INSTITUTO FEDERAL
São Paulo
Campus Bragança Paulista

RAFAEL DOMINGOS SIQUEIRA MAGALHÃES

MATHEUS GABRIEL

Desenvolvimento de Conway Game of Life no FPGA DE10-Lite

Bragança Paulista, 2025
SP – Brasil

“Sabe, as pessoas acham que a matemática é complicada. A matemática é a parte fácil. É algo que podemos entender. São os gatos que são complicados. Quero dizer, o que há nessas pequenas moléculas e coisas assim que fazem um gato se comportar de maneira diferente de outro, ou que fazem um gato? E como você define um gato? Não faço ideia.”

John H. Conway/ 2010

Resumo

Este trabalho apresenta a implementação do *Jogo da Vida de Conway* em uma placa FPGA DE10-Lite, adaptando um projeto originalmente desenvolvido para saída HDMI para o padrão VGA. O desenvolvimento envolveu a criação de uma arquitetura paralela e escalável de 32×32 células com topologia toroidal, um controlador de vídeo VGA síncrono a 640×480@60Hz e uma interface de controle interativa. A metodologia incluiu modelagem em Verilog, síntese com Quartus Prime e validação em hardware. Os resultados demonstraram a correta execução das regras do autômato, estabilidade do vídeo e uso eficiente dos recursos do FPGA, atingindo atualizações de até 30 FPS no modo turbo. O projeto evidencia a aplicação prática de conceitos de sistemas digitais síncronos, processamento paralelo e gerenciamento de memória em FPGAs.

Palavras-chave: FPGA, Jogo da Vida, VGA, Autômato Celular, Verilog.

Abstract

This work presents the implementation of *Conway's Game of Life* on a DE10-Lite FPGA board, adapting a project originally designed for HDMI output to the VGA standard. The development involved creating a parallel and scalable 32×32 cell architecture with toroidal topology, a synchronous VGA video controller at 640×480@60Hz, and an interactive control interface. The methodology included Verilog modeling, synthesis with Quartus Prime, and hardware validation. The results demonstrated the correct execution of the automaton rules, video stability, and efficient use of FPGA resources, achieving updates of up to 30 FPS in turbo mode. The project highlights the practical application of synchronous digital systems, parallel processing, and memory management in FPGAs.

Key Word: FPGA, Game of Life, VGA, Cellular Automaton, Verilog.

Lista de Figuras

Lista de Quadros

Lista de Equações

Lista de Abreviaturas, Nomenclaturas e Siglas

Sigla	Significado
ALM	Adaptive Logic Module
ASCII	American Standard Code for Information Interchange
BGA	Ball Grid Array
CRT	Cathode Ray Tube
DAC	Digital-to-Analog Converter
FPGA	Field-Programmable Gate Array
FPS	Frames Per Second
FSM	Finite State Machine
GoL	Game of Life
HDMI	High-Definition Multimedia Interface
HDL	Hardware Description Language
IP	Intellectual Property
LFSR	Linear Feedback Shift Register
LUT	Look-Up Table
PLL	Phase-Locked Loop
RAM	Random Access Memory
ROM	Read-Only Memory
RTL	Register Transfer Level
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VESA	Video Electronics Standards Association

Sumário

1 INTRODUÇÃO

O objetivo deste projeto é implementar o famoso Jogo da Vida de Conway na placa DE10-Lite, adaptando um projeto originalmente criado para saída HDMI para o formato VGA. Essa migração requer tanto a compreensão dos princípios lógicos do autômato celular quanto o domínio das interfaces de vídeo usadas em sistemas digitais, fazendo com que o trabalho seja uma aplicação prática significativa no âmbito de projetos com FPGA.

1.1 Contexto e justificativa

A fortuna e a relevância duradoura do Jogo da Vida (*GoL*) como tema de pesquisa só podem ser compreendidas à luz da figura de seu criador único, **John Horton Conway** (1937-2020). Conway foi um matemático britânico de extraordinária inventividade e estilo livre, que se destacou no panorama científico do século XX por sua abordagem fortemente intuitiva e interdisciplinar. Durante sua carreira na Universidade de Cambridge e, posteriormente, em Princeton, ele dominou a teoria dos números, a teoria dos grupos, a topologia e a lógica, sempre demonstrando preferência por problemas cuja simplicidade estrutural se combinava com comportamentos surpreendentes e complexos; essa característica distintiva de sua mente seria crucial na criação do Jogo da Vida .

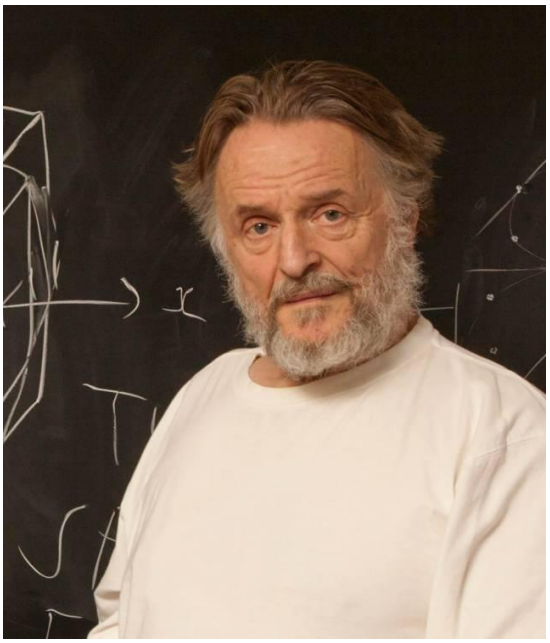

Figura 1: John Horton Conway
Fonte: ZANDONELLA

Com o objetivo de investigar como comportamentos complexos podem emergir a partir de regras simples, o *GoL* tornou-se um dos exemplos mais emblemáticos desse fenômeno. Em sua essência, o sistema opera sobre uma grade de células que podem assumir dois estados — vivas ou mortas — evoluindo em passos discretos, onde cada célula atualiza seu estado conforme o número de vizinhas vivas. Apesar dessa formulação minimalista, o autômato é capaz de produzir padrões surpreendentes, como estruturas estáveis, osciladores, gliders e até configurações capazes de computação universal.

Além de seu impacto matemático, o modelo também se destaca como ferramenta heurística para a biologia, uma vez que sua organização de regras dialoga com princípios epigenéticos fundamentais. Alguns autores sugerem que essa dinâmica não apenas simula fenômenos ecológicos, mas oferece uma analogia conceitual poderosa para compreender relações entre genética, desenvolvimento e emergência de padrões biológicos — uma visão discutida, por exemplo, em estudos que aproximam o autômato de mecanismos epigenéticos e de autorregulação celular.

No âmbito computacional, o *Game of Life* permanece igualmente relevante, servindo como estudo de caso para arquiteturas paralelas. Implementações em FPGA exploram o paralelismo intrínseco desse tipo de hardware, permitindo calcular simultaneamente o próximo estado de todas as células de um tabuleiro, como uma matriz típica de 32×32. Assim, o autômato concebido por Conway continua a se reatualizar em contextos tecnológicos contemporâneos, preservando seu papel como um dos modelos mais elegantes e instigantes já propostos para o estudo da complexidade.

