International Journal of Computer Architecture Education, 2025, 14:1
ISSN: 2316-9915 • doi: 10.5753/ijcae.2025.6436

ARTIGO DE PESQUISA

VeryGA - Interface Modular VGA para Simulação de Verilog

Talles de Sousa Costa [Universidade Federal de Viçosa | *talles.costa@ufv.br*]
Racyus Delano Garcia Pacífico [Universidade Federal Ouro Preto | *racyus.pacifico@ufop.edu.br*]
Ricardo dos Santos Ferreira [Universidade Federal de Viçosa | *ricardo@ufv.br*]

Departamento de Informática, Universidade Federal de Viçosa, Campus Universitário, Viçosa, MG, 36.570-900, Brasil

Resumo. O ensino de Verilog com FPGAs apresenta diversos desafios. Entre eles, destacam-se a complexidade das ferramentas comerciais, o acesso limitado às placas físicas, a própria linguagem de descrição de hardware e a necessidade de simulações com depuração por meio de formas de onda. Uma estratégia eficaz para motivar os estudantes a superar essas barreiras é o desenvolvimento de jogos e aplicações gráficas, cujos resultados podem ser verificados visualmente, promovendo o aprendizado de programação em nível intermediário a avançado. Neste trabalho, apresentamos o VeryGA, uma ferramenta modular que integra simulações em Verilog com entradas via teclado ou mouse e saída em um emulador de sinais VGA, possibilitando visualização gráfica em tempo real. Ao oferecer um ambiente intuitivo e interativo, o VeryGA auxilia iniciantes a compreender melhor e documentar seus projetos de sistemas computacionais em Verilog com saída VGA, reduzindo a curva de aprendizado e promovendo a construção colaborativa de um conjunto de exemplos. A solução é executada inteiramente no navegador por meio do Google Colab, configurado com o simulador Verilator e o CMake, utilizando uma versão em *WebAssembly* para alcançar desempenho próximo ao tempo real. Diversos exemplos são apresentados para ilustrar os recursos do ambiente desenvolvido. O código emulado é sintetizável, validado em placas FPGA reais, e pode ser executado nelas sem alterações. Outra contribuição é demonstrar que, mesmo em um ambiente virtualizado na nuvem, é possível obter taxas de quadros por segundo próximas a uma execução em tempo real.

Palavras-chave: FPGA, Verilog, VGA, Laboratório Remoto.

Recebido: 25 Agosto 2025 • **Aceito:** 01 Outubro 2025 • **Publicado:** 21 Janeiro 2026

1 Introdução

O ensino de FPGA é desafiador pois as placas geralmente têm um alto custo para os estudantes, e o acesso a elas costuma ser restrito, na maioria dos casos, a laboratórios presenciais nas universidades. Além disso, as ferramentas comerciais da AMD/Xilinx e da Intel/Altera são complexas e exigem grande espaço em disco, mesmo nas versões com licença estudantil, dependendo do projeto, o tempo de compilação pode ser muito longo, o que desmotiva os estudantes.

Outro desafio é o uso das linguagens de descrição de hardware (*Hardware Description Language*, HDL), que apresentam diferenças semânticas significativas em relação às linguagens de programação tradicionais, o que pode representar uma barreira para muitos alunos. A depuração e os testes também representam obstáculos: a análise por meio de formas de onda e temporização é complexa para iniciantes, e o uso de *testbenches* exige a compreensão clara da separação entre o código de teste e o código que será sintetizado em hardware.

Uma abordagem motivadora na ciência da computação é o uso de jogos, que também já foi explorada no ensino de hardware e FPGAs [Sanchez-Elez and Roman, 2015; Liu, 2018; Brunvand, 2011]. O desenvolvimento de jogos envolve a compreensão dos mecanismos de entrada e saída, o que contribui para o aprendizado prático. Muitas universidades utilizam exemplos com saída VGA em placas FPGA para motivar os estudantes por meio de aplicações gráficas. Este trabalho propõe a democratização do acesso a projetos em Verilog com saída VGA para o desenvolvimento de jogos e outras aplicações gráficas, com o objetivo de motivar os estudantes no aprendizado da linguagem Verilog e ao desenvolvimento de projetos com FPGAs em todos os níveis do básico ao avançado.

O primeiro desafio é o acesso as placas. As plataformas de ensino e treinamento remoto podem ajudar a superar o obstáculo de acesso aos FPGAs físico, oferecendo acesso remoto a kits de desenvolvimento de hardware [Soares *et al*., 2011; Navas-González *et al*., 2023; Alhammami, 2024; Cruz *et al*., 2024]. O acesso remoto traz diversos benefícios: amplia a acessibilidade ao permitir o aprendizado prático independente da localização, reduz custos ao eliminar a necessidade de aquisição de hardware físico, oferece flexibilidade de tempo e lugar para estudo e permite a experimentação com diferentes configurações e testes em cenários reais. Outra alternativa é o uso de emuladores que permite acesso local ou remoto sem a necessidade da placa física [Costa *et al*., 2023].

O segundo desafio é o aprendizado de Verilog, que é uma HDL amplamente utilizada na indústria e academia para descrever e simular circuitos digitais. No entanto, na maioria dos materiais didáticos, o processo de simulação de circuitos descritos em Verilog ainda é dependente da visualização de formas de onda geradas por ferramentas, tais como, GTKWave [Bybell, 2023], ModelSim [Mentor Graphics Corporation, 2022] e Vivado [Xilinx Inc., 2023].

Quando estudantes são expostos a projetos de hardware de nível intermediário ou avançado, com centenas ou milhares de sinais, a dificuldade em interpretar os resultados da simulação aumenta consideravelmente. Embora algumas ferramentas forneçam meios básicos de filtragem e agrupamento de sinais, elas não oferecem um mecanismo interativo visual adaptado a fins didáticos. Além disso, faltam soluções que integrem o fluxo de simulação com interfaces que permitam intervenções em tempo real ou a personalização de

saídas visuais com base em estímulos específicos.

Nos últimos anos, diversos esforços têm sido realizados para tornar o ensino de HDLs mais eficazes e didáticos, integrando interfaces iterativas com simulação de HDLs, tais como, Learn SystemVerilog [Lima, 2021], VGA playground [Project, 2025] e simuladores baseados em navegadores, tal como, CAD4U no Google Colab [Canesche *et al*., 2021] e DigitalJS [Materzok, 2019] que utilizam o simulador Icarus de Verilog [Williams and Baxter, 2002] e a visualização gráfica com Yosys [Wolf *et al*., 2013]. No entanto, essas soluções ainda são limitadas quanto à integração com projetos reais em Verilog. Além disso, a maioria dos simuladores gratuitos concentra-se em aspectos acadêmicos de pequena escala, sem considerar a escalabilidade ou modularidade necessária para projetos maiores. Apesar do uso de formas de onda ser eficaz para programadores experientes em hardware, ele pouco contribui para o aprendizado prático de estudantes, que se beneficiariam mais de uma abordagem visual intuitiva e didática. Nesse contexto, o desenvolvimento de projetos com jogos e gráficos, aliados a ferramentas com uma interface amigável e didática, pode não apenas acelerar o aprendizado, mas também estimular a experimentação e a criatividade na modelagem de circuitos digitais.

