MC542

Organização de Computadores
Teoria e Prática

2007

Prof. Paulo Cesar Centoducatte

ducatte@ic.unicamp.br

www.ic.unicamp.br/~ducatte

MC542

Circuitos Lógicos

Introdução a VHDL

"DDCA" - (Capítulo: 4)
"FDL" - (Capítulo: Apêndice A)

Introdução a VHDL
Sumário

- Níveis de Abstração
- Conceitos Básicos
- VHDL - uma visão geral
- Estrutura do Código VHDL
- Declarações
- Constantes
- Variáveis
- Tipos
- Operadores Aritméticos
  - rem e mod
- Exemplos de Números
- Ponto Flutuante
- Tipos Físicos
- Tipos Enumerados

Introdução a VHDL
Sumário

- Caracteres
- Booleans
- Bits
- Standard Logic
- Entidade
- Arquitetura
- Atribuições
  - Constante, variáveis e sinais
- Operadores Lógicos
- Operadores Relacionáis
- Modelo VHDL Completo
- Packages
- Library e Use
- Subtipos
  - Conversão de tipos

Introdução a VHDL
Sumário

- Processos
  - Construtores Seqüenciais
    - » if, case, null, for e wait
- Vetores
  - Atribuições
- Comandos Concorrentes
  - when, select
- Atributos

Introdução

- Vantagens do Uso de HDLs e Ferramentas de Sínteses:
  - Aumento da produtividade, diminuindo o ciclo de desenvolvimento
  - Redução dos custos NRE (Non-Recurring Engineering)
  - Reusabilidade
  - Facilidade em introduzir alterações nos projetos
  - Exploração de alternativas de arquiteturas
  - Exploração de alternativas tecnológicas
  - Geração de circuitos testáveis automaticamente
  - Facilidades na verificação do projeto

Introdução

- Metodologia de Projeto

  - Descrição do sistema completo em um nível de abstração usando uma linguagem de descrição de hardware (HDL - Hardware Description Language) e uso de ferramentas automáticas para particionamento e síntese.

  - A descrição do hardware deve ser independente da tecnologia a ser usada na implementação.
    - » PCB ou MCMs (multichip modules)
    - » IC, ASICs, FPGA, PLD, full-custom

Hardware Description Languages (HDLs)

- HDL - Linguagem de programação usada para modelar a operação de um hardware
  - VHDL, Verilog, SystemC, ...
  -
- VHDL - História
  - 1980
    - » USA Department of Defense (DOD)
      - Documentação
      - Metodologia de Projeto comum
      - re-usável com novas tecnologias
    - » O DOD, dentro do programa "Very High Speed Integreted Circuit" (VHSIC) criou um projeto com a finalidade de criar uma linguagem de descrição de hardware
      - VHSIC Hardware Description Language (VHDL)

Hardware Description Languages (HDLs)

- 1983
  - » Início do desenvolvimento de VHDL
    - IBM, Texas Instruments e Intermetrics

- 1987
  - » Todo projeto de eletrônica digital ligado ao DOD deveria ser descrito em VHDL
  - » IEEE - Institute of Electrical and Electronics Engineers
    - IEEE Standard 1076
    - F-22
      - Todos os subsistemas eletrônicos descritos em VHDL
      - O desenvolvimento dos subsistemas foram distribuídos em diversos subcontratos
      - Estabeleceu um marco no uso de VHDL e da metodologia de projeto Top-Down

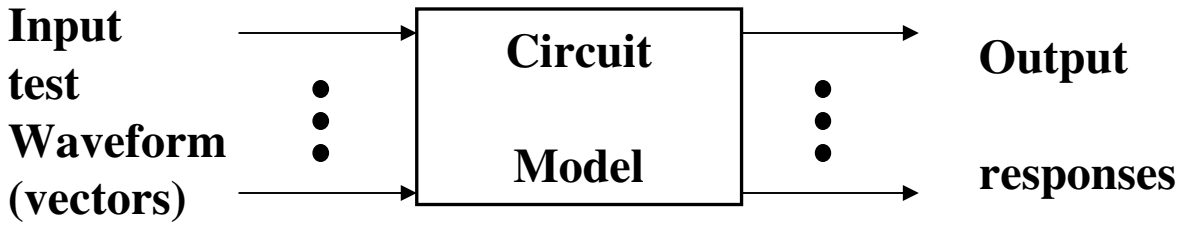
Hardware Description Languages (HDLs)

