Circuitos Digitais

Linguagens de Descrição de Hardware

Prof. Marcelo Grandi Mandelli

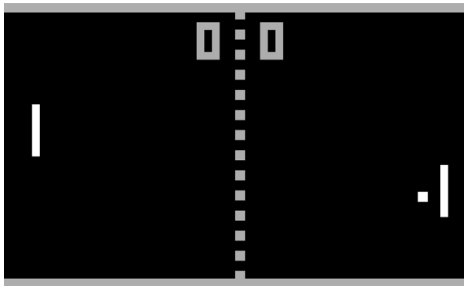
mgmandelli@unb.br

Introdução

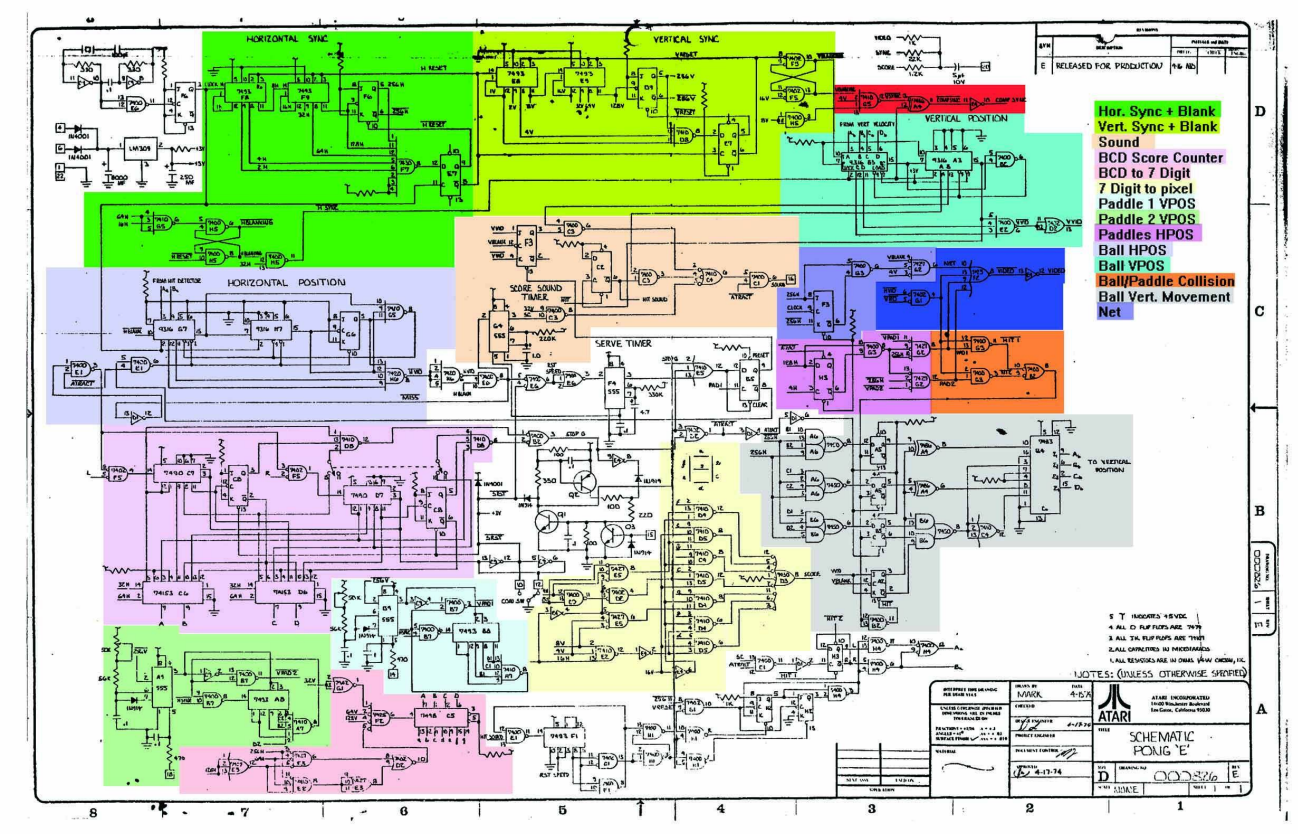
Sistemas Digitais complexos são praticamente impossíveis de se definir e estudar através do esquemático ao nível de portas lógicas

Introdução

- O processo de encontrar um conjunto eficiente de portas lógicas para realizar uma dada função é trabalhoso e propenso a erros:
  - simplificações manuais de tabelas verdade ou de equações Booleanas, etc.



