

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
Departamento de Computação

Curso de Engenharia de Computação



Relatório de Atividades Práticas

Implementação de Autômato Celular em FPGA com Interface VGA

Alunos:

Lucas Gabriel Valenti
Jonas Gabriel Fagundes
Lucas Mantovani

Disciplina: Arquiteturas de Alto Desempenho
Professor: Prof. Dr. Emerson Carlos Pedrino
Data: 19 de Junho de 2025

"Simple, clear purpose and principles give rise to complex intelligent behavior. Complex rules and regulations give rise to simple stupid behavior."

- Dee Hock

Sumário

1	Introdução	3
2	Desenvolvimento Teórico	3
2.1	O Jogo da Vida de Conway	3
2.1.1	Regras de Evolução	3
2.1.2	Exemplos de Padrões Clássicos	3
2.2	Autômatos Celulares em Arquiteturas Paralelas	6
2.2.1	Vizinhança de Moore	6
2.2.2	Atualização Síncrona em Sistemas Embarcados	7
2.3	Imagen Binária em VGA	7
2.3.1	Funcionamento de um Pixel	7
2.3.2	O Cabo VGA	8
3	Desenvolvimento Prático	9
3.1	Divisão em Módulos no Quartus	9
3.1.1	Módulo VGA	9
3.1.2	Módulo de Janela	12
3.1.3	Módulo Redutor de Clock	13
3.1.4	Multiplexador de Clock	14
3.1.5	Máquina de Estados	15
3.1.6	Módulo Ampliador de Pixels	17
3.2	Funcionamento Geral	18
4	Discussão dos Resultados	21
4.1	Funcionamento Geral do Sistema	21
4.2	Eficiência Visual da Imagem Ampliada	23
4.3	Interatividade com o Usuário	23
5	Conclusões	23

1 Introdução

Com o avanço das tecnologias embarcadas e o crescimento da demanda por sistemas capazes de processar informações em tempo real, tornou-se cada vez mais relevante o estudo e a implementação de arquiteturas digitais otimizadas para tarefas específicas. Nesse contexto, os autômatos celulares surgem como uma classe de modelos computacionais simples, porém expressivos, capazes de representar comportamentos complexos a partir de regras locais.

Entre os autômatos celulares mais conhecidos está o *Jogo da Vida de Conway*, uma simulação bidimensional onde cada célula evolui conforme o estado de seus vizinhos, gerando padrões dinâmicos, osciladores, estruturas auto-replicantes e comportamentos emergentes. Seu caráter iterativo e massivamente paralelo o torna particularmente adequado para implementação em plataformas de hardware como FPGAs, onde a atualização simultânea de múltiplas células pode ser explorada de maneira eficiente.

Este trabalho tem como objetivo implementar o Jogo da Vida de Conway em uma arquitetura digital utilizando a linguagem Verilog e a plataforma FPGA, com exibição gráfica do estado do sistema em tempo real através de interface VGA. A simulação foi dividida em módulos bem definidos, abrangendo desde o controle de clock até a geração da imagem ampliada em tela, com foco na modularidade, paralelismo e clareza visual.

2 Desenvolvimento Teórico

2.1 O Jogo da Vida de Conway

O Jogo da Vida, proposto por John Horton Conway em 1970, é um autômato celular bidimensional em que cada célula da matriz pode estar em um de dois estados: viva ou morta. A cada nova geração, o estado de cada célula é atualizado simultaneamente com base no estado de seus oito vizinhos imediatos, formando uma vizinhança conhecida como *vizinhança de Moore*. Apesar das regras simples, o sistema pode gerar padrões extremamente complexos, oscilatórios ou até mesmo auto-replicantes.

2.1.1 Regras de Evolução

As regras que definem a evolução do Jogo da Vida são aplicadas a todas as células da matriz de forma simultânea. Elas são:

- **Regra 1:** Qualquer célula viva com menos de dois vizinhos vivos morre por subpopulação.
- **Regra 2:** Qualquer célula viva com dois ou três vizinhos vivos continua viva para a próxima geração.
- **Regra 3:** Qualquer célula viva com mais de três vizinhos vivos morre por superpopulação.
- **Regra 4:** Qualquer célula morta com exatamente três vizinhos vivos torna-se viva por reprodução.

Essas regras são aplicadas a cada ciclo de atualização, gerando novos padrões a partir da configuração atual. O sistema evolui de forma determinística e paralela, sendo ideal para simulações em hardware como FPGAs.

2.1.2 Exemplos de Padrões Clássicos

Diversos padrões emergentes interessantes podem ser observados no Jogo da Vida, mesmo com configurações iniciais simples. Entre os mais conhecidos estão:

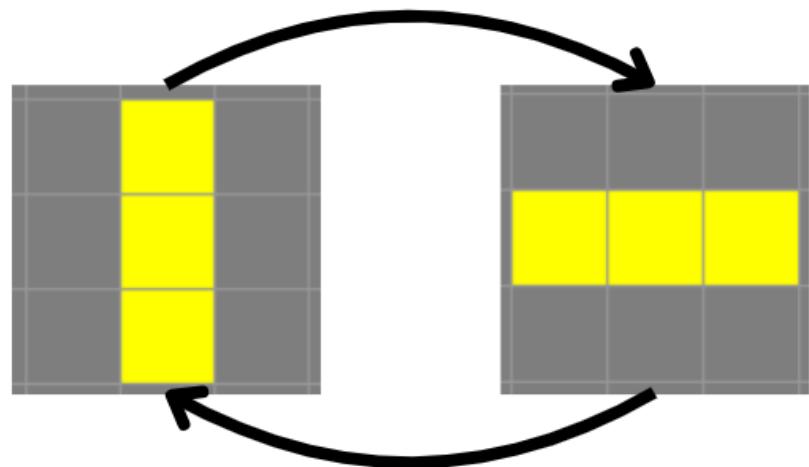


Figura 1: Padrão Blinker

- **Blinker:** Um oscilador com período 2, formado por três células vivas em linha.

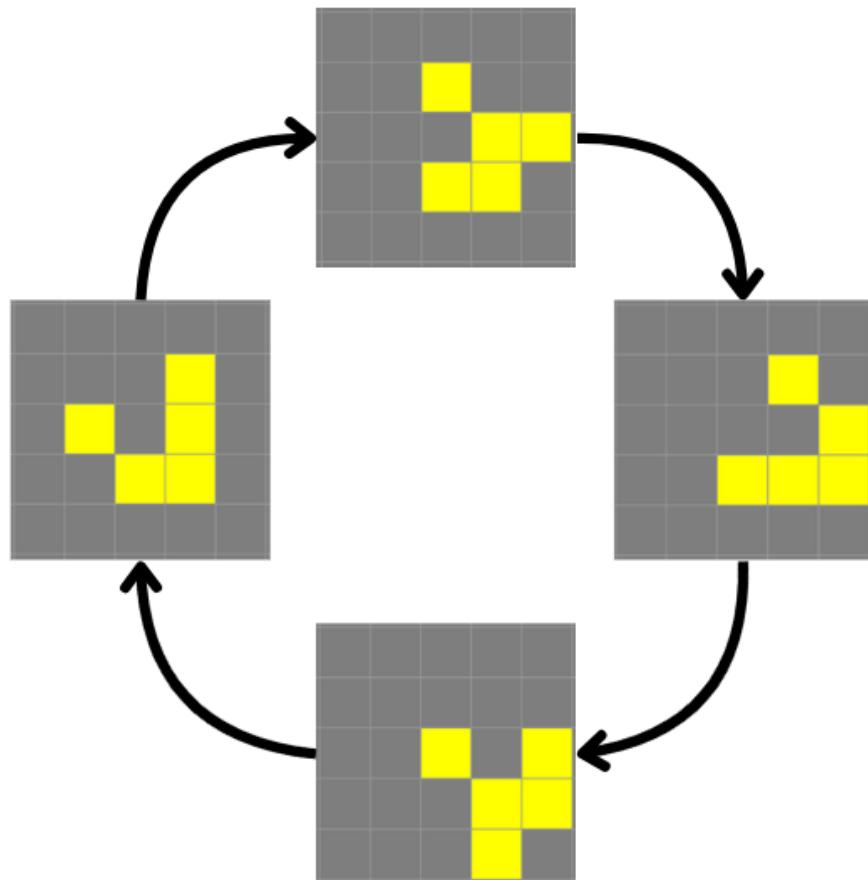


Figura 2: Padrão Glider

- **Glider:** Um padrão que se desloca diagonalmente ao longo da matriz, repetindo-se a cada 4 gerações.

