

Introdução ao VHDL

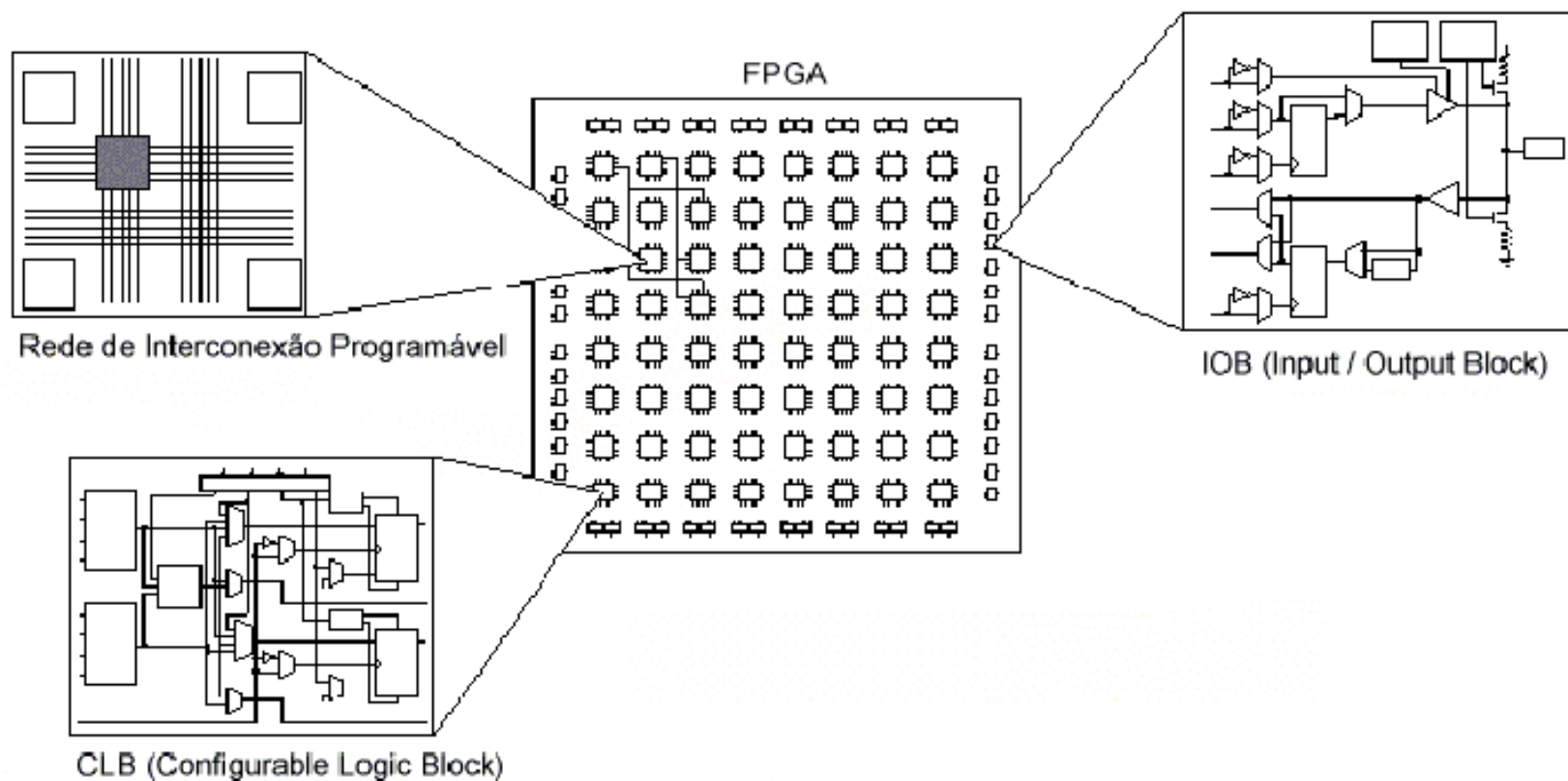
Introdução ao VHDL

- 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.

Prototipação em FPGA

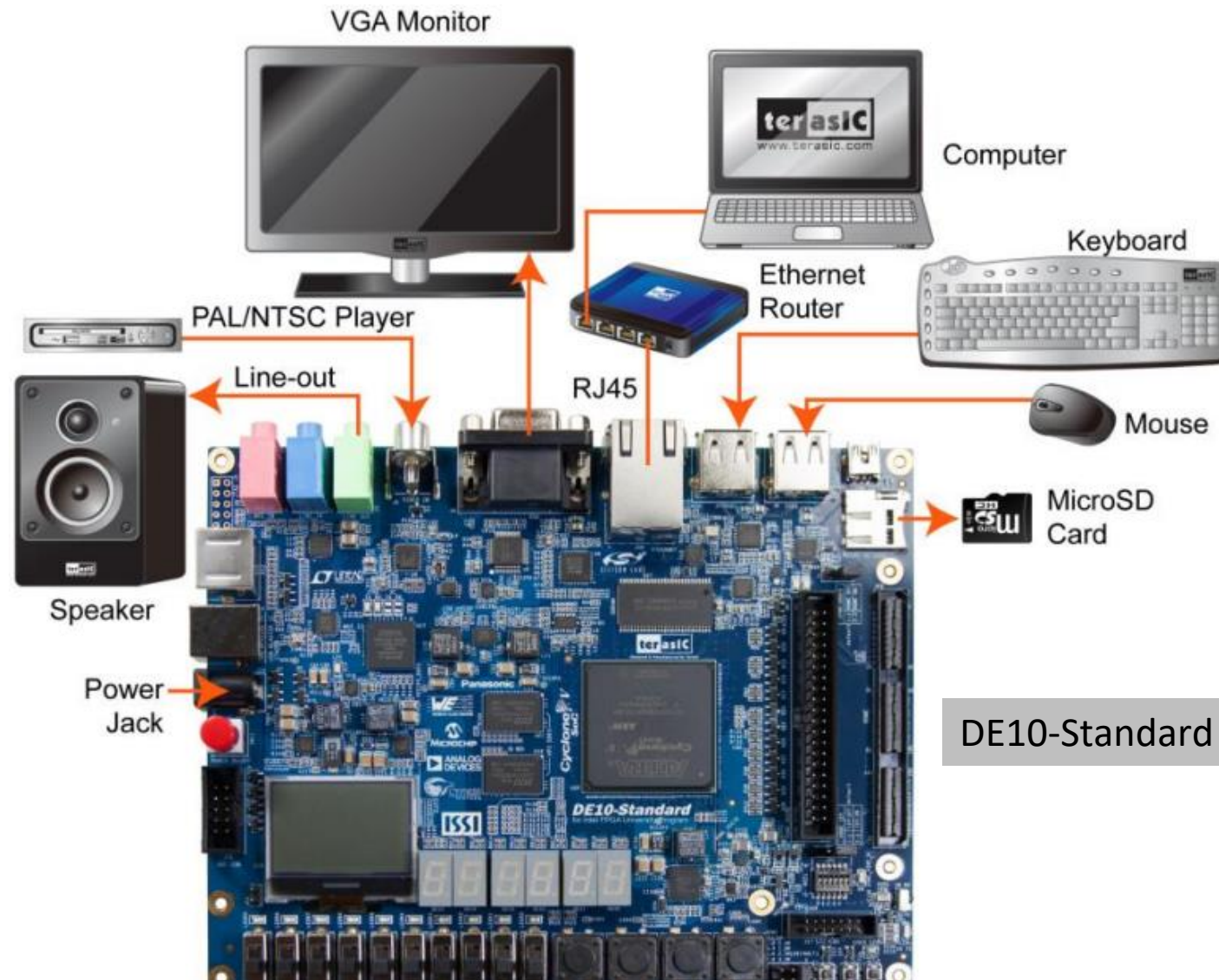
- Um dos dispositivos que encontramos no mercado para implementação de circuitos/sistemas reconfiguráveis é chamado de FPGA (*Field Programmable Gate Array*) Os FPGAs são dispositivos programáveis em campo, ou seja, podem ter sua configuração alterada após sua fabricação. Cada FPGA normalmente é composto 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.

Partes 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.

