Introdução à Linguagem VHDL

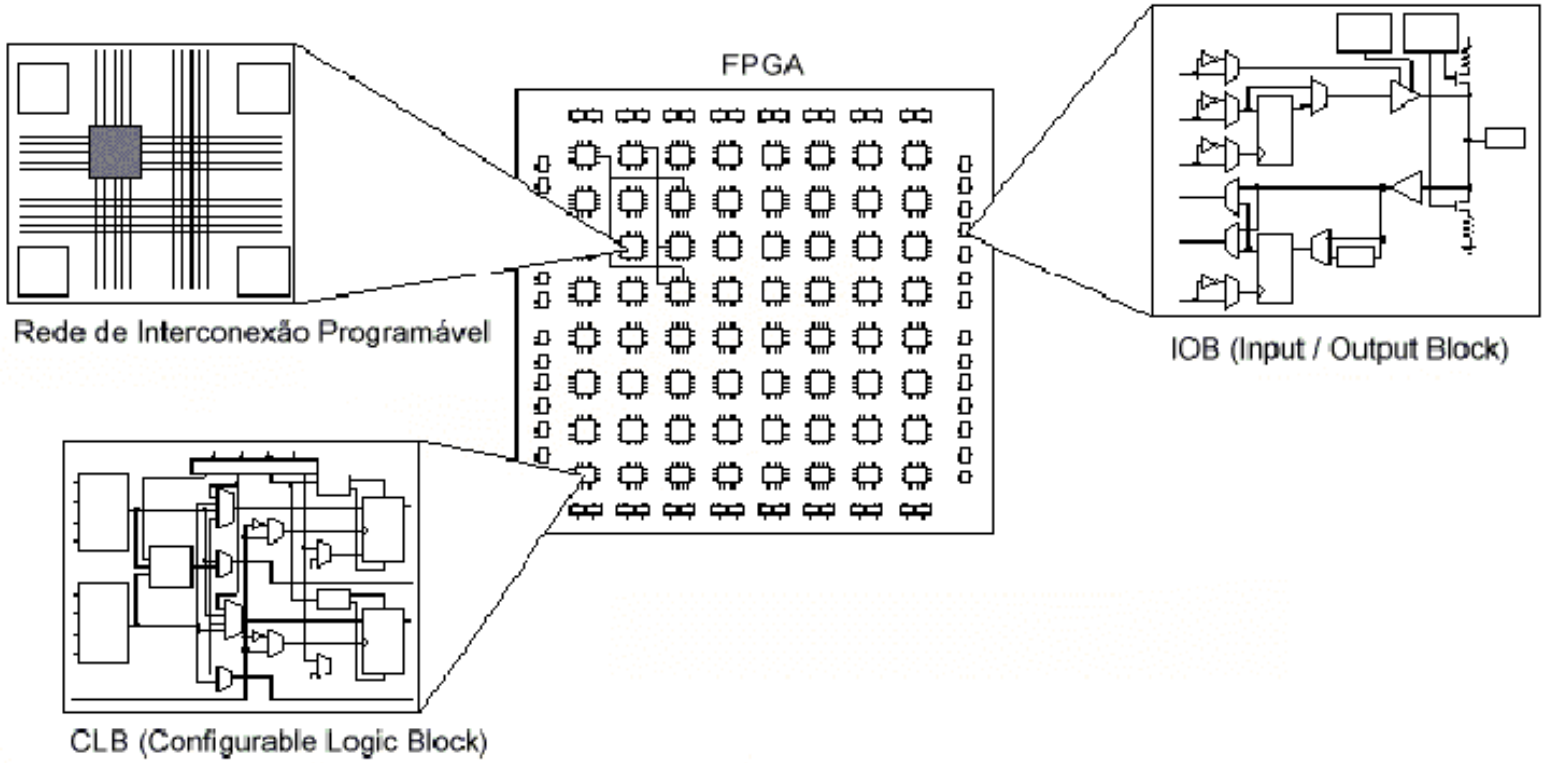
Introdução

- Ao final dos anos 70, o Departamento de Defesa dos Estados Unidos definiu um programa chamado VHSIC (*Very High Speed Integrated Circuit*).
- Em 1981, aprimorando-se as idéias do VHSIC, foi proposta uma linguagem de descrição de hardware mais genérica e flexível chamada de VHDL (*VHSIC Hardware Description Language*).
- Em 1987 se tornou um padrão pela organização internacional IEEE.
- VHDL foi projetada com princípios de programação estruturada.