Para atender esses requisitos, propomos o VeryGA, uma ferramenta modular e interativa voltado para a visualização e interpretação de simulações em Verilog com saída VGA. O VeryGA, intercepta sinais de entrada e saída, fornecendo uma camada visual interpretável em tempo real para conexão com um monitor VGA. Além disso, a ferramenta possui uma arquitetura extensível, permitindo que novos módulos de visualização e análise sejam incorporados de forma incremental. A usabilidade foi priorizada, com interface projetada para usuários com pouco ou nenhum conhecimento prévio em ferramentas de simulação via Google Colab, garantindo acesso remoto sem a necessidade de instalação. A ferramenta também tem potencial em ambientes educacionais, pois reduz significativamente o tempo de validação de projeto com saída VGA e pode aumentar o engajamento dos alunos em projetos de hardware com jogos.

As principais contribuições deste trabalho são:

- Contribuir no ensino/aprendizado na programação de HDLs usando jogos;
- Criar projetos Verilog usando o simulador Verilator integrado ao Google Colab;
- Validar os projetos desenvolvidos em placas reais, reduzindo o tempo de desenvolvimento.
- Visualizar a saída da simulação de um projeto em tempo real, comunicando com uma tela VGA virtual implementado de forma eficiente com *WebAssembly*;
- Simular circuitos digitais, interagindo por sinais enviados por entradas do teclado ou mouse.

Este artigo está organizado da seguinte forma: Seção 2 apresenta os trabalhos relacionados; Seção 3 contextualiza conceitos fundamentais e ferramentas utilizadas no VeryGA; Seção 4 descreve o funcionamento e detalhes de implementação da ferramenta; Seção 5 apresenta os casos de uso utilizados para avaliar e validar o VeryGA; Seção 6 aborda os resultados obtidos; Por fim, a Seção 7 aborda as conclusões, destacando as principais vantagens da ferramenta.

2 Trabalhos relacionados

No ensino de linguagens de descrição de hardware, muitos cursos usam apenas ferramentas tradicionais como GTKWave [Bybell, 2023] e ModelSim [Mentor Graphics Corporation, 2022], que oferecem apenas a visualização das formas de onda para análise de sinais, o que limita a compreensão do comportamento dos circuitos, especialmente na ausência de placas FPGA ou ASICs sintetizados.

Diversos projetos têm buscado estimular o ensino de HDLs de uma forma mais didática e interativa. Por exemplo, a ferramenta DigitalJS [Materzok, 2019] permite a visualização de projetos, a interação com sinais de entrada e saída, exploração da visualização de forma hierárquica e visualização das formas de onda. Uma extensão do DigitalJS, apresenta projetos do processador MIPS [Passe *et al*., 2020] para estimular o ensino de Verilog. Trabalhos recentes exploram as vantagens do Google Colab para criar laboratórios virtuais para o ensino de Verilog [Canesche *et al*., 2021; Ferreira *et al*., 2024b,a]. Uma alternativa explorada para motivar os estudantes no ensino de linguagens é o uso de jogos [Combéfis *et al*., 2016]. O ensino de jogos também é utilizado no ensino de FPGA e linguagens de hardware [Jarusauskas, 2009; Neebel *et al*., 2012; Sanchez-Elez and Roman, 2015].

Existem várias ferramentas para uso dos simuladores da linguagem Verilog como Icarus [Canesche *et al*., 2021] e Verilator [Snyder, 2004], um simulador de Verilog HDL que permite simular e sintetizar de forma rápida e simples circuitos digitais usando a linguagem C++. Além disso, suporta vários tipos de otimizações. No Verilator a simulação suporta estímulos, importa arquivos gerados de outros simuladores e os compila para um simulador específico.

Um ambiente virtual cliente servidor com interface para chaves e display usando Verilator foi proposto por [Dai and Cai, 2024]. Outro projeto similiar é o Learn SystemVerilog [Lima, 2021] foi desenvolvido para o ensino remoto de HDLs, oferecendo simulação no navegador via Verilator e *WebAssembly*. Ele permite interações básicas com periféricos como LEDs, chaves e display de sete segmentos, mas não suporta interfaces gráficas usando VGA. Além disso, não permite o carregamento de arquivos externos da memória.

A ferramenta mais próxima ao nosso trabalho é o VGA Playground [Venn, 2024], que destaca-se pelo foco na simulação de saídas gráficas via interface VGA, sendo um projeto associado ao movimento *TinyTapeout* [Venn, 2024]. Embora limitado a 64 cores e sem suporte a outros periféricos, sua capacidade de resposta visual e suporte a áudio o tornam um diferencial para o ensino de circuitos gráficos.

Em um contexto mais amplo, o ambiente EDA Playground [Doulos, 2024] permite a experimentação de código HDL (Verilog, SystemVerilog, VHDL) na nuvem, com suporte a diferentes simuladores. Apesar de ser uma ferramenta poderosa no aprendizado de HDLs, não possui recursos visuais que auxiliem na compreensão intuitiva dos circuitos.

Projetos como o Logisim Evolution [Burch *et al*., 2024] também oferecem simulação gráfica de circuitos, mas não trabalham diretamente com HDL. No entanto, sua interface amigável é citada em diversos estudos como inspiração para ferramentas voltadas ao ensino.

Além disso, o QUCS (*Quite Universal Circuit Simula-*

tor) [Qucs Team, 2024] e o *Falstad Circuit Simulator* [Falstad, 2024] fornecem simulações interativas para circuitos analógicos e digitais, mas não integram suporte a HDLs, o que os limita no ensino de HDLs.

Por fim, destaca-se o EDAboard [EDABoard Community, 2024] como fórum de discussão e compartilhamento de simulações HDL com foco educacional. Embora não seja uma ferramenta, é uma importante referência para soluções práticas utilizadas no ensino de HDLs.

A Tabela 1 apresenta as principais características das ferramentas baseado nos recursos de simulação (Sim), linguagem de descrição de hardware (HDL), acesso online na internet (textitWeb), recursos de visualização (Vis) e visualização VGA (VGA). Nenhuma das ferramentas contém todas as funcionalidades disponíveis no VeryGA e no VGA Playground. Entretanto, o VeryGA está embarcado no Google Colab, agregando todas os seus recursos para documentação com textos e gráficos, codificação em células para modularizar o código, compartilhamento, além de ser dimensionado para agregar extensões na interface de sinais de entrada e saída. Ou seja, oferece mais recursos didáticos que o VGA Playground.

Tabela 1. Comparação das ferramentas com o VeryGA.

Ferramenta	Sim	HDL	*Web*	Vis	VGA
ModelSIM	X	X		X	
CAD4U	X	X	X		
Icarus	X	X			
Verilator	X	X			
Games				X	
Games FPGA	X			X	X
Logisim	X		X	X	
Learn					
SystemVerilog	X	X	X	X	
EDA Playground	X	X	X	X	
DigitalJS	X	X	X	X	
QUCS	X		X	X	
Falstad Simulator	X		X	X	
VGA Playground	X	X	X	X	X
GTKWave				X	
EDAboard		X			
VeryGA	X	X	X	X	X

Ferramentas: ModelSIM [Mentor Graphics Corporation, 2022], CAD4U [Canesche *et al.*, 2021], Icarus [Williams, 2024], Verilator [Snyder, 2004], Games [Combéfis *et al.*, 2016], Games FPGA [Jarusauskas, 2009; Neebel *et al.*, 2012; Sanchez-Elez and Roman, 2015], Logisim [Burch *et al.*, 2024], QUCS [Qucs Team, 2024], Falstad Simulator [Falstad, 2024], VGA Playground [Venn, 2024], GTKWave [Bybell, 2023], EDAboard [EDABoard Community, 2024].

