Declarações de Atribuição Concorrentes

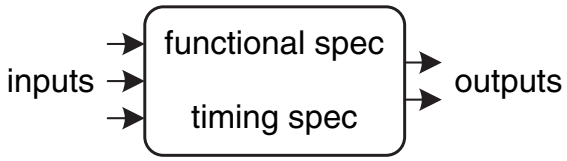
Objetivo

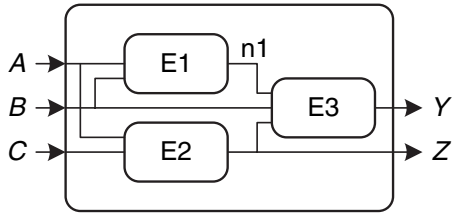
- Relembrar Combinacional vs. Sequencial
- Entender hardware gerado a partir de declarações concorrentes
- Comparar as declarações CONDICIONAL vs. SELEÇÃO
- Diretrizes de síntese

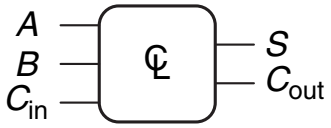
Introdução

Circuito Combinacional vs. Sequencial

- Circuito Combinacional:
  - Sem estado interno
  - Saída é função das entradas somente
  - Sem latches/FF sem realimentação



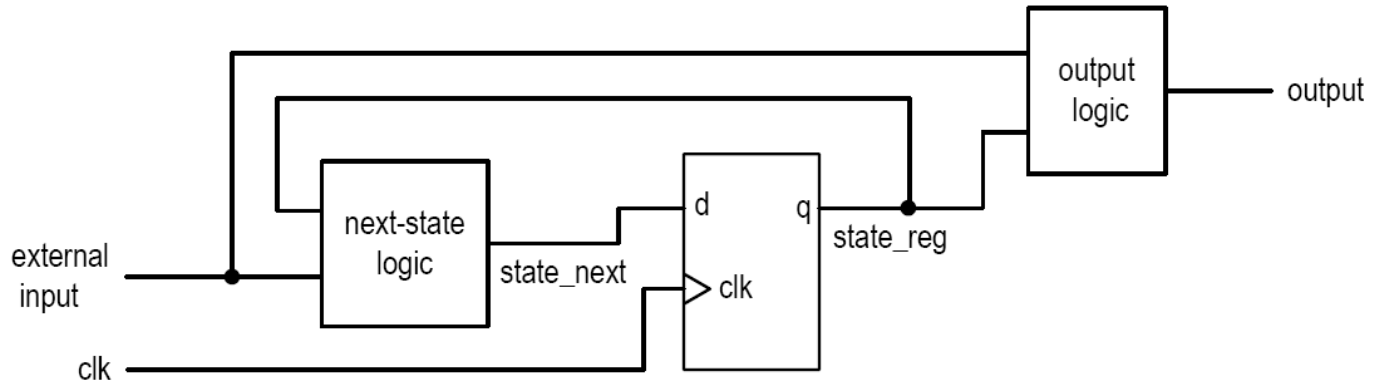




$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Circuito Combinacional vs. Sequencial

- Circuito Sequencial:
  - Com estado interno
  - Saída é função das entradas e do estado



- Circuito sequencial será discutido depois

Declarações de Atribuição de Sinal

- SIMPLES (<=)
- CONDICIONAL (*when else)*
- SELEÇÃO (*with select when*)

Declaração de atribuição SIMPLES

Sintaxe

- Syntax:

