Universidade de São Paulo - São Carlos,SP
Instituto de Ciências Matemáticas e de Computação

USP - ICMC - SSC
SCE 0703 (PISE) - 2o. Semestre 2008

Disciplina de Projeto e Implementação de Sistemas Embarcados I

Prof. Fernando Santos Osório
Email: fosorio [at] { icmc. usp. br , gmail. com }
Web: http://www.icmc.usp.br/~fosorio/

Aula 03

Aula 03 - HDL

Agenda:

1. **Introdução: HDL - *Harware Description Languages***
   - 1.1 Conceito de Linguagens de Descrição de Hardware
   - 1.2 VHDL, Verilog, AHDL, SystemC, ...
2. **Conceitos Básicos de VHDL**
   - 2.1 Elementos Gerais da Linguagem
     - Library (IEEE Std) / Entity / Architecture
   - 2.2 Componentes: Interfaces / Behavioral / Structural
     - Test Benches / Language Elements
   - 2.3 Exemplos
3. **Ferramentas da Altera**
   - Quartus II - VHDL
   - ModelSim - VHDL
4. **Discussão Final**

1. Introdução
HDL- *Harware Description Language*

HDL - Conceitos e Origem

- Uma Linguagem de Descrição de Hardware, ou HDL, é uma classe de linguagem de computação e/ou programação usada para descrever de modo formal Circuitos Eletrônicos;
- Através de uma HDL podemos descrever o *funcionamento* de um circuito, seu *projeto* e sua *organização*, bem como é possível usar a HDL para desenvolver *testes e verificações* do circuito através de simulações;

HDL - Descrição de Circuitos Eletrônicos:

Portas Lógicas, Circuitos Combinatórios, Circuitos Integrados Digitais (Memórias, Decodificadores, Contadores, ULA, ...), Circuitos Analógicos, Microprocessadores, Processadores Dedicados, ...

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos e Origem

- As primeiras linguagens HDL foram:
  ISP (*Instruction Set Processor*), desenvolvida em Carnegie Mellon, e
  KARL, desenvolvida na Univ. de Kaiserslautern, por volta de ~1977;
- ISP era similar a uma linguagem de programação de softwares usada para descrever relações entre entradas e saídas de um projeto;
- KARL incluia suporte para *VLSI chip floorplanning* e projetos de hardware;
- ABEL foi introduzida em 1983 pela Data-I/O. Era orientada ao projeto e descrição de PLDs (*Programmable logical devices*) e FSMs (*Finite state machines*);
- Verilog foi a 1a. linguagem HDL moderna, tendo sido introduzida em 1985 pela *Gateway Design Automation*, depois adquirida pela Cadence;
- VHDL foi criada por uma iniciativa do DoD (*US Department of Defense*) no final dos anos 80 com o suporte da IEEE, tornando-se um padrão para projetos => IEEE Standards 1076.1, 1076.2, 1076.3, 1164 / VHDL-4.0 2008
- SystemC, em 1999 é anunciada a *Open SystemC Initiative* (Synopsys)

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos e Origem

Referências

ABEL - Advanced Boolean Expression Language (now owned by Xilinx Inc.)

Verilog - Verilog Ressouces - http://www.verilog.com/

System Verilog - http://www.eda.org/sv/ - http://www.systemverilog.org/

IEEE Stardard 2001:

http://ieeexplore.ieee.org/xpl/standardstoc.jsp?isnumber=20656&isYear=2001

VHDL - Sites: http://www.vhdl.org/

http://www.eda.org/ - http://www.eda.org/rassp/vhdl/

VHDL FAQ - http://www.vhdl.org/comp.lang.vhdl/FAQ1.html

SystemC - Site Oficial: http://www.systemc.org/home

Board of Members: http://www.systemc.org/about/gallery/

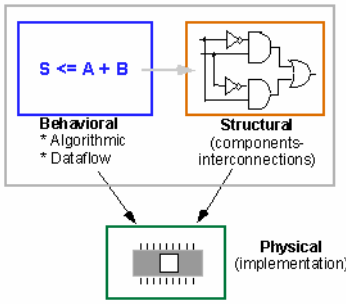
Veja também...

Wikipedia (en) - HDL, VHDL, Verilog, SystemC

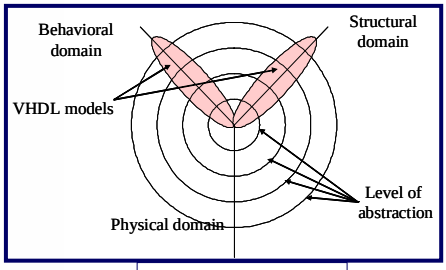
1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Domínios de Descrição / Níveis de Abstração



1: Levels of abstraction: Behavioral, Structural and Physical

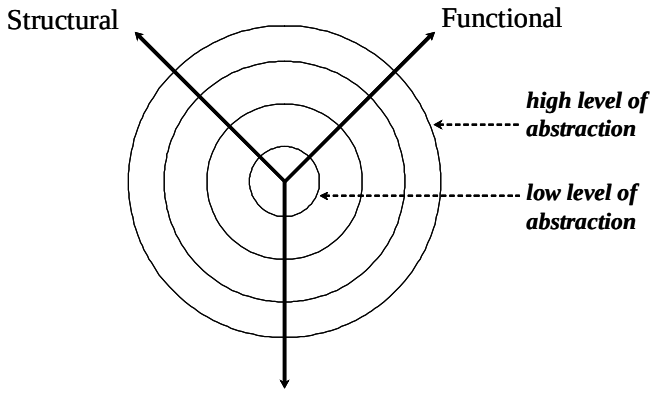


Domains of Description :
Gajski & Kahn
Y-Chart

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Domínios de Descrição / Níveis de Abstração



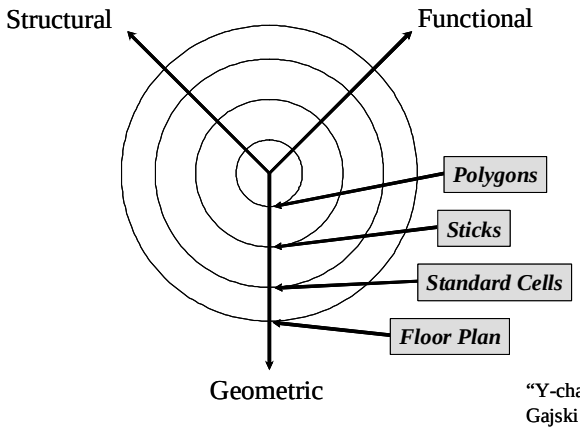
"Y-chart" due to Gajski & Kahn

USP - SCE-703 - 2008-2
Proj. e Implem. de Sistemas Embarcados I
Prof. Fernando Osório

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Domínios de Descrição / Níveis de Abstração

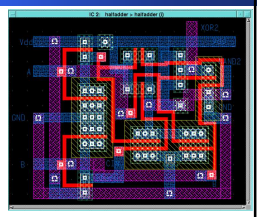


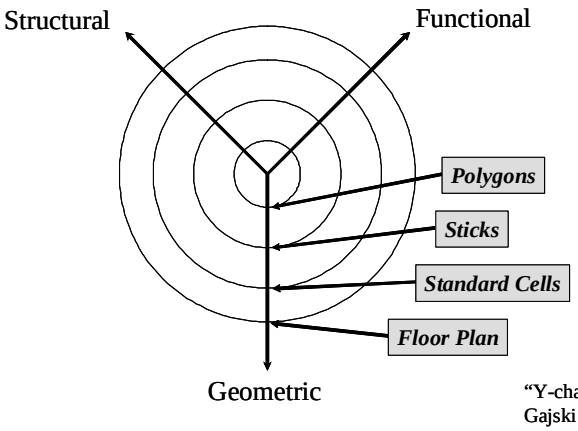
"Y-chart" due to Gajski & Kahn

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Domínios de Descrição / Níveis de Abstração