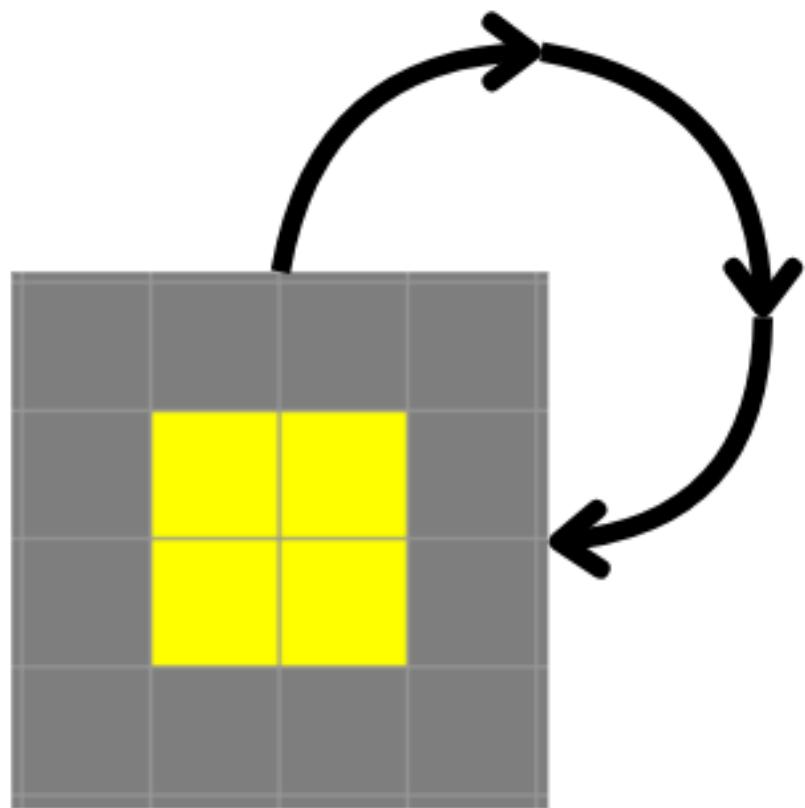


Figura 3: Padrão Block

- **Block:** Um padrão estático de quatro células em forma de quadrado 2x2, que não se altera.

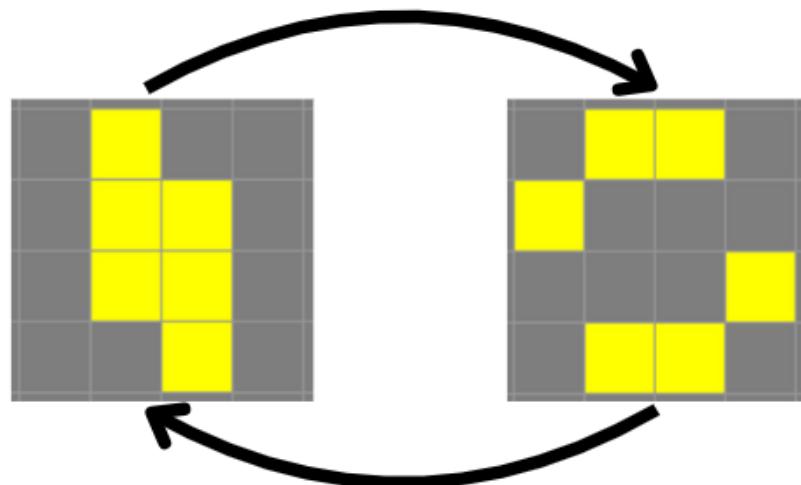


Figura 4: Padrão Toad

- **Toad:** Um oscilador de seis células com período 2.

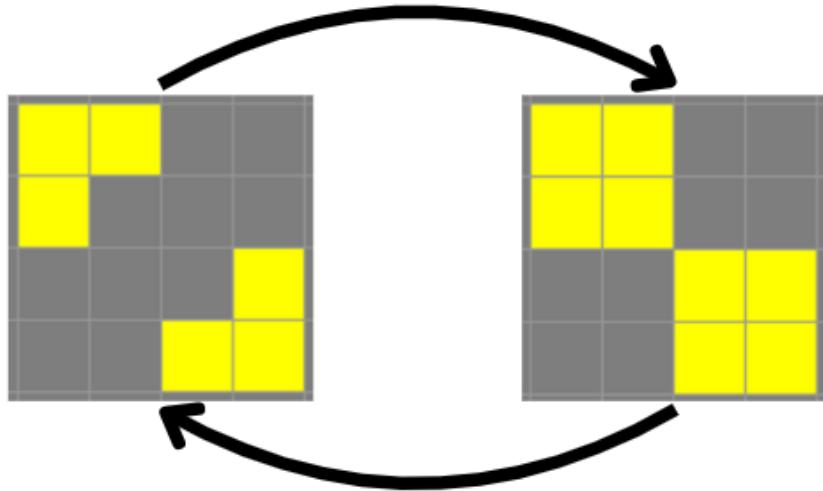


Figura 5: Padrão Beacon

- **Beacon:** Outro oscilador com quatro blocos alternando entre duas configurações.

Esses padrões são úteis tanto para testes da lógica implementada quanto para visualização de comportamentos emergentes no sistema.

2.2 Autômatos Celulares em Arquiteturas Paralelas

Autômatos celulares são modelos computacionais baseados em uma grade discreta de células, cada uma das quais pode assumir um estado finito. A evolução do sistema ocorre de maneira simultânea e em etapas discretas de tempo, segundo regras locais que consideram os estados vizinhos.

Essa natureza local e determinística torna os autômatos celulares especialmente bem adaptados à execução em arquiteturas paralelas. Como a atualização de uma célula depende apenas de um pequeno conjunto de estados ao seu redor, é possível calcular o novo estado de múltiplas células de forma independente e simultânea, utilizando módulos replicáveis ou blocos de hardware embarcado, como os oferecidos por FPGAs.

2.2.1 Vizinhança de Moore

No contexto dos autômatos celulares bidimensionais, a **vizinhança de Moore** é um dos modelos de interação mais comuns. Nela, cada célula considera os oito vizinhos que a cercam nas direções ortogonais e diagonais:

V8	V1	V2
V7	C	V3
V6	V5	V4

Figura 6: Vizinhança de Moore

Onde **C** representa a célula central e **V1** a **V8** são os vizinhos. Essa configuração é utilizada no Jogo da Vida para determinar a quantidade de células vivas ao redor de cada ponto da matriz e, assim, aplicar as regras de transição de estado.

2.2.2 Atualização Síncrona em Sistemas Embarcados

A atualização síncrona refere-se à estratégia de calcular o novo estado de todas as células de maneira simultânea, utilizando um sinal de clock comum. Esse modelo é fundamental em implementações com FPGAs, pois permite que o sistema evolua em etapas discretas e previsíveis, sincronizadas com um clock global.

No presente projeto, a atualização das células foi implementada por meio de uma máquina de estados que percorre toda a matriz binária a cada ciclo de clock. As atualizações são feitas de forma síncrona e coordenada, utilizando operadores lógicos para computar a quantidade de vizinhos vivos e aplicar as regras do Jogo da Vida. Essa abordagem garante consistência na transição entre gerações e facilita a integração com os demais módulos do sistema, como o controlador de exibição VGA.

2.3 Imagem Binária em VGA

A exibição da imagem binária processada foi realizada por meio de um monitor conectado via interface VGA (Video Graphics Array). Este método permite representar visualmente o conteúdo da memória processada em tempo real, o que é essencial para validar o pipeline morfológico e a etapa de detecção de bordas. A seguir, detalham-se os princípios fundamentais do funcionamento de um pixel e as características do cabo VGA.

2.3.1 Funcionamento de um Pixel

Em um monitor VGA, a imagem é formada por uma grade de pixels atualizada sequencialmente linha a linha (varredura raster). Cada pixel corresponde a uma coordenada única (x, y) e é ativado de acordo com os sinais de sincronização horizontal (HSYNC) e vertical (VSYNC), além dos sinais de cor (vermelho, verde e azul).

No caso de uma **imagem binária**, cada pixel possui apenas dois estados possíveis: *ativo (branco)* ou *inativo (preto)*. No sistema implementado neste projeto, essa ativação é controlada por um bit da memória que representa o valor do pixel. Durante a varredura da tela:

- Se o valor do pixel for 1, os três canais RGB recebem sinal alto (cor branca);

- Se o valor for 0, os três canais permanecem baixos (cor preta).