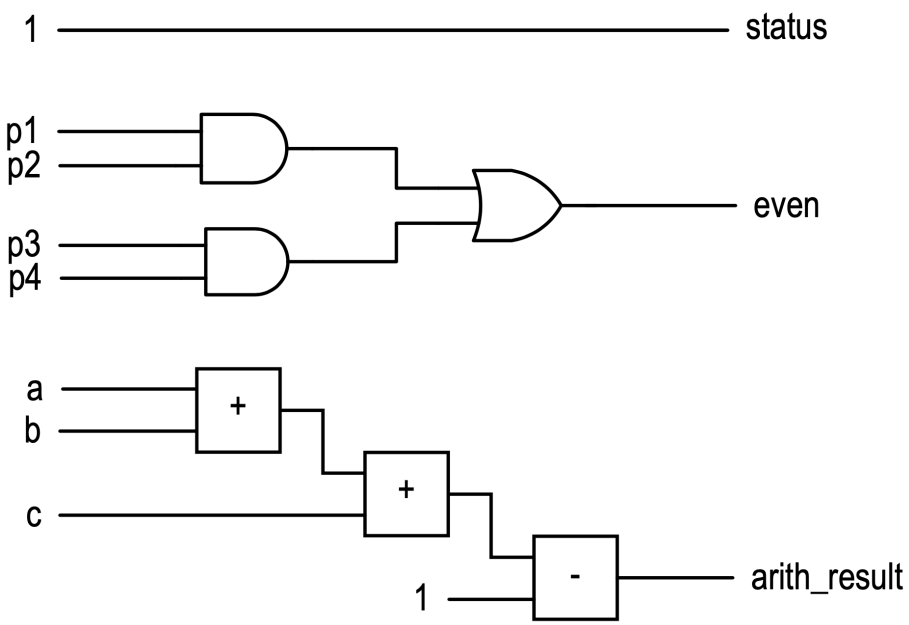
  signal_name <= projected_waveform;

- Exemplos:

  status <= '1';

  even <= (p1 and p2) or (p3 and p4);

  arith_out <= a + b + c - 1;



Problema Realimentação

- Um sinal aparece em ambos os lados de uma instrução de atribuição simultânea
- Exemplo:

    q <= ((not q) and (not en)) or (d and en);

- Sintaticamente correto
- Forma um ciclo de realimentação
- Deve ser evitado

Declaração de atribuição CONDICIONAL

when else

Implementação conceitual

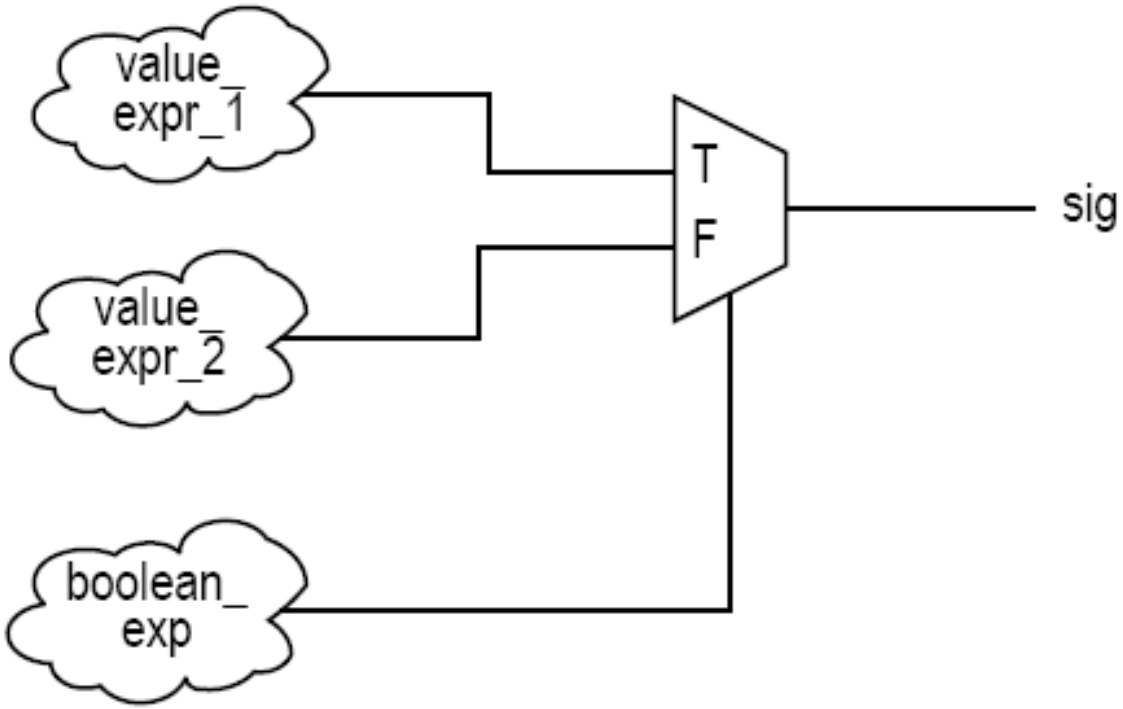
- Sintaxe:

signal_name
<= *value_expr_1* ***when*** *boolean_expr_1* ***else***
value_expr_2 ***when*** *boolean_expr_2* ***else***
value_expr_3 ***when*** *boolean_expr_3* ***else***
. . .
value_expo_n

- Avaliação em ordem crescente
- Realizado por “rede de roteamento prioritário”
- A expressão superior tem uma "prioridade mais alta"

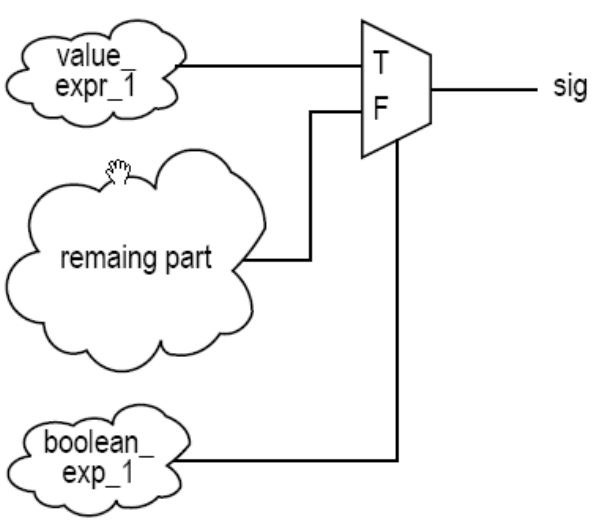
Implementação conceitual

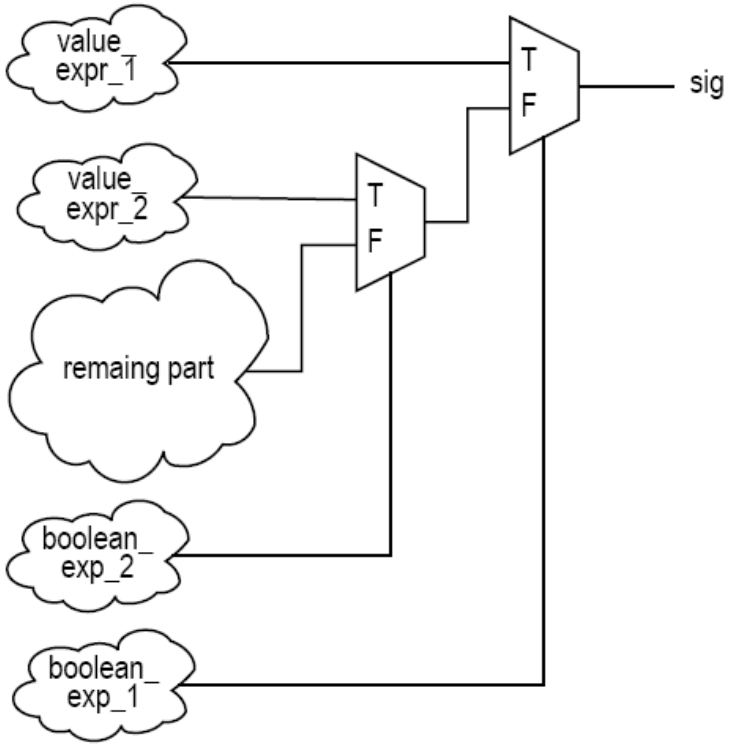
```
signal_name <= value_expr_1 when boolean_expr_1 else
               value_expr_2;
```



