Sistemas Digitais

Aula 04 - VHDL – Parte I

Apresentação

Nesta aula, teremos uma introdução a uma linguagem de descrição de *hardware*. Já estudamos como funcionam as portas lógicas. A linguagem de descrição de *hardware* é utilizada para o projeto de circuitos lógicos visando a sua síntese, ou seja, a construção de um circuito digital em *hardware*. Como já sabemos, existem os *hardwares* que podemos programar e reprogramar, dentre eles podemos citar os FPGAs (*Field Programmable Gate Arrays*). Para programar o FPGA, usaremos uma linguagem de descrição de hardware chamada de VHDL (*VHSIC Hardware Description Language*, onde VHSIC significa *Very High Speed Integrated Circuit*). Para podermos praticar o que estamos aprendendo na teoria, o VHDL será de grande ajuda. Na realidade, nesta aula e na próxima veremos alguns conceitos básicos de VHDL e durante o curso vocês terão a oportunidade de colocar esse conhecimento em prática.

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula você será capaz de:

- Definir o que é VHDL e como ele é usado.
- Reconhecer alguns componentes básicos de programação de VHDL: estrutura básica (*entity e architecture*), tipos de dados, bibliotecas e operadores.

Ponto de Partida

O VHDL é uma linguagem que foi desenvolvida pelo Departamento de Defesa Americano no início da década de 1980 como uma forma de documentar projetos no programa de circuitos integrados de velocidade muito alta (VHSIC).

Assim, o VHDL se tornou uma das principais linguagens para programação de *hardware* de alto nível. A linguagem foi padronizada pela IEEE (Instituto de Engenheiros Elétricos e Eletrônicos), o que a tornou atraente para engenheiros e criadores de ferramentas e *softwares* de circuitos digitais.

Sabemos que para programar devemos ter uma interface para programação. Utilizaremos um *software*, ou seja, uma interface chamada Quartus. Essa interface foi desenvolvida por um dos grandes fabricantes de FPGAs, a Altera Corporation <www.altera.com>.

Escolhemos essa interface porque os kits de desenvolvimento para as aulas práticas são desse fabricante, Altera.

Quartus da Altera

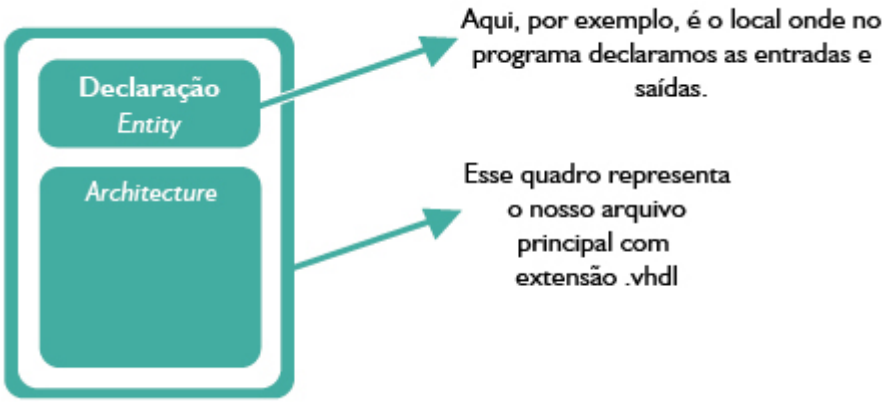
O *software* Quartus WebEdition será passado à vocês em aula presencial. Você pode visitar http://dl.altera.com/13.1/?edition=web (acesso em 25 Jun. 2015) e descobrir mais sobre o *software*.

Estruturas Básicas do VHDL

Podemos dizer que um sistema digital de VHDL consiste no projeto (*design*). Nesse projeto estaremos descrevendo o circuito lógico ou sistema digital. Se pensarmos, por exemplo, em uma porta lógica que já aprendemos, teremos as entradas, as saídas e a função da porta lógica. A função da porta lógica pode ser, por exemplo: AND, OR, NAND, NOR etc...

O nosso programa em VHDL seguirá essa mesma estrutura, entradas, saídas e função a ser executada no programa. A parte do programa em que colocamos, ou, podemos dizer, declaramos as nossas entradas e saídas, é chamado ENTITY (entidade). Cada entidade (*entity*) é modelada pela declaração da sua entidade e da arquitetura. Olhando a Figura 1, veremos que o nosso programa ou projeto conterá a declaração da entidade. Podemos considerar a declaração da entidade como a interface com o mundo externo que define os sinais de entrada e saída. A arquitetura (*architecture*) contém a descrição da entidade, ou seja, a descrição da função do circuito. Em um típico projeto podemos ter várias entidades conectadas para realizar uma função desejada.

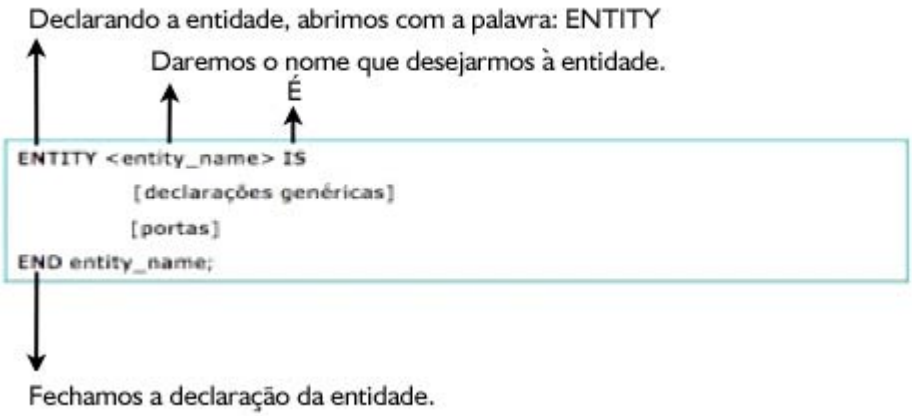
Figura 01 - Estruturas básicas de um projeto VHDL.



ENTITY (Entidade)

Em VHDL, como toda linguagem de programação, seguimos regras para declarar variáveis e funções. Vamos ver como, no nosso caso, declaramos uma ENTITY, uma entidade. Assim, escrevemos ENTITY <entity_name>, onde <entity_name> será o nome que vamos dar para a nossa entidade, seguida da palavra IS, que quer dizer em português: É. Depois, podemos declarar as nossas entradas e saídas das portas e declarações genéricas. As declarações genéricas, veremos alguns exemplos depois.

Figura 02 - A estrutura de declaração da entidade (ENTITY).



Agora uma parte muito importante é aprendermos como vamos declarar as nossas portas de entrada e de saída. Definimos os sinais de entrada e saída, descrevendo os tipos de dados e as portas.