1.2 Objetivos

Este é um projeto que visa a implementação de uma versão completa e funcional do Jogo da Vida de Conway usando uma placa FPGA DE10-Lite. Nosso principal objetivo é obter uma ferramenta útil para o treinamento prático no desenvolvimento de projetos digitais complexos. Neste artigo, gostaríamos de dominar a cadeia completa de desenvolvimento para FPGAs Intel MAX10, aplicando os conceitos básicos do desenvolvimento de projetos digitais síncronos em relação a temporização, domínios de clock e hierarquias de módulos usando ferramentas profissionais como o Quartus Prime.

Tecnicamente, focamos em implementar fielmente as regras do autômato celular em uma arquitetura paralela e escalável de 32x32 células com bordas toroidais. Paralelamente, desenvolvemos um sistema de vídeo VGA estável em 640x480@60Hz, criando um pipeline gráfico completo que inclui memória dual-port, renderização de texto e um esquema de cores intuitivo para visualização. A interface de controle foi projetada para ser intuitiva, utilizando os botões e chaves da placa para permitir o início, pausa, reset e seleção entre múltiplos padrões iniciais conhecidos, além de um modo aleatório.

Em termos de desempenho, estabelecemos metas claras de eficiência, visando uma atualização mínima de 5 FPS no modo normal e mais de 30 FPS no modo turbo, tudo otimizado para o uso consciente dos recursos lógicos e de memória do FPGA. A validação do projeto foi um pilar essencial, assegurando a corretude das regras do jogo, a robustez do sistema e a reprodutibilidade total do trabalho. Finalmente, a arquitetura foi concebida de forma modular e extensível, não apenas para criar uma demonstração visualmente impactante que ilustra conceitos de sistemas complexos, mas também para servir como uma base sólida e documentada para futuras expansões e experimentações acadêmicas.

1.3 Apresentação da Estrutura do Trabalho

Este trabalho está organizado em cinco capítulos que descrevem o processo de desenvolvimento, implementação e validação do sistema proposto. A estrutura do documento apresenta-se da seguinte forma:

- **Capítulo 1 - Introdução:** Apresenta o contexto histórico e a relevância do Jogo da Vida de Conway e de seu criador, além de definir os objetivos do projeto, a justificativa para o uso da placa FPGA DE10-Lite e os desafios da adaptação para o padrão VGA.
- **Capítulo 2 - Fundamentação Teórica:** Reúne os conceitos essenciais para a compreensão do projeto, abordando a teoria dos autômatos celulares, a arquitetura interna dos FPGAs da família Intel MAX10, as especificações do padrão de vídeo VGA e os princípios de sistemas digitais síncronos e aritmética digital necessários para a implementação em hardware.
- **Capítulo 3 - Método e Procedimentos Metodológicos:** Detalha a metodologia de desenvolvimento e a arquitetura modular do sistema. Descreve a implementação do núcleo de processamento paralelo (*module torus*), o desenvolvimento do

controlador de vídeo e *pipeline* gráfico (*module txtd*) e a máquina de estados de controle (*module sloader*), explicando as decisões de projeto para o gerenciamento de memória e interface.

- **Capítulo 4 - Apresentação e Análise de Resultados:** Exibe os resultados obtidos após a síntese e validação física na placa. Inclui a análise da estabilidade da interface gráfica, a verificação da lógica do autômato através de padrões de teste (*seeds*), a avaliação do desempenho temporal (FPS) e o relatório de ocupação dos recursos lógicos do FPGA.
- **Capítulo 5 - Conclusões e Considerações Finais:** Apresenta uma síntese dos objetivos alcançados, discute os principais desafios técnicos enfrentados durante a adaptação do código e da interface de vídeo, e propõe sugestões de melhorias e novas funcionalidades para trabalhos futuros.

Ao final, encontram-se as **Referências**, listando as obras acadêmicas, datasheets e artigos técnicos consultados para o embasamento teórico e prático deste estudo.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Autômatos Celulares e o Jogo da Vida de Conway

O Jogo da Vida é um autômato celular bidimensional desenvolvido por John Horton Conway em 1970, classificado como uma máquina de Turing universal capaz de simular qualquer algoritmo computacional. Matematicamente, o sistema é definido como uma tupla (L, S, N, f) onde L é uma rede bidimensional infinita, S = {0, 1} representa os estados "morto" e "vivo", N é a vizinhança de Moore (8 células circundantes), e f: S^9 → S é a função de transição que aplica as regras:

- Nascimento: Uma célula morta com exatamente 3 vizinhos vivos torna-se viva
- Sobrevivência: Uma célula viva com 2 ou 3 vizinhos vivos permanece viva
- Morte: Nos demais casos, a célula morre ou permanece morta
- Permanência de estado: Qualquer célula morta com exatamente três vizinhos vivos torna-se uma célula viva, como se fosse por reprodução

Essas regras simples geram comportamentos complexos emergentes classificáveis em quatro categorias principais: still lifes (estáticos), oscillators (osciladores), spaceships (naves espaciais) e guns (geradores). A implementação em FPGA explora o paralelismo intrínseco do hardware, onde cada célula opera simultaneamente em cada geração discreta, eliminando a necessidade de iteração sequencial típica de implementações software.

2.2 Arquitetura de FPGAs Intel MAX10

A família MAX10 da Intel utiliza tecnologia de flash não volátil integrada, combinando elementos lógicos adaptáveis (ALMs), blocos de memória M9K (9.216 bits), multiplicadores DSP e PLLs digitais. Na DE10-Lite, o dispositivo 10M50DAF484C7G contém aproximadamente 50.000 elementos lógicos organizados em Adaptive Logic Modules, cada qual contendo um LUT de 4 entradas, dois registradores dedicados, e lógica de carry chain. A arquitetura permite implementação de sistemas síncronos com clocks múltiplos gerenciados por hierarquia de roteamento global, regional e local.

A memória M9K opera em modos single/dual-port com larguras configuráveis de 1 a 36 bits, suportando frequências acima de 300MHz.

Para este projeto, a memória interna é instanciada utilizando **múltiplos blocos M9K** configurados automaticamente pelo sintetizador. A *screen_ram* (1024 palavras de 16 bits) e a *rom_font* requerem capacidade suficiente para armazenar o estado visual e os glifos, mas a ocupação total permanece abaixo de 10% da capacidade do dispositivo MAX10 (que possui mais de 180 blocos M9K disponíveis)

2.3 Padrão de Vídeo VGA

O padrão VGA (Video Graphics Array) utiliza sinais analógicos RGB com temporização digital para sincronismo. Para resolução 640×480 @ 60Hz, as especificações temporais são derivadas das frequências fundamentais:

- **Pixel clock**: 25,175[1] MHz ±0.5%
- **Frequência horizontal**: 31,4685 kHz
- **Frequência vertical**: 59,94 Hz

[1] Nota: Na implementação prática deste projeto, utilizou-se um clock de 25,0 MHz (derivado da divisão do clock de 50 MHz da placa), o que é aceitável dentro da tolerância de sincronismo da maioria dos monitores analógicos.