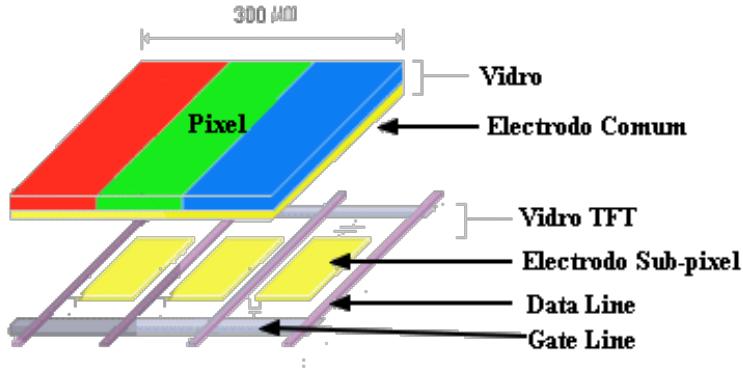


Figura 7: Partes de um Pixel

A lógica responsável por isso compara a posição atual de varredura com o endereço de memória correspondente, determinando a saída dos sinais RGB. Assim, a imagem binária é exibida em tempo real, sincronizada com os sinais VGA.

2.3.2 O Cabo VGA

O cabo VGA é um meio de transmissão analógico composto por **15 pinos**, entre os quais destacam-se os sinais de vídeo (RGB) e os sinais de sincronização horizontal e vertical. Os principais pinos utilizados no projeto foram:

- **H SYNC (pino 13)**: sinal de sincronização horizontal;
- **V SYNC (pino 14)**: sinal de sincronização vertical;
- **R, G, B (pinos 1, 2, 3)**: canais de vídeo analógicos — controlam a cor exibida em cada pixel.

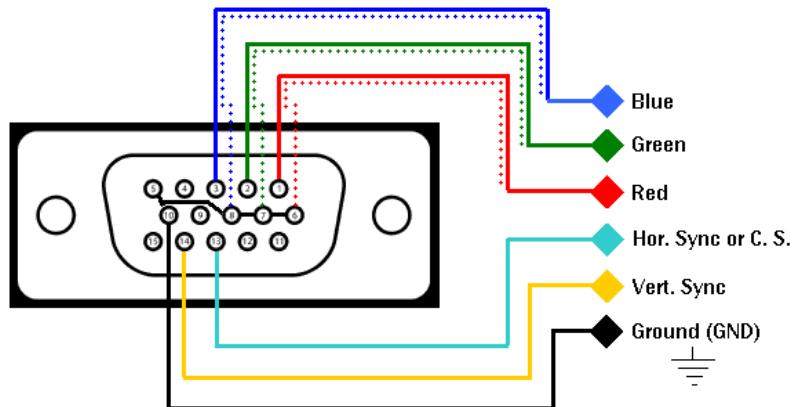


Figura 8: Pinos do Cabo VGA

Na implementação em FPGA, os sinais HSYNC e VSYNC são gerados por um contador de tempo configurado para atender à resolução desejada (por exemplo, 640×480 a 60 Hz). Os sinais de cor são controlados por meio de lógica combinacional, com base no valor do pixel na memória.

Essa estrutura permite exibir a imagem binária diretamente no monitor VGA, sem a necessidade de buffers intermediários ou processamento adicional. É uma abordagem eficiente, determinística e compatível com aplicações embarcadas de baixo custo.

3 Desenvolvimento Prático

A implementação do sistema foi realizada utilizando a plataforma Quartus Prime, com a linguagem de descrição de hardware Verilog. O projeto foi estruturado de forma modular, com o objetivo de garantir organização, reutilização e facilitada verificação de funcionamento. Cada módulo corresponde a uma etapa específica do processamento da imagem ou do controle de exibição no monitor VGA.

3.1 Divisão em Módulos no Quartus

A divisão funcional do projeto permitiu um desenvolvimento incremental e organizado. A seguir, são descritos os principais módulos desenvolvidos.

3.1.1 Módulo VGA

Este módulo é responsável por gerar os sinais de controle VGA, incluindo os pulsos de sincronização horizontal (**H SYNC**) e vertical (**V SYNC**), além da ativação dos sinais RGB. Ele também fornece as coordenadas de varredura (**x, y**), representadas pelas saídas `pixel_row` e `pixel_column`, que determinam a posição atual na tela. Essas coordenadas são utilizadas por outros módulos para buscar o valor correspondente na memória de imagem.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3 use IEEE.STD_LOGIC_ARITH.all;
4 use IEEE.STD_LOGIC_UNSIGNED.all;
5 -- Module Generates Video Sync Signals for Video Monitor Interface
6 -- RGB and Sync outputs tie directly to monitor connector pins
7 ENTITY VGA_SYNC IS
8     PORT( clock_50Mhz, red, green, blue : IN STD_LOGIC;
9             red_out, green_out, blue_out, horiz_sync_out,
10            vert_sync_out, video_on, pixel_clock : OUT STD_LOGIC;
11            pixel_row, pixel_column : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
12 END VGA_SYNC;
13 ARCHITECTURE a OF VGA_SYNC IS
14     SIGNAL horiz_sync, vert_sync, pixel_clock_int : STD_LOGIC;
15     SIGNAL video_on_int, video_on_v, video_on_h : STD_LOGIC;
16     SIGNAL h_count, v_count : STD_LOGIC_VECTOR(9 DOWNTO 0);
17     --
18     -- To select a different screen resolution, clock rate, and refresh rate
19     -- pick a set of new video timing constant values from table at end of code
20     -- section
21     -- enter eight new sync timing constants below and
22     -- adjust PLL frequency output to pixel clock rate from table
23     -- using MegaWizard to edit video_PLL.vhd
24     -- Horizontal Timing Constants
25     CONSTANT H_pixels_across : Natural := 640;
26     CONSTANT H_sync_low : Natural := 664;
27     CONSTANT H_sync_high : Natural := 760;
28     CONSTANT H_end_count : Natural := 800;
29     -- Vertical Timing Constants
30     CONSTANT V_pixels_down : Natural := 480;
31     CONSTANT V_sync_low : Natural := 491;
32     CONSTANT V_sync_high : Natural := 493;
33     CONSTANT V_end_count : Natural := 525;
34     COMPONENT video_PLL

```

```

34      PORT
35      (
36          inclk0      : IN STD_LOGIC  := '0';
37          c0         : OUT STD_LOGIC
38      );
39  end component;
40
41 BEGIN
42
43 -- PLL below is used to generate the pixel clock frequency
44 -- Uses UP 3's 48Mhz USB clock for PLL's input clock
45 video_PLL_inst : video_PLL PORT MAP (
46     inclk0    => Clock_50Mhz,
47     c0        => pixel_clock_int
48 );
49
50 -- video_on is high only when RGB pixel data is being displayed
51 -- used to blank color signals at screen edges during retrace
52 video_on_int <= video_on_H AND video_on_V;
53 -- output pixel clock and video on for external user logic
54 pixel_clock <= pixel_clock_int;
55 video_on <= video_on_int;
56
57 PROCESS
58 BEGIN
59     WAIT UNTIL(pixel_clock_int'EVENT) AND (pixel_clock_int='1');
60
61 --Generate Horizontal and Vertical Timing Signals for Video Signal
62 -- H_count counts pixels (#pixels across + extra time for sync signals)
63 --
64 -- Horiz_sync  -----
65 -- H_count      0                      #pixels           sync low      end
66 --
67     IF (h_count = H_end_count) THEN
68         h_count <= "0000000000";
69     ELSE
70         h_count <= h_count + 1;
71     END IF;
72
73 --Generate Horizontal Sync Signal using H_count
74     IF (h_count <= H_sync_high) AND (h_count >= H_sync_low) THEN
75         horiz_sync <= '0';
76     ELSE
77         horiz_sync <= '1';
78     END IF;
79
80 --V_count counts rows of pixels (#pixel rows down + extra time for V sync
81 -- signal)
82 --
83 -- Vert_sync  -----
84 -- V_count      0                      last pixel row   V sync low
85 -- end
86 --
87     IF (v_count >= V_end_count) AND (h_count >= H_sync_low) THEN
88         v_count <= "0000000000";
89     ELSIF (h_count = H_sync_low) THEN
90         v_count <= v_count + 1;
91     END IF;
92
93 -- Generate Vertical Sync Signal using V_count

```