3 Referencial Teórico

Protocolo VGA: É um padrão de interface analógica para saída de vídeo de computadores, desenvolvido pela IBM [Thompson, 1988] para equipar os computadores da empresa na década de 1980. Para a transmissão de imagem, o protocolo VGA utiliza dois sinais de sincronização digitais: um horizontal (`HSYNC`) e um vertical (`VSYNC`). Além disso, o protocolo utiliza três sinais de cores analógicos (`R`, `G` e `B`) com resolução padrão de 640 × 480 *pixels* com 262.144 cores, correspondendo a um sinal RGB com 6 bits por canal de cor ($2^6 \times 2^6 \times 2^6$).

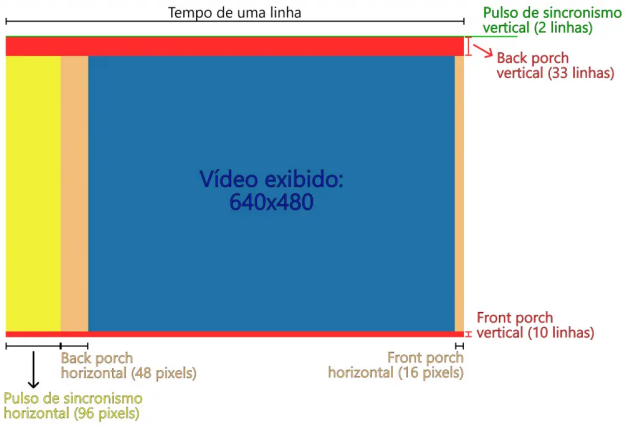

Figura 1. Temporização dos sinais VGA [Guimarães, 2019].

O escaneamento do *frame* é iniciado após um pulso de `VSYNC` e realizado linha por linha. Após cada pulso de `HSYNC`, o escaneamento da próxima linha começa, com cada *pixel* sendo desenhado em um intervalo de tempo determinado. São necessários 40 ns para desenhar cada pixel na área visível de 640 × 480 *pixels* dentro de uma área total de 800 × 525 *pixels*. Os *pixels* não visíveis são necessários para prover tempo de espera entre linhas e quadros, permitindo que o monitor reposicione o feixe de elétrons (nos tubos de raios catódicos, CRTs).

Existem três categorias de *pixels* não visíveis: o descanso antes do pulso (*Front porch*, 16 *pixels*), a sincronização real (*sync pulse*, 96 *pixels*) e o tempo até começar os dados úteis (*back porch*, 48 *pixels*). Na Figura 1 é possível observar o envio da imagem visível e dos outros três campos, com 33 linhas no topo antes de começar a parte visível, aplicados a cada linha, e na parte inferior com 10 linhas [Chinchanikar *et al.*, 2023].

Jupyter Notebook e Google Colab: O Jupyter Notebook permite a execução e exibição de células de código e texto em navegadores *Web*. Os códigos de linguagens de programação podem ser intercalados com texto formatado para documentar o processo de desenvolvimento, funcionando como um caderno de pesquisa interativo, razão pela qual recebe o nome de *notebook* [Kluyver *et al.*, 2016]. Além disso, suporta diferentes tipos de saída, incluindo texto, gráficos, fórmulas matemáticas formatadas e elementos gráficos interativos. Alguns comandos podem ter funções específicas, por exemplo, modificar a execução de uma célula, tais como os comandos mágicos. Inicialmente foi desenvolvido pelo projeto IPython [Perez and Granger, 2007] para uso com a linguagem Python. Atualmente possui integração com diversas linguagens, tais como, C++ e JavaScript por meio de *backends* e extensões. Um Jupyter Notebook pode ser hospedado gratuitamente no Google Colab [Research, 2017], um serviço do Google Cloud. Embora seja mais utilizado para prototipagem de modelos de aprendizado de máquina usando GPUs e TPUs [Bisong, 2019], também pode ser aplicado em outros tipos de domínios.

Verilator: É uma ferramenta de código aberto ampla-

mente utilizada para simulação e verificação de projetos em Verilog. Diferente de simuladores tradicionais baseados em interpretação, como o *ModelSim* ou *Icarus Verilog*, o Verilator realiza tradução de código Verilog para C++ ou SystemC, gerando um modelo de simulação compilado. Esse modelo é então executado como um programa nativo, o que permite simulações extremamente rápidas, especialmente em testes de longa duração. O processo de tradução envolve análise léxica e sintática, otimizações estáticas, eliminação de blocos redundantes e geração de funções C++ correspondentes aos módulos e comportamentos do hardware descrito [Snyder, 2004]. O Verilator foi escolhido para executar simulações rápidas usando linguagem de alto nível no VeryGA, gerando código C++ compatível com código Verilog.

Emscripten: É uma ferramenta que compila código LLVM (*Low Level Virtual Machine*) para *WebAssembly*, permitindo a execução de aplicações escritas em diversas linguagens diretamente no navegador, por exemplo, C, C++ e Rust. O uso do *WebAssembly* garante desempenho próximo ao nativo, tornando o *Emscripten* uma solução eficaz para aplicações que exigem execução em tempo real. Isso ocorre porque ele integra um ambiente que simula aspectos de um sistema operacional, tais como, sistema de arquivos virtual, suporte a *threads* e ponteiros, melhorando o desempenho da aplicação que está rodando no navegador. O Emscripten foi incorporado ao VeryGA para possibilitar a execução da simulação no navegador com alto desempenho, uma vez que a emulação de um monitor VGA em tempo real (ou próximo a isso) demanda uma implementação eficiente [Zakai, 2011; Zakai and contributors, 2025].

4 VeryGA

Nesta seção descrevemos detalhes de implementação e funcionamento do VeryGA, que integra simulação de código Verilog conectado a um emulador de monitor VGA e leitura do teclado ou mouse como entrada da ferramenta. Para o funcionamento do VeryGA no Google Colab, o primeiro passo consiste em realizar a instalação do repositório do projeto[1]. Para executar o respectivo código, deve-se criar uma célula. Em seguida, conectar o projeto do código Verilog que deseja validar, às interfaces de entrada (teclado ou mouse) e saída (monitor VGA). É importante ressaltar que o código emulado no VeryGA sob o Google Colab, pode ser executado diretamente no FPGA sem adaptações porque é totalmente compatível com as configurações do hardware.

O código da célula VeryGA é executado seguindo o fluxograma ilustrado na Figura 2. Primeiro, o usuário cria uma ou mais células com seu projeto em Verilog no Google Colab. Uma das células será a principal e as outras irão compor o projeto, permitindo ao usuário modularizar seu código e documentá-lo. O usuário pode implementar todo o projeto em uma única célula ou decompô-lo em células auxiliares, que devem ter na primeira linha o comando `%%veryga [NomeModulo.v]`, onde o nome do arquivo corresponde ao nome do módulo especificado. Todo projeto criado na ferramenta contém uma célula principal, representando o topo da hierarquia. Nesta célula, o comando mágico `%%veryga --top` deve ser inserido na primeira linha da célula, em seguida, adicionadoo código *template* apresentado na Figura 3.