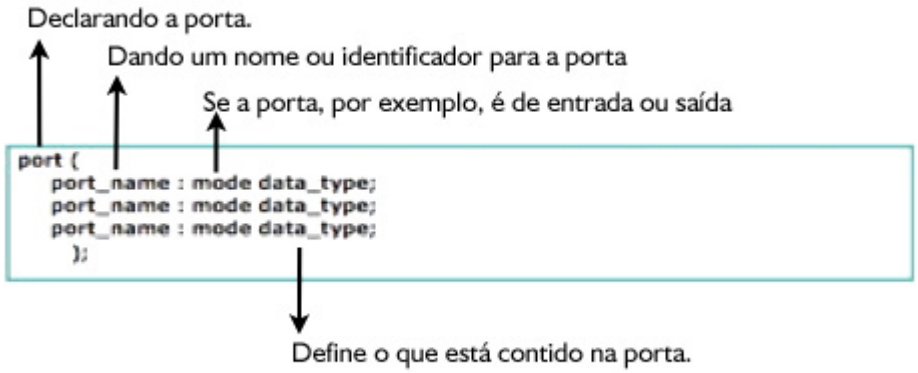
As portas podem ser definidas da seguinte maneira: colocamos o nome "*port*", abre-se parênteses, e, dentro dos parênteses, declaramos (damos um nome para) a porta.

Como se pode ver na figura 3, os itens listados possuem o seguinte significado:?

- **port_name**: é o nome dado à porta ou identificador;
- **mode**: define o tipo, se é de entrada, de saída, buffer ou entrada e saída ao mesmo tempo
- **data_type**: define o que está contido no objeto, qual é o tipo de dado.

As setas na figura 3 tentam definir e explicar melhor o que cada item tem como objetivo.

Figura 03 - A declaração da estrutura da porta



Na Tabela 1 temos os tipos de dados associados à porta, ou seja, o *data_type*.

bit	pode conter valor '0' ou '1'
bit_vector	bit_vector é um vetor de valores em binário. Ex.: bit_vector (0 to 7)
std_logic, std_ulogic	tipo especial de uma biblioteca do VHDL (pode assumir valores além de “0” e “1”).
std_logic_vector, std_ulogic_vector	vetor de valores em tipos especiais (pode assumir 9 diferentes valores)
boolean	pode ter o valor VERDADEIRO ou FALSO
integer	pode conter valores inteiros
real	pode conter valores reais
character	qualquer tipo de caracter
time	para indicar tempo

Tabela 1 - Tipos de dados no VHDL

Na entidade, podemos acrescentar as declarações genéricas e passar informações ao modelo como mostrado a seguir:

Figura 04

```
Generic (
    constante_name : data_type : valor;
    constante_name : data_type : valor;
      );
```

Onde "*constante_name*" é o identificador, "*data_type*" é o tipo do dado e "valor" é o valor passado à constante.

Figura 05 - Exemplo 1 da descrição da entidade

```
-------------------------------------------------------------------------------
-- Descrição da interface do circuito
-------------------------------------------------------------------------------
ENTITY ckt1_exemplo IS
        -- declarações genéricas...
        Generic (constant aux1, aux2  : time    := 5ns;
                          tap         : integer := 10;
                          cmd         : string  := "up";
                );

        -- declaração das portas...
        Port (
              in1 : in std_logic;
              in2 : in std_logic;
              out1 : out std_logic;
              );
END ckt1_exemplo;
```

Se observarmos, temos como declarações genéricas constantes aux1 e aux2 definidas com *data_type* (tipo de dado) de *time* e recebendo o valor de 5ns. A variável tap definida como *integer* (inteiro) e recebendo o valor de 10 e por último cmd recebendo o *data_type* de string e o valor "*up*". Já nas declarações de portas PORT, temos *in1* e *in2* sinais de entrada (**in**) com o tipo de dado chamado *std_logic* (mais tarde estudaremos melhor esse tipo, mas por enquanto sabemos que *std_logic* inclui "0" e "1") e *out1* é um sinal de saída (**out**) com o tipo de dado *std_logic*.

Vídeo 02 - VHDL

Atividade 01

1. Pesquise sobre outra linguagem de descrição de *hardware*, chamada Verilog, compare com VHDL.

Exemplo ENTITY (Entidade)

Vamos colocar mais dois exemplos de declaração de entidade para um circuito A e para um circuito B.

Para um circuito A, chamamos de Multiplexador de 4 entradas. Temos as entradas (*in* – **a, b, c, d**) e a saída (*out* – **f**) com vetores de tamanho 8. SEL será uma entrada seletora. Na aula 06 de circuitos lógicos combinacionais, você terá um entendimento maior do funcionamento desse circuito. Aqui, basta sabermos que é um circuito no qual temos várias entradas e a partir de uma entrada seletora, pode-se selecionar uma dessas entradas para ser colocada na saída. Todos os sinais têm o tipo de dado *std_logic_vector*.

```
1 entity mux is
2    port (a,b,c,d: in std_logic_vector(7 downto 0);
3       SEL: in std_logic_vector(1 downto 0);
4       f: out std_logic_vector(7 downto 0));
```

Para um circuito B teremos os sinais **d, clk, s, e r** como entrada (*-in*) recebendo o tipo de dado *std_logic* e os sinais de saída **q** e **qnot** recebendo os dados de saída e também *std_logic*.

```
entity circuitoB is
   port (d,clk,s,r: in std_logic;
      q, qnot: out std_logic);
   end circuitoB;
```

Atividade 02

1. Como vimos, a ENTITY, ou entidade, representa a interface de entrada e saída do nosso sistema. Podemos ver a entidade como uma caixa preta como a da Figura 6. Assim, podemos dizer, nesse caso, que as entradas seriam, in1 e in2 e a saída seria out1. Não importa o que está dentro da caixa preta por enquanto.

   Se pensarmos em um circuito simples como somador de dois números, você consegue definir as suas variáveis de entrada e saída

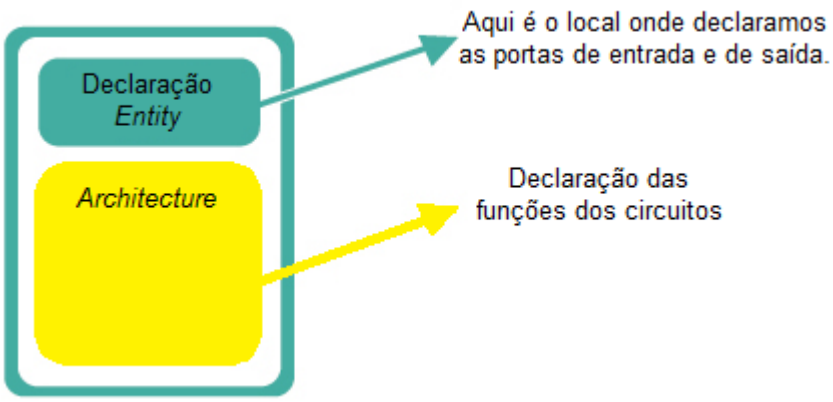
Figura 06 - Caixa preta

2. Para os exemplos dados anteriormente, do multiplexador (Mux) e do circuito B, descreva quais são as variáveis de entrada e saída.