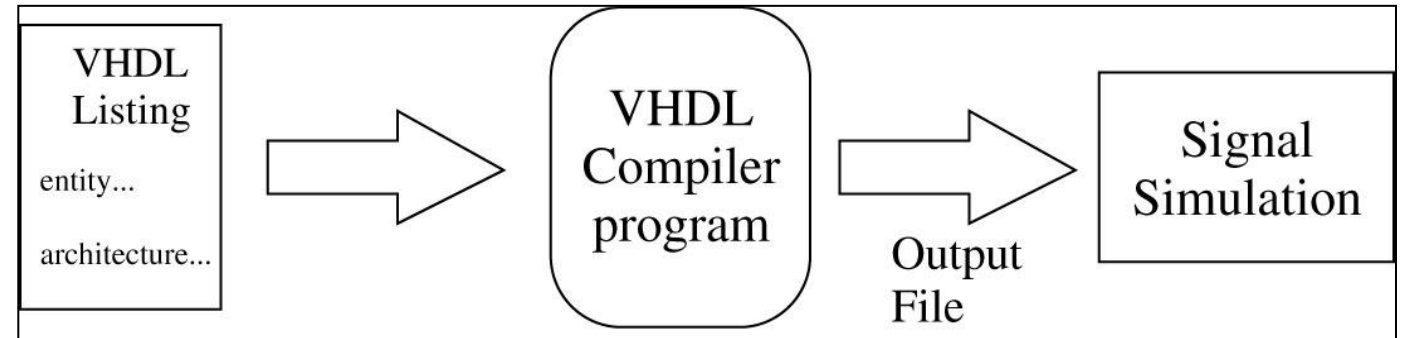
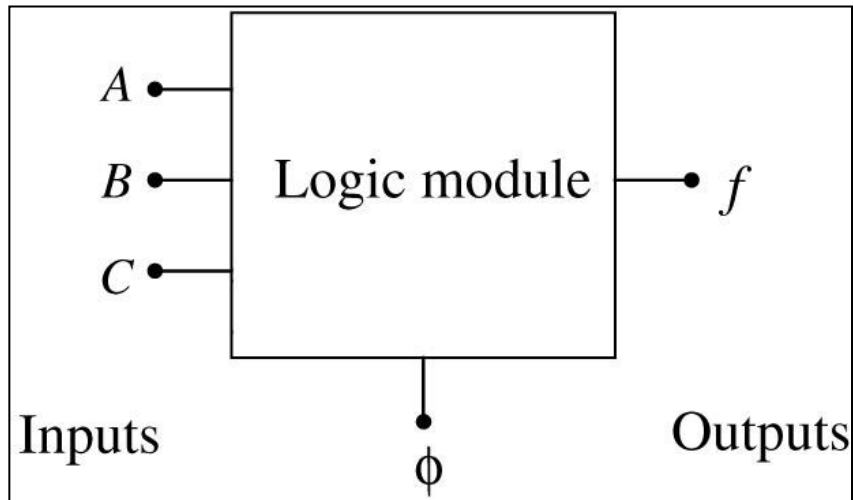
Exemplo de um kit FPGA de bancada



Palavras reservadas e símbolos em VHDL

				Symbol	Meaning
abs	file	of	sra	+	Addition, or positive number
access	for	on	srl	–	Subtraction, or negative number
after	function	open	subtype	/	Division
alias		or		=	Equality
all	generate	others	then	<	Less than
and	generic	out	to	>	Greater than
architecture	group		transport	&	Concatenator
array	guarded	package	type		Vertical bar
assert		port		;	Terminator
attribute	if	postponed	unaffected	#	Enclosing based literals
	impure	procedure	units	(Left parenthesis
begin	in	process	until)	Right parenthesis
block	inertial	pure	use	.	Dot notation
body	inout			:	Separates data object from type
buffer	is	range	variable	"	Double quote
bus		record		'	Single quote or tick mark
	label	register	wait	**	Exponentiation
case	library	reject	when	=>	Arrow meaning "then"
component	linkage	rem	while	=>	Arrow meaning "gets"
configuration	literal	report	with		
constant	loop	return			
		rol	xor		
disconnect	map	ror	xnor		
downto	mod			:=	Variable assignment
				/=	Inequality
	nand	select		>=	Greater than or equal to
else	new	severity		<=	Less than or equal to
elseif	next	shared		<=	Signal assignment
end	nor	signal		<>	Box
entity	not	sla			
exit	null	sll		--	Comment

Módulos em VHDL

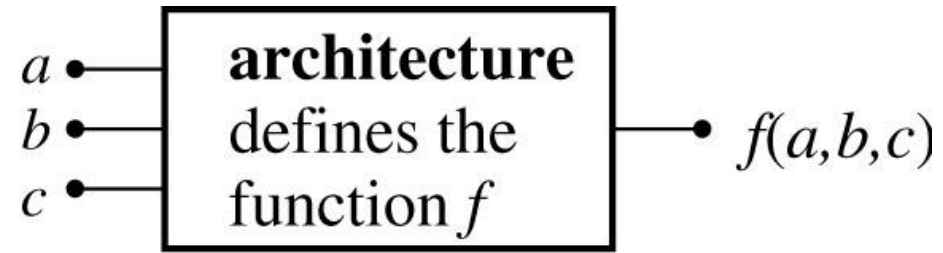


- Entidade (Entity): define as linhas de entrada e saída (portas).
- Arquitetura (architecture): módulo que descreve como as entradas e saídas estão relacionadas.

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 sequencial (**process**): os processos são por definição concorrentes, mas o conteúdo de cada processo é executado de forma sequencial. if...then...else – case – for – while.

Descrevendo um Entity



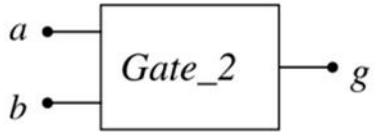
entity describes the unit

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

Descrevendo uma arquitetura

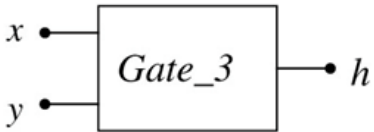
- No caso de *simples_porta* devemos definir o que $f(a, b, c)$ é. Em VHDL é possível definir uma arquitetura de algumas formas. As principais classificações são:
 - Descrição de comportamento: é fornecido explicitamente a relação entre as entradas e saídas.
 - Descrição estrutural: são construídas funções 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 std_logic;  
          g: out std_logic);  
end gate_2;
```

--declaração da arquitetura

```
architecture lógica of gate_2 is  
begin  
    g <= a nand b;  
end lógica;
```

--exemplo 2

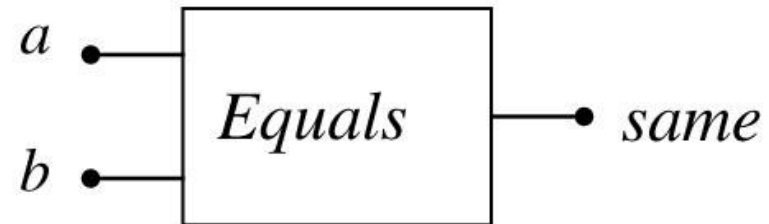
```
entity gate_3 is  
    port (x, y: in std_logic;  
          h: out std_logic);  
end gate_3;
```

--declaração da arquitetura

```
architecture lógica of gate_3  
    is  
begin  
    h <= x xnor y;  
end lógica;
```

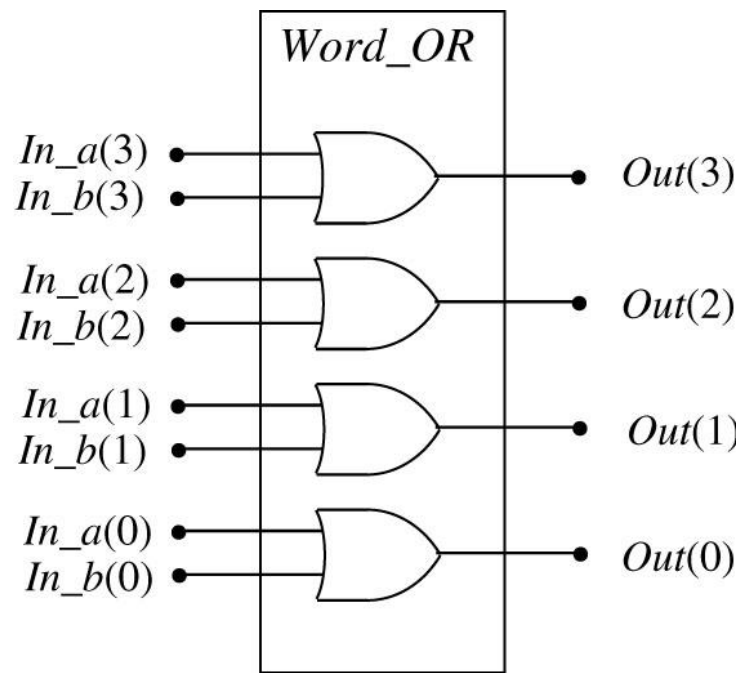
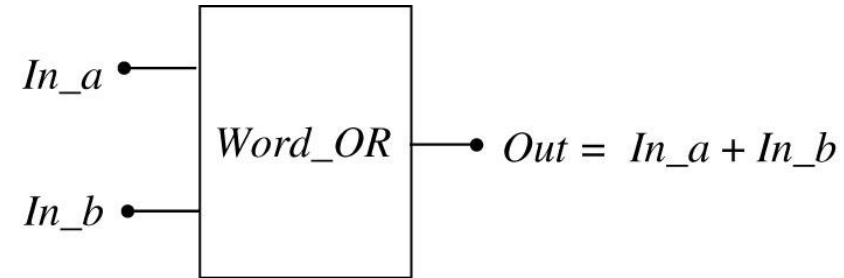
Exemplos de códigos em VHDL

<i>a</i>	<i>b</i>	<i>same</i>
0	0	1
0	1	0
1	0	0
1	1	1



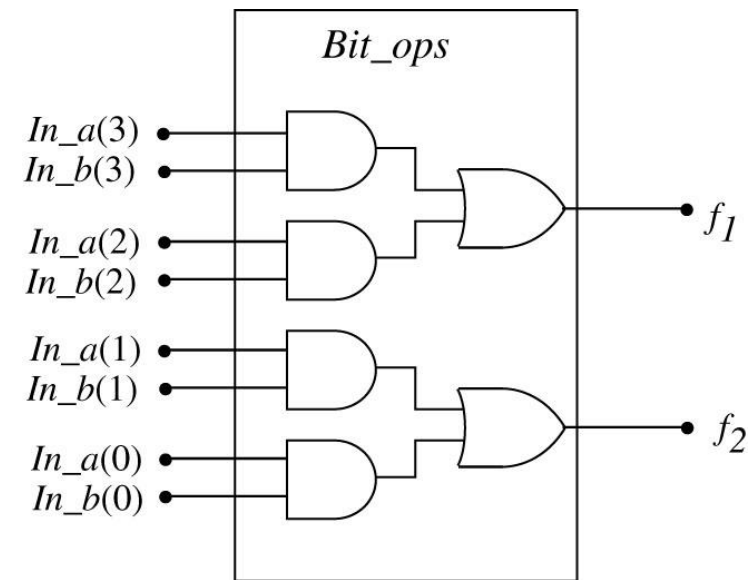
```
entity equals is  
    port (a, b: in std_logic;  
           same: out std_logic);  
end equals;  
architecture fluxo_dados of equals is  
begin  
    same <= '1' when a = b else  
           '0';  
end fluxo_dados;
```

Exemplos de códigos em VHDL