FPGA e Computação Reconfigurável

- Um dos dispositivos que encontramos no mercado para implementação de circuitos/sistemas reconfiguráveis é chamado de FPGA (*Field Programmable Gate Array*.) Os FPGAs são dispositivos programáveis em campo, ou seja, podem ter sua configuração alterada após sua fabricação. Cada FPGA normalmente é composto por matrizes de elementos. Antes que cada elemento seja utilizado ele deve ser configurado. Esta configuração se faz através de um conjunto de bits chamados de *bitstream*.
- A programação do dispositivo é feita para que todos ou alguns dos componentes do dispositivo se interliguem e implementem um circuito qualquer. Esta é a implementação realizada no primeiro momento. Se houver a programação deste dispositivo novamente para implementação de um novo circuito (ou mudança no mesmo circuito) consideramos que o circuito implementado foi reconfigurado.

Arquitetura de um FPGA



- CLB (*Configurable Logic Block*): Matriz de blocos lógicos configuráveis;
- Rede de Interconexão Programável: Blocos de interconexão que interligam todos os CLBs;
- IOB (*Input Output Block*): Na periferia de todo o circuito existem blocos de entrada e saída para interface externa.

Palavras reservadas em VHDL

abs
access
after
alias
all
and
architecture
array
assert
attribute

begin
block
body
buffer
bus

case
component
configuration
constant

disconnect
downto

else
elseif
end
entity
exit

file
for
function

generate
generic
group
guarded

if
impure
in
inertial
inout
is

label
library
linkage
literal
loop

map
mod

nand
new
next
nor
not
null

of
on
open
or
others
out

package
port
postponed
procedure
process
pure

range
record
register
reject
rem
report
return
rol
ror

select
severity
shared
signal
sla
sll
sra
srl
subtype

then
to
transport
type

unaffected
units
until
use

variable

wait
when
while
with

xor
xnor

Símbolos definidos em VHDL

Symbol	Meaning
+	Addition, or positive number
–	Subtraction, or negative number
/	Division
=	Equality
<	Less than
>	Greater than
&	Concatenator
|	Vertical bar
;	Terminator
#	Enclosing based literals
(	Left parenthesis
)	Right parenthesis
.	Dot notation
:	Separates data object from type
"	Double quote
'	Single quote or tick mark
**	Exponentiation
=>	Arrow meaning "then"
=>	Arrow meaning "gets"
:=	Variable assignment
/=	Inequality
>=	Greater than or equal to
<=	Less than or equal to
<=	Signal assignment
<>	Box
--	Comment

