)
75 VARIABLE lights :enumerated;
76 BEGIN
77     IF reset = '0' THEN lights := mgnr;
78     ELSIF (clock = '1' AND clock'EVENT) THEN
79         IF enable = '1' THEN                      -- espera que enable mude os estados da luz
80             CASE lights IS
81                 WHEN mgnr => lights := myel;
82                 WHEN myel => lights := sgrn;
83                 WHEN sgrn => lights := syel;
84                 WHEN syel => lights := mgnr;
85             END CASE;
86         END IF;
87     END IF;
88     CASE lights IS                         -- padrões para os estados da luz
89         WHEN mgnr=> lite <= "00";
90         WHEN myel=> lite <= "01";
91         WHEN sgrn=> lite <= "10";
92         WHEN syel=> lite <= "11";
93     END CASE;
94 END PROCESS;
95 END a;
96 -----
97 ENTITY lite_ctrl IS
98 PORT ( lite :IN BIT_VECTOR (1 DOWNTO 0);
99 END lite_ctrl;
```

# Máquina de Estado

```
100      mainred, mainyelo, maingrn :OUT BIT;
101      sidered, sideyelo, sidegrn :OUT BIT);
102  END lite_ctrl;
103  ARCHITECTURE patterns OF lite_ctrl IS
104  BEGIN
105    PROCESS (lite)
106    BEGIN
107      CASE lite IS
108        WHEN "00" => maingrn <= '1'; mainyelo <= '0'; mainred <= '0';
109          sidegrn <= '0'; sideyelo <= '0'; sidered <= '1';
110        WHEN "01" => maingrn <= '0'; mainyelo <= '1'; mainred <= '0';
111          sidegrn <= '0'; sideyelo <= '0'; sidered <= '1';
112        WHEN "10" => maingrn <= '0'; mainyelo <= '0'; mainred <= '1';
113          sidegrn <= '1'; sideyelo <= '0'; sidered <= '0';
114        WHEN "11" => maingrn <= '0'; mainyelo <= '0'; mainred <= '1';
115          sidegrn <= '0'; sideyelo <= '1'; sidered <= '0';
116      END CASE;
117    END PROCESS;
118  END patterns;
```

FIGURA 7.65 (continuação) Projeto de controlador de farol de trânsito em VHDL.

# Registrador de Entrada Paralela e saída serial em VHDL

```
1 ENTITY fig7_86 IS
2 PORT ( clk, shift, serial_in    :IN BIT;
3        serial_out           :OUT BIT );
4 END fig 7-86;
5 ARCHITECTURE vhdl OF fig 7-86 IS
6 BEGIN
7 PROCESS (clk)
8   VARIABLE q          :BIT_VECTOR (3 DOWNTO 0);
9   BEGIN
10    serial_out <= q(0);           -- leva à saída último bit do registrador
11    IF (clk'EVENT AND clk = '1') THEN
12      IF (shift = '1') THEN
13        q := (serial_in & q(3 DOWNTO 1)); -- concatena para deslocamento
14      END IF;                   -- caso contrário, guarda dados
15    END IF;
16  END PROCESS;
17 END vhdl;
```

FIGURA 7.86 Registrador de entrada serial/saída serial em VHDL.

# Registrador de Entrada Paralela e saída serial em VHDL

```
1  ENTITY fig7_89 IS
2    PORT (
3      clk, shift, load : IN BIT;
4      data           : IN BIT_VECTOR (3 DOWNTO 0);
5      serial_out      : OUT BIT
6    );
7  END fig 7-89;
8  ARCHITECTURE vhdl OF fig 7-89 IS
9  BEGIN
10    PROCESS (clk)
11      VARIABLE q :BIT_VECTOR (3 DOWNTO 0);
12    BEGIN
13      serial_out <= q(0);          -- leva à saída último bit do registrador
14      IF (clk'EVENT AND clk = '1') THEN
15        IF (load = '1') THEN q := data;          -- carga paralela
16        ELSIF (shift = '1') THEN q := ('0' & q(3 DOWNTO 1)); -- deslocamento
17        END IF;                                -- caso contrário, guarda
18    END IF;
19  END PROCESS;
20 END vhdl;
```

FIGURA 7.89 Registrador de entrada paralela/saída serial em VHDL.

# Contador em anel em VHDL