```
entity Word_OR is  
    port (In_a, In_b: in std_logic_vector (3 downto 0);  
          Out: out std_logic_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;
```

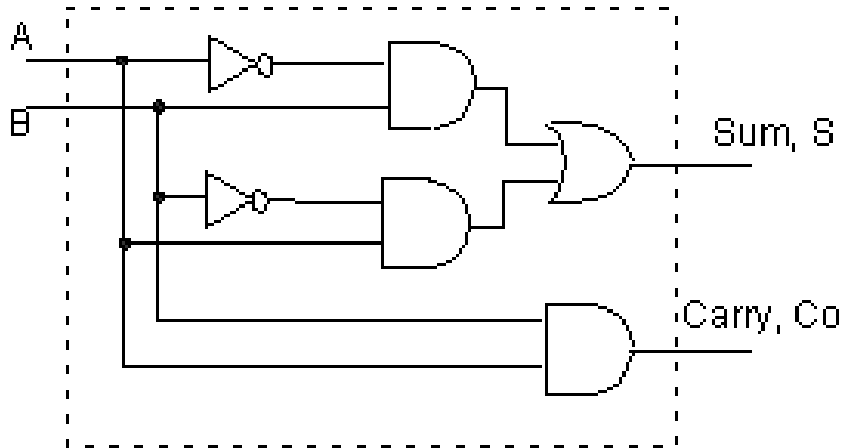
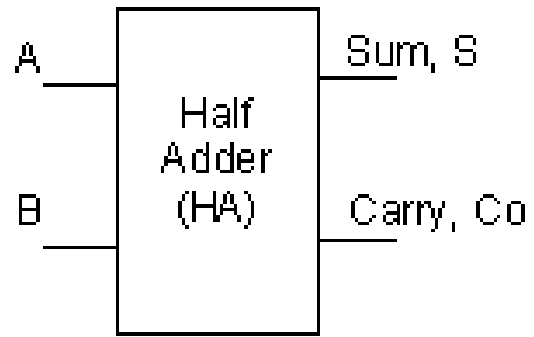
Exemplos de códigos em VHDL



```
entity bit_ops is
    port (In_a, In_b: in std_logic_vector (3 downto 0);
          f1, f2: out std_logic);
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;
```

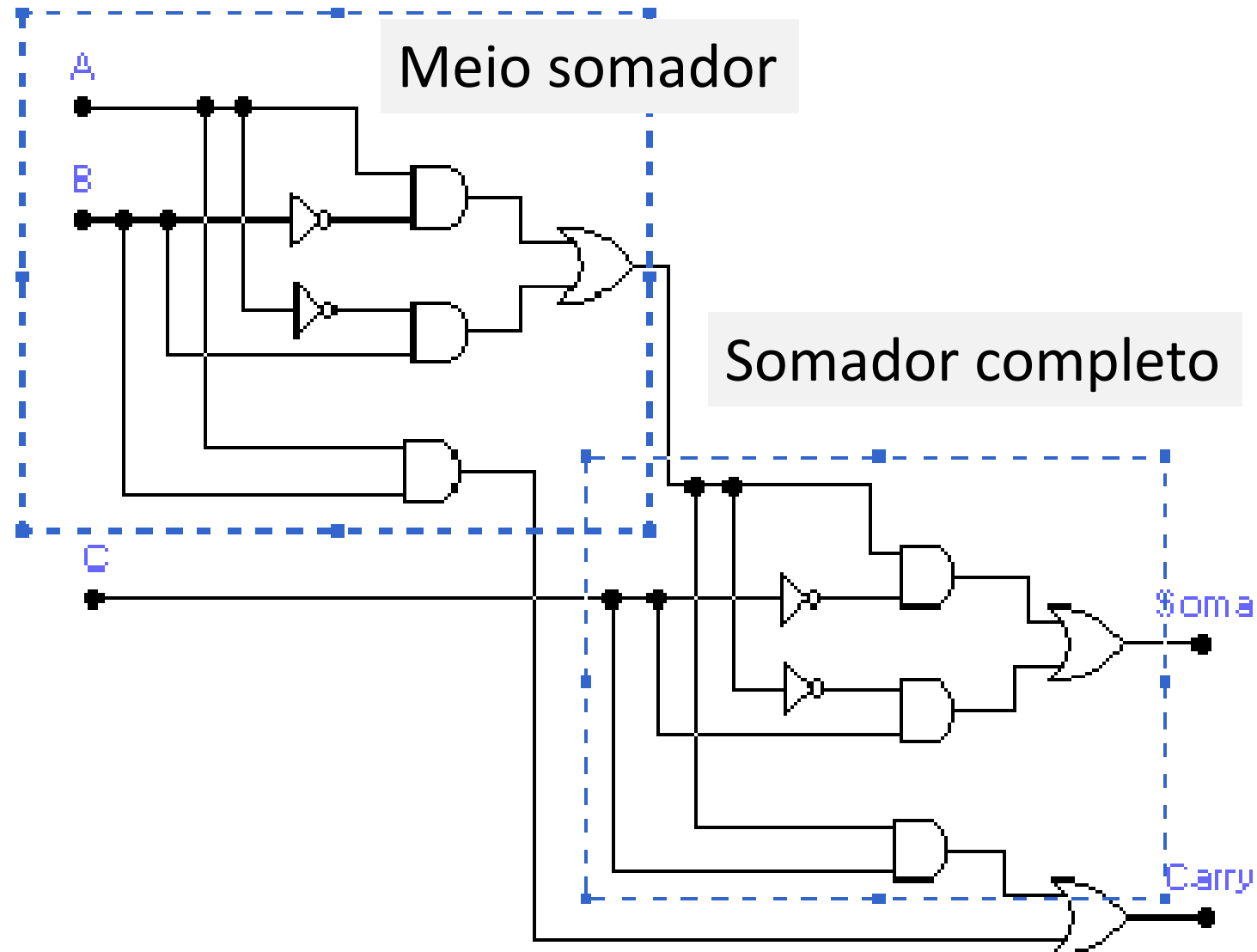
Meio Somador em VHDL



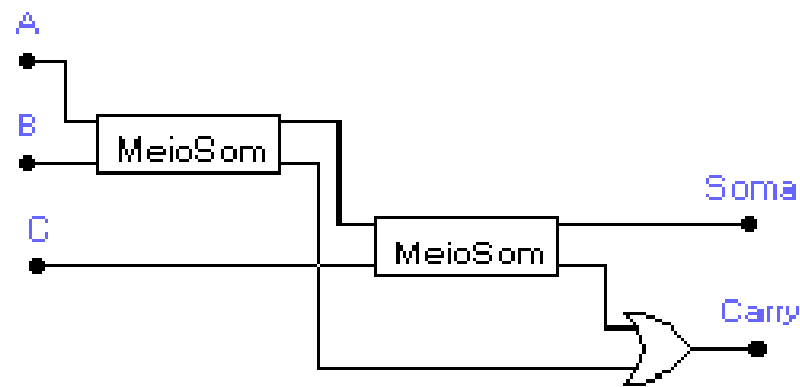
```
entity meio_somador is  
    port( a : in std_logic;  
          b : in std_logic;  
          soma : out std_logic;  
          carry : out std_logic );  
end meio_somador;
```

