VHDL

Sistemas Digitais II

PCS3225-2017S2

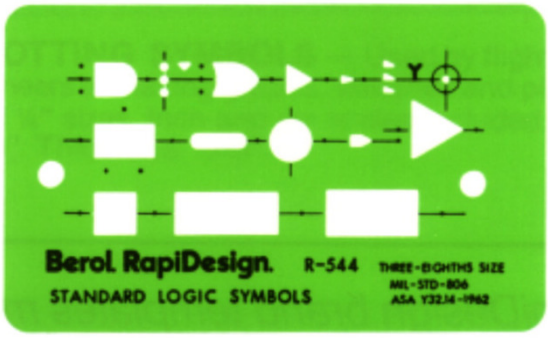
Antônio Mauro Saraiva
Bruno de Carvalho Albertini
Marco Túlio de Carvalho Andrade

1

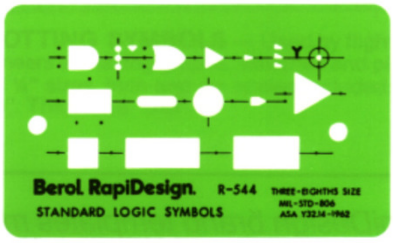
Introdução

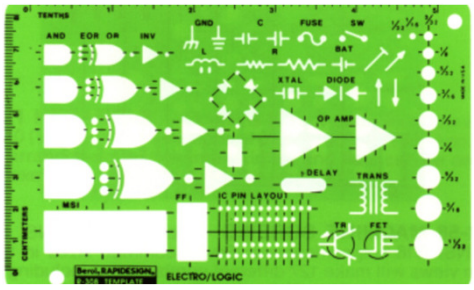
Descrição de Circuitos Digitais

- Há + de 30 anos:
  - Lápis, papel, gabarito
- Anos 80:
  - Editor de esquemáticos



Descrição de Circuitos Digitais





Problemas

- Como garantir que não cometemos nenhum erro no desenho de um circuito?
  - Linhas que se cruzam tem conexões ou não?
- Como expressar o comportamento de um circuito?

Soluções

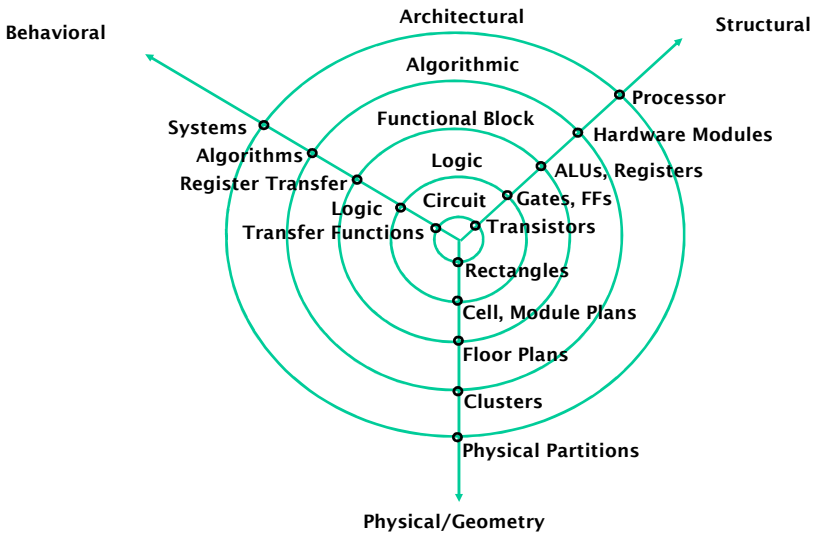
- Expressando o projeto em uma linguagem de descrição simulável e verificável
- Permitindo que a linguagem seja de alto nível de abstração

VHDL Verilog SystemC

Como estas linguagens são usadas?

