Apostila de Lógica Reconfigurável com VHDL: Fundamentos e Aplicações

Prof. Felipe Walter Dafico Pfrimer

9 de dezembro de 2025

Sumário

Capítulo 1

Introdução

Esta apostila apresenta conceitos fundamentais de VHDL utilizados na disciplina, introduzindo a base tecnológica e a linguagem de descrição. Especificamente, este capítulo aborda a definição de Dispositivos Lógicos Programáveis (PLDs), a importância das Linguagens de Descrição de Hardware (HDLs) e uma introdução ao VHDL.

1.1 O que são Dispositivos Lógicos Programáveis?

Em um futuro distante, um marceneiro chega ao seu galpão para mais um dia de trabalho. No interior do espaço, repousam alguns cubos perfeitos — cada um com exato um metro de lado — feitos de um material misterioso. O marceneiro pega um desses cubos, leva-o ao centro do galpão e, com um pequeno dispositivo em mãos, emite comandos que dão vida à matéria inerte. Diante de seus olhos, o cubo se reconfigura: transforma-se primeiro em uma mesa robusta, depois em uma cadeira ergonômica, em seguida em uma estante organizada — tudo de acordo com a necessidade daquele momento.

O segredo está na divisão de tarefas: o dispositivo armazena os *blueprints* (os planos detalhados de cada móvel), enquanto o cubo é feito de um material inteligente, capaz de se moldar dinamicamente, respeitando apenas os limites de sua massa e volume.

Embora soe como ficção científica, essa metáfora ilustra o funcionamento dos **Dispositivos Lógicos Programáveis** (PLDs – *Programmable Logic Devices*): componentes eletrônicos cuja arquitetura interna pode ser configurada pelo usuário para executar funções lógicas específicas, simplesmente carregando um novo "projeto" — ou seja, um novo arquivo de configuração.

No contexto dos PLDs, o "material inteligente" é representado por uma matriz de blocos lógicos e interconexões programáveis. Esses blocos podem ser configurados para realizar operações lógicas variadas, enquanto as interconexões permitem que os blocos se comuniquem, formando circuitos complexos. O *blueprint*, por sua vez, representa o projeto digital. Esse projeto pode ser descrito tanto por diagramas esquemáticos quanto por **Linguagens de Descrição de Hardware** (HDLs).

Existem diversas categorias de PLDs. Entre as mais comuns estão os **FPGAs** (*Field-Programmable Gate Arrays*) e os **CPLDs** (*Complex Programmable Logic Devices*). Os FPGAs são conhecidos por sua alta densidade lógica e flexibilidade, permitindo a implementação de sistemas digitais complexos. Já os CPLDs costumam ser mais simples, possuem arquitetura menos granular e são ideais para lógicas de controle e aplicações que exigem previsibilidade temporal rigorosa.

1.2 Linguagens de Descrição de Hardware

Uma HDL (*Hardware Description Language*) **não é uma linguagem de programação**. Como o próprio nome sugere, trata-se de uma linguagem utilizada para **descrever** o hardware. Diferentemente das linguagens de software (como C ou Python), que geram instruções sequenciais para um processador, as HDLs permitem que engenheiros definam a estrutura e o comportamento de circuitos eletrônicos, onde muitas coisas acontecem simultaneamente (concorrência).

Atualmente, as HDLs mais populares para configurar PLDs são o **VHDL** (*VHSIC Hardware Description Language*), o **Verilog** e o **SystemVerilog**. Todas permitem a descrição de circuitos em diferentes níveis de abstração, desde o comportamental (o que o circuito faz) até o estrutural (quais portas lógicas compõem o circuito). Esta apostila foca no uso do **VHDL**.