```
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 (parte 1)



```
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;

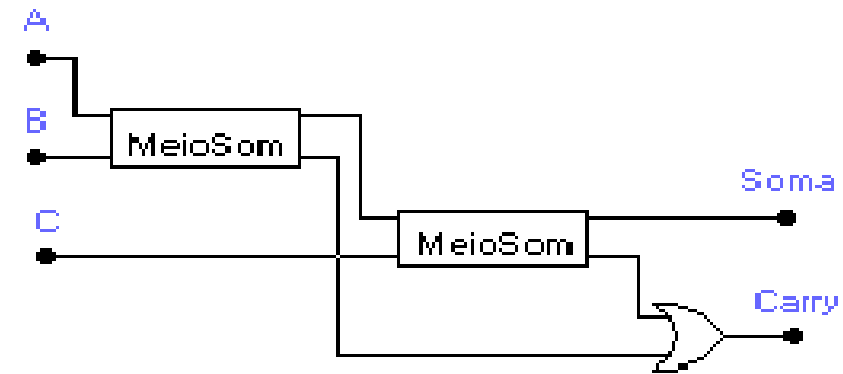
    signal S_primeira_soma :
        std_logic;
    signal S_primeiro_carry :
        std_logic;
    signal S_segundo_carry :
        std_logic;
```

Somador completo em VHDL (parte 2)

```
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;
```

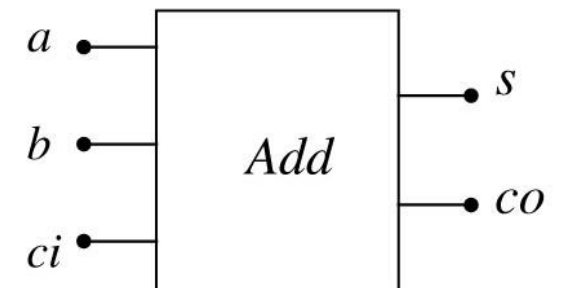


Somador completo em VHDL

```
entity add is
    port (a, b, ci: in std_logic;
          s, co: out std_logic);
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;
```

Somador completo

<i>a</i>	<i>b</i>	<i>ci</i>	<i>s</i>	<i>co</i>
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



Multiplexador 2x1 - 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_mult2x1;
```

Multiplexador 2x1 - execução sequencial *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_mult2x1;
```

Multiplexador 2x1 - execução sequencial *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_mult2x1;
```

Multiplexador 2x1 – 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_mult2x1;
```

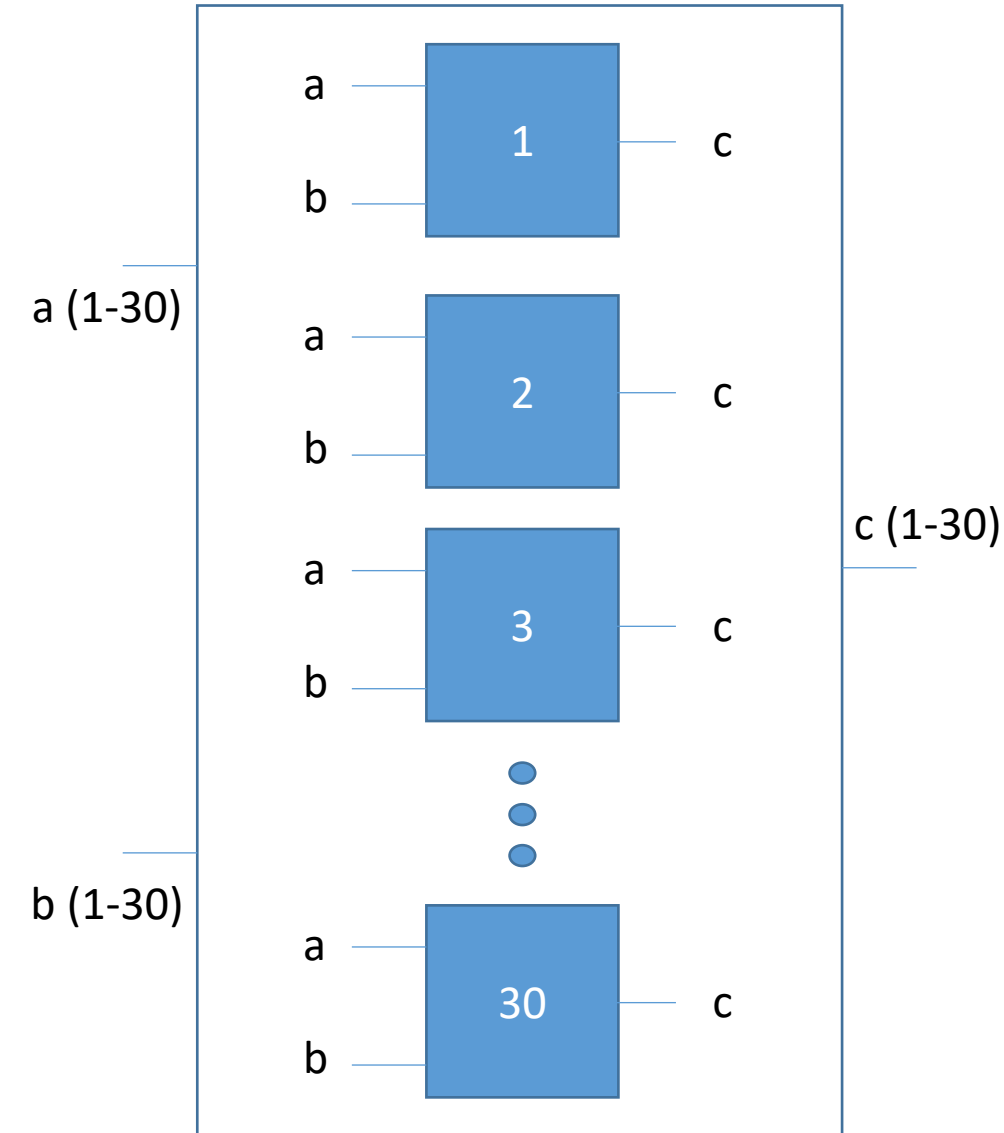
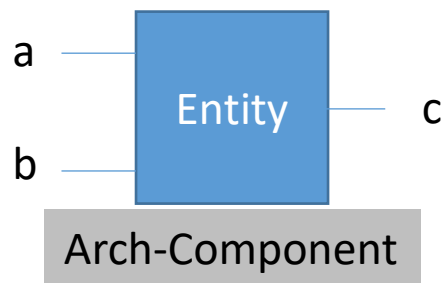
Como usar um *for* em VHDL

...

É realizada uma replicação espacial do circuito

begin

```
TESTE_1_30: for i in 1 to 30 generate  
  Nome_Map : Nome_Component  
  port map (a => a(i),  
             b => b(i),  
             c => c(i));  
end generate;  
end architecture
```