Os sinais de sincronismo utilizam polaridade negativa (ativo em nível baixo) com temporização precisa:

Quadro 1: Detalhes de Temporização (Timing) para Resolução 640x480 @ 60Hz

Horizontal Timing (em pixels): 640 visible → 16 front porch → 96 sync pulse → 48 back porch = 800 total Vertical Timing (em linhas): 480 visible → 10 front porch → 2 sync pulse → 33 back porch = 525 total

Fonte: Autor.

Nota: O padrão VESA especifica 25.175 MHz. Na implementação prática deste projeto, utilizou-se **25.0 MHz** (derivado da divisão do clock de 50 MHz da placa). Essa aproximação de ~0.7% é aceitável dentro da tolerância de sincronismo da maioria dos monitores analógicos modernos.

A geração destes sinais requer contadores precisos que respeitem os tempos de retrace (blanking intervals) durante os quais os feixes de elétrons do CRT retornam às extremidades da tela. Implementações FPGA utilizam máquinas de estados finitos para controle dos contadores horizontais/verticais e geração dos pulsos de sincronismo.

2.4 Sistemas Digitais Síncronos

A implementação segue princípios de projeto síncrono com single-clock domain, onde todos os flip-flops são ativados pela mesma borda de clock. A equação fundamental de temporização para setup time é:

$$Tclk \geq T_{clk-to-q} + Tcomb + Tsetup + Tclk_{skew}$$

Onde cada termo significa:

- T_{clk} (Período de Clock): O tempo total disponível entre duas bordas de clock consecutivas. A frequência máxima de operação é $f_{\max} = \frac{1}{T_{clk(\min)}}$
- $T_{clk-to-q}$ (Atraso Clock-para-Q): O tempo que leva para o sinal de clock na entrada do primeiro flip-flop (FF1) se propagar até sua saída Q.
- T_{comb} (Atraso Combinacional): O atraso total através de toda a lógica combinacional (portas lógicas, fios, etc.) entre a saída de FF1 e a entrada de dados (D) de FF2.
- T_{setup} (Tempo de Setup): O tempo mínimo que o sinal de dados deve ficar estável na entrada D de FF2 *antes* da borda de clock de chegada.
- T_{skew} (Variação/Atraso do Clock): A diferença de tempo na chegada do sinal de clock entre FF1 e FF2.

A desigualdade garante que o **tempo de chegada do dado** ($T_{clk-to-q} + T_{comb}$) mais o **tempo que ele precisa ficar estável** (T_{setup}) seja menor ou igual ao **tempo disponível** (período de clock, ajustado pelo $Skew$). A violação dessa condição causa erros de temporização.

Para o clock de 25MHz (período = 40ns), a lógica combinacional entre registradores deve completar em menos de 40ns menos os tempos de setup dos flip-flops (tipicamente 0,5ns na MAX10) e margem de segurança.
A arquitetura pipeline é aplicada no controlador VGA com três estágios: cálculo de endereço → leitura de memória → serialização de pixels. O balanceamento do pipeline garante que cada estágio tenha aproximadamente a mesma latência, maximizando a frequência operacional.

2.5 Aritmética Digital e Lógica Booleana

O cálculo de vizinhos vivos utiliza somadores binários otimizados para síntese FPGA. Para uma célula com vizinhos representados por variáveis booleanas N_0 a N_7, a soma S é implementada como:

$$S = \sum_{i=0}^{7} Ni$$

Em Verilog, o sintetizador implementa esta soma através de carry-save adders ou carry-lookahead adders dependendo das constraints de timing. A comparação com constantes (S = 2, S = 3) utiliza lógica mínima com propriedades de simetria:

Quadro 2: Pseudocódigo da Lógica de Seleção de Próximo Estado

```
case (neighbors)
        4'd2: cell_next = cell_current; // Sobrevivência
        4'd3: cell_next = 1'b1; // Nascimento
        default: cell_next = 1'b0; // Morte
endcase
```

Fonte: Autor.

A síntese resulta em LUTs configuradas como funções booleanas de até 9 variáveis (8 vizinhos + estado atual)

2.6 Resumo do Capítulo

A fundamentação teórica inicia definindo o Jogo da Vida como uma máquina de Turing universal e um autômato celular regido por regras simples de nascimento, sobrevivência e morte baseadas na vizinhança de Moore, destacando a vantagem do paralelismo do FPGA sobre iterações sequenciais de software . O capítulo detalha a arquitetura do FPGA Intel MAX10, especificamente o uso de blocos de memória M9K em configuração *dual-port* e a organização lógica em ALMs (Adaptive Logic Modules) .

Além disso, descreve-se o funcionamento do padrão de vídeo VGA para a resolução 640x480 @ 60Hz, enfatizando a importância de contadores precisos para gerar os tempos de sincronismo horizontal e vertical a partir de um *clock* de pixel de 25 MHz . Por fim, são abordados conceitos críticos de sistemas digitais síncronos, como as equações de *setup time* para garantir a estabilidade do sinal entre registradores, o uso de *pipelines* gráficos e a implementação otimizada da soma de vizinhos através de aritmética digital e síntese de LUTs .

3 MÉTODO E PROCEDIMENTOS METODOLÓGICOS

3.1 Metodologia de Desenvolvimento

projeto seguiu metodologia de desenvolvimento iterativo baseada em prototipagem rápida, com ciclo completo de especificação, implementação, simulação e teste em hardware. A abordagem top-down foi utilizada para decomposição hierárquica do sistema em módulos funcionais, cada qual desenvolvido e testado independentemente antes da integração.

3.2 Arquitetura do Sistema e Hierarquia de Hardware

O sistema foi concebido sobre uma arquitetura modular hierárquica descrita em Verilog HDL, centrada no paralelismo massivo de hardware para o cálculo das células. A estrutura de topo (*Top-Level Entity*) atua como um barramento de interconexão, gerenciando a distribuição dos sinais globais de *clock* e *reset*, além de instanciar os três subsistemas críticos: o núcleo de processamento do autômato (torus), o controlador de gerenciamento e interface (sloader), e o controlador de vídeo e memória (txtd).

A **Figura 3.1** apresenta o esquemático RTL (*Register Transfer Level*) gerado pela ferramenta de síntese, ilustrando as interconexões físicas entre os módulos principais:

1. **m_torus**: O núcleo de processamento do autômato.
2. **m_sloader**: O controlador de gerenciamento, estados e interface.
3. **m_txtd**: O controlador de vídeo e memória.