- Para a descrição de projetos
- Para modelagem funcional e detalhada de "timing"
- Para a documentação de projetos
- Como uma interface padronizada para ferramentas de projeto (M2M)
- Para a especificação de projetos em qualquer nível de abstração

Abstração



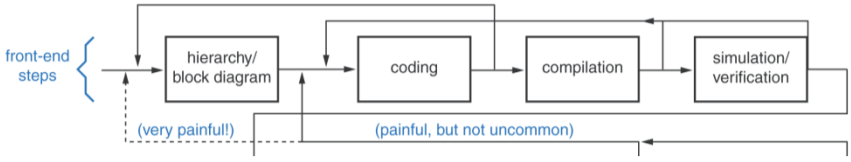
Abstração

- **Especificação funcional**: descrição mais abstrata possível
- **Algoritmo**: descrição do algoritmo lógico executado pelo sistema
- **Transferências entre Registradores** (RTL): descrição do algoritmo, porém em etapas que envolvam transferência de dados entre registradores
- **Chaveamento**: descrição do algoritmo como uma máquina de estados
- **Circuito lógico**: descrição do algoritmo através de elementos lógicos mapeáveis em componentes físicos existentes numa dada tecnologia

HDL *(Hardware Description Language)*

- Linguagem para:
  - Modelar circuitos eletrônicos
    - Reaproveitar modelos
  - Descrever circuitos
  - Testar circuitos
  - Descrever lista de ligações de um circuito
- Síntese de circuitos digitais

Ciclo de Desenvolvimento HDL



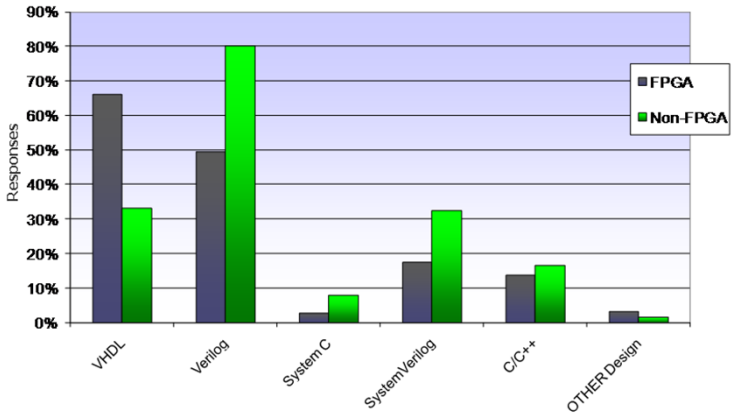
VHSIC *Hardware Description Language*

- [80s] DoD (Departamento de Defesa, EUA)
  - Documentar comportamento
  - Burocracia
  - IBM + Texas Instruments
- IEEE1076
  - [1987] Linguagem
    - Baseado em ADA (DoD)
    - IEEE1164 (Tipos)
  - [1993] Mais usada
  - [2000] Tipos protegidos
  - [2002] Buffer simplificado
  - [2008] Nomes externos

Padronização

- Modelagem
  - IDEs, desenho do diagrama, máquinas de estados, memórias, código, etc.
  - Descrição comportamental
- Verificação
  - Simulação e verificação funcional e temporizada
- Síntese
  - Fabricação do circuito, prototipação

Prototipação



Wilson Research Group & Mentor Graphics (2010)