- 1993
  - » Revisão de VHDL - IEEE 1076'93
- 1996
  - » Ferramentas comerciais para Simulação e Sínteses para o padrão IEEE 1076'93
  - » IEEE 1076.3 - VHDL package para uso com ferramentas de sínteses
  - » IEEE 1076.4 (VITAL) - padrão para modelagem de bibliotecas para ASICs e FPGAs em VHDL
- 1997
  - » Revisão de VHDL (VHDL 1997)
- 2000
  - » Revisão de VHDL (VHDL 2000)
- .....

Ferramentas para Automação de Projetos

- Computer Aided Design (CAD) e Computer Aided Engineering (CAE)

- Simulação
  - Processo usado para verificar as características funcionais do modelo em qualquer nível comportamental. Usa o timing definido no modelo HDL, antes da sínteses, ou o timing extraído das células da tecnologia alvo, após a sínteses.



Ferramentas para Automação de Projetos

- **Register Transfer Level Synthesis**
  - É o processo que mapeia um modelo de um hardware descrito em HDL no nível RTL em uma implementação otimizada no nível de gate, em uma tecnologia específica.

Ferramentas para Automação de Projetos

- Otimizações no nível RTL
  - Algumas transformações efetuadas
    - » expansão - sub-programas são expandidos in-line
    - » constant folding - Ex. A + 3 + 5 => A + 8
    - » loop unrolling
    - » dead code removel - código não utilizado é removido
    - » bit minimization - Ex.: codificação dos estados de FSM
  - Uso de CFG (Control-data Flow Graph) como representação interna
    - » escalonamento
    - » particionamento
    - » resource binding

Ferramentas para Automação de Projetos

- **Otimizações no nível lógico**
  - otimização da lógica combinacional
  - reestruturação das equações booleanas
  - minimização
  - equation flatting
  - fatorização das equações

Síntese e Otimizações

- **Restrições**
  - Global - Aplicada igualmente a todo o projeto
    - » Exemplos: Biblioteca do fabricante; tensão e temperatura de uso.
  - Específica por circuito - Aplicada a um circuito particular
    - » Exemplos:
      - Área - área máxima (# gates equivalentes, # transistores)
      - Timing - input e output loading, máx. fan-out, capacidade de driving das entradas, mínima freqüência do clock, etc.
      - Potência - máxima potência consumida
      - Testabilidade - tipo das células para scan, scan parcial ou full, boundary scan.

VHDL- Uma visão Geral

- Característica
  - Adequada à descrição de hardware
    - » programação seqüencial e paralela
  - Permite descrição em diferentes níveis de abstração
    - » Comportamental
    - » RTL
    - » Estrutural (gate)
  - Simulável
  - Sintetizável
  - Padrão

Níveis de Abstração

- **Comportamental**: Descrição utilizando construções de alto nível da linguagem
- **RTL**: Nível intermediário, inclui mapeamento de portas
- ***Gate Level***: Nível de portas lógicas

Conceitos Básicos

- **Time Step**: É o menor intervalo de tempo do simulador, em algumas ferramentas é definido pelo usuário. Para VHDL time-step é o tempo gasto para a resolução de uma iteração de todos os comandos concorrentes

- **Concorrência**: A cada time-step do simulador todos os comandos são executados concorrentemente. Também os processos (conjunto de comandos seqüenciais) ocorrem em concorrência com o restante dos comandos de um modelo VHDL

- **Tipo**: A linguagem VHDL é fortemente dependente dos tipos dos dados.

VHDL - Uma Visão Geral

- 5 tipos de unidades
  - **Entity** - define a interface do projeto, módulo, etc.
  - **Architecture** - descreve funcionalmente a entidade. Pode haver mais de uma arquitetura para uma mesma entidade.
  - **Package** - declarações comuns a todo o projeto. Exemplo: constantes, tipos de dados e subprogramas.
  - **Packge Body** - contém o corpo dos subprogramas definidos no Packge
  - **Configuration** - Faz a ligação de uma entidade com uma particular arquitetura, formando um componente.