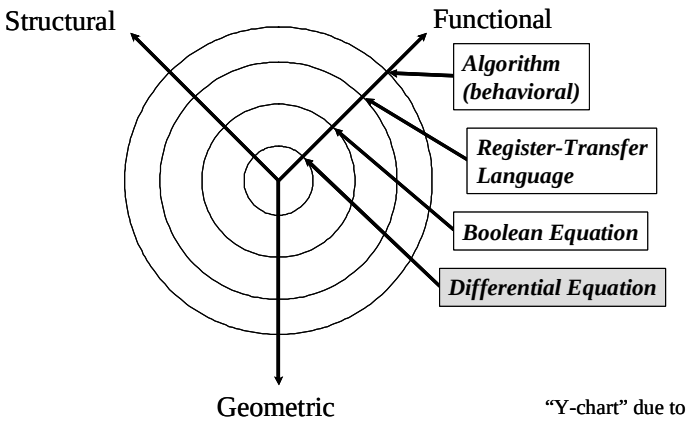
"Y-chart" due to Gajski & Kahn

USP - SCE-703 - 2008-2
Proj. e Implem. de Sistemas Embarcados I
Prof. Fernando Osório

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Domínios de Descrição / Níveis de Abstração



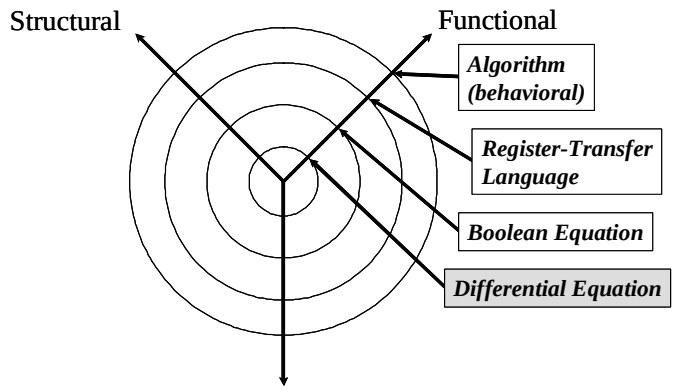
"Y-chart" due to Gajski & Kahn

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Domínios de Descrição / Níveis de Abstração

S:= ((not(A)) and B) Or
(A and (not(B)))
C:= A and B

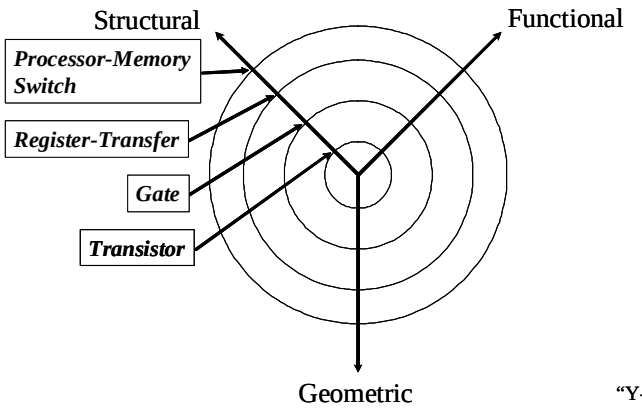


"Y-chart" due to Gajski & Kahn

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Domínios de Descrição / Níveis de Abstração

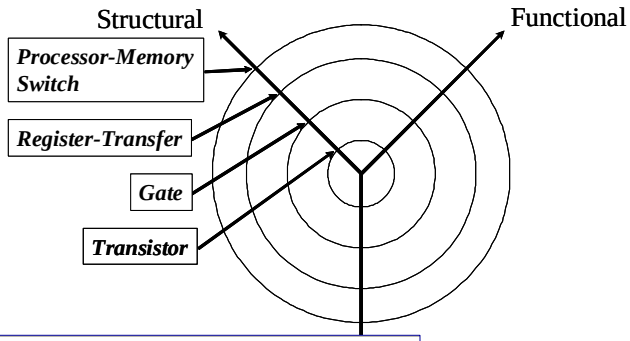


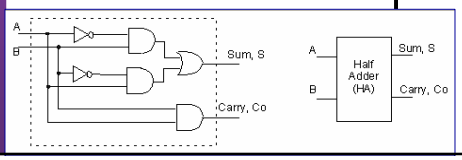
"Y-chart" due to Gajski & Kahn

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Domínios de Descrição / Níveis de Abstração



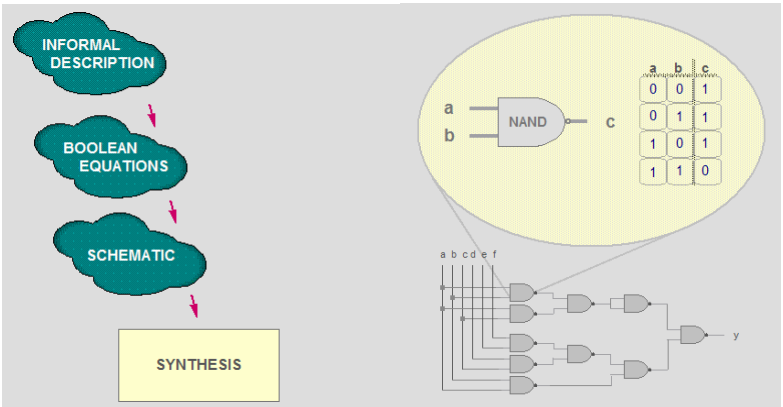


"Y-chart" due to Gajski & Kahn

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Projeto usando HDL

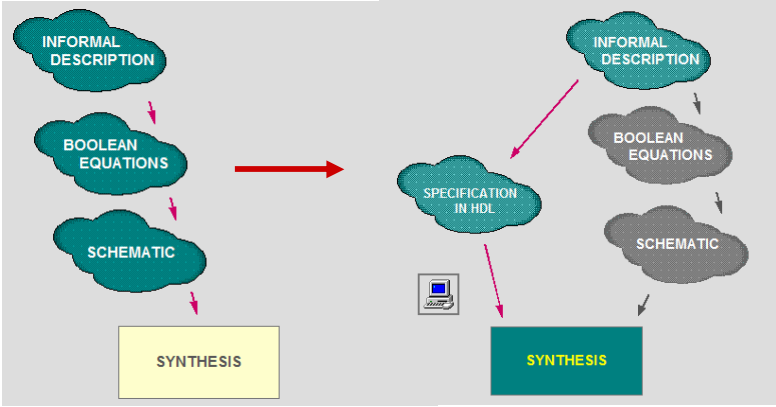


Antes: Projeto Tradicional

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Projeto usando HDL



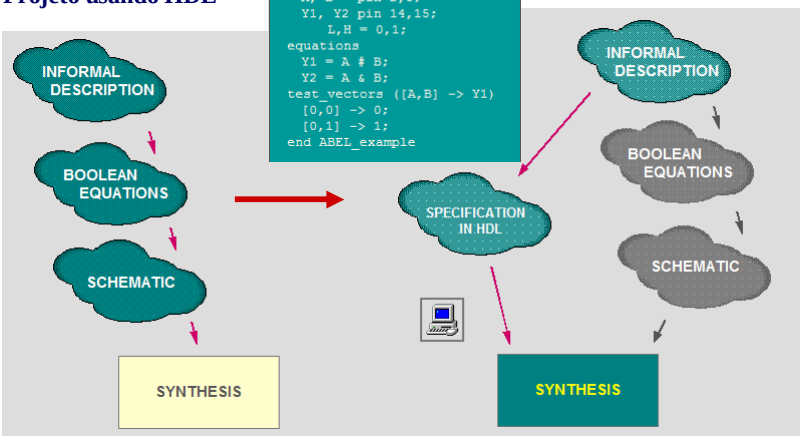
Depois: Projeto HDL

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Projeto usando HDL

```
module ABEL_example
title 'add & multiply'
US1 device 'p16L8';
  A, B    pin 2,3;
  Y1, Y2 pin 14,15;
     L,H = 0,1;
equations
  Y1 = A # B;
  Y2 = A & B;
test_vectors ([A,B] -> Y1)
  [0,0] -> 0;
  [0,1] -> 1;
end ABEL_example
```