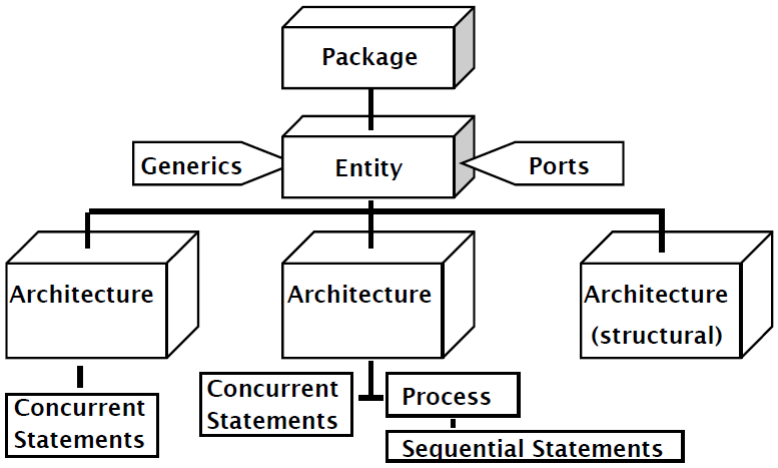
2

Modelagem

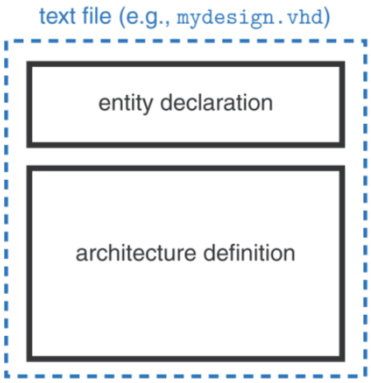
Componentes

- Bibliotecas de funções/procedimentos
  - *packages*
- Entidades
  - *entities*
- Arquiteturas
  - *architectures*
- Configurações
  - *configurations*

Componentes



Componentes



Exemplo

```
entity Inhibit is     -- also known as 'BUT-NOT'
  port (X,Y: in BIT;     --  as in 'X but not Y'
        Z:   out BIT);   --  (see [Klir, 1972])
```

Entidade

entity entity-name is
port (signal-names: mode signal-type;
signal-names: mode signal-type;
....
signal-names: mode signal-type);
end entity-name;

Arquitetura

- Associada a uma entidade pelo nome
  - Define o comportamento desta
- Pode definir a estrutura interna da entidade
- Uma entidade pode ter várias arquiteturas associadas a ela e tem ao menos uma
  - Comportamental
  - Estrutural
  - Mista

Arquitetura

```
architecture architecture-name of entity-name is
        type declarations
        signal declarations
        constant declarations
        function definitions
        procedure definitions
        component declarations
begin
        concurrent-statement
        concurrent-statement
end architecture-name;
```

Abstração em VHDL

- Comportamental
  - Síntese deixada para o compilador
- Data-flow
  - Controle de síntese médio
  - Modela o fluxo de dados
- Estrutural
  - Portas e suas conexões

Modelos de Execução em VHDL

- Concorrente
  - Ordem não importa
  - Cada *statement* é assíncrono
- Sequencial
  - Ordem importa
  - Pode gerar uma estrutura síncrona
  - Parecido com programação estruturada

Exemplo XOR dataflow-concorrente

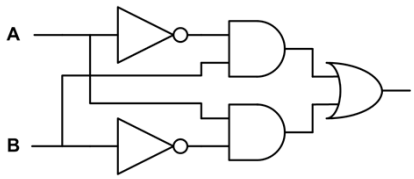
```
q <= a xor b;
```

Exemplo XOR comportamental-concorrente

```
q <= '1' when a/=b else '0';
```

Exemplo XOR estrutural

```
U1: inverter port map (a, ai);
U2: inverter port map (b, bi);
U3: and_gate port map (ai, b, t3);
U4: and_gate port map (bi, a, t4);
U5: or_gate port map (t3, t4, q);
```



A xor B = A'B + AB'

Exemplo XOR comportamental-sequencial

```
process(a,b)
begin
   if (a/=b) then
      q <= '1';
   else
      q <= '0';
   endif;
end process;
```

3

Linguagem

Tipos de Dados

- VHDL é fortemente tipada
  - Verificados na compilação
  - Conjunto definido de valores e operadores
  - Pode-se derivar tipos
- Tipos pré-definidos
  - bit
  - bit_vector
  - boolean
  - character
  - integer
  - real
  - severity_level
  - string
  - time

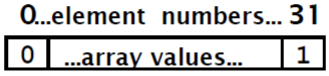
Tipos Enumerados

- Muitos tipos comuns em VHDL são tipos enumerados:
  - boolean (TRUE, FALSE);
  - bit ('1', '0')
  - char (...,'A', 'B', 'C', ...)
- Extensões de usuários:
  - type estados is (st0,st1,st2,st3,st4,st5,st6,st7,st8, st9,st10,st11,st12,st13,st14,st15);

