International Journal of Computer Architecture Education, 2025, 14:1
ISSN: 2316-9915 • doi: 10.5753/ijcae.2025.6542

ARTIGO DE PESQUISA

FPGA Unboxing: entendendo a arquitetura e as ferramentas de projeto para FPGAs

Deborah Caroline Rodrigues Oliveira [Universidade Federal de Ouro Preto | *deborah.rodrigues@aluno.ufop.edu.br*]
João Silva [Universidade Federal de Ouro Preto | *joao.cota@aluno.ufop.edu.br*]
José Augusto Miranda Nacif [Universidade Federal de Viçosa | *jnacif@ufv.br*]
Ricardo dos Santos Ferreira [Universidade Federal de Viçosa | *ricardo@ufv.br*]
Racyus Delano Garcia Pacífico [Universidade Federal Ouro Preto | *racyus.pacifico@ufop.edu.br*]

Departamento de Computação e Sistemas, Universidade Federal de Ouro Preto, Campus Universitário, João Monlevade, MG, 35931-008, Brasil.

Resumo. Dispositivos reconfiguráveis, como FPGAs, oferecem grande flexibilidade no desenvolvimento de soluções de hardware, porém seu ensino apresenta desafios devido à complexidade das ferramentas tradicionais e das linguagens de descrição de hardware (HDLs). Este trabalho propõe uma abordagem didática para introduzir as principais etapas do processo de síntese em FPGAs: mapeamento tecnológico, posicionamento, roteamento e geração do *bitstream*. Para alcançar esse objetivo, adotou-se uma metodologia com exemplos interativos, recursos visuais e analogias acessíveis para explicar cada fase, permitindo que o estudante compreenda a ocupação do arranjo de *Lookup Tables* (LUTs) e os processos internos do fluxo de compilação. Explorou-se o uso de modelos de linguagem de larga escala (LLMs) para apoiar a criação de ferramentas visuais em JavaScript, potencializando a aprendizagem ativa e interativa. A principal contribuição deste trabalho é o `unboxing` das ferramentas de FPGA, complementando materiais existentes e oferecendo aos iniciantes um caminho estruturado e intuitivo para compreender o funcionamento interno desses dispositivos e suas ferramentas.

Palavras-chave: FPGA, Verilog, Unboxing, LLMs, Ensino de hardware.

Recebido: 08 Setembro 2025 • **Aceito:** 01 Outubro 2025 • **Publicado:** 21 Janeiro 2026

1 Introdução

Com o avanço da computação e o desafio de desenvolver soluções de hardware, FPGAs destacam-se pela flexibilidade e alto poder computacional ao implementar soluções diretamente em hardware [Boutros and Betz, 2021]. Por serem reconfiguráveis após a fabricação, esses dispositivos aceleram o ciclo de desenvolvimento. O ensino de FPGAs em cursos de engenharia e ciência da computação possibilita o desenvolvimento de competência pelos estudantes [da Fonseca *et al*., 2021].

Embora relevante, gera dificuldade de compreensão para a maioria dos estudantes em relação ao funcionamento interno de FPGAs e o processo de mapeamento de um projeto nesses dispositivos. Isso ocorre devido à complexidade das ferramentas de síntese, ao nível de abstração das linguagens de descrição de hardware (HDLs) e à dificuldade dos ambientes de desenvolvimento. O ensino tradicional baseado exclusivamente em linguagens como VHDL e Verilog, dificulta a compreensão inicial. Embora as práticas com FPGAs favoreçam a aprendizagem ativa e eficaz, ainda há uma lacuna significativa no ensino, explicar as diferentes etapas de transformação da descrição de circuitos de hardware até a geração do *bitstream* [Navas-González *et al*., 2023].

FPGAs são formados por um arranjo de blocos reconfiguráveis. Cada bloco reconfigurável contém uma tabela verdade denominada *lookup table* (LUT). Uma LUT possui n entradas e armazena uma função lógica $f(x_1, \ldots, x_n)$. A LUT e vantajosa por permitir implementar qualquer função lógica com até n entradas. O primeiro passo, conhecido como mapeamento tecnológico, consiste em transformar o circuito lógico reescrevendo-o apenas com LUTs. Após descrito como um grafo de LUTs, o circuito lógico precisa ser mapeado no FPGA físico, que consiste em uma matriz de LUTs dispostas em linhas e colunas. A etapa seguinte é o posicionamento, no qual cada LUT do grafo é alocada em uma posição específica da matriz. O terceiro passo é interconectar as LUTs da matriz para implementar as arestas do grafo, conectando os fios entre as LUTs. A matriz do FPGA possui recursos programáveis, como ruas paralelas e cruzamentos, que devem ser configurados para implementar essas conexões. O quarto e último passo é a programação do FPGA, na qual os resultados do mapeamento, posicionamento e roteamento são convertidos nos bits de configuração do dispositivo, permitindo que ele execute a lógica desejada.

O objetivo deste trabalho é oferecer uma base para a compreensão das principais etapas do processo de mapeamento de um projeto de hardware em FPGA. Explora-se alternativas didáticas que simplifiquem esse processo, com exemplos iterativos e recursos visuais que permitem os estudantes compreenderem as tarefas realizadas pelas ferramentas de síntese. Investiga-se o uso de Modelos de Linguagem de Larga Escala (LLMs) como apoio pedagógico. Esses modelos oferecem tutorias automatizadas, geração dinâmica de código HDL, explicações conceituais e simulações educativas [Chu *et al*., 2025; Alsaqer *et al*., 2024; Thakur *et al*., 2024], podendo complementar o ensino tradicional e potencializar a aprendizagem.

A metodologia abordar cada etapa por meio de exemplos iterativos, permitindo aos estudantes a identificação das tarefas realizadas pelas ferramentas de FPGA, bem como compreendam sua complexidade. Explorou-se o uso de LLMs na geração de interfaces visuais interativas em *JavaScript*, para potencializar o processo de ensino. A transformação de uma descrição de hardware nos bits de configuração

de FPGAs envolve, as seguintes etapas: (i) mapeamento tecnológico; (ii) posicionamento; (iii) roteamento; e (iv) geração da configuração em formato binário, o *bitstream*.

As principais contribuições deste trabalho são:

- Abordagem `unboxing`, que explica, por meio de exemplos simples e recursos visuais, cada passo das ferramentas de FPGA até a geração do *bitstream*.
- Apresentação clara das etapas de mapeamento tecnológico, posicionamento, roteamento e geração do *bitstream*.
- Uso de uma metodologia baseada em exemplos iterativos, permitindo ao estudante identificar as tarefas executadas pelas ferramentas de FPGA.
- Desenvolvimento de um site com material introdutório que complementa o conteúdo já disponível em cursos e materiais online.
- Exploração de LLMs como ferramenta auxiliar para geração de interfaces visuais interativas em *JavaScript*, promovendo novas formas de aprendizagem.

Este artigo está organizado da seguinte forma: Seção 2 contextualiza conceitos fundamentais de FPGAs. Seção 3 descreve um *website* iterativo e introdutório sobre FPGAs. A Seção 4 apresenta como um circuito lógico combinacional de portas lógicas é transformado em um grafo de LUTs. As Seções 5 e 6 destacam como aprender sobre posicionamento e roteamento iterativo usando o Google Colab. A Seção 7 descreve os trabalhos relacionados. A Seção 8 aborda as conclusões do trabalho.

