Linguagem de descrição de Hardware

Atualmente, a tendência dentro de sistemas digitais é empregar linguagens baseadas em texto para a descrição de circuitos digitais, estas conhecidas como linguagens de descrição de hardware (HDL, do inglês, *Hardware Description Languages*). Um sistema digital é um conjunto de vários componentes que coordenam os dispositivos de entrada e saída (I/O), além da memória para a realização do processamento de algum dado. Um sistema de digital pode ser entendido como a união do Hardware (processador principal, memória, dispositivos de I/O e barramentos) e do Software (programa que usa os componentes do hardware). HDL é usada extensivamente na indústria para projetos de sistemas digitais, desde microprocessadores até componentes dentro de outros sistemas.

Como é mostrado na Figura 1, os sistemas digitais podem ser agrupados em três principais famílias:

- Lógica padrão: engloba os componentes digitais mais básicos (portas lógicas, flip-flops, decodificadores) que estão disponíveis como circuitos integrados, alguns mais de 30 anos. Eles já vêm sendo usados há anos para projetar sistemas digitais complexos. Eles possuem diversos encapsulamentos devido a diversidade de tipos existentes. O processo de manipulação é complexo e envolve um alto custo. O hardware já vem pronto de fábrica com uma função específica. A ULSI é um exemplo de empresa que lida com este tipo de projeto.
- Circuitos integrados de aplicação específica (ASICs - Application Specific Integrated Circuits): É a solução moderna em termos de hardware para os sistemas digitais. São circuitos integrados projetados para implementar uma aplicação específica. São sistemas mais complexos, que demandam muito tempo de desenvolvimento e muitos recursos. Isso os torna sistemas mais caros, em contrapartida, o sistema resultante oferece um ótimo desempenho com um baixo consumo de energia. Exemplos são os PLDs (dispositivos lógicos programáveis), como o FPGA e até os PSoCs.
- Microcontroladores/Microprocessadores/processadores digitais de sinais (DSP - do inglês Digital Signal Processor): Diferente do projeto de sistemas digitais com componentes digitais básicos, esses dispositivos possuem vários tipos de blocos funcionais dentro de uma placa. Eles possuem um hardware fixo, mas com a flexibilidade de adequar o código a cada problema. O grande problema associados a esse tipo de dispositivo é a velocidade, uma vez que a solução de hardware para o seu projeto de sistema digital sempre será mais rápido do que uma solução de software.

A HDL é necessária, uma vez que, mesmos os potentes computadores que estão disponíveis no mercado, não são capazes de dado uma descrição de um circuito lógico em linguagem comum converter isso para um circuito lógico funcional (físico). A subfamília que mais se vale das linguagens de descrição de hardware são os PLDs.

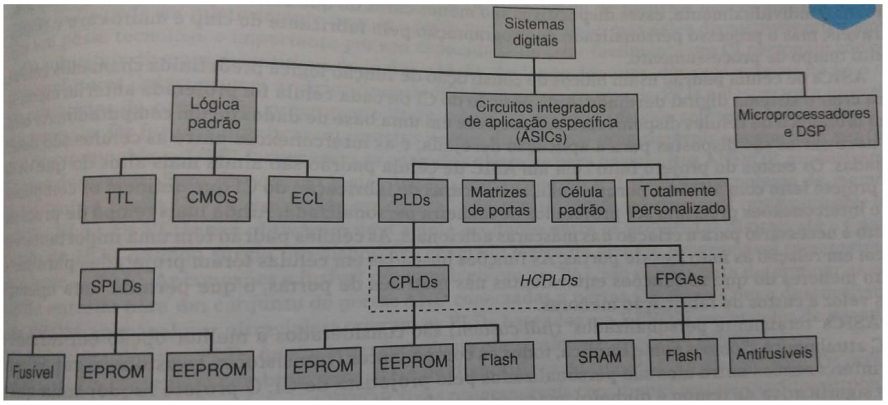


Figura 1: Famílias de sistemas digitais

HDL é usada para descrever a estrutura de sistemas eletrônicos e seu comportamento ao longo do tempo utilizando expressões padrão baseadas em texto. HDL utiliza sintaxe e semântica para deixar explícito notações que expressam simultaneidade, além de noções explícitas de tempo, a qual é um atributo primário do hardware. Ele é usado para escrever especificações executáveis para o hardware.

A criação da HDL visa fornecer ao projetista de hardware a capacidade de modelar, simular e testar uma peça de hardware antes mesmo que ela seja criada fisicamente. Isso é feito ao implementar a semântica subjacente das instruções de linguagens e simular o progresso de execução com o passar do tempo. Há HDLs direcionadas para suportar modelagem de eventos discretos (digital) e de tempo contínuo (analógico).