VHDL - Uma Visão Geral

- **Packages**: Assim como em linguagens de programação são utilizadas bibliotecas (de funções, procedimentos, definições de tipos, declarações de constantes etc), em VHDL isto é feito com a utilização de Packages e Bibliotecas de componentes.

- **Entidades**: Define a interface de um componente: nome, tipo dos sinais de entrada e/ou saída, ...

- **Arquiteturas**: Define a funcionalidade de um componente e sua temporização. Uma mesma entidade pode possuir múltiplas arquiteturas e para efeito de simulação e síntese é usada a última arquitetura compilada.

VHDL - Uma Visão Geral

- **Processo:** É uma porção de código delimitada pelas palavras Process e End Process que contém comandos seqüenciais (são simulados em delta-delays que somados resultam em zero). Todos os processos de um modelo VHDL de um componente são executados concorrentemente em um time-step.

- **Função:** Uma função em VHDL tem comportamento similar as funções em outras linguagens, contudo funções em VHDL não afetam os parâmetros de entrada, simplesmente retornam um valor com tipo definido.

VHDL - Uma Visão Geral

- **Procedimento:** Um procedimento em VHDL tem comportamento similar aos procedimentos de outras linguagens e distinguem-se das funções pela possibilidade de alteração nos parâmetros de chamada.

- **Bloco:** O comando BLOCK pode ser utilizado tanto para definir hierarquia de circuitos como também em conjunto com expressões GUARD definindo condições de uso de sinais de escopo restrito.

- **Componente:** É descrito pelo par Entity e Architecture. Um modelo VHDL é dito estrutural se faz uso de instanciação de componentes.

Estrutura do Código VHDL

- **Declarações**
  - Objetos que serão usados em comandos concorrentes ou seqüenciais
  - Declarados antes da clausula begin em arquiteturas, blocos, processos, procedimentos e funções
- **Comandos concorrentes**
  - block e process - comandos que serão executados em paralelo, independentemente uns dos outros
- **Comandos seqüenciais**
  - comandos que serão executados de forma seqüencial, obedecendo o fluxo de controle
  - comandos após a clusula begin em processos

Declarações

- **Constante**: nome assinalado a um valor fixo
  - » CONSTANT vdd: real := 4.5;
  - » CONSTANT cinco: integer := 3 +2;

- **Variável**: nome assinalado a um valor que muda de acordo com um determinado processo
  - » VARIABLE largura_pulso: time range 1ns to 15ns := 3ns;
  - » VARIABLE memoria: bit_vector (0 to 7);

- **Sinal**: conectam entidades e transmitem mudanças de valores entre os processos (todo port é um sinal).
  - » SIGNAL contador : integer range 0 to 63;
  - » SIGNAL condicao : boolean := TRUE;

Constantes

```
constant number_of_bytes : integer := 4;

constant number_of_bits : integer := 8 * number_of_bytes;

constant e : real := 2.718281828;

constant prop_delay : time := 3 ns;

constant size_limit, count_limit : integer := 255;
```

Variáveis

```
variable index : integer := 0;

variable sum, average, largest : real;

variable start, finish : time := 0 ns;
```

Onde declarar?

```
architecture exemplo of entidade is
   constant pi : real := 3.14159;
begin
   process is
         variable contador : integer;
   begin
         … -- uso de pi e contador
   end process;
end architecture exemplo;
```

Tipos

- Pré-Definidos
  - Integer
  - Boolean
  - Bit
  - Time
- Declaração de tipos

```
type apples is range 0 to 100;
type oranges is range 0 to 100;
type grapes is range 100 downto 0;
```

- O tipo *apples* é incompatível com *oranges*

```
constant number_of_bits : integer := 32;
type bit_index is range 0 to number_of_bits-1;
```

OBS.: Variáveis do tipo integer são inicializadas com o valor padrão (valor mais a esquerda do intervalo)

