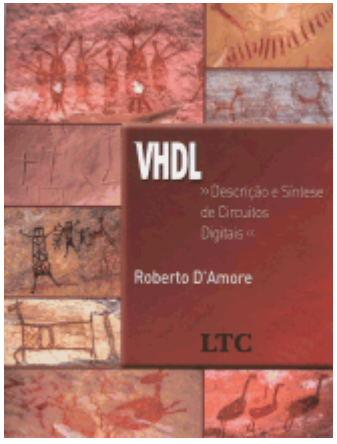
Demonstração de imagens de auxílio didático

VHDL - Descrição e Síntese de Circuitos Digitais

Roberto d'Amore

ISBN 85-216-1452-7
Editora LTC www.ltceditora.com.br



Para imagens de um curso completo consulte: www.ele.ita.br/~damore/vhdl

Tópicos

- Aspectos gerais da linguagem
- Síntese de circuitos
- Entidade de projeto
- Classes de objetos: constante, variável e sinal
- Tipos
- Operadores
- Construção concorrente WHEN ELSE
- Construção concorrente WITH SELECT
- Processos e lista de sensibilidade
- Construção seqüencial IF ELSE
- Construção seqüencial CASE WHEN
- Circuitos síncronos

Aspectos gerais da linguagem

- Suporta diversos níveis de hierarquia
  - uma descrição pode ser: conjunto de descrições interligadas

- Estilo de uma descrição
  - diversos níveis de abstração são suportados
  - pode conter descrições com diferentes níveis de abstração

- Ferramentas de síntese:
  - suportam diferentes estilos de descrição
  - normalmente: modos preferenciais devem ser empregados

- Linguagem concorrente
  - ordem dos comandos: não importa
  - comandos seqüenciais: somente em regiões específicas

Síntese de circuitos

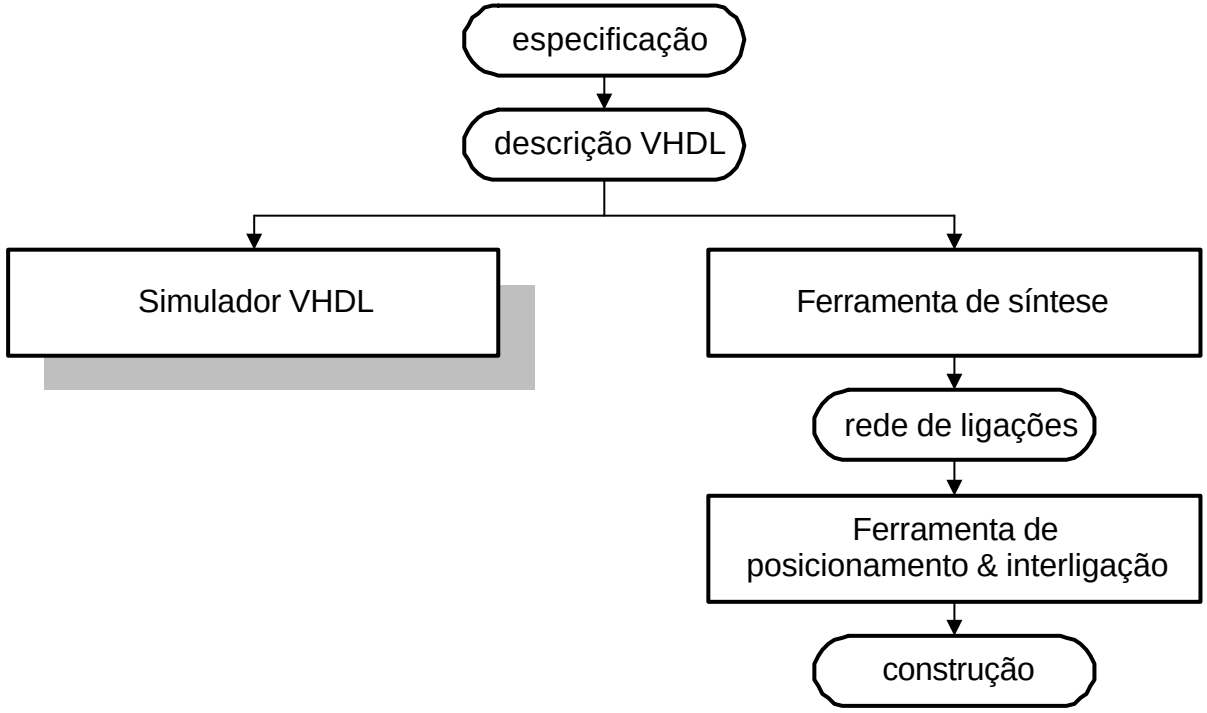
- VHDL: não foi concebida para síntese de circuitos
  - conseqüência: nem todas construções são suportadas

- Motivos da limitação:
  - falta de correspondência da construção / descrição com um circuito
    exemplo: *flip flop* com dois terminais de relógio
  - impossibilidade da síntese direta
    exemplo: multiplicação de dois números reais

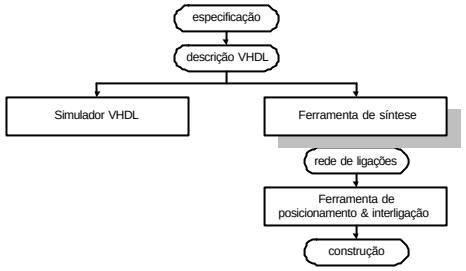
Etapas gerais de um processo de síntese

Simulador VHDL:

- compilação / simulação do código



Ferramenta de síntese

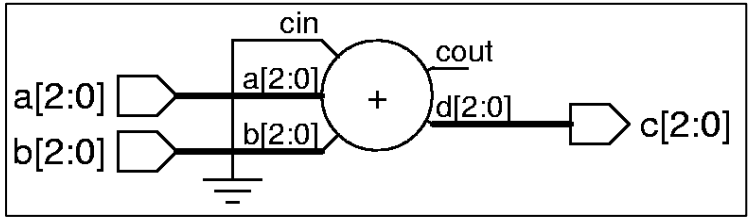


- Síntese da descrição - (ilustração das operações)
  - descrição do circuito
  - circuito nível RTL sintetizado
  - circuito nível portas (final)

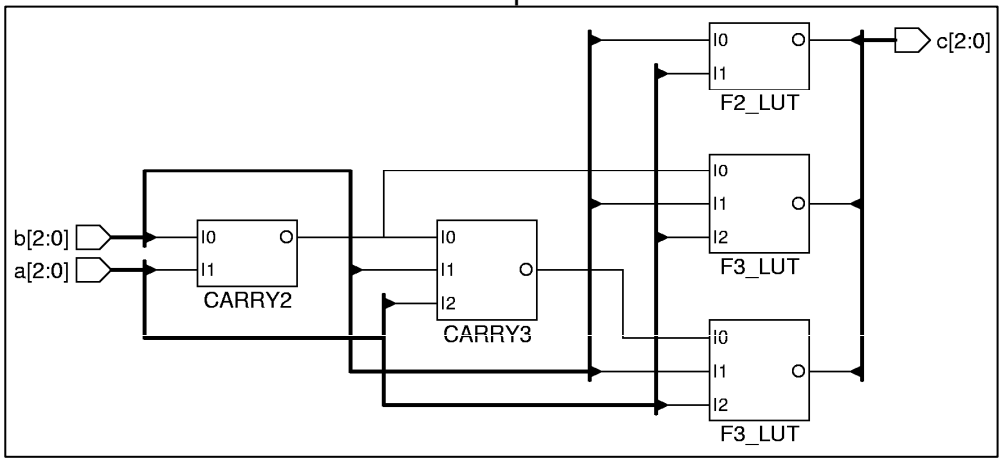
descrição VHDL

```
ENTITY soma IS
  PORT (a, b : IN  INTEGER RANGE 7 DOWNTO 0;
        c    : OUT INTEGER RANGE 7 DOWNTO 0);
END soma;

ARCHITECTURE teste OF soma IS
BEGIN
  c <= a+b;
END teste;
```