```

92      IF (v_count <= V_sync_high) AND (v_count >= V_sync_low) THEN
93          vert_sync <= '0';
94      ELSE
95          vert_sync <= '1';
96      END IF;
97
98      -- Generate Video on Screen Signals for Pixel Data
99      -- Video on = 1 indicates pixel are being displayed
100     -- Video on = 0 retrace - user logic can update pixel
101     -- memory without needing to read memory for display
102     IF (h_count < H_pixels_across) THEN
103         video_on_h <= '1';
104         pixel_column <= h_count;
105     ELSE
106         video_on_h <= '0';
107     END IF;
108
109    IF (v_count <= V_pixels_down) THEN
110        video_on_v <= '1';
111        pixel_row <= v_count;
112    ELSE
113        video_on_v <= '0';
114    END IF;
115
116    -- Put all video signals through DFFs to eliminate any small timing delays
117    -- that cause a blurry image
118    horiz_sync_out <= horiz_sync;
119    vert_sync_out <= vert_sync;
120
121    -- Also turn off RGB color signals at edge of screen during vertical and
122    -- horizontal retrace
123    red_out <= red AND video_on_int;
124    green_out <= green AND video_on_int;
125    blue_out <= blue AND video_on_int;
126
127 END PROCESS;
128
129 END a;

```

Listing 1: Código do Módulo VGA Fornecido

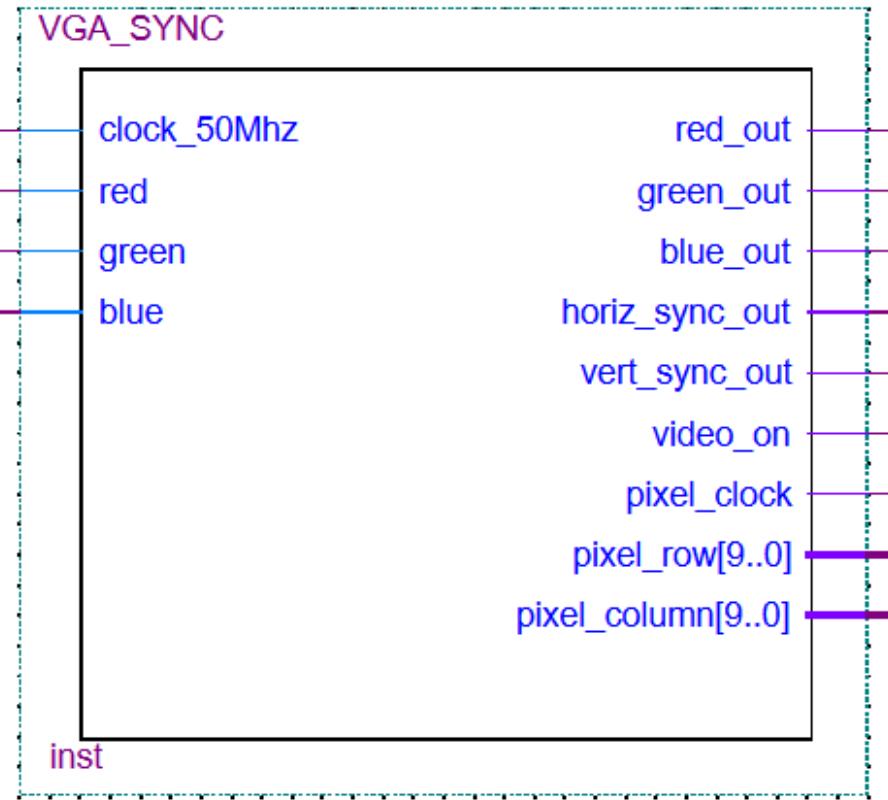


Figura 9: Módulo VGA

Além dos sinais básicos de sincronização e cor, o módulo também fornece:

- **pixel_clock**: sinal utilizado para marcar o instante exato de atualização do pixel, garantindo que as leituras de memória e mudanças nos sinais RGB ocorram de forma sincronizada com a taxa de varredura da tela;
- **video_on**: sinal lógico que indica se o pixel atual se encontra dentro da área visível da tela. Este sinal é essencial para garantir que apenas os dados válidos sejam enviados ao monitor, evitando artefatos fora da área ativa de exibição.

Os sinais de cor (**red**, **green**, **blue**) são ativados apenas quando **video_on** está em nível alto, assegurando uma imagem estável e consistente. O módulo VGA atua, portanto, como a base do sistema de exibição em tempo real do projeto.

3.1.2 Módulo de Janela

O módulo de janela tem como função verificar se o pixel atual está localizado dentro da área útil da imagem binária, a qual ocupa apenas uma região da tela VGA. Como a imagem a ser processada pode eventualmente ser configurada para possuir dimensões menores que as do monitor, é necessário restringir o processamento aos pixels válidos.

```

1 // módulo para definir se
2 // pixel pertence a janela
3 module janela(
4   // entradas e saídas
5   input [9:0]a,
6   input [9:0]b,
7   output s);
8

```

```

9 // se pertencer a janela
10 assign s = ((a < 640) & (b < 480));
11
12 endmodule

```

Listing 2: Código Verilog Módulo de Janela

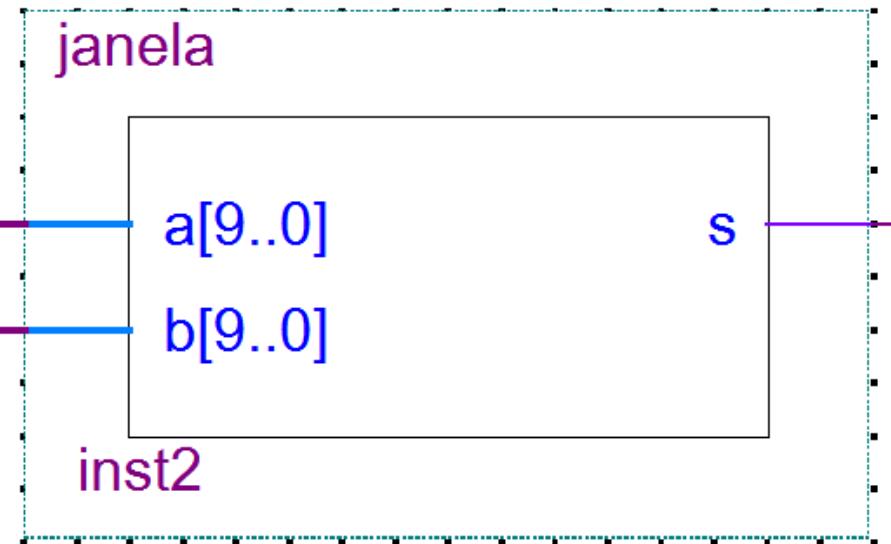


Figura 10: Módulo Janela

Para isso, o módulo recebe como entrada as coordenadas `pixel_column` e `pixel_row`, fornecidas pelo módulo VGA, e compara esses valores com os limites da janela predefinida de visualização. Quando o pixel se encontra dentro desses limites, o módulo emite um sinal de ativação indicando que o dado correspondente pode ser processado e exibido.

Essa verificação é essencial para garantir que os módulos de leitura de memória e exibição no VGA operem apenas sobre a região útil da imagem, evitando leituras inválidas e artefatos na borda da tela.

3.1.3 Módulo Redutor de Clock

Esse módulo tem como objetivo gerar sinais de clock com frequências menores a partir do clock principal da placa. Isso é necessário para controlar a velocidade de atualização do sistema, especialmente em contextos visuais, como a simulação do Jogo da Vida, onde uma atualização muito rápida pode impedir que o usuário acompanhe as mudanças na tela.

```

1 // módulo para reduzir a frequência de clock
2 module clk_1_seg(
3     input clk,          // sinal de clock da placa
4     output clk_lento,   // saída de clock mais lento
5     output clk_rapido); // saída de clock mais rápido
6
7 // contador
8 reg [25:0] contador;
9
10 // inicialmente contador é zero
11 initial begin
12     contador = 0;
13 end
14
15 // a cada clock da placa

```