Operadores Aritméticos

+ → soma ou identidade

- → subtração ou negação

* → multiplicação

/ → divisão

mod → módulo

rem → resto da divisão

abs → valor absoluto

** → exponenciação

rem e mod

REM

- A = (A / B) * B + (A rem B) : tem o mesmo sinal que A

5 rem 3 = 2
(-5) rem 3 = -2
5 rem (-3) = 2
(-5) rem (-3) = -2

MOD

- A = B * N + (A mod B) com |(A mod B)| < |B| : tem o mesmo sinal que B

5 mod 3 = 2
(-5) mod 3 = 1
5 mod (-3) = -1
(-5) mod (-3) = -2

Exemplos de Números

23		0		146
23.1		0.0		3.14159
46E5		1E+12		19e00
1.234E09		98.6E+21		34.0e-08
2#11111101#	=	16#FD#	=	16#0fd#
2#0.100#	=	8#0.4#	=	12#0.6#
2#1#E10	=	16#4#E2	=	10#1024#E+00
123_456		3.131_592_6		2#1111_1100_0000_0000#

Ponto Flutuante

- Pré-Definido: REAL
  - VHDL 2000 exige, no mínimo, 64 bits
  - VHDL 87 e VHDL 93 exigem, no mínimo, 32 bits

```
variable y : real;
```

- Definido pelo usuário

```
type input_level is range -10.0 to +10.0;
variable x : input_level;
```

Tipos Físicos

- Um número + uma unidade

```
type length is range 0 to 1E9
  units
    um;
    mm = 1000 um;
    m = 1000 mm;
    inch = 25400 um;
    foot = 12 inch;
  end units length;
```

Tipos Físicos

- Atenção para as unidades
  - Só é possível somar e subtrair tipos da mesma unidade
  - Multiplicação e divisão por inteiro ou real mantém a unidade
- Uso

```
variable distancia : length := 100 m;
```

Tipos Enumerados

- Exemplo

```
type alu_funcion is (disable, pass, add, subtract, multiply,
  divide);
type octal_digit is ('0', '1', '2', '3', '4', '5', '6', '7');
type control is (stop, pass);
```

- Uso

```
variable command : alu_function := pass;
variable status : control := pass;
```

Caracteres

- VHDL 97 só aceitava ASCII padrão (128 valores)
  - Gerava problemas, inclusive, com os comentários

- VHDL 2000 usa ISO 8859 Latin 1, representado com 8 bits

```
variable cmd_char, terminator : character;
cmd_char := 'P';
terminator := cr;
```

Booleans

```
type boolean is (false, true);
```

- Operadores
  - and, or, nand, nor, xor, xnor, not

- OBS.: and, or, nand e nor fazem *lazy evaluation* (ou *short circuit*)
  - O segundo operando não será avaliado se o primeiro já indicar a resposta

Bits

```
type bit is ('0', '1');
```

- Todos os operadores lógicos valem para bits
- Valores entre aspas simples

```
variable switch : bit := '0';
```

Standard Logic

- Pacote std_logic_1164

```
type std_ulogic is (
        'U', -- não iniciado (unitialized)
        'X', -- desconhecido (unknow) forte
        '0', -- zero forte
        '1', -- um forte
        'Z', -- alta impedância ou desconectado (tri-state)
        'W', -- desconhecido fraco
        'L', -- zero fraco
        'H', -- um fraco
        '-'); -- indiferente (don't care)
```

OBS.: -- inicia um comentário que vai até o fim da linha

Entidade

- Define a interface do componente com o restante do sistema

```
Entity nome IS
   GENERIC (lista_dos_genericos);
    PORT (lista_dos_ports);
END nome;
```

Port

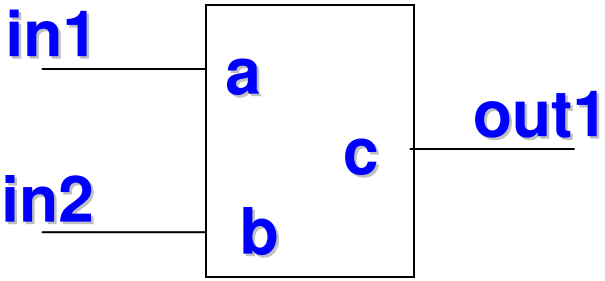
- Formato:

Port (nome: MODO tipo);

  - Modo: in, out, inout, buffer, linkage

- Exemplo:

```
Port (a, b : in    bit;
      c:  out bit);
```



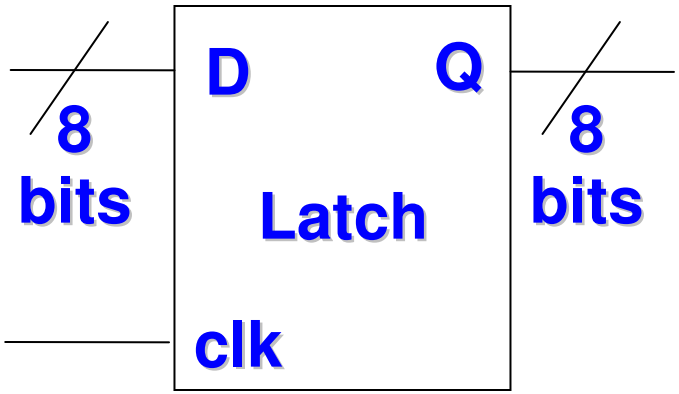
Exemplo

```
Entity porta_and IS
   GENERIC (numero_de_entradas: integer := 4);
   PORT (entradas : in bit_vector (1 to numero_de_entradas);
         saida    : out bit);
END porta_end;
```

- Port MODE:
  - IN: sinal é somente de entrada
  - OUT: sinal é somente de saída
  - Buffer: sinal é de entrada e saída (um de cada vez)
  - Inout: sinal é bidirecional, implica em um BUS
  - Linkage: direção do sinal é desconhecida

Exemplo

- Qual é a Entidade?




```
Entity latch is
  port (d    : in   bit_vector(7 downto 0);
        clk :  in   bit;
        q    :  out out bit_vector(7 downto 0);
End latch;
```

Exemplo

- Usando *generic*

```
Entity latch is
  generic(w : integer := 8);
  port (d     : in   bit_vector(w-1 downto 0);
        clk : in   bit;
        q    : out bit_vector(w-1 downto 0);
End latch;
```

Arquitetura

- Estabelece a relação entre entradas e saídas
  - Funções
  - Procedimentos
  - Execução paralela de processos
  - Instanciação de componentes

- Arquiteturas múltiplas
  - Utiliza a última compilada

Arquitetura

```
ARCHITECTURE  label OF nome_entidade IS
  --  parte declarativa (declarações de tipos, subtipos, sinais,
  --  funções, procedimentos, ...)

BEGIN
     --  comandos concorrentes
END label;
```

```
   ARCHITECTURE  rtl OF porta_and IS
    constant atraso : time := 5 ns;

   BEGIN
        y <=  a AND b AFTER atraso;
   END porta_and;
```

Qual a entidade?

Arquitetura

```
Entity porta_and is
  port (a, b  : in   bit;
        y     :  out bit);
End porta_and;
```

```
Entity porta_and is
  generic(w : integer := 8);
  port (a, b   :  in   bit_vector(w-1 downto 0);
        y      :  out bit_vector(w-1 downto 0);
End latch;
```

Atribuições

- Atribuição a sinal:

<=

- Atribuição a variável:

:=

- Inicialização (constante, sinal e variável):

:=

Atribuições de Dados

- **Constante:**

```
Constant cnt_reset : bit_vector(0 to 3) := "1001";
```

- **Variável:**

```
var1 := 2005;
```

- **Sinal:**

```
dado <= '1';
d <= '1', '0' AFTER 5 ns;
```

Atribuições e Operadores Aritméticos

Operador	Operação	Exemplo
+	Adição	I := i + 2;
-	Subtração	J <= j -10;
*	Multiplicação	M := fator * n;
/	Divisão	K := i / 2;
**	Potenciação	I := i ** 3;
ABS	Valor absoluto	Y <= ABS(tmp)
MOD	Módulo	Z <= A MOD B;
REM	resto	R <= A REM B;

Operadores Lógicos

- Predefinidos para os tipos: bit; boolean, std_logic, std_ulogic
  - NOT
  - AND
  - NAND
  - OR
  - NOR
  - XOR
  - XNOR

