



ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Departamento de Engenharia de Computação e Sistemas Digitais

PCS3635 – LABORATÓRIO DIGITAL I

PROJETO AEX – TEAtris

Relato da Bancada B1 – Turma 1 – Prof. Edson Toshimi Midorikawa

Data de Emissão: 02 de abril de 2025.

Nome: Claudio Lucio Cunha da Silva	Número USP: 14565003
Nome: Daniel Henrique Braga da Silva	Número USP: 14565431
Nome: Wesley Oliveira Cunha	Número USP: 14612367

1. DESCRIÇÃO DO JOGO “TEAtris”

TEAtris é um jogo inspirado no Tetris, projetado para rodar em uma FPGA usando duas matrizes de LEDs 8x8. A princípio, o jogo usa peças fixas de três ou dois blocos, que não giram. O jogador controla o jogo por meio de quatro botões, cada um correspondente a um par de colunas, escolhendo onde encaixar a peça exibida na matriz superior. Ao iniciar o jogo, o usuário vê um “mapa” (cenário

previamente preenchido) com um espaço reservado para encaixar a peça na matriz inferior. Se o usuário encaixar a peça no local correto (esperado), o mapa será atualizado para uma nova configuração, em que o jogador terá de encaixar outra peça em um local esperado diferente. Caso o jogador não coloque a peça no local certo, o mapa também será atualizado para uma nova configuração e será contabilizado um erro.

A relação do TEAtris com o acompanhamento do TEA está na mecânica de contagem de erros e na análise da capacidade do usuário de percepção espacial e resolução de padrões visuais. Além disso, a resposta do jogador a um erro - como a insistência em determinadas escolhas ou a adaptação de estratégias - pode fornecer *insights* sobre impulsividade, planejamento e aprendizado.

Dessa forma, o TEAtris pode ser utilizado como uma ferramenta lúdica para análise cognitiva e motora, ajudando no acompanhamento de indivíduos com TEA.