O objetivo da HDL é expressar a conectividade do circuito entre uma hierarquia de blocos que estão adequadamente classificadas como uma linguagem de arquitetura do circuito (rede de ligações entre os elementos disponíveis na plataforma - *netlist*) independente de tecnologia e fabricante. Ela é utilizada no projeto auxiliado por computador (CAD - computer-aided design). Já o uso de HDL é focado em escrever especificações executáveis em um pedaço de hardware. Como as HDLs são independentes de tecnologia e fabricante, ela pode ser usada para troca de informações entre projetistas, fabricantes e fornecedores. Outro uso é a síntese de circuitos para CPLDs (*Complex Programmable Logic Device*) e FPGAs (*Field Programmable Gate Array*), além da criação layout e geração de máscaras para ASICs. Antes de ser criado fisicamente, pode-se usar programas de simulação, estes usados para implementar a semântica da HDL, em conjunto com progresso da simulação com o passar do tempo. Há linguagens podem ser simuladas, entretanto, não podem ser sintetizadas. As principais ferramentas de HDL são: Quartus II (síntese e simulação gráfica - Altera); ISE (síntese e simulação - Xilinx); Precision RTL (síntese - Mentor Graphics); ModelSim (simulação - Mentor Graphics); e NC-Sim (simulação - Cadence).

Existe um conjunto de linguagens classificadas como linguagens de netlist usadas no projeto auxiliado por computador (CAD - Computer-Aided Design) que são usadas apenas para descrever a conectividade do circuito entre uma hierarquia de blocos. Além de descrever a arquitetura, a HDL pode ser usada para expressar projetos comportamentais e abstração de arquiteturas RTL (register-transfer level), o qual é um modelo de fluxo de um circuito ao longo do tempo. Nesses dois últimos casos, o sintetizador (conversor de projeto de descrição de hardware para projeto físico) decide a arquitetura e layout das portas

podem ser simuladas, entretanto, não podem ser sintetizadas. Outra distinção importante é que, apesar de uma linguagem ter um subconjunto que pode ser sintetizado, isso não implica que ela é uma linguagem de descrição de hardware. As principais ferramentas de HDL são: Quartus II (síntese e simulação gráfica - Altera); ISE (síntese e simulação - Xilinx); Precision RTL (síntese - Mentor Graphics); ModelSim (simulação - Mentor Graphics); e NC-Sim (simulação - Cadence).

Existe um conjunto de linguagens classificadas como linguagens de netlist usadas no projeto auxiliado por computador (CAD - *Computer-Aided Design*) que são usadas apenas para descrever a conectividade do circuito entre uma hierarquia de blocos. Além de descrever a arquitetura, a HDL pode ser usada para expressar projetos comportamentais e abstração de arquiteturas RTL (*register-transfer level*), o qual é um modelo de fluxo de um circuito ao longo do tempo. Nesses dois últimos casos, o sintetizador (conversor de projeto de descrição de hardware para projeto físico) decide a arquitetura e layout das portas lógicas.

Esse conceito de simulação passa a ilusão que HDL serem linguagens de programação, todavia, a HDL não é uma linguagem de programação, elas são classificados como linguagens de especificação ou linguagens de modelagem. Essa confusão ocorre pois ambas usam uma linguagem para programar um dispositivo. A HDL visa descrever a configuração de hardware de um circuito, enquanto as linguagens de programação representam uma sequência de instruções a serem executadas por um computador a fim de realizar alguma tarefa. Além disso, HDL incluem anotações explícitas para expressar simultaneidade, com sintaxe e semântica próprias, e noções implícitas de tempo, como um atributo primário de hardware.

As linguagens HDL vêm ganhando cada vez mais espaço devido a grande evolução e complexidade que os circuitos eletrônicos digitais vêm tendo desde 1970. Tecnologias específicas como a CMOS e TTL são usadas para criar soluções de hardware específicos. A HDL entra como uma solução para a criação de sistemas digitais em alto nível sem estar atrelado a tecnologias específicas.

É possível utilizar linguagens de programação tradicionais, como C++, em conjunto com bibliotecas específicas. Aliado a dificuldade de manejo atrelada a essas bibliotecas específicas, há a falta de artifícios de expressar explicitamente de tempo. Esses dois fatores fazem com linguagens de programação sirvam para descrição de hardware. Todavia elas não podem ser classificadas como HDL, uma vez que não apresentam recursos para expressar tempo explicitamente.

Não era comum em HDL programação orientadas a objetos, a forma mais comum era a integração de C++ com simuladores de lógica. Visando esse cenário, houve a introdução do SystemVerilog, a qual oferece orientação a objetos e *garbage collection*.