Operadores Relacionais

Operador	Operação
=	Igual
/=	Diferente
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que

Modelo Completo

- Indica as bibliotecas utilizadas
- Faz uso das definições contidas nas bibliotecas
- Descreve a entidade
- Descreve a arquitetura

Exemplo: or de 2 entradas

```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity or2 is
          --  porta or de duas entradas
  port (i1, i2 :  in   bit;
        out1 : out bit);
End or2;

Architecture rtl of or2 is
Begin
     out1 <= i1 or i2;
End rtl;
```

Packages

- Coleção de declarações comuns definidas fora dos modelos (corpo + declaração)

- Uso de:
  - Tipos; Subtipos
  - Subprogramas (Funções e Procedimentos)
  - Constantes; Sinais; Aliases
  - Atributos
  - Componentes

Packages

```
PACKAGE label IS

    --  declarações;

END label;

Package BODY label IS

   --  Corpo de subprogramas;

END label;
```

Library e USE

- LIBRARY nome_da_biblioteca;

```
Library IEEE;
```

- A cláusula USE torna os pacotes visíveis em entidades e arquiteturas
  - USE nome_biblioteca.nome_package;
  - USE nome_package.identificador;

```
USE IEEE.std_ulogic_1164.all;
USE math.all;
USE textio.all;
```

Subtipos

- **Formato:**

```
SUBTYPE natural IS INTEGER RANGE 0 to 2147483647;

SUBTYPE positive IS INTEGER RANGE 1 to 2147483647;
```

Conversão de Tipos

- Possível para conversão de alguns tipos
  - Inteiro --> Ponto Flutuante

- Formato: tipo(objeto)
  - Exemplo:

```
soma := REAL(N1) + N2;
```

- OBS.: Funções de Conversão definidas nos pacotes.

VHDL - Exemplo

- Qual o circuito implementado (em termos de portas lógicas)?

```
Library IEEE;
use IEEE.std_logic_1164.all;
Entity cct1 IS
   Port (in1, in2,in3,in4 : IN std_logic;
         out1, out2 : OUT std_logic);
End cct1;
Architecture rtl OF cct1 IS
  signal out_i : std_logic;
Begin
   out_i  <=  in1 and in2 and in3;
   out1   <=  out_i;
   out2   <=  in4 XOR out_i;
End rtl
```

Processos

- Um processo equivale a um comando concorrente.
- As instruções no interior do processo são instruções seqüênciais.
- Todos os processos são executados concorrentemente.
- Cada instrução dentro de um processo leva um tempo "delta time" que somados são iguais a zero

Processo

```
Label: PROCESS (lista de sensitividade)
            cláusulas declarativas;
       BEGIN
            --  comentários
               inicializações;
               cláusulas acertivas;
          END PROCESS label;
```

OBS.: A lista de sensibilidade funciona como um comando WAIT ON

Process - exemplo

```
Architecture bhv OF generic_decoder IS
Begin
PROCESS (sel, en)
   BEGIN
      y <= (others => '1');
      FOR i IN y'range LOOP
          IF (en = '1' and (bvtoi(To_Bitvector(sel)) = i))
          THEN
              y(i) <= '0';
          END IF;
      END LOOP;
END PROCESS;
END bhv
```

atributo

Função de conversão de tipo bit_vector para inteiro

Construtores Seqüenciais

- Válidos quando dentro de processos
- Nem todos são sintetizáveis
  - if, case, null, for
  - while, loop, exit, next, wait

If

```
[if_label:] if expressão_lógica then
  ...
elsif expressão_lógica then
  ...
else
  ...
end if [if_label];
```

Case

```
[case_label:] case expression is
  when opção1 =>
        ...
  when opção2 =>
        ...
  when others =>
        ...
end case [case_label];
```

case (Exemplo)

```
case opcode is
  when load | add | subtract =>
       operand := memory_operand;
  when store | jump | jumpsub | branch =>
       operand := address_operand;
  when others =>
       operand := none;
end case;
```

case

- Também pode ser indicado intervalo

```
when 0 to 9 =>
```

- OBS.: Para síntese, é necessário cobrir todas as opções
  - Usar others quando em dúvida
  - std_logic tem 9 valores!