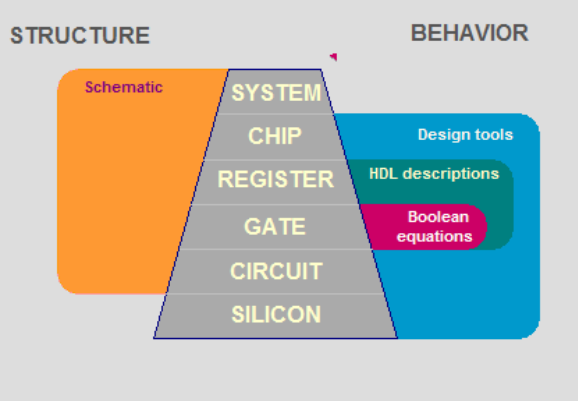


Depois: Projeto HDL

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Projeto usando HDL



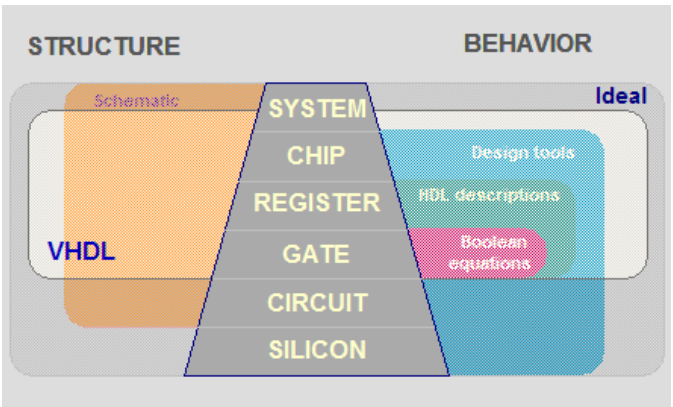
Depois: Projeto HDL

USP - SCE-703 - 2008-2
Proj. e Implem. de Sistemas Embarcados I
Prof. Fernando Osório

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Projeto usando HDL

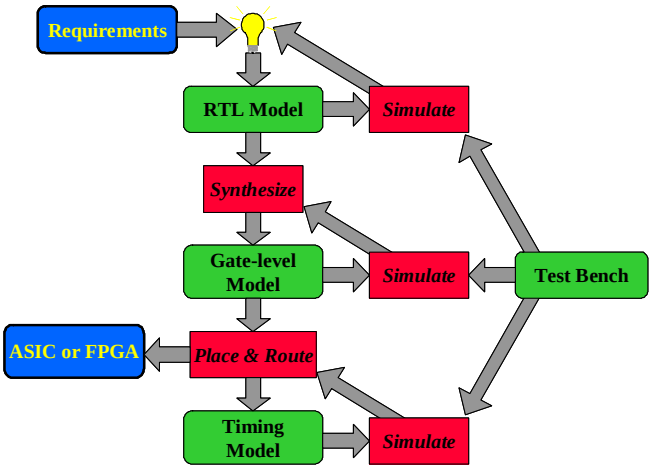


Depois: Projeto VHDL + Design Tools (EDA)

1. Introdução
HDL - *Harware Description Language*

HDL - Conceitos

Projeto usando HDL



2. *VHDL*
VHSIC Hardware Description Language

VHDL - Conceitos Básicos

Siglas VHDL = VHSIC HDL
= VHSIC Hardware Description Language
VHSIC = Very High Speed Integrated Circuit

VHDL => *Very High Speed Integrated Circuit Hardware Description Language*

- Criada em meados dos anos 80 pelo militares americados:
American Department of Defense "DoD";

- Objetivos:

> Descrição de alto nível (linguagem formal) de um sistema digital;

> Permitir a *simulação* de circuitos eletrônicos;

> Permitir a *síntese automática* de circuitos integrados (VLSI, ASIC, FPGA);

> Padronização (IEEE 1076-1993), Reuso, Prototipação

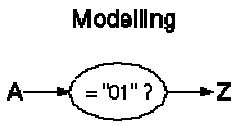
"Rapid Prototyping of Application Specific Circuits and Systems"

VHDL - Histórico

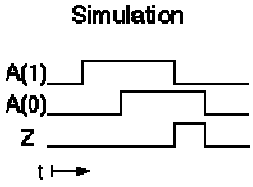
- Early `70s: Initial discussions (ABEL)
- Late `70s: Definition of requirements
- Mid -`82: Contract of development with IBM, Intermetrics and TI
- Mid -`84: Version 7.2
- Mid -`86: IEEE-Standard
- 1987: DoD adopts the standard -> IEEE.1076-1987
- Mid -`88: Increasing support by CAE manufacturers
- Late `91: Revision
- 1993: New standard IEEE.1076-1993
- 1999: VHDL-AMS extension - IEEE 1076.1 (VHDL-AMS)
- IEEE Standards 1076.2, 1076.3, 1076.4, 1076,5 / 1164 STD Logic
- 2006: VHDL-3.0 October 2006
- 2008: VHDL-4.0 - IEEE.1076-2008 [http://www.accellera.org/]

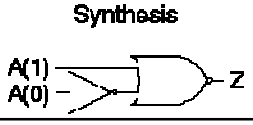
VHDL - Conceitos

* Requisitos da Linguagem VHDL:
  Abstração / Modularidade / Concorrência (paralelismo) / Hierarquia
* Objetivos da Linguagem VHDL: Modelagem, Simulação e Síntese




```
Z <= '1' when A="01"
     else '0';
```





2. *VHDL*

VHDL - Conceitos

*** Requisitos da Linguagem VHDL:**
Abstração

- **VHDL suporta descrições de componentes, bem como, de sistemas em diferentes níveis de abstração:** ***Gates, RTL, Processor-Memory Switch, Microprocessor and Systems;***
- **Portas e atrasos de componentes;**
- **Entradas e Saídas, Ciclos de Clock;**
- **Comportamento abstrato, sem noção de atrasos;**
- **Blocos: componentes vistos como "caixas pretas", apresentando uma visão externa da interface;**
- **Conexão de Macro-Blocos;**
- **Descrição: estrutural (*schematic*) ou comportamental (*algorithmic* / *data flow*).**

2. *VHDL*

VHDL - Conceitos

*** Requisitos da Linguagem VHDL:**
Modularidade

- **Cada componente em VHDL é referido como uma entidade (entity) e tem uma interface claramente definida;**
- **A parte externa de um componente (interface) é chamada de "*entity declaration*";**
- **A parte interna de um componente (structure/behavior) é chamada de "*architecture declaration*";**
- **Podem haver múltiplas arquiteturas, inclusive em diferentes níveis de abstração, associadas com uma mesma entidade;**