Definindo módulos em VHDL

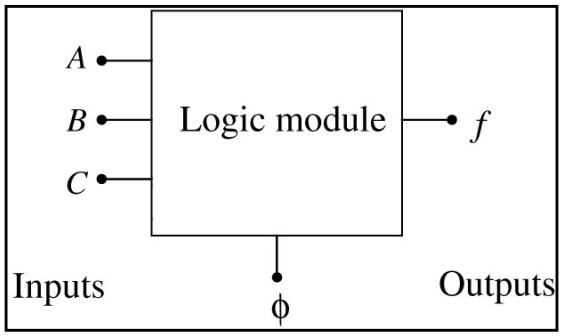


Figura 1

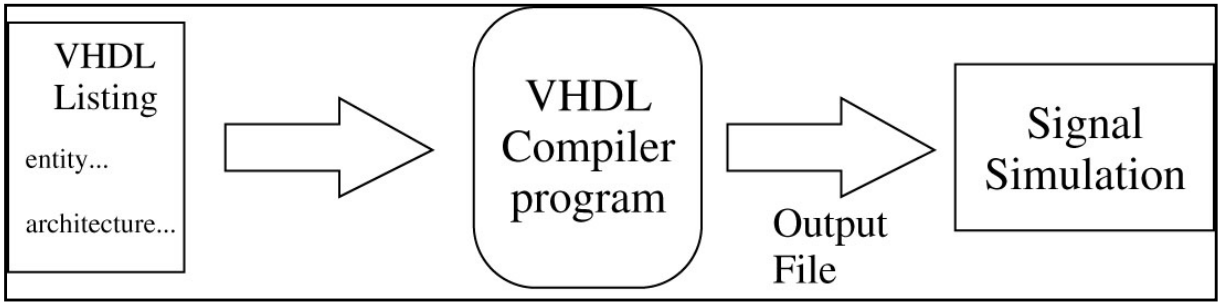


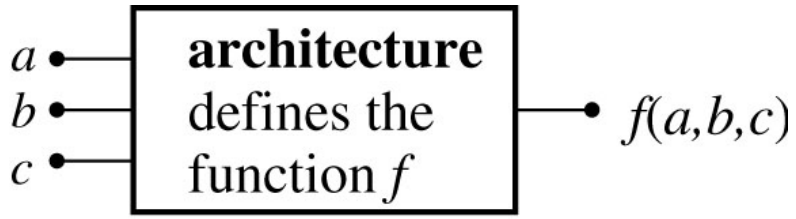
Figura 2

- Entidade (Entity): define as linhas de entrada e saída (portas).
- Arquitetura (architecture): módulo que descreve como as entradas e saídas estão relacionadas.

Outros módulos em VHDL

- Componentes (component): entidades usadas dentro de um módulo. Serve para referenciar, instanciar e replicar uma determinada entidade.
- Pacotes (package): é uma coleção de tipos, constantes, subprogramas, etc.
- Configuração (configuration): permite especificar os mínimos enlaces entre componente-entidade através da parte declarativa de uma arquitetura.
- Procedimentos (procedure) e funções (function).
- Execução concorrente: when...else... – with...select...when.
- Execução seqüencial (process): os processos são por definição concorrentes, mas o conteúdo de cada processo é executado de forma seqüencial. if...then...else – case – for – while.

Descrevendo uma *Entity*



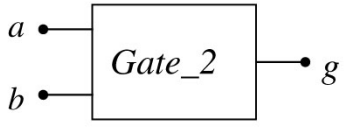
entity describes the unit

```
entity simples_porta is
      port (a, b, c: in bit;
            f: out bit);
end simples_porta;
```

Declaração de arquitetura

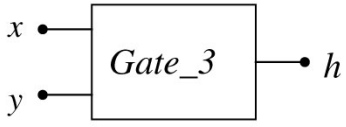
- No caso de *simples_porta* devemos definir o que f(a, b, c) é. Em VHDL é possível definir uma arquitetura de algumas formas. As principais classificações são:
  - Descrição de comportamento: é fornecido explicitamente a relação entre as entradas e saídas.
  - Descrição estrutural: são construídas funções lógicas pela combinação de elementos mais primitivos, como portas lógicas.