Array

- Um ou mais elementos de tipos compatíveis em uma única construção

```
TYPE data_bus IS ARRAY (0 TO 31) OF BIT;
```




```
VARIABLE X:  data_bus;
VARIABLE Y:  BIT

Y := X(12);  -- Y gets value of 12th element
```

```
TYPE register IS ARRAY (15 DOWNTO 0) OF BIT;
```

IEEE 1164

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

std_logic
std_logic_vector
```

'U' - *Uninitialized*
'X' - *Forcing Unknown*
'0' - *Forcing* 0
'1' - *Forcing* 1
'Z' - *High Impedance*
'W' - *Weak Unknown*
'L' - *Weak* 0
'H' - *Weak* '1'
'-' - *Don't care*

Operadores

Aritméticos	
adição	+
subtração	-
Multiplicação	*
Divisão	/
Módulo	mod
Resto da divisão	rem
Valor absoluto	abs
Exponenciação	**

Booleanos	
AND	and
OR	or
NAND	nand
NOR	nor
OU exclusivo	xor
NOU exclusivo	xnor
complementação	not

Objetos que contém dados

- Sinais (*signal*)
  - Como fios em um diagrama esquemático
  - Podem ter valores atuais e futuros (!)
- Variáveis e Constantes
  - Usadas somente para modelagem
  - Não gera HW diretamente
  - Usadas em processos, procedimentos e funções

Constantes

- Contém um valor de um determinado tipo
- Valor não pode ser mudado em simulação
- Dentro de arquitetura: globais
- Dentro de processos: locais

CONSTANT *constant_name* : *type_name* [:= value];
CONSTANT rise_fall_time : TIME := 2 ns;
CONSTANT data_bus : INTEGER := 16;

Variáveis

- Usadas para armazenamento local
- Todas atribuições a variáveis tem efeito imediato
- Mais convenientes que sinais para armazenamento temporário

```
VARIABLE variable_name : type_name [:=value];

VARIABLE opcode : BIT_VECTOR(3 DOWNTO 0) := "0000";
VARIABLE freq : INTEGER;
```

Sinais

- Usados para comunicação entre componentes
- Declarados fora dos processos
- Único objeto de dados com mapeamento físico
  - Sempre tem atraso!

```
SIGNAL signal_name : type_name [:=value];

SIGNAL brdy : BIT;
SIGNAL output : INTEGER := 2;
```

Atribuição (sinais)

```
ARCHITECTURE signals OF test IS
   SIGNAL a, b, c, out_1, out_2: BIT;
BEGIN
   out_1 <= a NAND b;
   out_2 <= out_1 XOR c;
END signals;
```

Time	a	b	c	out_1	out_2
0	0	1	1	1	0
1	1	1	1	1	0
1+d	1	1	1	0	0
1+2d	1	1	1	0	1

Atribuição (variáveis)

```
ARCHITECTURE variables OF test IS
BEGIN
   PROCESS (a, b, c)
   VARIABLE a,b,c,out_3,out_4: BIT;
   BEGIN
     out_3 := a NAND b;
     out_4 := out_3 XOR c;
   END PROCESS;
END example;
```

Time	a	b	c	out_3	out_4
0	0	1	1	1	0
1	1	1	1	0	1

4

Modelagem Sequencial

process

```
architecture arch of ent is
begin
nome_processo: process (clock)
   begin
      comando sequencial;
      comando sequencial;
      wait until (condição);
      comando sequencial;
      . . .
      wait for (time);
      . . .
   end process;
end arch;
```

Um processo deve ter uma lista de sensibilidade OU ...

... OU, pelo menos, deve ter um comando *wait*.

WAIT

- Causa a suspensão do processo

wait [sensitivity_clause][condition_clause][timeout_clause];

- Sensitivity_clause ::= on signal_name
  - wait on CLOCK;
- Condition_clause ::= until boolean_expression
  - wait until Clock = '1';
- Timeout_clause ::= for time_expression
  - wait for 150 ns;

Lista de Sensibilidade ou Wait

```
Summation:
  PROCESS( A, B, Cin)
  BEGIN
    Sum <= A xor B xor Cin;
END PROCESS Summation;
```

=

```
Summation:  PROCESS
  BEGIN
    Sum <= A xor B xor Cin;
      WAIT ON A, B, Cin;
END PROCESS Summation;
```

Equivalência

- Cada *statement* concorrente equivale a um processo