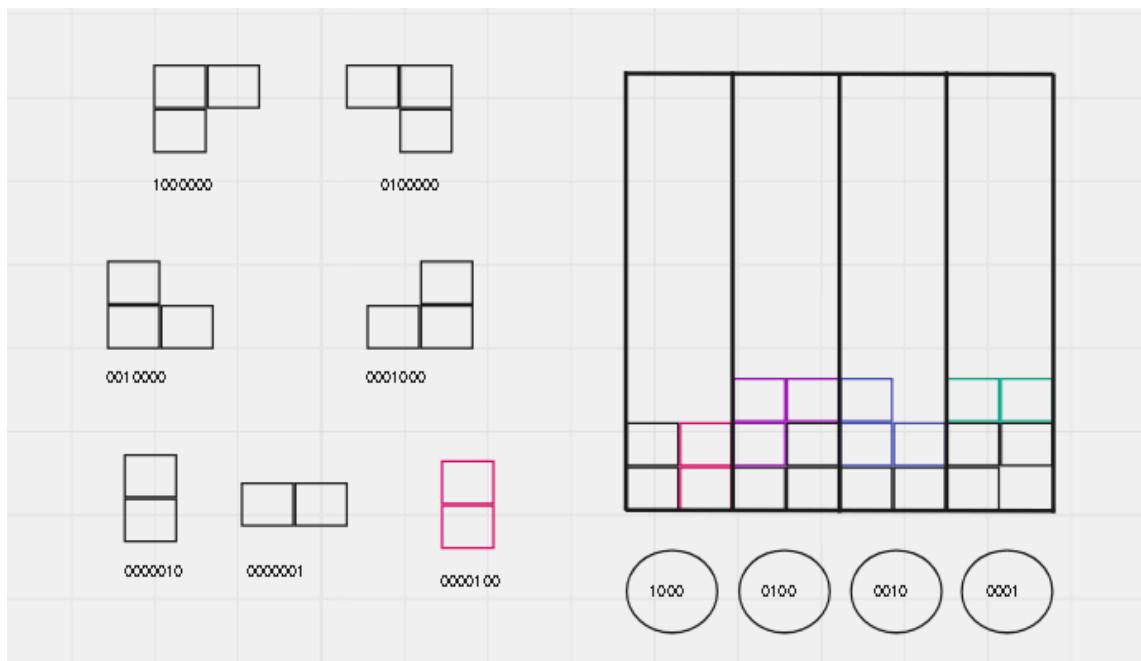


Figura 1: Conceito inicial do jogo

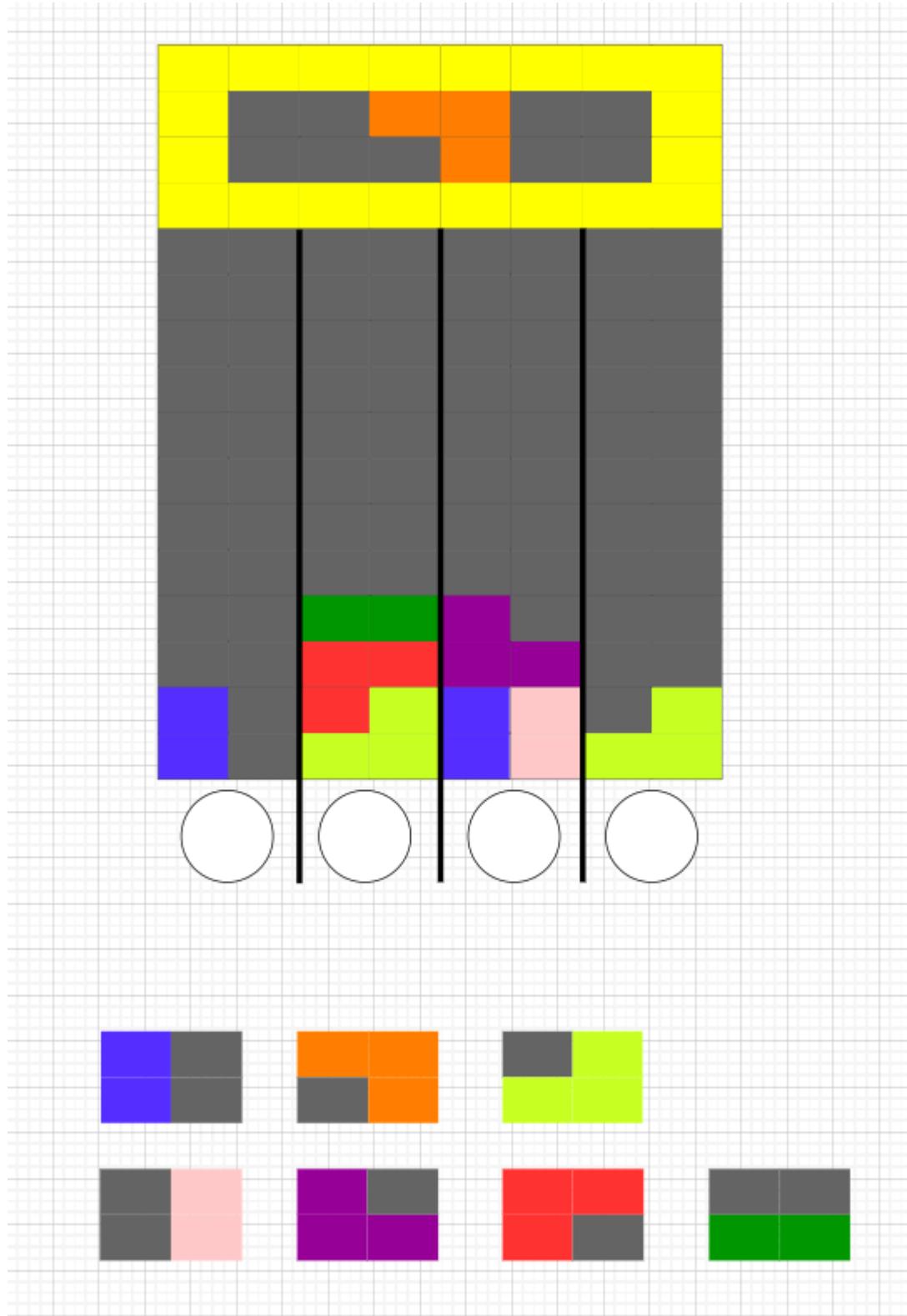


Figura 2: Visão do jogo, com possibilidades de peças que aparecem pra jogada, em cinza os leds apagados