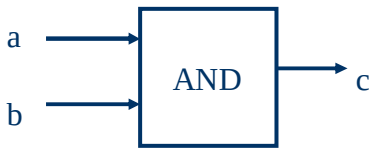
Concorrência Concurrent / Sequential - Behavior

- ***Sequential Statements*: Procedural statements**
- ***Concurrent Statements*: Process statements and Component Instances**

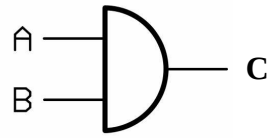
Algorithmic
Data Flow Style

VHDL - Exemplo

Interface: Entity

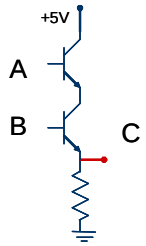


Structural: Schematic / Gate




```
entity  AND2 is
  port (a, b: in bit ;
              c : out bit);
  end  AND2;

architecture  behavioral_2 of AND2 is
begin
     c <= a and b;
end  behavioral_2;
```

VHDL



Behavioral: Truth Table

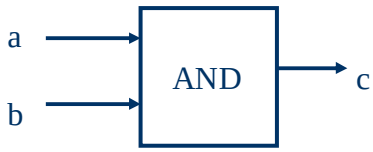
B \ A	0	1
0	0	0
1	0	1

Structural: Transistor

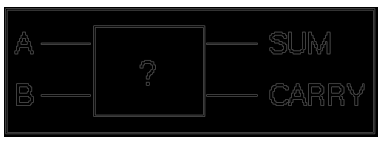
VHDL - Exemplo

Entity

Interface: Entity (external)



Entity - Describes only the interface (I/O)
Circuit as a "Black-Box"



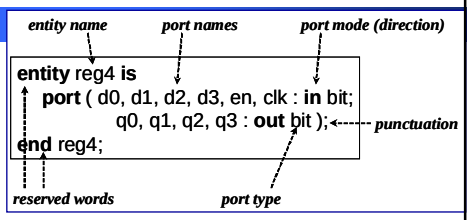

```
entity  AND2 is
  port (a, b: in bit ;
              c : out bit);
  end  AND2;
```

```
entity HALF_ADDER is
      port (
        A, B:              in   bit;
        SUM, CARRY: out bit);
   end HALF_ADDER;
```

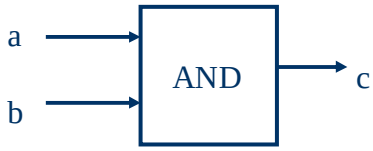
2. *VHDL*

VHDL - Exemplo

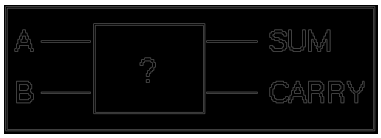
Entity



Interface: Entity (external)



Entity - Describes only the interface (I/O)
Circuit as a "Black-Box"




```
entity  AND2 is
  port (a, b: in bit ;
              c : out bit);
  end  AND2;
```

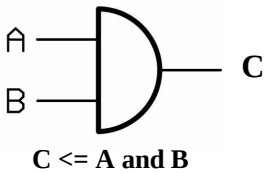
```
entity HALF_ADDER is
      port (
        A, B:              in   bit;
        SUM, CARRY: out bit);
    end HALF_ADDER;
```


2. *VHDL*

VHDL - Exemplo

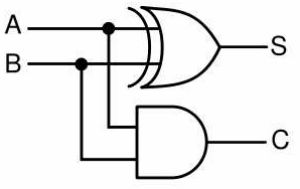
Architecture

Structural / Behavioral Description
Architecture (internal)



C <= A and B

Architecture: Describes circuit "function"
Structure / Behavior / Data Flow

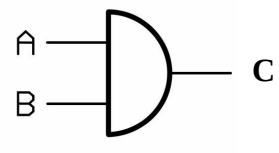


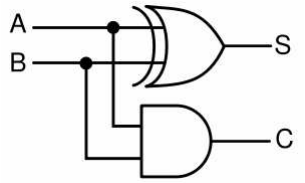

```
architecture  behavioral_2 of AND2 is
begin
     c <= a and b;
end  behavioral_2;
```

```
architecture RTL of HALF_ADDER is
begin
     SUM      <= A xor B;
     CARRY <= A and B;
end RTL;
```

2. *VHDL*

VHDL - Exemplo





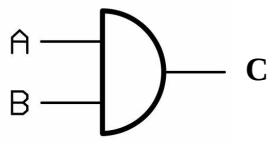

```
entity  AND2 is
  port (a, b: in bit ;
              c : out bit);
  end  AND2;

architecture  behavioral_2 of AND2 is
begin
     c <= a and b;
end  behavioral_2;
```

```
entity HALF_ADDER is
       port (
          A, B:              in   bit;
          SUM, CARRY: out bit);
end HALF_ADDER;

architecture RTL of HALF_ADDER is
begin
      SUM      <= A xor B;
      CARRY <= A and B;
end RTL;
```


2. *VHDL*

VHDL - Exemplo

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

entity  AND2 is
  port (a, b: in std_logic;
              c : out std_logic);
  end  AND2;

architecture  behavioral_2 of AND2 is
begin
     c <= a and b;
end  behavioral_2;
```



- BIT, BIT_VECTOR
  - Valores: “0” ou “1”
  - Atribuição de valor: bit_signal <= '0';
  - Nativos da linguagem VHDL, não precisam de declaração de biblioteca
- STD_LOGIC, STD_LOGIC_VECTOR
  - Valores: “0”, “1”, “-” (don’t care), “Z” (alta impedância), “X” (indeterminado)
  - Biblioteca ieee
  - Declarações necessárias
    - LIBRARY
    - USE

Library: Usa a biblioteca padrão IEEE
Evite misturar BIT com STD_LOGIC

2. *VHDL*

VHDL - Exemplo

Tipos de Sinais (I/O)

- **IN**: dados fluem para dentro da Entidade, que não pode escrever estes sinais
  > Ex. Clock, entradas de controle, entradas unidirecionais de dados
- **OUT**: dados fluem para fora da Entidade, que não pode ler estes sinais.
  > O modo OUT é usado quando a Entidade nunca lê estes dados
- **BUFFER**: dados fluem para fora da Entidade, que pode ler estes sinais, permitindo realimentação interna
  > No entanto, o BUFFER não pode ser usado para entrada de dados
- **INOUT**: dados podem fluir para dentro ou para fora da Entidade
  > Só deve ser usado se necessário
  Ex. Barramento de dados bidirecional
  > Design menos compreensível

Tipos de Dados

- **BIT, BIT_VECTOR**
  - Valores: “0” ou “1”
  - Atribuição de valor: bit_signal <= '0';
  - Nativos da linguagem VHDL, não precisam de declaração de biblioteca
- **STD_LOGIC, STD_LOGIC_VECTOR**
  - Valores: “0”, “1”, “-” (don’t care), “Z” (alta impedância), “X” (indeterminado)
  - Biblioteca ieee
  - Declarações necessárias
    * LIBRARY
    * USE

Vetores Declaração:

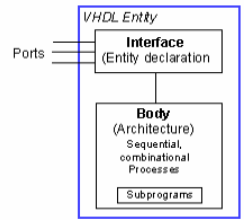
```
bit_vector_signal : BIT_VECTOR(
        maximum_index DOWNTO 0 );
```

Vetores Atribuição:

```
bit_vector_signal(0) <= ‘1’;
bit_vector_signal(0) <= bit_signal;
bit_vector_signal <= "0001";
```


2. *VHDL - Linguagem*

Linguagem VHDL - Componentes



Entity Declaration

The entity declaration defines the NAME of the entity and lists the input and output ports.
The general form is as follows:

entity NAME_OF_ENTITY **is** [**generic** *generic_declarations*);]
  port (*signal_names*: **mode** *type*;
    signal_names: **mode** *type*;
      :
    signal_names: **mode** *type*);
end [NAME_OF_ENTITY] ;

-- Lines which starts with "--" are comments
-- Este é um comentário (termina no final da linha)
-- A linguagem não é "case-sensitive" mas usualmente adota-se um estilo próprio

http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html

2. *VHDL - Linguagem*

Entity Declaration (cont.)

The NAME_OF_ENTITY is a user-selected identifier
* **Signal_names** consists of a comma separated list of one or more user-selected identifiers that specify external interface signals.
* **Mode**: is one of the reserved words to indicate the signal direction:
  > in – indicates that the signal is an input
  > out – indicates that the signal is an output of the entity whose value can only be read by other entities that use it.
  > buffer – indicates that the signal is an output of the entity whose value can be read inside the entity's architecture
  > inout – the signal can be an input or an output.

* **Type**: a built-in or user-defined signal type. Examples of types are: bit, bit_vector, Boolean, character, std_logic, and std_ulogic.
  > bit – can have the value 0 and 1
  > bit_vector – is a vector of bit values (e.g. bit_vector (0 to 7)
  > std_logic,std_ulogic,std_logic_vector,std_ulogic_vector: can have 9 values to indicate the value and strength of a signal. Std_ulogic and std_logic are preferred over the bit or bit_vector types.
  > boolean – can have the value TRUE and FALSE
  > integer – can have a range of integer values
  > real – can have a range of real values
  > character – any printing character
  > time – to indicate time

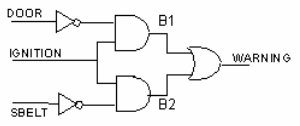
2. *VHDL - Linguagem*

Entity Declaration (cont.)

Generic: generic declarations are optional and determine the local constants used for timing and sizing (e.g. bus widths) the entity. A generic can have a default value.
The syntax for a generic follows:

```
generic (
constant_name: type [:=value] ;
constant_name: type [:=value] ;
:
constant_name: type [:=value] );
```

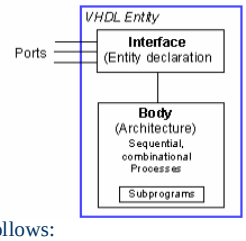
For the example of this figure, the entity declaration looks as follows:




```
-- comments: example of the buzzer circuit of the above figure
entity BUZZER is
    port (DOOR, IGNITION, SBELT: in std_logic;
          WARNING: out std_logic);
end BUZZER;
```

2. *VHDL - Linguagem*

Linguagem VHDL - Componentes



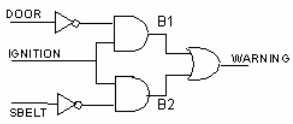
Architecture Declaration

The architecture body specifies how the circuit operates and how it is implemented. As discussed earlier, an entity or circuit can be specified in a variety of ways, such as behavioral, structural (interconnected components), or a combination of the above. The architecture body looks as follows:

```
architecture architecture_name of NAME_OF_ENTITY is
-- Declarations
        -- components declarations
        -- signal declarations
        -- constant declarations
        -- function declarations
        -- procedure declarations
        -- type declarations
        :
begin
-- Statements
        :
end architecture_name;
```


2. *VHDL - Linguagem*

Linguagem VHDL - Componentes



Architecture Declaration

The architecture body specifies how the circuit operates and how it is implemented. As discussed earlier, an entity or circuit can be specified in a variety of ways, such as **behavioral**, structural (interconnected components), or a combination of the above. The architecture body looks as follows:

```
architecture architecture_name of NAME_OF_ENTITY is
-- Declarations
        -- components declarations
        -- signal declarations
        -- constant declarations
        -- function declarations
        -- procedure declarations
        -- type declarations
        :
begin
-- Statements
        :
end architecture_name;
```

Behavioral model
The architecture body for the previous example, described at the behavioral level, is given below,

```
architecture behavioral of BUZZER is
begin
   WARNING <= (not DOOR and IGNITION) or
              (not SBELT and IGNITION);
end behavioral;
```

Linguagem VHDL - Componentes

Architecture Declaration: More Examples

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

entity AND2 is
    port (in1, in2: in std_logic;
          out1: out std_logic);
end AND2;

architecture behavioral_2 of AND2 is
begin
    out1 <= in1 and in2;
end behavioral_2;
```

Nota: observe que as atribuições "<=" são *comandos concorrentes*, ou seja, são executados ao mesmo tempo, onde sua ordem no texto não importa.

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

entity XNOR2 is
    port (A, B: in std_logic;
          Z: out std_logic);
end XNOR2;

architecture behavioral_xnor of XNOR2 is
    -- signal declaration (of internal signals X, Y)
    signal X, Y: std_logic;
begin
    X <= A and B;
    Y <= (not A) and (not B);
    Z <= X or Y;
end behavioral_xnor;
```

This implies that the statements are executed when one or more of the signals on the right hand side change their value (i.e. an event occurs on one of the signals).

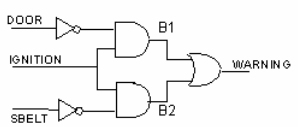
Linguagem VHDL - Componentes

Architecture Declaration

Architecture specifies how circuit operates and how it is implemented. Example of a **structural** (interconnected components) description.

```
architecture structural of BUZZER is
    -- Declarations
    component AND2
        port (in1, in2: in std_logic; out1: out std_logic);
    end component;
    component OR2
        port (in1, in2: in std_logic; out1: out std_logic);
    end component;
    component NOT1
        port (in1: in std_logic; out1: out std_logic);
    end component;
    -- declaration of signals used to interconnect gates
    signal DOOR_NOT, SBELT_NOT, B1, B2: std_logic;
```

```
begin
    -- Component instantiations statements
    U0: NOT1 port map (DOOR, DOOR_NOT);
    U1: NOT1 port map (SBELT, SBELT_NOT);
    U2: AND2 port map (IGNITION, DOOR_NOT, B1);
    U3: AND2 port map (IGNITION, SBELT_NOT, B2);
    U4: OR2  port map (B1, B2, WARNING);
end structural;
```



Linguagem VHDL - Componentes

Architecture Declaration

Architecture specifies how circuit operates and how it is implemented. Example of a **stru** (interconnected components) des

```
architecture structural of BUZ
   -- Declarations
   component AND2
       port (in1, in2: in std_logic; ou
   end component;
   component OR2
       port (in1, in2: in std_logic; ou
   end component;
   component NOT1
       port (in1: in std_logic; out1: o
   end component;
   -- declaration of signals used to interconnect gates
   signal DOOR_NOT, SBELT_NOT, B1, B2: std_logic;
```

```
begin
-- Alternative association between the ports
U0: NOT1 port map (in1 => DOOR, out1 => DOOR_NOT);
U1: NOT1 port map (in1 => SBELT, out1 => SBELT_NOT);
U2: AND2 port map (in1 => IGNITION, in2 => DOOR_NOT, out1 => B1);
U3: AND2 port map (in1 => IGNITION, in2 => SBELT_NOT, B2);
U4: OR2  port map (in1 => B1, in2 => B2, out1 => WARNING);
end structural;
```

