Introdução ao VHDL
(Versão 1.0)

Felipe Valencia de Almeida

2019

Sumário

Introdução

Essa apostila foi feita com o propósito de auxiliar os alunos das disciplinas de Sistemas Digitais e de Laboratório Digital. Não é o meu objetivo aqui substituir os livros já existentes, como por exemplo o *VHDL - Descrição e Síntese de Circuitos Digitais* do Roberto D'amore ou o Free Range VHDL, mas sim, fornecer aos alunos um material que seja simples, não contendo todas os comandos e sintaxe do VHDL, e conciso.

Desde 2015 quando fiz a disciplina de Sistemas Digitais 2 tenho acompanhado a dificuldade dos alunos (e minha na época) em aprender o VHDL. Parte desta dificuldade está no fato que o VHDL não é uma linguagem de programação, erro de relação muito comum cometido pelos alunos, mas sim uma linguagem de descrição de hardware. Por isso, uma série de características e atributos de linguagens de programação antes estudadas como C, C++, Python dentre outras não são válidas aqui.

Desta forma, optei por redigir essa apostila com base em minha experiência de 4 anos como aluno, sendo 2 deste anos como monitor das disciplinas de Sistemas Digitais e Laboratório Digital. Espero que você leitor possa aproveitar esse material para diminuir uma parcela considerável do seu sofrimento com o VHDL. Boa leitura!

PS: Agradecimentos à tia Fatima

1 O que é o VHDL?

VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) é uma **linguagem de descrição de hardware**. Originalmente feita pelo Departamento de Defesa dos Estados Unidos, com o propósito de documentação dos circuitos digitais, ela foi posteriormente padronizada pelo IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos).

Por se tratar de uma linguagem de descrição de hardware, o VHDL é **concorrente**, ao invés de **sequencial**, como são as linguagens de programação. Tente aqui se lembrar de quando você fazia os seus códigos em uma linguagem de programação qualquer. O que acontecia quando o código ficava preso em um loop infinito? Em geral ele estava errado, pois um código feito em uma linguagem de programação costuma ter começo, meio e fim, onde suas linhas são executadas **sequencialmente**, sendo finalizado após a execução de todas as linhas.

O mesmo não ocorre em uma linguagem de descrição de hardware. Aqui o código é "infinito", ou seja, sua execução nunca é finalizada. Para aqueles que já tiveram alguma experiência com programação em Arduino, vocês devem se lembrar que nele a função main é um loop. Este conceito deve ser muito intuitivo, mas se mesmo assim ele não é para você, vamos pensar em um exemplo na prática. Imagine que a sua geladeira desliga sozinha. Obviamente você vai achar que ela está com algum problema, porque a geladeira deve sempre estar ligada enquanto houver energia. Perceba aqui que o comportamento de um hardware então é sempre funcionar, por isso dizemos que o seu funcionamento é "infinito".

Além disso, como já foi dito, o VHDL é concorrente, pois todas as suas linhas de código são executadas aos mesmo tempo. Novamente pensando em um circuito físico, não faz sentido dizer por exemplo que em um fio passa corrente antes do outro, já que por todos os fios pode passar corrente ao mesmo tempo. A figura 1 ilustra dois exemplos de execução concorrente. Perceba que o primeiro caso segue a ordem que deve ser mais comum para você, onde primeiro ocorre a declaração do valor das variáveis e depois sua soma. Já o segundo caso é mais estranho, e não compila em grande parte das linguagens de programação, mas está correto em VHDL, devido ao caráter concorrente.

Figura 1: Exemplo de execução concorrente

Mas você pode estar se perguntando qual a vantagem de se descrever um circuito em VHDL ao invés de fazer sua montagem física, utilizando circuitos integrados, por exemplo. Dentre as vantagens, pode-se citar a facilidade em adaptar componentes e copiar circuitos já feitos, não ter limitação de componentes comerciais, não mexer com fios, dentre outras.

2 Estrutura de um Arquivo VHDL

Um arquivo VHDL é dividido em três partes, são elas o pacote, a entidade e a arquitetura. A figura 2 apresenta a estrutura de um arquivo VHDL, consulte-a enquanto estiver lendo as subseções seguintes para entende-las melhor. A seguir é definida cada uma de suas partes.