2. ARQUITETURA COMPORTAMENTAL

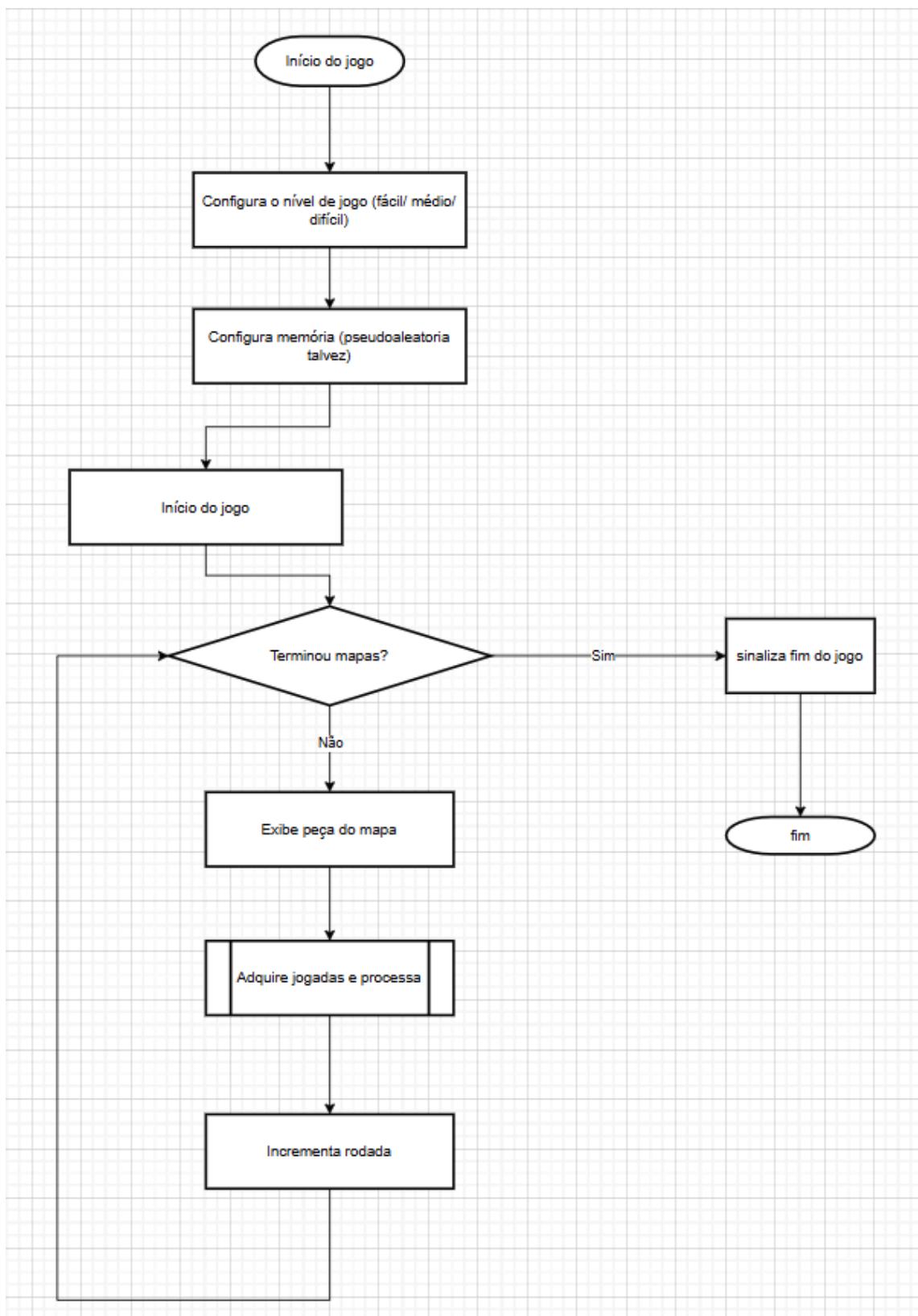


Figura 3: Arquitetura comportamental geral

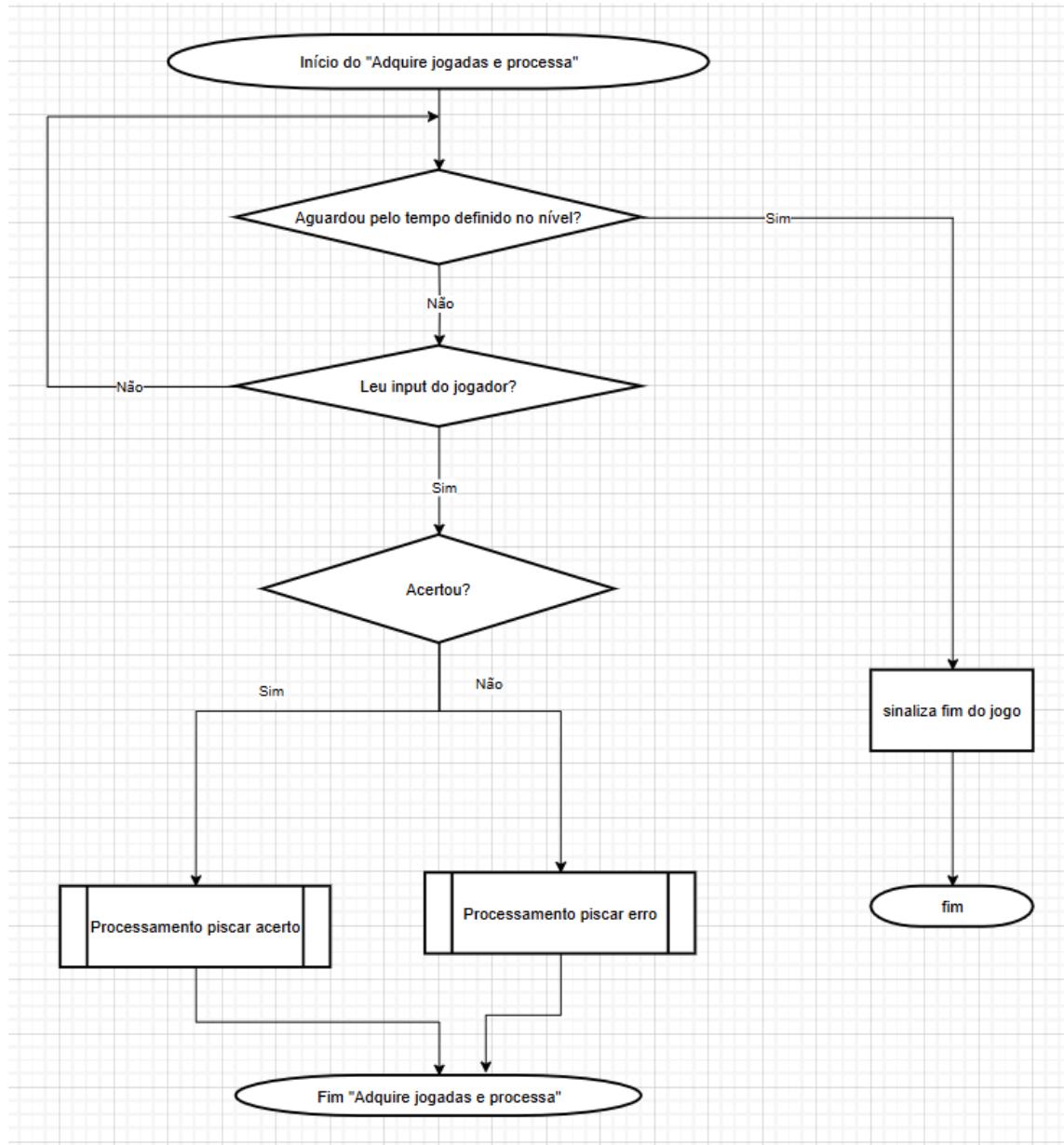


Figura 4: Arquitetura comportamental do processamento de jogadas

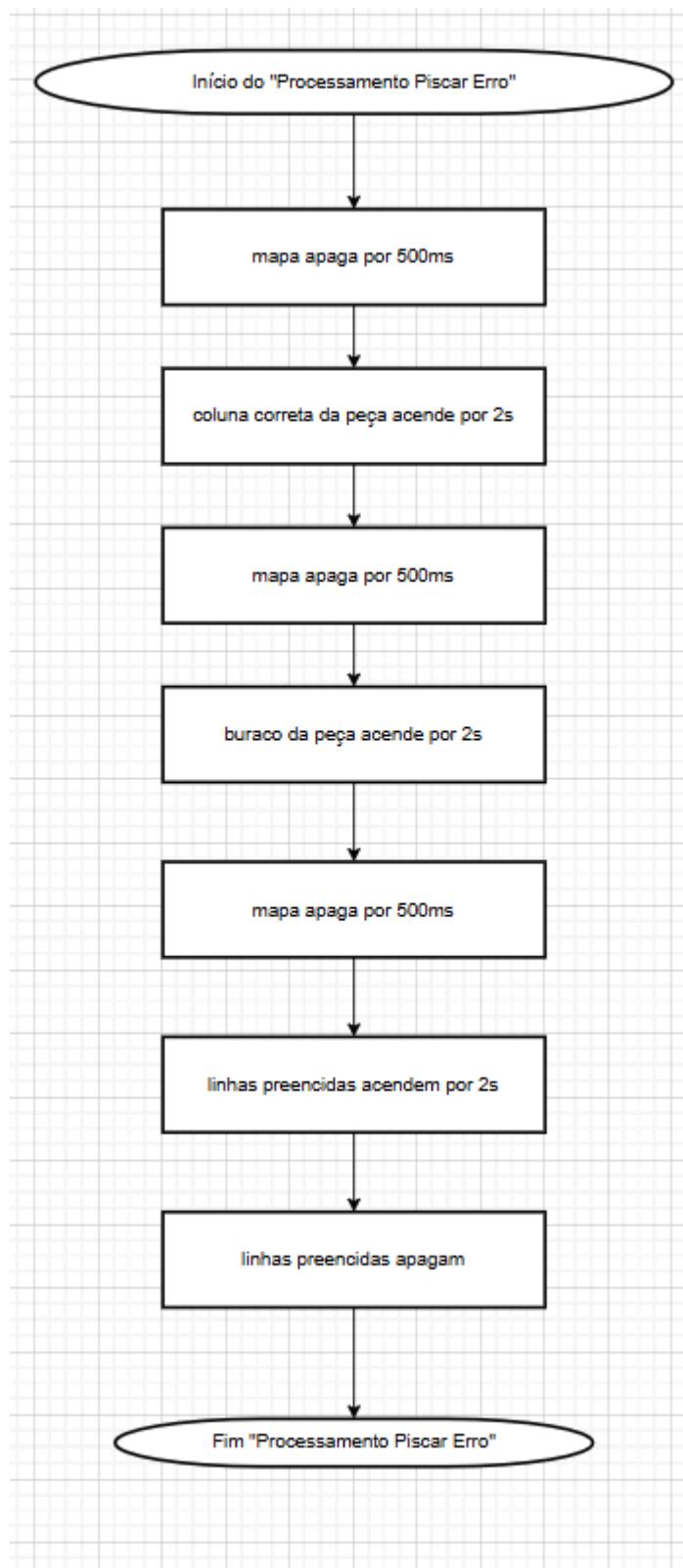


Figura 5: Arquitetura comportamental de exibição de jogada errada

3. ARQUITETURA ESTRUTURAL

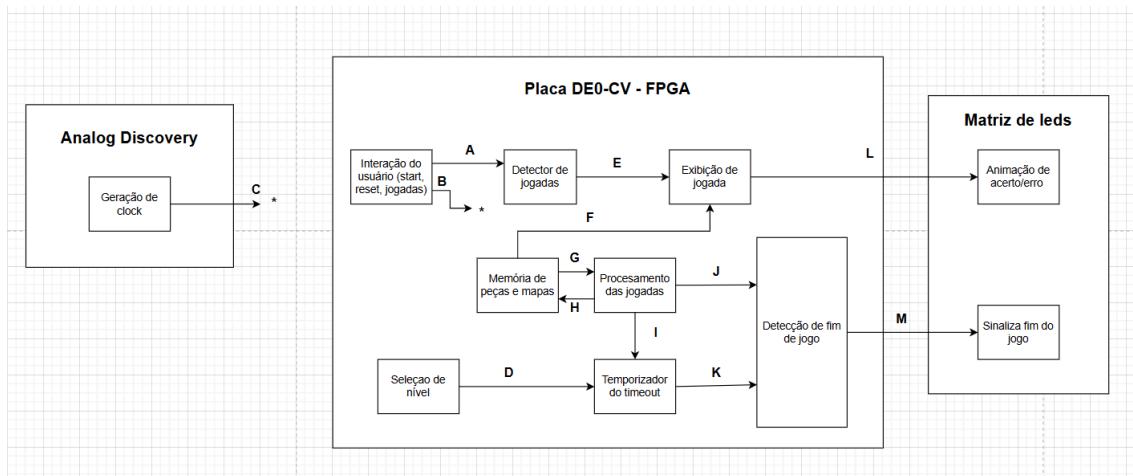


Figura 6: Arquitetura estrutural do projeto

Interface	Descrição
A	Jogadas do usuário
B	Início/Reset
C	Clock
D	Seletor de nível
E	Jogada realizada
F	Peça/Mapa da memória
G	Peça/Mapa da memória
H	Endereço da memória
I	Reinício do temporizador
J	Fim do tabuleiro
K	Timeout
L	Indicações do tabuleiro
M	Sinaliza fim de jogo

Figura 7: Descrição dos itens da figura 6

4. ESPECIFICAÇÃO DOS REQUISITOS