Atari Pong



Esquemático do Atari Pong 1974

Introdução

- Na década de 90, os projetistas descobriram que eram muito mais produtivos se trabalhassem num **nível maior de abstração**, especificando apenas a função lógica e permitindo que uma **ferramenta de projeto auxiliado por computador** (CAD - computer-aided design) produzisse as portas otimizadas.

- Linguagens de descrição de hardware (HDL - Hardware Description Languages)

  - **Verilog, VHDL,** SystemC, Chisel, Handel-C, SDL, ISP, … (existem dezenas)

  - Apenas especificam funções lógicas

  - Ferramentas de CAD produzem ou sintetizam o circuito com as portas otimizadas

HDLs são linguagens de programação?

- Resposta curta: **Não**, é uma linguagem de descrição de hardware!

- Código é executado em um simulador
  - não há um "compilador" de VHDL, não há um "código executável" visível

HDLs – Vantagens e Desvantagens

- **Benefícios (em relação a diagramas de esquemáticos)**
  - Projetos independentes da tecnologia (implementação física é postergada)
  - Flexibilidade: re-utilização, escolha de ferramentas e fornecedores
  - Facilidade de atualização dos projetos
  - Permite explorar, em um nível mais alto de abstração, diferentes alternativas de implementação
  - Permite, através de simulação, verificar o comportamento do sistema digital

- **Desvantagens (em relação a diagramas de esquemáticos)**
  - Hardware gerado pode ser menos otimizado
  - Controlabilidade / Observabilidade de projeto reduzidas

SystemVerilog vs VHDL

HISTÓRICO

SystemVerilog

- Verilog foi desenvolvida pela Gateway Design Automation como uma linguagem proprietária em 1984
- A Gateway foi adquirida pela Cadence em 1989
- Verilog se tornou um padrão aberto em 1990, sob o controle da Open Verilog International
- A linguagem se tornou um padrão IEEE em 1995
- 2012 → SystemVerilog, orientação a objetos
- Os nomes de arquivos em SystemVerilog normalmente terminam em **.sv**
- Sintaxe semelhante a linguagem C

VHDL

- VHSIC Hardware Description Language
- VHSIC → acrônimo para o programa Very High Speed Integrated Circuits do Departamento de Defesa dos EUA.
- desenvolvida em 1981
- baseada na linguagem de programação Ada
- foi inicialmente imaginada para documentação, mas foi rapidamente adotada para simulação e síntese.
- O IEEE a padronizou em 1987 e atualizou o padrão muitas vezes desde então
- Nesses slides → revisão de 2008
- Os arquivos VHDL tem os seus nomes normalmente terminados em **.vhd**

SystemVerilog vs VHDL

- A SystemVerilog e VHDL são construídas sobre princípios similares, mas possuem sintaxes diferentes.

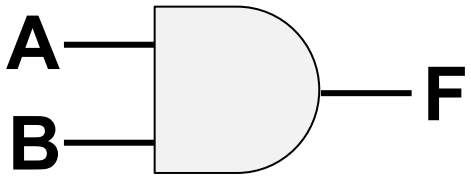
- Ambas as linguagens são totalmente capazes de descrever qualquer sistema de hardware, e ambas possuem as suas peculiaridades.

- A melhor linguagem para se usar é aquela em uso no seu local de trabalho ou aquela que os seus clientes exigem.

- A maioria das ferramentas CAD hoje em dia permitem que as duas linguagens sejam misturadas, de modo que diferentes módulos possam ser descritos em linguagens diferentes.