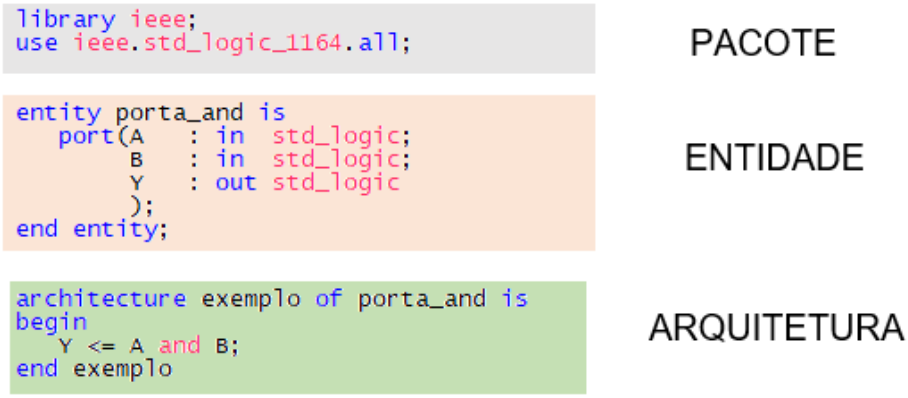


Figura 2: Estrutura de um circuito digital descrito em VHDL

2.1 Pacote

Os pacotes (*library/package*) são conjuntos de informações que você inclui no seu arquivo VHDL. É possível aqui fazer uma analogia perfeita com o *include* do C. Com eles é possível importar funções, tipos de dados, constantes, etc. Um dos pacotes mais importantes é o *ieee.std_logic_1164.all*, da biblioteca *ieee*. Ele disponibiliza os tipos *std_logic* e *std_logic_vector*, que serão descritos futuramente.

2.2 Entidade

A entidade é uma declaração do tipo caixa preta, onde é apenas definido as entradas e as saídas, que são as portas do seu circuito. Perceba que na figura 2 temos duas entradas A e B e uma saída Y. Você pode definir isso a partir do identificador da porta. Ao todo existem quatro identificadores para as portas, são eles:

- In: Porta de entrada. Pode apenas ser **lida** pelo programa
- Out: Porta de saída. Pode apenas ser **escrita** pelo programa
- Inout: Porta de entrada ou de saída. Este tipo de porta é bidirecional, sendo utilizada por exemplo em barramentos de memórias. Ela é apresentada aqui apenas por curiosidade, não sendo utilizada nas disciplinas.

- Buffer: Porta de saída com buffer. Está aqui é um tipo especial de porta de saída que pode ser lida, pois ela possui um buffer acoplado.

A entidade sempre necessita de um nome, que por boa prática deve refletir o circuito descrito. No caso da figura 2 o nome da entidade é *porta_and*, pois o circuito em questão descreve uma porta AND. Cuidado para não utilizar espaço nem caracteres especiais aqui.

Além das portas, outra informação que pode aparecer na declaração da entidade são as constantes do circuito, aqui denominadas de *generic*. É possível fazer uma analogia entre o *generic* do VHDL e o *define* da linguagem C, onde ambos tem a função da colocar um rótulo em determinado valor. A figura 3 apresenta um caso de uma entidade com *generic*. No caso em questão, sempre que for utilizado o *size*, ele é substituído pelo valor 8, como ocorre na própria entidade.

```
entity reg is
    generic (size         : integer := 8);
    port (
        clock          : in  std_logic;
        clear          : in  std_logic;
        enable         : in  std_logic;
        D              : in  std_logic_vector(size-1 downto 0);
        Q              : out std_logic_vector(size-1 downto 0)
    );
end reg;
```

Figura 3: Exemplo de entidade com *generic*

2.3 Arquitetura

A arquitetura é a descrição do que acontece dentro da caixa preta descrita na entidade. A figura 4 ilustra os conceitos de entidade e arquitetura do circuito da figura 2.

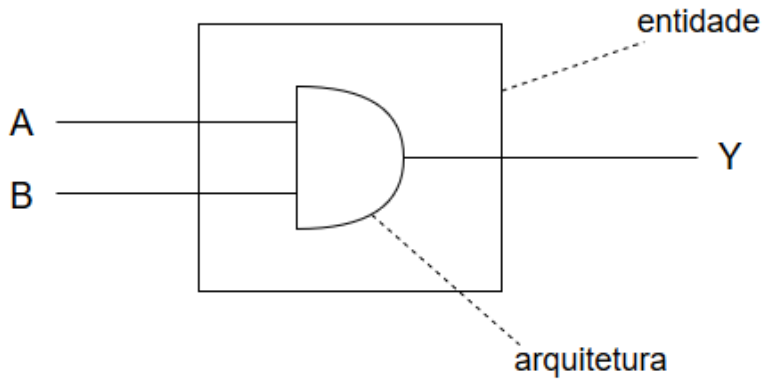


Figura 4: Entidade e arquitetura do circuito da figura 2

Ela é a parte mais trabalhosa da descrição em VHDL, e possui uma série de comandos específicos da linguagem que serão citados posteriormente. No momento você só precisa saber de duas coisas. A primeira é que toda arquitetura possui um nome, sendo que

esse nome não possui tanta importância quanto o nome da entidade. A segunda é que a arquitetura pode ser dividida em duas partes, que são a "área declarativa", onde você pode declarar sinais (variáveis) e outros circuitos e uma "área descritiva", onde você realmente descreve a arquitetura. Isso é ilustrado pela figura 5. A separação destas duas áreas ocorre com a palavra *begin*, e o término da área descritiva ocorre com a palavra *end*, seguida pelo nome dado para a arquitetura.

```
architecture exemplo of porta_and is
-- Area declarativa
begin
-- Area descritiva
end exemplo;
```

Figura 5: Divisão de áreas da arquitetura em VHDL

É importante ressaltar aqui que no VHDL a atribuição de valores é feita com o <= ao invés do =, como você pode observar na figura 2. Esta sintaxe costuma ser utilizada em pseudocódigo.