```
q <= a xor b after 5 ns;

process
begin
 q <= a xor b after 5 ns;
 wait on a, b;
end process;
```

IF e CASE

```
if (a='1') then
      q <= '1';
elsif (b='1') then
      q <= '1';
else
      q <='0';
end if;
```

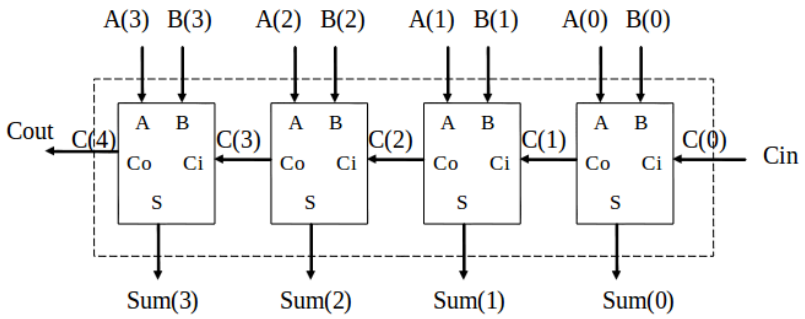
```
case (a&b) is
      when "00" =>
            q <= '0';
      when others =>
            q <= '1';
end case;
```

FOR e WHILE

```
for i in 0 to 9 loop
      q(i) <= a(i) and b(i);
end loop;


i:=0;
while (i<9) loop
      q <= a(i) and b(i);
      WAIT ON clk UNTIL clk='1';
end loop;
```

Exemplo: somador de 4 bits



Exemplo: somador de 4 bits

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

entity adder4bit is
   port ( a,b: in std_logic_vector(3 downto 0);
            cin : in std_logic;
            cout: out std_logic;
            sum: out std_logic_vector(3 downto 0)
  );
end adder4bit;
```

Exemplo: somador de 4 bits

```
architecture bruteforce of adder4bit  is
   signal  c : std_logic_vector(4 downto 0); .
begin
    process (a, b, cin, c)
    begin
      c(0) <= cin;
      -- full adder 0
      sum(0) <= a(0) xor b(0) xor c(0);
      c(1)      <= (a(0) and b(0)) or (c(0) and (a(0) or b(0)));
```

Exemplo: somador de 4 bits

```
  -- full adder 1
     sum(1) <= a(1) xor b(1) xor c(1);
     c(2)      <= (a(1) and b(1)) or (c(1) and (a(1) or b(1)));
     -- full adder 2
     sum(2) <= a(2) xor b(2) xor c(2);
     c(3)      <= (a(2) and b(2)) or (c(2) and (a(2) or b(2)));
     -- full adder 3
     sum(3) <= a(3) xor b(3) xor c(3);
     c(4)      <= (a(3) and b(3)) or (c(3) and (a(3) or b(3)));
     cout      <= c(4);
  end process;
end bruteforce;
```

Só processos?

- Esquecer de um sinal da lista de sensibilidade
- Fazer algo assíncrono sem querer
  - Boa prática: procurar usar processos COM clock
- Processo reentrante ou em cascata
- Múltiplos *drivers*
- Processos longos aumentam o caminho crítico

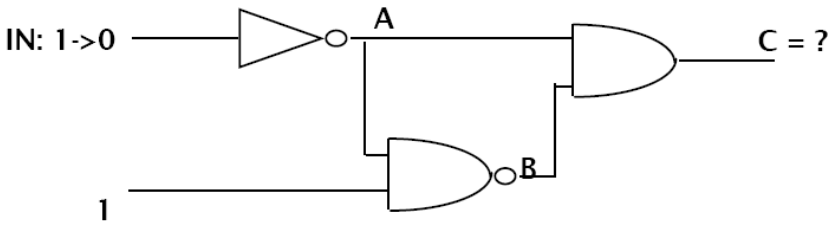
Regra prática: só use processos em projetos ou subprojetos onde você claramente enxerga uma máquina de estados ou flip-flop