SystemVerilog vs VHDL

Exemplo – Porta AND



SystemVerilog

```
module portaE
(
        input logic A,B,
        output logic F
);

        assign F = A & B;
        /* Porta AND */

endmodule
```

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity portaE is
        port
        (
                A, B: in  STD_LOGIC;
                F: out STD_LOGIC
        );
end;

architecture pe of portaE is
begin
        F <= A and B;
end pe;
```

Módulos

- Um bloco de hardware com entradas e saídas é chamado de **módulo**.

- Uma porta AND, um multiplexador, e um somador são todos exemplos de módulos de hardware.

- Os dois modelos gerais para a descrição da funcionalidade do módulo são:

  - **modelos comportamentais** → descrevem o que o módulo faz (maior abstração)

  - **modelos estruturais** → descrevem como o módulo é construído a partir de partes simples; é uma aplicação de hierarquia.

SystemVerilog

Estrutura básica de um módulo

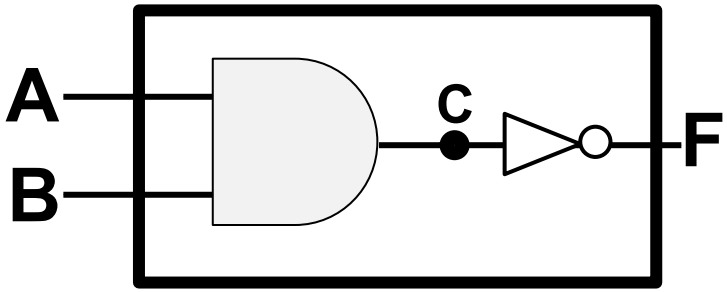
```
module exemplo
(
        input logic A,B,
        output logic F
);
        logic C;

        assign C = A & B;
        assign F = ~C;

endmodule
```

Definição do nome do módulo

Declaração de entradas e saídas

Declaração de sinais internos

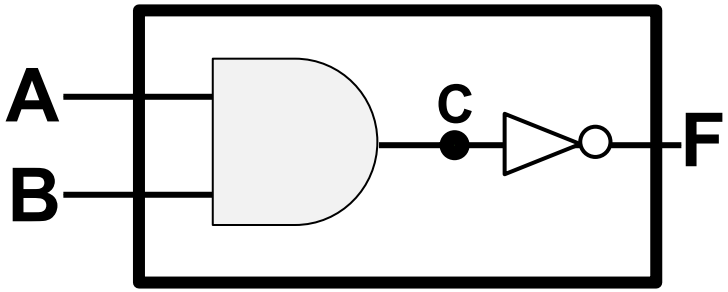
Definição do comportamento do modulo. Cada linha executa em paralelo! **Não interessa a ordem!**



Comportamental vs Estrutural

```
module exemplo
(
        input logic A,B,
        output logic F
);
        logic C;

        assign C = A & B;
        assign F = ~C;

endmodule
```

Esse é um modulo descrito em um modelo comportamental → nível mais alto de abstração



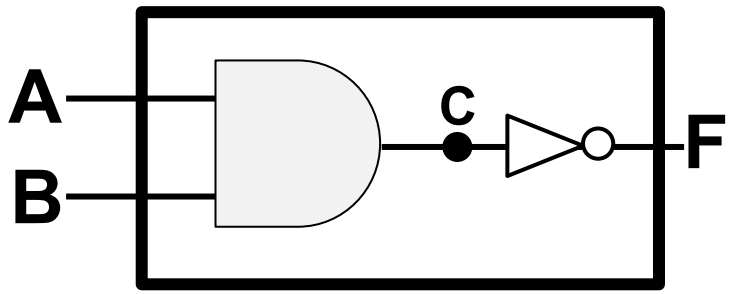
Comportamental vs Estrutural

```
module and2
(
        input logic A,B,
        output logic F
);
        assign F = A & B;

endmodule
```

```
module inv
(
        input logic A,
        output logic F
);
        assign F = ~A;

endmodule
```

```
module exemplo
(
        input logic A,B,
        output logic F
);
        logic C;

        and2 and2gate(A, B, C);
        inv  invgate(C, F);