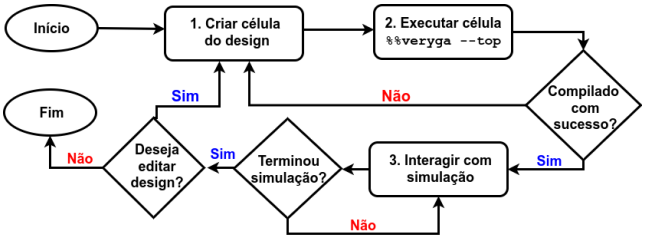

Figura 2. Fluxograma de execução VeryGA.

O segundo passo é executar a célula principal. Neste momento, o projeto é compilado e os erros de compilação são sinalizados. Caso não haja erros, o projeto Verilog é transformado em C++ pelo Verilator e, posteriormente, convertido em *WebAssembly* para execução. Este processo pode demorar alguns segundos, devido a complexidade da simulação de eventos em hardware. Finalmente, quando o código inicia a execução, a saída VGA é exibida e o usuário pode interagir com o projeto durante a simulação usando teclado e/ou mouse.

```
1 module veryga(input wire [63:0] in,
2               output  wire [63:0] out
3 );
4     wire clk, rst;
5     wire [3:0] mov;
6     wire hsync, vsync;
7     wire [3:0] r, g, b;
8
9     assign out = {50'b0,hsync,vsync,r,g,b};
10    assign clk = in[0];
11    assign rst = in[1];
12    assign mov = in[5:2];
13
14    /*
15       Adicionar código Verilog que
16       deseja conectar com o VeryGA.
17    */
18 endmodule
```

Figura 3. Código *template* da célula principal do projeto veryga.v.

A Figura 3 apresenta o módulo `veryga` e sua interface de entrada e saída, ambas com 64 bits cada. Para possibilitar futuras extensões, o módulo foi projetado para enviar e receber 64 sinais simultaneamente em uma simulação. A versão atual possui o seguinte mapeamento de sinais:

Sinais de entrada: O barramento `in` possui apenas seis conexões iniciais. Dois sinais de controle: o *clock* principal (`clk` conectado em `in[0]`), autogerado com 25 MHz na simulação, e o sinal de *reset* (`rst` conectado em `in[1]` e na tecla A. Pressionando a tecla A, o projeto será reiniciado. Para conectar ao teclado, a versão atual possui uma interface de quatro bits com o sinal `mov` conectado em `in[5:2]` e às teclas direcionais (seta para cima, seta para baixo, seta para esquerda e seta para direita).

Sinais de saída: O barramento `out` possui os seguintes campos conectados: o canal de cor `B` (`out[3:0]`), o canal de cor `G` (`out[7:4]`) e o canal de cor `R` (`out[11:8]`), proporcionando uma interface de 4096 cores, além dos sinais de sincronização horizontal `HSYNC` (`out[12]`) e sincronização vertical `VSYNC` (`out[13]`). Os sinais *VSYNC*, *HSYNC*, *R*, *G*

[1] !pip install git+https://github.com/hamsty/VeryGA.git, % load_ext plugin

```
module veryga(
    input wire [63:0] in,
    output  wire [63:0] out
);
    wire clk,rst;
    wire hsync, vsync;
    wire [3:0] r, g, b;

    assign out = {50'b0, hsync, vsync, r, g, b};
    assign clk = in[0];
    assign rst = in[1];

    wire display_en;
    wire [11:0] x, y;

    vga_driver vga1 (clk, ~rst, hsync, vsync,
    display_en, x, y);

    assign r = ((display_en && rst)?((x < 80)? 4'
    hF:((x < 160 || (x>=320 && x<480))?4'hB:0))
    :4'h0);
    assign g = ((display_en && rst)?((x < 80)? 4'
    hF:((x < 320)?4'hB:0)):4'h0);
    assign b = ((display_en && rst)?((x < 80)? 4'
    hF:(((x % 160) < 80)?4'hB:4'h0)):4'h0);

endmodule
```

Figura 4. EBU bars em Verilog.

e *B* seguem o padrão VGA descrito na Figura 1 e foram implementados de forma a emular o monitor VGA. Todos os projetos testados foram validados em placas de FPGA, onde funcionaram corretamente seguindo o padrão VGA.

Um exemplo clássico para ensino e teste do monitor VGA é o padrão de barras verticais com cores, conhecido como padrão EBU cores, que consiste em conjunto de padrões de cores estabelecidos pela União Europeia de Radiodifusão (EBU) para a produção de vídeo e televisão. Estes padrões visam garantir consistência e compatibilidade na reprodução de cores em diferentes equipamentos e plataformas. A Figura 4 apresenta o código que deve ser adicionado ao *template* do projeto Verilog que foi ilustrado na Figura 3. A compilação e execução do código EBU irá gerar a saída exibida na Figura 5.

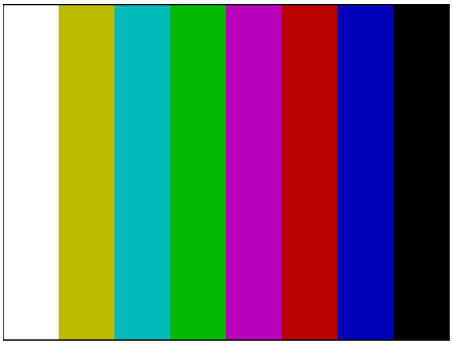

Figura 5. EBU bars.

O código da Figura 4 demonstra a definição da interface de conexão com o VGA, implementando um padrão de cores EBU (*European Broadcasting Union*) através do controle dos canais de cor `R`, `G` e `B` em função da coordenada horizontal `X`. O módulo `vga_driver` é responsável por gerar os sinais de sincronização e fornecer as coordenadas de *pixel* atual. A implementação das cores segue uma lógica condicional que só é executada quando o *reset* está desativado (sinal `rst` em nível alto) e o *display* está habilitado (`display_en` ativo).

O padrão EBU é implementado a partir de diferentes combinações de cores baseadas na posição horizontal `X`:

- **Branco (`X` < 80)**: Todos os canais RGB em máxima intensidade (4'hF), criando uma barra branca na região inicial da tela.
- **Amarelo (80 ≤ `X` < 160)**: Canal vermelho (`R`) e verde (`G`) em alta intensidade (4'hB), com azul (`B`) alternando entre alta e baixa intensidade conforme `X` % 160, resultando em tons de amarelo.
- **Ciano (160 ≤ `X` < 320)**: Apenas o canal verde (`G`) em alta intensidade (4'hB), com vermelho (`R`) desligado e azul (`B`) alternando, produzindo tons de ciano.
- **Verde (320 ≤ `X` < 480)**: Canal vermelho (`R`) e verde (`G`) em alta intensidade (4'hB), com azul (`B`) seguindo o padrão de alternância, gerando tons esverdeados.
- **Preto/Azul (`X` ≥ 480)**: Canal vermelho (`R`) e verde (`G`) desligados, com apenas o azul (`B`), alternando entre alta e baixa intensidade.

A intensidade 4'hB (valor hexadecimal 11, equivalente a 75% da intensidade máxima) e 4'hF (valor hexadecimal 15, intensidade máxima) são utilizadas para criar as diferentes tonalidades do padrão EBU, proporcionando uma referência visual para calibração e teste de monitores.