3 Tipos de Dados em VHDL

Alguns dos tipos de dados disponíveis em VHDL são descritos na tabela a seguir. Os tipos *std_logic* e *bit* costumam ser os mais utilizados na descrição de circuitos digitais.

Tipo de Dado	Valores
bit, bit_vector	'0', '1'
std_logic, std_logic_vector	'X', 'Z', '0', '1'
boolean	True, False
Natural	0 até 2147483647
Integer	-2147483648 até +2147483647
Unsigned	0 até 2147483647
Signed	-2147483648 até +2147483647

Como você pode perceber nas duas primeiras linhas da tabela, o VHDL possui tipos vetor (*vector*). Os tipos vetor (std_logic_vector por exemplo) podem ter duas designações durante sua criação, são elas *to* e *downto*. O *to* considera o conceito normal de linguagem de programação, onde a posição inicial do vetor é 0, seguido de 1, 2... O *downto* considera o conceito espelhado, e mais condizente com a lógica digital, onde a primeira posição do vetor é a mais significativa. A figura 6 ilustra a diferença entre o *downto* e o *to*. Cabe ressaltar que nenhum dos tipos é o melhor, podendo você escolher qual utilizar, porém, em geral o *downto* é mais utilizado.

Figura 6: Diferença *downto* vs *to*

3.1 Conversões entre Tipos de Dados

É possível realizar a conversão entre diversos tipos de dados diferentes no VHDL, onde cada conversão pode ser feita por uma função ou um *casting*. Para realizar estas conversões as vezes é necessário importar algum pacote, então caso apareça um erro na linha de código onde é feita a conversão, é sempre importante confirmar se os devidos pacotes foram importados. A figura 7 ilustra alguns dos tipos de conversões possíveis.

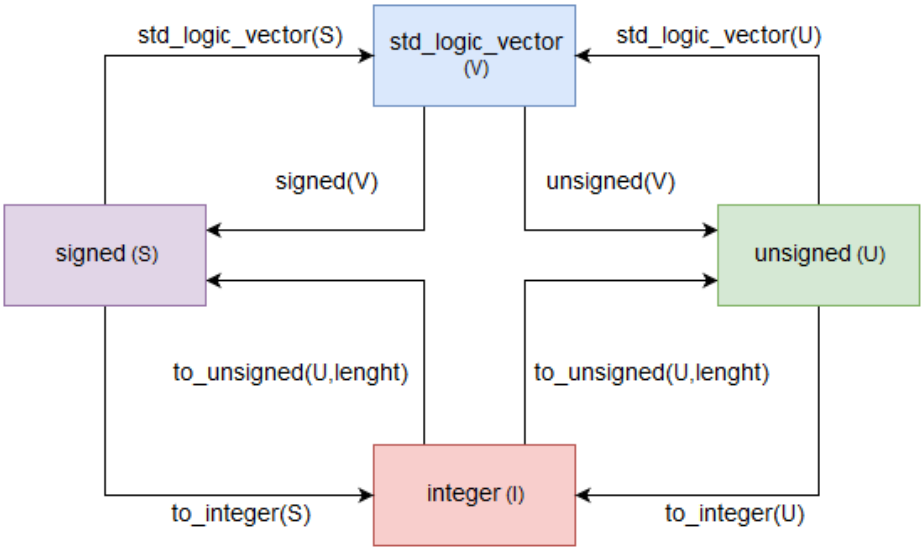


Figura 7: Alguns tipos de conversão de dados em VHDL

3.2 Sinais

Os sinais (*signals*) são as variáveis em VHDL, podendo ser de qualquer tipo de dado apresentado anteriormente. Como estamos descrevendo um circuito digital, eles representam os fios do circuito. Eles são criados na área declarativa da arquitetura (antes do *begin*), e ao contrário de declarar uma porta, como é feito na entidade, os *signals* não possuem direção (*in*, *out*). A figura 8 ilustra a declaração de um *signal*.

```
signal x: std_logic;
```

Figura 8: Declaração de um *signal*

3.3 *Variables*

Variables são um tipo específico de variáveis em VHDL. Elas só existem dentro de uma zona de código definida pelo comando *process*, que será mencionado futuramente. Sua especificidade é que nelas os valores são atribuídos de maneira sequencial, e não concorrente, por isso elas são limitadas aos circuitos sequenciais. Não irei me estender aqui pois em geral não é necessária sua utilização, sendo bem possível realizar o projeto de todos os circuitos solicitados nas disciplinas utilizando apenas *signals*.