Exemplos de códigos em VHDL



a	b	g
0	0	1
0	1	1
1	0	1
1	1	0

(a) *Gate_2* module



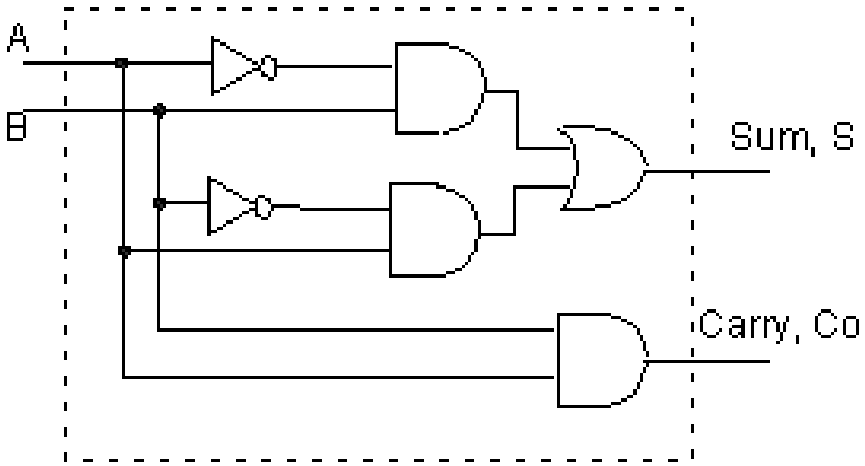
x	y	h
0	0	1
0	1	0
1	0	0
1	1	1

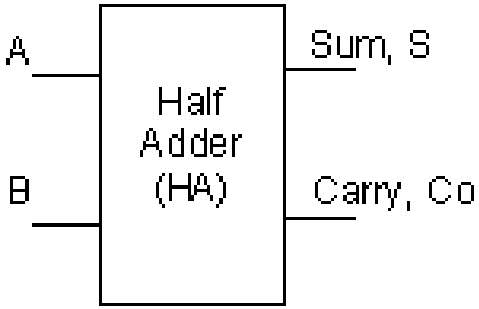
(b) *Gate_3* module

```
--exemplo 1
entity gate_2 is
        port (a, b: in bit;
                g: out bit);
end gate_2;

--declaração da arquitetura
architecture lógica of gate_2 is
begin
        g <= a nand b;
end lógica;
```

```
--exemplo 2
entity gate_3 is
        port (x, y: in bit;
                h: out bit);
end gate_3;

--declaração da arquitetura
architecture lógica of gate_3 is
begin
        h <= x xnor y;
end lógica;
```

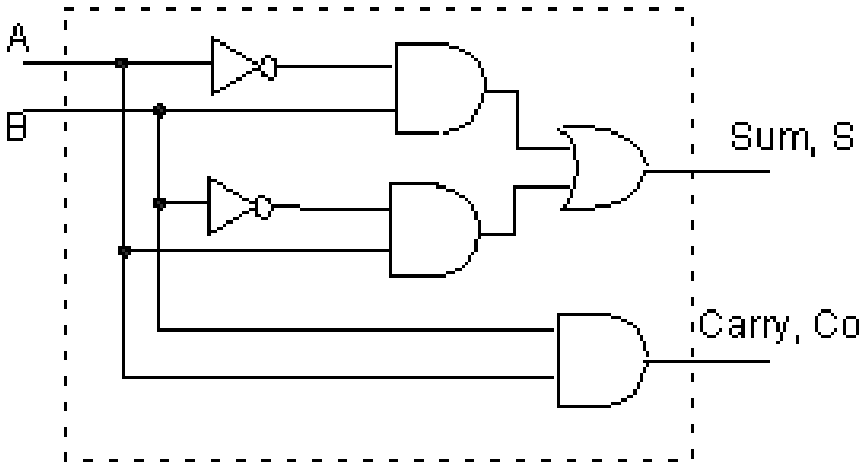
Meio Somador em VHDL

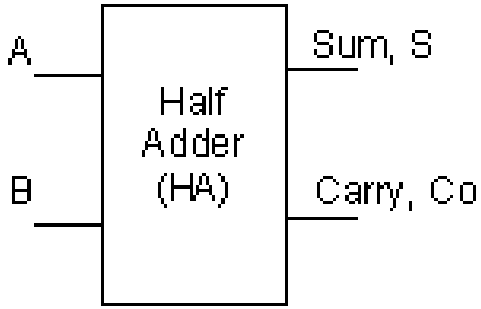





```
entity meio_somador is
     port( a : in std_logic;
           b : in std_logic;
           soma : out std_logic;
           carry : out std_logic );
end meio_somador;
```