2 Fundamentos

Para contextualizar o trabalho, esta seção apresenta conceitos de uma arquitetura básica do FPGA (Seção 2.1) e o fluxo das etapas internas das ferramentas de FPGA (Seção 2.2). Embora esse conteúdo básico já esteja disponível em vários materiais didáticos, existem muitas lacunas sobre como cada parte funciona e quais são suas complexidades.

2.1 Arquitetura FPGA

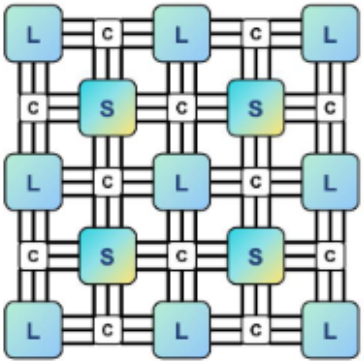

Figura 1. C: Conectores, L: LUTs e S: Switches.

A Figura 1 apresenta a arquitetura clássica dos FPGAs, ilustrando três componentes fundamentais: os blocos lógicos configuráveis (LUTs ou *Lookup Tables*), os segmentos de interconexão organizados em múltiplos barramentos, e os elementos programáveis de chaveamento (*switches*) que estabelecem as conexões entre os segmentos de fios.

A reprogramabilidade do FPGA manifesta-se em duas dimensões principais. Primeiro, na funcionalidade lógica: cada LUT pode ser reconfigurada para implementar qualquer função booleana com granularidade de bit, permitindo a implementação de circuitos digitais arbitrários. Segundo, na conectividade: os elementos de chaveamento podem ser reprogramados após a fabricação para estabelecer diferentes padrões de interconexão entre os segmentos de fios, determinando como os sinais são roteados através do dispositivo.

É fundamental distinguir que o roteamento em FPGAs é estático e baseado em circuitos dedicados, contrastando com o roteamento dinâmico por troca de mensagens encontrado em redes de computadores ou *Networks-on-Chip* (NoCs). Uma vez programado, cada caminho de sinal permanece fixo durante a operação do circuito.

2.2 Ferramentas

O processo das ferramentas para implementar um circuito lógico em FPGA (a partir da descrição do projeto) consiste nas seguintes etapas sequenciais [Chen and Chang, 2017], ilustrado na Figura 2:

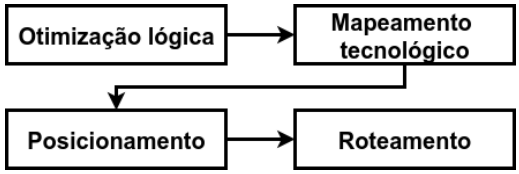

Figura 2. Processo ferramentas implementar circuitos lógicos em FPGA.

Otimização lógica: Constitui a primeira etapa, realizando a minimização das funções *booleanas* e *flipflops*, temporização e no uso de FPGA heterogêneos, de blocos de memória RAM e unidades lógico-aritméticas a nível de palavra. Esta fase visa otimizar área, atraso ou uma combinação de ambos, reduzindo a complexidade lógica e preparando o circuito para as etapas subsequentes.

Mapeamento tecnológico: Transforma as funções *booleanas* otimizadas, em geral representadas por um grafo, em um circuito composto por blocos lógicos específicos do FPGA. Esta etapa também realiza otimizações direcionadas, seja minimizando o número total de blocos lógicos necessários (otimização de área) ou reduzindo o número de blocos lógicos nos caminhos críticos temporais (otimização de atraso).

Posicionamento (*placement*): Seleciona a localização específica para cada bloco lógico dentro da matriz do FPGA. O objetivo principal desta etapa é minimizar o comprimento total das interconexões necessárias, considerando a proximidade física entre blocos que precisam se comunicar.

Roteamento (*routing*): Conecta os recursos de interconexão disponíveis no FPGA com os blocos lógicos distribuídos pela ferramenta de posicionamento. Esta etapa estabelece os caminhos físicos que transportam os sinais desde onde são gerados até onde são utilizados, completando a implementação física do circuito.

3 Desmistificando o FPGA

Como material introdutório, propôs-se um *Website* simples e interativo, desenvolvido sem a necessidade de codificação e baseado em descrições em linguagem natural, com apoio de recursos visuais e analogias. Esse material está descrito na Seção 3.1. A Seção 3.2 apresenta o conteúdo de outros *websites* de introdução ao FPGA, que são complementares à proposta apresentada. O objetivo é fornecer uma abordagem introdutória, permitindo ao estudante começar a construir os conceitos que servirão de base para a compreensão do *unboxing* das ferramentas de FPGA.

3.1 Primeiros passos em FPGA

Para desmistificar o universo dos FPGAs e torná-lo acessível a um público amplo, desde estudantes da área até entusiastas de tecnologia, foi desenvolvida uma ferramenta didática em formato de *website* interativo e autoexplicativo, ilustrada na Figura 3. A missão da ferramenta é demonstrar que, com o uso de analogias adequadas, é possível compreender o poder do hardware reconfigurável.

Figura 3. Página inicial do site.

A construção do *website* foi realizada a partir de uma plataforma *no-code*, ou seja, sem a necessidade de programação tradicional. A plataforma utilizada, foi o `Lovable.ai`, representado na Figura 4, ao qual facilita o desenvolvimento *web* ao empregar inteligência artificial para criar aplicações a partir de descrições em linguagem natural. Os usuários apenas descrevem a funcionalidade desejada, e a ferramenta gera o projeto automaticamente. Na Figura 5, observa-se a simplicidade da interface do `Lovable`.

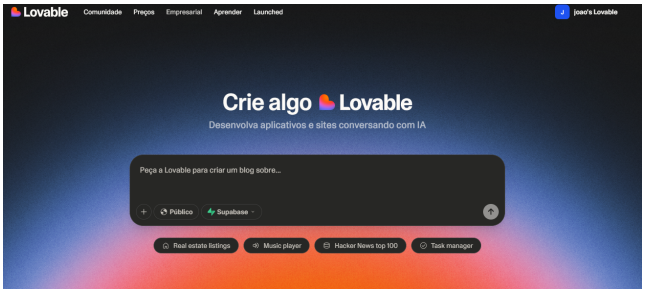

Figura 4. Página inicial do `Lovable.dev`.

Após a geração inicial, a plataforma permite que o desenvolvedor realize alterações, adicione imagens e vídeos, bem como integre APIs ou bancos de dados, oferecendo uma flexibilidade comparável à programação convencional. A versão gratuita da plataforma opera com um sistema de créditos, consumidos a cada comando enviado, mas renovados diariamente, o que permite a continuidade do desenvolvimento. Com a versão final do *website*, é possível publicá-lo com um domínio personalizado, utilizando a hospedagem fornecida gratuitamente, ou hospedá-lo em um servidor próprio.

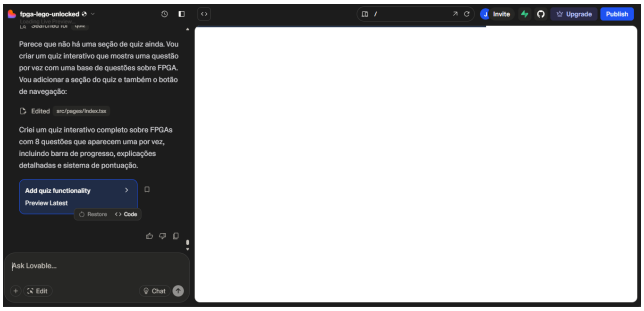

Figura 5. Interface do `Lovable.dev`.