Architecture (Arquitetura)

Agora vamos falar da *architecture*, ou seja, da arquitetura. Vamos observar a Figura 7. O que nos interessa agora é a segunda parte do nosso arquivo de VHDL, que está em AMARELO.

Figura 07 - Estrutura básica de projeto, com ênfase para arquitetura em amarelo



A arquitetura descreve como a função é implementada, ou seja, descreve o circuito propriamente dito, o que está dentro da caixa preta (como a que vimos na atividade anterior, Figura 6). Assim podemos dizer sobre a arquitetura:

- É a parte onde descrevemos as funções no modelo.
- Deve estar sempre associada a uma ENTITY.
- Podem haver várias ARCHITECTURES para uma mesma ENTITY.
- Os procedimentos descritos na arquitetura ocorrem concorrentemente, ou seja, ao mesmo tempo.

Vamos descrever como devemos apresentar a estrutura de uma arquitetura:

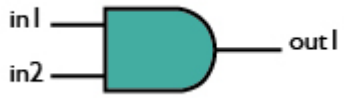
Figura 08

```
ARCHITECTURE <architecture_name> OF <entity_name> IS
        [declaração de sinais]
        [declaração de constantes]
        [definição de funções]
        [definição de procedimentos] .....
BEGIN
        [blocos de processos]
        [declarações - concorrentes]
        [instanciação de componenets] .....
       .......
END <architecture_name>;
```

Como você pode perceber na arquitetura, podemos ter as declarações de sinais, constantes, funções e procedimentos. Depois da cláusula BEGIN, escrevemos a função propriamente dita, utilizando blocos de processos e declarações, ou seja, as instruções.

Vamos ver agora um exemplo de um código VHDL, tanto da declaração da entidade, como também da arquitetura. Primeiro definimos a entidade, que nomeamos como ExemploAND. Na entidade definimos as entradas (in1, in2) e a saída (out1).

Figura 09 - Uma porta AND, com suas entradas, in1, in2 e sua saída out1



Depois, na arquitetura, descrevemos o que a função ou circuito executa, neste caso é uma porta AND. O código está com comentários inseridos que estão indicados por "--" no começo da frase, ou seja, em VHDL os comentários são precedidos de "--". No nosso exemplo, os comentários estão destacados em azul.

```
1   entity ExemploAND is
2   -- Entidade <nome da entidade> é (is)
3   port (in1, in2: in std_logic;
4   -- Declaramos as portas de entrada e saída
5
6   out1: out std_logic);
7   end ExemploAND;                 -- Finalizamos a entidade
8   architecture exemplo2 of ExemploAND is
9
10  -- Arquitetura <nome da arquitetura> pertencente (of) à entidade <nome da entidade> é (is)
11
12
13  begin                           -- Início da descrição de arquitetura, com termo "begin"
14        out1 <= in1 and in2;
15  end exemplo2;                   -- Finalizamos a arquitetura
```

Atividade 03

1. Quais são as duas principais estruturas de projeto do VHDL?
2. Defina as diferenças e funções destas duas estruturas.

3. O exemplo apresentado anteriormente mostrava a descrição VHDL de uma porta AND. Agora escreva o código para a porta OR de duas entradas, sabendo que em VHDL a porta or é definida como OR.

Library (Biblioteca)

Dentre as estruturas de VHDL, é importante não deixarmos de citar as bibliotecas. Como em qualquer linguagem, podemos fazer as chamadas a bibliotecas (*library*, em inglês) para utilizarmos, em nossos programas, variáveis, sinais, operações já prontas. A biblioteca pode ser considerada o lugar onde guardamos informações do projeto.

O pacote (*package*) de VHDL é um arquivo ou módulo que contém as declarações ou objetos que desejamos utilizar. Aqui, vamos nos ater a uma biblioteca mais utilizada em VHDL para que possamos utilizar o tipo de dado *std_logic*. Isso é possível se utilizarmos a biblioteca (*library*) da IEEE e a palavra use para chamar o pacote. Assim, antes de definir a entidade devemos realizar as seguintes chamadas:

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

A extensão **.all** indica que estamos utilizando todos os pacotes. Existem vários outros pacotes que também podem ser utilizados. Conforme a necessidade, eles serão introduzidos.

Agora vamos ver o exemplo de um programa completo que descreve uma porta AND.

```
1  library ieee;                    -- Declarando as bibliotecas e pacotes a serem utilizados.
2  use ieee.std_logic_1164.all;
3
4  entity ExemploAND is             -- Entidade <nome da entidade> é (is)
5  port (in1, in2: in std_logic;    -- Declaramos as portas de entrada e saída
6  out1: out std_logic);
7  end ExemploAND;                  -- Finalizamos a entidade
8  architecture exemplo2 of ExemploAND is
9
10 -- Arquitetura <nome da arquitetura> pertencente (of) à entidade <nome da entidade> é (is)
11
12 begin                           -- Início da descrição de arquitetura, com termo "begin"
13       out1 <= in1 and in2;
14 end exemplo2;                   -- Finalizamos a arquitetura
```

Depois de um breve introdução sobre as estruturas básicas, vamos conhecer algumas regras básicas da linguagem:

- VHDL não é sensitivo ao padrão da letra, maúsculo ou minúsculo.
  Ex: **Out1 <= A and B;** é a mesma coisa que **out1 <= a AND b;**

- Não é sensitivo a espaço em branco.
  Ex: **Saida <= A or B;** é a mesma coisa que **Saida <= A or B;**

- Comentários são feitos com "--"
  Ex: **--Isso é um comentário.**

- Terminação dos comandos com ponto e virgula (;)

- Os parênteses não tem o uso obrigatório;

- Palavras reservadas não devem ser utilizadas nos códigos para definir variáveis e sinais
  Ex: **access, exit, after, all, in, open, loop,....**

- Identificadores (nomes de variáveis, sinais, portas,...)

    - Devemos usar nomes que indiquem a função da variável ou do sinal.

    - Podem ser letras (A,a – Z,z), dígitos (0-9) e *underline* (“_”).

- Devem começar com letra alfabética.
- Não podem terminar com *underline*.

Atividade 04

1. Qual o objetivo de utilizarmos as bibliotecas?
2. Qual a biblioteca necessária quando utilizamos o tipo de dado *std_logic* e como declaramos ela no código VHDL?
3. Que caractere é utilizado para inserir comentários em uma linha de código VHDL?