As células com módulos auxiliares além do módulo principal devem ser executadas para salvar os arquivos em um diretório temporário. Somente após a execução de todas as células com os módulos auxiliares deve-se executar a célula do topo. Como já mencionado, a execução inicia com a etapa de compilação, que executa uma configuração de *build* do CMake e, posteriormente, compila a *build* utilizando o *make*. Os módulos são mapeados em código C++ pelo compilador Verilator, e os arquivos gerados são conectados a uma camada visual que utiliza *Simple DirectMedia Layer 2* (SDL2) para renderizar a tela.

O projeto é compilado utilizando o Emscripten, possibilitando sua execução no navegador, incluindo em ambientes de desenvolvimento que utilizam o Jupyter Notebook. Para que essa integração seja possível, foram adicionadas ao arquivo `main.cpp` as linhas da Figura 6, onde a função `emscripten_set_main_loop` define a função de *loop* que será executada pelo navegador e um intervalo de tempo reservado. As funções anteriores definem as funções de *callback* para as entradas do teclado.

```
    emscripten_set_keydown_callback("#canvas", 0,
     1, key_callback);
    emscripten_set_keyup_callback("#canvas", 0,
    1, key_callback);
    emscripten_set_main_loop(loop, -1, true);
    emscripten_set_main_loop_timing(
    EM_TIMING_SETTIMEOUT, 1000);
```

Figura 6. Integração com Emscripten.

Para a integração com ambientes *online* onde não é possível definir as configurações do servidor que hospeda a página principal, por exemplo, o Google Colaboratory, foi importante configurar a compilação para desabilitar o uso de *threads* pelo programa. A execução de múltiplas threads com *WebAssembly* depende do uso de memória compartilhada através do objeto JavaScript `SharedArrayBuffer`. Para evitar que conteúdo malicioso seja executado por fontes desco-

nhecidas, é exigido que a resposta do servidor retorne os cabeçalhos *Cross-Origin-Opener-Policy: same-origin* e *Cross-Origin-Embedder-Policy: require-corp*.

Após a etapa de compilação, caso o servidor *Web* ainda não esteja ativo, um novo servidor é iniciado, apontando para a pasta de saída da *build*. Retorna-se então um *iframe* como saída, direcionando para o endereço do servidor. O resultado do código na Figura 4, após o processo mencionado acima, pode ser visualizado na Figura 5.

Caso o projeto seja implementado utilizando o `wire mov` é possível interagir com a simulação usando as telas direcionais, como em uma placa FPGA ou ASIC sintetizado com o mesmo código. A simulação pode ser encerrada a qualquer momento utilizando a tecla B e essa retorna status de execução da simulação, como tempo de execução e tempo de simulação.

5 Casos de uso

Um dos objetivos deste trabalho é motivar estudantes a aprender Verilog com exemplos lúdicos de jogos e efeitos gráficos. Vários trabalhos utilizam jogos clássicos de videogames para exemplificar projetos em hardware de forma divertida [Jarusauskas, 2009; Neebel *et al.*, 2012; Sanchez-Elez and Roman, 2015], como jogos do videogame Atari (*breakout*, *space invaders*) ou clássicos como *snake*, *tetris*, *pong*, dentre outros.

Esta seção apresenta vários exemplos com complexidade incremental para validar a ferramenta e motivar o aprendizado de Verilog. Foram escolhidos cinco exemplos, além do EBU já apresentado, baseados no nível de complexidade de implementação. A proposta é que estudantes possam fazer pequenas modificações para entender como desenhar em um monitor VGA e, posteriormente, aprender técnicas básicas de construção de jogos, ao mesmo tempo que reforçam o aprendizado de Verilog e de projetos de hardware com paralelismo, máquinas de estados e outras construções. Cada exemplo possui uma funcionalidade específica, como descrito a seguir:

1. **Retângulos:** Demonstra um exemplo simples de projeto de uma forma geométrica;
2. **Capivara:** Realiza a leitura de uma imagem de um arquivo para uma memória ROM integrada ao ambiente;
3. **Movimentação de Retângulos:** Apresenta um exemplo simples de extensão do projeto do retângulo para utilizar a interface de entrada com interação via teclado, fazendo uso de comandos direcionais (setas) para movimentar o retângulo dinamicamente no monitor VGA;
4. **Breakout:** Valida um código desenvolvido em uma disciplina de graduação da Unicamp [Martins Jacob *et al.*, 2024], projetado para executar em uma placa de FPGA, demonstrando que o emulador é capaz de reproduzir um jogo em tempo real com pequeno atraso aceitável devido ao ambiente virtual multicamadas que possibilita a execução;
5. **Labirinto:** Exemplo de jogo que inclui a criação de um labirinto maior que a tela visível, demonstrando como implementar um projeto com navegação através do deslocamento da janela visível no monitor, além de interagir com o teclado.

Nas Seções 5.1 a 5.5 são apresentados mais detalhes de cada exemplo validado na ferramenta VeryGA.

5.1 Retângulos

Este exemplo é um projeto simples que desenha retângulos com borda. Ele é desenvolvido em torno do módulo *rectangle* (Figura 7), que permite criar retângulos na tela com tamanho, posição, cor, espessura de borda e cor de borda variáveis.

```
1
2 module rectangle
3 #( parameter X0 = 20, parameter Y0 = 20,
4     parameter WIDTH = 600,
5     parameter HEIGHT = 440,
6     parameter COLOR = 12'hFFF,
7     parameter BORDER = 20,
8     parameter BORDER_COLOR = 12'hFFF
9 )(
10     input wire clk, rst,
11     input wire [11:0] x, [11:0] y,
12     output wire enable,
13     output reg [11:0] p_color
14 );
15
16   assign p_color = (x > (X0+BORDER) && x < (X0+
       WIDTH-BORDER)) && (y > (Y0+BORDER) && y < (Y0
       +HEIGHT-BORDER))?COLOR:BORDER_COLOR;
17
18   assign enable = (x > X0 && x < (X0+WIDTH)) && (
       y > Y0 && y < (Y0+HEIGHT))?1'b1:1'b0;
19
20 endmodule
```

Figura 7. Código do módulo *retangle*.

Na Figura 8 é possível ver a saída da simulação de um exemplo com duas instâncias do módulo retângulo executando na ferramenta. A implementação de um módulo parametrizado ilustrado na Figura 9 permite que o usuário modifique o código e visualize mudanças na imagem mostrada com pouco conhecimento de Verilog.

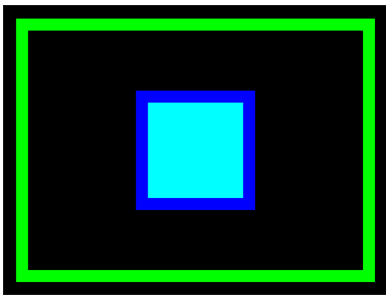

Figura 8. Retângulos estáticos.

```
1 rectangle #(.X0(20), .Y0(20), .WIDTH(600),
2             .HEIGHT(440), .COLOR(12'h000),
3             .BORDER_COLOR(12'h0F0))
4 r1(clk, ~rst, x, y, rect_en, p_color(pixel));
```

Figura 9. Código com módulo parametrizado.

O código do retângulo cria um retângulo com o vértice superior esquerdo na posição (20,20), com 600 *pixels* de largura e 440 de altura, na cor verde. A cor é definida pelo valor hexadecimal 12'h0F0, onde os canais vermelho e azul estão desligados (valor 0) e o verde está no valor máximo (0xF). A cor de fundo é preta.