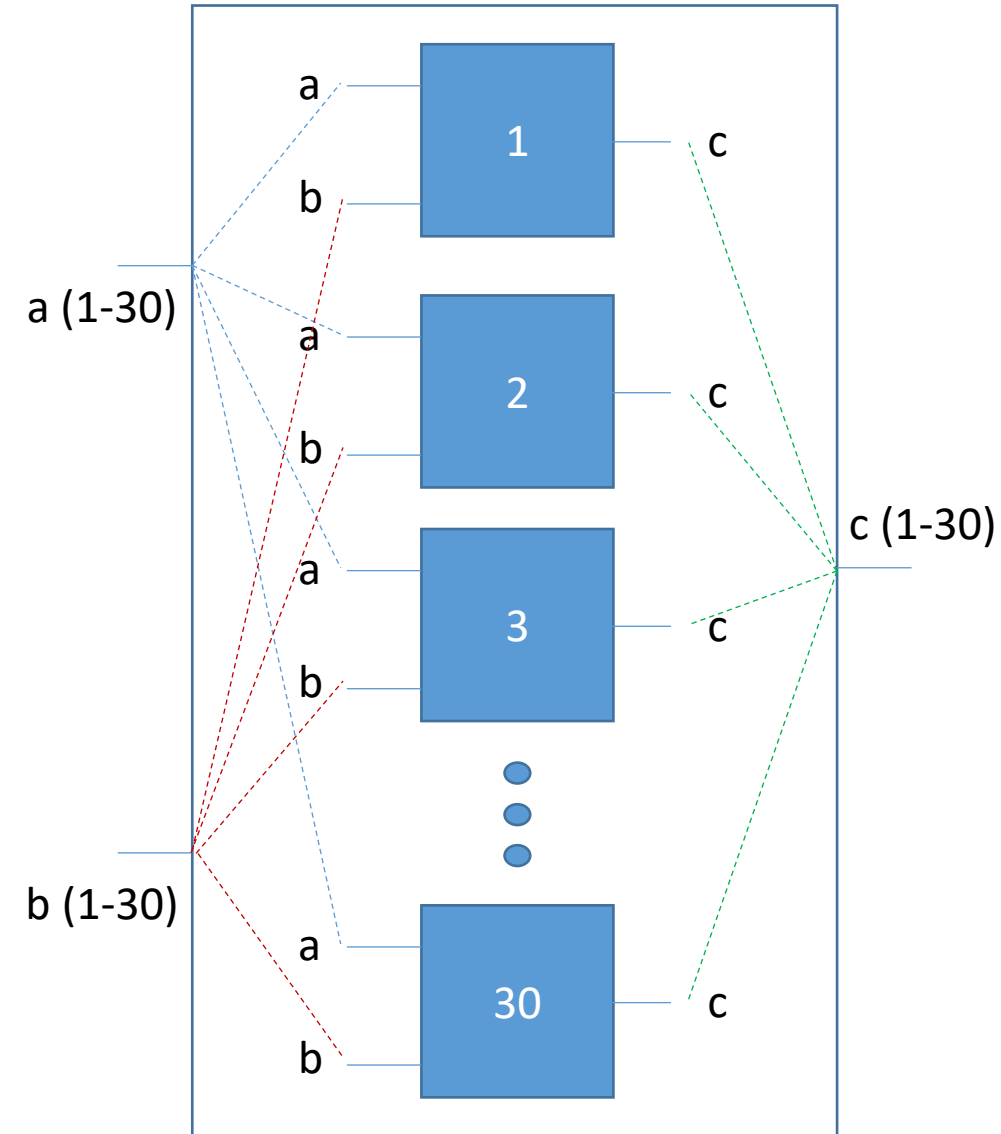
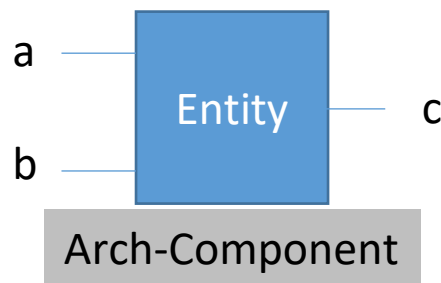
Como usar um *for* em VHDL

...

É realizada uma replicação espacial do circuito

begin

```
TESTE_1_30: for i in 1 to 30 generate  
  Nome_Map : Nome_Component  
  port map (a => a(i),  
             b => b(i),  
             c => c(i));  
end generate;  
end architecture
```



Arquivo de teste (*testbench*)

- Depois de finalizar a descrição do *hardware* ¹ para realizar uma simulação, é necessário descrever o *testbench*. O arquivo de descrição do *testbench* contém os dados de estímulo de entrada para seu circuito. ² ³

```
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_mult2x1;
```

1

Mux2X1.vhd

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

entity mult2x1_tb is
end mult2x1_tb;

architecture testmult2x1 of mult2x1_tb is
    component mult2x1
        port(e1, e2, sel: in std_logic;
             s: out std_logic);
    end component;

    signal e1 : std_logic;
    signal e2 : std_logic;
    signal sel : std_logic;
    signal s : std_logic;
```

2

TestMux.vhd

```
begin
    uut: mult2x1 port map(
        e1 => e1;
        e2 => e2;
        sel => sel;
        s => s; );

    tb: process
    begin
        wait for 10 ns;
        e1 <= '1';
        e2 <= '1';
        sel <= '1';
        wait for 10 ns;
    end process
end testmult2x1
```

3

continuação

Arquivo de teste (*testbench*)

- Depois de finalizar a descrição do *hardware* ¹ para realizar uma simulação, é necessário descrever o *testbench*. O arquivo de descrição do *testbench* contém os dados de estímulo de entrada para seu circuito. ² ³

```
library ieee;  
use ieee.std_logic_1164.all;  
entity mult2x1 is  
    port (e1, e2, sel: in std_logic;
```

Mux2X1.vhd

```
library ieee;  
use ieee.std_logic_1164.all;  
entity mult2x1_tb is  
end mult2x1_tb;
```

TestMux.vhd

```
begin  
    uut: mult2x1 port map(  
        e1 => e1;  
        e2 => e2;  
        sel => sel;
```

continuação

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

1

```
        s: out std_logic);  
end component;  
signal e1 : std_logic;  
signal e2 : std_logic;  
signal sel : std_logic;  
signal s : std_logic;
```

2

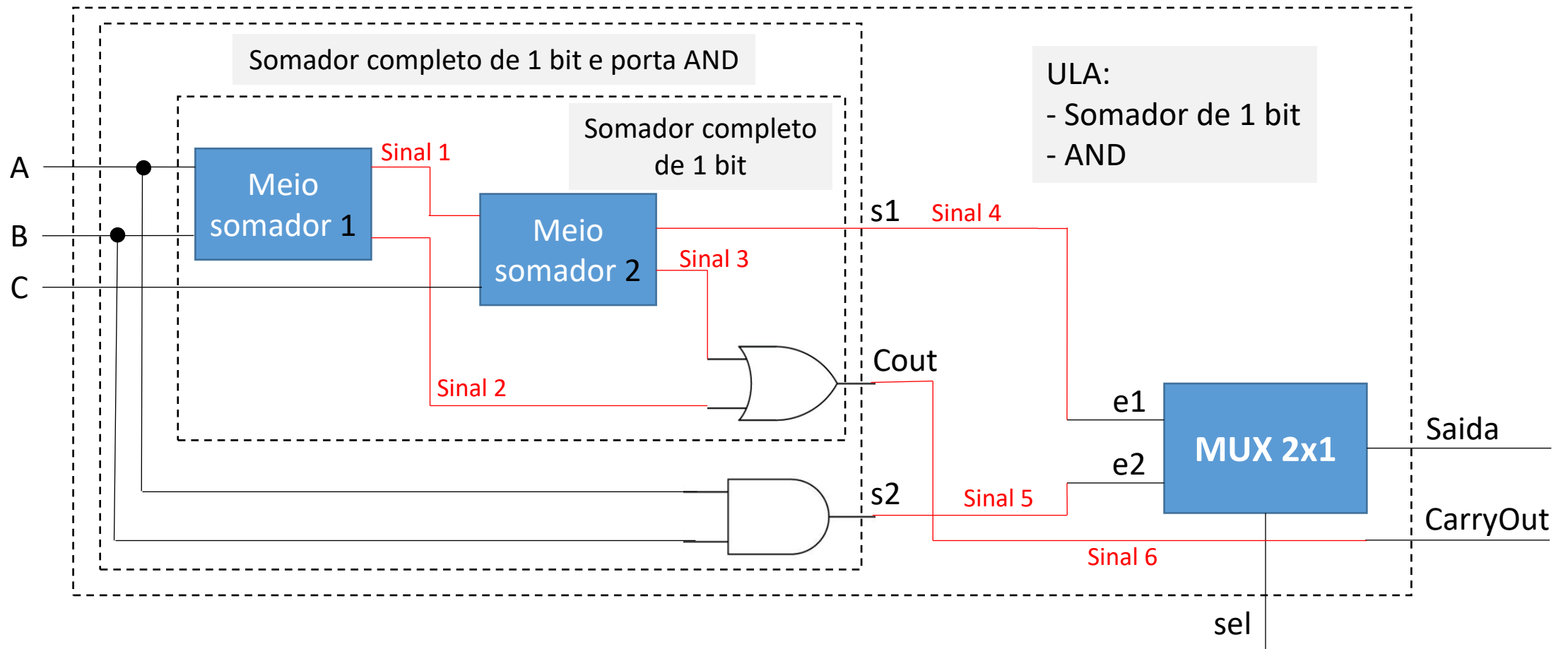
```
wait for 10 ns;  
e1 <= '1';  
e2 <= '1';  
sel <= '1';  
wait for 10 ns;  
end process  
end testmult2x1
```

3

Há alguns erros de digitação neste código.

Serão úteis para discussão em um primeiro exercício com simulador.

Exercício de VHDL



EDA Playground (https://www.edaplayground.com/home)

← → ↻ 🏠 edaplayground.com/x/A4 ☆ ⋮

EDA playground Run Copy* Aldec Riviera Pro 2020.04 is now available. It supports some VHDL-2019. Examples [here](#) and [here](#). ? 🧪 ↗ Playgrounds ▾ 👤 Profile ▾

Brought to you by **DOULOS**

▼ Languages & Libraries

Testbench + Design

VHDL ▾

Libraries ⓘ

None
OVL 2.8.1
OSVVM

Top entity

testbench

☐ Enable VUnit ⓘ

▼ Tools & Simulators ⓘ

Aldec Riviera Pro 2020.04 ▾

Compile Options

-2008

Run Options

Run Options

Run Time: 10 ms

☒ Open EPWave after run

☐ Download files after run

▼ Examples

VHDL
Verilog/SystemVerilog
UVM

testbench.vhd +

VHDL Testbench