```

16    always@(posedge clk) begin
17        //incrementa o contador em 1 unidade
18        contador = contador + 1;
19    end
20
21    // clock lento é bit 25 do contador
22    assign clk_lento = contador[25];
23    //clock rápido é o bit 24 do contador
24    assign clk_rapido = contador[24];
25
26 endmodule

```

Listing 3: Código Verilog Módulo Redutor de Clock

O módulo implementa um *divisor de clock* baseado em um contador binário de 26 bits. A cada borda de subida do clock original da placa (tipicamente 50 MHz), o contador é incrementado em uma unidade. Com isso, os bits mais significativos do contador passam a oscilar com uma frequência significativamente menor do que o clock original.

A saída `clk_rapido` corresponde ao bit de ordem 24 do contador (`contador[24]`), que oscila com uma frequência cerca de 2^{24} vezes menor que o clock de entrada. Já a saída `clk_lento` corresponde ao bit de ordem 25 (`contador[25]`), e portanto tem uma frequência ainda mais reduzida, metade da frequência gerada pelo bit 24.

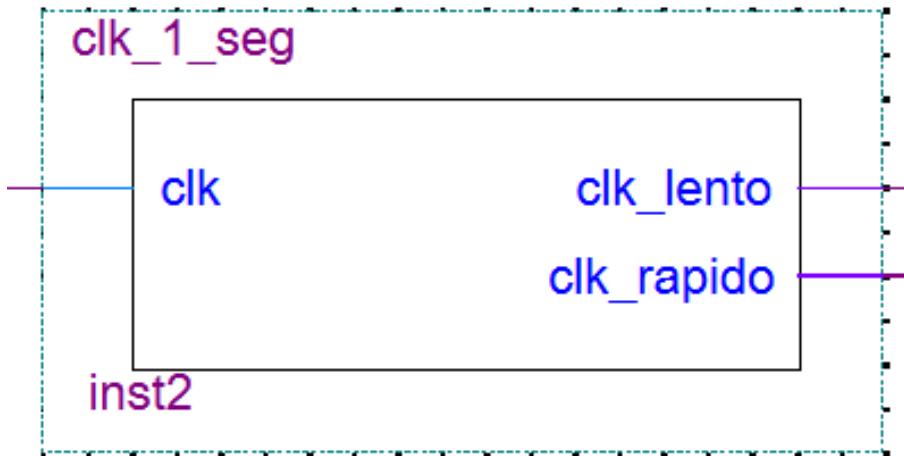


Figura 11: Módulo Redutor de Clock

Ambos os sinais de clock são disponibilizados para o restante do sistema, permitindo selecionar entre uma atualização mais rápida (útil para testes) ou mais lenta (útil para visualização passo a passo). A escolha entre os dois é feita através de um multiplexador descrito em outra seção.

Essa técnica é comum em sistemas digitais que requerem diferentes escalas de tempo e permite ajustar dinamicamente a velocidade da simulação sem alterar o clock global da FPGA.

3.1.4 Multiplexador de Clock

O módulo `muxecp` é um multiplexador 2:1, cuja função é selecionar qual dos dois sinais de entrada será encaminhado para a saída, com base em um sinal de controle.

```

1 // módulo multiplexador
2 module muxecp(
3     // entradas e saídas
4     input a,
5     input b,
6     input sel,

```

```

7   output s);
8
9   // true implica s=a
10  // false implica s=b
11  assign s = sel ? a : b;
12
13 endmodule

```

Listing 4: Código Verilog Módulo Multiplexador

Ele possui três entradas:

- **a**: primeiro sinal de entrada (neste projeto, o clock lento);
- **b**: segundo sinal de entrada (o clock rápido);
- **sel**: sinal seletor. Quando **sel** = 1, a saída **s** recebe o valor de **a**; caso contrário, recebe o valor de **b**.

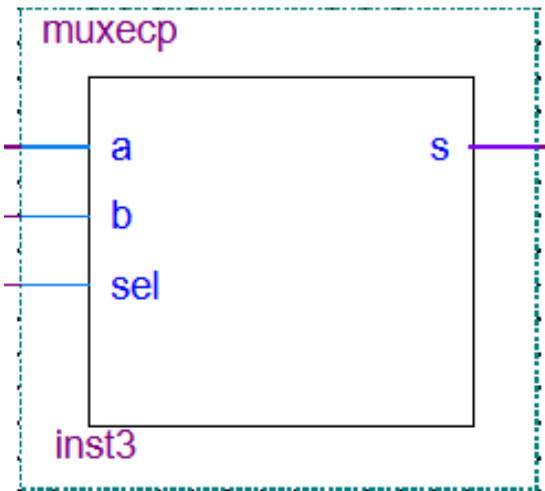


Figura 12: Módulo Multiplexador

No contexto deste projeto, o multiplexador é utilizado exclusivamente para escolher qual frequência de clock será utilizada para controlar a atualização da simulação. Essa escolha é feita dinamicamente pelo usuário por meio de um switch físico da placa FPGA.

3.1.5 Máquina de Estados

O módulo **state_machine** implementa a lógica do Jogo da Vida de Conway, atualizando o estado de uma matriz de pixels com base nas regras do autômato celular. Ele opera com um sinal de clock (**clk**), um sinal de reset (**rst**) e manipula uma matriz de 66 colunas por 50 linhas, totalizando 3300 pixels, representados como um vetor de 3299 bits.

```

1 //módulo da máquina de estados
2 module state_machine (
3   input wire clk,           // sinal de clock
4   input wire rst,           // sinal de reset
5   input wire [3299:0] data_in,    // entrada: estado atual
6   output reg [3299:0] data_out     // saída: próxima geração
7 );
8   // auxiliares
9   integer x, y, idx;
10  integer vizinhos;

```

```

11 // apenas para carregar arquivo inicial e de reset
12 reg [0:0] temp_mem [0:3299];
13
14 // inicialmente
15 initial begin
16     // faz a leitura do arquivo contendo o estado inicial
17     $readmemb("game_of_life_66x50.txt", temp_mem);
18     // copia esses dados para a saída
19     for (idx = 0; idx < 3300; idx = idx + 1)
20         data_out[idx] = temp_mem[idx];
21     end
22
23 // a cada sinal de clock
24 always @(posedge clk) begin
25     // se o sinal de reset estiver ativado
26     if (rst) begin
27         // copia os dados do estado inicial na saída
28         for (idx = 0; idx < 3300; idx = idx + 1)
29             data_out[idx] <= temp_mem[idx];
30
31     end
32     // não há sinal de reset
33     else begin
34         // percorre cada bit do sinal
35         for (y = 1; y < 49; y = y + 1) begin
36             for (x = 1; x < 65; x = x + 1) begin
37                 // posição do bit no vetor
38                 idx = y * 66 + x;
39
40                 // identifica quantos pixels ativos há na vizinhança
41                 vizinhos =
42                     data_in[(y-1)*66 + (x-1)] +
43                     data_in[(y-1)*66 + x] +
44                     data_in[(y-1)*66 + (x+1)] +
45                     data_in[y*66 + (x-1)] +
46                     data_in[y*66 + (x+1)] +
47                     data_in[(y+1)*66 + (x-1)] +
48                     data_in[(y+1)*66 + x] +
49                     data_in[(y+1)*66 + (x+1)];
50
51                 // se o pixel atual está ativo
52                 if (data_in[idx] == 1'b1)
53                     // se há 2 ou 3 vizinhos, permanece ativo, do contrário
54                     // desliga
55                     data_out[idx] <= (vizinhos == 2 || vizinhos == 3) ?
56                         1'b1 : 1'b0;
57                 // se o pixel atual está desligado
58             else
59                 // se há 3 vizinhos é ativado, do contrário permanece
60                 // desligado
61                     data_out[idx] <= (vizinhos == 3) ? 1'b1 : 1'b0;
62             end
63         end
64
65         // zera as bordas
66         for (y = 0; y < 50; y = y + 1)
67             for (x = 0; x < 66; x = x + 1)
68                 if (y == 0 || y == 49 || x == 0 || x == 65)
69                     data_out[y * 66 + x] <= 1'b0;
70     end

```