nível RTL



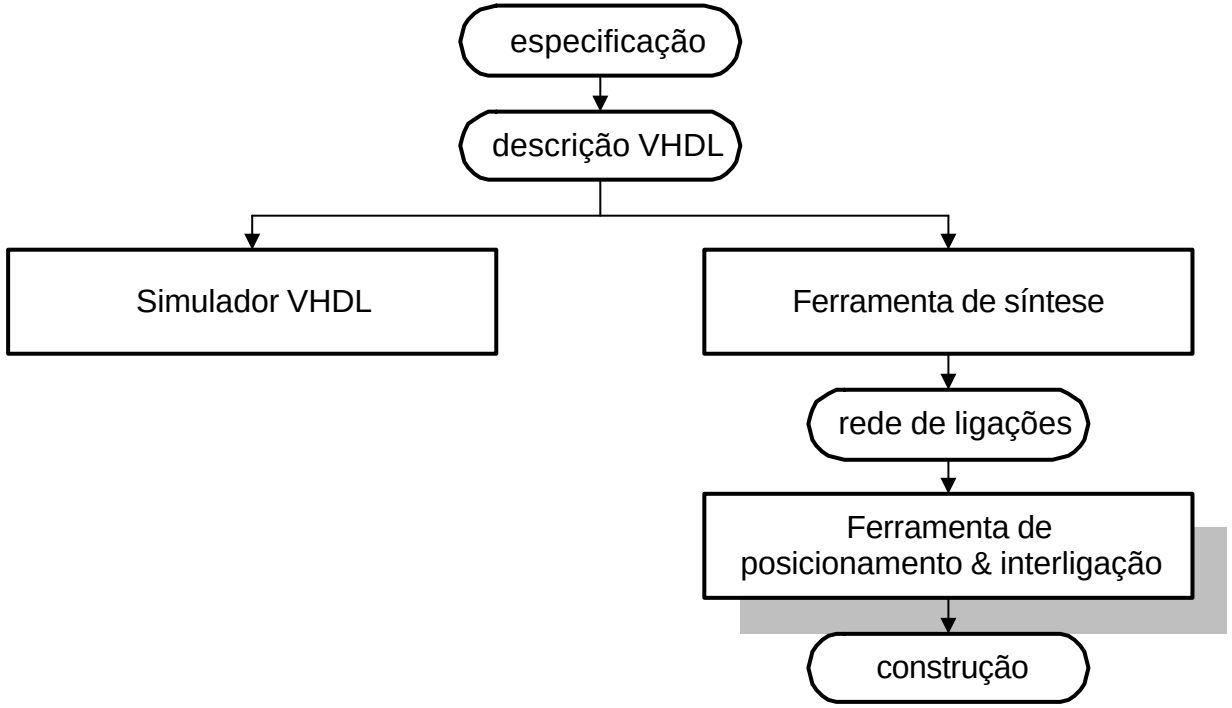
nível portas



otimização velocidade / área

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



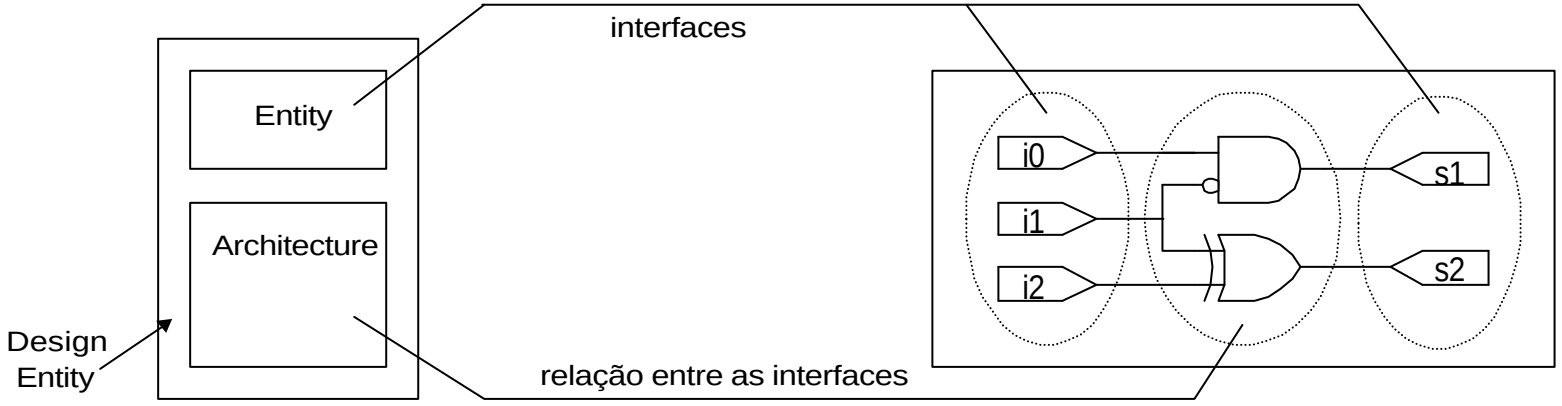
Primeiro contato com a linguagem

Entidade de projeto

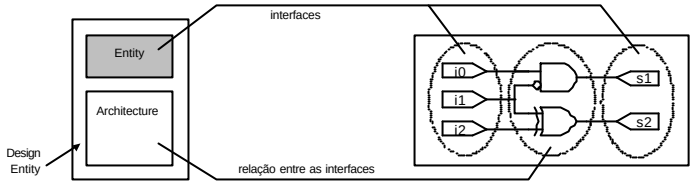
- Pode representar:

  uma simples porta lógica ..... a um sistema completo

- Composta de duas partes:
  - Declaração da entidade
    - define portas de entrada e saída da descrição
      (equivalente ao símbolo de um bloco em captura esquemática)
  - Arquitetura
    - descreve as relações entre as portas
      (equivalente ao esquema contido no bloco em cap. esquemática)



Declaração da entidade

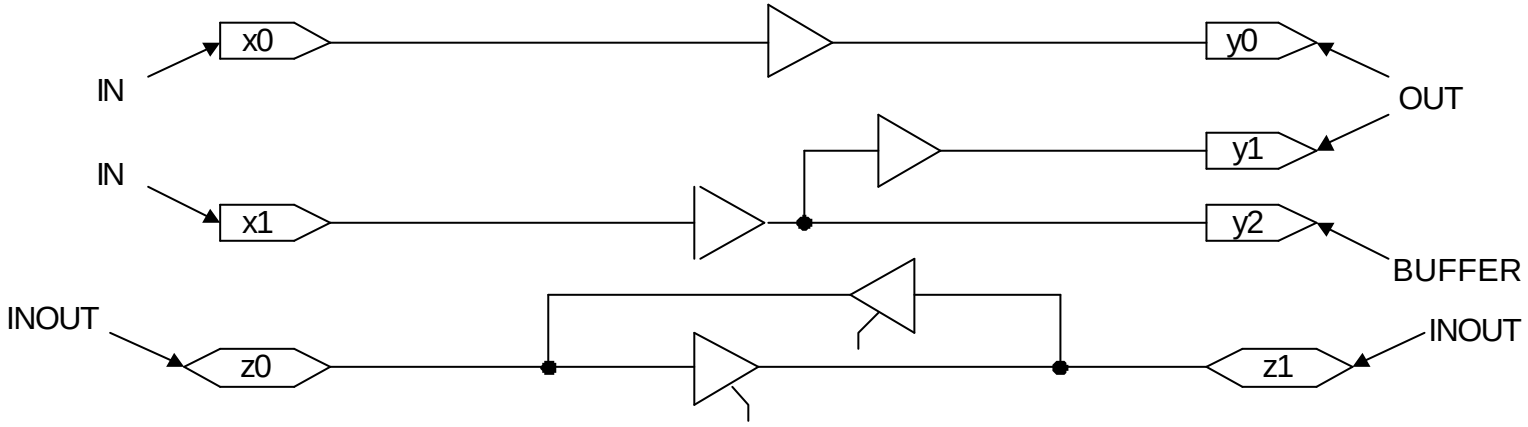


- ENTITY: inicia a declaração
- PORT: define modo e tipo das portas
  - modo IN : entrada
  - modo OUT : saída
  - modo BUFFER : saída - pode ser referenciada internamente
  - modo INOUT : bidirecional
- END: termina a declaração

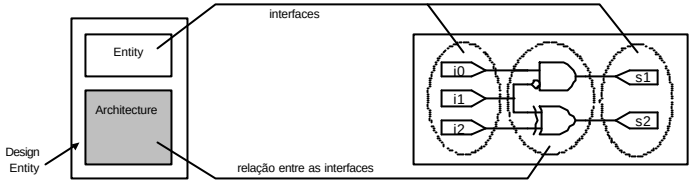
```
ENTITY entidade_abc IS


  PORT (x0, x1     : IN        tipo_a;    -- entradas
        y0, y1     : OUT       tipo_b;    -- saidas
        y2         : BUFFER    tipo_c;    -- saida
        z0, z1     : INOUT     tipo_d);   -- entrada / saida

END entidade_abc;
```

- Exemplo: declaração de uma entidade & esquema da arquitetura

```
ENTITY entidade_abc IS

  PORT (x0, x1     : IN        tipo_a;    -- entradas
        y0, y1     : OUT       tipo_b;    -- saidas
        y2         : BUFFER    tipo_c;    -- saida
        z0, z1     : INOUT     tipo_d);   -- entrada / saida

END entidade_abc;
```