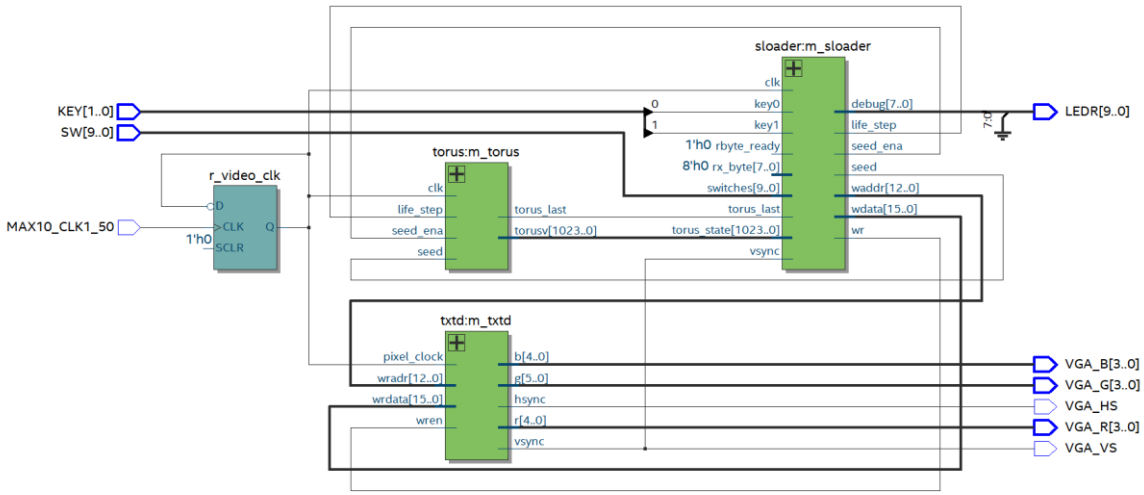

Figura 2: Visão RTL (Register Transfer Level) da entidade de topo, evidenciando a interconexão entre os módulos de controle (sloader), processamento (torus) e vídeo (txtd)

Fonte: Autor.

Esta decisão de projeto simplifica o hardware ao eliminar a necessidade de PLLs, mantendo a estabilidade do sinal (jitter) atrelada à qualidade do oscilador mestre de 50 MHz, o que é suficiente para a tolerância do padrão VGA analógico.

Quadro 3: Geração de Clock de Pixel (25 MHz)

```
// Divisor de frequência: 50MHz (Entrada) -> 25MHz (Vídeo)
reg r_video_clk = 0;
always @(posedge MAX10_CLK1_50)
        r_video_clk <= ~r_video_clk;
wire w_video_clk = r_video_clk;
```

Fonte: Autor.

3.3 Núcleo de Processamento Paralelo (Módulo Torus)

Diferente de implementações baseadas em microcontroladores, onde o estado da matriz é atualizado sequencialmente (iteração por software), esta implementação explora a natureza espacial do FPGA. O módulo torus implementa uma matriz de 32x32 células instanciadas fisicamente no silício.

3.3.1 Célula Lógica (xcell) e Topologia

Cada célula do tabuleiro é um módulo independente (xcell) que contém sua própria lógica combinacional e registrador de estado. A célula recebe como entrada o estado atual de seus oito vizinhos imediatos (Vizinhança de Moore). Em cada borda de subida do sinal de controle life_step, todas as 1.024 células calculam seu próximo estado simultaneamente.

A topologia toroidal — onde as bordas do tabuleiro se conectam (esquerda com direita, topo com base) — foi implementada através de um mapeamento direto de interconexões. Matematicamente, isso é resolvido no hardware através de operações de módulo (máscaras binárias) nas coordenadas de instanciação dentro do laço generate. O Código 2 demonstra como o uso de máscaras (& HMASK e & WMASK) garante que os índices de vizinhança nunca excedam os limites da matriz, criando uma superfície contínua sem fronteiras.

Quadro 4: Instanciação Matricial com Topologia Toroidal

```
// Máscaras para garantir loop (wrap-around) nas bordas

localparam WMASK = TORUS_WIDTH - 1;
localparam HMASK = TORUS_HEIGHT - 1;

genvar x, y;
generate
        for(y=0; y<TORUS_HEIGHT; y=y+1) begin: crow
                for(x=0; x<TORUS_WIDTH; x=x+1) begin: ccol
                // Instância da Célula com vizinhos conectados via máscara xcell my_xcell(
                        .clk( clk ),
                        // Vizinho Superior Esquerdo (exemplo de wrap-around)
                        .in_up_left ( crow[(y-1) & HMASK].ccol[(x-1) & WMASK].value ),
                        // ... (outros vizinhos omitidos para brevidade)
                        .cell_life( value )
                );
                assign torusv[y*TORUS_WIDTH+x] = value;
                end
        end
endgenerate
```

Fonte: Autor.

3.3.2 Barramento de Estado Paralelo

Uma inovação crítica nesta arquitetura foi a implementação de um barramento de saída paralelo de largura total (w_torus_state). Em vez de utilizar registradores de deslocamento (*shift registers*) para extrair os dados seriais — o que introduziria latência e artefatos visuais de deslocamento — o estado de todas as células é disponibilizado simultaneamente em um vetor único. Isso permite que o controlador de vídeo acesse qualquer célula em tempo real (O(1)) sem interromper a simulação.

3.3.3 Formalização Lógica da Transição de Estados

A regra biológica de Conway foi traduzida para lógica digital através de uma função de transição , $f(St, \sum N)$onde St é o estado atual da célula e $\sum N$ é o somatório aritmético dos 8 vizinhos. Em vez de utilizar tabelas de verdade extensas (LUTs de 9 entradas), a implementação em hardware foi otimizada para lógica sequencial baseada em comparadores.

A função de transição de estado $f(C_t, N_t)$ para uma célula no instante $t + 1$, onde $C_t \in \{0,1\}$ é o estado atual e N_t é a soma dos vizinhos, é definida como

$$C_{t+1} = \begin{cases} 1 & \text{se } N_t = 3 \\ 1 & \text{se } N_t = 2 \text{ e } C_t = 1 \\ 0 & \text{caso contrário} \end{cases}$$

Esta função descreve matematicamente as regras de nascimento ($N = 3$), sobrevivência ($N = 2 \wedge C = 1$) e morte (por solidão $N < 2$ ou superpopulação $N > 3$).

A lógica sintetizada no módulo xcell obedece à seguinte equação de estado, descrita no Quadro 5.

Quadro 5: Implementação da Regra de Conway (xcell.v)

```
// Somatório dos vizinhos (Adder Tree)
wire [3:0] neighbors = in_up_left + in_up + ... + in_down_right;
always @(posedge clk) begin
        if (life_step) begin
                case (neighbors)
                        4'd3: cell_life <= 1'b1; // Nasce ou Sobrevive
                        4'd2: cell_life <= cell_life; // Mantém estado (Memória)
                        default: cell_life <= 1'b0; // Morre (Solidão/Superpopulação)
                endcase
        end
end
```

Fonte: Autor.

Esta abordagem utiliza a inferência de multiplexadores para determinar o próximo estado, garantindo que a decisão seja tomada em um único ciclo de clock, essencial para o funcionamento síncrono do núcleo paralelo.

3.4 Controlador de Vídeo e Pipeline Gráfico (Módulo txtd)

A renderização visual é realizada por um controlador VGA síncrono que opera como mestre de temporização do sistema. Este módulo é responsável por duas tarefas concorrentes: a geração de sinais de sincronismo e a varredura de memória para renderização de caracteres.