A netlist equivalente de hardware genérico usada para especificar o comportamento do circuito, além das operações lógicas de hardware das instruções de linguagem, podem ser inferido usando HDL (ou qualquer linguagem que tenha um subconjunto para isso) atrelado a um programa para sintetizar essa linguagem.

As primeiras linguagens usadas para descrever hardware apareceram no final da década de 1960, início da década de 1970. Elas se aproximavam muito da estrutura das linguagens de programação, ou eram bibliotecas extras. A primeira com efeito foi descrita em C. O conceito de RTL foi introduzido na linguagem ISP (Processador de Conjunto de Instruções) usada para descrever o comportamento do Digital Equipment Corporation (DEC) PDP-8. A ISP foi desenvolvida pela Carnegie Mellon University. A ISP era uma linguagem mais próxima as linguagens de programação. Ela era usada para descrever as relações de entradas e saídas de sistema digital. Além disso, ISP possuíam Módulos de RTL (RTMs). Ela é um tipo de linguagem que pode ser usada para simular o design, mas não para sintetizá-lo.

Apesar dos avanços que ISP introduziram em conjunto com RTMs, os produtos com essas tecnologias não foram unanimidade no mercado. À medida que novas técnicas e, em particular, a integração em larga escala (VLSI) se tornaram mais populares, a tecnologia IPS foi descontinuada em meados da década de 1980.

Uma outra linguagem chamada KARL (*KAiserslautern Register Transfer Language*) foi desenvolvida pela Universidade de Kaiserslautern em 1979. Essa linguagem incluía suporte para VLSI e design de hardware estruturado. Ela foi base para a criação da linguagem de diagramas de blocos (ABL - *A Block diagram Language*).

O uso de HDL foi muito alavancado principalmente devido aos dispositivos lógicos programáveis (PLDs). O design usando PLDs tornou-se muito popular ao final dos anos 1970. Os PLDs, principalmente os FPGAs, utilizam a HDL para descrever o seu hardware e fomentar o desenvolvimento de outros dispositivos, como SoCs (*System on Chip*).

Como o design foi direcionado para VLSI, novas linguagens e técnicas foram introduzidas, como o Verilog que foi desenvolvido pela *Gateway Design Automation* em 1985. Em 1987, o Departamento de Defesa dos EUA solicitou o desenvolvimento do VHDL (VHSIC Hardware Description Language), o qual é for baseado na programação Ada e na ISP. A Ada é uma linguagem de programação alto nível estruturada e orientada de objetos ampliada do Pascal e de outras linguagens.

A princípio, Verilog e VHDL foram amplamente usadas para documentar e simular projetos escritos em outros formatos, como a forma esquemática. A simulação de HDL tem um papel importante no processo de design de Sistemas Embarcados em geral. Ela permite que os projetistas trabalhem em um nível mais alto de abstração do que a simulação no nível esquemático. Isso favorece no ponto em que aumenta a capacidade de projeto de centenas de transistores para milhares.

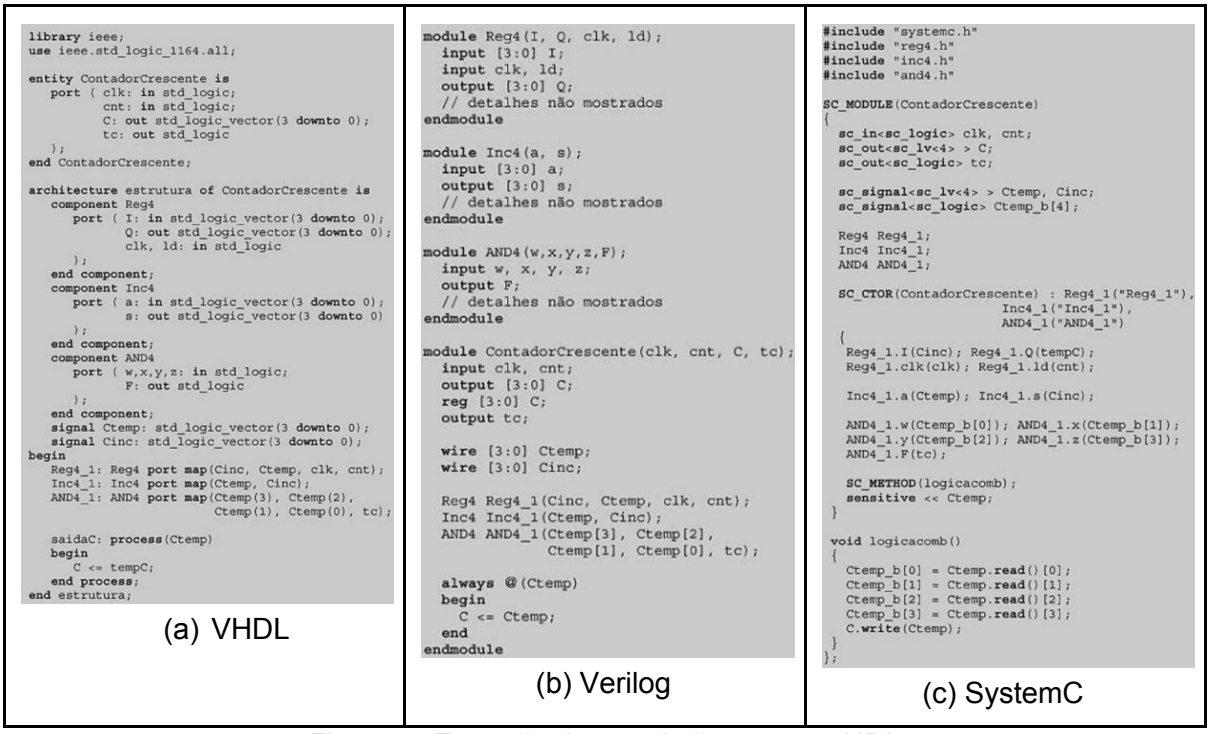
Quando as HDLs começaram a ser usadas para síntese, houve um impulsionamento e as HDLs que até então eram segundo plano para o design digital, entraram em evidência. A função das ferramentas de síntese é compilar os arquivos de origem HDL (escritos em formato restrito RTL - VHDL e Verilog) em uma descrição de netlist fabricável em termos de portas e transistores.