Declaração da arquitetura



- ARCHITECTURE: inicia a declaração
- linhas que seguem podem conter:
  - 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;
```

Exemplos de classe de objetos: constante sinal

- Objetos: são elementos que contêm um valor armazenado
- Exemplo:
  - CONSTANT: valor estático
  - SIGNAL: valor imposto pode ser alterado
    regiões de código seqüencial e concorrente

- Exemplos de transferência de valores: constante e sinal

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

- Exemplo de uma descrição completa
  - definidas três portas: uma entrada, duas de saída
  - tipo das portas: INTEGER
  - operações: transferência de valores
  - declarados: s1 sinal interno, e c1 constante
  - concorrência no código: (próxima imagem)

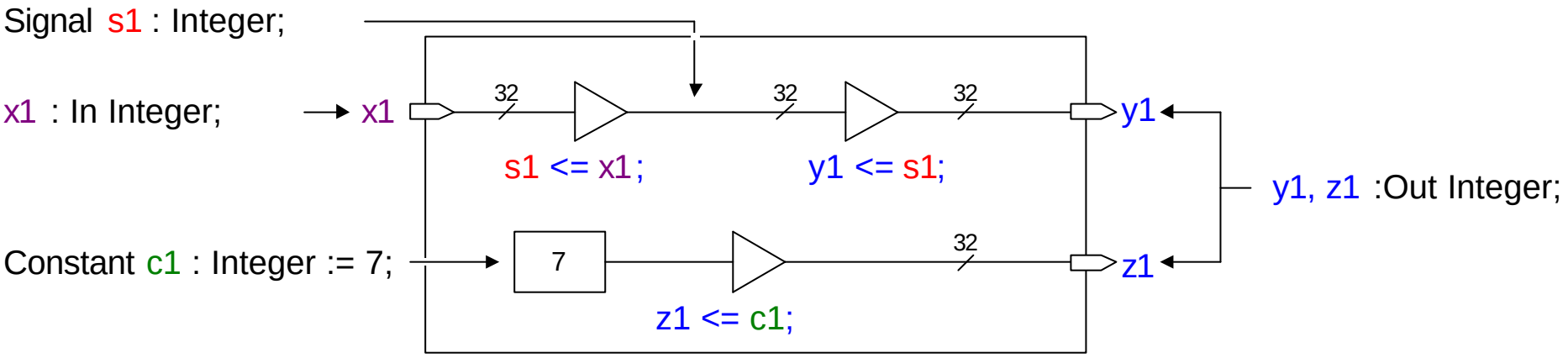
```
 1 ENTITY atrib_1 IS
 2   PORT (x1     : IN  INTEGER;       -- porta entrada
 3         y1,z1  : OUT INTEGER);      -- portas saida
 4 END;
 5
 6 ARCHITECTURE teste OF atrib_1 IS
 7   SIGNAL   s1 : INTEGER;           -- declaracao de um sinal tipo inteiro
 8   CONSTANT c1 : INTEGER := 7;      -- declaracao de uma constante tipo inteiro
 9 BEGIN
10   y1 <= s1;                       -- regiao de codigo concorrente
11   s1 <= x1;
12
13   z1 <= c1;
14 END teste;
```

- Exemplo de uma descrição completa

- observar concorrência no código:

linha 10: y1<= s1 -- valor de s1 transferido para saída y1

linha 11: s1<= x1 -- s1 recebe o valor da entrada x1



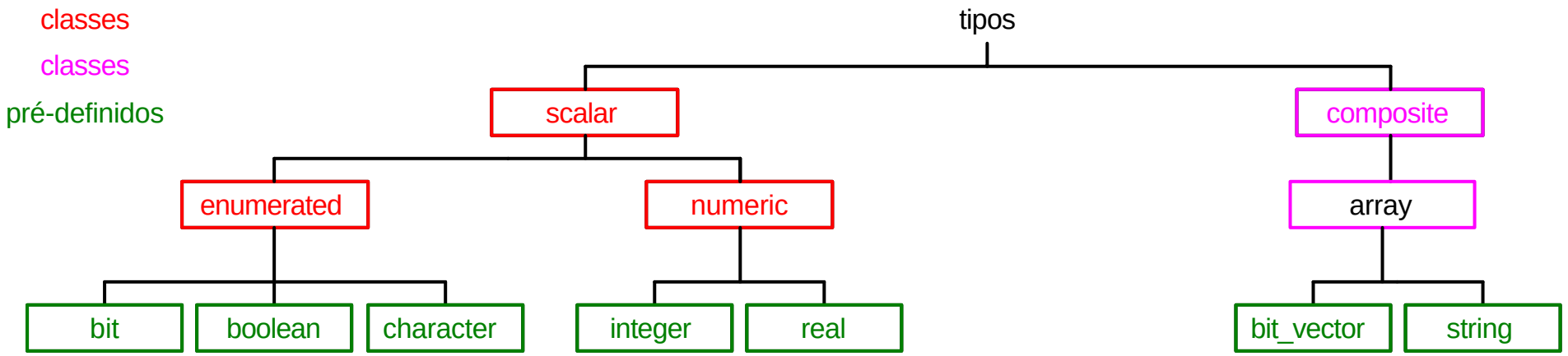