```

69     end
70
71 endmodule

```

Listing 5: Código Verilog Máquina de Estados

Na inicialização, o módulo carrega o estado inicial a partir de um arquivo chamado `game_of_life_66x50.txt`, que contém a configuração inicial das células (vivas ou mortas). Essa leitura ocorre no bloco `initial` e os valores são armazenados em um vetor auxiliar chamado `temp_mem`. Esses dados são então copiados para a saída `data_out`.

Durante o funcionamento normal, a cada borda de subida do clock, o módulo verifica o sinal de reset. Esse reset é síncrono, ou seja, só é avaliado no momento da borda de subida do clock. Caso o reset esteja ativado, o sistema é restaurado para o estado original carregado do arquivo. Se não houver reset, a lógica do Jogo da Vida é aplicada:

- Para cada célula que não pertence às bordas da matriz (ou seja, para x de 1 a 64 e y de 1 a 48), é realizada a contagem de vizinhos vivos considerando os 8 vizinhos adjacentes.
- Se a célula atual estiver viva, ela permanece viva apenas se tiver exatamente 2 ou 3 vizinhos vivos; do contrário, morre.
- Se a célula estiver morta, ela nasce apenas se tiver exatamente 3 vizinhos vivos.

Ao final de cada ciclo de atualização, as bordas da matriz são forçadas a zero, garantindo que elas permaneçam inertes e evitando leituras inválidas fora do vetor.

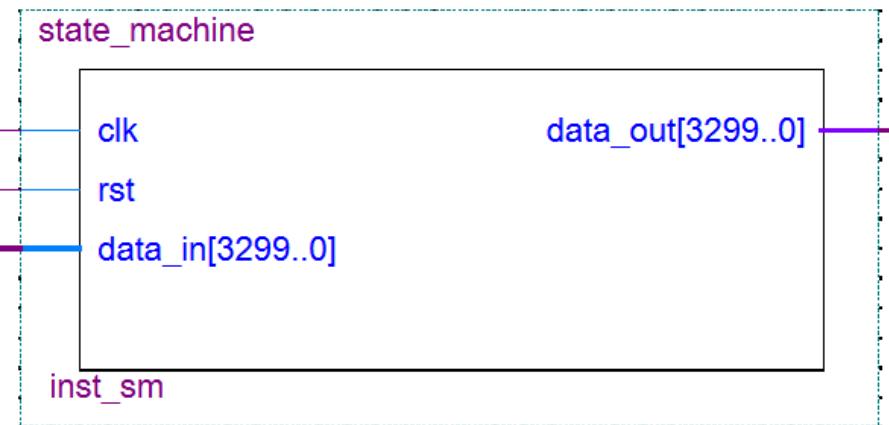


Figura 13: Módulo Máquina de Estados

A saída `data_out` é retroalimentada como entrada `data_in` para a próxima geração, permitindo a evolução contínua da simulação a cada pulso de clock. Esse módulo representa o núcleo funcional do projeto, sendo responsável pela evolução do autômato celular em tempo real.

3.1.6 Módulo Ampliador de Pixels

O módulo é responsável por adaptar a imagem lógica do Jogo da Vida, originalmente com resolução de 64×48 pixels, à resolução da tela VGA (640×480 pixels). Para isso, ele amplia visualmente cada célula lógica, fazendo com que um único bit do vetor `pixels_in` ocupe uma área de 10×10 pixels na tela.

```

1 // módulo ampliador de pixels
2 module big_pixel (
3   input [3299:0] pixels_in,  // imagem original: 66 x 50 = 3300 bits
4   input [9:0] hcount,        // contador horizontal do VGA (0 a 639)
5   input [9:0] vcount,        // contador vertical do VGA (0 a 479)

```