Normalmente, as netlists RTL criadas pela sintetização da HDL eram maiores em área e mais lentas no desempenho quando comparado com esquemas tradicionais de layout.

Geralmente, um projeto de projetista experiente usando captura esquemática tende a superar seu equivalente logicamente sintetizado. A vantagem dos circuitos logicamente sintetizados está relacionado a produtividade em áreas problemáticas para síntese RTL: circuitos de alta velocidade, baixa consumo de energia e assíncronos.

Com o desenvolvimento da tecnologia, as linguagens VHDL e Verilog dominaram a indústria eletrônica enquanto HDLs mais antigos e mais limitadas foram gradualmente desaparecendo e caindo em desuso. Todavia, ambas as linguagens possuem algumas limitações compartilhadas: não trabalham muito bem com sinais analógicos e mistos e problemas de estruturas lógicas geradas de forma recursiva. Algumas linguagens HDL específicas foram introduzidas para contornar esses problemas, como o Confluence, mas sem o intuito de substituir nenhuma das duas linguagens.

Conforme a necessidade de novas funcionalidade, como a reutilização de código, hierarquia de design e randomização, foram surgindo novas linguagens como a SystemVerilog e SystemC. Há a introdução de conceitos como classe, variáveis aleatórias e assertions. Atualmente, as linguagens modernas possuem sua estrutura bem próximas, todavia, cada uma possui suas peculiaridades na sua sintaxe. A Figura 2 mostra uma comparação da implementação de um contador de 4 bits usando três das linguagens mais populares, SystemC, VHDL e Verilog.



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

entity ContadorCrescente is
   port ( clk: in std_logic;
          cnt: in std_logic;
          C: out std_logic_vector(3 downto 0);
          tc: out std_logic
   );
end ContadorCrescente;

architecture estrutura of ContadorCrescente is
   component Reg4
      port ( I: in std_logic_vector(3 downto 0);
             Q: out std_logic_vector(3 downto 0);
             clk, ld: in std_logic
      );
   end component;
   component Inc4
      port ( a: in std_logic_vector(3 downto 0);
             s: out std_logic_vector(3 downto 0)
      );
   end component;
   component AND4
      port ( w,x,y,z: in std_logic;
             F: out std_logic
      );
   end component;
   signal Ctemp: std_logic_vector(3 downto 0);
   signal Cinc: std_logic_vector(3 downto 0);
begin
   Reg4_1: Reg4 port map(Cinc, Ctemp, clk, cnt);
   Inc4_1: Inc4 port map(Ctemp, Cinc);
   AND4_1: AND4 port map(Ctemp(3), Ctemp(2),
                         Ctemp(1), Ctemp(0), tc);

   saidaC: process(Ctemp)
   begin
      C <= tempC;
   end process;
end estrutura;
```

(a) VHDL

```verilog
module Reg4(I, Q, clk, ld);
  input [3:0] I;
  input clk, ld;
  output [3:0] Q;
  // detalhes não mostrados
endmodule

module Inc4(a, s);
  input [3:0] a;
  output [3:0] s;
  // detalhes não mostrados
endmodule

module AND4(w,x,y,z,F);
  input w, x, y, z;
  output F;
  // detalhes não mostrados
endmodule

module ContadorCrescente(clk, cnt, C, tc);
  input clk, cnt;
  output [3:0] C;
  reg [3:0] C;
  output tc;

  wire [3:0] Ctemp;
  wire [3:0] Cinc;