Tipos de Dados

Os dados em VHDL são divididos em 4 classes:

- Escalares: representam um único valor;
- Compostos: representam uma coleção de valores;
- Acessos: são similares a ponteiros;
- Arquivo: referencia objetos que têm uma sequência de valores.

O nosso interesse estará, nesse momento, nas classes dos tipos escalares e compostos.

Os tipos escalares são:

- Bit
- Boolean (Booleano)
- Integer (Inteiro)
- Real
- Physical (Físico)
- STD_LOGIC

Para o tipo composto temos, por exemplo, o tipo vetor (array), que é uma coleção de elementos do mesmo tipo.

Ex: Podemos representar um vetor (um conjunto de valores) da seguinte forma:

VetorA = “0001111” é um vetor de 7 elementos, temos três “0”s e quatro “1”s.

VetorB = “0001” é um vetor de 4 elementos, temos três “0”s e um elemento “1”.

Quando no nosso programa em VHDL precisamos declarar um tipo vetor, fazemos da seguinte maneira:

Exemplos:

i) std_logic_vetor (6 downto 0). Entre parênteses determinamos o tamanho do vetor, no caso com 7 elementos. O elemento mais significativo (MSB) é o elemento mais à esquerda; e o menos significativo (LSB), o último elemento à direita. Lembre-se da sua primeira aula, na qual abordamos o assunto sistema de numeração. O vetor definido terá os elementos: 6, 5, 4, 3, 2, 1 e 0.

ii) std_logic_vetor (3 downto 0). Entre parênteses, determinamos o tamanho do vetor, no caso com 4 elementos. O elemento mais significativo é o elemento mais à esquerda e o menos significativo o último elemento à direita.

iii) std_logic_vector (0 to 6). Entre parênteses determinamos o tamanho do vetor, no caso com 7 elementos. O elemento menos significativo é o elemento mais à esquerda e o mais significativo o último elemento à direita, ou seja, teremos um vetor com os elementos: 0, 1, 2, 3, 4, 5 e 6.

Nesses 3 casos, notamos que o tipo de dado é o std_logic, mas poderíamos utilizar outros tipos dados, como, por exemplo, bit

Vídeo 03 - VHDL

Atividade 05

1. Quais são as 4 classes de tipos de dados?
2. Quais são os tipos escalares?
3. Escreva um vetor binário (de 0s e 1s) que contenha 8 elementos?

Operadores

Existem vários tipos de operadores como: lógicos, relacionais, deslocamento, adição, multiplicação e outros. Vamos nos deter a conhecer melhor os operadores lógicos, relacionais, de adição e multiplicação.

Operadores Lógicos

São os operadores das portas lógicas que já aprendemos: **and, nand, or, nor,** e **not**.

Operadores Relacionais

Olhando o Quadro 1, podemos deduzir a função desses operadores.

Operador	Nome	Explicação
=	Equivalente	Esse valor é equivalente a outro valor?
/=	Não equivalente	Esse valor não é equivalente a outro valor?
<	Menor que	Esse valor é menor que outro valor?
<=	Menor que ou igual	Esse valor é menor ou igual a outro valor?
>	Maior que	Esse valor é maior que outro valor?
>=	Maior que ou igual	Esse valor é maior ou igual a outro valor?

Quadro 1 - Operadores relacionais

Outros Operadores

No Quadro 2 temos mais alguns operadores bem comuns, que veremos em projetos durante o curso. São os operadores aritméticos. Ainda existem outros tipos de operadores, mas eles extrapolam o escopo da nossa disciplina.

Nome	Operador	Explicação
Adição	+	Adição de valores
Subtração	-	Subtração de valores
Multiplicação	*	Multiplicação de valores
Divisão	/	Divisão de valores

Quadro 2 - Operadores aritméticos

Vídeo 04 - Exemplos VHDL

Atividade 06

1. Cite dois tipos de operadores lógicos.
2. Cite dois tipos de operadores relacionais.

Roteiro Prático da Aula 04

Apresentação

Nesta aula você começará a desenvolver atividades práticas com circuitos digitais. Aqui será apresentado o software Quartus II para realizar a construção e a simulação de circuitos digitais, para isto, são necessários conceitos de construção de circuitos lógicos, como também sobre a linguagem VHDL. Nesta primeira aula prática você vai aprender a utilizar o software Quartus II e o simulador do mesmo. Esta aula é baseada na versão 13.0 do Quartus II.

Video 05 - Apresentação

Objetivos

Ao final das atividades previstas para esta aula, você será capaz de:

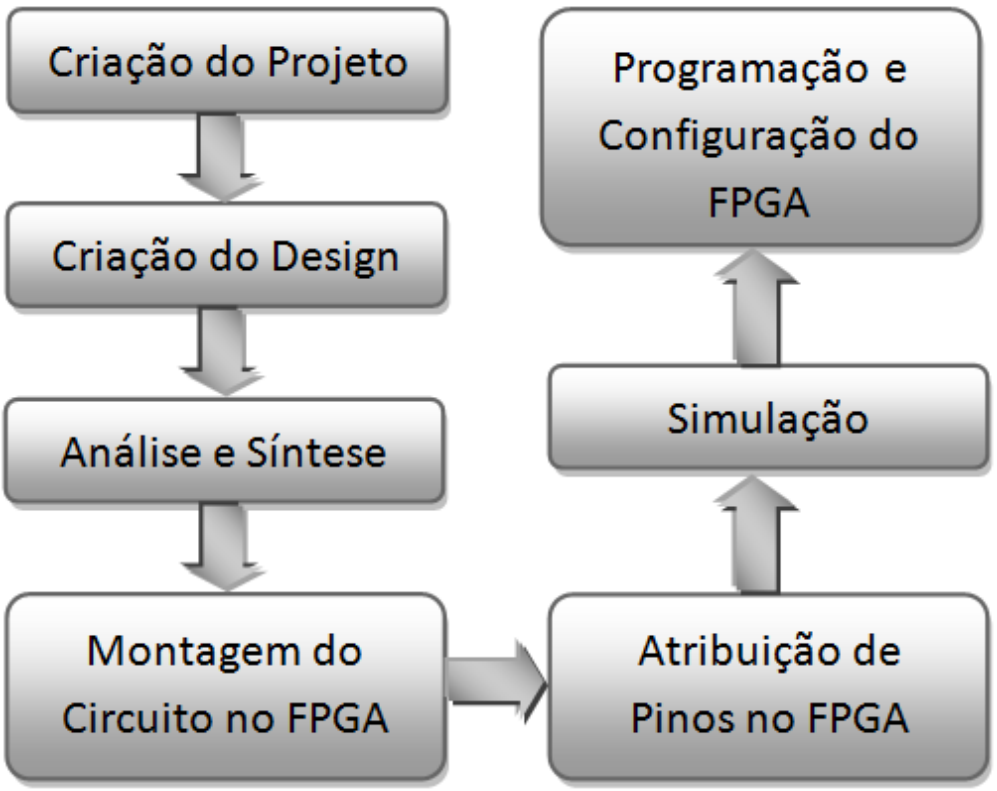
- Construir circuitos digitais no software Quartus II;

- Conhecer as etapas para a construção de circuitos digitais utilizando diagrama esquemático ou linguagem VHDL;
- Simular circuitos digitais no simulador do Quartus II;
- Comparar valores de entradas e saídas através de simulação de circuitos.