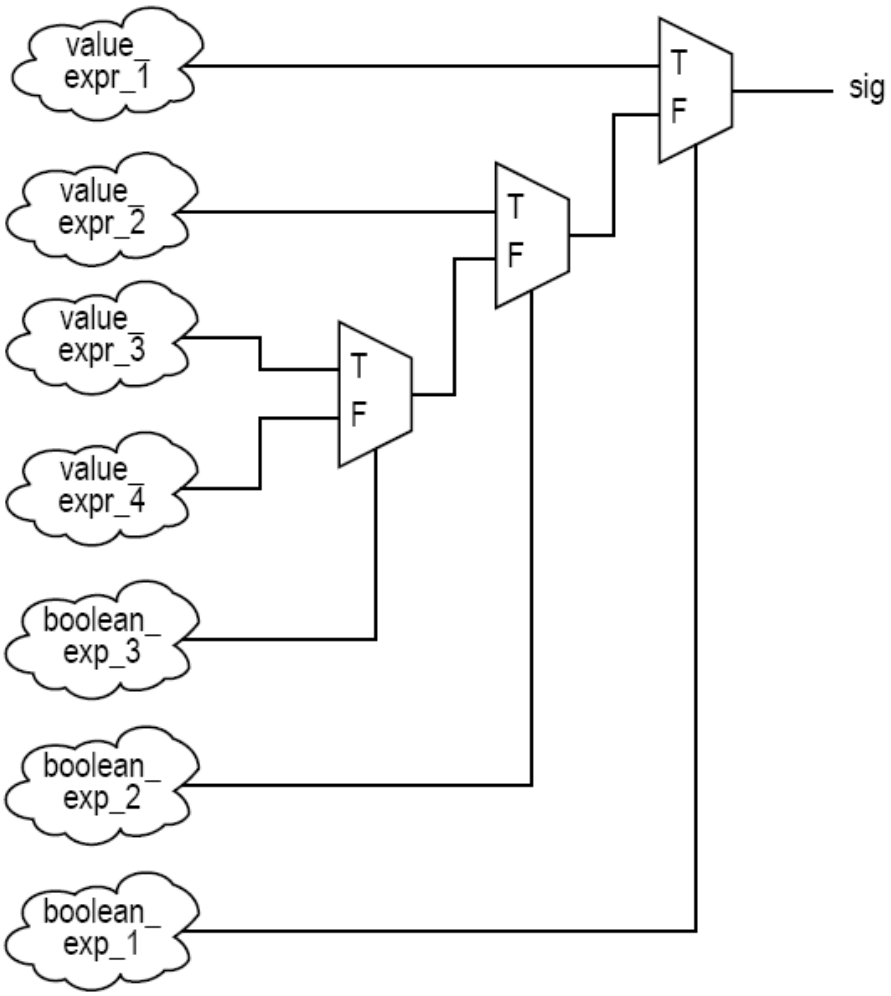
Implementação conceitual

```
signal_name <= value_expr_1 when boolean_expr_1 else
               value_expr_2 when boolean_expr_2 else
               value_expr_3 when boolean_expr_3 else
               value_expr_4;
```