4 Operadores em VHDL

Nesta seção são apresentados alguns dos operadores disponíveis no VHDL. Eles podem ser lógicos, aritméticos ou relacionais.

4.1 Operadores Lógicos

Operador	**Operação**
and	Lógica AND
or	Lógica OR
not	Lógica NOT
nand	Lógica NAND
nor	Lógica NOR
xor	Lógica XOR
xnor	Lógica XNOR
&	Operador de concatenação. Concatena bits gerando um vetor

4.2 Operadores Aritméticos

Aqui serão apresentados apenas alguns dos operadores mais simples. Cabe ressaltar que essas operações em geral vão requerer uma conversão para o tipo **unsigned**, não sendo normalmente realizadas com os tipos *std_logic* e *bit*.

Operador	**Operação**
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
**	Exponenciação

4.3 Operadores Relacionais

A seguir são apresentados os operadores relacionais. Estes operadores são utilizados em geral em momentos de decisão.

Operador	**Operação**
=	Igualdade
/=	Desigualdade
>	Maior que
>=	Maior ou igual que
<	Menor que
<=	Menor ou igual que

OBS: Cuidado para não confundir o <= da atribuição com o utilizado como operador relacional.

5 Comandos Combinatórios e Sequenciais

Os comandos utilizados na linguagem VHDL são divididos em duas classes. São elas: os comandos combinatórios e os comandos sequenciais, e serão descritos a seguir.

5.1 Comandos Combinatórios

Os comandos combinatórios são utilizados para descrever circuitos combinatórios, ou seja, circuitos onde a saída depende apenas da entrada. Como exemplo temos as portas lógicas, somadores, multiplexadores, decodificadores, dentre outros.

Existem dois comandos combinatórios, o *with select* e o *when else*. Ambos os comandos tem o intuito de selecionar um valor diferente para a saída a partir da entrada do circuito. Em geral, qualquer circuito combinatório pode ser descrito com um destes comandos ou a composição deles. A figura 9 apresenta a sintaxe destes comandos.

```
with funcao select pivo <=     b <= "1000" when a = "00" else
   A + B when '0',                  "0100" when a = "01" else
   A - B when '1';                  "0010" when a = "10" else
                                    "0001" when a = "11";
```

Figura 9: Sintaxe dos comandos combinatórios em VHDL

5.2 Comandos Sequenciais

Os comandos sequenciais são utilizados para descrever circuitos sequenciais, ou seja, circuitos onde a saída depende da entrada e de uma memória interna. Como exemplo temos os flip flops, registradores, contadores, memórias, dentre outros.

A palavra chave importante aqui é o **sequencial**. Como fazer para descrever um circuito que é sequencial se o VHDL é uma linguagem concorrente? É nesse contexto que surge um dos comandos mais importantes e mal utilizados pelos alunos, que é o comando *process*.

O comando *process* é utilizado para descrever uma zona **sequencial** do código, ou seja, uma zona com começo, meio e fim, muito parecido com o código de uma linguagem de programação. Mas como foi dito logo no início da apostila, um hardware possui execução "infinita", então não faria sentido ele terminar sua execução. Por isso o *process* possui uma lista de sensibilidade, que é declarada entre parênteses. A função dessa lista é executar novamente todo o bloco de código, caso alguns dos sinais da lista de sensibilidade sofra mudança de valores. Desta forma, repetindo continuamente a zona de código sequencial é possível tornar sua execução "infinita". A figura 10 apresenta a sintaxe do comando *process*, onde o *clock* está dentro de sua lista de sensibilidade. Perceba que assim como a arquitetura, o *process* também necessita de um *begin*, para iniciar a zona sequencial, e de um *end* para finaliza-la.

```
process(clock)
begin
    if clear = '1' then
        x <= '0';
    end if;
end process;
```

Figura 10: Sintaxe dos comando *process*

Circuitos sequenciais realizam transições em bordas de *clock* (normalmente na borda de subida). Para capturar a borda de subida do *clock*, e, por conseguinte, realizar uma transição no circuito, pode-se utilizar a função *event* (que retorna *true* se ocorreu mudança de valor da variável vinculada) ou *rising_edge()*, conforme ilustrado pela figura 11.

```
if clock'event and clock = '1' then
                ou
if rising_edge(clock) then
```

Figura 11: Captura da borda de subida do *clock* em VHDL

Como você deve ter observado na figura 11, foi utilizado o comando *if*, que costuma ser empregado em linguagens de programação. Na descrição de circuitos sequenciais, o VHDL utiliza comandos muito parecidos com os das linguagens de programação. Os últimos comandos para apresentar que descrevem circuitos sequenciais são o *if-elsif-else* e o *case* (este comando lembra o *switch* da linguagem C). A figura 12 ilustra suas sintaxes.