```
37
38 a_in <= '1';
39 b_in <= 'X';
40 wait for 1 ns;
41 assert(q_out='1') report "Fail 1/X" severity error;
42
43 a_in <= '1';
44 b_in <= '1';
45 wait for 1 ns;
46 assert(q_out='1') report "Fail 1/1" severity error;
47
48 -- Clear inputs
49 a_in <= '0';
50 b_in <= '0';
51
52 assert false report "Test done." severity note;
53 wait;
54 end process;
55 end tb;
56
```

design.vhd +

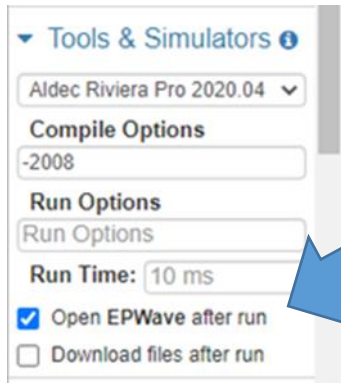
VHDL Design

```
1 -- Simple OR gate design
2 library IEEE;
3 use IEEE.std_logic_1164.all;
4
5 entity or_gate is
6 port(
7   a: in std_logic;
8   b: in std_logic;
9   q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13 begin
14   process(a, b) is
15   begin
16     q <= a or b;
17   end process;
18 end rtl;
19
```

Log Share