endmodule
```

Esse é um modulo descrito em um modelo estrutural



Sintaxe

- Case sensitive
  - Exemplo: reset e Reset não são o mesmo sinal
- Nenhum nome começa por números
  - Exemplo: 2mux é um nome inválido
- Espaços são ignorados
- Comentários:
  - // → comentário de linha simples
  - /* comentário
    multilinha */

Operações

Alta prioridade

~	NOT
*, /, %	mult, div, mod
+, -	add,sub
<<, >>	shift
<<<, >>>	arithmetic shift
<, <=, >, >=	comparison
==, !=	equal, not equal
&, ~&	AND, NAND
^, ~^	XOR, XNOR
|, ~|	OR, NOR
?:	ternary operator

Baixa prioridade

Representação de Números

Formato: <tamanho>'<base><número>

<tamanho>'<base> é opcional, mas recomendado

Por padrão, a base é decimal

Número	Tamanho	Base	Equivalente em Decimal	Armazenado
3'b101	3	binary	5	101
'b11	unsized	binary	3	00…0011
8'b11	8	binary	3	00000011
8'b1010_1011	8	binary	171	10101011
3'd6	3	decimal	6	110
6'o42	6	octal	34	100010
8'hAB	8	hexadecimal	171	10101011
42	Unsized	decimal	42	00…0101010

Outros aspectos

- Valores:

  [0,1]: níveis lógicos
  x : desconhecido, z: alta impedância

  - Ex.: 32'bz 8'h0x 4'b1z0x

- Tipos: wire, logic, como input, output, inout ou sinal
  - Exemplos: input logic [31:0] A; output wire x;

  logic [3:0] vetor[0:15]

- Tipos abstratos: integer, real (IEEE 754), time

VHDL

Estrutura básica de um módulo

Libraries	Declaração e uso de *libraries* ocorre no início da descrição. ***Libraries* são conjuntos de *packages***
Entidade	Descreve a interface do módulo, seus sinais de entrada e saída
Arquitetura	Descreve a implementação do módulo

Estrutura básica de um módulo

```
library ieee;
use ieee.std_logic_1164.all;
entity exemplo is
port (
        A: in std_logic;
        B: in std_logic;
        F: out std_logic
   );
end exemplo;

architecture ex of exemplo is
signal C: std_logic;
begin
        C <= A and B;
        F <= not C;

end ex;
```

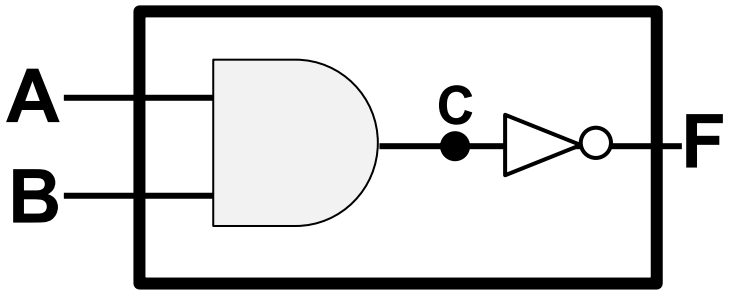
Definição e uso de bibliotecas

Definição do nome do módulo

Declaração de entradas e saídas

Declaração de sinais internos

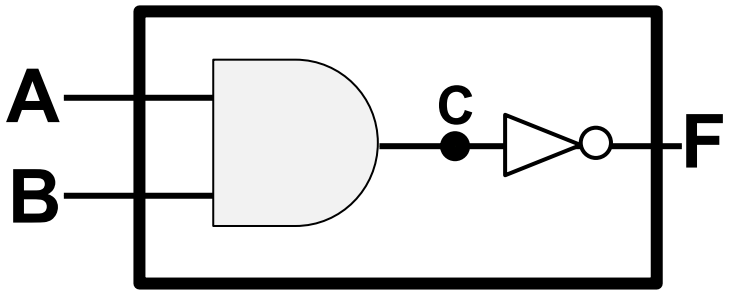
Definição do comportamento do modulo. Cada linha executa em paralelo! **Não interessa a ordem!**



Comportamental vs Estrutural

```
library ieee;
use ieee.std_logic_1164.all;
entity exemplo is
port (
        A: in std_logic;
        B: in std_logic;
        F: out std_logic
   );
end exemplo;

architecture ex of exemplo is

signal C: std_logic;

begin

        C <= A and B;
        F <= not C;

end ex;
```

Esse é um modulo descrito em um modelo comportamental → nível mais alto de abstração

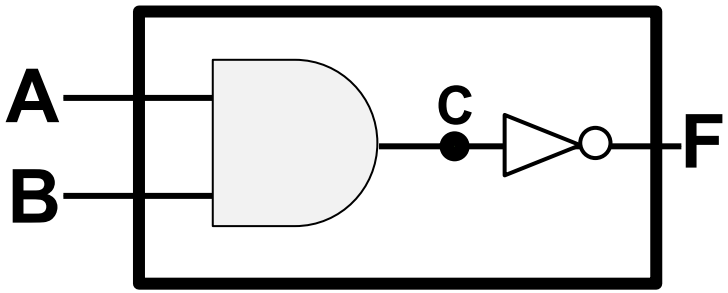


Comportamental vs Estrutural