  Reg4 Reg4_1(Cinc, Ctemp, clk, cnt);
  Inc4 Inc4_1(Ctemp, Cinc);
  AND4 AND4_1(Ctemp[3], Ctemp[2],
              Ctemp[1], Ctemp[0], tc);

  always @(Ctemp)
  begin
    C <= Ctemp;
  end
endmodule
```

(b) Verilog

```systemc
#include "systemc.h"
#include "reg4.h"
#include "inc4.h"
#include "and4.h"

SC_MODULE(ContadorCrescente)
{
  sc_in<sc_logic> clk, cnt;
  sc_out<sc_lv<4> > C;
  sc_out<sc_logic> tc;

  sc_signal<sc_lv<4> > Ctemp, Cinc;
  sc_signal<sc_logic> Ctemp_b[4];

  Reg4 Reg4_1;
  Inc4 Inc4_1;
  AND4 AND4_1;

  SC_CTOR(ContadorCrescente) : Reg4_1("Reg4_1"),
                               Inc4_1("Inc4_1"),
                               AND4_1("AND4_1")
  {
   Reg4_1.I(Cinc); Reg4_1.Q(tempC);
   Reg4_1.clk(clk); Reg4_1.ld(cnt);

   Inc4_1.a(Ctemp); Inc4_1.s(Cinc);

   AND4_1.w(Ctemp_b[0]); AND4_1.x(Ctemp_b[1]);
   AND4_1.y(Ctemp_b[2]); AND4_1.z(Ctemp_b[3]);
   AND4_1.F(tc);