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

```
case funcao is
    when "00"   => R <= A + B;
    when "01"   => R <= A - B;
    when "10"   => R <= A and B;
    when "11"   => R <= A or B;
    when others => R <= "ZZZZ";
end case;
```

Figura 12: Sintaxe dos comandos *if-elsif-else* e *case* em VHDL

Uma informação importante apresentada na figura 12 é a palavra reservada *others*. Como o VHDL descreve um circuito digital, e necessário sempre descrever o comportamento do circuito em todos os casos possíveis, caso contrário seu circuito pode ter *bugs*. Aqui, a palavra *others* é utilizada para a condição de qualquer outro caso que não seja um dos casos declarados anteriormente. Em geral utilizamos ela para representar casos que não acontecerão na prática em nosso circuito.

6 Máquina de Transição de Estados

A Máquina de Transição de Estados é um tipo especifico de circuito muito utilizado nos projetos como uma Unidade de Controle, merecendo assim uma seção apenas para ela. A especificidade deste circuito é que a sua descrição é razoavelmente diferente dos outros circuitos, seguindo um modelo estilo "receita de bolo". Ou seja, uma vez descrito um circuito de uma Máquina de Estados genérica, é muito fácil modifica-lo para que ele possa se adequar a qualquer problema.

Existem várias maneiras diferentes de se descrever uma Maquina de Estados, podendo ser Moore ou Mealy e variando o número de comandos *process* utilizados. Aqui será apresentado o modelo de Moore com apenas 1 *process*. Caso você deseje ver outros modelos, diversas fontes na internet monstram isso, como por exemplo [1].

Por se tratar de uma "receita de bolo", é apresentado uma código genérico de uma Máquina de Transição de Estados, responsável pelo controle de um semáforo na figura 13. Nele, foram utilizadas três entradas (A, B, C) para realizar as transições entre os estados do semáforo (vermelho->verde, verde->amarelo, amarelo->vermelho respectivamente).

```
1   -- VHDL de uma Unidade de Controle
2
3   library ieee;
4   use ieee.std_logic_1164.all;
5
6   entity uc is
7      port(
8         clock : in   std_logic;
9         reset : in   std_logic;
10        A     : in   std_logic;
11        B     : in   std_logic;
12        C     : in   std_logic;
13        Y     : out  std_logic;
14        dep   : out  std_logic_vector(1 downto 0)
15     );
16  end uc;
17
18  architecture exemplo of uc is
19     type tipo_estado is (vermelho, amarelo, verde);
20     signal estado   : tipo_estado;
21  begin
22     process (clock, reset, estado, A, B, C)
23     begin
24
25     if reset = '1' then
26        estado <= vermelho;
27
28     elsif (clock'event and clock = '1') then
29        case estado is
30           when vermelho =>
31              if A = '1' then
32                 estado <= verde;
33              else
34                 estado <= vermelho;
35              end if;
36
37           when verde =>
38              if B = '1' then
39                 estado <= amarelo;
40              else
41                 estado <= verde;
42              end if;
43
44           when amarelo =>
45              if C = '1' then
46                 estado <= vermelho;
47              else
48                 estado <= amarelo;
49              end if;
50
51        end case;
52     end if;
53
54     end process;
55
56     with estado select Y <=
57        '1' when verde,
58        '0' when others;
59
60     with estado select dep <=
61        "00" when vermelho,
62        "01" when verde,
63        "10" when amarelo,
64        "11" when others;
65
66  end exemplo;
```

Figura 13: Exemplo de Máquina de Transição de Estados

A seguir será explicado cada parte do código em questão. É importante que você entenda cada uma das partes, pois assim será mais fácil realizar modificações no futuro.

A primeira parte do arquivo é a entidade. Além dos sinais já descritos anteriormente, existe uma entrada *reset* e uma saída *dep*. O *reset* é necessário para garantir a condição inicial do circuito. Toda execução de uma Máquina de Estados deve começar acionando o

[1] www.balbertini.github.io

reset, para garantir que ela se encontra no estado inicial. A função da saída *dep* é indicar ao usuário em que estado a máquina se encontra. Ela é uma saída de depuração, não tendo função dentro do circuito.

```
entity uc is
   port(
      clock : in   std_logic;
      reset : in   std_logic;
      A     : in   std_logic;
      B     : in   std_logic;
      C     : in   std_logic;
      Y     : out  std_logic;
      dep   : out  std_logic_vector(1 downto 0)
   );
end uc;
```