```
library ieee;
use ieee.std_logic_1164.all;
entity exemplo is
port (
          A: in std_logic;
          B: in std_logic;
          F: out std_logic
   );
end exemplo;

architecture ex of exemplo is

signal C: std_logic;

begin

          and2gate : entity work.and2
                     port map(A, B, C);

          invgate : entity work.inv
                    port map(C, F);

end ex;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity and2 is
    port ( A, B : in std_logic;
             F : out std_logic);
end and2;

architecture basic of and2 is
begin
        F <= A and B;

end basic;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity inv is
    port ( A : in std_logic;
             F : out std_logic);
end inv;

architecture basic of inv is
begin
        F <= not A;

end basic;
```



Esse é um modulo descrito em um modelo estrutural

Outro Exemplo

library ieee; **use** ieee.std_logic_1164.all; **use** ieee.numeric_std.all;	Declaração e uso das libraries
entity signed_adder **is** **generic** (SIZE: **natural** := 8); -- opcional **port** (a: **in signed** ((SIZE-1) **downto** 0); b: **in signed** ((SIZE-1) **downto** 0); result: **out signed** ((SIZE-1) **downto** 0)); **end** entity;	Declaração da entidade signed_adder e suas portas de entrada e saída com largura parametrizada por SIZE

architecture rtl **of** signed_adder **is**
begin
result <= a + b;
end rtl;
	Arquitetura da entidade signed_adder

Operações

Tabela 4.2 Precedência de operadores em VHDL

	Op	Meaning
Highest	`not`	NOT
	`*`, `/`, `mod`, `rem`	MUL, DIV, MOD, REM
	`+`, `-`	PLUS, MINUS
	`rol`, `ror`, `srl`, `sll`	Rotate, Shift logical
	`<`, `<=`, `>`, `>=`	Relative Comparison
	`=`, `/=`	Equality Comparison
Lowest	`and`, `or`, `nand`, `nor`, `xor`, `xnor`	Logical Operations

Números

Tabela 4.4 Números em VHDL

Numbers	Bits	Base	Val	Stored
3B"101"	3	2	5	101
B"11"	2	2	3	11
8B"11"	8	2	3	00000011
8B"1010_1011"	8	2	171	10101011
3D"6"	3	10	6	110
60"42"	6	8	34	100010
8X"AB"	8	16	171	10101011
"101"	3	2	5	101
B"101"	3	2	5	101
X"AB"	8	16	171	10101011

Sintaxe

- A linguagem NÃO é *case sensitive*
  - mas frequentemente são usadas maiúsculas para as palavras reservadas

- Comentários
  - -- → comentário de linha simples
  - /* comentário
    multilinha */

- VHDL é uma linguagem fortemente "tipada"
  (integer 1 ≠ real 1.0 ≠ bit '1')