```
 5
 6 ARCHITECTURE teste OF atrib_1 IS
 7   SIGNAL   s1 : INTEGER;           -- declaracao de um sinal tipo inteiro
 8   CONSTANT c1 : INTEGER := 7;      -- declaracao de uma constante tipo inteiro
 9 BEGIN
10   y1 <= s1;                        -- regiao de codigo concorrente
11   s1 <= x1;
12
13   z1 <= c1;
14 END teste;
```

Tipos

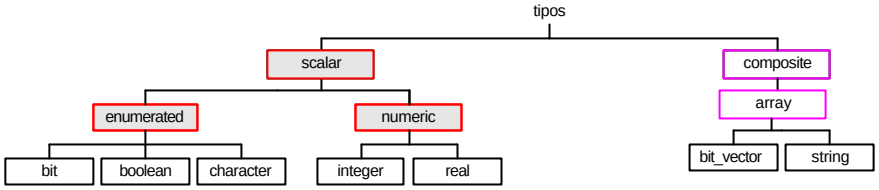
- Objetos:
  - devem ser declarados segundo uma especificação de tipo

- Objetos de tipos diferentes:
  - não é permitida a transferência de valores

- Exemplo de tipos pré-definidos no pacote padrão VHDL:



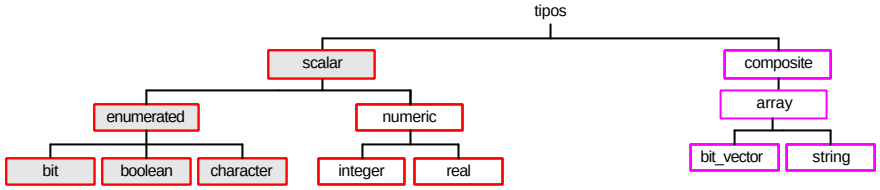
Tipos escalares



- São ordenados
  - podem ser aplicados operadores: maior, menor
- Classes: enumerado e numérico

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	$-2^{31} \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B# 2#11_11_011#
	REAL	$-3.65\times10^{47} \leq x \leq +3.65\times10^{47}$	1.23, 1.23E+2, 16#7.B#E+1

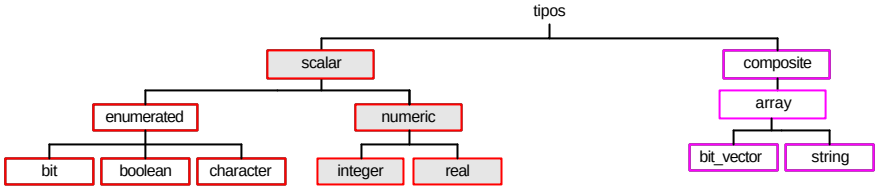
Tipos escalares



- Classe enumerado (pré-definidos):
  - BIT: empregado para representar níveis lógicos “0” e “1”
  - BOOLEAN: empregado em declarações que executam uma decisão
  - CHARACTER: qualquer caracter ASCII padrão

classe	tipo	valor	exemplos
enumerado	BIT	um, zero	1, 0
	BOOLEAN	verdadeiro, falso	TRUE, FALSE
	CHARACTER	caracteres ASCII	a, b, c, A, B, C, ?, (

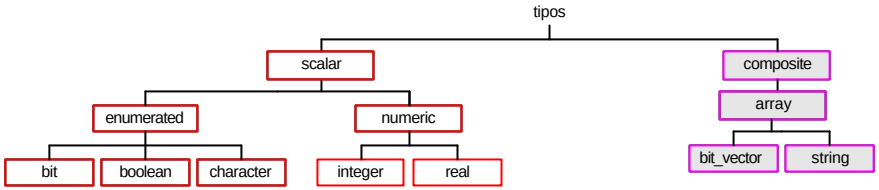
Tipos escalares



- Classe numérico (pré-definidos):
  - INTEGER: representa um número inteiro entre $-2^{31} \leq x \leq 2^{31}-1$
    - necessário uma linha de 32 bits para representação!
    - conveniente limitar a faixa de valores na declaração
  - REAL: ponto flutuante
    - não suportado pelas ferramentas de síntese

classe	tipo	valor	exemplos
numérico	INTEGER	$-2^{31} \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B# 2#11_11_011#
	REAL	$-3.65\times10^{47} \leq x \leq +3.65\times10^{47}$	1.23, 1.23E+2, 16#7.B#E+1

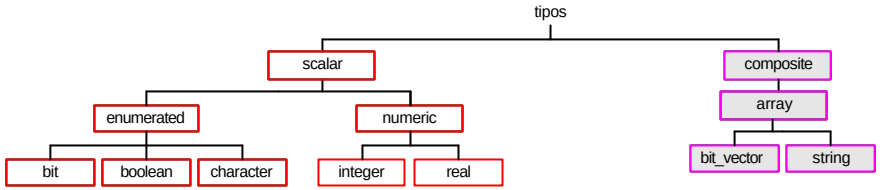
Tipos compostos



- Classe vetor (pré-definidos):
  - 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”, ””incluindo_aspas””

Tipos compostos



- Declarações:
  - limites definidos por TO e DOWNTO
- Exemplos de declarações tipos: bit_vector e string:

CONSTANT a: BIT_VECTOR(0 TO 7) := "10110011"

1	0	1	1	0	0	1	1
a(0)	a(1)	a(2)	a(3)	a(4)	a(5)	a(6)	a(7)

CONSTANT b: BIT_VECTOR(7 DOWNTO 0) := "10110011"

1	0	1	1	0	0	1	1
b(7)	b(6)	b(5)	b(4)	b(3)	b(2)	b(1)	b(0)

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

A	l	o		m	u	n	d	o
c(1)	c(2)	c(3)	c(4)	c(5)	c(6)	c(7)	c(8)	c(9)

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

A	l	o		m	u	n	d	o
d(9)	d(8)	d(7)	d(6)	d(5)	d(4)	d(3)	d(2)	d(1)

Exemplo de operadores

- Divididos em classes:
  - as 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

- Tipos envolvidos na operação
  - maioria das operações: operadores do mesmo tipo

- Operadores lógicos:
  - operandos: tipos bit e boolean
  - podem ser empregados em vetores (arrays):
    exemplo: bit_vector
    - vetores devem ter o mesmo tamanho
    - operação executada entre elementos de mesma posição

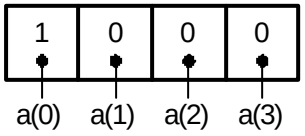
operadores	operando “L”	operando “R”	retorna
not	-	bit	bit
	-	boolean	boolean
and or nand nor xor xnor	bit	bit	bit
	boolean	boolean	boolean
Nota: O operador “not” pertence a classe “diversos”			

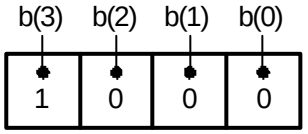
- Operadores relacionais:
  - igualdade e desigualdade (= /=): qualquer tipo
    - a=b para escalares: a mesmo valor de b
    - a=b para compostos: cada elemento de mesma posição igual
  - ordenação (> < >= <=): tipos escalares (bit, boolean, character, integer, real, time)

operadores	operando “L”	operando “R”	retorna
= /=	qualquer tipo	mesmo tipo de “L”	boolean
> < >= <=	qualquer tipo escalar	mesmo tipo de “L”	boolean

- exemplo de valores tipo bit_vector iguais:

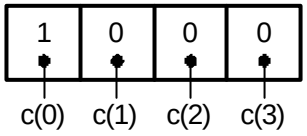
CONSTANT a: BIT_VECTOR(0 TO 3) := "1000"





CONSTANT b: BIT_VECTOR(3 DOWNTO 0) := "1000"

CONSTANT c: BIT_VECTOR(0 TO 3) := "1000"



- Operadores 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

- Exemplo:

```
a <= b & c;   -- ”a” bit_vector 8 elementos,   ”b”,”c” bit_vetor 4 elementos
x <= y & ’1’; -- ”x” bit_vector 5 elementos,      ”y” bit_vetor 4 elementos
```

Exemplo: Atribuição de valores em sinais, tipos BIT_VECTOR

- operação: valor 1011 atribuído a todas as portas de saída
- diferentes bases de representação:
  tipos integer, real formato: 16#b# 16#b.0#
  tipos bit_vector formato: X"B"
- linha 10: caracter "_" como separador (melhora leitura do valor)
- linha 12: valor definido para cada elemento,
  palavra reservada OTHERS especifica elementos restantes

```
 1 ENTITY std_a IS
 2  PORT (s1,s2,s3,s4,s5  : OUT BIT_VECTOR (3 DOWNTO 0));
 3 END std_a;
 4
 5 ARCHITECTURE teste OF std_a IS
 6   CONSTANT c1  : BIT_VECTOR(3 DOWNTO 0) := "1011";  -- constante
 7 BEGIN
 8   s1 <= c1;                                  -- definindo atraves de constante
 9   s2 <= "1011";                              -- definindo valor bit a bit
10   s3 <= B"1_0_11";                           -- binario default com separadores
11   s4 <= X"B";                                -- hexadecimal
12   s5 <= (3 =>'1', 2 =>'0', OTHERS =>'1'); -- uso da palavara reservada "others"
13 END teste;
```

Exemplo: Operadores classe adição

- linhas 10 e 11: operação de concatenação de dois vetores (tipo bit_vector)

- linha 12: soma de dois tipos inteiros

```
 1 ENTITY std_xc IS
 2   PORT (bv_a,  bv_b  : IN  BIT_VECTOR(1 DOWNTO 0);
 3         int_a, int_b : IN  INTEGER RANGE -32 TO 31;
 4         bv_c,  bc_d  : OUT BIT_VECTOR(3 DOWNTO 0);
 5         int_c        : OUT INTEGER RANGE -64 TO 63);
 6 END std_xc;
 7
 8 ARCHITECTURE teste OF std_xc IS
 9 BEGIN
10   bv_c  <= bv_a & bv_b;
11   bc_d  <= bv_a & '1' & '0';
12   int_c <= -int_a +int_b;
13 END teste;
```

Exercícios

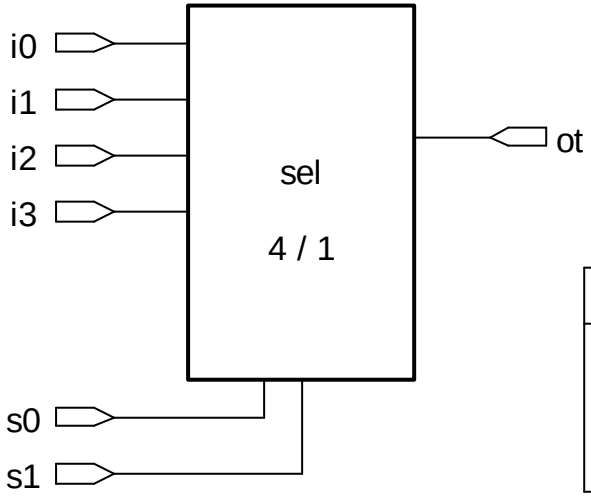
- Compilar e simular as descrições:
  - std_a arquivo: std_a.vhd
  - std_xc arquivo: std_xc.vhd

Comandos concorrentes básicos

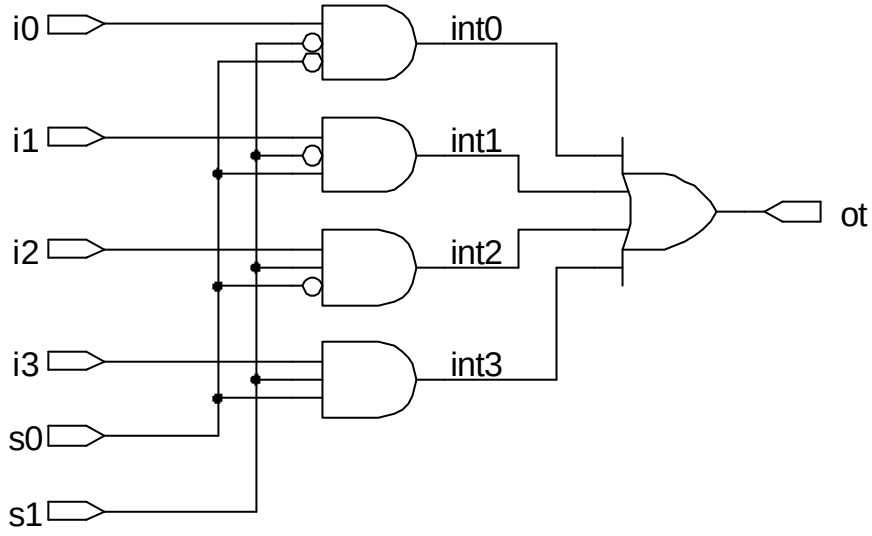
- Construção WHEN ELSE
- Construção WITH SELECT

Descrição de um circuito de seleção

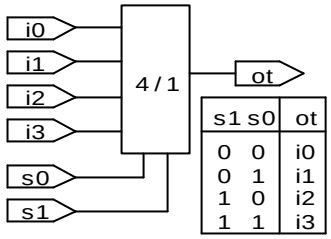
- entradas: i0, i1, i2 e i3
- saída: ot
- controle da seleção: s0 e s1



s1	s0	ot
0	0	i0
0	1	i1
1	0	i2
1	1	i3



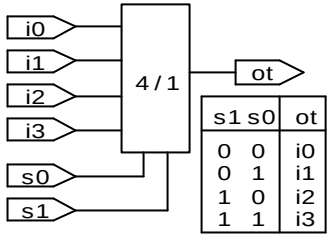
Exemplo: descrição do circuito de seleção



- descrição emprega uma única expressão
- observar: uso de parêntesis → AND e OR igual precedência