```
1  ENTITY fig7_94 IS
2    PORT ( clk :IN BIT;
3           q   :OUT BIT_VECTOR (3 DOWNTO 0));
4  END fig7_94;
5
6  ARCHITECTURE vhdl OF fig7_94 IS
7  SIGNAL ser_in :BIT;
8  BEGIN
9  PROCESS (clk)
10    VARIABLE ff :BIT_VECTOR (3 DOWNTO 0);
11    BEGIN
12      IF (ff(3 DOWNTO 1) = "000") THEN ser_in <= '1'; -- autoinício
13      ELSE ser_in <= '0';
14      END IF;
15      IF (clk'EVENT AND clk = '1') THEN
16        ff(3 DOWNTO 0) := (ser_in & ff(3 DOWNTO 1)); -- deslocamento para a direita
17      END IF;
18      q <= ff;
19    END PROCESS;
20  END vhdl;
```

FIGURA 7.94 Contador de quatro bits em anel em VHDL.

# Monoestáveis simples em VHDL

```
1 ENTITY fig7_96 IS
2 PORT (
3     clock, trigger, reset :IN BIT;
4     delay                 :IN INTEGER RANGE 0 TO 15;
5     q                     :OUT BIT
6 );
7 END fig 7_96;
8
9 ARCHITECTURE vhdl OF fig7_96 IS
10 BEGIN
11     PROCESS (clock, reset)
12     VARIABLE count : INTEGER RANGE 0 TO 15;
13     BEGIN
14         IF reset = '0' THEN count := 0;
15         ELSIF (clock'EVENT AND clock = '1' ) THEN
16             IF trigger = '1' AND count = 0 THEN
17                 count := delay;          -- carrega contador
18             ELSIF count = 0 THEN count := 0;
19             ELSE count := count - 1;
20             END IF;
21         END IF;
22         IF count /= 0 THEN q <= '1';
23         ELSE q <= '0';
24         END IF;
25     END PROCESS;
26 END vhdl;
```

FIGURA 7.96 Monoestável não redispersável em VHDL.

# Monoestáveis redisparáveis por borda em VHDL

```
1 ENTITY fig7_100 IS
2 PORT ( clock, trigger, reset : IN BIT;
3         delay                 : IN INTEGER RANGE 0 TO 15;
4         q                      : OUT BIT);
5 END fig7_100;
6
7 ARCHITECTURE vhdl OF fig7_100 IS
8 BEGIN
9     PROCESS (clock, reset)
10    VARIABLE count : INTEGER RANGE 0 TO 15;
11    VARIABLE trig_was : BIT;
12    BEGIN
13        IF reset = '0' THEN count := 0;
14        ELSIF (clock'EVENT AND clock = '1' ) THEN
15            IF trigger = '1' AND trig_was = '0' THEN
16                count := delay;           -- carrega contador
17                trig_was := '1';         -- 'lembra' a borda detectada
18            ELSIF count = 0 THEN count := 0; -- guarda @ 0
19            ELSE count := count - 1;   -- decrementa
20            END IF;
21            IF trigger = '0' THEN trig_was := '0';
22            END IF;
23        END IF;
24        IF count /= 0 THEN q <= '1';
25        ELSE q <= '0';
26        END IF;
27    END PROCESS;
28 END vhdl;
```

FIGURA 7.100 Monoestável redisparável com disparo por borda em VHDL.

# Decodificadores em VHDL

```
1 ENTITY fig9_54 IS
2 PORT(
3     a                  :IN BIT_VECTOR (2 DOWNTO 0);
4     e3, e2bar, elbar :IN BIT;
5     y                  :OUT BIT_VECTOR (7 DOWNTO 0)
6 );
7 END fig9_54 ;
8 ARCHITECTURE truth OF fig9_54 IS
9 SIGNAL inputs: BIT_VECTOR (5 DOWNTO 0); -- combina habilitações a entrada binária
10 BEGIN
11     inputs <= e3 & e2bar & elbar & a;
12     WITH inputs SELECT
13         y <= "11111110" WHEN "100000", -- Y0 ativa
14             "11111101" WHEN "100001", -- Y1 ativa
15             "11111011" WHEN "100010", -- Y2 ativa
16             "11110111" WHEN "100011", -- Y3 ativa
17             "11101111" WHEN "100100", -- Y4 ativa
18             "11011111" WHEN "100101", -- Y5 ativa
19             "10111111" WHEN "100110", -- Y6 ativa
20             "01111111" WHEN "100111", -- Y7 ativa
21             "11111111" WHEN OTHERS;   -- desabilitadas
22 END truth;
```

FIGURA 9.54 Equivalente em VHDL ao decodificador 74138.

# Decodificador /driver em VHDL