Conhecendo o Quartus II

O software Quartus II é utilizado para descrever circuitos digitais, essa descrição pode ser feita de várias formas, seja por linguagem VHDL ou por diagrama esquemático. Este software permite também a gravação em chip FPGA da Altera. Abaixo segue um fluxograma de algumas etapas permitidas pelo software.

Figura 10 - Fluxograma de Prototipagem de Circuitos Digitais



Observe na figura anterior que as últimas etapas do fluxograma dependem do FPGA, que deve ser especificado na etapa de criação do projeto, no entanto, algumas dessas etapas podem ser omitidas de acordo com as necessidades do projetista. Nesta aula não vamos programar o circuito em um chip FPGA, mas vamos utilizar o EP3C16F484C6 da família Cyclone III por definição de projeto.

Caso queira saber mais sobre o software Quartus ou até mesmo baixar seu instalador utilize o endereço abaixo. Para executar as aplicações desta disciplina, baixe os arquivos “Quartus II Software” versão 13.0 ou 13.1, “ModelSim-Altera Edition” e os Devices “Cyclone III, Cyclone IV device support” em uma única pasta e instale o Quartus II e ModelSim. É necessário criar um usuário para baixar estes arquivos.

http://dl.altera.com/13.1/?edition=web (acesso em 01 Jul. 2015)

Construindo o Circuito Digital

Agora que você já sabe o fluxo de construção de circuitos digitais, vamos começar a colocar em prática o que aprendemos nas aulas anteriores e construir nosso primeiro circuito no Quartus. Veja o vídeo a seguir e aprenda como criar um projeto no Quartus II.

Video 06

Compilando e Simulando o Circuito Digital

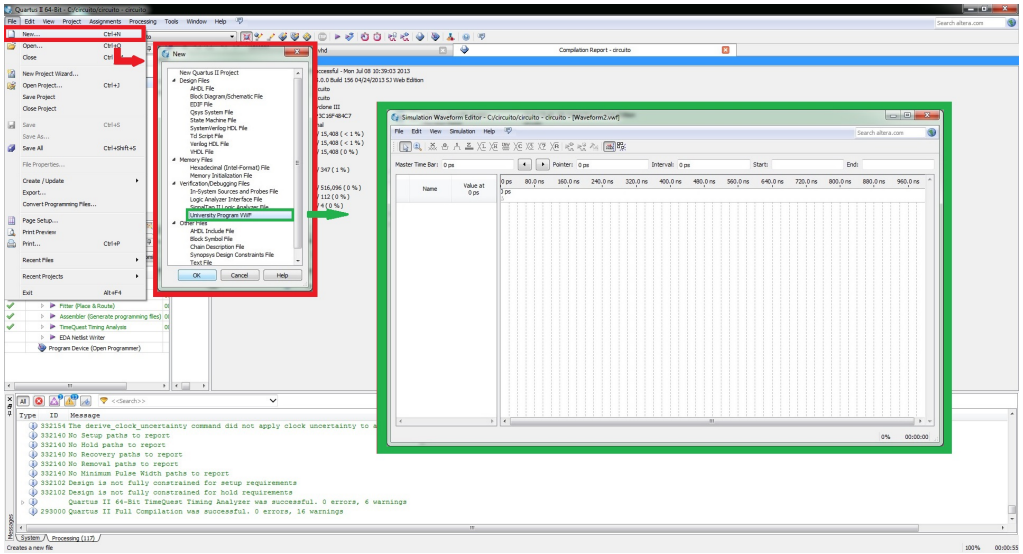
Após criar o projeto, é necessário digitar todo o código e simulá-lo para verificar a sua corretude. O vídeo a seguir mostra como compilar o código e simular utilizando o Simulation Waveform Editor. Saiba que existem vários vários outros softwares que realizam a simulação de circuitos digitais, como EWB, Multisim, etc.

Video 07

Para facilitar seu entendimento, mostraremos a seguir o passo a passo, já comentado no vídeo anterior, necessário para simular um código no Simulation Waveform Editor.

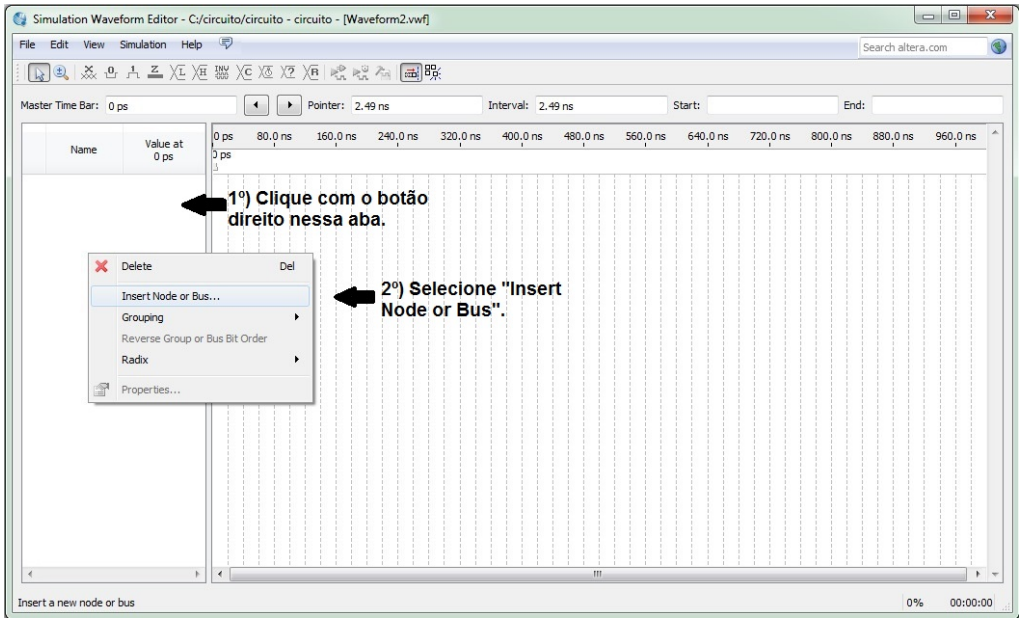
Passo: Depois que o projeto já está criado, clique em cima do projeto e depois: **File -> New... -> University Program VWF**.