A jornada de aprendizagem no *website* inicia-se com uma abordagem histórica sobre FPGA, detalhando os principais avanços e inovações ao longo dos anos. Após o contexto histórico, a ferramenta explora os componentes fundamentais para o funcionamento do FPGA. São abordados conceitos essenciais, como as portas lógicas AND, OR, NOT e XOR . A Figura 6 ilustra o exemplo da porta AND usando um exemplo interativo. Em seguida, o conteúdo avança para componentes complexos, como *flip-flops*, descrito como uma peça com memória, responsável por armazenar um estado (ligado ou desligado, 1 ou 0).

Figura 6. Exemplo porta lógica AND.

Também é apresentado o conceito de multiplexador, referido na ferramenta como *conector programado*, que, em uma analogia com trens, atua como um trocador de trilhos programável. Por fim, é apresentada a LUT, descrita como uma tabela mágica com a capacidade de implementar qualquer combinação de portas lógicas. Para facilitar a compreensão, a ferramenta estabelece comparações claras entre o FPGA e um processador comum (CPU), utilizando analogias diretas, como demonstrado nas Figuras 7 e 8. Uma dessas analogias é a da garagem: o processador é comparado a um carro (hardware), onde se pode trocar o motorista (software),

mas o veículo permanece o mesmo. O FPGA é análogo a um *Transformer*, que se molda e se transforma no veículo ideal para cada missão. Outra analogia é a da comunicação: para falar um novo idioma, a CPU utiliza um tradutor (uma camada de software intermediária), enquanto o FPGA se tornaria um falante nativo do idioma.

Figura 7. Comparador FPGA.

Figura 8. Comparador CPU.

O processo de reconfiguração do FPGA é detalhado em uma seção sobre as *Quatro fases da mágica*, que explica passo a passo como FPGAs são projetados e se transformam em um hardware específico para cada cenário. A ferramenta também contextualiza a aplicação prática de FPGAs, mostrando sua relevância em tecnologias, redes 5G, carros autônomos e entre outras aplicações. Por fim, para consolidar o conhecimento adquirido, a experiência de aprendizado é finalizada com um quiz de perguntas e respostas interativo, ilustrado na Figura 9. Este questionário avalia a compreensão do usuário sobre os temas abordados, e ao terminar, exibe a pontuação com o número de acertos e erros.

Figura 9. Quiz de perguntas e respostas.

3.2 Comparativo com abordagens de ensino atuais

A maioria dos sites de introdução ou ensino de FPGAs apresenta, em geral, uma perspectiva técnica, voltada para um público que já possui conhecimento prévio na área. Isso pode criar uma barreira de entrada para iniciantes. Essa é a principal diferença da abordagem proposta, que foi pensada desde o início para ser uma porta de entrada intuitiva, acessível e divertida para quem está começando.

Dentre os diversos sites disponíveis, pode-se destacar o `FPGA4Fun` por sua abordagem prática, oferecendo um repositório de projetos baseado na filosofia do *Aprender Fazendo*. A página inicial está exemplificada na Figura 10. O site utiliza projetos lúdicos e criativos, como gerar sinais de vídeo VGA ou criar pequenos jogos, para despertar o interesse dos estudantes dinamicamente, diferente dos exemplos teóricos tradicionais. Essa abordagem pressupõe que o usuário já possua, no mínimo, uma boa noção da área para compreender os detalhes de implementação e executar os exemplos em um FPGA.

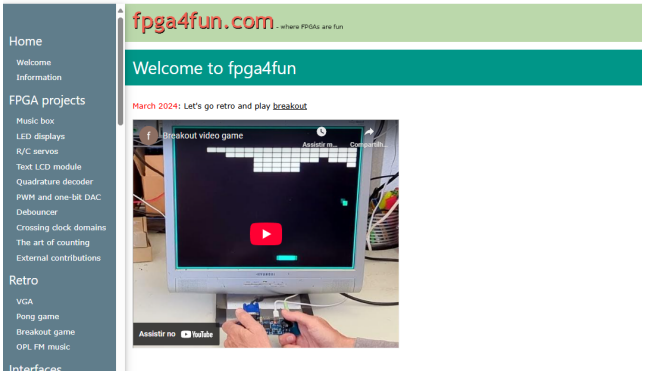

Figura 10. Página inicial do site `FPGA4Fun`.

Apesar de seu grande valor motivacional, a metodologia do `FPGA4Fun` apresenta limitações para um ensino formal e completo. Por se tratar de um compilado de projetos isolados, em vez de uma trilha de aprendizado estruturada, pode deixar os estudantes um pouco perdidos, sem um caminho claro a seguir. A ausência de um percurso definido e de mecanismos de avaliação formal dificulta a construção de um conhecimento sistemático. Portanto, o aprendizado, embora prático, corre o risco de tornar-se pontual e focado na replicação de projetos, em vez de fomentar uma compreensão profunda dos princípios de arquitetura dos FPGAs.

O site proposto constitui uma síntese inovadora dessas duas vertentes. Em vez de focar no desenvolvimento de projetos práticos elaborados, ele visa garantir uma compreensão fundamental dos conceitos, oferecendo interatividade e forte apelo visual por meio de analogias, ao mesmo tempo em que fornece um caminho de aprendizado guiado, com validação de conhecimento. Com os conceitos fundamentais introduzidos sobre os FPGAs, passou-se as etapas seguintes, para compreender a arquitetura interna em detalhes e fazer o `unboxing` dos principais passos das ferramentas de projeto.

4 Mapeamento em LUTs

Esta seção apresenta uma proposta de ensino do problema de mapeamento tecnológico de um circuito digital em um grafo de LUTs que são os blocos básicos do FPGA. O laboratório prático descrito nesta seção foi implementado no Google Colab, utilizando-se as linguagens Python e Verilog.

A proposta visa demonstrar que o primeiro passo da síntese é reescrever o circuito digital usando apenas LUTs. Destaca-se que, neste trabalho, considerou-se apenas a implementação de circuitos combinacionais.

Uma LUT é um bloco de memória reconfigurável de um bit de largura com n entradas. Ela possui 2^n linhas, formando uma tabela verdade de uma função $f(x_1, x_2, \ldots, x_n)$ de n variáveis. Assim, é capaz de implementar qualquer função lógica com até n variáveis.

O objetivo do laboratório é compreender como um circuito é descrito por LUTs e validar o mapeamento por meio da execução em Verilog. A LUT é o componente básico do FPGA, que consiste em um grande arranjo de milhares ou até milhões de LUTs. Para implementar as LUTs, utilizaram-se módulos de memória em Verilog. Para interligar as LUTs, empregaremos uma descrição estrutural em Verilog, com visualização do grafo de LUTs usando o comando *print_verilog* do pacote CAD4U [Canesche *et al.*, 2021].

O laboratório é composto por seis seções. A primeira seção configura o ambiente, realizando a instalação do Verilog e dos pacotes complementares no Google Colab [Canesche *et al.*, 2021]. A segunda seção apresenta um exemplo de um somador de dois números a e b, cada um de 3 bits, utilizando uma única memória Figura 11.