```
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 5400 kB (elbread=427 elab2=4829 kernel=142 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL: PLI/VHPI kernel's engine initialization done.
# PLI: Loading library '/usr/share/Riviera-PRO/bin/libsysstf.so'
# EXECUTION:: NOTE : Test done.
# EXECUTION:: Time: 4 ns, Iteration: 0, Instance: /testbench, Process: line__26.
# KERNEL: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./dump.vcd
[2020-09-21 21:31:14 EDT] Opening EPWave...
Done
```

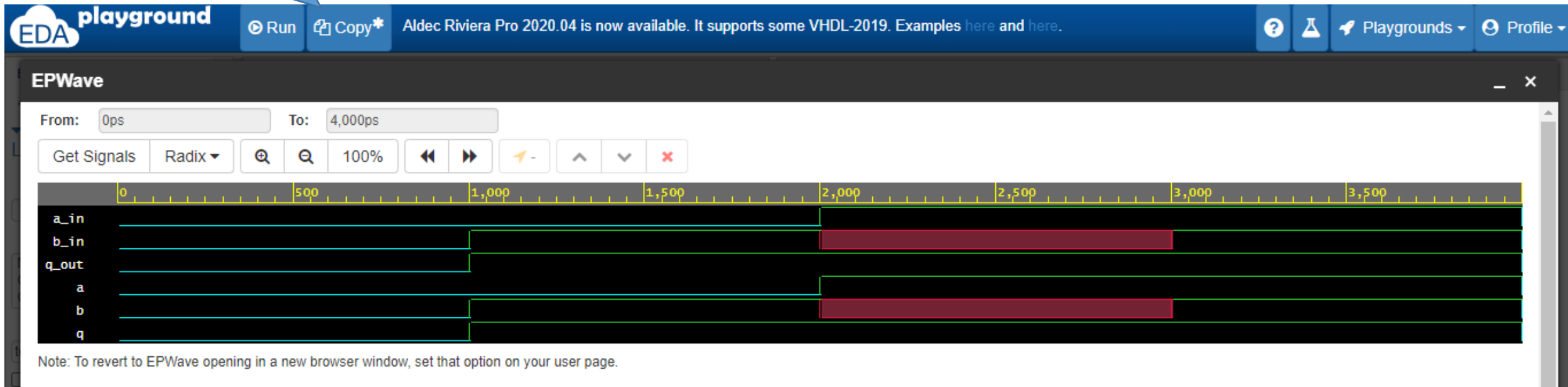
EDA Playground (<https://www.edaplayground.com/home>)



Manter habilitado.
Em seguida clicar em Run.

Esta simulação vai aparecer

3



Symphony EDA (<http://www.symphonyeda.com/>)

*Escolher opção
free license no setup
de configuração*

The screenshot displays the Symphony EDA Sonata software interface, which is used for digital logic simulation. The main window is titled "dennis - Symphony EDA Sonata - [Waveform0.wfs *]". It features a menu bar (File, Edit, Project, Compile, Simulate, Window, Help) and a toolbar. The left pane shows the project hierarchy with the top-level entity "tb_vhd_psi_lpm05". The center pane displays a list of signals and their current values, such as "bit_rate_clk_in" (0), "ref_clk_in_p" (1), and "xmit_data_sel0_in" (0). The right pane shows a waveform simulation with a time scale of 1 us. The waveform displays various digital signals over time, with specific time points like 270ns, 275ns, 280ns, 285ns, 290ns, 295ns, and 300ns marked. A cursor is positioned at 311012.725ps. Below the waveform, a console window shows simulation output, including an assertion warning: "ASSERT: WARNING a". The bottom right pane shows the VHDL code for the "adv_virtex" entity, which includes library declarations and entity definitions. A context menu is open over the code, offering options like "Open 't2.vhd', line 56", "Compile 't2.vhd'", "Remove 't2.vhd' from Library", "Set 'adv_cdreger(fast)' as top level", and "Settings...".

Workspace: Work = dennis
Toplevel = tb_vhd_psi_lpm05

Workspace: Work = adv_virtex
Toplevel = top

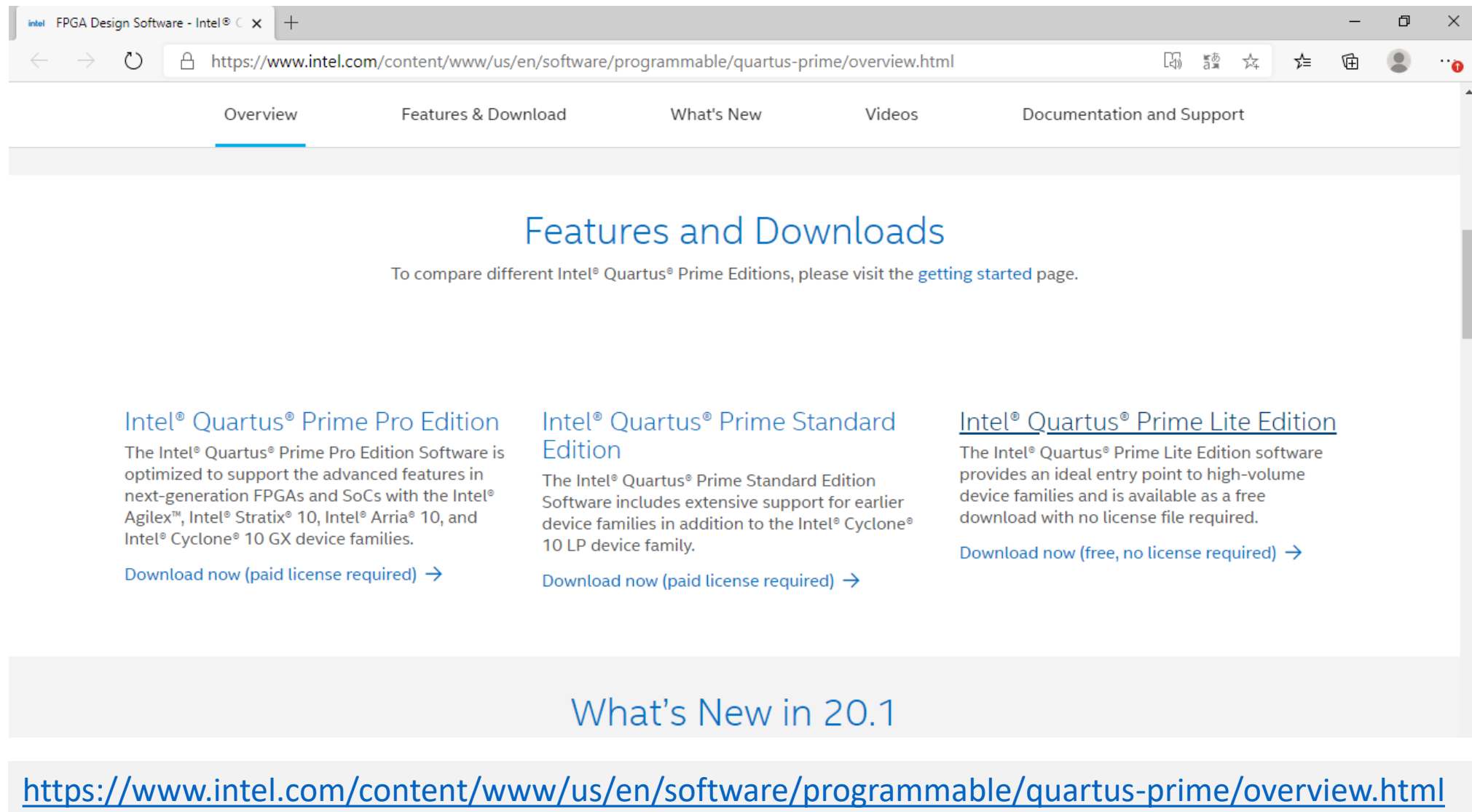
Library ieee => \$SYMPHONYEDA
Library synplify => c:/Test/Patrick
Library adv_virtex => c:/Test/Patrick
adv_virtex
adv_virtex(body)
t2.vhd (Dir: C:/Test/Patrick)
adv_cdreger
adv_cdreger(fast)
top
top(arch)

Open 't2.vhd', line 56
Compile 't2.vhd'
Remove 't2.vhd' from Library
Set 'adv_cdreger(fast)' as top level
Settings...

At tb_vhd_psi_lpm05.vhd: (line 552)
Instance = :tb_vhd_psi_lpm05(archtb_serdes_test):test:chk_rcvd_par_out:
Instance
ASSERT: WARNING a
At tb_vhd_psi_lpm05.vhd: (line 552)
Instance
Simulation stoppe
Elapsed Time: 00h
% ls -lrt

Scanning file tb_vhd_psi_lpm05.vhd

Intel Quartus Prime Lite Edition – FPGAs Intel



The screenshot shows a web browser window with the URL <https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html>. The page has a navigation bar with links: Overview, Features & Download, What's New, Videos, and Documentation and Support. The 'Overview' link is active. The main content area is titled 'Features and Downloads' and includes a sub-header 'To compare different Intel® Quartus® Prime Editions, please visit the [getting started](#) page.' Below this, there are three columns of information:

Intel® Quartus® Prime Pro Edition	Intel® Quartus® Prime Standard Edition	Intel® Quartus® Prime Lite Edition
The Intel® Quartus® Prime Pro Edition Software is optimized to support the advanced features in next-generation FPGAs and SoCs with the Intel® Agilex™, Intel® Stratix® 10, Intel® Arria® 10, and Intel® Cyclone® 10 GX device families.	The Intel® Quartus® Prime Standard Edition Software includes extensive support for earlier device families in addition to the Intel® Cyclone® 10 LP device family.	The Intel® Quartus® Prime Lite Edition software provides an ideal entry point to high-volume device families and is available as a free download with no license file required.
Download now (paid license required) →	Download now (paid license required) →	Download now (free, no license required) →

Below the comparison table, there is a section titled 'What's New in 20.1' and a large blue link: <https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html>

Xilinx ISE WebPACK – FPGAs Xilinx

The screenshot shows the Xilinx website's product page for ISE WebPACK Design Software. The browser's address bar shows the URL: <https://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html>. The page features a dark navigation bar with 'Solutions', 'Products', and 'Support' links, and the XILINX logo. A breadcrumb trail indicates the path: Home / Developer Tools / ISE Design Suite / ISE WebPACK Design Software. The main heading is 'ISE WebPACK Design Software'. The text describes it as a free, front-to-back FPGA design solution for Linux, Windows XP, and Windows 7, offering HDL synthesis, simulation, implementation, device fitting, and JTAG programming. It mentions that ISE WebPACK delivers a complete design flow at no cost. A new feature is highlighted: ISE Design Suite 14 - WebPACK now supports embedded processing design for the Zynq®-7000 SoC for the Z-7010, Z-7020, and Z-7030. A 'Download ISE WebPACK Now!' section includes a link to download the software for Windows and Linux. On the right, there are 'Quick Links' (Free Evaluation, Support and Documentation, IP Center, Licensing Solutions, EDA Solutions Partners) and 'Key Documentation' (ISE Design Suite Product Brief, ISE Design Suite Product Table). A 'Feedback' button is visible on the right side of the page.

ISE WebPACK Design Software

ISE® WebPACK™ design software is the industry's only FREE, fully featured front-to-back FPGA design solution for Linux, Windows XP, and Windows 7. ISE WebPACK is the ideal downloadable solution for FPGA and CPLD design offering HDL synthesis and simulation, implementation, device fitting, and JTAG programming. ISE WebPACK delivers a complete, front-to-back design flow providing instant access to the ISE features and functionality at no cost. Xilinx has created a solution that allows convenient productivity by providing a design solution that is always up to date with error-free downloading and single file installation.

And new in ISE Design Suite 14 - WebPACK now supports embedded processing design for the Zynq®-7000 SoC for the Z-7010, Z-7020, and Z-7030.

Download ISE WebPACK Now!

- [Download ISE WebPACK software for Windows and Linux.](#)

Key Features

Quick Links

- [Free Evaluation](#)
- [Support and Documentation](#)
- [IP Center](#)
- [Licensing Solutions](#)
- [EDA Solutions Partners](#)

[More ▾](#)

Key Documentation

- [ISE Design Suite Product Brief](#)
- [ISE Design Suite Product Table](#)

Feedback

<https://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html>

Descrição em VHDL para Flip-flop D sensível ao nível

entity ff_D is

port(

D: in std_logic;

clk: in std_logic;

Q: out std_logic);

end ff_D;

architecture sensível_nível of ff_D is

begin

process (clk, D)

begin

if (clk = '1') then

Q <= D;

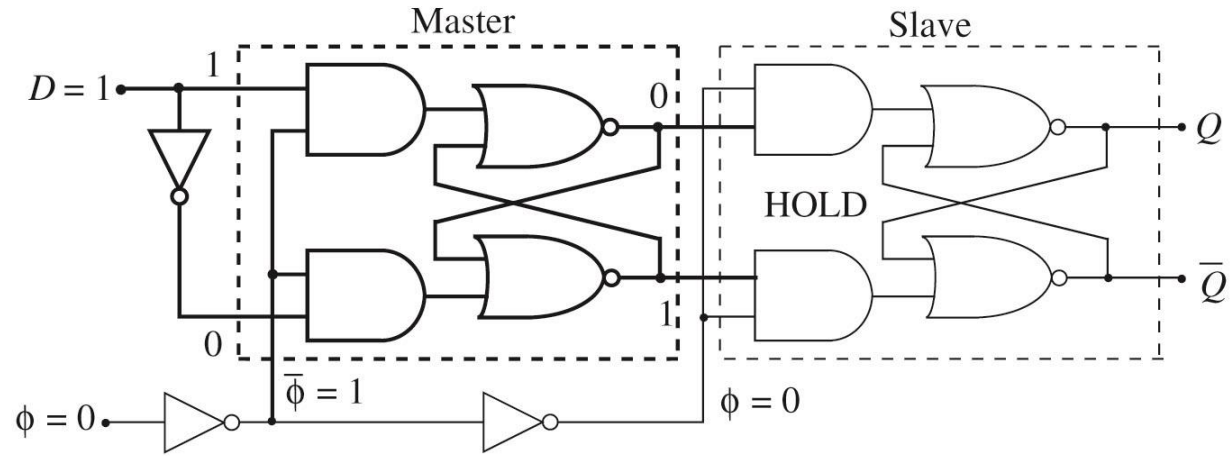
end if;

end process;

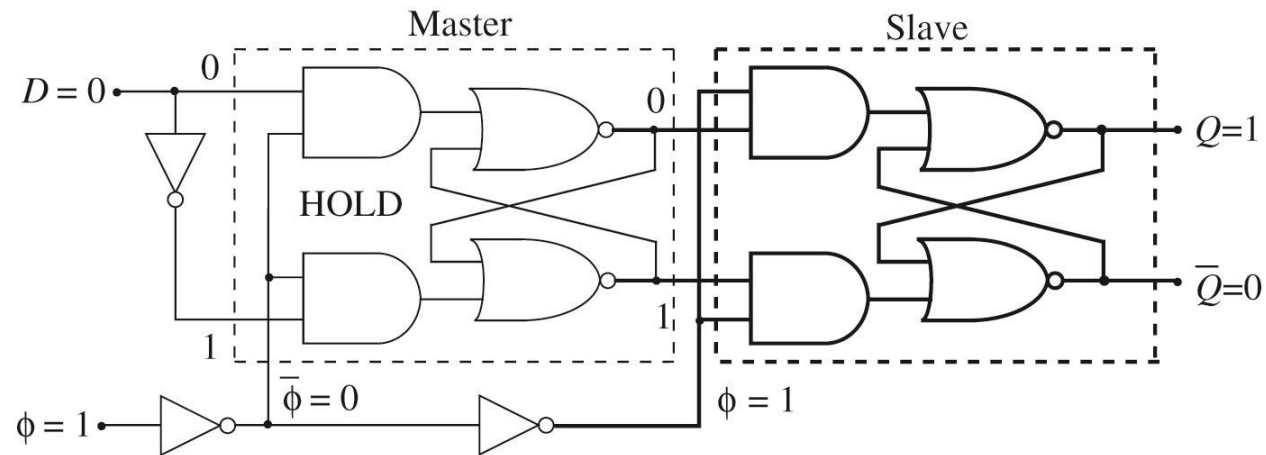
end sensível_nível;

Enable

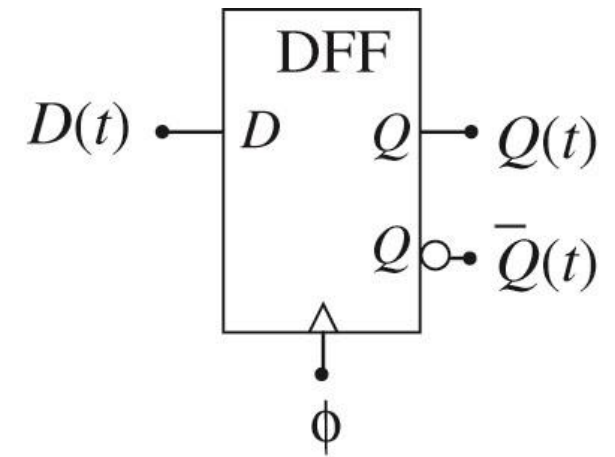
Flip-Flop Mestre-Escravo Tipo D



(a) Load master



(b) Transfer to slave



Descrição em VHDL para Flip-flop D sensível a borda

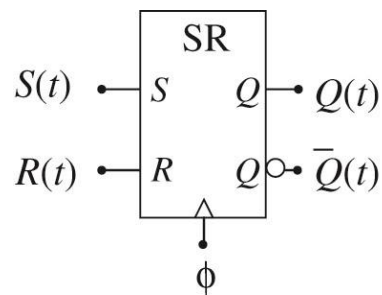
```
entity FFD is
  port(
    D: in std_logic;
    clk: in std_logic;
    Q: out std_logic);
end FFD;
```