```
begin
    -- Component instantiations statements
    U0: NOT1 port map (DOOR, DOOR_NOT);
    U1: NOT1 port map (SBELT, SBELT_NOT);
    U2: AND2 port map (IGNITION, DOOR_NOT, B1);
    U3: AND2 port map (IGNITION, SBELT_NOT, B2);
    U4: OR2  port map (B1, B2, WARNING);
end structural;
```

Nota: estes comandos também são concorrentes, não importando a ordem de sua declaração

Linguagem VHDL - Componentes da Linguagem

Data Types defined in the Standard Package.

VHDL has several predefined types in the standard package.

To use this package one has to include the following clause:

library std, work;
use std.standard.all;

Cont...

Types defined in the Package *Standard* of the *std* Library		
Type	**Range of values**	**Example**
bit	'0', '1'	signal A: bit :=1;
bit_vector	an array with each element of type bit	signal INBUS: bit_vector(7 downto 0);
boolean	FALSE, TRUE	variable TEST: Boolean :=FALSE'
character	any legal VHDL character (see package standard); printable characters must be placed between single quotes (e.g. '#')	variable VAL: character :='$';
file_open_kind*	read_mode, write_mode, append_mode	
file_open_status*	open_ok, status_error, name_error, mode_error	
integer	range is implementation dependent but includes at least $-(2^{31} - 1)$ to $+(2^{31} - 1)$	constant CONST1: integer :=129;
natural	integer starting with 0 up to the max specified in the implementation	variable VAR1: natural :=2;
positive	integer starting from 1 up the max specified in the implementation	variable VAR2: positive :=2;
real*	floating point number in the range of -1.0×10^{38} to $+1.0 \times 10^{38}$ (can be implementation dependent. *Not supported by the Foundation synthesis program.*	variable VAR3: real :=+64.2E12;
severity_level	note, warning, error, failure	

2. *VHDL - Linguagem*

VHDL

Type Conversion

```
entity QUAD_NAND2 is
     port (A, B: in bit_vector(3 downto 0);
      out4: out std_logic_vector (3 downto 0));
end QUAD_NAND2;

architecture behavioral_2 of QUAD_NAND2 is
begin
          out4 <= to_StdLogicVector(A and B);
end behavioral_2;
```

Since VHDL is a strongly typed language one cannot assign a value of one data type to a signal of a different data type. In general, it is preferred to the same data types for the signals in a design, such as std_logic (instead of a mix of std_logic and bit types). Sometimes one cannot avoid using different types. To allow assigning data between objects of different types, one needs to convert one type to the other. Fortunately there are functions available in several packages in the ieee library, such as the std_logic_1164 and the std_logic_arith packages. As an example, the std_logic_1164 package allows the following conversions:

Conversions supported by std_logic_1164 package	
Conversion	**Function**
std_ulogic to bit	to_bit(*expression*)
std_logic_vector to bit_vector	to_bitvector(*expression*)
std_ulogic_vector to bit_vector	to_bitvector(*expression*)
bit to std_ulogic	To_StdULogic(*expression*)
bit_vector to std_logic_vector	To_StdLogicVector(*expression*)
bit_vector to std_ulogic_vector	To_StdUlogicVector(*expression*)
std_ulogic to std_logic_vector	To_StdLogicVector(*expression*)
std_logic to std_ulogic_vector	To_StdUlogicVector(*expression*)

The IEEE std_logic_unsigned and the IEEE std_logic_arith packages allow additional conversions such as from an integer to std_logic_vector and vice versa.

USP - SCE-703 - 2008-2
Proj. e Implem. de Sistemas Embarcados I
Prof. Fernando Osório

2. *VHDL - Linguagem*

VHDL

Operators

7. Operators

VHDL supports different classes of operators that operate on signals, variables and constants. The different classes of operators are summarized below.

Class						
1. Logical operators	and	or	nand	nor	xor	xnor
2. Relational operators	=	/=	<	<=	>	>=
3. Shift operators	sll	srl	sla	sra	rol	ror
4.Addition operators	+	=	&			
5. Unary operators	+	-				
6. Multiplying op.	*	/	mod	rem		
7. Miscellaneous op.	**	abs	not			

The order of precedence is the highest for the operators of class 7, followed by class 6 with the lowest precedence for class 1. Unless parentheses are used, the operators with the highest precedence are applied first. Operators of the same class have the same precedence and are applied from left to right in an expression. As an example, consider the following std_ulogic_vectors, X (='010'), Y(='10'), and Z ('10101'). The expression

not X & Y **xor** Z **rol** 1

is equivalent to ((**not** X) & Y) **xor** (Z **rol** 1) = ((101) & 10) **xor** (01011) =(10110) **xor** (01011) = 11101. The xor is executed on a bit-per-bit basis.

2. *VHDL - Linguagem*

VHDL

Sequential (Process)

x

Parallel (Concurrent)

Paralelismo: VHDL (Hw) versus Programação (Sw)

* Parallel:

```
A <= B;
B <= A;
```

* Sequential:

```
Aux := A;
A := B;
B := Aux;
```

Signal x Variable

> Signal

Parallel:

```
A <= B;
B <= A;
```

> Variable

Sequential

```
Aux := A;
A := B;
B := Aux;
```


2. *VHDL - Linguagem*

Signal x Variable

Example of a process using Variables

```
architecture VAR of EXAMPLE is
     signal TRIGGER, RESULT: integer := 0;
begin
     process
          variable variable1: integer :=1;
          variable variable2: integer :=2;
          variable variable3: integer :=3;
     begin
          wait on TRIGGER;
          variable1 := variable2;
          variable2 := variable1 + variable3;
          variable3 := variable2;
          RESULT <= variable1 + variable2 + variable3;
     end process;
end VAR
```

Example of a process using Signals

```
architecture SIGN of EXAMPLE is
     signal TRIGGER, RESULT: integer := 0;
     signal signal1: integer :=1;
     signal signal2: integer :=2;
     signal signal3: integer :=3;
begin
     process
     begin
          wait on TRIGGER;
          signal1 <= signal2;
          signal2 <= signal1 + signal3;
          signal3 <= signal2;
          RESULT  <= signal1 + signal2 + signal3;
     end process;
end SIGN;
```

2. *VHDL - Linguagem*

Signal x Variable

Example of a process using Variables

```
architecture VAR of EXAMPLE is
     signal TRIGGER, RESULT: integer := 0;
begin
     process
           variable variable1: integer :=1;
           variable variable2: integer :=2;
           variable variable3: integer :=3;
     begin
           wait on TRIGGER;
           variable1 := variable2;
           variable2 := variable1 + variable3;
           variable3 := variable2;
           RESULT <= variable1 + variable2 + variab
     end process;
end VAR                    RESULT=12
```

Example of a process using Signals

```
architecture SIGN of EXAMPLE is
     signal TRIGGER, RESULT: integer := 0;
     signal signal1: integer :=1;
     signal signal2: integer :=2;
     signal signal3: integer :=3;
begin
     process
     begin
          wait on TRIGGER;
          signal1 <= signal2;
          signal2 <= signal1 + signal3;
          signal3 <= signal2;
          RESULT  <= signal1 + signal2 + signal3;
     end process;
end SIGN;
```

In the first case (using variable), the variables "variable1, variable2 and variable3" are computed sequentially and their values updated instantaneously after the TRIGGER signal arrives.

Next, the RESULT, which is a signal, is computed using the new values of the variables and updated a time delta after TRIGGER arrives. This results in the following values (after a time TRIGGER):
variable1 = 2
variable2 = 5 (=2+3)
variable3= 5.

Since RESULT is a signal it will be computed at the time TRIGGER and updated at the time TRIGGER + Delta.

Its value will be **RESULT=12**.

USP - SCE-703 - 2008-2
Proj. e Implem. de Sistemas Embarcados I
Prof. Fernando Osório

2. *VHDL - Linguagem*

Signal x Variable

Example of a process using Variables