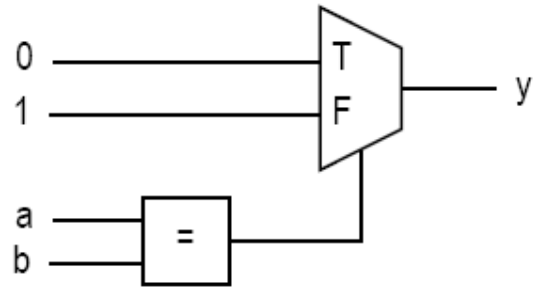
Implementação conceitual

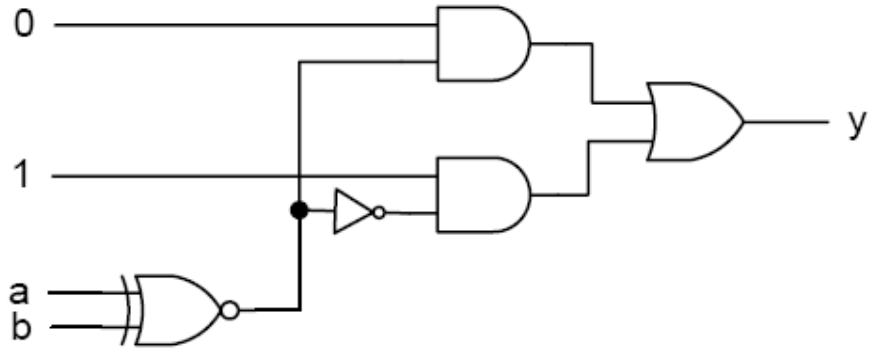


Implementação conceitual

- Exemplo:

```
. . .
signal a,b,y: std_logic;
. . .
y <= '0' when a=b else
     '1';
. . .
```

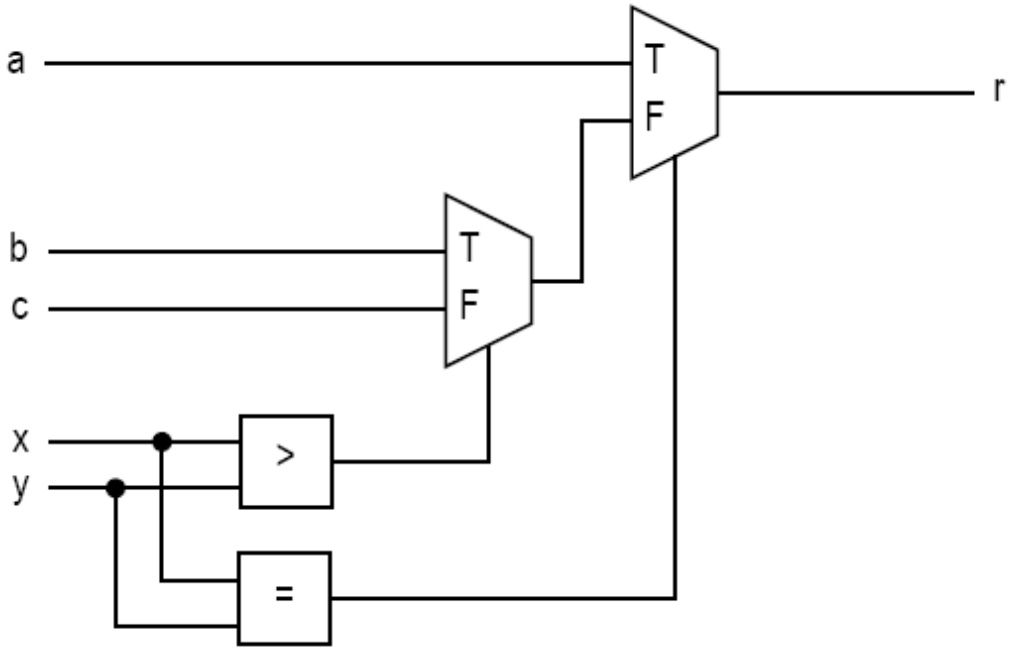




Implementação conceitual

- Exemplo:

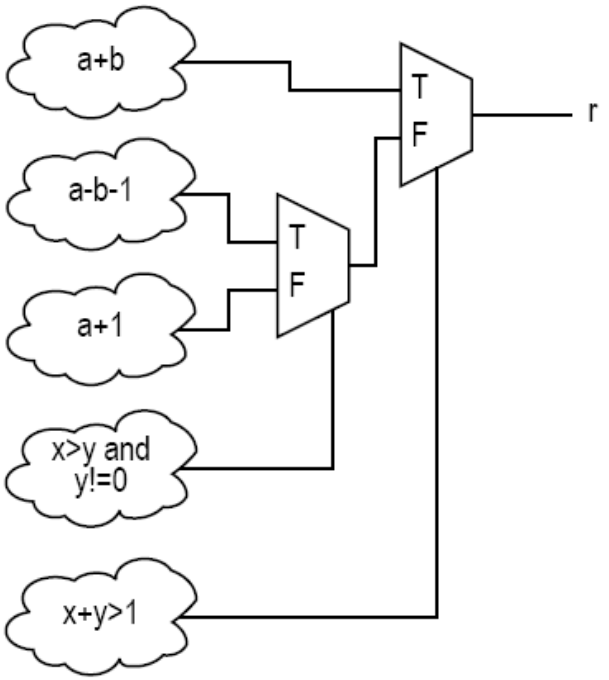
```
signal a,b,c,x,y,r: std_logic;
. . .
r <= a when x=y else
     b when x>y else
     c;
```

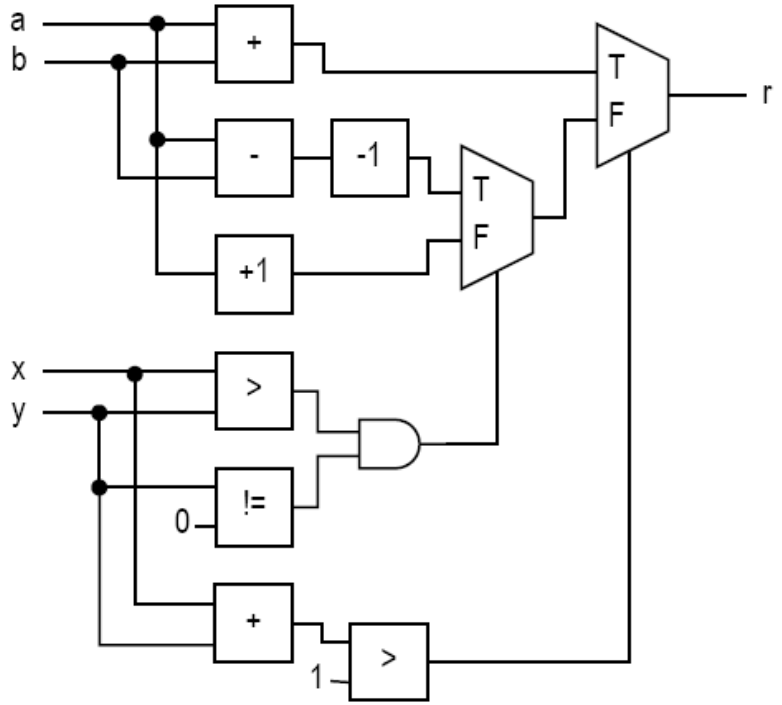


Implementação conceitual

- Exemplo:

```
. . .
signal a,b,r: unsigned(7 downto 0);
signal x,y: unsigned(3 downto 0);
. . .
r <= a+b when x+y>1 else
     a-b-1 when x>y and y!=0 else
     a+1;
```





Declaração de SELEÇÃO de atribuição

with select when

Sintaxe

- Sintaxe:

with *select_expression* ***select***
signal_name <=
value_expr_1 ***when*** *choice_1,*
value_expr_2 ***when*** *choice_2,*
value_expr_3 ***when*** *choice_3,*
. . .
value_expr_n ***when*** *choice_n;*

Sintaxe

- *select_expression*
  - Tipo discreto ou array
  - Com valores finitos possíveis
- *choice_i*
  - Um valor do tipo de dados
- As escolhas devem ser
  - Mutualmente exclusivo
  - tudo incluso
  - ***others*** pode ser usado como última *choice_i*