```
 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;
```

Exemplo: - nova descrição do circuito de seleção



- emprega 5 expressões
- sinais internos: int0, int1, int2 e int3
- observar concorrência do código:
  - valor de ot,
    determinado pelas expressões linhas 11, 12, 13 e 14

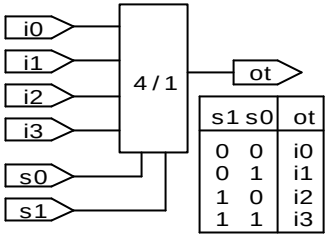
```
 1 ENTITY mux_00 IS
 2   PORT (i0, i1, i2, i3    :  IN  BIT;  -- entradas
 3         s0, s1            :  IN  BIT;  -- selecao
 4         ot                :  OUT BIT); -- saida
 5 END mux_00;
 6
 7 ARCHITECTURE teste OF mux_00 IS
 8   SIGNAL int0, int1, int2, int3 : BIT; -- sinais internos
 9 BEGIN
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;
```

Construção WHEN ELSE

- Transferência condicional de um sinal
- Contém: uma lista de condições e expressões
- Primeira condição verdadeira: define expressão transferida
- Formato da construção:

```
sinal_destino <= expressao_a  WHEN condicao_1 ELSE    -- condicao_1 = verdadeira
                 expressao_b  WHEN condicao_2 ELSE    -- condicao_2 = verdadeira
                 expressao_c;                         -- nenhuma condicao verdadeira
```

Exemplo: - circuito de seleção - WHEN ELSE



- nível de abstração mais elevado
- descrição mais próxima do comportamento do circuito
- opção de escolha:
- linhas 10 e 11: operação AND entre s0 e s1
- linha 12: s0 e s1 concatenados

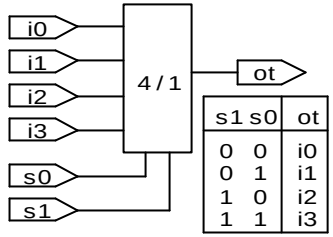
```
 1 ENTITY mux_1 IS
 2   PORT (i0, i1, i2, i3    :  IN  BIT;
 3         s0, s1            :  IN  BIT;
 4         ot                :  OUT BIT);
 5 END mux_1;
 6
 7 ARCHITECTURE teste OF mux_1 IS
 8   SIGNAL s1_s0 : BIT_VECTOR(1 DOWNTO 0);
 9 BEGIN
10   ot <= i0 WHEN s1= '0' AND s0='0' ELSE
11    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

Construção 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: - caracter | 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
```

Exemplo: circuito de seleção - WITH SELECT

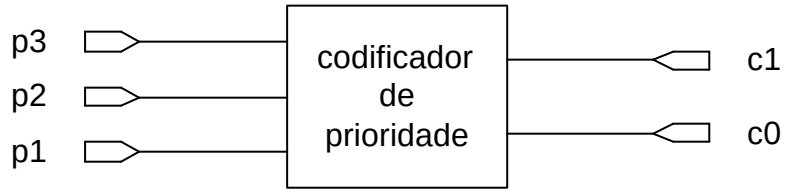


- nível de abstração mais elevado
- descrição mais próxima do comportamento do circuito
- expressão de escolha:
  sinal sel = s1 e s0 concatenados

```
 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;
```

Exercício

- Desenvolver uma descrição para um codificador de prioridade empregando a construção WHEN ELSE
  - três entradas denominadas p3, p2 e p1
  - p3 a de maior privilégio
  - código da entrada: saídas c1 e c2



p3	p2	p1	c1	c0
1	-	-	1	1
0	1	-	1	0
0	0	1	0	1
0	0	0	0	0

- ⇒ não importa

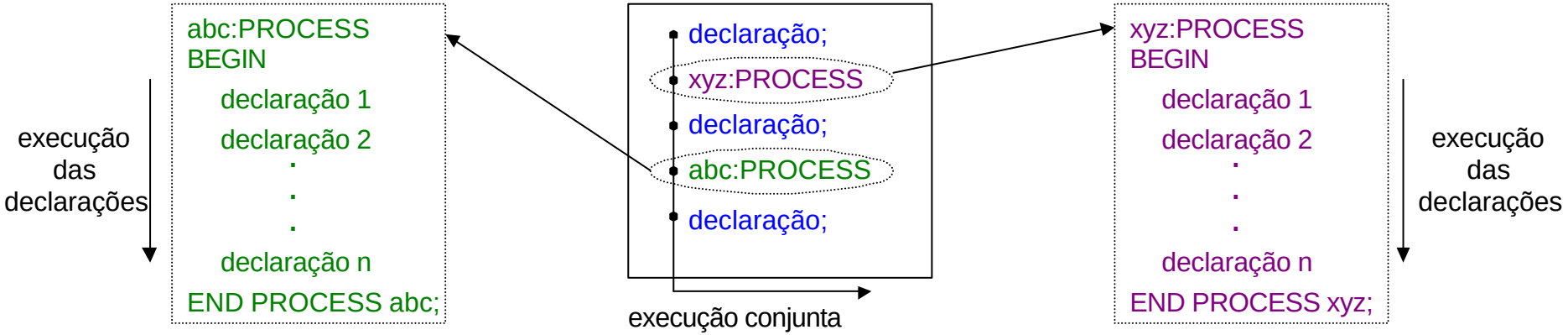
- Repita o exercício anterior empregando a construção WITH SELECT

Comandos seqüenciais básicos

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

Comandos seqüenciais

- Comandos seqüenciais podem ocorrer em:
  - processos
  - subprogramas
- Processo
  - é um comando concorrente
  - delimita uma região contendo código seqüencial
- 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)

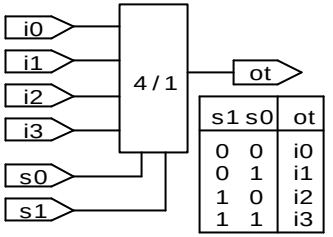
```
abc: PROCESS(sinal_a, sinal_b)     -- (lista de sensibilidade)
     BEGIN
        comando_1;
        comando_2;
           ..
        comando_n;
END PROCESS abc;
```

Construção seqüencial IF ELSE (similar: construção WHEN 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
- Formato da construção:

```
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;
```

Exemplo: - circuito de seleção - IF ELSE



- Comando seqüencial: necessário definir um processo
- Lista de sensibilidade: sinais i0 i1 i2 i3 sel
  - remoção de um destes sinais: conseqüência?

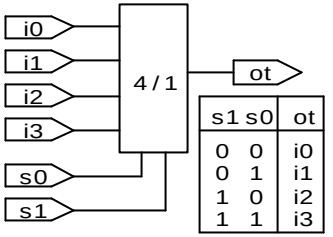
```
 1 ENTITY mux_4aa IS
 2   PORT (i0, i1, i2, i3  :  IN  BIT;  -- entradas
 3         s0, s1          :  IN  BIT;  -- selecao
 4         ot              :  OUT BIT); -- saida
 5 END mux_4aa;
 6
 7 ARCHITECTURE teste OF mux_4aa IS
 8   SIGNAL sel : BIT_VECTOR (1 DOWNTO 0);
 9 BEGIN
10   sel <= s1 & 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;
```

Construção CASE WHEN (similar: construção WITH SELECT)

- 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
- Formato da construção:

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

Exemplo : - circuito de seleção - CASE WHEN



- Comando seqüencial: necessário definir um processo
- Nota: s1 e s2 agrupados no sinal sel