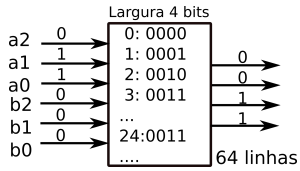

Figura 11. Somador de dois números de 3 bits implementado com um módulo de memória.

Qualquer circuito digital pode ser implementado usando memória; entretanto, o tamanho da memória cresce exponencialmente com o número de entradas. Os FPGAs resolvem esse problema utilizando módulos menores de memória interligados. Primeiro, ilustramos um exemplo simples, fácil de entender e verificar: a soma de dois números a e b. Em seguida, mostraremos como esse exemplo pode ser refinado, substituindo um módulo maior por vários menores interligados.

A Figura 11 mostra a implementação usando uma única memória, onde o endereço é formado pela concatenação das entradas a e b. O endereço 1 corresponde a $a = 0$ e $b = 1$, portanto o conteúdo armazenado que representa a soma será $a + b = 0 + 1 = 1$ ou 0001 em binário. Já o endereço 24 (ou 011000 em binário) corresponde a $a = 011 = 3$ e $b = 000 = 0$, portanto a soma será $3+0 = 3$, como ilustrado na Figura 11 e na Tabela 1.

As seções três e quatro do laboratório apresentam exemplos utilizando LUTs de 3 variáveis, equivalentes a memórias de 8 linhas com 1 bit cada. A terceira seção demonstra como

Tabela 1. Exemplo de mapeamento da soma $a + b$ em memória

Endereço (binário)	a (binário)	b (binário)	$a + b$ (binário)
000000	000	000	0000
000001	000	001	0001
000010	000	010	0010
001000	001	000	0001
011000	011	000	0011

construir a tabela verdade de um multiplexador 2:1, que é um circuito de 3 variáveis. A quarta seção ilustra como montar o mesmo circuito do somador de 3 bits apresentado na seção dois, agora utilizando LUTs de 3 entradas.

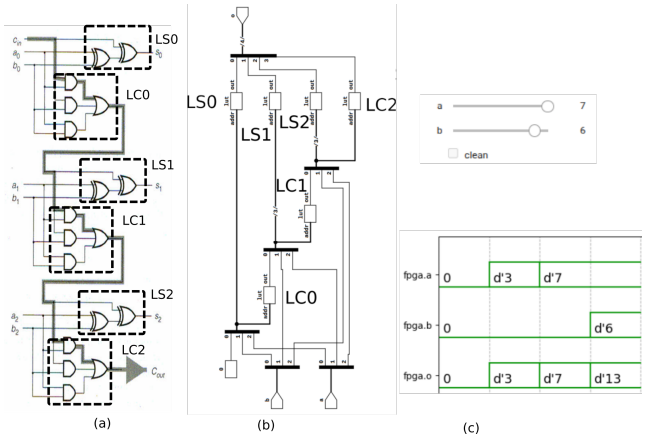

Figura 12. Somador de dois números de 3 bits com 6 LUTs: (a) Cobertura das portas lógicas; (b) Circuito gerado pelo *print_verilog*; (c) Simulação.

O primeiro passo é o mapeamento do circuito, substituindo conjuntos de portas lógicas com até 3 entradas por uma LUT. A Figura 12(a) mostra um somador *ripple carry* de 3 bits, onde as portas lógicas podem ser cobertas por 6 LUTs destacadas com retângulos pontilhados. A Figura 12(b) ilustra a saída do comando *print_verilog*, que gera o esquemático a partir do código Verilog.

Foram utilizados dois tipos de LUTs: LS_i, que implementa a função soma de 3 bits, e LC_i, que implementa a função do vai-um (*carry*). Este exemplo demonstra que é possível decompor uma tabela em um grafo de tabelas menores e mostra como as LUTs substituem as portas lógicas. Em outras palavras, qualquer circuito digital pode ser implementado como um grafo de LUTs.

```
module lut (input [2:0] addr, output out);
parameter filename = "filename.txt";
  reg [7:0] memory [0:7]; // *** MEMORY ***
  // READ
  assign out = memory[addr];
endmodule
module fpga(input [2:0] a,b, output [3:0] o);
wire [2:0] c;
lut LS0({a[0],b[0],1'b0},o[0]);
lut  LS1({a[1],b[1],c[0]},o[1]);
lut  LS2({a[2],b[2],c[1]},o[2]);
lut  LC0({a[0],b[0],1'b0},c[0]);
lut  LC1({a[1],b[1],c[0]},c[1]);
lut  LC2({a[2],b[2],c[1]},o[3]);
endmodule
```

Listagem 1: Somador de 3 bits com 6 módulos de LUT-3 descrito em Verilog.

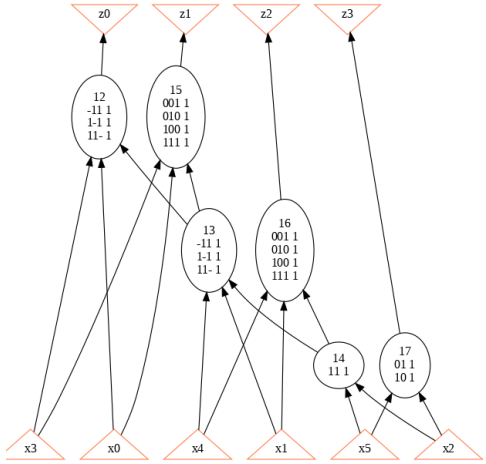

Figura 13. Somador de dois números de 3 bits com 6 LUTs de 3 entradas (k=3) gerado pelo ABC.

O código Verilog correspondente está apresentado na Listagem 1, enquanto a Figura 12(c) exibe o resultado da simulação realizada em Verilog. Para facilitar o uso, empregou-se uma interface em Python, na qual o *testbench* é gerado dinamicamente a partir de botões deslizantes (*sliders*) que permitem inserir os valores de a e b. No exemplo, foram aplicados os seguintes estímulos: $0+0$, $3+0$, $7+0$ e $7+6$, com as respectivas respostas 0, 3, 7 e 13.

A quinta seção apresenta um terceiro exemplo de circuito: um contador de bits (*bitcount*). A entrada é um vetor de 5 bits, e o circuito conta quantos desses bits são iguais a 1, resultando em valores entre 0 e 5. Esse circuito pode ser implementado com 8 LUTs, aproveitando as LUTs de soma e de propagação de vai-um.

A última seção trata de um caso avançado, no qual utilizamos o software de síntese lógica ABC [Brayton and Mishchenko, 2010] para ler qualquer circuito e mapeá-lo em um grafo de LUTs. Apresentamos dois exemplos: o somador de dois números de 3 bits e um selecionador de prioridade com frutas, este último ilustrando também a codificação de objetos e seu mapeamento em circuitos.

Primeiro, gerou-se a tabela verdade no formato PLA utilizando-se um código em Python. O formato PLA, usado pelo *Espresso*, descreve cada função lógica como uma lista de termos do tipo produto. Cada linha contém, na primeira coluna, os valores das variáveis de entrada (com `0`, `1` ou `-` para *don't care*) e, após um espaço, uma coluna que especifica quais funções de saída são ativadas por aquele termo.