Sintaxe

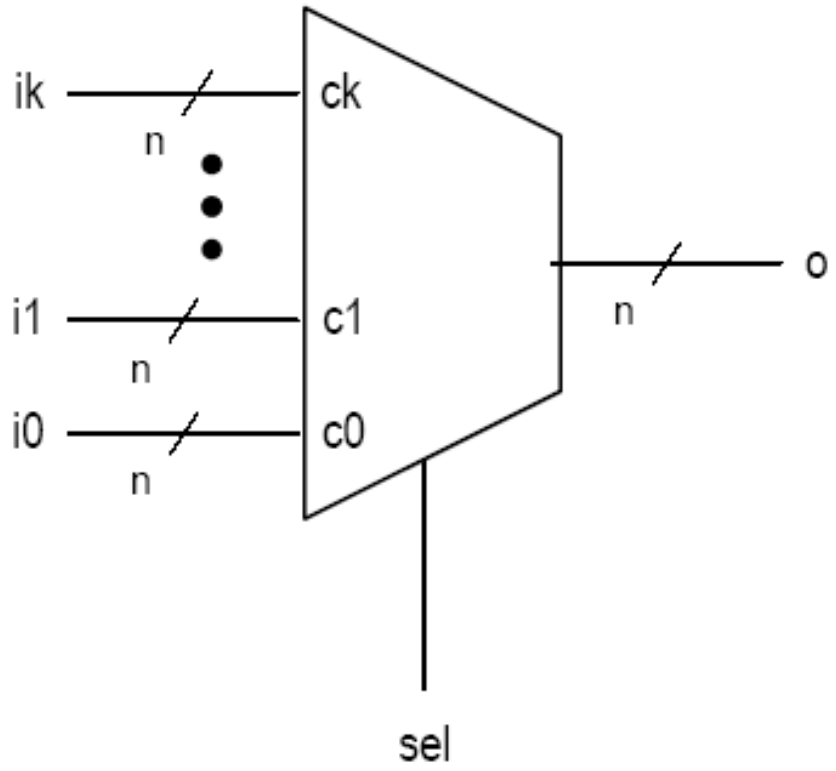
```
architecture sel_arch of mux4 is
begin
   with s select
     x <= a when "00",
          b when "01",
          c when "10",
          d when others;
end sel_arch;
```

```
with s select
  x <= a when "00",
       b when "01",
       c when "10",
       d when "11";
```

```
with s select
  x <= a when "00",
       b when "01",
       c when "10",
       d when "11",
      'X' when others;
```

Implementação conceitual

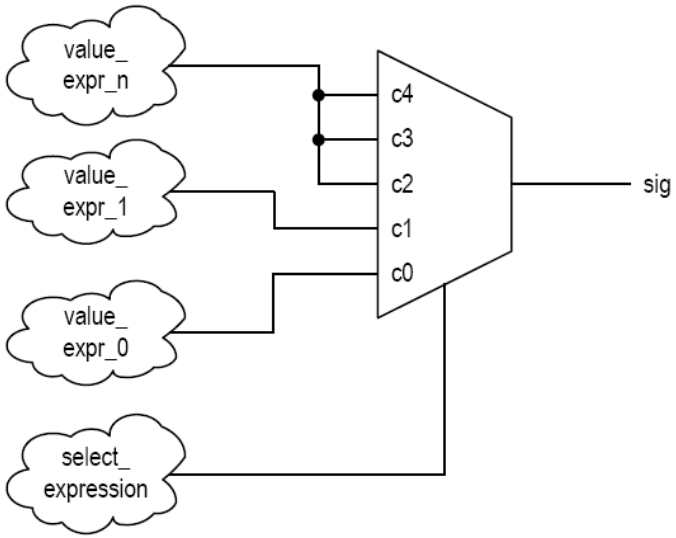
- Circuito multiplexador K+1:1.