```
 1 ENTITY mux_3aa IS
 2   PORT (i0, i1, i2, i3  :  IN  BIT;
 3         s1, s0          :  IN  BIT;
 4         ot              :  OUT BIT);
 5 END mux_3aa;
 6
 7 ARCHITECTURE teste OF mux_3aa 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     CASE sel IS
14        WHEN "00"   => ot <= i0;
15        WHEN "01"   => ot <= i1;
16        WHEN "10"   => ot <= i2;
17        WHEN OTHERS => ot <= i3;
18     END CASE;
19   END PROCESS abc;
20 END teste;
```

Estratégias de descrição de circuitos síncronos

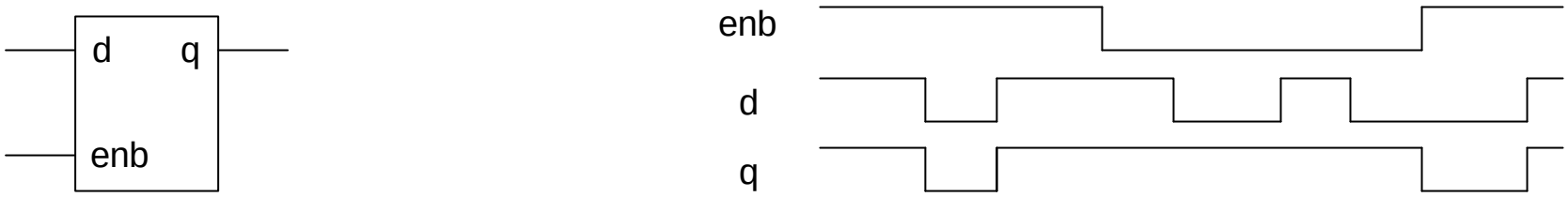
- Registrador sensível a nível
- Registrador sensível a borda - inicialização síncrona
- Registrador sensível a borda - inicialização assíncrona
- Máquinas de estado finito

Inferência de um elemento de memória em VHDL

- Inferência de um elemento de memória:
  - um valor é atribuído a um sinal - variável em pelo menos uma condição e nenhum valor é atribuído a este objeto em pelo menos uma condição

Registrador sensível a nível

- Exemplo:



- Formato geral: empregando processo

  - sinal d: deve estar na lista de sensibilidade

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

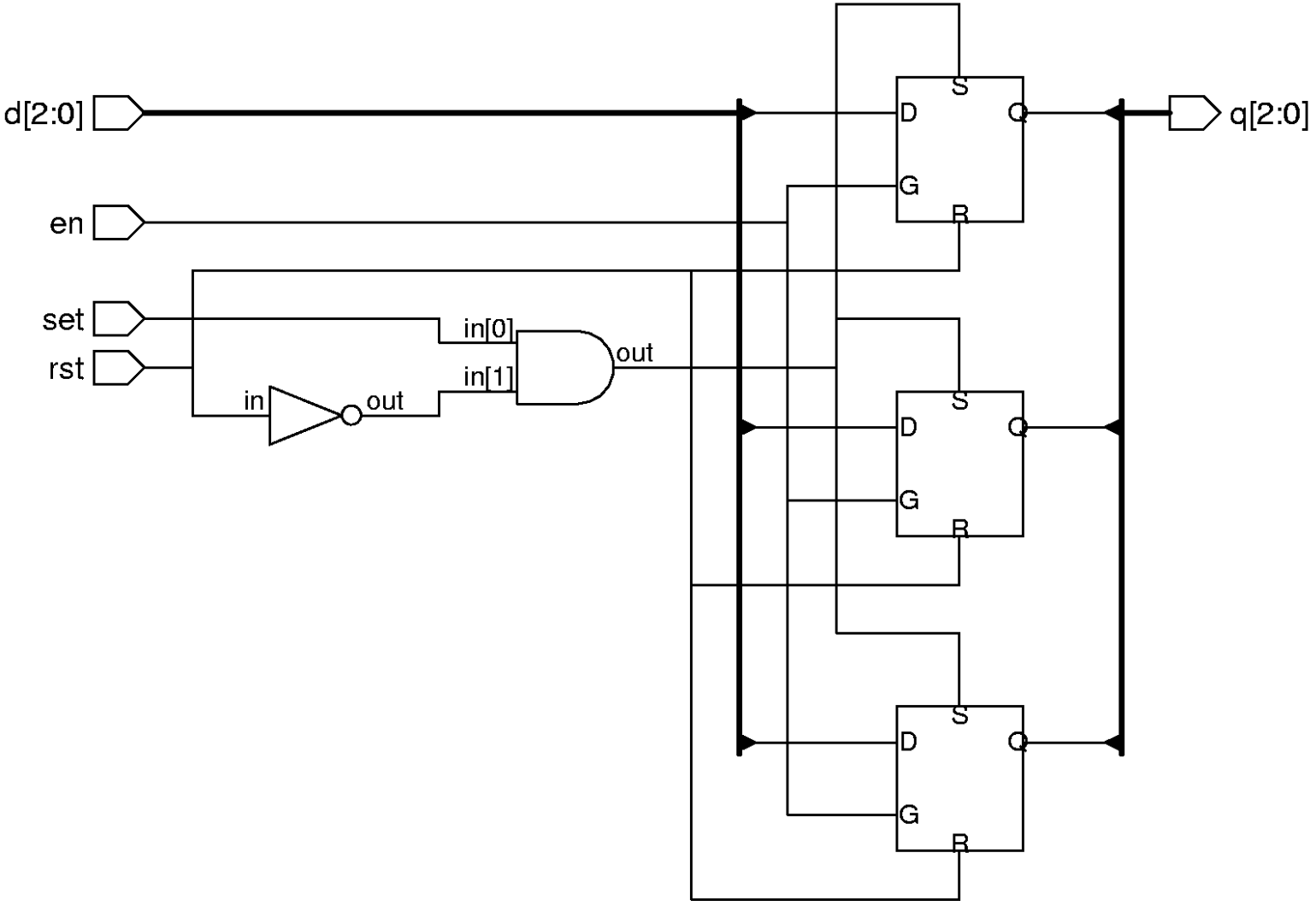
Registrador sensível a nível

- Exemplo: *reset* e *set* assíncronos (devem ser incluídos na lista de sensibilidade)
- Qual o comportamento da descrição se rst ou set forem removidos?

```
 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;
```

Registrador sensível a nível

- Resultado da síntese (nível RTL):

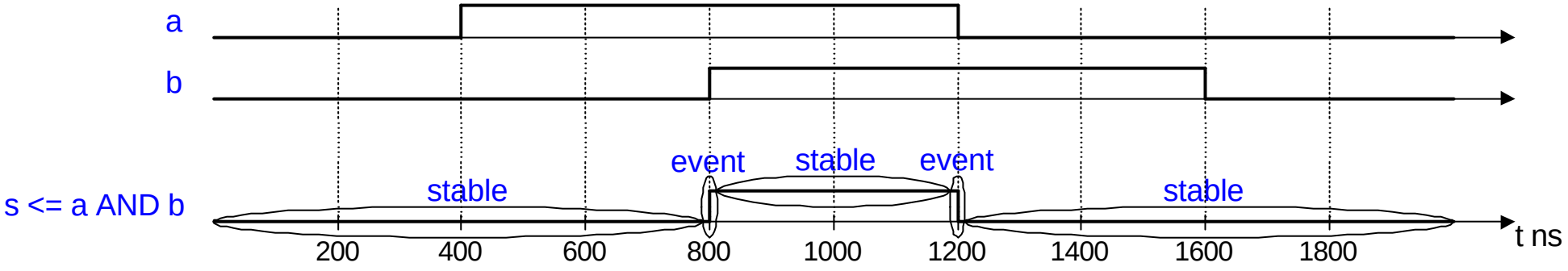


Atributos

- 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 - exemplos

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

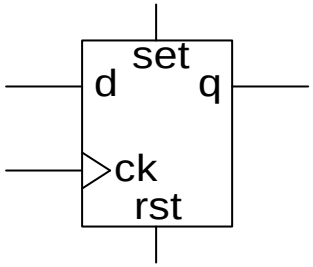


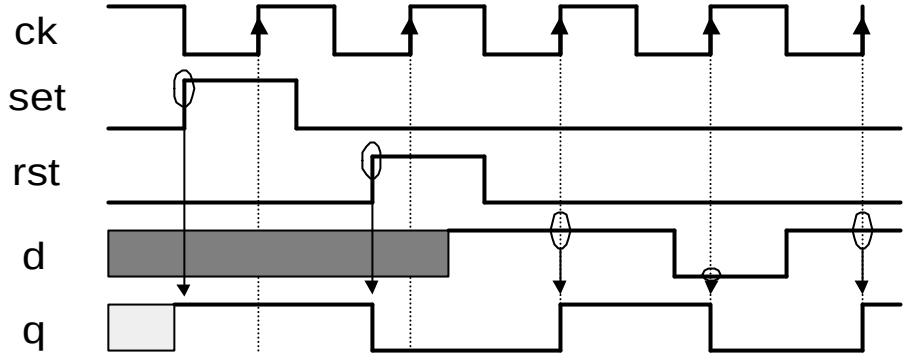
Atributos - aplicação

- verificação de bordas de subida ou descida em sinais