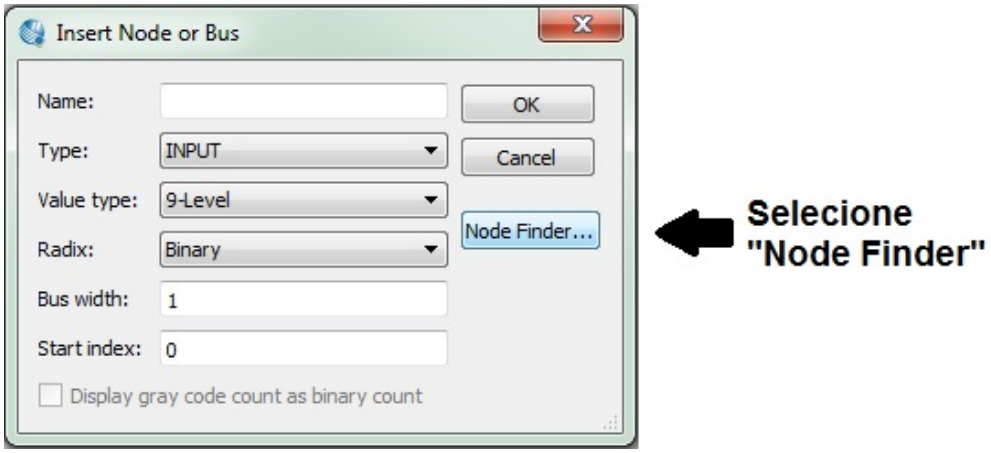
Figura 11 - Criando um arquivo de Simulação

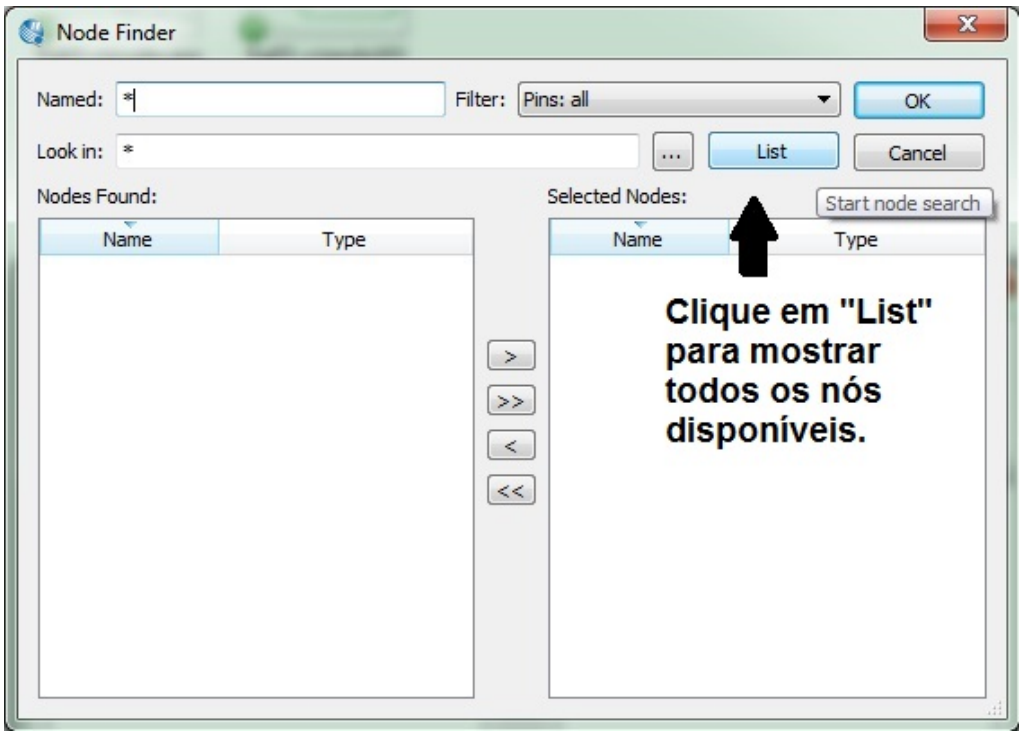


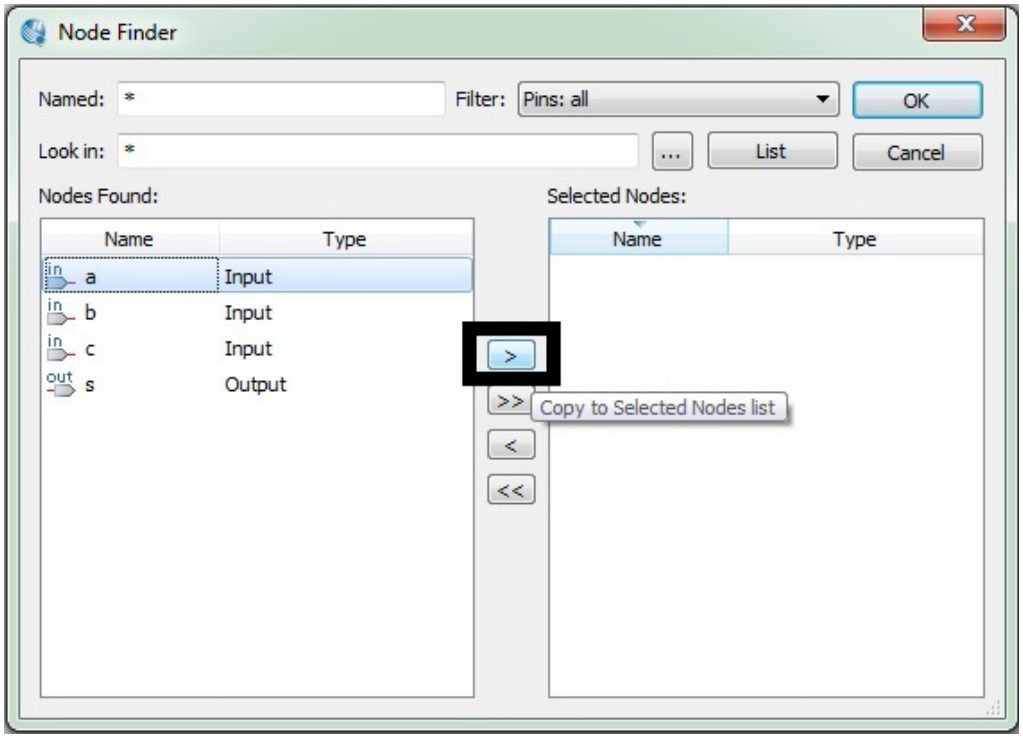
Nesse módulo serão adicionados todos os nós de entrada e saída do circuito para realizar os testes. Para cada nó é definido um tipo de forma de onda ou sinal, e serão nesses tipos de sinais que iremos trabalhar. Os sinais são definidos no menu **“Edit”** na aba **“Value”**.