Código: 001	<input checked="" type="checkbox"/> Funcional	<input type="checkbox"/> Não Funcional
Requisito: Seletor de Nível		
Descrição: O sistema deve possuir 3 níveis de jogo: fácil, médio e difícil. O que diferencia cada um desses níveis é o tempo que o jogador terá para realizar a sua jogada (apertar os botões)		
Prioridade: <input type="checkbox"/> Alta <input checked="" type="checkbox"/> Média <input type="checkbox"/> Baixa		
Estabilidade: <input checked="" type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa		
Rationale: Este requisito deve determinar a dificuldade que o usuário terá ao iniciar o jogo. Método de Verificação: simulações de vitória e derrota em diferentes níveis de dificuldade.		
Requisitos associados: Timeout		

Código: 002	<input type="checkbox"/> Funcional	<input checked="" type="checkbox"/> Não Funcional
Requisito: Timeout		
Descrição: Após uma peça ser exibida na matriz, o sistema deverá aguardar um tempo que dependerá da dificuldade do jogo, para que o jogador aperte algum botão. Caso ele não aperte, o jogo terminará.		
Prioridade: <input checked="" type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa		

Estabilidade:	<input type="checkbox"/> Alta	<input checked="" type="checkbox"/> Média	<input type="checkbox"/> Baixa
Rationale: Este requisito deve determinar o tempo que o usuário tem disponível para jogar.			
Método de Verificação: simulações de derrota esperando mais do que o tempo limite em uma jogada.			
Requisitos associados: Seletor de Nível			

Código: 003	<input checked="" type="checkbox"/> Funcional	<input type="checkbox"/> Não Funcional
Requisito: Contador de erros		
Descrição: O sistema deve possuir um contador de erros para que, ao final do jogo, seja exibido o número de erros que o usuário teve.		
Prioridade: <input checked="" type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa		
Estabilidade: <input checked="" type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa		
Rationale: Este requisito deve possibilitar o acompanhamento da evolução cognitiva e motora do jogador.		
Método de Verificação: simulação de uma rodada e verificação de resultados.		
Requisitos associados: Atualiza Display.		

Código: 004	<input checked="" type="checkbox"/> Funcional	<input type="checkbox"/> Não Funcional
Requisito: Atualiza Display		

Descrição: O sistema deve atualizar, a cada jogada, o mapa do jogo.