1.3 O que é VHDL?

A sigla VHDL significa **VHSIC Hardware Description Language**, onde VHSIC é o acrônimo para *Very High Speed Integrated Circuit*. Desenvolvida na década de 1980 pelo Departamento de Defesa dos Estados Unidos, a linguagem foi criada para documentar e simular o comportamento de circuitos integrados complexos. Com o tempo, o VHDL evoluiu e se tornou um padrão internacional (IEEE 1076), amplamente adotado na indústria eletrônica para o design de sistemas digitais.

Para exemplificar, considere o trecho de código 1.3.1 escrito em VHDL, que descreve uma porta AND de duas entradas. Não se preocupe em entender todos os detalhes agora; o objetivo é apenas ilustrar a estrutura básica da linguagem.

Código 1.3.1 Exemplo de uma porta AND em VHDL

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity AND2 is
5      port (
6          A, B : in  std_logic;
7          Y    : out std_logic
8      );
9  end AND2;
10
11 architecture rtl of AND2 is
12 begin
13     Y <= A and B;
14 end rtl;
```

Observe no Código 1.3.1 a estrutura básica de um arquivo VHDL:

- **Bibliotecas:** As linhas iniciais importam a biblioteca `ieee`, necessária para usar o tipo de dado padrão `std_logic`. Outras bibliotecas podem ser incluídas conforme a complexidade do projeto.
- **Entidade (Entity):** Define a interface do componente, ou seja, suas entradas (`A`, `B`) e saídas (`Y`). É a "caixa preta"vista de fora. Aqui, o projetista deve prever quais sinais o componente receberá e quais ele fornecerá.

- **Arquitetura (Architecture):** Descreve o comportamento interno. Neste caso, a saída `Y` recebe o resultado da operação lógica AND entre `A` e `B`. É nessa região que os iniciantes em VHDL possuem maior dificuldade, pois já estão acostumados a pensar em termos de sequências de comandos comuns nas linguagens de programação. Dessa forma, o VHDL exige uma mentalidade de descrição concorrente. Na arquitetura, todas as operações são consideradas simultâneas, refletindo a natureza paralela do *hardware* alvo.

Além da descrição para síntese (criação do circuito físico), o VHDL é amplamente utilizado para **simulação**. Isso permite que os desenvolvedores validem o comportamento do circuito no computador antes de implementá-lo no PLD, economizando tempo e garantindo a confiabilidade do projeto.

O código a seguir apresenta um exemplo simples de testbench em VHDL, utilizado para simular o comportamento da porta AND definida no Código 1.3.1.

Código 1.3.2 Exemplo de Testbench para a porta AND

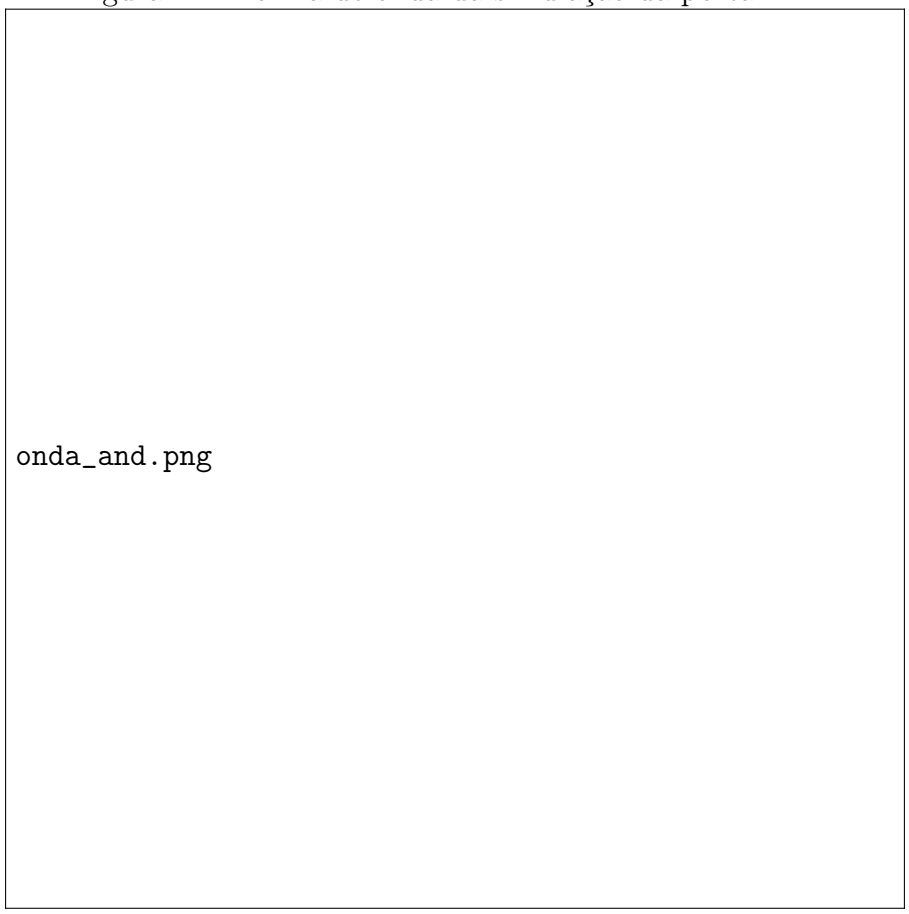
```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  entity tb_AND2 is
5  end tb_AND2;
6  architecture behavior of tb_AND2 is
7      signal A, B : std_logic := '0';
8      signal Y    : std_logic;
9
10     component AND2
11         port (
12             A, B : in  std_logic;
13             Y    : out std_logic
14         );
15     end component;
16 begin
17     uut: AND2 port map (A => A, B => B, Y => Y);
18     process
19     begin