   SC_METHOD(logicacomb);
   sensitive << Ctemp;
  }

  void logicacomb()
  {
    Ctemp_b[0] = Ctemp.read()[0];
    Ctemp_b[1] = Ctemp.read()[1];
    Ctemp_b[2] = Ctemp.read()[2];
    Ctemp_b[3] = Ctemp.read()[3];
    C.write(Ctemp);
  }
};
```

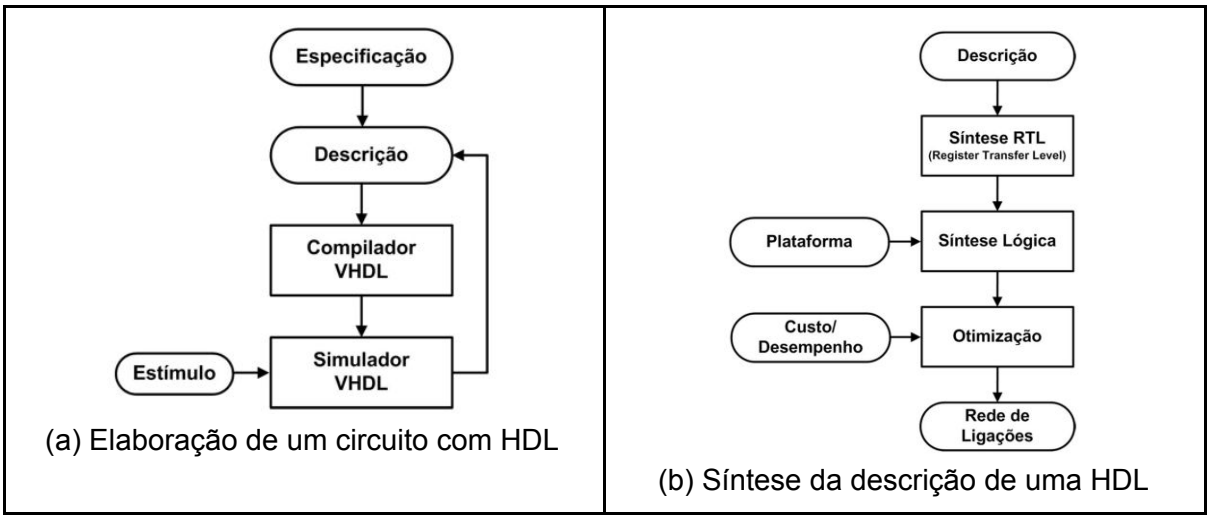
(c) SystemC

Figura 2: Exemplo do uso de linguagens HDL

VHDL e Verilog possui algumas diferenças em suas sintaxes. O VHDL é muito mais tipado e verboso do que o Verilog, o qual é fracamente tipado e bastante conciso. A sintaxe e estrutura do Verilog é parecida com a C, diferente da VHDL. A linguagem VHDL tende a ser bastante determinística, em contrapartida a Verilog que somente é determinística se

algumas regras são seguidas cuidadosamente. O SystemC é uma linguagem mais próxima a estrutura usada em C/C++.

O processo de design usando HDL pode ser dividido em duas etapas: elaboração da descrição HDL e síntese da descrição criada. O processo de elaboração da descrição HDL, o qual é mostrado na Figura 3a, envolve o todo o processo antes da síntese da HDL. Geralmente, ele é totalmente independente da tecnologia e do fabricante. Já a síntese da descrição criada (Figura 3b) envolve a síntese da descrição na etapa anterior para uma tecnologia e fabricante dependentes. Dependendo da tecnologia física (FPGA, ASIC gate array, célula padrão ASIC), as HDLs podem ou não desempenhar um papel significativo no fluxo de fabricação. Quanto mais próximo do sistema físico, mais carregado com informações específicas de tecnologia fica o projeto. Essas informações específicas não ficam armazenadas em HDL, uma vez que ela é uma descrição genérica.



(a) Elaboração de um circuito com HDL

(b) Síntese da descrição de uma HDL

Figura 3: Processo envolvidos na criação de uma HDL para um sistema.

O primeiro passo da elaboração da descrição HDL é o levantamento do conjunto de requisitos do projeto ou um diagrama arquitetural de alto nível. Essas especificações serão então convertidos em requisitos funcionais (requisitos de sistema que especifica uma necessidade funcional que pode ser parte de uma funcionalidade que o sistema deve ser capaz de realizar, como adicionar um contato em um smartphone) e requisitos não funcionais (apesar de não ser uma funcionalidade, ainda é algo que precisa ser realizado para que a aplicação atenda seu propósito, como tempo máximo para que o processamento de um dado seja feito). O passo seguinte é definição comportamental do sistema. Esta envolve a relação entre eventos e estados (sequencial), relação das entradas e saídas (combinacional) e a máquina de estado do circuito, todas estas são geralmente prototipadas em forma de fluxograma ou diagrama de estados. O processo de transcrever um circuito para HDL é fortemente dependente da natureza do circuito e a forma como o projetista codifica. Como o HDL é um linguagem de descrição, até mesmo captura, geralmente começa-se com uma descrição algorítmica de alto nível. Para descrever estruturas repetitivas, normalmente, projetistas utilizam linguagens de script, como Perl ou Python.

Uma vez que a descrição em HDL está pronta, o próximo passo é revisar o código. O código pode tanto passar por uma auditoria, quanto por verificações automáticas. Esta última visa relatar desvios das diretrizes de código padronizadas, identificar possíveis

construções de código ambíguas que possam a vir ocasionar interpretações erradas e verificar erros comuns de codificação lógica, como portas flutuantes ou saídas em curto. Empresas como a Cadence Design Systems, empregam tecnologias para verificação de sistemas e são de grande importância nesse cenário. Esse passo é importante, uma vez que busca resolver erros de código antes mesmo que ele seja sintetizado. A descrição é compilada e o resultado é simulado e comparado com a descrição dos requisitos, caso esteja de acordo, vai para o próximo passo, caso contrário, o passo anterior deve ser refeito.