Prioridade: Alta Média Baixa

Estabilidade: Alta Média Baixa

Rationale: Este requisito deve atualizar os cenários de jogo para que o usuário encaixe novas peças em novas configurações.

Método de Verificação: simulações de jogo em diferentes níveis de dificuldade.

Requisitos associados: : Contador de erros.

Código: 005

Funcional Não Funcional

Requisito: Memória de Mapas

Descrição: O sistema deve possuir uma memória com, no mínimo, 16 mapas diferentes.

Prioridade: Alta Média Baixa

Estabilidade: Alta Média Baixa

Rationale: Este requisito deve possuir todos os cenários de jogo que compõem uma rodada.

Método de Verificação: simulações de uma rodada inteira.

Requisitos associados: Contador de erros.

Código: 006	<input type="checkbox"/> Funcional	<input checked="" type="checkbox"/> Não Funcional
Requisito: Interface de Leds		
Descrição: O sistema deve possuir duas interfaces de leds 8x8 para exibir tanto a peça que será jogada como o cenário em que o usuário jogará.		
Prioridade:	<input checked="" type="checkbox"/> Alta	<input type="checkbox"/> Média
Estabilidade: <input type="checkbox"/> Alta <input checked="" type="checkbox"/> Média <input type="checkbox"/> Baixa		
Rationale: Este requisito deve mostrar ao usuário seu progresso no jogo. Método de Verificação: simulações de jogo em diferentes níveis de dificuldade.		
Requisitos associados: nenhum.		

5. CRONOGRAMA

Semana 1 - Estrutura Base e Entrada do Usuário

- Configuração das 2 matrizes de LEDs 8x8 (Simulada no Digital);
- Mapeamento da primeira matriz de LEDs para exibição da próxima peça;
- Mapeamento da segunda matriz de LEDs para a apresentação do mapa de jogo
- Implementação dos botões de entrada (4 botões, um para cada par de colunas);
- Testes iniciais de ativação de LEDs e resposta aos botões.

Entrega: LEDs respondem corretamente ao controle básico.

Semana 2 - Geração dos mapas e Encaixe das Peças

- Criação do banco de mapas fixos em ROM;

- Exibição da próxima peça na primeira matriz de LEDs;
- Implementação da lógica de encaixe das peças no mapa;
- Bloqueio de encaixe em espaços já ocupados;
- Testes da mecânica de encaixe.

Entrega: O jogador pode visualizar e encaixar peças corretamente no mapa.

Semana 3 - Fixação das Peças

- Fixação das peças na matriz após encaixe;
- Geração de novo mapa após o encaixe, sendo ele certo ou errado;
- Detecção de fim de jogo (game over).

Entrega: O jogo processa os encaixes, muda os mapas até o término do jogo.

Semana 4 - Refinamento e Testes Finais

- Otimização e ajustes finais do código;
- Implementação do contador de erros (espaços vazios);
- Testes finais e documentação.

Entrega: Jogo otimizado e funcional na FPGA, com documentação técnica.

6. SEMANA 1

6.1. Objetivos da semana

O primeiro objetivo desta semana é entender e configurar a matriz de LEDs, simulando seu funcionamento no software Digital.

Com essa base, será possível configurar as matrizes de LEDs 8x8, cada uma desempenhando uma função específica no sistema. A primeira será responsável por exibir a próxima peça, permitindo que o jogador a visualize antecipadamente, enquanto a segunda representará o mapa do jogo, exibindo dinamicamente as peças conforme o progresso da partida.

Além do controle visual das matrizes, também está prevista a implementação do sistema de botões de entrada, composto por quatro botões, cada um acionando

um par de colunas da matriz. Esse mecanismo permitirá que o jogador interaja com o jogo, influenciando diretamente a posição das peças na matriz de LEDs.

Por fim, serão realizados testes iniciais para validar o acionamento dos LEDs e a resposta dos botões, garantindo que cada entrada resulte na ativação correspondente na matriz.