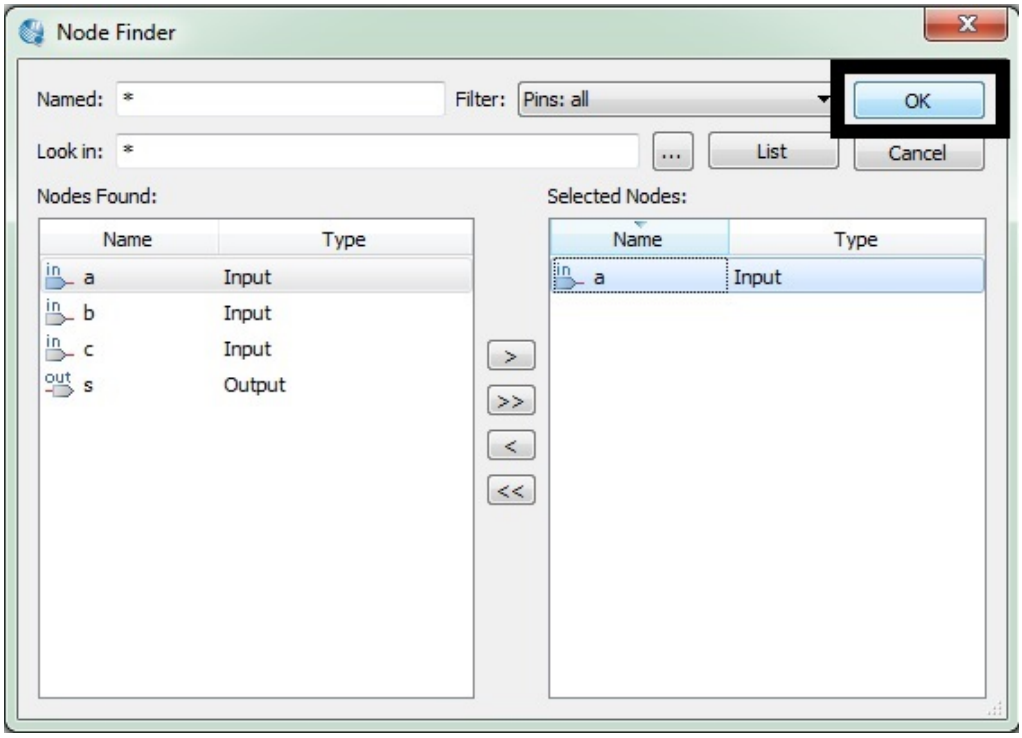
Veja como definir o sinal do nó nas Imagens abaixo.

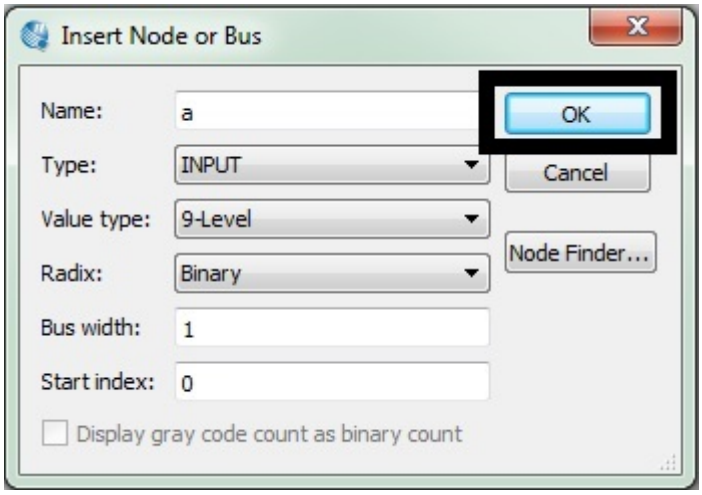


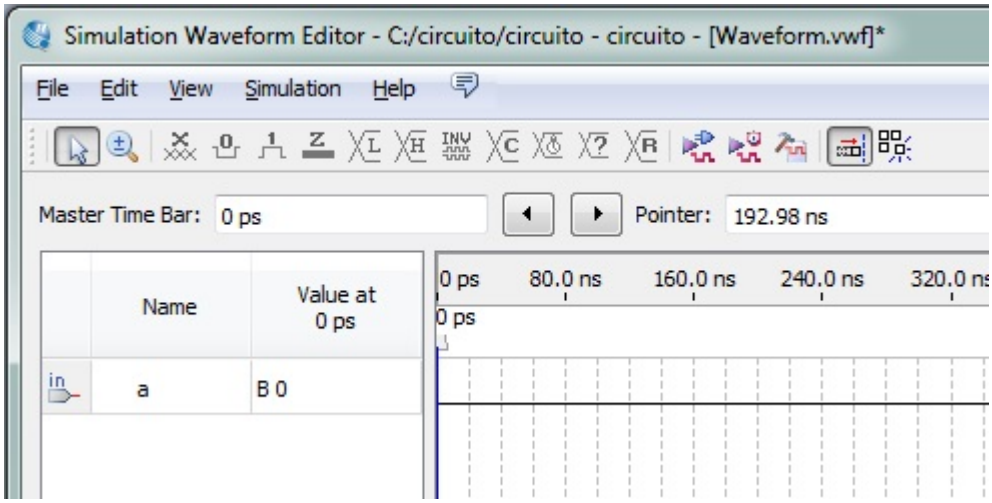



Node Finder
Named: *
Filter: Pins: all
OK
Look in: *
...
List
Cancel
Nodes Found:
Selected Nodes:
Start node search
Name
Type
Name
Type
>
>>
<
<<
Clique em "List" para mostrar todos os nós disponíveis.


Node Finder
Named: *
Filter: Pins: all
OK
Look in: *
...
List
Cancel
Nodes Found:
Selected Nodes:
Name
Type
a
Input
b
Input
c
Input
s
Output
Name
Type
>
>>
<
<<
Copy to Selected Nodes list







Agora só falta rodar a simulação, utilizando o passo abaixo:

- Passo: Atribua uma onda (valores de entrada no decorrer do tempo) para cada variável de entrada. Para cada onda, clique em **Edit -> Value -> Overwrite Clock** e atribua um período.

Dica

É interessante simular o circuito com todas as possibilidades de valores das entradas. Uma das maneiras de realiza-lá é atribuindo a uma nova entrada o dobro do período atribuído a uma entrada anterior.