- Registradores / Memórias
- Contadores

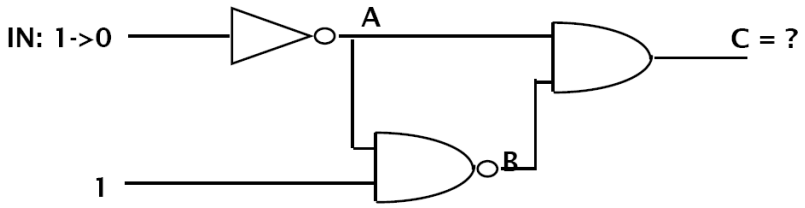
5

Simulação e verificação

Simulação



Simulação



NAND gate evaluated first:
IN: 1->0
A: 0->1
B: 1->0
C: 0->0

AND gate evaluated first:
IN: 1->0
A: 0->1
C: 0->1
B: 1->0
C: 1->0

Tufts VHDL Course

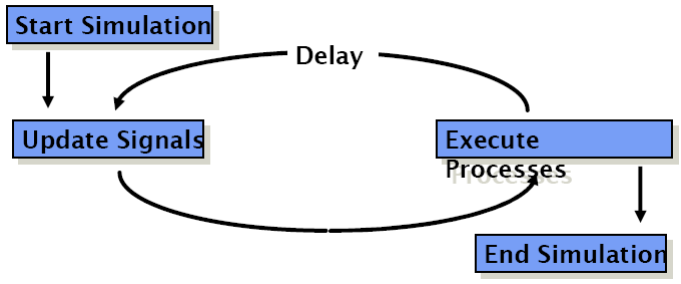
Tempo em VHDL

- É uma variável
  - Contínua
  - Assíncrona
  - Varia em intervalos discretos.
- As atribuições podem especificar um instante onde elas devem acontecer
  - ready <= '1' after 10 ns;

Ciclo de Simulação

- O tempo de simulação avança até o tempo do próximo evento na lista de eventos
- Todas as transações escalonadas para este tempo são executadas
  - Pode incluir novos eventos para serem executados neste instante ou em instantes posteriores
  - Atribuições a sinais são definidas e escalonadas para o instante atual ou instantes posteriores
- Quando todos os processos que foram reativados forem suspensos, o ciclo de simulação se encerra e podemos iniciar um novo ciclo
- Quando se atinge um estágio onde não existe nenhuma transação escalonada a simulação termina

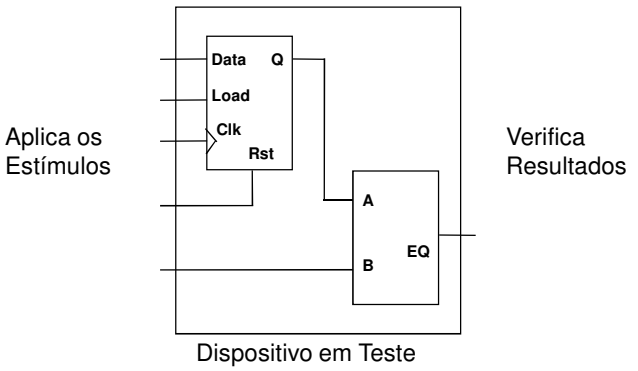
Ciclo de Simulação



Testbenchs

- Semelhante à bancada de laboratório
- Descrita usando comandos de comportamento (sequencial) do VHDL
- Use uma descrição de estrutura para incluir o componente em teste
- Em geral, é utilizada para gerar testes automáticos