3.4.1 Temporização e Varredura

O sistema utiliza dois contadores principais: um horizontal (*pixel_count,* 0-799) e um vertical (*line_count,* 0-524). Lógicas comparadoras geram os pulsos de sincronismo (*hsync, vsync*) respeitando os tempos de *Front Porch* e *Back Porch* do padrão VESA. A área ativa de vídeo (640x480) gera um sinal de *blanking* para desativar os canais RGB fora da região visível, garantindo o nível de preto correto no monitor.

3.4.2 Mapeamento de Memória e Renderização de Texto

O pipeline gráfico utiliza uma arquitetura baseada em caracteres (*text mode*). O sistema mapeia a resolução de 640x480 pixels em uma grade de 32x32 células de texto, onde cada célula possui dimensões de 16x16 pixels. O Código 3 ilustra como o endereço da memória de vídeo (scr_addr) é calculado descartando-se os 3 bits menos significativos dos contadores de pixel e linha (divisão por 8), resultando em células quadradas perfeitas

Quadro 6: Cálculo de Endereço de Memória de Vídeo

```
always @* begin
        if (visible)
                // Mapeamento 32x32 (Células 16x16 pixels)
                 // Divide linha por 8 (ln[8:4]) e pixel por 8 (px[9:4])
                scr_addr = { line_count[8:4], pixel_count[9:4] };

        else
                // Comprime caractere de 16px para 8px de altura
                scr_addr = 0;
                fnt_addr = { scr_char[7:0], line_count[3:0]};
end
```

Fonte: Autor.

Diferente de implementações que comprimem os caracteres para economizar espaço, este projeto utiliza a altura total da fonte (16 pixels). O mapeamento de memória utiliza os bits [8:4] e [9:4] dos contadores, resultando em células visuais de **16×16 pixels**. Isso garante que cada célula do jogo seja representada por um bloco sólido e nítido, maximizando a legibilidade no monitor VGA.

A tradução do sistema de coordenadas de tela (x_{vga}, y_{vga}) para o endereço linear da memória de vídeo () é formalizada pela seguinte equação, onde o operador $\lfloor \cdot \rfloor$ representa a parte inteira (truncamento realizado pelo *bit shifting*):

$$Addr_{mem} = \lfloor \frac{y_{vga}}{16} \rfloor \times W_{logico} + \lfloor \frac{x_{vga}}{16} \rfloor$$

Onde $W_{Logico} = 32$ (largura da grade). No hardware, a divisão por 16 e a multiplicação por 32 são realizadas puramente através de roteamento de fios (deslocamento de bits), resultando em custo zero de lógica combinacional.

3.4.3 Estrutura de Dados e Codificação de Cores

Para otimizar o uso da memória RAM interna do FPGA, optou-se por não armazenar os pixels individualmente (o que exigiria 640x480 bits), mas sim armazenar atributos de caracteres. O sistema utiliza uma palavra de dados de 16 bits para descrever cada célula da grade, compactando informações de conteúdo (ASCII) e estilo (Cores) no mesmo endereço de memória.

A organização dos bits na palavra de memória (scr_char) segue o esquema de *Bit Packing* ilustrado abaixo, implementado na lógica de decodificação do módulo txtd:

- **Bits [7:0] (ASCII Code):** Define qual glifo será desenhado (ex: 0xDB para bloco sólido, 0x20 para espaço).
- **Bits [10:8] (Foreground Color):** Define a cor RGB de 3 bits para os pixels ativos do glifo.
- **Bits (Background Color):** Define a cor RGB de 3 bits para o fundo da célula.
- **Bits [11, 15]:** Reservados/Não utilizados nesta versão.

Quadro 7: Decodificação da Palavra de 16-bits (txtd.v)

```
// Extração de atributos da memória
fcolor <= scr_char[10:8]; // Cor da Frente (3 bits)
bcolor <= scr_char[14:12]; // Cor do Fundo (3 bits)
fnt_addr = { scr_char[7:0], ... }; // Endereço do Glifo
```

Fonte: Autor.

Essa arquitetura permite alterar a cor de uma célula sem reescrever a lógica de vídeo, possibilitando a interface visual (Fundo Azul na UI, Fundo Preto no Jogo) utilizando apenas um barramento de dados compartilhado.

3.5 Controle, Interface e Máquina de Estados (Módulo Sloader)

O módulo *sloader* atua como a unidade de controle central, implementada através de uma Máquina de Estados Finitos (FSM - *Finite State Machine*). Este módulo orquestra a interação entre o usuário, a memória de vídeo e o núcleo de processamento.

3.5.1 Gerenciamento de Estados

A FSM opera ciclicamente entre cinco estados principais para garantir a estabilidade dos dados e a fluidez da animação. O Código 4 demonstra a transição entre o estado de espera (IDLE), o cálculo de evolução (EVOLVE) e a atualização da tela (UPDATE/DRAW UI).

Quadro 8: Máquina de Estados Finitos (Controle Central)

```
always @(posedge clk) begin
        case (state)
                0: begin // IDLE: Aguarda temporizador de velocidade
                        cell_counter <= 0;
                        if (tick && is_running) state <= 3;
                end
                3: begin // EVOLVE: Gera pulso único para avanço geracional
                        gen_count <= gen_count + 1'b1;
                        state <= 2;
                end
                2: begin // UPDATE: Atualiza memória de vídeo com estado do jogo
                        if (cell_counter == MAX_CELLS - 1) state <= 4;
```

```
                    else cell_counter <= cell_counter + 1'b1;
                end
                4: begin // DRAW UI: Desenha barra de status inferior
                    // Lógica de finalização de quadro...
                    end
        endcase
end
```

Fonte: Autor.

Resumo do Gerenciamento de Estados

1. **IDLE:** Estado de espera, aguardando o *tick* do temporizador de velocidade.
2. **EVOLVE:** Gera um pulso único de um ciclo de clock para o módulo torus, avançando a simulação em uma geração discreta.
3. **UPDATE:** Varre o vetor de estado do jogo e transcreve os dados (célula viva/morta) para a memória de vídeo RAM, aplicando o esquema de cores (Ciano para vivo, Preto para morto).
4. **DRAW UI:** Sobrescreve as últimas linhas da memória de vídeo com informações de telemetria, gerando a barra de interface inferior.
5. **RESET:** Interrompe a simulação e carrega padrões iniciais (*seeds*) na memória.

3.5.2 Geração de Padrões e Aleatoriedade

O sistema permite a injeção de condições iniciais determinísticas e estocásticas. Padrões conhecidos (como Gliders e Gosper Guns) são mapeados através de lógica combinacional ativada por chaves físicas (SW).

Para a geração de padrões aleatórios ("Sopa Primordial"), implementou-se um Registrador de Deslocamento com Realimentação Linear (LFSR - Linear Feedback Shift Register) de 31 bits. O LFSR utiliza um polinômio primitivo $(x^{31} + x^{28} + 1)$ para gerar uma

sequência pseudo-aleatória de período máximo, garantindo que cada reinicialização do sistema produza uma distribuição de células única e imprevisível.