5.2 Capivara

Neste caso de uso foi criado um projeto que mapeia o conteúdo de um arquivo e produz uma saída com imagem estática de uma capivara espacial. A imagem foi criada usando o modelo generativo da ferramenta Copilot da Microsoft [Stratton, 2024]. A resolução da imagem foi inicialmente reduzida para 160×120 *pixels* e posteriormente convertida para 12 bits (4 bits por canal de cor). Em seguida, cada *pixel* foi armazenado em uma linha de um arquivo de texto no formato hexadecimal.

O arquivo foi passado como parâmetro para a instância de memória ROM, que carrega os dados no momento da simulação a partir da função *$readmemh*, nativa do Verilog. No caso de execução em hardware real, os dados devem ser gravados no módulo em tempo de síntese. A saída da simulação na ferramenta pode ser vista na Figura 10.

Figura 10. Imagem de Capivara carregada na ROM.

5.3 Movimentação de Retângulos

Para demonstrar a utilização dos comandos direcionais mapeados para a entrada da simulação, foi criado um exemplo baseado nos retângulos na Seção 5.1. A posição `X0` e `Y0`, anteriormente passadas como parâmetro da instância (Figura 7), agora são sinais recebidos pelo módulo `rectangle` e que modificam a posição do retângulo conforme o usuário pressiona as teclas direcionais. A Figura 11 mostra a simulação em andamento, após o retângulo ser movido da sua posição inicial, que é o centro da tela.

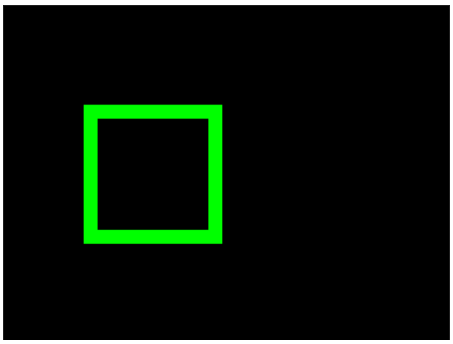

Figura 11. Retângulo fora da sua posição inicial.

5.4 Breakout

Esse caso de uso foi criado originalmente como projeto final da disciplina MC613 - Laboratório de Circuitos Digitais, ministrada no primeiro semestre de 2024 pelo Prof. Dr. Rodolfo Jardim De Azevedo, Unicamp. O grupo composto pelas alunas Gabriela Martins Jacob, Letícia Lopes Mendes da Silva e Luísa de Melo Barros Penze criou uma versão simplificada do jogo Breakout [Kent, 2010] para rodar em uma placa DE1 da Terasic.

O objetivo do jogo é rebater uma bolinha branca em blocos coloridos no topo da tela, utilizando uma plataforma que movimenta horizontalmente. O jogador possui três vidas e perde uma vida a cada vez que a bolinha bate no chão. Ele vence o jogo ao eliminar todos os blocos. Caso o jogador perca todas as vidas, o jogo é finalizado, e ele perde. Pequenos ajustes na temporização foram feitos no código, pois a placa possui um *clock* principal de 50 Mhz e a ferramenta assume que o *clock* principal seja 25Mhz. É possível ver a saída da simulação na Figura 12.

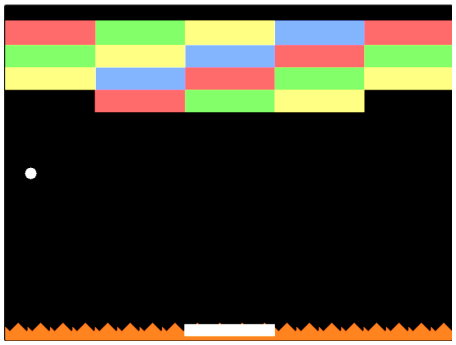

Figura 12. Retângulo fora da sua posição inicial.

5.5 Labirinto

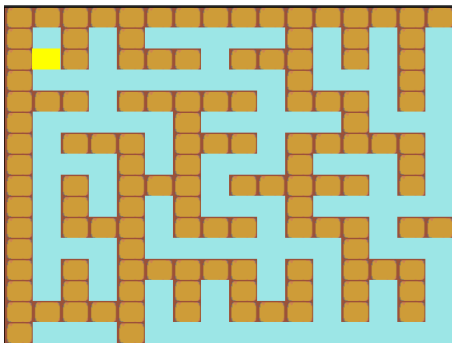

Figura 13. Início do jogo Labirinto.

Para o teste do uso de todas as funcionalidades da plataforma VeryGA, foi criado um jogo de labirinto. Nesse jogo o usuário começa no canto superior de um labirinto 48x48, carregado em memória durante o processo de síntese do código. O labirinto não é visto em sua totalidade, com sua visão sendo restrita a uma área 16x16 do labirinto. Cada bit do labirinto 48x48, sendo 0 o muro e 1 o caminho, é desenhado como um bloco 40x30 *pixels*, que em uma visão 16x16 ocupa a totalidade da resolução VGA implementada (640x480), como pode ser visto na Figura 13.

Para compatibilidade com a placa FPGA citada nos testes abaixo, foi mapeado 4 bits por canal de cor, totalizando 4096 cores disponíveis. Isso não impede que, em futuras versões, seja desenvolvida uma implementação que aceite 8 bits por canal, totalizando 16.777.216 de cores presentes no padrão RGB 888, amplamente utilizado nos monitores atuais.

O jogador (quadrado amarelo) pode andar com as entradas passadas por sinal ao `wire mov`, sendo que para

Tabela 2. Resultado simulação casos de uso do VeryGA.

	Velocidade (ms/s)				Taxa de Quadros (fps)			
Casos de uso	Min	Max	Média	Desvio	Min	Max	Média	Desvio
Retângulo	936,18	1007,00	982,84	24,8338	56,90	59,86	58,62	1,1589
EBU bars	800,78	874,58	851,81	20,5762	47,79	52,02	50,68	1,1802
Capivara	348,12	371,23	363,75	7,1565	20,71	22,08	21,64	0,4248
Retângulo ambulante	984,18	1008,00	1000,45	9,1901	58,54	59,91	59,49	0,5338
Breakout	195,76	197,94	196,63	0,7106	12,22	12,35	12,30	0,0487
Labirinto	413,85	426,02	421,44	3,9980	24,64	25,33	25,07	0,2307

`mov=4'b0001` anda para direita, `mov=4'b0010` anda para baixo, `mov=4'b0100` anda para cima e `mov=4'b1000` anda para esquerda. Para outros valores de `mov`, o jogador não executará nenhuma ação, evitando padrões não desejados, como andar na diagonal.

O jogador pode andar pelas casas azuis e as casas as quais ele não pode andar são desenhadas com uma imagem carregada em memória. Ao andar sete casas quando está nas bordas do labirinto, a câmera se movimenta pelo mapa, uma casa na direção do movimento, a fim de ver o caminho à frente, parando ao chegar próximo a outra borda.

O jogo termina ao chegar no quadrado vermelho, na borda direita inferior do labirinto, apresentado na Figura 14.

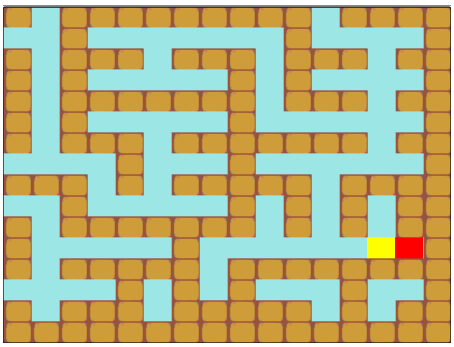

Figura 14. Fim do jogo labirinto.