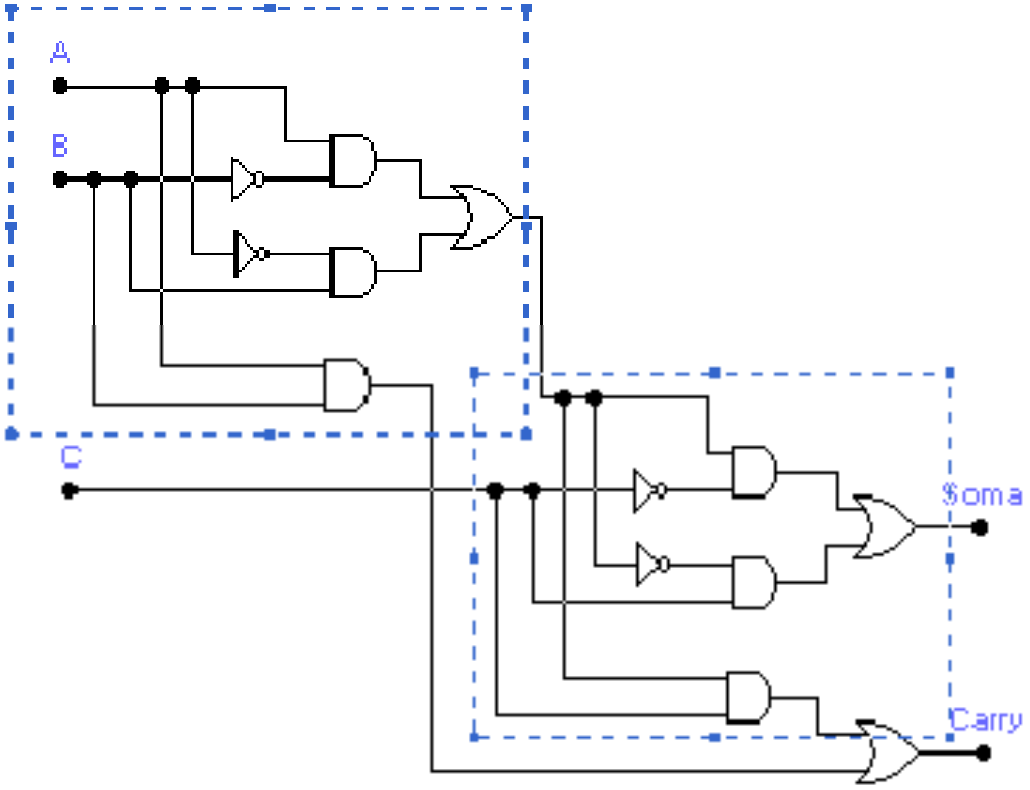
Meio Somador em VHDL

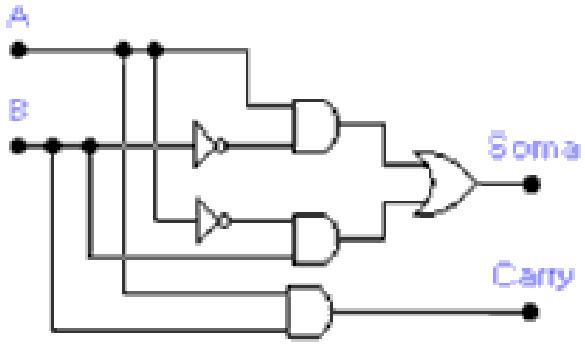


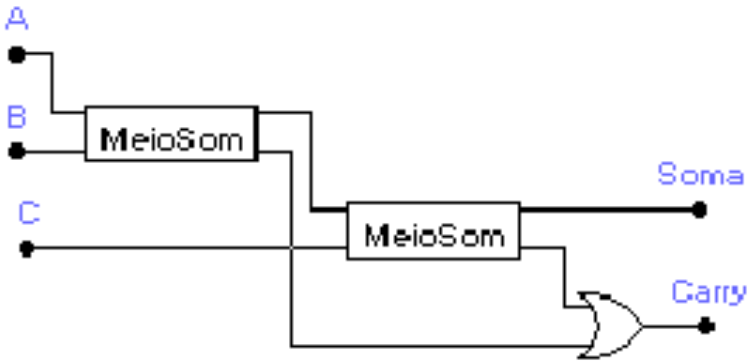



```
architecture meio_somador_arch of
  meio_somador is
begin
  soma <= (a and not b) or (not a and b);
  carry <= a and b;
end meio_somador_arch;
```