Testbenchs



Exemplo

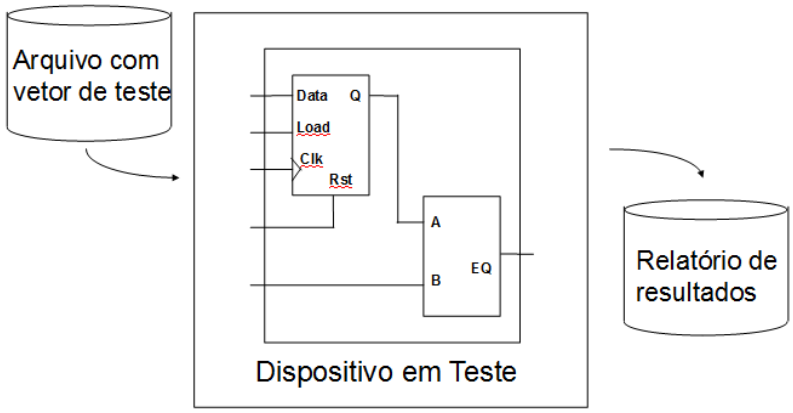
```
entity testbnch is
end testbnch;

architecture behavior of  testbnch is
   component rotcomp is
      port(Clk, Rst, Load: std_ulogic;
           Init:   std_ulogic_vector(0 to 7);
           Test: std_ulogic_vector(0 to 7);
           Limit: out std_ulogic;
   end component;
   signal s_Clk, s_Rst, s_Load: std_ulogic;
   signal s_Init:   std_ulogic_vector(0 to 7);
   signal s_Test: std_ulogic_vector(0 to 7);
   signal s_Limit: std_ulogic;
```

Exemplo (cont)

```
begin
   DUT: rotcomp port map -- Mapeamento
      (s_Clk, s_Rst, s_Load, s_Init, s_Test, s_Limit);
clock: process
   begin
      s_Clk <= '0';
      s_Rst <= '0';
      s_Load <= '1';
      s_Init <= "00001111";
      s_Test <= "11110000";
      wait for 10 ns;
      s_Clk <= '1';
      wait for 10 ns;
      s_Load <= '1';
      . . .
   end process;
```

Vetores de Teste



Vetores de Teste

```
stimulus: process
    file vecFile : text is in "test4.vec";
    variable vecLine : line;
    variable vecString : string;
begin
    while not endfile(vecFile) loop
        readline(vecFile, vecLine);
        read(vecLine, r, good => good_number);
        read (file_line, vecString);
        - - Converte os dados de entrada em estímulos
        - - Aplica os estímulos, espere algum tempo, etc.
    end loop;
    assert false report "Teste completo";
    wait;
end process;
```

6

Exercício

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity exercicio is
port(   clk   : in std_logic;
        reset : in std_logic;
        S_in  : in std_logic;
        S_out : out std_logic);
end exercicio ;
```

```
architecture Comportamental of exercicio is
type state_type is (S0,S1,S2,S3,S4);
signal Current_State, Next_State : state_type;
begin

process(clk)
begin
   if( reset = '1' ) then
      Current_State <= 'S0';
   elsif (clk'event and clk = '1') then
      Current_State <= Next_State
   end if;
end process;
```

```
process(Current_State, S_in)
   begin
      case Current_State is
         when S0 =>
            S_out <= '0';
            if ( s_in = '0' ) then
               Next_State <= S0;
            else
               Next_State <= S1;
            end if;
         when S1 =>
            S_out <= '1';
            if ( S_in = '0' ) then
               Next_State <= S3;
            else
               Next_State <= S2;
            end if;
         when S2 =>
            S_out <= '0';
            if ( S_in = '0' ) then
               Next_State <= S0;
            else
               Next_State <= S3;
            end if;
         when S3 =>
            S_out <= '1';
            if (S_in = '0' ) then
               Next_State <= S2;
            else
               Next_State <= S4;
            end if;
         when S4 =>
            S_out <= '1';
            if ( S_in = '0' ) then
               Next_State <= S2;
            else
               Next_State <= S1;
            end if;
         when others =>
            NULL;
      end case;
   end if;
end process;
end exercicio;
```

Referências

- Créditos slides anteriores:
  - Cíntia B. Margi
  - Marco Túlio C. Andrade
- Wakerly, John F. Digital Design: Principles and Practices. Prentice-Hall, 4th edition, 2006.
  - Leitura mínima: capítulo 5, seções 5.1 e 5.3
- Peter J. Ashenden VHDL Tutorial. Elsevier, 2004

VHDL

Sistemas Digitais II

PCS2304-2015S1

Antônio Mauro Saraiva
Bruno de Carvalho Albertini
Marco Túlio de Carvalho Andrade