Com a descrição pronta, realiza-se o estágio de síntese, o qual pode ser visto na Figura 4. Esse passo é o responsável por mapear a descrição HDL em uma netlist. A síntese da descrição inicia-se com a síntese RTL. Ela envolve a conversão do circuito para módulos padrão primitivos, tais como somadores, comparadores, registradores, entre outros. No próximo passo ocorre a síntese lógica e é onde entra a dependência da plataforma, já que se emprega o uso de portas lógicas, LUTs (*Look Up Table* - tecnologia de FPGA), flip-flops, entre outros. O próximo passo é a otimização do circuito criado. Esse passo tem o foco de melhorar a eficiência do circuito, de tal forma com que o custo e desempenho do circuito criado seja compatível com o planejado. O resultado será a netlist do circuito. Após esse processo, ocorre mais uma simulação, todavia, dessa vez da netlist.

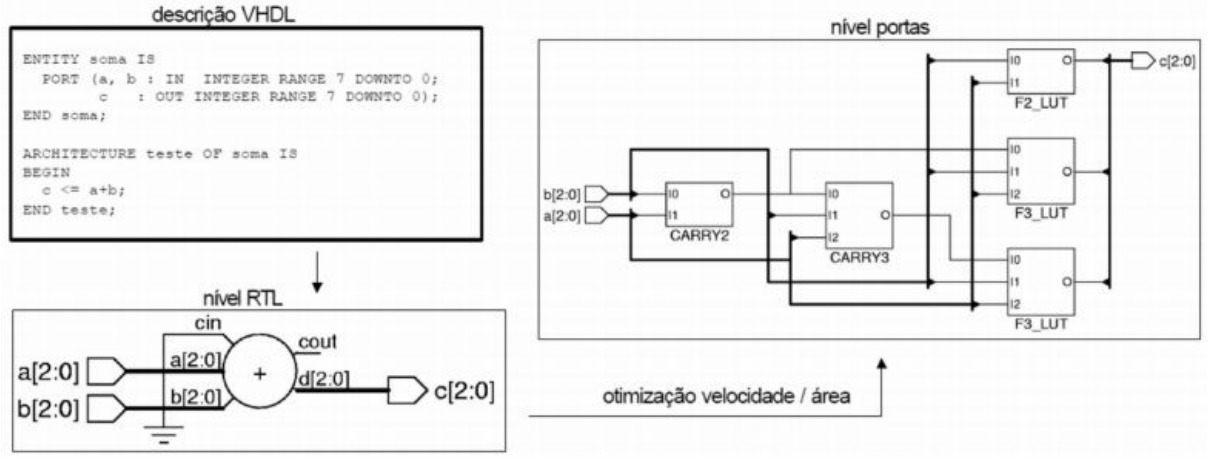


Figura 4: Processo de síntese de uma descrição HDL, neste caso, VHDL.

A capacidade de simular HDL é parte essencial para o processo de design HDL. Além de essencial é fundamental para o sucesso do design de HDL. Para a simulação, cria-se um modelo a partir da HDL. Isso favorece ao projetista desenvolver diversas variações de um projeto base e comparar o comportamento de cada na simulação. Além disso, facilita o processo de verificação do projeto, a qual valida as especificações iniciais pretendidas do projeto.

O processo de simulação de qualquer modelo HDL envolve a descrição de um ambiente de simulação de alto-nível criado pelo projetista (chamado de banco de testes). O banco de testes contém pelo menos uma instanciação do modelo HDL, declaração de sinais de I/O do modelo e uma forma de onda do clock. Todo o código do banco de testes é orientado a eventos. Todos os eventos ocorrem apenas nos instantes descritos no banco de testes, ou em reação a estímulos e eventos desencadeados. O simulador de HDL (executa o banco de

testes) mantém o relógio do simulador, a qual é a principal referência de todos os eventos de simulação. Atualmente, há implementações de simuladores de HDL que oferecem interfaces gráficas completas para usuário, além de um conjunto de ferramenta de depuração.

O processo de simulação também envolve a verificação do projeto. A verificação é uma parte longa do processo de design usando HDL, isso porque a especificação funcional do dispositivo final, mais a interpretação do projetista e a imprecisão da linguagem HDL não são totalmente conectadas e podem haver possíveis falhas em diversos pontos. A maioria do ciclo inicial de teste acontece no ambiente simulado, isso porque o início do projeto está sujeito a mudanças frequentes e importantes. Pode ocorrer a prototipação e testes em hardware, para esse cenário usa-se os FPGAs. Usar FPGAs é mais custosa do que somente a simulação de HDL, contudo, ela oferece uma visão real do design e é capaz de ser utilizada para verificar a interface com outros dispositivos de hardware, incluindo outros protótipos. Normalmente, esse tipo de prototipação tendem a ser mais rápidas do que a pura simulação do HDL, mesmo em FPGAs mais lentos.

Normalmente, quando se descreve o design a nível de Hardware, as duas etapas descritas (elaboração da descrição HDL e síntese da descrição criada) podem ser agrupadas em única, esta chamada de *front-end*. Essa etapa é a responsável por todo o processo de criação do hardware até antes da sua implementação em hardware, ela para na criação da netlist. O processo de implementar a netlist em hardware é conhecida como *back-end*. Nessa etapa, ocorre o posicionamento dos elementos descritos na netlist na plataforma e o roteamento através da matriz de interconexão. O resultado desse posicionamento então é testada e simulada novamente para checar se a temporização do circuito é precisa (ou está dentro do esperado) e para a detecção de bugs. A Figura 5 mostra esse cenário[1].

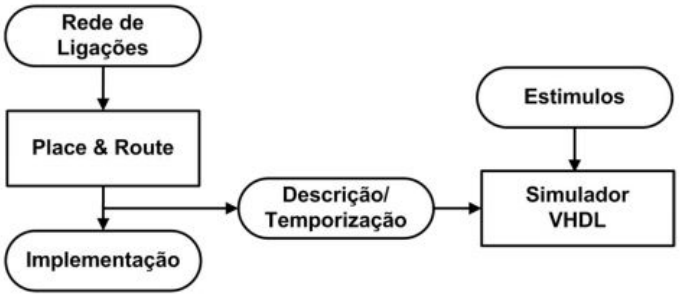


Figura 5: Implementação da HDL

O nível de abstração que as primeiras HDLs ofereciam eram comparados com as linguagens assembly. Com o intuito de facilitar e reduzir a complexidade da programação em HDLs, foi proposto a síntese de alto nível. Ele visa utilizar linguagens de alto nível como modelos de simultaneidade. Empresas como Cadence, Synopsys e Agility Design Solutions estão promovendo esse tipo de abordagem, a qual possibilita ciclos de projeto mais rápidos para FPGAs quando comparados as HDLs tradicionais. O foco é aumentar a produtividade dos engenheiros de hardware existentes, e não tornar os FPGAs mais acessíveis. A evolução da tecnologia possibilitou que módulos de hardware fossem projetados utilizando