```
architecture borda_positiva of FFD is
  begin
    process (clk, D)
    begin
      if (clk'event and clk = '1') then
        Q <= D;
      end if;
    end process;
  end borda_positiva;
```

- Como seria com borda de descida?

Descrição em VHDL para FF-SR

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity srff is  
    port (reset, set: in std_logic;  
          q: out std_logic);  
end srff;
```



(a) Symbol

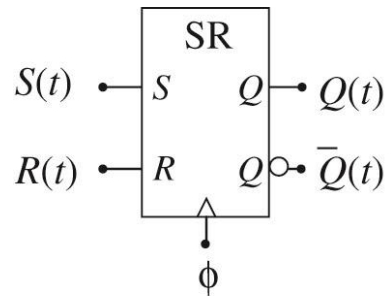
$S(t)$	$R(t)$	$Q(t+T)$	Operation
0	0	$Q(t)$	Hold
1	0	1	Set
0	1	0	Reset
1	1	?	Not used

(b) Operation summary

```
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;
```

Descrição em VHDL para FF-SR sensível ao clock

```
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;
```



(a) Symbol

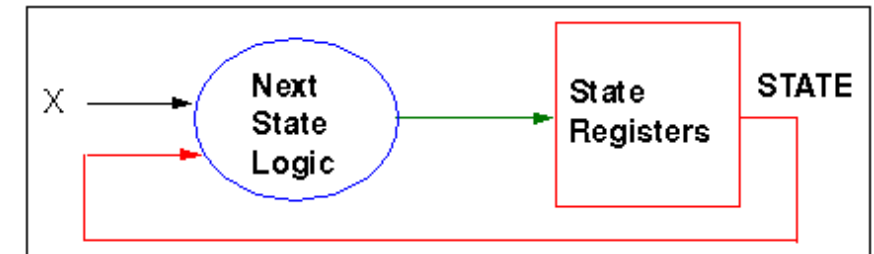
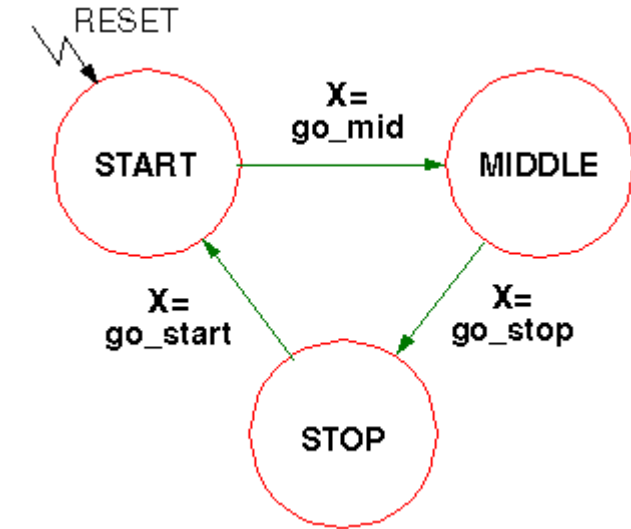
$S(t)$	$R(t)$	$Q(t+T)$	Operation
0	0	$Q(t)$	Hold
1	0	1	Set
0	1	0	Reset
1	1	?	Not used

(b) Operation summary

```
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;
```

Exemplo em VHDL para uma máquina de estados

```
FSM_FF: process (CLK, RESET)
begin
  if RESET='1' then
    STATE <= START ;
  elsif CLK'event and CLK='1' then
    case STATE is
      when START => if X=GO_MID then
        STATE <= MIDDLE ;
      end if ;
      when MIDDLE => if X=GO_STOP then
        STATE <= STOP ;
      end if ;
      when STOP => if X=GO_START then
        STATE <= START ;
      end if ;
      when others => STATE <= START ;
    end case ;
  end if ;
end process FSM_FF ;
```



<https://stackoverflow.com/questions/26618736/what-does-1-2-or-3-process-mean-for-an-fsm-in-vhdl>

Trabalho

- O segundo trabalho é uma descrição em VHDL de um pequeno banco de registradores a exemplo da figura abaixo:



- O objetivo do Trabalho 2 é desenvolver em VHDL um banco de 8 registradores de 8 bits para armazenar e ler uma palavra de 8 bits.
 - O projeto em VHDL do banco de registradores deve conter obrigatoriamente:
 - Código em VHDL de um Flip-Flop tipo D de 1 bit.
 - Código em VHDL de um registrador de 8 bits baseado no component do FF-D 1bit.
 - Código em VHDL de um banco de 8 registradores baseado no component do registrador de 8 bits.
 - Códigos em VHDL de um MUX 8x1 e um DEMUX 1x8, ambos para palavras de 8 bits.
 - Código em VHDL do banco de registradores completo com as seguintes entradas e saídas:
 - Entrada de CLK de 1 bit; entrada de R/W de 1 bit; entrada de endereço de 3 bits; entrada de dados de 8 bits.
 - Saída de dados de 8 bits.
 - Código em VHDL do testbench do banco de registradores.
- A entrega do trabalho deve ser feita via upload de um vídeo curto no Canvas com os seguintes itens:
 - Apresentação e explicação da arquitetura do banco de registradores.
 - Apresentação e explicação de cada código em VHDL.
 - Apresentação e explicação de testes via testbenchs.
 - Apresentação usando um simulador.

Demux com 8 saídas e palavra de 8 bits

entrada: in std_logic_vector (7 downto 0);

sel: in std_logic_vector(2 downto 0);

s1, s2, s3, s4, s5, s6, s7, s8: out std_logic_vector(7 downto 0)

Variação de clock no arquivo de testbench

-- clock (1000 ciclos de um total de 1000 ns, conforme período do clk abaixo):

```
CLK : process
begin
for i in 0 to 1000 loop
clk_sig <= '1' after 0.5 ns when clk_sig = '0' else
'0' after 0.5 ns when clk_sig = '1';
end loop;
end process;
```

Array para o banco de registradores

O banco de registradores possui 8 entradas de 8 bits cada, ou seja E1, E2.... E8: in std_logic_vector (7 downto 0).

Da mesma forma, há 8 saídas de 8 bits cada, ou seja, S1, S2.... S8: out std_logic_vector (7 downto 0).

Há um código de registrador de 8 bits que é um component no banco que precisa ser gerado 8 vezes, um registrador conectado a cada entrada e saída. E1 -> Registrador1 -> S1.

A melhor forma de fazer isso é via Array. Para isso, é necessário criar um tipo de dados dentro do architecture:

Ex: type t_Data is array (0 to 7) of std_logic_vector(7 downto 0);

Depois, você define um signal para conectar com entradas e outro com saídas do tipo t_Data, assim:

signal DadoEnt : t_Data := (E1, E2, E3, E8);

signal DadoSai : t_Data := (S1, S2, S3, S8);

O generate para o banco de 0 a 7 (8 registradores do component adicionado) deve fazer o port map assim:

EntradaComponent -> DadoEnt(i)

SaidaComponent -> DadoSai(i)