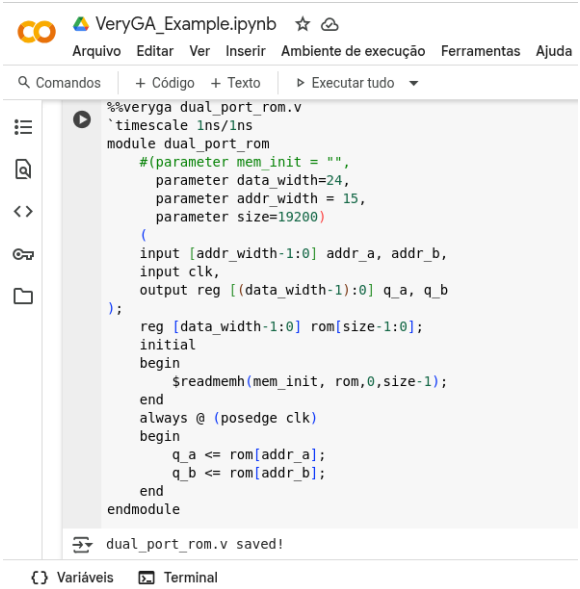

Figura 15. Célula Google Colab com o código Verilog da memória ROM do Labirinto.

Os arquivos que compõem o projeto foram divididos em oito células de código com o comando `%%veryga` antes da etapa de síntese, oferecendo portabilidade ao projeto. A Figura 15 ilustra um módulo que pode ser facilmente substituído ao sobrescrever o arquivo. Esse exemplo testa o comportamento da plataforma para reagir a mudanças contínuas nas entradas e mostrar diferentes telas, ilustrando a funcionalidade da simulação interagindo com estímulos externos.

6 Resultados

Nesta seção apresentamos os resultados da ferramenta em um ambiente realista, demonstrando que o código simulado no Google Colab pode ser executado em hardware. O ambiente de teste foi composto por:

- Uma placa FPGA, DE10-Lite da Terasic que possui um chip Intel MAX 10M50DAF484C7G, com 50 k de elementos lógicos;
- Para simulação do VeryGA, usamos um notebook Samsung Galaxy Book 3 360 com a seguinte configuração: processador Intel(R) Core(TM) i5-1335U, 16 GB de memória RAM e sistema operacional Windows 11. Nesta máquina executamos o Google Colab com o código do VeryGA usando o navegador Microsoft Edge.

Os cálculos dos testes do VeryGA foram realizados utilizando o status retornado pela simulação ao ser finalizada pressionando a tecla B. Os testes foram realizados de forma automatizada utilizando *scripts* para simular a entrada do mouse e teclado. Para validar o VeryGA rodando em uma FPGA, escolhemos o estudo de caso mais complexo, labirinto, devido ao grau de dificuldade de implementação. A Figura 16 apresenta o labirinto rodando em uma FPGA com saída VGA conectada em um monitor LCD, com frequência de 60 Hz ou 60 fps.

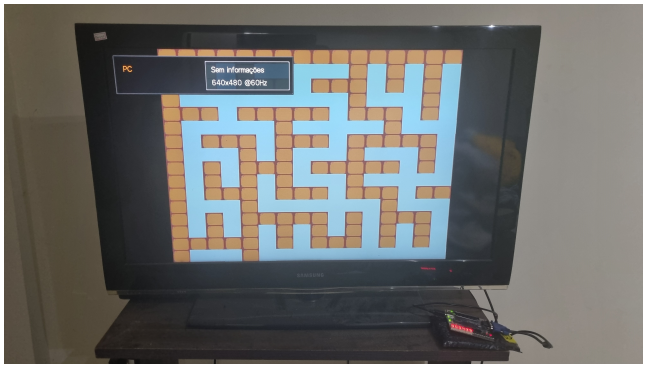

Figura 16. Labirinto rodando em FPGA conectado em TV LCD via VGA.

Além de validarmos o VeryGA em hardware, medimos a velocidade (ms/s) e fps da simulação dos casos de uso para avaliar o desempenho de cada um rodando no navegador. Repetimos a simulação para cada estudo de caso 10 vezes estaticamente, sem movimentos, por um minuto. Os resultados obtidos da simulação são apresentados na Tabela 2.

Em relação à execução no FPGA houve uma perda de mais da metade do desempenho, mas a simulação continuou gerando uma taxa de quadros que não comprometeu a visualização e permitiu a interação com ela.

7 Conclusão

Neste trabalho propomos o VeryGA, uma ferramenta interativa voltado para a visualização e interpretação de simulações em Verilog com saída VGA, que representa um recurso atrativo no ensino HDLs, especialmente no contexto do uso de FPGAs e da linguagem Verilog. A ferramenta contorna de forma eficaz desafios existentes nesse contexto, como a limitação de acesso físico às placas, a complexidade das ferramentas comerciais e a dificuldade de interpretação das simulações por meio de formas de onda. Ao integrar a visualização de saída VGA em tempo real a um ambiente acessível via Google Colab, o VeryGA possibilita que estudantes interajam com projetos de hardware de maneira prática, intuitiva e motivadora, independentemente de sua localização ou da disponibilidade de equipamentos físicos.

Além de democratizar o acesso a recursos de simulação, a abordagem adotada potencializa o uso de jogos como meio de aprendizado, incentivando a experimentação, a criatividade e o engajamento. A arquitetura modular e extensível da ferramenta favorece sua evolução futura, permitindo a incorporação de novos módulos e funcionalidades que ampliem seu alcance e aplicação. Assim, o VeryGA contribui para aproximar teoria e prática, tornando o processo de aprendizado de HDLs mais acessível, didático e estimulante, e abrindo caminho para novas metodologias de ensino voltadas à formação de profissionais mais preparados para os desafios do desenvolvimento de sistemas digitais. Um exemplo prático do VeryGA é integrá-lo ao processador RISC-V por meio de uma interface de saída com o professor podendo preparar o experimento e guiar o estudante a partir das células do Google Colab.

Como trabalhos futuros, desenvolveremos uma interface para suportar a criação de periféricos personalizados e simulação de outras tecnologias de entrada e saída de vídeo, por exemplo, HDMI, DVP, MIPI CSI e MIPI DSI. Além disso, o VeryGA suportará GPUs para acelerar a simulação dos projetos, melhorando o desempenho da ferramenta.

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

O autor Talles de Sousa Costa implementou a ferramenta VeryGA e realizou parte da redação do texto. O autor Ricardo Ferreira elaborou os exemplos base dos simuladores e realizou a redação do texto. O autor Racyus Delano Garcia Pacífico colaborou com sugestões e revisão no texto.

Conflitos de interesse

Os autores declaram não haver conflitos de interesse.

Disponibilidade de dados e materiais

As ferramentas desenvolvidas neste trabalho são de código aberto e estão disponíveis no link `https://github.com/hamsty/VeryGA.git`.

Outras informações relevantes

O texto deste artigo é de responsabilidade dos autores, onde ferramentas de IA foram usadas apenas para revisão ortográfica e gramatical, gerar uma imagem e sugestões.

Referências

Alhammami, M. (2024). Fpga hardware kit for remote training platforms. *Discover Education*, 3(1):102. DOI: 10.1007/s44217-024-00203-w.

Bisong, E. (2019). Google colaboratory. *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pages 59–64. DOI: 10.1007/978-1-4842-4470-8_7.