null

- Comando nulo

for

```
[for_label:] for identificador in intervalo loop
  ...
end loop [for_label];
```

- **OBS.:**
  - Para hardware, significa replicação
  - *identificador* não precisa ser declarado
    - » *e não pode ter valor atribuído*

for (exemplo)

```
for count_value in 0 to 127 loop
  count_out <= count_value;
  wait for 5 ns;
end loop;
```

for (exemplo)

```
type controller_state is (initial, idle, active, error);
…
for state in controller_state loop
   …
end loop;
```

wait

Formato

WAIT [ON lista]
espera atividade em algum sinal da lista

WAIT [UNTIL condição]
espera até que condição ocorra (condição seja verdadeira)

WAIT [FOR tempo]
espera pelo tempo especificado

Wait: exemplo

```
Process
    variable i : integer := 0;
begin
    i := i + 1;
    j <= j + 1;
    wait on k;
    wait for 100 ns;
    wait on m until j > 5; -- Condição só é testada
                           -- quando ocorre um evento em m
end process;
```

wait

```
wait;  --  espera infinita
```

```
wait until clk = '1'; -- espera até que um evento satisfaça
  a condição clk = '1'
```

```
wait on clk until reset = '0'; -- espera até que ocorra um
  evento em clk e que se satisfaça a condição reset = '0'
```

```
wait until trigger = '1' for 1 ms; -- espera até que
  ocorra um evento em trigger ou se passe 1 milisegundo
```

Wait: exemplo

```
exemplo: process
        variable var : integr := 0;
        signal x, y : bit := '0';
    begin
        var := var + 1;
        var := var + var;
        x <= not y;
        x <= y;
        wait;   --  por quanto tempo?
    End process exemplo;
```

Vetores

```
type word is array (0 to 31) of bit;

type word is array (31 downto 0) of bit;

type controller_state is (initial, idle, active, error);

type state_counts is array (idle to error) of natural;

type matrix is array (1 to 3, 1 to 3) of real;
```

Atribuição de Valores para Vetores

```
subtype coeff_ram_address is integer range 0 to 63;

type coeff_array is array (coeff_ram_address) of real;

variable coeff : coeff_array := (0 => 1.6, 1 => 2.3, 2
⇒1.6, 3 to 63 => 0.0);

variable coeff: coeff_array := (0 => 1.6, 1 => 2.3,
  others => 0.0);

variable coeff : coeff_array := (0 | 2 => 1.6, 1 =>
  2.3, others => 0.0);
```

Vetores

```
type bit_vector is array (natural range <>) of bit;

subtype byte is bit_vector(7 downto 0);

type std_ulogic_vector is array (natural range <>) of
  std_ulogic;

variable channel_busy_register : bit_vector(1 to 4);

variable  current_test : std_ulogic_vector(0 to 13) :=
  "ZZZZZZZZZZ----";
```

Exemplos

- Escrever um modelo para um latch tipo D

```
Library ieee;
use ieee.std_logic_1164.all;

entity latch_d is
  port (dado, enable : in std_logic;
        q, notq : out std_logic);
end latch_d;
```

```
architecture rtl of latch_d is
begin
latch: process
begin
   wait until enable = '1'
   q <= dado;
   notq <= not dado;
end process latch;
end rtl;
```

Exemplos

- Latch D com if

```
Library ieee;
use ieee.std_logic_1164.all;
entity latch_d is
  port (dado, enable : in std_logic;
          q, notq : out std_logic);
end latch_d;
architecture rtl of latch_d is
begin
latchD: process (dado,enable)
begin
  IF enable = '1' THEN
          Q <= dado;
          notQ <= not dado
    END IF;
end process latchD;
end rtl;
```

Exemplo: Flip-Flop D

```
entity FlipFlopD is
     port (d, clk : in bit;
           q, notq : out bit);
end entity FlipFlopD;
architecture behavior of FlipFlopD is
   signal state : bit;
begin
   process (clk) is
   begin
        if (clk'event and clk = '1') then
              state <= d;
        end if;
        q <= state;
        notq <= not state;
   end process;
end architecture behavior;
```

tem que incluir state também?

```
q <= d;
notq <= not d;
```

Comandos Concorrentes (paralelos)