Dentro da arquitetura do sistema existe a declaração de um novo tipo de dado. Esse tipo, aqui denominado de "tipo_estado", deve poder assumir os diferentes valores de estado. Esses valores são definidos através de uma lista, separados por vírgulas dentro dos parênteses. Logo em seguida é declarado um sinal com esse novo tipo de dado definido.

```
architecture exemplo of uc is
   type tipo_estado is (vermelho, amarelo, verde);
   signal estado   : tipo_estado;
begin
```

Em seguida é feito um *process*. Esse *process* é utilizado para descrever o estado atual e a lógica de transição de estados da máquina. As transições entre os estados ocorrem apenas em bordas de subida do *clock*, e elas podem depender ou não, adicionalmente, de alguma condição.

```
process (clock, reset, estado, A, B, C)
begin

if reset = '1' then
   estado <= vermelho;

elsif (clock'event and clock = '1') then
   case estado is
      when vermelho =>
         if A = '1' then
            estado <= verde;
         else
            estado <= vermelho;
         end if;

      when verde =>
         if B = '1' then
            estado <= amarelo;
         else
            estado <= verde;
         end if;

      when amarelo =>
         if C = '1' then
            estado <= vermelho;
         else
            estado <= amarelo;
         end if;

   end case;
end if;

end process;
```

Por último é feita a lógica de saída da máquina. Cabe ressaltar aqui que as máquinas descritas em VHDL geralmente são do tipo Moore, devido a sua maior facilidade de descrição. Nelas, a saída depende apenas do estado atual, podendo então ser representadas com comandos do tipo *with select*. É necessário então realizar um comando para cada saída, tendo o devido cuidado de cobrir o valor da saída em todos os possíveis estados (observe novamente a palavra *others* sendo utilizada).

```
with estado select Y <=
    '1' when verde,
    '0' when others;

with estado select dep <=
    "00" when vermelho,
    "01" when verde,
    "10" when amarelo,
    "11" when others;
```

7 Projeto Hierárquico

O projeto hierárquico é a utilização de vários componentes descritos em VHDL seguindo uma hierarquia, onde um componente depende dos demais. Aqui você possui uma pasta contendo vários arquivos VHDL e necessita conecta-los, para criar um sistema digital.

7.1 Abordagem Estrutural x Comportamental

O primeiro ponto para entender o projeto hierárquico é a diferença entre a abordagem estrutural e a abordagem comportamental quando se descreve um circuito em VHDL.

A abordagem estrutural consiste em descrever um componente digital com base na sua estrutura interna. Pegando como exemplo um somador, considera-se que ele é composto de portas lógicas AND, OR e XOR. A abordagem comportamental consiste em descrever o comportamento do circuito digital, sendo mais simples que a estrutural. No caso do somador, seu comportamento é somar, logo, pode-se descreve-lo utilizando o operador de soma (+).

Em geral, a base da hierarquia de um projeto hierárquico é descrita de maneira comportamental. Por exemplo, descrevemos um multiplexador utilizando o comando combinatório *with select*, ao invés de implementar sua função com portas lógicas. A medida que vamos subindo na hierarquia do projeto, sua abordagem passa a ser estrutural, onde os circuitos mais complexos instanciam os circuitos mais simples, já descritos em VHDL. Esse método de começar pelos circuitos mais simples e ir subindo na hierarquia é denominado método *bottom-up*.

7.2 Comando *port map*

O comando *port map* é o comando mais importante de todos para a criação de qualquer projeto em VHDL. Sua função é permitir que você utilize um circuito já descrito em VHDL em outro circuito.

Como esse comando costuma ser difícil de entender, irei fazer várias analogias com a linguagem C, para tentar facilitar o seu entendimento.

Imagine que você fez uma função *fatorial* na linguagem C, e esta função está salva em um arquivo fat.c. Para você utiliza-la na sua função *main*, salva em um arquivo main.c, é preciso primeiro chamar o protótipo da função, para depois fazer sua instanciação.

Para chamar o protótipo é declarado:

$$int\ fat(int\ n)$$

E depois para instanciar a função dentro da função *main*:

$$x\ =\ fat(n)$$

Exatamente a mesma coisa ocorre em VHDL. Para instanciar um circuito já descrito em outro, é necessário primeiro declarar o protótipo do componente (circuito) para depois fazer sua instanciação.

Para facilitar, vamos ilustrar agora um circuito muito simples, com o objetivo meramente didático. O circuito em questão é apresentado na figura 14. Ele poderia facilmente ser descrito com apenas uma linha de código utilizando o operador lógico AND, porém aqui ele será descrito como um projeto hierárquico.