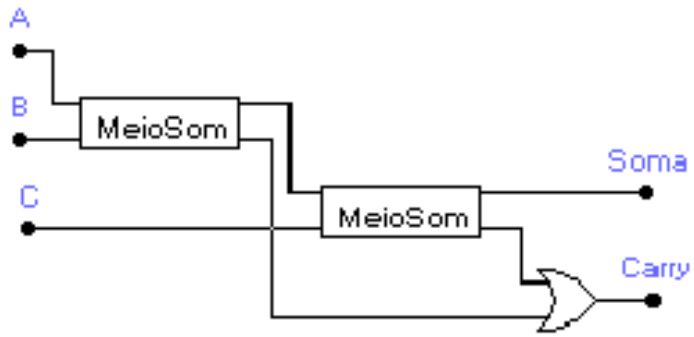
Somador completo em VHDL







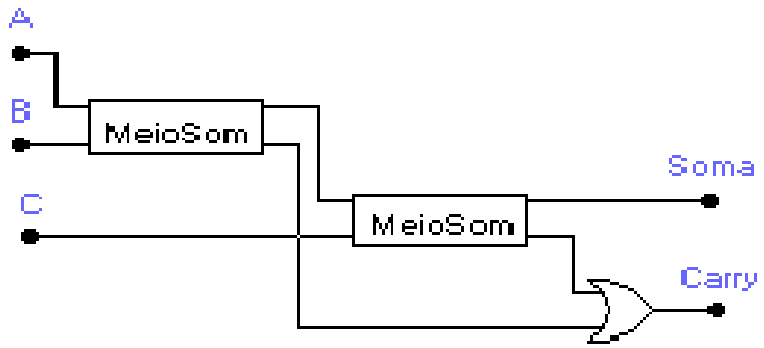
Somador completo em VHDL




```
entity somador1b is
  port(a : in std_logic;
       b : in std_logic;
       c : in std_logic;
       soma : out std_logic;
       carry : out std_logic);
end somador1b;
```

```
architecture somador1b_arch of
  somador1b is
  component meio_somador is
    port(a : in std_logic;
         b : in std_logic;
         soma : out std_logic;
         carry : out std_logic);
  end component;
    signal S_primeira_soma :
         std_logic;
    signal S_primeiro_carry :
         std_logic;
    signal S_segundo_carry :
         std_logic;
```

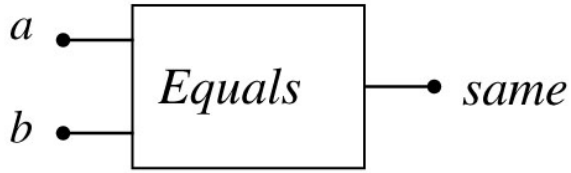
Somador completo em VHDL




```
begin
   somador1 : meio_somador
     port map (a => a,
               b => b,
               soma => S_primeira_soma,
               carry => S_primeiro_carry);

   somador2 : meio_somador
     port map (a => S_primeira_soma,
               b => c,
               soma => soma,
               carry => S_segundo_carry);

   carry <= S_primeiro_carry or S_segundo_carry;
end somador1b_arch;
```

Modelos Condicionais

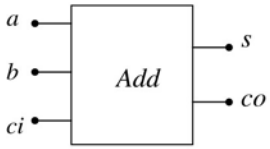
a	*b*	*same*
0	0	1
0	1	0
1	0	0
1	1	1




```
entity equals is
      port (a, b: in bit;
            same: out bit);
end equals;
architecture fluxo_dados of equals is
begin
      same <= '1' when a = b else
            '0';
end fluxo_dados;
```