```

6   output out           // pixel de saída sincronizado com VGA
7 );
8
9   // converte posição atual do VGA para coordenadas de pixel original
10  wire [6:0] x_pixel = (hcount / 10) + 1; // de 1 até 64
11  wire [6:0] y_pixel = (vcount / 10) + 1; // de 1 até 48
12
13  // índice linear no vetor pixels_in (66 colunas por linha)
14  wire [12:0] idx = y_pixel * 66 + x_pixel;
15
16  // proteção contra estouro fora da imagem (bordas visuais, etc)
17  assign out = (x_pixel < 65 && y_pixel < 49) ? pixels_in[idx] : 1'b0;
18
19 endmodule

```

Listing 6: Código Verilog Módulo Ampliador de Pixels

As entradas `hcount` e `vcount` representam as coordenadas horizontais e verticais do ponto atual sendo processado pela controladora VGA. A lógica do módulo converte essas coordenadas para as coordenadas da imagem lógica por meio de uma divisão inteira por 10. Em seguida, soma-se 1 a cada coordenada para compensar a borda da imagem, que é ignorada no processamento principal.

A posição lógica identificada é convertida para um índice linear no vetor `pixels_in`, considerando que a imagem possui 66 colunas por linha. O valor do pixel correspondente é então atribuído à saída `out`, desde que a posição esteja dentro dos limites válidos da imagem.

`pixels_in` representa o estado atual da simulação, e esse módulo atua apenas como um multiplicador para exibição, sem modificar o conteúdo lógico. A proteção contra estouro (bordas visuais da tela) é feita verificando se as coordenadas lógicas `x_pixel` e `y_pixel` estão dentro dos limites máximos de 64 e 48, respectivamente.

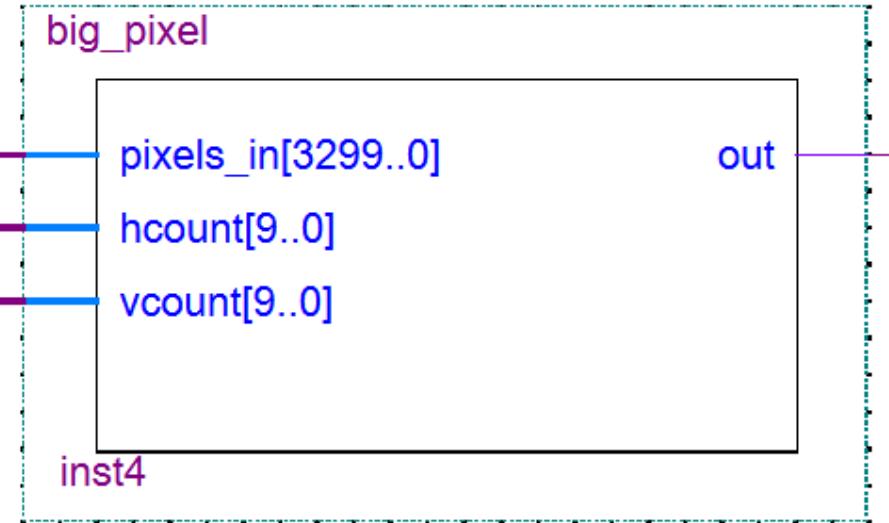


Figura 14: Módulo Ampliador de Pixel

Essa ampliação permite que cada célula seja exibida com clareza na tela VGA, mesmo com uma resolução lógica reduzida, facilitando a observação da evolução do autômato celular.

3.2 Funcionamento Geral

Nesta seção vamos discutir a forma como os módulos interagem durante o processo de filtragem da imagem.

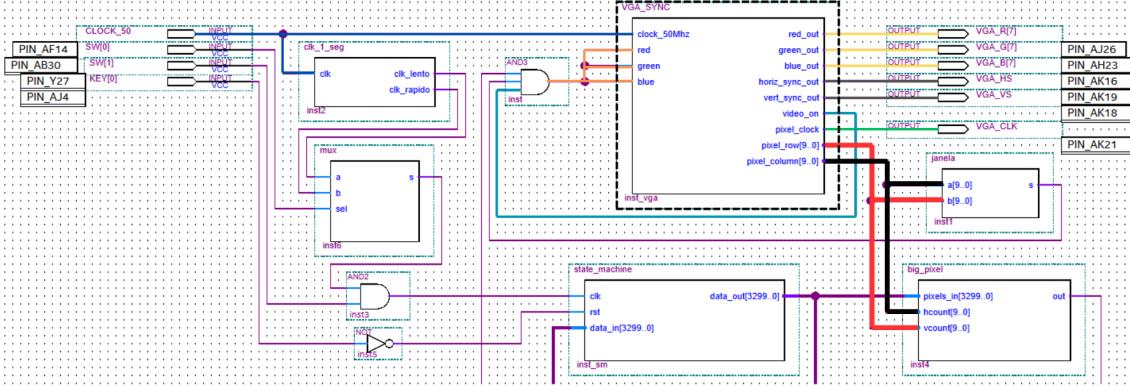


Figura 15: Corte do Diagrama de Blocos

O módulo VGA recebe o sinal de clock de 50 MHz diretamente da placa (fio azul escuro). Os sinais *red*, *green* e *blue* vêm de uma porta AND (fio laranja) e são interligados, justamente porque queremos uma imagem binária, logo, se o sinal recebido é 0, as três cores estão desligadas e o pixel fica preto, e em caso de sinal 1, todas as cores estão ligadas e o resultado é branco.

Na saída, os fios amarelos carregam as informações de cor do pixel e vão diretamente para a placa, juntamente com as informações dos fios cinzas, que são responsáveis pela sincronização horizontal e vertical.

Os sinais vermelho e preto são responsáveis por informar a linha e coluna à qual o pixel sendo processado atualmente pertence. Esses sinais são enviados ao módulo ampliador de pixel (que irá verificar qual deve ser o valor do pixel na tela com base nisso), e também para o módulo janela (que irá identificar se o pixel em questão pertence à janela da nossa imagem).

O sinal verde é utilizado para sincronização dos pixels, sendo enviado para a placa.

O sinal azul claro, utilizado para informar que o vídeo está ativo, vai para a porta AND responsável por controlar se o pixel atual está ativo ou não.

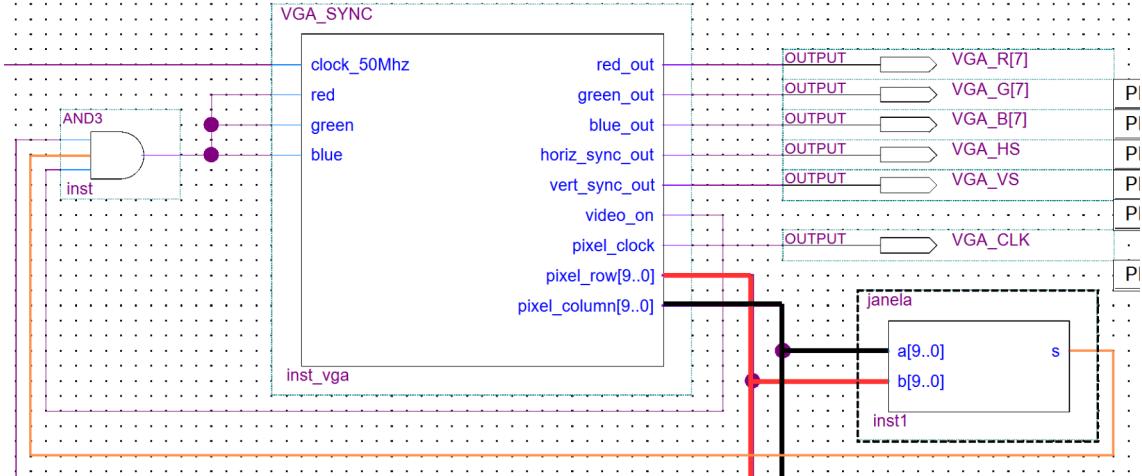


Figura 16: Corte do Diagrama de Blocos

O módulo janela, como já mencionado, recebe a informação de linha e coluna do pixel sendo processado atualmente. Ele verifica se o sinal pertence à janela, e envia a informação (fio laranja) para a porta AND mencionada no parágrafo anterior.

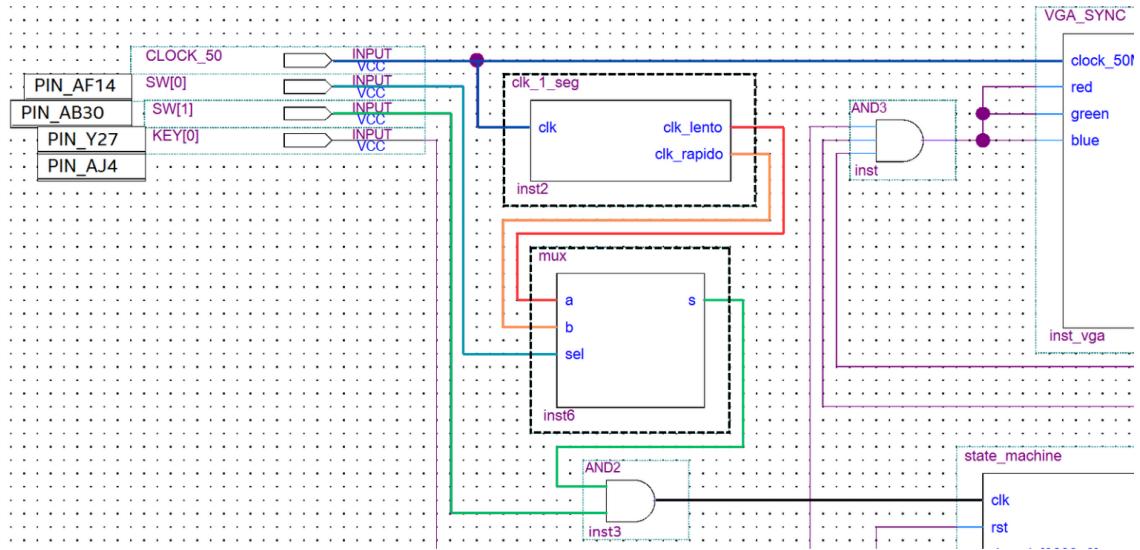


Figura 17: Corte do Diagrama de Blocos

O módulo redutor de clock coleta o sinal de clock da placa (fio azul escuro) e reduz o mesmo para duas frequências menores. Em seguida, ele disponibiliza os dois novos sinais (fios vermelho e laranja) para o multiplexador, que vai selecionar qual dos dois vai operar a máquina de estados com base no sinal de seleção (fio azul claro). Uma porta AND recebe o sinal de clock do multiplexador e o sinal de um switch da placa (fios verdes). Dessa forma, o switch atua como uma forma de pausar a execução da máquina de estados quando necessário, seja para alguma observação mais detalhada ou algo do gênero. O resultado dessa porta AND (fio preto) é enviado para a entrada de clock da máquina de estados.

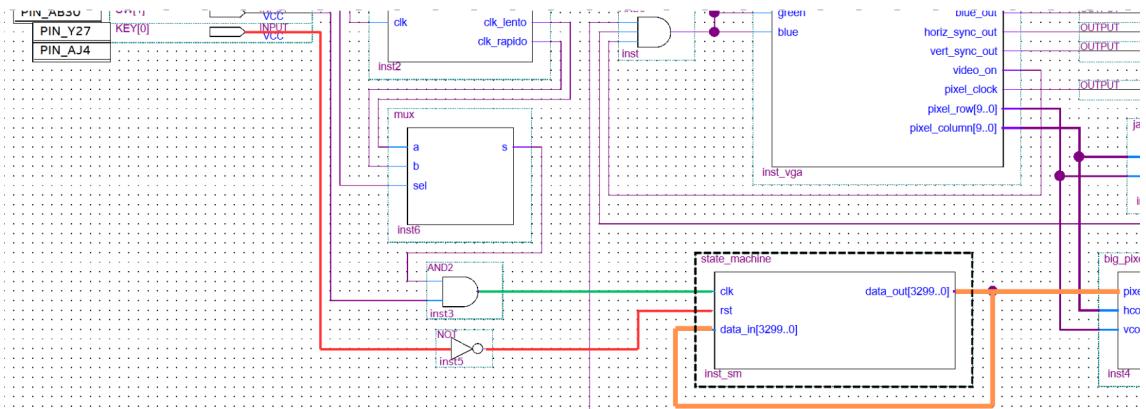


Figura 18: Corte do Diagrama de Blocos

A máquina de estados recebe o sinal de clock processado (com frequência reduzida, selecionado pelo multiplexador e liberado ou pausado pelo switch), o sinal reset de um botão da placa e o sinal contendo o estado atual dos pixels. A saída contendo o novo estado dos pixels vai para o módulo ampliador de pixels e também retroalimenta a máquina para que ela processe o novo estado com base no estado atual.

Note que o sinal de reset é antes processado por uma porta NOT. Isso ocorre por que o botão da placa é *active low*, ou seja, quando não está pressionado ele tem sinal 1, e quando pressionado tem sinal 0, portanto precisamos fazer essa pequena correção. Todavia, isso não tem qualquer relação com o funcionamento geral do circuito nem com a lógica do projeto, é apenas um detalhe da placa.

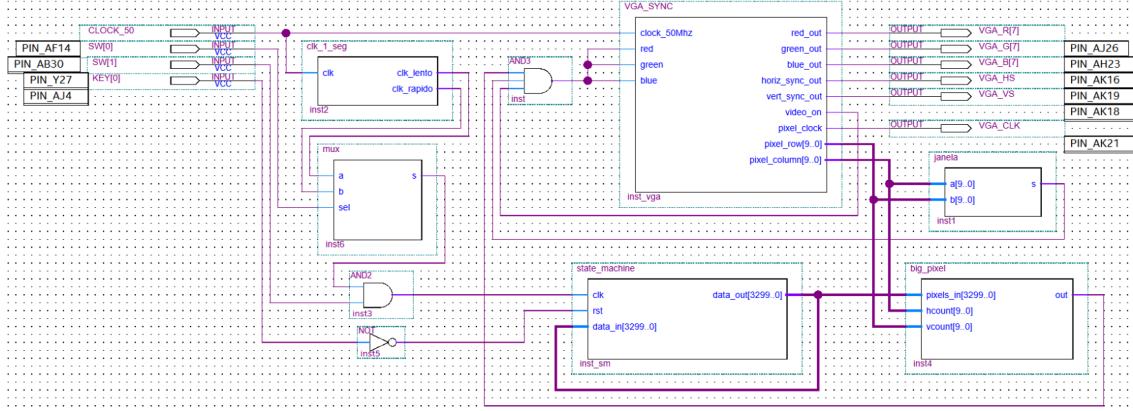


Figura 20: Diagrama de Blocos

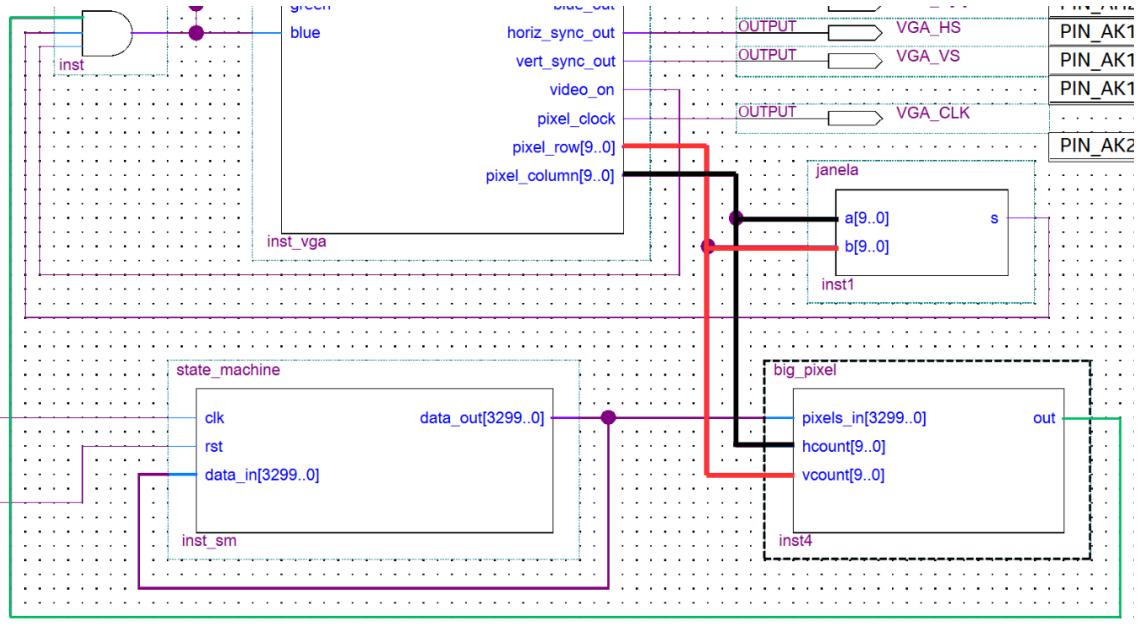


Figura 19: Corte do Diagrama de Blocos

O módulo ampliador de pixels recebe a informação da linha e coluna do pixel que está sendo processado atualmente e, com base nessas informações, identifica qual deve ser o valor do mesmo para representar corretamente a imagem na tela. A informação de saída (pixel ativo ou não) vai para a porta AND já mencionada anteriormente.