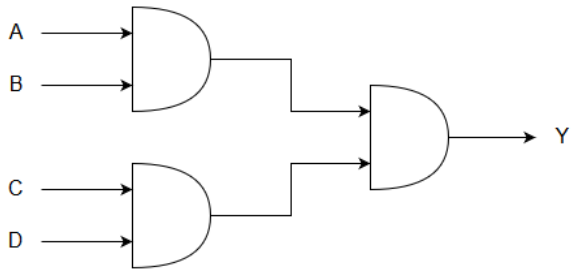


Figura 14: Circuito exemplo

A figura 15 apresenta o código que descreve o circuito em questão. Perceba que são instanciadas 3 portas lógicas AND, onde o início de cada linha da instanciação tem um identificador (and1, and2, and3), que é um nome qualquer. As duas primeiras portas tem suas saídas salvas nos sinais x1 e x2, que entram na terceira porta, cuja saída é a própria saída do circuito.

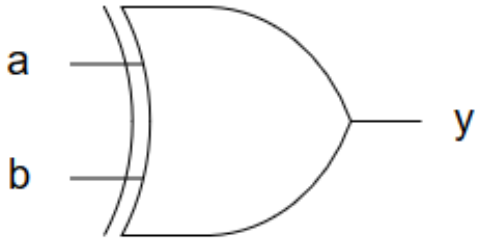
Recomendo que você demore certo tempo observando o código até se sentir seguro que você entendeu a sintaxe do comando *port map*. Cabe ressaltar aqui que esse código só irá funcionar se na mesma pasta do projeto existir o arquivo *porta_and.vhd*.

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

entity circuito is
    port (
        a   : in  std_logic;
        b   : in  std_logic;
        c   : in  std_logic;
        d   : in  std_logic;
        y   : out std_logic
    );
end circuito;

architecture exemplo of circuito is

   entity porta_and is
       port (
           a   : in  std_logic;
           b   : in  std_logic;
           y   : out std_logic
       );
   end porta_and;

   signal x1: std_logic;
   signal x2: std_logic;

begin
   and1: porta_and port map(a,b,x1);
   and2: porta_and port map(c,d,x2);
   and3: porta_and port map(x1,x2,y);
end exemplo;
```

Figura 15: Circuito exemplo em VHDL

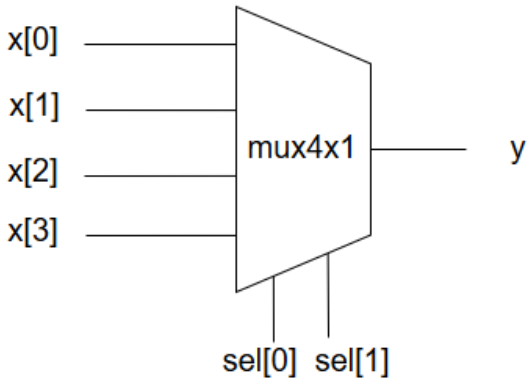
8 Exemplos de Circuitos Descritos em VHDL

Nesta seção serão apresentados nas figuras a seguir alguns circuitos digitais simples descritos em VHDL. O objetivo aqui é que você consiga relacionar o que foi visto durante a leitura da apostila com os circuitos aqui apresentados.



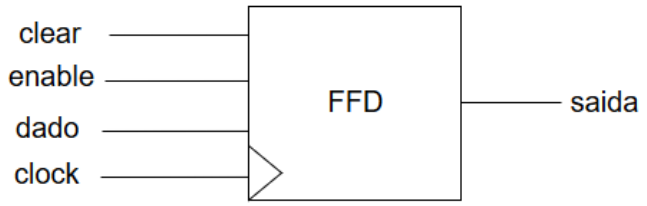

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity porta_xor is
    port (
        a   : in  std_logic;
        b   : in  std_logic;
        y   : out std_logic
    );
end porta_xor;

architecture exemplo of porta_xor is
begin
   y <= a xor b;
end exemplo;
```

Figura 16: Porta XOR em VHDL



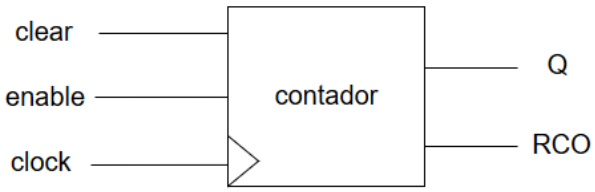

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity mux4x1 is
    port (
        x   : in  std_logic_vector(3 downto 0);
        sel : in  std_logic_vector(1 downto 0);
        y   : out std_logic
    );
end mux4x1;

architecture exemplo of mux4x1 is
begin
   with sel select y <=
      x(0) when "00",
      x(1) when "01",
      x(2) when "10",
      x(3) when "11";
end exemplo;
```

Figura 17: Mux 4x1 em VHDL




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

entity ffd is
    port (clock   : in  std_logic;
          clear   : in  std_logic;
          enable  : in  std_logic;
          dado    : in  std_logic;
          saida   : out std_logic
    );
end ffd;

architecture exemplo of ffd is
begin
   process(clock,clear,enable,dado)
   begin

   if clear = '1' then
      saida <= '0';

   elsif clock'event and clock = '1' then
      if enable = '1' then
         saida <= dado;
      end if;
   end if;

   end process;
end exemplo;
```