```
architecture VAR of EXAMPLE is
     signal TRIGGER, RESULT: integer := 0;
begin
     process
           variable variable1: integer :=1;
           variable variable2: integer :=2;
           variable variable3: integer :=3;
     begin
           wait on TRIGGER;
           variable1 := variable2;
           variable2 := variable1 + variable3;
           variable3 := variable2;
           RESULT <= variable1 + variable2 + variab
     end process;
end VAR                    RESULT=12
```

Example of a process using Signals

```
architecture SIGN of EXAMPLE is
     signal TRIGGER, RESULT: integer := 0;
     signal signal1: integer :=1;
     signal signal2: integer :=2;
     signal signal3: integer :=3;
begin
     process
     begin
          wait on TRIGGER;
          signal1 <= signal2;
          signal2 <= signal1 + signal3;
          signal3 <= signal2;
          RESULT  <= signal1 + signal2 + signal3;
     end process;
end SIGN;                  RESULT=6
```

On the other hand, in the second example, using signals, the signals will be computed at the time TRIGGER.

All of these signals are computed at the same time, using the old values of signal1, 2 and 3.

All the signals will be updated at Delta time after the TRIGGER has arrived. Thus the signals will have these values:

signal1= 2
signal2= 4 (=1+3)
signal3= 2 and

RESULT=6.

2. *VHDL - Linguagem*

VHDL

Behavioral Modeling: Sequential Statements

VHDL provides means to represent digital circuits at different levels of representation of abstraction, such as the behavioral and structural modeling. Here we will discuss different constructs for describing the behavior of components and circuits in terms of sequential statements.

The basis for sequential modeling is the process construct.

The process construct allows us to model complex digital systems, in particular sequential circuits.

Syntax:

[process_label:] **process** [(sensitivity_list)] [**is**]
[process_declarations]
begin
list of sequential statements such as:
signal assignments
variable assignments
case statement
exit statement
if statement
loop statement
next statement
null statement
procedure call
wait statement
end process [process_label];

2. *VHDL - Linguagem*

VHDL

Behavioral Modeling: Sequential Statements

Syntax:

[process_label:] **process** [(sensitivity_list)] [**is**]
[process_declarations]
begin
list of sequential statements such as:
signal assignments
variable assignments
case statement
exit statement
if statement
loop statement
next statement
null statement
procedure call
wait statement
end process [process_label];

Example of a positive edge-triggered D flip-flop with asynchronous clear input follows.

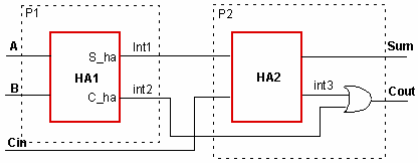
```
library ieee;
use ieee.std_logic_1164.all;

entity DFF_CLEAR is
    port (CLK, CLEAR, D : in std_logic;
          Q : out std_logic);
end DFF_CLEAR;

architecture BEHAV_DFF of DFF_CLEAR is
begin
DFF_PROCESS: process (CLK, CLEAR)
    begin
        if (CLEAR = '1') then
            Q <= '0';
        elsif (CLK'event and CLK = '1') then
            Q <= D;
        end if;
    end process;
end BEHAV_DFF;
```

2. *VHDL - Linguagem*

VHDL

Behavioral Modeling: Sequential Statements

Example: Full-Adder composed of 2 Half-Adders



Full Adder composed of two Half Adders, modeled with two processes P1 and P2.

Example of Sequential Execution: P1 => P2

HA1
S_ha = (A xor B)
C_ha = A.B
HA2
Sum = (A xor B) xor Cin = S_ha xor Cin
Cout = (A xor B).Cin + A.B = S_ha.Cin + C_ha

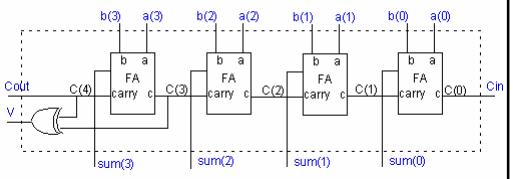
```
library ieee;
use ieee.std_logic_1164.all;

entity FULL_ADDER is
    port (A, B, Cin : in std_logic;
          Sum, Cout : out std_logic);
end FULL_ADDER;

architecture BEHAV_FA of FULL_ADDER is
signal int1, int2, int3: std_logic;
begin
-- Process P1 that defines the first half adder
P1: process (A, B)
    begin
        int1<= A xor B;
        int2<= A and B;
    end process;
-- Process P2 defines the second half adder and the OR
P2: process (int1, int2, Cin)
    begin
        Sum <= int1 xor Cin;
        int3 <= int1 and Cin;
        Cout <= int2 or int3;
    end process;
end BEHAV_FA;
```


2. *VHDL - Linguagem*

VHDL

Hierarchical Design

Example: 4-bit adder consisting of full adder modules

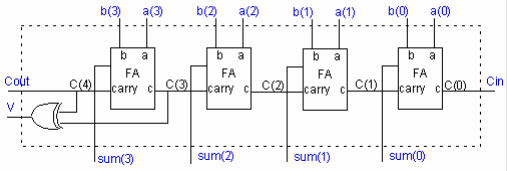



```
-- Example of a four bit adder
library ieee;
use ieee.std_logic_1164.all;
-- definition of a full adder
entity FULLADDER is
    port (a, b, c: in std_logic;
          sum, carry: out std_logic);
end FULLADDER;

architecture fulladder_behav of FULLADDER is
begin
    sum <= (a xor b) xor c ;
    carry <= (a and b) or (c and (a xor b));
end fulladder_behav;
```

```
-- 4-bit adder
library ieee;
use ieee.std_logic_1164.all;

entity FOURBITADD is
    port (a, b: in std_logic_vector(3 downto 0);
          Cin : in std_logic;
          sum: out std_logic_vector (3 downto 0);
          Cout, V: out std_logic);
end FOURBITADD;

architecture fouradder_structure of
                FOURBITADD is
...
end fouradder_structure;
```

2. *VHDL - Linguagem*

```
-- 4-bit adder
library ieee;
use ieee.std_logic_1164.all;

entity FOURBITADD is
    port (a, b: in std_logic_vector(3 downto 0);
          Cin : in std_logic;
          sum: out std_logic_vector (3 downto 0);
          Cout, V: out std_logic);
end FOURBITADD;

architecture fouradder_structure of
                FOURBITADD is
    signal c: std_logic_vector (4 downto 0);
    component FULLADDER
        port (a, b, c: in std_logic;
              sum, carry: out std_logic);
    end component;
begin
        FA0: FULLADDER
            port map (a(0), b(0), Cin, sum(0), c(1));
        FA1: FULLADDER
            port map (a(1), b(1), C(1), sum(1), c(2));
        FA2: FULLADDER
            port map (a(2), b(2), C(2), sum(2), c(3));
        FA3: FULLADDER
            port map (a(3), b(3), C(3), sum(3), c(4));
        V <= c(3) xor c(4);
        Cout <= c(4);
end fouradder_structure;
```