Considere duas entradas (a, b) e duas saídas (f_1, f_2). O formato PLA para este caso pode ser descrito como:

```
.i 2
.o 2
00 10
01 01
1- 11
.e
```

Nesse exemplo:

- A primeira linha (`00 10`) significa $a = 0, b = 0$, e ativa apenas f_1.
- A segunda linha (`01 01`) significa $a = 0, b = 1$, e ativa apenas f_2.

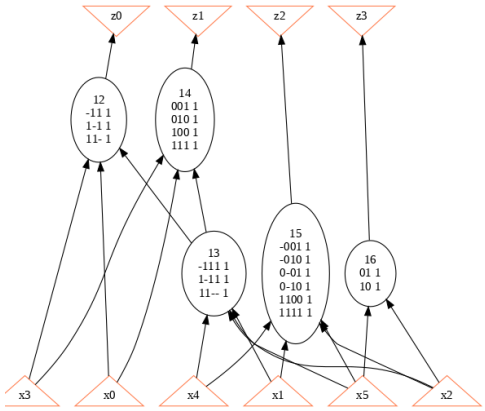

Figura 14. Somador de dois números de 3 bits com 6 LUTs de 4 entradas (k=3) gerado pelo ABC.

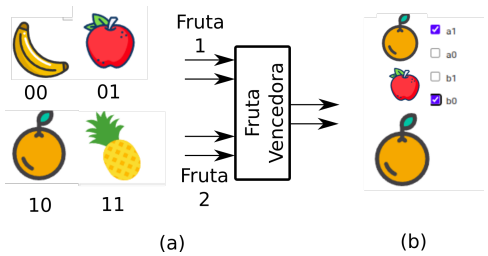

Figura 15. Seletor de frutas com prioridade.

- A terceira linha (`1- 11`) significa $a = 1$, b indiferente, e ativa f_1 e f_2.

Em seguida, um *script* no ABC faz a leitura do arquivo PLA e gera a descrição equivalente em LUTs com K entradas. Como exemplo, utilizou-se o caso $K = 3$, no qual o ABC encontra uma implementação com apenas 6 LUTs, representada como um grafo ilustrado na Figura 13. Nos vértices do grafo observa-se as funções que programam as LUTs internamente.

O interessante é que o ABC possui diversas opções de otimização e mapeamento. Basta alterar o valor de $K = 4$ para obtermos um novo circuito com LUTs de 4 entradas, ilustrado na Figura 14. Os FPGA comerciais geralmente utilizam valores de K variando entre 4 e 6, e algumas famílias permitem LUTs de até 8 entradas.

O segundo exemplo é o seletor de frutas: banana, maçã, laranja e abacaxi, codificados como 00, 01, 10 e 11, respectivamente. Nesse seletor, a banana possui a menor prioridade em caso de empate, enquanto o abacaxi possui a maior. A Figura 15 mostra tanto a codificação das frutas quanto o circuito seletor, que recebe duas entradas (a e b) e produz como saída a fruta vencedora, de acordo com a prioridade definida.

No exemplo, a entrada a recebe uma laranja e a entrada b uma maçã. Como a laranja tem prioridade sobre a maçã, ela aparece na saída. O estudante deve primeiro codificar a tabela verdade correspondente e, em seguida, usar o ABC para gerar o circuito e as LUTs.

Neste exemplo, realizou-se primeiro o mapeamento no ABC para obter o circuito com portas lógicas. Em seguida, solicitou-se ao ABC o mapeamento em LUTs de 3 entradas, isto é, com $K = 3$. O resultado pode ser observado na Figura 16(a), que mostra as portas lógicas. Utilizou-se cores para indicar como cada porta é mapeada em uma LUT.

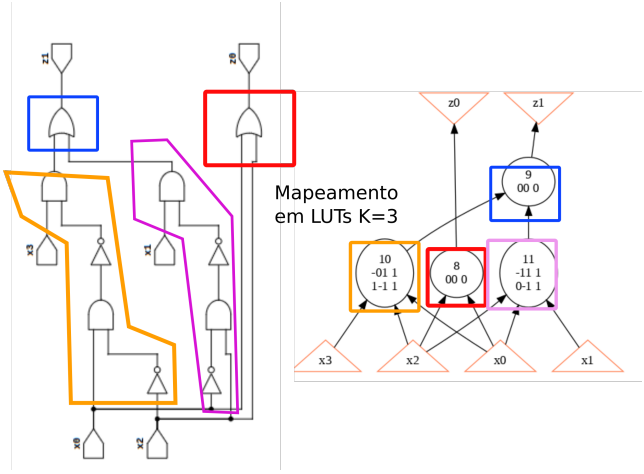

Figura 16. Seletor de frutas com prioridade: (a) Circuito com portas; (b) Mapeamento em LUT K=3.

Observa-se que foram utilizadas duas LUTs com $K = 3$ e duas LUTs com $K = 2$, pois nem sempre é necessário empregar o valor máximo de K.

O problema de mapeamento é NP-completo, e o ABC aplica heurísticas que realizam tanto a minimização quanto a cobertura do grafo durante o processo de mapeamento.

Este laboratório ilustra o processo de geração do circuito e seu mapeamento em LUTs. Com exemplos didáticos que utilizam poucas LUTs, o estudante já pode perceber como a complexidade do problema cresce à medida que o circuito se torna maior. Para casos com milhares ou até centenas de milhares de LUTs, torna-se claro por que o tempo de execução das ferramentas aumenta significativamente.

Nas próximas seções, será apresentado o processo de mapeamento das LUTs do grafo em LUTs físicas dentro do FPGA. Para simplificar a explicação, trabalhou-se com grafos reduzidos e simples que os circuitos reais, que podem conter LUTs com muitas entradas. O objetivo é compreender o processo de mapeamento físico, que envolve as etapas de posicionamento e roteamento.

5 Posicionamento

Primeiro, apresentou-se o problema de posicionamento, que dá sequência ao mapeamento tecnológico. O grafo de LUTs, obtido a partir do mapeamento, precisa ser posicionado em uma matriz de LUTs dispostas em linhas e colunas, buscando minimizar as distâncias entre os nós conectados. Destaca-se o problema de posicionamento é *NP-completo*, o que significa que, para projetos maiores, a busca por uma solução ótima pode levar de minutos a horas. Mesmo trabalhando com exemplos de grafos simples em pequenas matrizes de LUTs, o estudante começa a compreender como o arranjo de LUTs que constitui o FPGA é ocupado para implementar um circuito.

Para facilitar a compreensão dos processos de posicionamento e roteamento em matrizes FPGA, desenvolveu-se uma série de ferramentas didáticas que rodam no sobre o Google Colab. A ideia principal consistiu em criar um ambiente interativo que permite estudantes visualizar, na prática, como esses processos funcionam internamente em FPGAs.

Para construir as ferramentas visuais, utilizou-se o modelo de linguagem ChatGPT-4.0, que auxiliou na geração dos códigos. Todo o projeto foi implementado no Google Colab, com código em *JavaScript* e o HTML/CSS para estruturar e estilizar a interface de maneira responsiva, garantindo desempenho suficiente para execução diretamente nos navegadores.

A ferramenta de posicionamento funciona da seguinte maneira: primeiro, é gerado um grafo com nós distribuídos aleatoriamente. Cabe ao estudante posicionar esses nós manualmente sobre a matriz do FPGA, utilizando as LUTs disponíveis. A cada posicionamento, calcula-se a distância de Manhattan entre os nós conectados, o que permite avaliar a eficiência da alocação do grafo. Quando a posição de dois nós adjacentes no grafo está correta, a aresta correspondente muda para a cor amarela, sinalizando ao estudante que aquelas LUTs estão posicionadas para minimizar a distância entre elas.