Observando o diagrama completo, e com as considerações feitas anteriormente, podemos notar que a porta AND, que decide se o pixel será branco ou preto, depende do sinal do módulo de janela, do sinal de vídeo ativo e do sinal do módulo ampliador de pixel, o que faz todo sentido.

4 Discussão dos Resultados

4.1 Funcionamento Geral do Sistema

Após a implementação dos módulos descritos na seção anterior, o sistema mostrou-se funcional, apresentando corretamente a evolução do Jogo da Vida em tempo real. A comunicação entre os módulos ocorreu de maneira estável, e os ciclos de atualização foram realizados de forma síncrona com o clock selecionado. A lógica de vizinhança foi corretamente aplicada a cada geração, e os

padrões oscilatórios e móveis foram visualmente reconhecidos durante a execução.



Figura 21: Jogo da Vida Funcionando: Etapa 1



Figura 22: Jogo da Vida Funcionando: Etapa 2



Figura 23: Jogo da Vida Funcionando: Etapa 3



Figura 24: Jogo da Vida Funcionando: Etapa 4

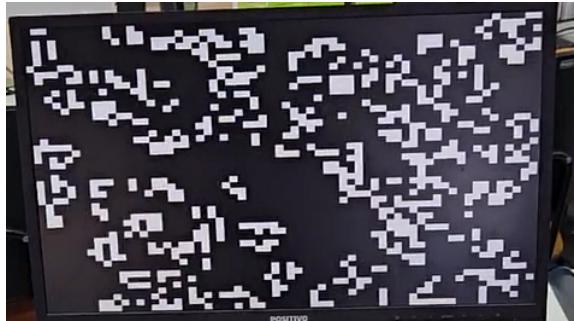


Figura 25: Jogo da Vida Funcionando: Etapa 5

O botão de reset funcionou como esperado, reiniciando a simulação a qualquer instante, juntamente com o switch utilizado para pausar a máquina de estados. O sistema foi capaz de manter a estabilidade do VGA e da imagem durante as atualizações do estado.

4.2 Eficiência Visual da Imagem Ampliada

A utilização do módulo `big_pixel` foi fundamental para garantir a legibilidade dos padrões na tela. A estratégia de replicar cada pixel da matriz original como um bloco 10×10 permitiu a observação clara dos comportamentos ao longo do tempo, sem prejudicar o desempenho do sistema.

Com essa abordagem, mesmo estruturas pequenas como o *Blinker* ou o *Glider* tornaram-se facilmente visíveis, o que contribuiu para uma melhor interpretação dos padrões e da dinâmica da simulação. A ampliação não introduziu artefatos visuais e foi bem integrada ao sistema de exibição.

4.3 Interatividade com o Usuário

A adição de controles físicos no projeto — como o botão de reinício e o switch de pausa — agregou interatividade ao sistema. O botão foi conectado a um sinal de reset que, quando ativado, recarrega o estado inicial da simulação. Por padrão da placa, esse botão opera em lógica inversa, necessitando de uma inversão de sinal (`NOT`) para funcionar corretamente.

Já o switch de pausa foi conectado via uma porta `AND` ao sinal de `clock`, permitindo que o usuário congelasse o sistema ao desejar observar um padrão com mais atenção. Essa pausa não interfere na exibição da imagem, pois o VGA continua operando normalmente — apenas o avanço das gerações é interrompido.

A presença desses mecanismos tornou a simulação mais dinâmica e controlável, evidenciando a utilidade de se projetar interfaces simples, porém eficazes, para o usuário final.

5 Conclusões

A implementação do Jogo da Vida de Conway em hardware, utilizando linguagens de descrição de hardware como Verilog e dispositivos FPGA, permitiu explorar de forma prática conceitos fundamentais de arquiteturas paralelas e sistemas embarcados. O projeto demonstrou como autômatos celulares podem ser eficientemente implementados com atualizações síncronas, aproveitando a natureza paralela das FPGAs para realizar simulações em tempo real.

Durante o desenvolvimento, foi possível aplicar técnicas de divisão modular, projetar circuitos com controle de clock, implementar lógica de atualização baseada em vizinhança e integrar a saída gráfica ao monitor por meio do protocolo VGA. A ampliação dos pixels contribuiu significativamente para a visualização dos padrões gerados, enquanto os mecanismos de controle (pausa e reinício) tornaram a simulação interativa e acessível ao usuário.

O sistema final atendeu aos objetivos propostos, funcionando de maneira estável e eficiente. Como aprimoramento futuro, seria possível permitir a inserção dinâmica de padrões durante a execução, armazenar múltiplos estados iniciais na memória ou mesmo implementar regras variantes para criar novas versões do autômato.

Em suma, este projeto consolidou o entendimento de diversos tópicos abordados na disciplina e evidenciou o potencial do uso de FPGAs no desenvolvimento de aplicações visuais, interativas e computacionalmente paralelas.

Referências

- [1] MARTIN, Edwin *O Jogo da Vida de Conway*. Disponível em: <https://playgameoflife.com/>. Acesso em: junho de 2025.
- [2] THE MEANING OF LIFE. *Stephen Hawking's Grand Design*. EUA: Discovery, 2012. Episódio 1, temporada 1, 44 minutos. Direção: Matthew Huntley.
- [3] NUMBERPHILE. *Inventing Game of Life (John Conway - Numberphile)*. [S.l.]: YouTube, 2014. Disponível em: <https://www.youtube.com/watch?v=R9Plq-D1gEk>. Acesso em: junho de 2025.
- [4] ROBERTS, Siobhan. *John Horton Conway: the world's most charismatic mathematician*. The Guardian, 2015. Disponível em: <https://www.theguardian.com/science/2015/jul/23/john-horton-conway-the-most-charismatic-mathematician-in-the-world>. Acesso em: junho de 2025.
- [5] COOK, Mariana. *John Conway Solved Mathematical Problems With His Bare Hands*. Quanta Magazine, 2020. Disponível em: <https://www.quantamagazine.org/john-conway-solved-mathematical-problems-with-his-bare-hands-20200420/>. Acesso em: junho de 2025.
- [6] AWATI, Rahul. *Cellular Automaton (CA)*. TechTarget, 2021. Disponível em: <https://www.techtarget.com/searchenterprisedesktop/definition/cellular-automaton>. Acesso em: junho de 2025.