20         A <= '0'; B <= '0'; wait for 10 ns;
21         A <= '0'; B <= '1'; wait for 10 ns;
22         A <= '1'; B <= '0'; wait for 10 ns;
23         A <= '1'; B <= '1'; wait for 10 ns;
24         wait;
25     end process;
26 end behavior;
```

No Código 1.3.2, o testbench cria sinais de entrada (`A` e `B`) e conecta a unidade sob teste (UUT - *Unit Under Test*), que é a porta AND definida anteriormente. O processo dentro do testbench aplica diferentes combinações de entradas, aguardando 10 nanosegundos entre cada mudança, permitindo observar a saída `Y` em resposta às entradas. O resultado da simulação pode ser visualizado em uma forma de onda, facilitando a verificação do comportamento correto do circuito, como pode ser visto na Figura 1.1.

Dessa forma, o VHDL vai possuir duas formas principais de uso:

- **Síntese:** Onde o código VHDL é infere o circuito lógico que será implementado no PLD.

Figura 1.1: Forma de onda da simulação da porta AND



- **Simulação:** Onde o código VHDL é utilizado para validar o comportamento do circuito antes da síntese.

Essas duas vertentes são fundamentais no fluxo de projeto digital, garantindo que o design atenda aos requisitos funcionais e de desempenho antes da implementação física. No entando, cada uma dessas vertentes possui suas próprias regras e boas práticas de codificação, que serão abordadas nos capítulos seguintes. Isso acontece porque certos construtos do VHDL são adequados para simulação, mas não para síntese, e vice-versa. Por exemplo, estruturas como `wait` e `after` são úteis para simulação, mas não podem ser sintetizadas em hardware.

1.4 Histórico e Evolução da Linguagem VHDL

A linguagem VHDL (*VHSIC Hardware Description Language*) possui uma origem distinta da maioria das linguagens de programação de software, tendo sido concebida inicialmente não para a execução, mas para a documentação e preservação de projetos eletrônicos complexos.

1.4.1 Origens e o Programa VHSIC (Década de 1980)

No início da década de 1980, o Departamento de Defesa dos Estados Unidos (DoD) enfrentava uma crise logística relacionada ao ciclo de vida de seus componentes eletrônicos. Com o avanço rápido da tecnologia, chips utilizados em equipamentos militares tornavam-se obsoletos rapidamente, e a falta de documentação padronizada impedia a reprodução desses componentes por novos fabricantes.

Para solucionar este problema, o DoD iniciou o programa VHSIC (*Very High Speed Integrated Circuits*). O objetivo era criar uma linguagem agnóstica de tecnologia que pudesse descrever o comportamento e a estrutura de circuitos integrados. Em 1983, um contrato foi firmado com a Intermetrics, IBM e Texas Instruments para desenvolver a primeira versão da linguagem. Uma decisão crucial de design foi basear a sintaxe na linguagem Ada, herdando desta a tipagem forte e a não-sensibilidade a letras maiúsculas/minúsculas, visando reduzir erros de ambiguidade em projetos críticos.

1.4.2 Padronização e Evolução (IEEE)