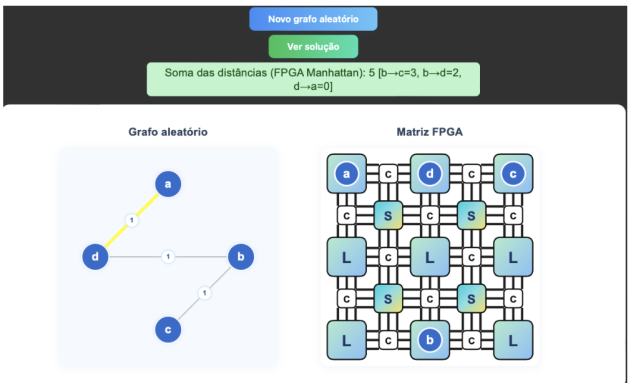

Figura 17. Exemplo da interface da ferramenta de posicionamento mostrando a aresta $a \rightarrow d$ na cor amarela na matriz FPGA com distância de Manhattan igual a zero.

Essa abordagem interativa, aliada ao uso de ferramentas visuais e assistência de modelos de linguagem, permite que o estudante compreenda intuitivamente a complexidade do problema de posicionamento e comece a construir uma base sólida para entender o roteamento e a ocupação da matriz de LUTs em projetos reais de FPGA.

No exemplo da Figura 17, observa-se um grafo simples com quatro nós (a, b, c e d) e suas conexões, posicionado sobre uma matriz de LUTs de uma FPGA.

A aresta entre os nós a e d está colorida de amarelo, indicando que essas duas LUTs são vizinhas na matriz e, portanto, a distância de Manhattan entre elas é mínima, contribuindo para um custo total de conexão eficiente. As demais arestas, como $b{-}c$ e $b{-}d$, não estão amarelas, o que significa que os pares de LUTs correspondentes não são vizinhos e a distância de Manhattan ainda é maior, aumentando o custo de interconexão.

O estudante deve interagir com a ferramenta arrastando e reposicionando manualmente as LUTs b e c sobre a matriz FPGA, buscando reduzir a distância das conexões. O objetivo é fazer com que todas as arestas do grafo fiquem amarelas, ou seja, que todos os nós conectados sejam posicionados como vizinhos na matriz, minimizando o custo total do posicionamento. Essa atividade permite compreender na prática como o problema de posicionamento influencia a eficiência do arranjo de LUTs em FPGAs. A Figura 18 ilustra o exemplo do grafo com o posicionamento ótimo na matriz de LUTs

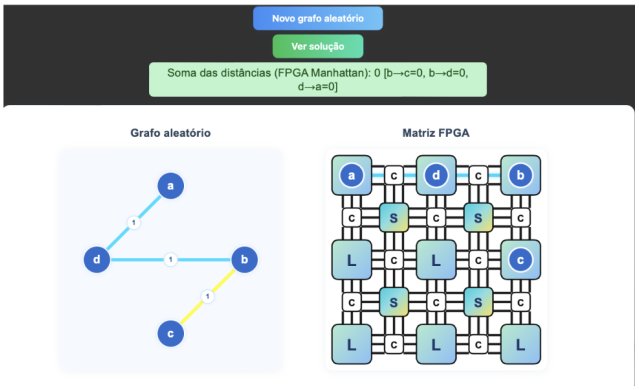

Figura 18. Posicionamento otimizado com distância de Manhattan igual a zero.

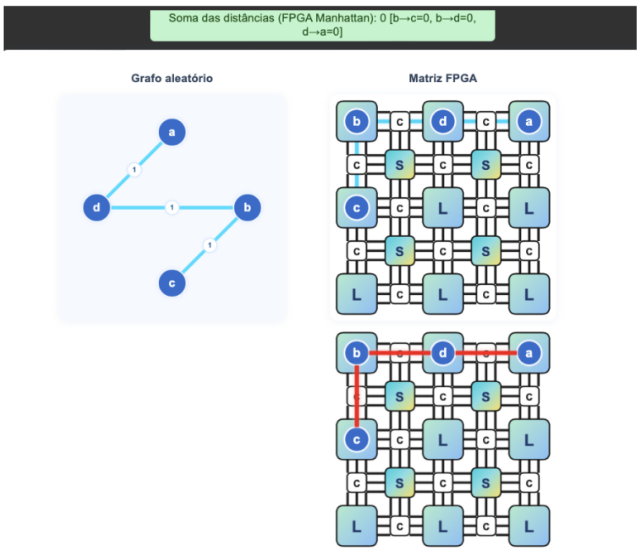

Figura 19. Posicionamento e roteamento dos nós na matriz FPGA com resolução visual.

simplificada para introduzir a arquitetura física do FPGA.

6 Roteamento

Depois do posicionamento, o usuário pode começar o roteamento, esse processo é feito desenhando trilhas de conexão sobre a trilha central da matriz, arrastando o mouse para criar os caminhos, as conexões podem ser feitas passando por conectores do tipo C e *switches* S, ou diretamente entre os conectores e as LUTs que contêm os nós, quando uma conexão é feita corretamente, a aresta no grafo muda para azul, indicando que o roteamento está de acordo com o grafo original. É possível apagar qualquer trilha criada clicando duas vezes sobre ela, o sistema impede que o usuário crie conexões em LUTs vazias, o que obriga a seguir uma lógica organizada.

O objetivo principal dessa ferramenta é permitir que o usuário consiga, a partir de um grafo aleatório, realizar tanto o posicionamento quanto o roteamento dos nós dentro da FPGA, sempre respeitando as ligações originais e buscando a menor distância de Manhattan quando possível igual à zero. A aplicação ainda conta com botões para gerar novos grafos e para visualizar possíveis soluções, tanto do posicionamento quanto do roteamento.

Desse modo, ao final do processo de roteamento, o usuário consegue visualizar a correspondência entre as conexões do grafo original e as trilhas traçadas na matriz FPGA, respeitando a lógica estrutural do sistema e as restrições de conectividade impostas pelo ambiente simulado. A mudança de cor das arestas para azul funciona como indicativo visual da conformidade entre o grafo e o circuito desenhado, permitindo ao estudante verificar imediatamente a correção das conexões estabelecidas, essa etapa finaliza a proposta didática da aplicação, consolidando-se os conhecimentos sobre o funcionamento interno de uma FPGA por meio de uma experiência interativa e progressiva.

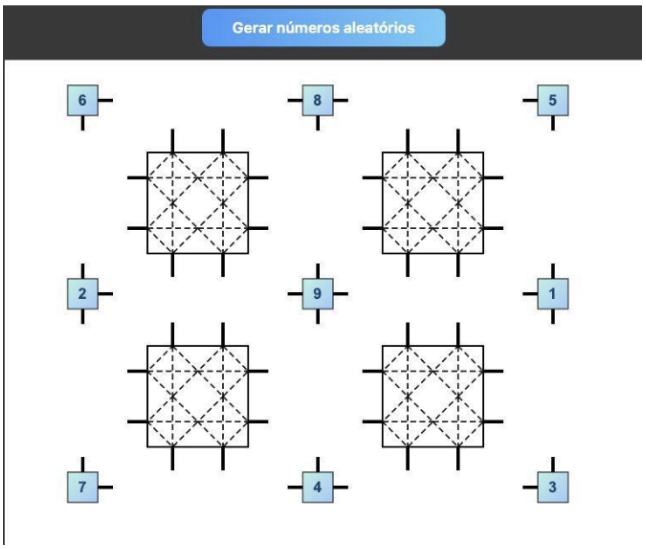

Figura 20. Interface proposta para o módulo de roteamento via *switch boxes*.