[1] Vale ressaltar que todos os três cenários mostrados (Figura 3, 4 e 5) são interligados.

abstrações ainda maiores, por exemplo, as ferramentas Matlab/Simulink usando a ferramenta Mathworks HDL Coder ou o Xilinx System Generator (XSG) (antigo Accel DSP) da Xilinx ou LabView, que em conjunto com os compiladores adequados, oferecem a opção de criação de projetos usando diagramas de blocos.

A evolução das linguagens de descrição de hardware vêm possibilitando a evolução de várias áreas. Uma dessas áreas é a criação/evolução de SoCs (System-On-Chip), os quais são grandes e complexos sistemas em um único chip, tais como smartphones. Com a evolução dos SoCs e da HDL, entra em cena os PSoCs, os quais são SoCs programáveis. Em uma visão macro, é um misto de SoC com FPGA, os quais são muito versáteis. Exemplos de PSoCs são os PSoC® 5 da CyPress.

Atualmente, outros tipos de aplicação mais complexas vêm se favorecendo com a evolução das HDLs e a possibilidade que elas oferecem de criar sistemas mais completos e complexos sem grandes perdas de eficiência. Um exemplo são os sistemas ciber-físicos (Cyber Physical Systems - CPS). Os CPSs são a convergência entre computação, comunicação e controle, sendo a integração da computação com processos físicos. Acaba-se por utilizar os sistemas criados com HDL para interagir com processos físicos, os quais adicionam novas propriedades ao sistema.

Com a crescente evolução das HDLs e tecnologias como o FPGA, os SoCs também evoluem. Eles realizam todo o processamento e interface entre o mundo e o sistema digital de um CPS. São eles que oferecem a base necessária para o CPS desenvolver e também a Internet-das-coisas. Isso ocorre pois eles estão presentes em praticamente todos os dispositivos eletrônicos digitais que são usados diariamente, além de permitir a evolução de equipamentos que há anos não evoluíam (carros que estão com sistemas cada vez mais inteligentes). O CPS está um nível abaixo da Internet-das-coisas.

No contexto de Internet-das-coisas, os CPSs possuem um papel de destaque, uma vez que colaboram para construir sistemas inteligentes, como por exemplo as cidades inteligentes, as quais podem ser vistas como sistemas ciber-físicos em ampla escala. As cidades inteligentes possuem sensores monitorando indicadores virtuais e físicos e por meio de atuadores, muda dinamicamente o ambiente urbano complexo de alguma maneira. Os CPSs podem ser usados na produção a âmbito industrial para construir arquiteturas baseadas na internet que facilitam o controle remoto de sistemas de produção *stand-alone*. Os CPSs vêm se destacando em vários campos, desde governos, organizações até indústrias de tecnologia, os quais direcionam seus esforços para integrar diversos sistemas em redes inteligentes, a fim de, por exemplo, controlar uma rede de energia ou o tráfego de trânsito para torná-lo mais seguro.

A cada ano que passa, as HDLs ficam cada vez mais completas e oferecem novos recursos. Isso só é possível devido a evolução da tecnologia de hardware (FPGA, CPLD) e o poder de processamento que eles oferecem. Normalmente, o processo de design do hardware de um SoC envolve o uso de HDLs. A qualidade do projetista que utiliza as HDLs, causa um grande impacto na eficiência computacional da aplicação final. Por esse motivo, busca-se sempre a evolução das mesmas para facilitar e aumentar a produtividade dos projetistas.

Disponível em: https://github.com/plsilva/Files_DigitalSystems

https://stories.mlh.io/hardware-description-languages-what-are-they-and-why-should-i-care-307dd0b2ba9b

http://academic.csuohio.edu/chu_p/rtl/chu_rtL_book/silde/chap02_1.pdf

http://www.decom.ufop.br/alex/arquivos/sist_emb/Verilog.pdf

http://www.decom.ufop.br/alex/arquivos/sist_emb/HDLS.pdf

https://pt.wikipedia.org/wiki/Linguagem_de_descri%C3%A7%C3%A3o_de_hardware

https://en.wikipedia.org/wiki/Hardware_description_language

https://en.wikipedia.org/wiki/SystemVerilog

http://www.ec.ufc.br/professores/elmano/disciplinas/taed/aulas/Aula04.pdf