Embora criada para documentação, os engenheiros perceberam rapidamente que uma descrição comportamental precisa poderia ser executada por softwares de simulação. Para fomentar a adoção pela indústria civil e garantir a interoperabilidade entre ferramentas de CAD (*Computer-Aided Design*), o DoD transferiu os direitos da linguagem para o IEEE (*Institute of Electrical and Electronics Engineers*) em 1986.

O primeiro padrão oficial foi publicado como **IEEE Standard 1076-1987** (conhecido como VHDL-87). Desde então, a linguagem passou por revisões periódicas para incorporar novas capacidades de síntese e verificação:

- **VHDL-93:** Introduziu uma sintaxe mais consistente, identificadores estendidos e melhorias na manipulação de arquivos.
- **IEEE 1164:** Embora não seja uma revisão da linguagem em si, a padronização do pacote `std_logic_1164` foi um marco fundamental. Ela introduziu o sistema lógico de 9 valores (incluindo 'Z' para alta impedância e 'X' para desconhecido), permitindo a modelagem realista de circuitos digitais.
- **VHDL-2000 e 2002:** Pequenas revisões que adicionaram o modificador `protected` para tipos.
- **VHDL-2008 (IEEE 1076-2008):** Uma grande modernização que incorporou recursos inspirados em SystemVerilog, facilitando a verificação e reduzindo a verbosidade do código para operações comuns.
- **VHDL-2019:** A iteração mais recente, focada em melhorias para interfaces de verificação e introspecção de tipos.

1.4.3 Estado da Arte e Aplicação Atual

Atualmente, o VHDL é uma das linguagens dominantes no design de hardware digital, dividindo o mercado global com o Verilog/SystemVerilog. Sua aplicação primordial migrou da documentação para a **síntese lógica** e a **simulação**.

O VHDL é amplamente empregado no desenvolvimento de FPGAs (*Field-Programmable Gate Arrays*) e ASICs (*Application-Specific Integrated Circuits*). Devido à sua natureza fortemente tipada e determinística, o VHDL mantém uma forte preferência em setores que exigem alta confiabilidade e segurança crítica, como as indústrias aeroespacial, automotiva, médica e de defesa, especialmente na Europa e nas Américas. Ferramentas modernas de síntese da Intel (Quartus) e AMD/Xilinx (Vivado) oferecem suporte abrangente aos padrões mais recentes, permitindo que a linguagem descreva desde portas lógicas simples até processadores *soft-core* complexos e aceleradores de inteligência artificial.

Capítulo 2

Introdução

Esta apostila apresenta conceitos fundamentais de VHDL utilizados na disciplina.

2.1 Primeiro Exemplo

A seguir temos um exemplo simples de entidade e arquitetura em VHDL.

Código 2.1.1 Exemplo de entidade e arquitetura

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity AND2 is
5      port (
6          A, B : in  std_logic;
7          Y    : out std_logic
8      );
9  end AND2;
10
11 architecture rtl of AND2 is
12 begin
13     Y <= A and B;
14 end rtl;
```

Podemos referenciar o código acima como Código 2.1.1.

2.2 Conclusão

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.