Figura 18: Flip flop tipo D em VHDL




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

entity contador is
    port (
        clock   : in  std_logic;
        clear   : in  std_logic;
        enable  : in  std_logic;
        Q       : out std_logic_vector(3 downto 0);
        RCO     : out std_logic
    );
end contador;

architecture exemplo of contador is
   signal IQ: unsigned (3 downto 0);
begin

   process (clock, clear, enable, IQ)
   begin
      if clear = '1' then
         IQ <= (others => '0');
      elsif clock'event and clock = '1' then
         if enable = '1' then
            IQ <= IQ + 1;
         end if;
      end if;
      Q <= std_logic_vector(IQ);
   end process;

   RCO <= '1' when (IQ = 15) else '0';
end exemplo;
```

Figura 19: Contador em VHDL

9 Erros Comuns Cometidos

Nesta seção serão apresentados os erros mais comuns cometidos pelos alunos quando estão descrevendo um circuito em VHDL. Será dado um foco aos erros no Quartus, que é o ambiente de desenvolvimento utilizado no Laboratório Digital.

- Leitura de uma saída

  Conforme foi dito na seção da entidade, uma porta com sentido *out* não pode ser lida. A solução para este caso é utilizar um *signal*, assim você pode realizar todas as operações neste *signal* (inclusive leitura) e então atribui-lo à saída.

- Atribuição múltipla de valores

  Esse erro costuma aparecer como *Multiple Constant Drivers*, e significa que você atribuiu dois valores diferentes para uma mesma saída ou sinal. Não se esqueça, o VHDL é concorrente.

- Top-level indefinido

  Utilizando o Quatus, ele espera que o nome do seu arquivo top-level seja o mesmo nome do seu projeto, caso contrário ele não irá realizar o processo de síntese do seu circuito. Para resolver esse problema você pode renomear o seu arquivo e sua entidade ou então mudar manualmente o nome do seu top-level, abrindo um arquivo do seu projeto e selecionando a opção *Set as top level*.

- Problema com reset assíncrono

  Conforme visto na seção de comandos sequenciais, para capturar a situação de borda de *clock*, é necessário utilizar um comando *if*. Porém, quando seu reset é assíncrono, você não pode utilizar um *if* para o reset e outro para a borda de *clock*. Neste caso, é necessário fazer um cascateamento do tipo *if-elsif*, onde o reset é a condição de *if*, por ter maior prioridade, e a borda de *clock* é o *elsif*. Isso foi feito na figura 18.

10 Exercícios

1. Qual a diferença entre um código feito em uma linguagem de programação e um circuito descrito em uma linguagem de descrição de hardware?

2. Como você faria para capturar a condição de borda de descida do *clock* em VHDL?

3. Qual a função do comando *port map*?

4. Em que tipo de circuito não se deve utilizar o comando *process*?

5. Qual a diferença em VHDL do tipo *bit* para o tipo *std_logic*?

6. Como você faria para cascatear contadores em VHDL, visando a descrição de um circuito que tenha mais de 1 dígito de contagem? Dica: É necessário utilizar o comando *port map*.

7. Descreva em VHDL um circuito somador completo de 1 bit. A figura 20 apresenta o diagrama lógico do circuito.

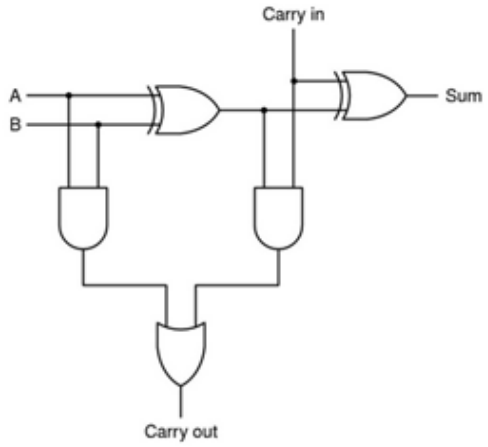


Figura 20: Somador completo de 1 bit

8. Descreva em VHDL um circuito contador decimal. Para tal, você pode se basear no contador binário apresentado na figura 19.

9. Descreva em VHDL um registrador de deslocamento, com deslocamento linear para a direita. Dica: a operação de deslocamento pode ser feita com o auxílio do operador &, concatenando um bit 0 no início do conteúdo do registrador.

10. Descreva em VHDL uma Máquina de Estados responsável pelo controle de um ventilador. Considere que o ventilador possui duas entradas, uma para aumentar a velocidade e outra para diminui-la, e três estados onde cada estado representa uma velocidade diferente.