Brunvand, E. (2011). Games as motivation in computer design courses: I/o is the key. In *ACM technical symposium on Computer science education*. DOI: 10.1145/1953163.1953178.

Burch, C. *et al*. (2024). Logisim evolution. Available at: `https://github.com/logisim-evolution/logisim-evolution`.

Bybell, T. (2023). Gtkwave: A vcd and lxt waveform viewer. Available at: `http://gtkwave.sourceforge.net`.

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 *International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE. DOI: 10.1109/iscas51556.2021.9401151.

Chinchanikar, A., Chandankhede, P. H., and Titarmare, A. (2023). Vga controller design & implementation on fpga. In *IEEE Global Conf. for Advancement in Technology (GCAT)*. DOI: 10.1109/GCAT59970.2023.10353298.

Combéfis, S., Beresnevičius, G., and Dagienė, V. (2016). Learning programming through games and contests: overview, characterisation and discussion. *Olympiads in Informatics*, 10(1). Available at: `https://ioinformatics.org/journal/v10_2016_39_60.pdf`.

Costa, A. S., Silveira, L. D., and Reis, A. I. (2023). Live demonstration: Pitanga platform for virtual fpga remote laboratories. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE. DOI: 10.1109/iscas46773.2023.10181345.

Cruz, C., Gil, R., de la Llana, A., Bravo, I., Gardel, A., and Lázaro, J. L. (2024). Remote laboratory based on a reconfigurable hardware platform. In *Congreso de Tecnología,*

Aprendizaje y Enseñanza de la Electrónica (TAEE). IEEE. DOI: 10.1109/taee59541.2024.10604937.

Dai, Z. and Cai, L. (2024). Autotrinity: An heterogeneous runner based remote digital system lab system. In *Int. Conference on Computer and Communications (ICCC)*. IEEE. DOI: 10.1109/iccc62609.2024.10941836.

Doulos (2024). Eda playground. Available at: `https://www.edaplayground.com`.

EDABoard Community (2024). Edaboard – forum for eda tools and hdl development. Available at: `https://www.edaboard.com`.

Falstad, P. (2024). Falstad circuit simulator. Available at: `https://www.falstad.com/circuit/`.

Ferreira, R., Canesche, M., Jamieson, P., Neto, O., and Nacif, J. (2024a). Examples and tutorials on using google colab and gradio to create online interactive student-learning modules. *Computer Applications in Engineering Education*. 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.

Guimarães, F. (2019). Como funciona o vga - mundo projetado. Available at: `https://mundoprojetado.com.br/como-funciona-o-vga/`.

Jarusauskas, A. (2009). Fpga based vga driver and arcade game. *University of Sussex*, 2010. Available at: `https://static.armandas.lt/res/fpga_based_vga_driver_and_arcade_game.pdf`.

Kent, S. L. (2010). *The ultimate history of video games, volume 1: From Pong to Pokemon and beyond... the story behind the craze that touched our lives and changed the world*. Crown. Book.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., *et al*. (2016). Jupyter Notebooks–a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agenda*. IOS Press. Available at: `https://eprints.soton.ac.uk/403913/1/STAL9781614996491-0087.pdf`.

Lima, I. D. A. (2021). *Scalable Web-Based FPGA Board Simulator*. Trabalho de conclusão de curso, UFCG. Available at: `http://dspace.sti.ufcg.edu.br/jspui/handle/riufcg/24990`.

Liu, C. C. (2018). Use of fpgas in a digital system design course with computer gaming applications. In *ASEE*. DOI: 10.18260/1-2–31188.

Martins Jacob, G., Lopes Mendes da Silva, L., and de Melo Barros Penze, L. (2024). Github - ic-unicamp/2024s1-mc613-projeto-elektras: Projeto final de mc613. Available at: `https://github.com/ic-unicamp/2024s1-mc613-projeto-elektras`.

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.

Mentor Graphics Corporation (2022). Modelsim user's manual. Available at: `https://eda.sw.siemens.com/en-US/ic/modelsim/`.

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

Neebel, D. J., Burek, N. J., and Griebel, T. (2012). Fpgarcade: Motivating the study of digital hardware. In *2012 ASEE Annual Conference & Exposition*, pages 25–648. DOI: 10.18260/1-2–21405.

Passe, F., Canesche, M., Neto, O. P. V., Nacif, J. A., and Ferreira, R. (2020). Mind the gap: Bridging verilog and computer architecture. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE. DOI: 10.1109/iscas45731.2020.9180650.

Perez, F. and Granger, B. E. (2007). Ipython: A system for interactive scientific computing. *Computing in Science & Engineering*, 9:21–29. DOI: 10.1109/mcse.2007.53.

Project, V. (2025). Vga playground. Available at: `https://vga-playground.com/` Accessed: 2025-08-03.

Qucs Team (2024). Quite universal circuit simulator. Available at: `https://qucs.sourceforge.net/`.

Research, G. (2017). Google colaboratory. `https://colab.research.google.com/`. Available at: `https://colab.research.google.com/`.

Sanchez-Elez, M. and Roman, S. (2015). Learning hardware design by implementing student's video-game on a fpga. In *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*, page 24. The Steering Committee of The World Congress in Computer Science, Computer …. Avaialble at: `https://www.researchgate.net/publication/283450129_Learning_Hardware_Design_by_implementing_student%27s_Video-Game_on_a_FPGA`.

Snyder, W. (2004). Verilator and systemperl. Available at: `https://veripool.org/papers/verilator_systemperl_nascug.pdf`.

Soares, J., Lobo, J., and DEEC, F. (2011). A remote fpga laboratory for digital design students. In *Portuguese meeting on reconfigurable systems (REC)*. Avaialble at: `https://www.researchgate.net/publication/228744476_A_Remote_FPGA_Laboratory_for_Digital_Design_Students`.

Stratton, J. (2024). An introduction to microsoft copilot. In *Copilot for Microsoft 365: Harness the Power of Generative AI in the Microsoft Apps You Use Every Day*. DOI: 10.1007/979-8-8688-0447-2_2.

Thompson, S. (1988). Vga—sign choices for a new video subsystem. *IBM Systems Journal*, 27:185–197. DOI: 10.1147/sj.272.0185.

Venn, M. D. (2024). Tiny tapeout: A shared silicon tapeout platform accessible to everyone. DOI: 10.36227/techrxiv.172055642.27780676/v1.

Williams, S. (2024). Icarus verilog. Available at: `https://github.com/steveicarus/iverilog`.

Williams, S. and Baxter, M. (2002). Icarus verilog: open-source verilog more than a year later. *Linux Journal*,

2002(99):3. Available at: https://dl.acm.org/doi/10.5555/513581.513584.

Wolf, C., Glaser, J., and Kepler, J. (2013). Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97. Available at: https://yosyshq.net/yosys/files/yosys-austrochip2013.pdf.

Xilinx Inc. (2023). Vivado design suite user guide. https://www.xilinx.com/products/design-tools/vivado.html. Available at: https://www.xilinx.com/products/design-tools/vivado.html.

Zakai, A. (2011). Emscripten: An llvm-to-javascript compiler. In *ACM Int Conf Companion on Object Oriented Programming Systems Languages and Applications Companion (OOPSLA '11)*. ACM. DOI: 10.1145/2048147.2048224.

Zakai, A. and contributors (2025). Emscripten official website. Available at: https://emscripten.org.