Exemplo: Um circuito possui três entradas A, B e C. Se atribuir período de A como 40ns, é interessante atribuir B com 80ns (dobro de 40ns) e C como 160ns (dobro de 80ns).

- Passo: Clique em **Simulation -> Run Functional Simulation**.

Veja o resultado da nossa simulação:

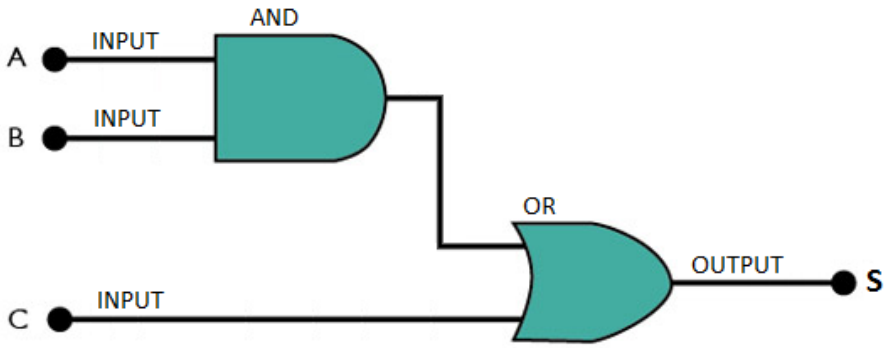
Figura 12 - Exemplo de resultado da uma possível simulação do circuito

Atividade 07

Atividade Prática

1. Crie um novo projeto no Quartus de nome “circuito” e defina o EP3C16F484C6 como FPGA.

2. Utilize o código VHDL dado para construir o circuito do projeto a seguir.




```
   library IEEE;
use IEEE.std_logic_1164.all;

entity circuito is
    PORT( a, b, c: IN std_logic;
    s: OUT std_logic);
end circuito;

architecture estrutura of circuito is
begin
    s <= (a and b) or c;
end estrutura;

```

3. Compile o projeto.
4. Simule o circuito utilizando o Simulation Waveform Editor e todas as possíveis combinações das entradas.
5. Monte a tabela verdade para $A.B + C$ e compare com os valores do exercício 4. Todas as combinações são iguais?

Conclusão

Chegamos ao fim da nossa primeira aula sobre VHDL. Com ela você já é capaz de entender partes de um código simples em VHDL e também pode elaborar algumas construções. A proxíma aula será prática e teremos exercícios para criarmos confiança em escrever algumas funções básicas como essas, que foram apresentadas nos exemplos. Você ainda precisa de mais conhecimento, mas não fique ansioso, na próxima aula veremos a parte de instruções que é muito importante para poder descrever as funções que desejamos que seja realizado no *hardware*. Até lá!

Leitura Complementar

- www.dcc.ufrj.br/~gabriel/circlog/vhdl.pdf. Acesso em: 10 ago. 2012.
- DÁMORE, Roberto. **VHDL Descrição e Síntese de Circuitos Digitais**. Porto Alegre: LTC, 2005.

Resumo

Nessa aula, estudamos noções básicas da linguagem de descrição de hardware, VHDL. Vimos as estruturas básicas, entity e architecture, os tipos de dados e alguns operadores.

No roteiro prático você teve seu primeiro contato com o software Quartus II. Você conheceu como este software realiza a construção e a simulação de circuitos digitais e aprendeu também a utilizar um circuito exemplo para a simulação através do aplicativo de simulação Quartus.

Autoavaliação

1. Das opções abaixo, quais são as duas estruturas principais de projeto em VHDL:

    a. Entity

    b. Architecture

    c. Loop

    d. Integer

2. O que acrônimo VHDL quer dizer?

    a. Very Hardware Description Language

    b. Very High Description Language

c. *Very High Dinamic Language*

d. *VHSIC (Very High Speed Integrated Circuit) Hardware Description Language*

3. São tipos de dados em VHDL:

a. bit, std_logic, std_logic_vector

b. bit, integer, for

c. std_logic, std_logic_vector, entity

d. architecture, integer, std_logic

4. Analisando a entidade definida abaixo, escolha a resposta correta:

```
1 Entity circuitoB is
2 port (d,clk,s,r: in std_logic;
3    q, qnot: out std_logic);
4 end circuitoB;
```

a. **d, clk, s** e **r** são sinais de entrada e **q** e **qnot** são sinais de saída definidas com tipo de dado *std_logic* na arquitetura.

b. **d** e **clk**, são sinais de saída e **q** e **qnot** são sinais de entrada definidas com tipo de dado bit na entidade.

c. **d, clk, s** e **r** são sinais de entrada e **q** e **qnot** são sinais de saída definidas com tipo de dado *std_logic* na entidade.

d. **d, clk, s** e **r** são sinais de entrada e **q** e **qnot** são sinais de saída definidas com tipo de dado *std_logic* na arquitetura.

5. Qual das alternativas abaixo é verdadeira:

a. A entidade tem a sua analogia a um esquemático, descrevendo as funções do modelo, do circuito, sempre está associado a uma ENTITY.

b. A arquitetura tem a sua analogia a um esquemático, descrevendo as funções do modelo, do circuito, sempre está associado a uma ENTITY.

c. A entidade descreve a relação que existe com as arquiteturas.

d. Os procedimentos são sempre sequenciais.

6. O sinal A é definido com um vetor “0001011” que têm:

    a. 7 bits

    b. 6 bits

    c. 8 bits

    d. 2 bits

Referências

ASHENDEN, Peter J. **EDA CONSULTANT**. Disponível em: <www.ashenden.com.au>. Acesso em: 10 ago. 2012.

CASILIO, Leonardo; SARAIVA, Ivan. **Semântica de VHDL**. 2003. aula 02.

GUERREIRO, Ana M G. **Aulas de Circuitos Digitais**. Natal: Universidade Federal do Rio Grande do Norte, [2011].

MEALY, Bryan. **The Low-Carb VHDL Tutorial**. 2004. Disponível em: 10 ago. 2012.

TOCCI, Ronald; NEAL, S. Widner; GREGORY, L. Moss. **Sistemas Digitais:** Princípios e aplicações. 10. ed. São Paulo: Prentice Hall, [2007].

UNIVERDIDADE FEDERAL DE ITAJUBÁ. **Tutorial de VHDL**: grupo de microeletrônica. Itajubá, MG, [2012?]

VAN DER SPIEGEL, Jan. **VHDL Tutorial**. [2001]. Disponível em: <http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html#_Toc526061341>. Acesso em: 10 ago. 2012.

COSTA, Cesar da. **Projetos de Circuitos Digitais com FPGA**. Editora Érica, 2009.

PEDRONI, Volnei. **Eletrônica Digital moderna e VHDL**. Rio de Janeiro: Elsevier, 2010.