```
(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 - (exemplo)





- Formato geral: empregando processo

  - sinais ck rst e set necessitam estar na lista de sensibilidade

```
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;
```

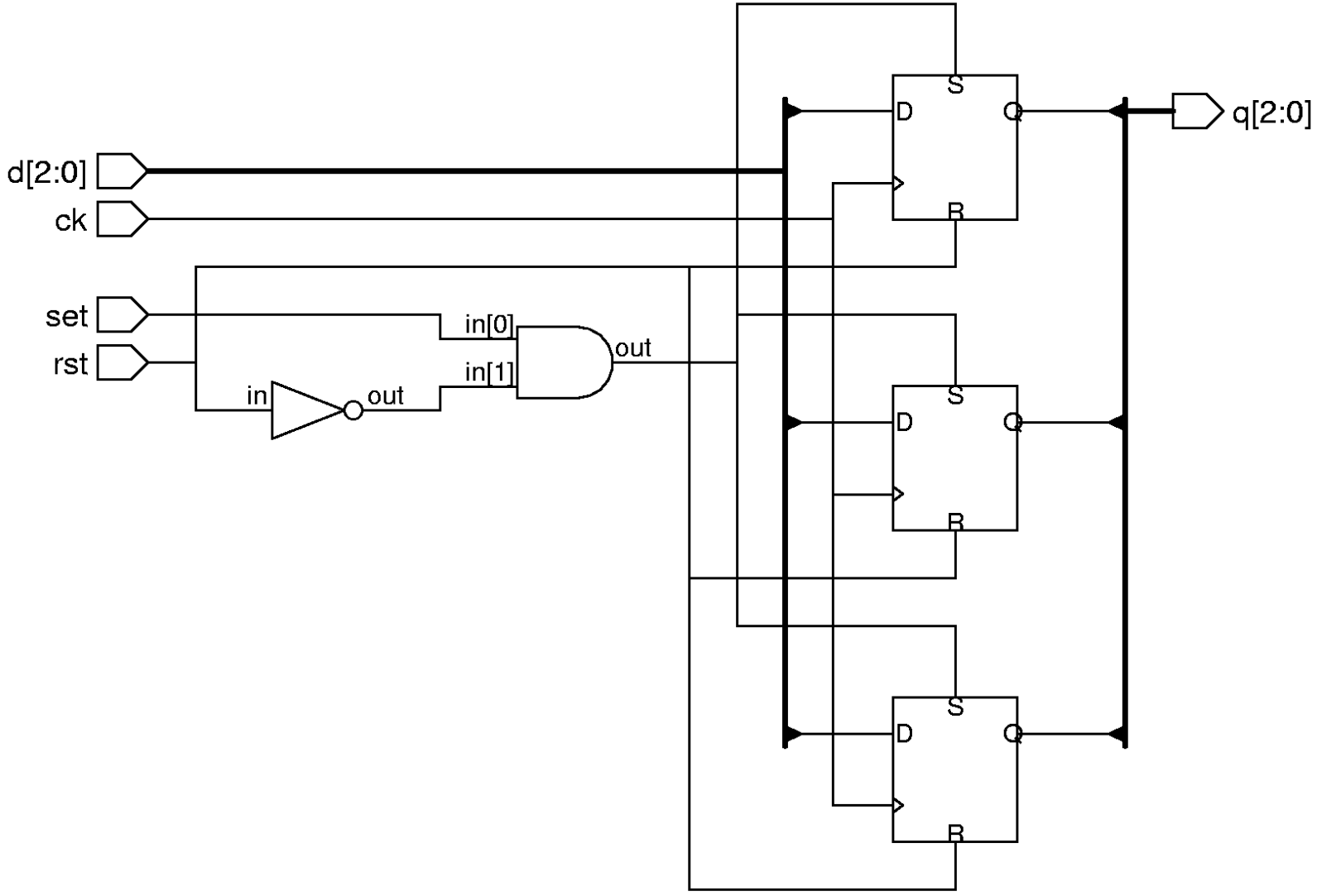
Registrador sensível a borda - inicialização assíncrona

- Exemplo:
  - 3 bits
  - *reset* e *set* assíncronos (é necessário inclusão na lista de sensibilidade)

```
 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

- Resultado da síntese (nível RTL):



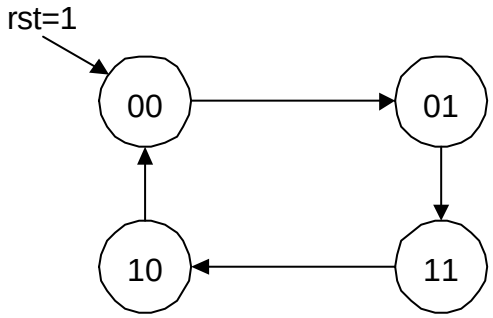
Máquinas de estado finito

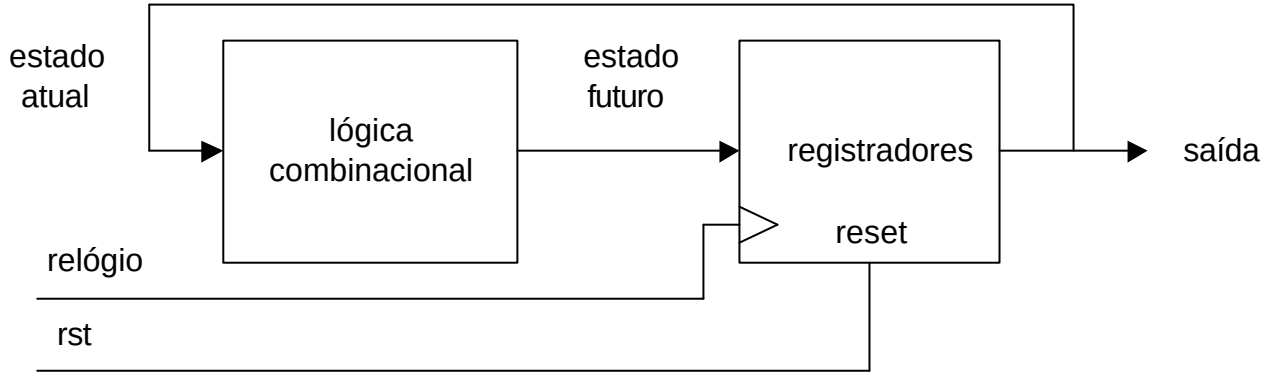
- Construção do tipo IF ELSE detecta:
  - inicialização rst =1
  - ocorrência de uma borda de subida no sinal de relógio
- Construção CASE WHEN:
  - definição das transições de estado
  - força a especificação de todos os estados