```
-- Example of a four bit adder
library ieee;
use ieee.std_logic_1164.all;
-- definition of a full adder
entity FULLADDER is
    port (a, b, c: in std_logic;
          sum, carry: out std_logic);
end FULLADDER;

architecture fulladder_behav of FULLADDER is
begin
    sum <= (a xor b) xor c ;
     carry <= (a and b) or (c and (a xor b));
end fulladder_behav;
```


2. *VHDL - Linguagem*

VHDL Test Benches

- Testing a design by simulation
- Use a ***test bench*** model
  - An architecture body that includes an instance of the design under test
  - Applies sequences of test values to inputs
  - Monitors values on output signals
    - either using simulator
    - or with a process that verifies correct operation

2. *VHDL - Linguagem*

VHDL

Test Bench Example

```
entity test_bench is
end entity test_bench;

architecture test_reg4 of test_bench is
    signal d0, d1, d2, d3, en, clk, q0, q1, q2, q3 : bit;
begin
    dut : entity work.reg4(behav)
        port map ( d0, d1, d2, d3, en, clk, q0, q1, q2, q3 );
    stimulus : process is
    begin
        d0 <= '1';  d1 <= '1';  d2 <= '1';  d3 <= '1';  wait for 20 ns;
        en <= '0';  clk <= '0';  wait for 20 ns;
        en <= '1';  wait for 20 ns;
        clk <= '1';  wait for 20 ns;
        d0 <= '0';  d1 <= '0';  d2 <= '0';  d3 <= '0';  wait for 20 ns;
        en <= '0';  wait for 20 ns;
        …
        wait;
    end process stimulus;
end architecture test_reg4;
```


2. *VHDL - Linguagem*

VHDL **Test Benches**

Regression Testing

- Test that a refinement of a design is correct
  - that lower-level structural model does the same as a behavioral model
- Test bench includes two instances of design under test
  - behavioral and lower-level structural
  - stimulates both with same inputs
  - compares outputs for equality
- Need to take account of timing differences

2. *VHDL - Linguagem*

VHDL

Regression Test Example

```
architecture regression of test_bench is
    signal d0, d1, d2, d3, en, clk : bit;
    signal q0a, q1a, q2a, q3a, q0b, q1b, q2b, q3b : bit;
begin
    dut_a : entity work.reg4(struct)
        port map ( d0, d1, d2, d3, en, clk, q0a, q1a, q2a, q3a );
    dut_b : entity work.reg4(behav)
        port map ( d0, d1, d2, d3, en, clk, q0b, q1b, q2b, q3b );
    stimulus : process is
    begin
        d0 <= '1';  d1 <= '1';  d2 <= '1';  d3 <= '1';  wait for 20 ns;
        en <= '0';  clk <= '0';  wait for 20 ns;
        en <= '1';  wait for 20 ns;
        clk <= '1';  wait for 20 ns;
        …
        wait;
    end process stimulus;
    ...
```

Cont...

2. *VHDL - Linguagem*

VHDL

Regression Test Example

```
    …
    verify : process is
    begin
        wait for 10 ns;
        assert q0a = q0b and q1a = q1b and q2a = q2b and q3a = q3b
            report "implementations have different outputs"
            severity error;
        wait on d0, d1, d2, d3, en, clk;
    end process verify;
end architecture regression;
```

2. *VHDL - Linguagem*

VHDL

Exemplos

> Ver referências...

Sugestão de Atividade

> Tutorial Interativo da ALDEC sobre VHDL

2. *VHDL - Linguagem*

VHDL *x* Verilog*

VHDL	Verilog
All abstraction levels	All abstraction levels
Complex grammar	Easy language
Describe a system (everything)	Describe a digital system
Lots of data types	Few data types
User-defined package & library	No user-defined packages
Full design parameterization	Simple parameterization
Easier to handle large designs	
Very consistent language. Code written and simulated in one simulator will behave exactly the same in another simulator. E.g. strong typing rules.	**L**ess consistent language. If you don't follow some adhoc methodology for coding styles, you will not get it right.
* Comparison according to: Dr. Gábor Hosszú http://nimrud.eet.bme.hu/cae/ppt/2vhdl/1intro.ppt	It executes differently on different platforms unless you follow some adhoc coding rules.

2. *VHDL - Linguagem*

VHDL *x* Verilog*

- It does seem that **Verilog** is easier for designing at the gate-level, but that people who do higher level simulations express a preference for **VHDL**
- VHDL places constraints on evaluation order that limit the optimizations that can be performed
  - Verilog allows the simulator greater freedom
  - For example, multiple levels of zero-delay gates can be collapsed into a single super-gate evaluation in Verilog
  - VHDL requires preserving the original number of delta cycles of delay in propagating through those levels

VHDL	Verilog
In Europe the VHDL is the most popular language	
Based on Pascal language	Based on C language
Most FPGA design in VHDL	Most ASIC design in Verilog

* Comparison according to: Dr. Gábor Hosszú - http://nimrud.eet.bme.hu/cae/ppt/2vhdl/1intro.ppt

HDL / VHDL
Harware Description Language

HDL - Referências

Aldec
Tutorials: VHDL, Verilog, SystemC
Software : Active-HDL Student Edition (Free)
http://www.aldec.com/Downloads/default.aspx

HDL - Hardware Description Languages
History and Languages
http://en.wikipedia.org/wiki/Hardware_description_language

VHDL: On-Line Material

> Introdução à Linguagem VHDL - Luís Henrique Costa / UFRJ
http://www.gta.ufrj.br/ensino/EEL480/Introducao-VHDL.ppt

> VHDL Quick Start - Peter Ashenden / Univ. of Adelaide
http://www.ashenden.com.au/designers-guide/VHDL-quick-start.pdf

> Introduction to VHDL - M. Balakrishnan / IIT Delhi
http://www.cse.iitd.ernet.in/~mbala/csl316/slides/ (Sec6.ppt)

> VHDL Tutorial - ESE201 / UPENN
http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html

> VHDL Tutorial On-Line
http://www.vhdl-online.de/~vhdl/tutorial/

Continua...

HDL / VHDL
Harware Description Language

HDL - Referências

VHDL: On-Line Material

> VHDL Class
http://web.cecs.pdx.edu/~mperkows/CLASS_VHDL/index.html

> RASSP - DARPA Rapid Prototyping of Application Specific Signal Processors (RASSP)
http://www.eda.org/rassp/

> VHDL Tutorial: Learn by Example - Weijun Zhang
http://esd.cs.ucr.edu/labs/tutorial/

> VHDL Faq
http://www.vhdl.org/comp.lang.vhdl/FAQ1.html

=> Disciplina SCE 0703 - Ver Material Complementar! http://www.icmc.usp.br/~fosorio/

VHDL: Referências

> Digital Fundamental with VHDL - Floyd (Biblioteca)
> Projeto e Prototipação de Sistemas Digitais - Luigi Carro (Biblioteca)
> VHDL Intro - Eduardo Simões

That's all folks!

Universidade de São Paulo - São Carlos,SP
Instituto de Ciências Matemáticas e de Computação

INFORMAÇÕES SOBRE A DISCIPLINA

USP - Universidade de São Paulo - São Carlos, SP
ICMC - Instituto de Ciências Matemáticas e de Computação
SSC - Departamento de Sistemas de Computação

Prof. Fernando Santos OSÓRIO
Web institucional: http://www.icmc.usp.br/ssc/
Página pessoal: http://www.icmc.usp.br/~fosorio/
E-mail: fosorio [at] icmc. usp. br ou fosorio [at] gmail. com

Disciplina de Proj. e Implementação de Sistemas Embarcados I
Ferramentas: Altera - Quartus, NIOS II, Cyclone Dev-Kit
Web disciplina: Http://www.icmc.usp.br/~fosorio/
> Programa, Material de Aulas, Critérios de Avaliação,
> Material de Apoio, Trabalhos Práticos