- Executados fora dos processos
- Possuem equivalentes para serem executados dentro dos processos
- Cada um tem o seu lugar, não é permitida a troca

when (≈ if)

```
architecture bhv of M is
begin
   process (sel, d0, d1) is
         if (sel = '0') then
           z <= d0;
         else
           z <= d1;
         end if;
   end process;
end architecture bhv;
```

```
architecture bhv of M is
begin
   z <= d0 when sel = '0' else
   d1;
end architecture bhv;
```

when

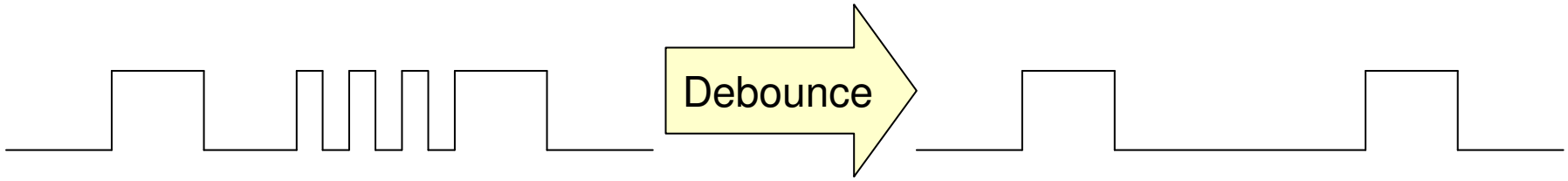
```
architecture bhv of M is
begin
  z <=  d0 when sel0 = '0' and sel1 = '0' else
        d1 when sel0 = '0' and sel1 = '1' else
        unaffected when sel0 = '1' and sel1 = '0'
       else
           d2;
end architecture bhv;
```

select (≈ case)

```
with alu_function select
  result <= a + b when alu_add | alu_add_u,
            a - b when alu_sub | alu_sub_u,
            a and b when alu_and,
            a or b when alu_or
            a when alu_pass_a;
```

Exercício

- Circuito para fazer debounce
  - Remove oscilações do circuito



- Assuma que as oscilações durem menos de 5 ciclos de clock

Atributos

- Atributos
  - A'left(N)
  - A'rirht(N)
  - A'low(N)
  - A'high(N)
  - A'range(N)
  - A'reverse_range(N)
  - A'length(N)
  - A'ascending(N)

type A is array (1 to 4 , 31 downto 0) of boolean;

- A'left(1) = 1
- A'right(2) = 0
- A'low(1) = 1
- A'high(2) = 31
- A'range(1) is 1 to 4
- A'reverse_range(2) is 0 to 31
- A'length(2) = 32
- A'ascending(1) = true
- A'ascending(2) = false

Atributos

```
type resistance is range 0 to
  1E9
    units
      ohm;
      kohm = 1000 ohm;
      Mohm = 1000 kohm;
    end units resistance;
```

```
resistance'left = 0 ohm;
resistance'right = 1E9 ohm;
resistance'low = 0 ohm;
resistance'high = 1E9 ohm;
resistance'ascending = true;
resistance'image(2 kohm) = "2000
   ohm";
resistance'value("5 Mohm") =
   5_000_000 ohm;
```

Atributos

```
type set_index_range is range 21
   downto 11;
type logic_level is (unknown, low,
   undriven, high);
```

```
set_index_range'left = 21;
set_index_range'right = 11;
set_index_range'low = 11;
set_index_range'high = 21;
set_index_range'ascending = false;
set_index_range'image(14) = "14";
set_index_range'value("20") = 20;
```

```
logic_level'left = unknown;
logic_level'right = high;
logic_level'low = unknown;
logic_level'high = high;
logic_level'ascending = true;
logic_level'image(undriven) =
   "undriven";
logic_level'value("Low") = low;
logic_level'pos(unknown) = 0;
logic_level'val(3) = high;
logic_level'succ(unknown) = low;
logic_level'pred(undriven) = low;
```