Como melhoria futura, pretende-se implementar na ferramenta um módulo interativo dedicado ao roteamento por meio dos *switch boxes* presentes na matriz FPGA, permitindo ao usuário a compreensão prática de como esses elementos viabilizam a interligação entre pistas horizontais e verticais. Diferentemente do roteamento atual, que conecta diretamente as LUTs ou passa por conectores e *switches* simplificados, essa nova etapa possibilitará explorar a flexibilidade do *switch box* e seu papel na escolha de caminhos alternativos para otimizar a conectividade, reduzindo o congestionamento e mantendo a eficiência do circuito.

A funcionalidade proposta consistirá na criação de atividades em que o estudante deverá conectar pontos específicos utilizando unicamente as rotas disponíveis através dos *switch boxes*, respeitando as restrições arquiteturais do FPGA. Serão representadas topologias distintas como planar e Wilton que influenciam diretamente a capacidade de completar as conexões, estimulando o raciocínio sobre como a escolha da topologia e do caminho impacta no resultado final. Ao propor diferentes cenários com graus variados de complexidade, será possível aproximar a experiência do aluno das condições encontradas em arquiteturas reais, onde a tomada de decisão no roteamento influencia tanto o desempenho quanto a viabilidade do projeto.

Essa abordagem permitirá também simular situações de congestionamento, em que determinados caminhos estarão indisponíveis, obrigando o usuário a buscar alternativas por meio de trilhas menos diretas. Com isso, será possível demonstrar claramente como a flexibilidade (Fs) dos *switch*

boxes afeta a capacidade de roteamento e como técnicas de busca, como as utilizadas em algoritmos do tipo *Maze routing* ou *Pathfinder*, podem encontrar a solução ótima. Ao final de cada desafio, a ferramenta poderá apresentar o trajeto correto e compará-lo com a solução encontrada pelo usuário, reforçando o aprendizado e consolidando-se a compreensão sobre a importância dos *switch boxes* no processo de interconexão do FPGA.

7 Trabalhos relacionados

O ensino tradicional de FPGAs consolidou-se como prática central na formação em engenharia e ciência da computação, mas ainda apresenta limitações que dificultam a aprendizagem prática. A necessidade de dominar linguagens de descrição de hardware, como VHDL e Verilog, impõe uma curva de aprendizado acentuada aos estudantes, que frequentemente enfrentam obstáculos com a sintaxe e com o alto nível de abstração exigido no design digital. Além disso, a dependência de infraestrutura laboratorial com placas físicas restringe o acesso contínuo às atividades experimentais, o que pode comprometer o envolvimento dos alunos e dificultar a consolidação de conhecimentos práticos.

Enquanto esse cenário evidencia a complexidade do ensino convencional, surgiram iniciativas que buscam simplificar e democratizar o aprendizado de hardware reconfigurável. O MiniFPGA [Moreno-Villalón *et al.*, 2014] representa uma dessas tentativas, oferecendo um ambiente móvel para a prática de conceitos de particionamento, roteamento e uso de LUTs em arquiteturas FPGA reduzidas. Essa proposta destaca-se por dispensar recursos laboratoriais físicos, favorecendo um aprendizado acessível e interativo. Entretanto, por não ter sido projetada para escalabilidade e por ter sido descontinuada, sua aplicação prática permaneceu restrita.

Em contraste, o VPR [VPR, 2025], componente central do fluxo Verilog To Routing, adota uma abordagem mais aprofundada e técnica. Sua interface gráfica baseada na biblioteca EZGL possibilita a visualização detalhada da arquitetura FPGA, do floorplan e dos estágios de posicionamento e roteamento. Embora esse ambiente favoreça a análise visual e a depuração do fluxo CAD, ele se mantém voltado à observação e não à experimentação direta, já que não permite ao usuário executar manualmente o posicionamento ou o roteamento. Assim, o VPR atua de maneira complementar às abordagens educacionais ao oferecer uma visão técnica avançada, porém sem a interatividade necessária ao ensino introdutório.

De forma semelhante à busca por maior interação, o DigitalJS [Materzok, 2019] emprega JavaScript para criar uma interface gráfica que permite simular códigos em Verilog diretamente no navegador, favorecendo a visualização de circuitos digitais. Essa iniciativa aproxima-se de propostas que exploram aspectos lúdicos, como o uso de programação em blocos para o ensino de sistemas lógicos [Castro and Azevedo, 2020]. Ainda que essas ferramentas ampliem a acessibilidade do ensino de lógica digital, seu foco principal permanece em conceitos de nível de porta, sem abordar as etapas internas do fluxo de síntese de FPGAs.

De modo complementar, trabalhos mais recentes [Canesche *et al.*, 2021; Ferreira *et al.*, 2024a,b] exploram o Google Colab como ambiente de aprendizagem ativa para o ensino de sistemas digitais e arquitetura de computadores, utilizando linguagens HDL e simulações interativas. Apesar de promoverem autonomia e experimentação, essas iniciativas não tratam diretamente da estrutura interna dos FPGAs, o que abre espaço para propostas que tornem esse conhecimento mais acessível e visual.

Por outro lado, a introdução dos Modelos de Linguagem de Larga Escala (LLMs) vem transformando o modo como o ensino de linguagens HDL é conduzido. Ferramentas baseadas em LLMs permitem que estudantes descrevam circuitos em linguagem natural e recebam o código correspondente em Verilog ou VHDL, encurtando a distância entre a formulação da ideia e sua implementação [Chu *et al.*, 2025; Thakur *et al.*, 2024]. Essa abordagem automatizada acelera o processo de aprendizagem e reduz a dependência da familiaridade com a sintaxe tradicional.

Além disso, os LLMs têm sido aplicados como agentes tutoriais capazes de oferecer acompanhamento personalizado, simulações interativas e explicações adaptadas ao perfil do estudante. Enquanto iniciativas anteriores focavam em ambientes estáticos ou parcialmente guiados, os LLMs proporcionam uma aprendizagem dinâmica e contínua, com feedback imediato e potencial de personalização. Estudos recentes [Rama and Truong, 2024; Du *et al.*, 2023] apontam que, quando utilizados em ciclos iterativos de validação, esses modelos obtêm resultados consistentes até mesmo em aplicações embarcadas em FPGAs. Dessa forma, observa-se uma convergência entre metodologias tradicionais de ensino e novas abordagens mediadas por inteligência artificial, fortalecendo o movimento em direção a experiências educacionais mais acessíveis, interativas e integradas.