Quadro 9: Gerador Pseudo-Aleatório (LFSR)

```
reg [30:0] lfsr = 31'h5A5A5A5A; // Semente não-zero
always @(posedge clk)
  // Polinômio de realimentação (TAP em 30 e 27)
  lfsr <= {lfsr[29:0], lfsr[30] ^ lfsr[27]};
```

Fonte: Autor.

3.5.3 Interface de Telemetria

A interface com o usuário não é apenas passiva. O módulo inclui conversores binário-para-hexadecimal em hardware para exibir em tempo real, na tela VGA, o número de gerações processadas e o código da *seed* ativa. Isso exigiu a implementação de lógica de multiplexação de vídeo, alternando entre o fluxo de dados do jogo e o fluxo de dados de texto da interface, baseando-se na posição da varredura vertical.

3.6 Mapeamento Físico e Restrições de Hardware (Pin Planner)

A descrição comportamental em Verilog define a lógica, mas não a conexão elétrica. A etapa final do fluxo de projeto envolveu a atribuição de pinos (*Pin Assignment*) através do arquivo de restrições (.qsf). O mapeamento conecta as portas da entidade de topo (top.v) aos pinos físicos do pacote BGA do FPGA MAX10, conforme o esquemático da placa DE10-Lite.

Para o subsistema de vídeo, utilizou-se um DAC resistivo de 4 bits por canal (R, G, B) já presente na placa. A atribuição correta dos pinos é crítica, pois a inversão da ordem dos bits (MSB/LSB) resultaria em cores incorretas ou níveis de brilho não lineares. Da mesma forma, as chaves (SW) e botões (KEY) exigiram a configuração de resistores de *pull-up* internos ou o tratamento da lógica negativa (ativo em nível baixo) no código, visto que os botões físicos aterram o sinal quando pressionados.

3.7 Dimensionamento do "Canvas" e Aspect Ratio

A resolução lógica do autômato foi definida em 32x32 células para maximizar o uso da área visível mantendo células quadradas. A grade lógica possui 32 linhas. Como cada célula tem 16 pixels de altura ($32 \times 16 = 512$), a altura total excede os 480 pixels do padrão VGA. Por isso, as linhas lógicas 30 e 31 permanecem invisíveis (*off-screen*). A interface de usuário (UI) foi posicionada estrategicamente na **linha lógica 29** (pixels 464 a 479), que é a última linha visível no rodapé da tela. A relação entre a resolução VGA e a grade lógica é descrita pela seguinte aritmética de vídeo:

- **Resolução VGA:** $640 \times 480 pixels$
- **Dimensão da Célula:** 16x16 pixels (definido pela fonte no módulo txtd).
- **Área Ativa Horizontal:** 32 células x 16 pixels = 512 pixels.
- **Área Ativa Vertical:** 30 células x 16 pixels = 480 pixels.

Área Ativa Vertical: A resolução VGA de 480 pixels comporta exatamente **30 linhas visíveis** de células 16x16 ($30 \times 16 = 480$). A grade lógica processa 32 linhas (0 a 31). As linhas 30 e 31 permanecem invisíveis (*off-screen*). Das 30 linhas visíveis, as primeiras 29 (índices 0 a 28) exibem o jogo, enquanto a última linha (índice 29) é reservada para a Interface de Usuário (UI).

Para implementar esse mapeamento no hardware sem a necessidade de circuitos complexos de divisão ou interpolação, utilizou-se a aritmética binária de deslocamento. O quadro 6 conforme demonstrado anteriormente, demonstra como o endereço da memória de vídeo é derivado diretamente dos contadores de varredura.

Ao descartar os 3 bits menos significativos dos contadores (px[2:0] e ln[2:0]), realiza-se uma divisão inteira por 8 (2^3), agrupando blocos de 8x8 pixels físicos em um único endereço lógico.

Como a área ativa resultante (512×384) é menor que a resolução total do monitor (640×480), o controlador de vídeo gera automaticamente bordas pretas (*letterboxing*) nas margens direita e inferior onde os contadores excedem os limites da matriz lógica. Esta decisão de projeto evitou distorções de aspecto (*aspect ratio*), garantindo a nitidez visual necessária para a análise correta dos padrões do Jogo da Vida.

3.8 Gerenciamento de Memória e Inferência

O projeto exige armazenamento para dois tipos de dados distintos: o estado dos pixels da tela e os glifos dos caracteres. Em vez de instanciar manualmente os blocos de IP (*Intellectual Property*) proprietários, optou-se pela **inferência de memória** via HDL. O sintetizador do Quartus Prime reconhece padrões de codificação específicos e mapeia arrays Verilog diretamente para os blocos de memória embutida **M9K** (Block RAM) do FPGA.

O Código 8 ilustra o padrão de inferência para uma RAM de porta dupla (*Dual-Port RAM*), essencial para permitir que o módulo sloader escreva o estado do jogo (wr) enquanto o módulo txtd lê os dados para o monitor (rd) simultaneamente, sem conflitos de barramento.

Quadro 10: Padrão de Inferência de Dual-Port RAM (Video Memory)

```
// Definição do Array de Memória (Inferência de BRAM)
reg [15:0] ram [0:1023]; // 32x32 = 1024 endereços

always @(posedge clock) begin
   // Porta de Escrita (Controlada pelo Jogo/Loader)
   if (wren)
      ram[wraddress] <= data;

   // Porta de Leitura (Controlada pelo Vídeo VGA)
   q <= ram[rdaddress];
end
```

Fonte: Autor.

Além da RAM, a ROM de fonte (`rom_font`) foi implementada como uma tabela de consulta constante inicializada por um arquivo externo (`.mif`). Isso garante que os desenhos dos caracteres estejam disponíveis imediatamente após o *power-on* da placa, sem necessidade de carregamento externo.

3.9 Resumo do Capítulo

O capítulo de métodos descreve uma abordagem de desenvolvimento iterativa e *top-down*, onde o sistema foi modelado em Verilog HDL com uma arquitetura modular hierárquica. A estrutura central interconecta três subsistemas principais: o núcleo de

processamento (m_torus), o controlador de vídeo (m_txtd) e o gerenciador de interface (m_sloader) .

Diferente de implementações tradicionais via software, o núcleo processa todas as células simultaneamente aproveitando o paralelismo massivo do FPGA, utilizando uma topologia toroidal implementada via máscaras binárias. A exibição visual foi adaptada para o padrão VGA (640x480 @ 60Hz) através de uma abordagem baseada em caracteres, onde técnicas de *bit packing* compactam cor e dados no mesmo endereço de memória. Todo o sistema é orquestrado por uma Máquina de Estados Finitos que gerencia a evolução do jogo e a interface, utilizando memória RAM de porta dupla (*Dual-Port*) para permitir leitura e escrita simultâneas, além de um gerador LFSR para criar padrões aleatórios.