  - auxilia para detectar erros no início do projeto
  - exemplo: conectar um barramento de 4 bits a um barramento de 8 bits

VHDL Packages

- std_logic_1164
  - define tipos de dados e operações: std_logic, std_logic_vector, std_ulogic...
- std_logic_signed
  - processamento de std_logic como inteiros
- std_logic_unsigned
  - processamento de std_logic como naturais
- std_logic_arith
  - define tipos unsigned e signed e funções aritméticas
- numeric_std
  - define funções aritméticas para vetores lógicos. Usado como alternativa ao std_logic_arith

Recomenda-se usar o std_logic_1164 e o numeric_std

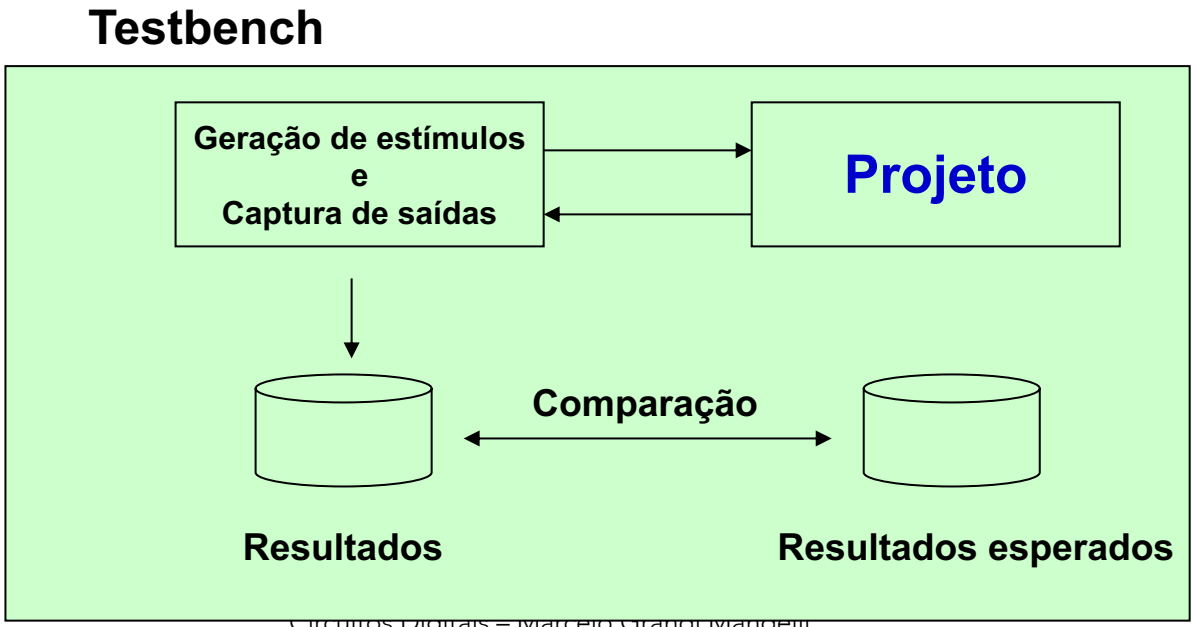
Tipos

most common

Type name	Sim. only	Note
`std_logic, std_logic_vector`		Actually enumeration, you'll need pkg `ieee_1164`, use these instead of bit/bit_vector, use `downto` indexing
`integer`		Limit range for synthesis
`unsigned, signed`		Similar to std_logic_vector, but safer for artihmetic
`array`		E.g. std_logic_vector is array. Define the array type first and then signal/constnat/variable of that type
`enumeration`		bit and std_logic are actually enumerations, use this at least for states of an FSM
`record`		Synthesizable, but not very common
`file`	x	For reading input data and storing trace/log during simulation-based verification
`physical`	x	For detailed gate-level simulation with timing
`real`	x	Quite rare because cannot be (always) synthesized
`access`	x	Very rare

Testbench

- Testbench: descrição HDL para teste do projeto em desenvolvimento
  - especificação comportamental do ambiente externo ao projeto
  - interage com o projeto
  - não precisa ser descrito em HDL (o projeto pode ser validado em ambiente C/C++!!)



Exemplo Testbench em VHDL

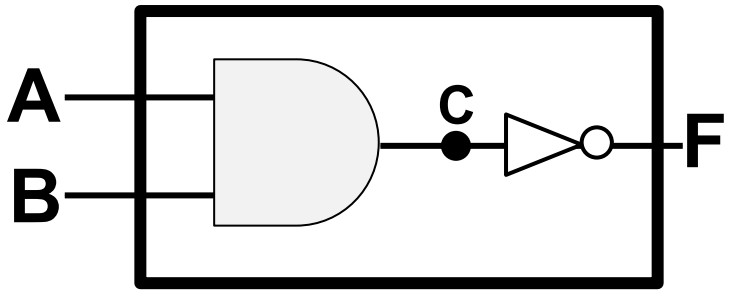
```
library ieee;
use ieee.std_logic_1164.all;

entity exemplo_tb is
end exemplo_tb;

architecture TB_ARCHITECTURE of exemplo_tb is

   signal aa, bb, ff : std_logic;

begin

        UUT : entity work.exemplo
              port map ( A => aa, B => bb, F => ff);

        aa <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 30 ns;

        bb <= '0', '1' after 20 ns;

end TB_ARCHITECTURE;
```

Test bench não tem pinos externos

Seleção da arquitetura

Instanciação do projeto

Geração dos estímulos



Exemplos

SystemVerilog

Multiplexador 4:1