```
1  ENTITY fig9_56 IS
2    PORT (
3      bcd          :IN INTEGER RANGE 0 TO 15;
4      lt, bi, rbi :IN BIT;
5      a,b,c,d,e,f,g,rbo :OUT BIT
6      );
7  END fig9_56 ;
8
9  ARCHITECTURE vhdl OF fig9_56 IS
10 BEGIN
11   PROCESS (bcd, lt, bi, rbi)
12   VARIABLE segments      :BIT_VECTOR (0 TO 6);
13   BEGIN
14     IF bi = '0' THEN
15       segments := "1111111"; rbo <= '0'; -- apaga tudo
16     ELSIF lt = '0' THEN
17       segments := "0000000"; rbo <= '1'; -- testa segmentos
18     ELSIF (rbi = '0' AND bcd = 0) THEN
19       segments := "1111111"; rbo <= '0'; -- apaga Os iniciais
```

FIGURA 9.56 Decodificador-display BCD para 7 segmentos em VHDL (*continua*).

# Decodificador /driver em VHDL

```
20      ELSE
21          rbo <= '1';
22          CASE bcd IS    -- display padrão anodo comum para 7 segmentos
23              WHEN      0 => segments := "0000001";
24              WHEN      1 => segments := "1001111";
25              WHEN      2 => segments := "0010010";
26              WHEN      3 => segments := "0000110";
27              WHEN      4 => segments := "1001100";
28              WHEN      5 => segments := "0100100";
29              WHEN      6 => segments := "1100000";
30              WHEN      7 => segments := "0001111";
31              WHEN      8 => segments := "0000000";
32              WHEN      9 => segments := "0001100";
33              WHEN OTHERS => segments := "1111111";
34          END CASE;
35      END IF;
36      a <= segments(0);      -- atribui bits de vetor a pinos de saída
37      b <= segments(1);
38      c <= segments(2);
39      d <= segments(3);
40      e <= segments(4);
41      f <= segments(5);
42      g <= segments(6);
43  END PROCESS;
44 END vhdl;
```

FIGURA 9.56 (continuação) Decodificador-display BCD para 7 segmentos em VHDL.

# Codificador com Prioridade em VHDL

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3
4 ENTITY fig9_60 IS
5 PORT(
6     sw :IN BIT_VECTOR (9 DOWNTO 0);      -- lógica IEEE padrão não é necessária
7     oe :IN BIT;                      -- lógica IEEE padrão não é necessária
8     d :OUT STD_LOGIC_VECTOR (3 DOWNTO 0)-- lógica IEEE padrão para alta impedância
9 );
10 END fig9_60;
11
12 ARCHITECTURE a OF fig9_60 IS
13 BEGIN
14     d <= "2222" WHEN ((oe = '0') OR (sw = "1111111111")) ELSE
15         "1001" WHEN sw(9) = '0' ELSE
16         "1000" WHEN sw(8) = '0' ELSE
17         "0111" WHEN sw(7) = '0' ELSE
18         "0110" WHEN sw(6) = '0' ELSE
19         "0101" WHEN sw(5) = '0' ELSE
20         "0100" WHEN sw(4) = '0' ELSE
21         "0011" WHEN sw(3) = '0' ELSE
22         "0010" WHEN sw(2) = '0' ELSE
23         "0001" WHEN sw(1) = '0' ELSE
24         "0000" WHEN sw(0) = '0';
25 END a;
```

FIGURA 9.60 Codificador de prioridade usando atribuição de sinal condicional em VHDL.

# MUX e Demux em VHDL

```
1 ENTITY fig9_64 IS
2 PORT (
3     ch0, ch1, ch2, ch3 :IN BIT_VECTOR (3 DOWNTO 0);
4     s                 :IN INTEGER RANGE 0 TO 3;
5     dout              :OUT BIT_VECTOR (3 DOWNTO 0)
6 );
7 END fig9_64;
8
9 ARCHITECTURE selecter OF fig9_64 IS
10 BEGIN
11     WITH s SELECT
12         dout <= ch0 WHEN 0, -- seleciona o canal 0 para a saída
13             ch1 WHEN 1, -- seleciona o canal 1 para a saída
14             ch2 WHEN 2, -- seleciona o canal 2 para a saída
15             ch3 WHEN 3; -- seleciona o canal 3 para a saída
16 END selecter;
```

FIGURA 9.64 MUX de quatro bits x quatro canais emVHDL.

```
1 ENTITY fig9_65 IS
2 PORT (
3     s                 :IN INTEGER RANGE 0 TO 3;
4     din              :IN BIT_VECTOR (3 DOWNTO 0);
5     ch0, ch1, ch2, ch3 :OUT BIT_VECTOR(3 DOWNTO 0)
6 );
7 END fig9_65;
8
9 ARCHITECTURE selecter OF fig9_65 IS
10 BEGIN
11     ch0 <= din WHEN s = 0 ELSE "1111";
12     ch1 <= din WHEN s = 1 ELSE "1111";
13     ch2 <= din WHEN s = 2 ELSE "1111";
14     ch3 <= din WHEN s = 3 ELSE "1111";
15 END selecter;
```

FIGURA 9.65 DEMUX de quatro bits x quatro canais emVHDL.