```
PROCESS (ck, rst)
 BEGIN
   IF rst = '1' THEN                                  -- estado inicial
     estado <= estado_inicial;
   ELSIF (ck'EVENT and ck ='1') THEN                  -- __/-- proximo estado
     CASE estado IS                                   -- ciclo de estados
       WHEN estado_inicial => estado <= estado_1;     --  .
       WHEN estado_1       => estado <= estado_2;     --  .
                                                      --  .
       WHEN estado_x       => estado <= estado_final;
     END CASE;
   END IF;
 END PROCESS;
```

Máquinas de estado finito - exemplo

- Contador
- Saída dos registradores corresponde ao valor de saída
  - código do estado = valor de saída
    - não é necessário decodificar



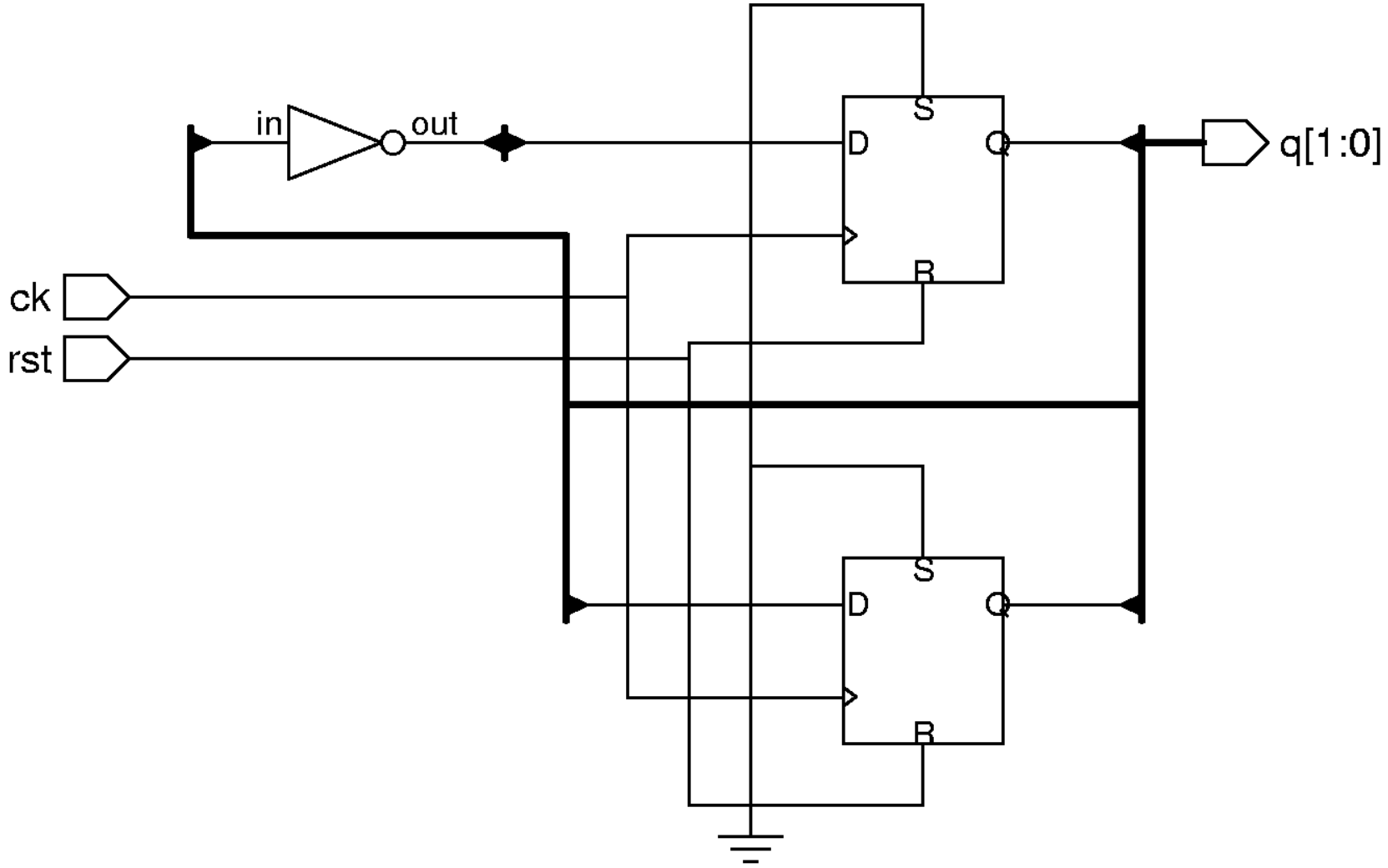


Máquinas de estado finito - descrição

```
 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;
```

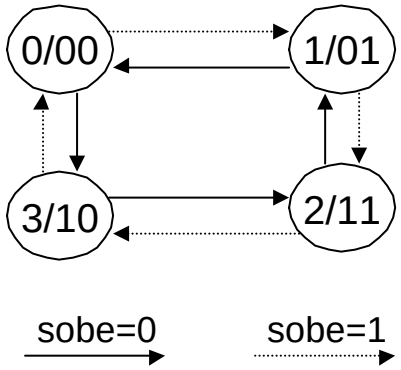
Máquinas de estado finito - descrição

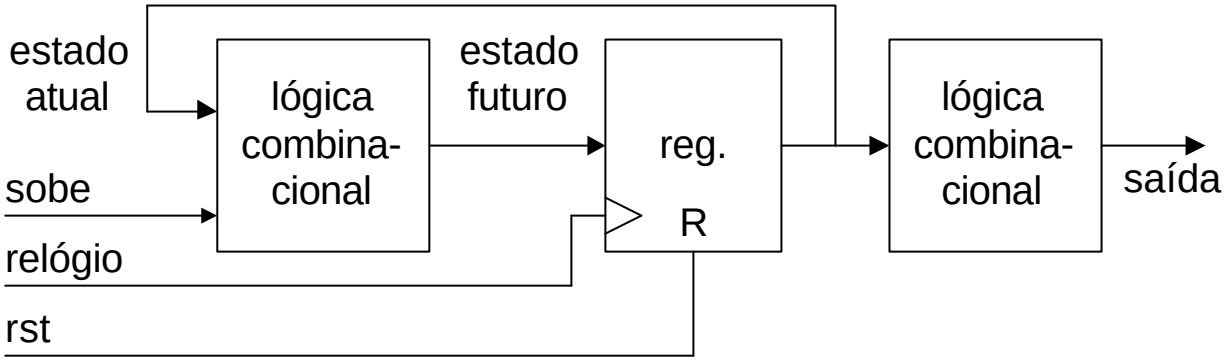
- Resultado da síntese (nível RTL):



Exercício

- Implementar uma máquina de estados
  - contador crescente decrescente código GRAY
  - entrada sobe define sentido crescente / decrescente





- Simule e verifique a descrição

Fim

Codificador de prioridade: possíveis soluções

- solução empregando a construção concorrente: WHEN ELSE

```
 1 ENTITY pr_cod1 IS
 2 PORT (p      : IN  BIT_VECTOR(3 DOWNTO 1);
 3       c      : OUT BIT_VECTOR(1 DOWNTO 0));
 4 END pr_cod1;
 5
 6 ARCHITECTURE teste OF pr_cod1 IS
 7 BEGIN                              -- p3 p1 p0  c1 c0
 8   c <= "11" WHEN p(3)='1' ELSE     --  1  -  -   1  1
 9        "10" WHEN p(2)='1' ELSE     --  0  1  -   1  0
10        "01" WHEN p(1)='1' ELSE     --  0  0  1   0  1
11        "00";                       --  0  0  0   0  0
12 END teste;
```

Codificador de prioridade: possíveis soluções

- solução empregando a construção concorrente: WITH SELECT

```
 1 ENTITY pr_cod2 IS
 2 PORT (p      : IN  BIT_VECTOR(3 DOWNTO 1);
 3       c      : OUT BIT_VECTOR(1 DOWNTO 0));
 4 END pr_cod2;
 5
 6 ARCHITECTURE teste OF pr_cod2 IS
 7 BEGIN
 8   WITH p SELECT                              -- p3 p1 p0  c1 c0
 9   c <= "11" WHEN "111"|"110"|"101"|"100",    --  1  -  -   1  1
10        "10" WHEN "011"|"010",                --  0  1  -   1  0
11        "01" WHEN "001",                      --  0  0  1   0  1
12        "00" WHEN "000";                      --  0  0  0   0  0
13 END teste;
```

Máquina de estado: possível solução

```
ENTITY maq_est2 IS
  PORT (ck      :  IN     BIT;                     -- relogio borda subida
        sobe    :  IN     BIT;                     -- sobe=1, q=00,01,11,10,00...
        rst     :  IN     BIT;                     -- rst=1, q=00
        q       :  OUT    BIT_VECTOR (1 DOWNTO 0)); -- saida codigo Gray
END maq_est2;

ARCHITECTURE teste OF maq_est2 IS
  SIGNAL estado : INTEGER RANGE 3 DOWNTO 0;
BEGIN
  abc: PROCESS (ck, rst)
  BEGIN
    IF rst = '1' THEN                    -- estado inicial
      estado <= 0;
    ELSIF (ck'EVENT and ck ='1') THEN    -- ciclo de estados
      CASE estado IS
        WHEN 0 =>                         -- s0
          IF sobe = '1' THEN estado <= 1;  -- s1
          ELSE               estado <= 3;  -- s3
          END IF;
        WHEN 1 =>                         -- s1
          IF sobe = '1' THEN estado <= 2;  -- s2
          ELSE               estado <= 0;  -- s0
          END IF;
        WHEN 2 =>                         -- s2
          IF sobe = '1' THEN estado <= 3;  -- s3
          ELSE               estado <= 1;  -- s1
          END IF;
        WHEN 3 =>                         -- s3
          IF sobe = '1' THEN estado <= 0;  -- s0
          ELSE               estado <= 2;  -- s2
          END IF;
       END CASE;
    END IF;
  END PROCESS abc;

  WITH estado SELECT
    q <= "00" WHEN 0,
         "01" WHEN 1,
         "11" WHEN 2,
         "10" WHEN 3;
END teste;
```