Modelos Condicionais

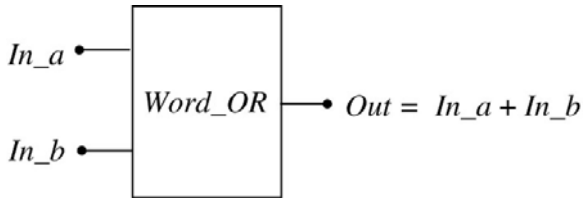
Somador completo

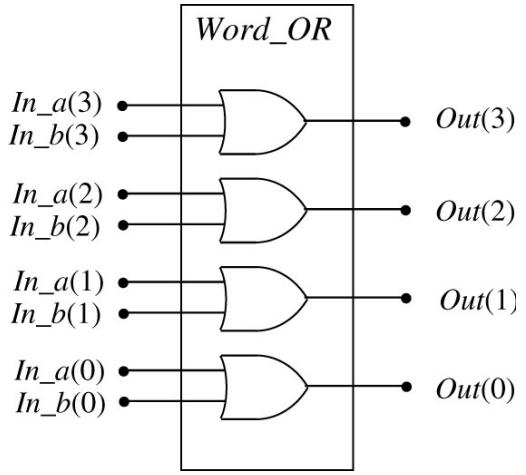


a	b	ci	s	co
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

```
entity add is
        port (a, b, ci: in bit;
              s, co: out bit);
end add;
architecture fluxo_dados of add is
begin
        s <= '1' when (a = '0' and b = '1' and ci = '0') else
             '1' when (a = '1' and b = '0' and ci = '0') else
             '1' when (a = '0' and b = '0' and ci = '1') else
             '1' when (a = '1' and b = '1' and ci = '1') else
             '0';
        co <= '1' when (a = '1' and b = '1' and ci = '0') else
              '1' when (a = '0' and b = '1' and ci = '1') else
              '1' when (a = '1' and b = '0' and ci = '1') else
              '1' when (a = '1' and b = '1' and ci = '1') else
              '0';
end fluxo_dados;
```

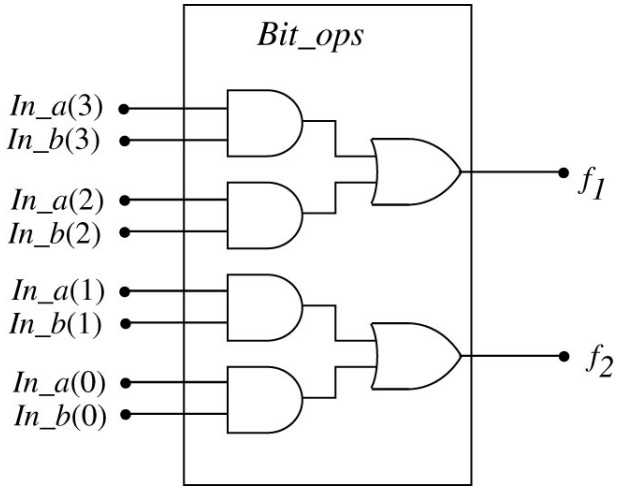
Palavras Binárias






```
entity Word_OR is
       port (In_a, In_b: in bit_vector (3 downto 0);
             Out: out bit_vector (3 downto 0));
end Word_OR;
architecture listagem of Word_OR is
begin
       Out(3) <= In_a(3) or In_b(3);
       Out(2) <= In_a(2) or In_b(2);
       Out(1) <= In_a(1) or In_b(1);
       Out(0) <= In_a(0) or In_b(0);
end listagem;
```

Palavras Binárias




```
entity bit_ops is
        port (In_a, In_b: in bit_vector (3 downto 0);
              f1, f2: out bit);
end bit_ops;
architecture Basico of bit_ops is
begin
        f1 <= (In_a(3) and In_b(3)) or (In_a(2) and In_b(2));
        f2 <= (In_a(1) and In_b(1)) or (In_a(0) and In_b(0));
end Basico;
```