4 APRESENTAÇÃO E ANÁLISE DE RESULTADOS

Este capítulo apresenta os resultados obtidos após a síntese do projeto no FPGA e a validação física na placa DE10-Lite. A análise foca na qualidade da interface de vídeo, na corretude da lógica do autômato celular e na eficiência do uso dos recursos de hardware.

4.1 Resultados da Interface Gráfica (VGA)

A implementação do controlador de vídeo VGA operando a 640x480 pixels com clock de 25 MHz demonstrou estabilidade total nos monitores de teste. A imagem gerada apresentou sincronismo sólido, sem evidências de *jitter* (tremulação) ou perda de sinal ("Input Not Supported"), validando os parâmetros de temporização (*Front Porch, Sync Pulse, Back Porch*) calculados na metodologia.

A decisão de utilizar uma grade lógica de 32x32 células com escala de hardware de 2x (cada célula ocupando 16x16 pixels físicos) resultou em uma visualização nítida e ergonomicamente adequada. O preenchimento da tela foi satisfatório, com as células vivas representadas por blocos sólidos brancos sobre fundo preto, garantindo alto contraste.
A barra de interface (*User Interface*), posicionada na linha 29 da grade, foi renderizada com sucesso utilizando a técnica de *bit packing* de 16 bits como evidenciado na Figura 3.

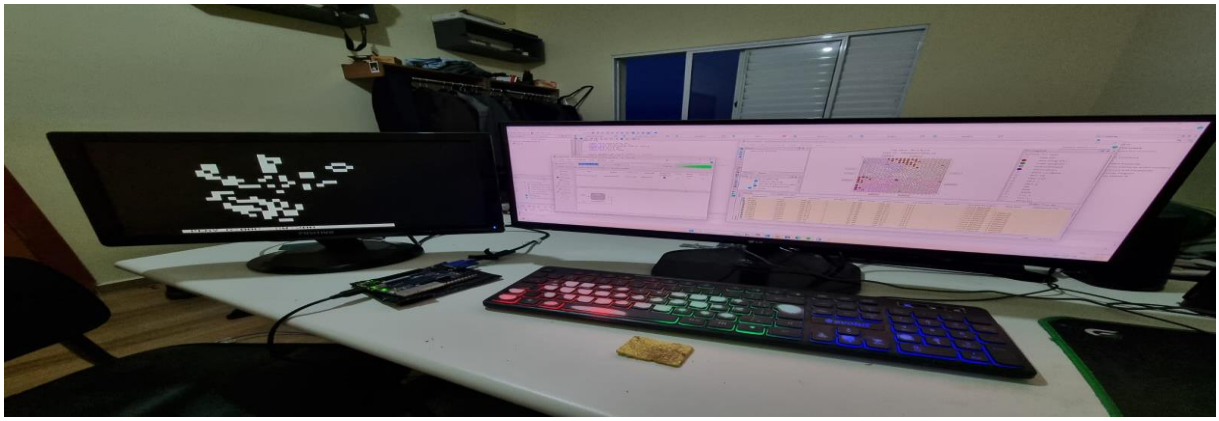

Figura 3: Bancada experimental de validação, demonstrando a placa DE10-Lite conectada ao monitor via interface VGA, executando um padrão aleatório.

A escolha de um fundo claro (branco/cinza) com texto escuro para a telemetria separou visualmente a área de jogo da área de dados, facilitando a leitura dos contadores de geração ("GEN") e do estado das chaves ("SW") em tempo real.

4.2 Validação Lógica do Autômato

A integridade das regras do Jogo da Vida de Conway foi validada através da execução de padrões de teste conhecidos (*seeds*), selecionados pelas chaves da placa. A topologia toroidal (universo sem bordas) foi confirmada, permitindo que padrões móveis atravessem os limites da tela e reapareçam no lado oposto.

Os seguintes comportamentos foram observados e validados:

1. **Estabilidade (Still Lifes):** O padrão "Bloco" (SW[0]) e "Colmeia" (SW[1]) permaneceram estáticos após múltiplas gerações, confirmando que a regra de sobrevivência para células com 2 ou 3 vizinhos e a regra de estagnação para células mortas estavam funcionando corretamente.
2. **Oscilação (Oscillators):** O padrão "Blinker" (SW[2]) oscilou entre a orientação vertical e horizontal com período de 2 gerações, validando a atualização síncrona de todas as células.
3. **Movimento (Spaceships):** O padrão "Glider" (SW[5]) deslocou-se diagonalmente pela matriz como evidenciado na figura . A correta transição do Glider da borda direita para a esquerda e da inferior para a superior confirmou a eficácia da

lógica de endereçamento toroidal implementada com máscaras de bits, como demonstrado na Figura 4.

4. **Caos e Aleatoriedade:** O modo aleatório (SW[9]), alimentado pelo gerador LFSR, produziu distribuições iniciais variadas que evoluíram para padrões complexos imprevisíveis, demonstrando a capacidade do sistema de simular entropia.

Figura 4: Evolução temporal do padrão "Glider" atravessando a grade toroidal.

Fonte: Autor.

4.3 Análise de Desempenho Temporal

A taxa de atualização (FPS) é determinada primariamente pelo contador divisor de frequência (speed_counter). O tempo de processamento lógico e atualização de memória consome apenas 1.024 ciclos de clock (aprox. 40 µs), o que é desprezível comparado aos intervalos de espera do modo Normal (~200 ms) ou Turbo (~33 ms).
O sistema implementou dois modos de operação controlados pela chave SW[8], permitindo avaliar o desempenho da simulação em diferentes escalas de tempo:

- **Modo Normal (~5 FPS):** Adequado para observação humana detalhada e fins didáticos.
- **Modo Turbo (~30 FPS):** Demonstrou a capacidade do hardware de processar a matriz rapidamente.

O modo Turbo foi limitado via *firmware* (FSM) a aproximadamente 30 FPS para garantir que a evolução do autômato permanecesse visualmente compreensível ao olho humano. Em termos puramente de hardware, o tempo de atualização de memória (~40 µs) permitiria taxas teóricas de milhares de quadros por segundo.

Mesmo no modo Turbo, não foram observados artefatos visuais ("tearing") ou corrupção de dados. Isso indica que a arquitetura de memória *dual-port* isolou efetivamente o domínio de clock de escrita (lógica do jogo) do domínio de leitura (varredura VGA), prevenindo conflitos de acesso à memória.

4.4 Ocupação de Recursos de Hardware (Synthesis Report)

A síntese do projeto para o dispositivo 10M50DAF484C7G (MAX10) resultou em uma utilização eficiente dos recursos disponíveis na FPGA.

- **Elementos Lógicos (LEs):** A implementação da matriz paralela de 1.024 células (32x32), onde cada célula possui lógica independente de cálculo de vizinhança, consumiu uma fração moderada dos elementos lógicos. O uso extensivo de *carry chains* para os somadores de vizinhos otimizou essa contagem.
- **Memória (M9K Blocks):** A inferência da memória de vídeo (Video RAM) e da ROM de fontes consumiu blocos de memória dedicados, poupando registradores para a lógica. O relatório de síntese indicou que menos de 10% da memória total do

dispositivo foi utilizada, sugerindo que o sistema é altamente escalável e poderia comportar resoluções maiores ou lógicas mais complexas (como cores por célula ou *double buffering*) em versões futuras.