Implementação conceitual

- *select_expression* com um tipo de dados de 5 valores:
  - c0, c1, c2, c3, c4

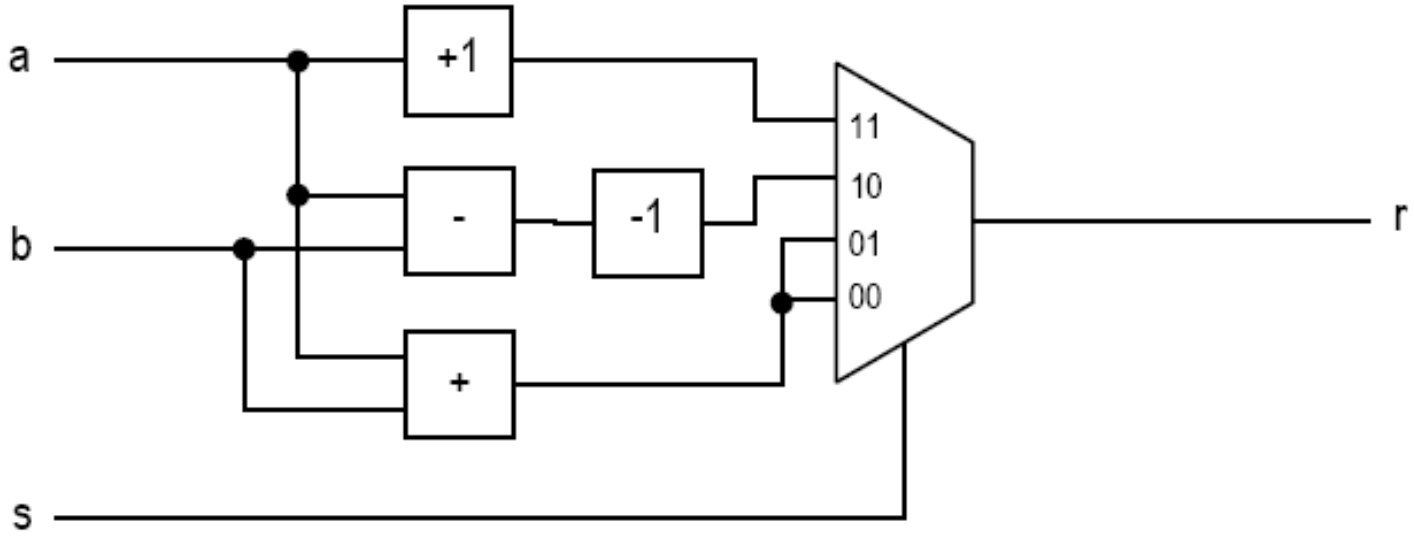
```
with select_expression select
   sig <= value_expr_0 when c0,
          value_expr_1 when c1,
          value_expr_n when others;
```



Implementação conceitual

- Exemplo:

```
signal a,b,r: unsigned(7 downto 0);
signal s: std_logic_vector(1 downto 0);
   . . .
   with s select
      r <= a+1    when "11",
           a-b-1  when "10",
           a+b    when others;
```



Comparação

CONDICIONAL

- Bom para um circuito que precisa dar tratamento preferencial para certas condições ou priorizar as operações.
- Ex.: codificador de prioridade
- Pode lidar com condições complicadas. Por exemplo:

```
pc_next <=
    pc_reg + offset when (state=jump and a=b) else
    pc_reg + 1 when (state=skip and flag='1') else
. . .
```

CONDICIONAL

- Pode ser complicado para tabelas verdade:

```
x <= a when (s="00") else
     b when (s="01") else
     c when (s="10") else
     d;

x <= c when (s="10") else
     a when (s="00") else
     b when (s="01") else
     d;

x <= c when (s="10") else
     b when (s="01") else
     a when (s="00") else
     d;
```

SELEÇÃO

- Bom para um circuito descrito por uma tabela verdade
- Ex.: Decodificador binário, multiplexador
- Menos eficaz quando um padrão de entrada recebe um tratamento preferencial

Diretrizes de Síntese

- Evite ciclo de realimentação de um sinal concorrente.
- Pense nas declarações CONDICIONAL e de SELEÇÃO como estruturas de roteamento em vez de construções de controle sequenciais.
- Declaração CONDICIONAL infere uma estrutura de roteamento de prioridade e um número grande de opções (*when else)* leva a uma longa cadeia em cascata.
- Declaração de SELEÇÃO infere uma estrutura de multiplexação e um grande número de escolhas leva a um multiplexador com muitas entradas.

Fim!