Multiplexador 2x1 usando comando de execução concorrente with select

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

entity mult2x1 is
  port (e1, e2, sel: in std_logic;
                  s: out std_logic);
end mult2x1;

architecture arch_mult2x1 of mult2x1 is
begin
  with sel select
      s <= e1 when '0',
           e2 when others;
end arch_multi2x1;
```

Multiplexador 2x1 usando comando de execução seqüencial if...then...else

```
library ieee;
use ieee.std_logic_1164.all;
entity mult2x1 is
   port (e1, e2, sel: in std_logic;
                      s: out std_logic);
end mult2x1;
architecture arch_mult2x1 of mult2x1 is
begin
   process (e1, e2, sel)
   begin
        if sel = '0' then
                s <= e1;
        else
                s <= e2;
        end if;
   end process;
end arch_multi2x1;
```

Multiplexador 2x1 usando comando de execução seqüencial CASE

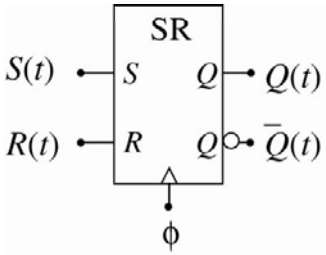
```
library ieee;
use ieee.std_logic_1164.all;
entity mult2x1 is
   port (e1, e2, sel: in std_logic;
                      s: out std_logic);
end mult2x1;
architecture arch_mult2x1 of mult2x1 is
begin
   process (e1, e2, sel)
   begin
        case sel is
               when '0' => s <= e1;
               when others => s <= e2;
        end case;
   end process;
end arch_multi2x1;
```

Multiplexador 2x1 usando portas lógicas

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

entity mult2x1 is
   port (e1, e2, sel: in std_logic;
                    s: out std_logic);
end mult2x1;

architecture arch_mult2x1 of mult2x1 is
begin
   s <= (e1 and not (sel)) or (e2 and sel);
end arch_multi2x1;
```

Descrição em VHDL para FFSR

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

entity srff is
   port (reset, set: in std_logic;
                  q: out std_logic);
end srff;

architecture arch_srff of srff is
Begin
process(set, reset)
begin
   if reset = '1' then
        q <= '0';
   if set = '1' then
        q <= '1';
end process;
end arch_srff;
```



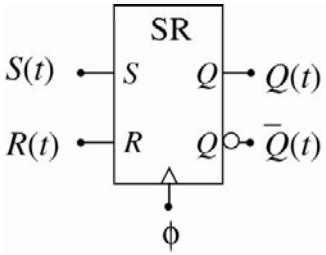
(a) Symbol

$S(t)$	$R(t)$	$Q(t+T)$	Operation
0	0	$Q(t)$	Hold
1	0	1	Set
0	1	0	Reset
1	1	?	Not used

(b) Operation summary

Descrição em VHDL para FFSR

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

entity srff is
   port (reset, set, clk: in std_logic;
                 q, qbar: out std_logic);
end srff;

architecture arch_srff of srff is
begin
process(clk)
begin
 if (clk'event and clk = '1') then
   if reset = '1' then
        q <= '0'; qbar <= '1';
   elsif set = '1' then
        q <= '1'; qbar <= '0';
   end if;
 end if;
end process;
end arch_srff;
```



(a) Symbol

$S(t)$	$R(t)$	$Q(t+T)$	Operation
0	0	$Q(t)$	Hold
1	0	1	Set
0	1	0	Reset
1	1	?	Not used

(b) Operation summary

Contador de 4 bits

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity cont4bits is
   port (clk: in std_logic;
         dout: out std_logic_vector (3 downto 0));
end cont4bits;

architecture arch_cont4bits of cont4bits is
signal i: std_logic_vector (3 downto 0);
begin
process(clk)
begin
   if clk`event and clk = '1' then
        i <= i + '1';
   end if;
end process;
Dout <= i;
end arch_cont4bits;
```