8 Conclusão

Através deste estudo, conclui-se que o FPGA Unboxing representa uma contribuição significativa para o ensino introdutório de FPGAs, ao tornar compreensíveis, interativamente e acessível, as principais etapas do fluxo de síntese mapeamento tecnológico, posicionamento, roteamento e geração do bitstream. Constatou-se que a abordagem proposta cumpre o propósito de revelar os mecanismos internos tradicionalmente ocultos pelas ferramentas de desenvolvimento, proporcionando aos estudantes uma compreensão prática e visual sobre o funcionamento do hardware reconfigurável.

Conclui-se que a estratégia didática adotada, baseada em exemplos iterativos, analogias e recursos visuais, contribui para reduzir a complexidade percebida pelos iniciantes, promovendo uma aprendizagem ativa e significativa. Verificou-se que a utilização do Google Colab como ambiente de execução facilitou o acesso às atividades práticas, permitindo que os estudantes experimentassem o processo de síntese e implementação em FPGAs. Concluiu-se também que a integração de modelos de linguagem de larga escala (LLMs) demonstrou ser uma ferramenta eficaz na geração automática do código da interface gráfica da aplicação, tornando o desenvolvimento ágil e ampliando o caráter interativo das experiências propostas. Essa integração reforça o potencial da inteligência artificial como suporte pedagógico

na criação de recursos voltados ao ensino de hardware.

Apesar das conclusões alcançadas a partir da análise da literatura e dos resultados obtidos com a ferramenta, pretende-se, em estudos futuros, aplicar o FPGA Unboxing em contextos reais de ensino, buscando coletar feedback de usuários e aprofundar a investigação sobre seus impactos pedagógicos e sua eficácia na aprendizagem prática.

Por fim, compreende-se que o desenvolvimento deste trabalho consolidou o entendimento sobre a relação entre teoria e prática no ensino de sistemas digitais, evidenciando que abordagens visuais e interativas podem contribuir para o fortalecimento da autonomia e da capacidade analítica dos estudantes. Ainda que de caráter exploratório, o trabalho estabelece uma base concreta para investigações futuras voltadas à validação empírica da metodologia, com vistas a mensurar seu impacto pedagógico em contextos reais de ensino de engenharia.

Declarações complementares

Agradecimentos

Gostaríamos de agradecer as instituições de financiamento pelos recursos disponibilizados para o desenvolvimento das atividades desta pesquisa.

Financiamento

Apoio financeiro do Projeto FAPEMIG APQ-01577-22, CNPq e CAPES.

Contribuições dos autores

A autora Deborah Oliveira desenvolveu o ambiente interativo no Google Colab que ensina sobre as etapas dos FPGAs e descreveu parte da redação do texto. O autor João Silva implementou o *website* e realizou parte da redação do texto. Os autores Ricardo Ferreira, José Augusto Nacif e Racyus Delano colaboraram com sugestões e redação do texto.

Conflitos de interesse

Os autores declaram não haver conflitos de interesse.

Disponibilidade de dados e materiais

As ferramentas desenvolvidas no âmbito deste trabalho são de código aberto e estão disponíveis no link `https://colab.research.google.com/drive/1eIXibxbwpLQJV1J91o9lD99F9sw0zoh9`

Outras informações relevantes

O texto deste artigo é de responsabilidade dos autores, onde ferramentas de IA foram usadas para gerar interfaces em *JavaScript*, o site, revisão ortográfica e gramatical.

Referências

Alsaqer, S., Alajmi, S., Ahmad, I., and Alfailakawi, M. (2024). The potential of llms in hardware design. *Journal of Engineering Research*. DOI: 10.1016/j.jer.2024.08.001.

Boutros, A. and Betz, V. (2021). Fpga architecture: Principles and progression. *IEEE Circuits and Systems Magazine*, 21(2):4–29. DOI: 10.1109/mcas.2021.3071607.

Brayton, R. and Mishchenko, A. (2010). Abc: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification*, pages 24–40. Springer. DOI: 10.1007/978-3-642-14295-6$_5$.

Canesche, M., Bragança, L., Neto, O. P. V., Nacif, J. A., and Ferreira, R. (2021). Google colab cad4u: Hands-on cloud laboratories for digital design. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE. DOI: 10.1109/iscas51556.2021.9401151.

Castro, L. and Azevedo, R. (2020). Circuitly: A visual and constructive framework for teaching digital circuits. *International Journal of Computer Architecture Education*, 9(1):10–15. DOI: 10.5753/ijcae.2020.4839.

Chen, S.-C. and Chang, Y.-W. (2017). Fpga placement and routing. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 914–921. IEEE. DOI: 10.1109/iccad.2017.8203878.

Chu, Z., Wang, S., Xie, J., Zhu, T., Yan, Y., Ye, J., Zhong, A., Hu, X., Liang, J., Yu, P. S., *et al.* (2025). Llm agents for education: Advances and applications. *arXiv preprint arXiv:2503.11733*. DOI: 10.18653/v1/2025.findings-emnlp.743.

da Fonseca, M. H. K., da Silva Miranda, R., dos Santos Lima, T. H., and Pires, R. (2021). Controle de placa didática com simultaneidade de processos utilizando fpga. *REGRASP-Revista para Graduandos/IFSP-Câmpus São Paulo*, 6(2):117–137. Available at: `https://regrasp.spo.ifsp.edu.br/index.php/regrasp/article/view/714`.

Du, Y., Liew, S., Chen, K., and Shao, Y. (2023). The power of large language models for wireless communication system development: A case study on fpga platforms. arxiv. *arXiv preprint arXiv:2307.07319*. DOI: 10.48550/arXiv.2307.07319.

Ferreira, R., Canesche, M., Jamieson, P., Neto, O. P. V., and Nacif, J. A. (2024a). Examples and tutorials on using google colab and gradio to create online interactive student-learning modules. *Computer Applications in Engineering Education*, 32(4):e22729. DOI: 10.1002/cae.22729.

Ferreira, R., Sabino, C., Canesche, M., Neto, O. P. V., and Nacif, J. A. (2024b). Aiot tool integration for enriching teaching resources and monitoring student engagement. *Internet of Things*, 26:101045. DOI: 10.1016/j.iot.2023.101045.

Materzok, M. (2019). Digitaljs: A visual verilog simulator for teaching. In *Proceedings of the 8th Computer Science Education Research Conference*, pages 110–115. DOI: 10.1145/3375258.3375272.

Moreno-Villalón, A., Guerra-Martin, A., and Boemo, E. (2014). Minifpga: An educational app for teaching partitioning, placemnent and routing on andriod devices. In *2014 IX Southern Conference on Programmable Logic (SPL)*, pages 1–4. IEEE. DOI: 10.1109/SPL.2014.7002217.

Navas-González, R., Oballe-Peinado, Ó., Castellanos-Ramos, J., Rosas-Cervantes, D., and Sánchez-Durán, J. A. (2023). Practice projects for an fpga-based remote laboratory to teach and learn digital electronics. *Information*, 14(10):558. DOI: 10.3390/info14100558.

Rama, K. and Truong, P. (2024). Designing embedded systems with large language models. Available at: `https://scholarcommons.scu.edu/elec_senior/91/`.

Thakur, S., Ahmad, B., Pearce, H., Tan, B., Dolan-Gavitt, B., Karri, R., and Garg, S. (2024). Verigen: A large language

model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31. DOI: 10.1145/3643681.

VPR (2025). Graphics: Vpr (verilog-to-routing) documentation. Available at: `https://docs.verilogtorouting.org/en/latest/vpr/graphics/` Acessado em: 26 de agosto de 2025.