```
module mux4(input logic [3:0] d0, d1, d2, d3,
            input logic [1:0] s,
            output logic [3:0] y);

  assign y = s[1] ? (s[0] ? d3 : d2)
                  : (s[0] ? d1 : d0);
endmodule
```

VHDL

Multiplexador 4:1

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux4 is
  port(d0, d1,
       d2, d3: in STD_LOGIC_VECTOR(3 downto 0);
       s: in STD_LOGIC_VECTOR(1 downto 0);
       y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth1 of mux4 is
begin
  y <= d0 when s = "00" else
       d1 when s = "01" else
       d2 when s = "10" else
       d3;
end;
```

SystemVerilog

Registrador de 4 bits

```
module flop(input logic clk,
            input logic [3:0] d,
            output logic [3:0] q);

 always_ff @(posedge clk)
   q <= d;
endmodule
```

Em geral , uma declaração *always* em SystemVerilog é escrita da seguinte foma:

```
always @(lista de sensibilidade)
  declaração;
```

VHDL

Registrador de 4 bits

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flop is
  port(clk: in STD_LOGIC;
       d: in STD_LOGIC_VECTOR(3 downto 0);
       q: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of flop is
begin
  process(clk) begin
    if rising_edge(clk) then
      q <= d;
    end if;
  end process;
end;
```

```
process(clk) begin
  if clk'event and clk = '1' then
     q <= d;
  end if;
end process;
```

Outra forma

SystemVerilog

Registrador de 4 bits com RESET síncrono

```
module flopr(input logic clk,
             input logic reset,
             input logic [3:0] d,
             output logic [3:0] q);

// synchronous reset
always_ff @(posedge clk)
  if (reset) q <= 4'b0;
  else q <= d;
endmodule
```

VHDL

Registrador de 4 bits com RESET síncrono

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flopr is
  port(clk, reset: in STD_LOGIC;
       d: in STD_LOGIC_VECTOR(3 downto 0);
       q: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synchronous of flopr is
begin
  process(clk) begin
    if rising_edge(clk) then
      if reset then q <= "0000";
      else q <= d;
      end if;
    end if;
  end process;
end;
```

SystemVerilog

Decodificador de Display de 7 segmentos

```
module sevenseg(input logic [3:0] data,
                output logic [6:0] segments);

  always_comb
    case(data)
      // abc_defg
      0: segments = 7'b111_1110;
      1: segments = 7'b011_0000;
      2: segments = 7'b110_1101;
      3: segments = 7'b111_1001;
      4: segments = 7'b011_0011;
      5: segments = 7'b101_1011;
      6: segments = 7'b101_1111;
      7: segments = 7'b111_0000;
      8: segments = 7'b111_1111;
      9: segments = 7'b111_0011;
      default: segments = 7'b000_0000;
    endcase
endmodule
```

VHDL

Decodificador de Display de 7 segmentos

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity seven_seg_decoder is
  port(data: in STD_LOGIC_VECTOR(3 downto 0);
    segments: out STD_LOGIC_VECTOR(6 downto
0));
end;

architecture synth of seven_seg_decoder is
begin
  process(all) begin
    case data is
      -- abcdefg
      when X"0" => segments <= "1111110";
      when X"1" => segments <= "0110000";
      when X"2" => segments <= "1101101";
      when X"3" => segments <= "1111001";
      when X"4" => segments <= "0110011";
      when X"5" => segments <= "1011011";
      when X"6" => segments <= "1011111";
      when X"7" => segments <= "1110000";
      when X"8" => segments <= "1111111";
      when X"9" => segments <= "1110011";
      when others => segments <= "0000000";
    end case;
  end process;
end;
```

SystemVerilog

Decodificador 3:8

```
module decoder3_8(input  logic [2:0] a,
                  output logic [7:0] y);
  always_comb
    case(a)
      3'b000:  y=8'b00000001;
      3'b001:  y=8'b00000010;
      3'b010:  y=8'b00000100;
      3'b011:  y=8'b00001000;
      3'b100:  y=8'b00010000;
      3'b101:  y=8'b00100000;
      3'b110:  y=8'b01000000;
      3'b111:  y=8'b10000000;
      default: y=8'bxxxxxxxx;
    endcase
endmodule
```

VHDL

Decodificador 3:8

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity decoder3_8 is
  port(a: in STD_LOGIC_VECTOR(2 downto 0);
       y: out STD_LOGIC_VECTOR(7 downto 0));
end;
```

```
architecture synth of decoder3_8 is
begin
  process(all) begin
    case a is
      when "000" => y <= "00000001";
      when "001" => y <= "00000010";
      when "010" => y <= "00000100";
      when "011" => y <= "00001000";
      when "100" => y <= "00010000";
      when "101" => y <= "00100000";
      when "110" => y <= "01000000";
      when "111" => y <= "10000000";
      when others => y <= "XXXXXXXX";
    end case;
  end process;
end;
```