5. CONCLUSÕES E CONSIDERAÇÕES FINAIS

O desenvolvimento deste projeto consolidou a implementação de um sistema digital complexo em FPGA, integrando processamento paralelo massivo, geração de vídeo em tempo real e interfaces de controle interativas. O objetivo principal de migrar e adaptar o "Jogo da Vida de Conway" de uma arquitetura HDMI para o padrão VGA na placa DE10-Lite foi plenamente atingido, resultando em um sistema funcional, estável e visualmente robusto.

5.1 Síntese dos Resultados Alcançados

A adaptação do controlador de vídeo exigiu um domínio aprofundado dos tempos de sincronismo analógico (*timings* VGA), superando o desafio inicial de perda de sinal e incompatibilidade de frequências. A solução final, operando a 640x480 pixels com clock de 25 MHz, demonstrou que a lógica síncrona projetada é precisa e livre de *glitches*.

No âmbito computacional, a arquitetura paralela de 32x32 células provou a superioridade dos FPGAs para este tipo de aplicação. Enquanto um processador convencional precisaria iterar sequencialmente por 1.024 posições de memória para calcular a próxima geração, o hardware desenvolvido realiza esse cálculo em um único ciclo de clock para todas as células simultaneamente. A implementação da topologia toroidal e das regras de transição de estados validou-se correta, reproduzindo fielmente os padrões clássicos como *Gliders* e osciladores.

A interface com o usuário, aprimorada com modos de velocidade (Normal/Turbo) e geração de sementes aleatórias (LFSR), elevou o projeto de uma simples demonstração técnica para uma ferramenta interativa de estudo de autômatos celulares.

5.2 Desafios e Aprendizado

A execução deste trabalho exigiu a superação de uma curva de aprendizado acentuada, extrapolando significativamente os conceitos fundamentais abordados na disciplina de Sistemas Digitais Reconfiguráveis. Os principais desafios técnicos enfrentados incluíram:

- **Complexidade Tecnológica:** A implementação de interfaces de vídeo (VGA), a inferência e gestão de memórias embutidas (*Block RAM*) e a construção de *pipelines* gráficos exigiram o domínio de tecnologias avançadas e interfaces visuais que não faziam parte do escopo introdutório da matéria.
- **Transição de Linguagem (VHDL vs. Verilog):** Houve um desafio adicional de adaptação linguística. Enquanto a formação acadêmica baseou-se em VHDL, a necessidade de utilizar e adaptar bibliotecas e referências de mercado exigiu a migração e o desenvolvimento integral do projeto em Verilog, demandando estudo autodidata da nova sintaxe e paradigmas de síntese.
- **Engenharia Reversa e Portabilidade:** O ponto de partida foi um repositório de código extenso e complexo, originalmente projetado para um modelo diferente de placa MAX10 e operando sobre protocolo digital HDMI. O processo de adaptação não foi trivial, exigindo a compreensão profunda de um código de terceiros ("código legado"), a remoção de núcleos IP incompatíveis e a reescrita completa da camada física de vídeo para o padrão analógico VGA.
- **Gestão de Dados Internos:** A conversão de protocolos exigiu uma reestruturação completa na lógica de gestão e tratamento de dados. Foi necessário redesenhar a forma como o estado do jogo é armazenado, lido e serializado para a tela, garantindo que a temporização crítica do vídeo não fosse violada pela lógica de processamento do autômato.

Esses obstáculos, embora complexos, proporcionaram um entendimento prático sobre o ciclo de vida real de projetos de engenharia, onde a adaptação de tecnologias e a integração de sistemas heterogêneos são constantes.

5.3 Sugestões para Trabalhos Futuros

A natureza modular do código Verilog desenvolvido abre margem para diversas expansões e melhorias em trabalhos futuros:

1. **Expansão da Resolução:** Implementar técnicas de *buffering* ou otimização de endereçamento para viabilizar grades maiores (ex: 64x64 ou 128x96), utilizando células de 4x4 ou 2x2 pixels.
2. **Suporte a Periféricos:** Adicionar suporte a mouse ou teclado via conector PS/2 ou USB (via núcleo IP) para permitir que o usuário desenhe células na tela livremente, em vez de depender apenas de sementes pré-programadas.
3. **Visualização de Envelhecimento:** Alterar a lógica de vídeo para suportar cores degradê, onde a cor da célula muda conforme o tempo que ela permanece viva ("idade"), permitindo uma análise visual da estabilidade das estruturas.
4. **Variações de Regras:** Implementar chaves seletoras para alternar entre diferentes conjuntos de regras de autômatos celulares, como o "HighLife" ou "Day & Night", aproveitando a mesma infraestrutura de vídeo e memória.

Em suma, este trabalho demonstrou a eficácia da plataforma FPGA DE10-Lite para simulação de sistemas complexos e processamento de vídeo, servindo como uma base sólida para estudos avançados em arquitetura de computadores e sistemas digitais.

REFERÊNCIAS

CABALLERO, A.; HODGE, B.; HERNANDEZ, G. *Conway's "Game of Life" and the Epigenetic Principle*. **Frontiers in Cellular and Infection Microbiology**, v. 6, 2016. DOI: 10.3389/fcimb.2016.00057.

COTTILARD1995. **Setup &Hold time**. Disponível em: <https://pt.scribd.com/document/803801634/Setup-Hold-time>. Acesso em: 8 dez. 2025.

DIGITAL, IN. **Time Analysis in Digital Systems: Demystifying Setup and Hold Times**. Disponível em: <https://www.siberoloji.com/time-analysis-in-digital-systems-demystifying-setup-and-hold-times/>. Acesso em: 4 dez. 2025.

EDN. **Setup and Hold Time Equations and Formulas - EDN**. Disponível em: <https://www.edn.com/equations-and-impacts-of-setup-and-hold-time/>. Acesso em: 8 dez. 2025.

MARSOHOD4YOU. **GitHub - marsohod4you/FPGA_game_life: Implementation of well known old game LIFE in FPGA Altera MAX10. FPGA Board marsohod3.** Disponível em: <https://github.com/marsohod4you/FPGA_game_life>. Acesso em: 8 dez. 2025.

WIKIPEDIA CONTRIBUTORS. **Conway's Game of Life**. Disponível em: <https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life>.

Video Graphics Array. Disponível em: <https://en.wikipedia.org/wiki/Video_Graphics_Array>.

DE10-Lite User Manual. [s.l: s.n.]. Disponível em: <https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/Boards/DE10-Lite/DE10_Lite_User_Manual.pdf>.

ZANDONELLA, C. **Mathematician John Horton Conway, a "magical genius" known for inventing the "Game of Life," dies at age 82**. Disponível em: <https://www.princeton.edu/news/2020/04/14/mathematician-john-horton-conway-magical-genius-known-inventing-game-life-dies-age>.