SystemVerilog

Testbench

```
module testbench1();
  logic a, b, c, y;

  // instantiate device under test
  sillyfunction dut(a, b, c, y);

  // apply inputs one at a time
  initial begin
    a = 0; b = 0; c = 0; #10;
    c = 1; #10;
    b = 1; c = 0; #10;
    c = 1; #10;
    a = 1; b = 0; c = 0; #10;
    c = 1; #10;
    b = 1; c = 0; #10;
    c = 1; #10;
  end
endmodule
```

VHDL

Testbench

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity testbench1 is -- no inputs or outputs
end;

architecture sim of testbench1 is
  component sillyfunction
    port(a, b, c: in STD_LOGIC;
         y: out STD_LOGIC);
  end component;
  signal a, b, c, y: STD_LOGIC;
begin
  -- instantiate device under test
  dut: sillyfunction port map(a, b, c, y);
  -- apply inputs one at a time
  process begin
    a <= '0'; b <= '0'; c <= '0'; wait for 10
ns;
    c <= '1'; wait for 10 ns;
    b <= '1'; c <= '0'; wait for 10 ns;
    c <= '1'; wait for 10 ns;
    a <= '1'; b <= '0'; c <= '0'; wait for 10
ns;
    c <= '1'; wait for 10 ns;
    b <= '1'; c <= '0'; wait for 10 ns;
    c <= '1'; wait for 10 ns;
    wait; -- wait forever
  end process;
end;
```

SystemVerilog

$$Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C.$$

Testbench com mensagens de debug

```
module testbench2();
  logic a, b, c, y;
  // instantiate device under test
  sillyfunction dut(a, b, c, y);
  // apply inputs one at a time
  // checking results
  initial begin
   a = 0; b = 0; c = 0; #10;
  assert (y === 1) else $error("000 failed.");
  c = 1; #10;
  assert (y === 0) else $error("001 failed.");
  b = 1; c = 0; #10;
  assert (y === 0) else $error("010 failed.");
  c = 1; #10;
  assert (y === 0) else $error("011 failed.");
  a = 1; b = 0; c = 0; #10;
  assert (y === 1) else $error("100 failed.");
  c = 1; #10;
  assert (y === 1) else $error("101 failed.");
  b = 1; c = 0; #10;
  assert (y === 0) else $error("110 failed.");
  c = 1; #10;
  assert (y === 0) else $error("111 failed.");
 end
endmodule
```

VHDL

$$Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C.$$

Testbench com mensagens de debug

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity testbench2 is -- no inputs or outputs
end;

architecture sim of testbench2 is
 component sillyfunction
  port(a, b, c: in STD_LOGIC;
   y: out STD_LOGIC);
  end component;
 signal a, b, c, y: STD_LOGIC;
begin
 -- instantiate device under test
 dut: sillyfunction port map(a, b, c, y);
 -- apply inputs one at a time
 -- checking results
 process begin
  a <= '0'; b <= '0'; c <= '0'; wait for 10 ns;
   assert y = '1' report "000 failed.";
  c <= '1'; wait for 10 ns;
   assert y = '0' report "001 failed.";
  b <= '1'; c <= '0'; wait for 10 ns;
   assert y = '0' report "010 failed.";
  c <= '1'; wait for 10 ns;
   assert y = '0' report "011 failed.";
  a <= '1'; b <= '0'; c <= '0'; wait for 10 ns;
   assert y = '1' report "100 failed.";
  c <= '1'; wait for 10 ns;
   assert y = '1' report "101 failed.";
  b <= '1'; c <= '0'; wait for 10 ns;
   assert y = '0' report "110 failed.";
  c <= '1'; wait for 10 ns;
   assert y = '0' report "111 failed.";
  wait; -- wait forever
 end process;
end;
```