6.2. Requisitos da semana

O modelo de ambas as matrizes de LEDs adquiridas é 2088BS, com configuração de ânodo comum nas colunas e cátodo comum nas linhas, conforme indicado em seu datasheet. Além disso, seu mapeamento foi realizado por meio de análises e testes durante a montagem, permitindo a identificação precisa dos pinos, conforme as ilustrações a seguir.

Nesta semana, o foco está no desenvolvimento da Interface de LEDs, garantindo sua correta implementação e funcionamento dentro do sistema.

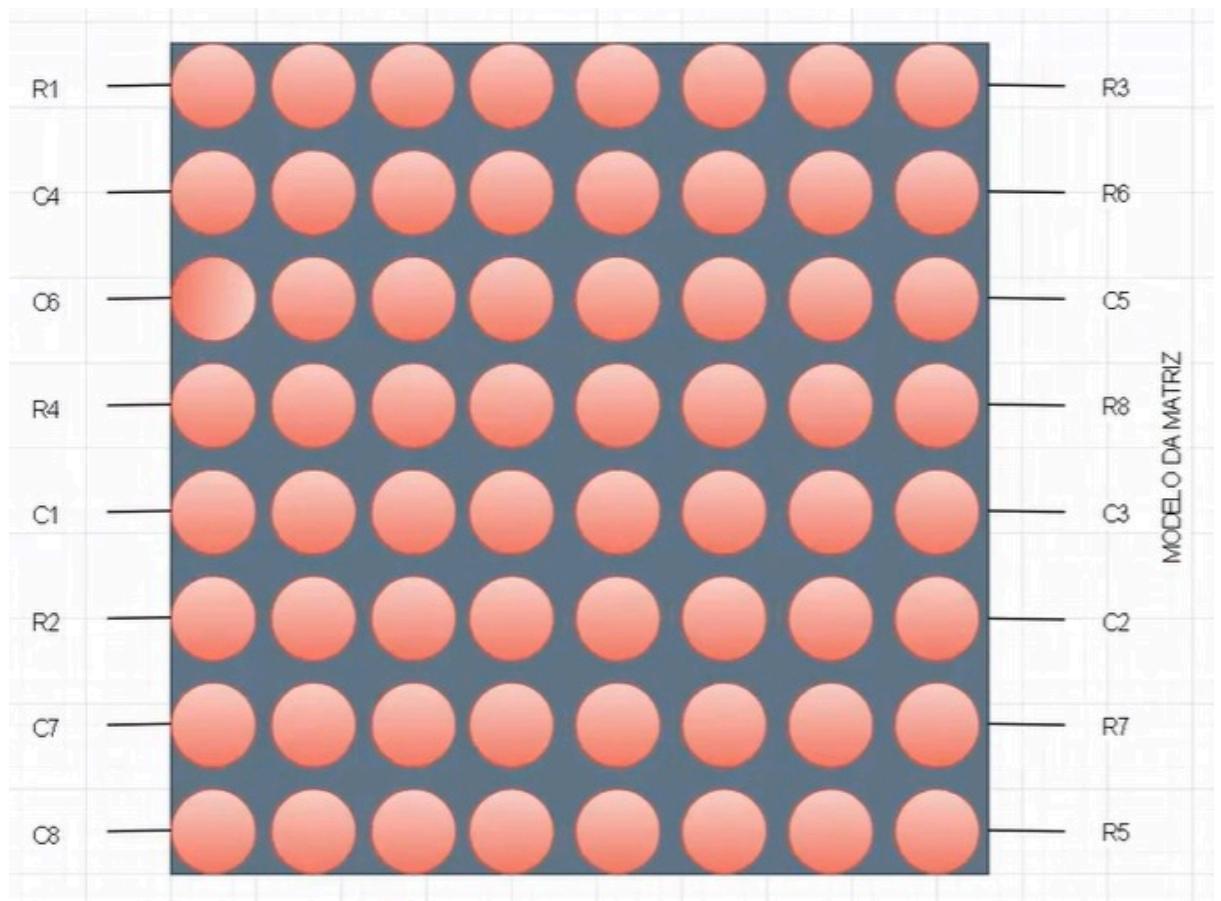


Figura 8: Mapeamento de pinos da Matriz de LEDs 2088BS

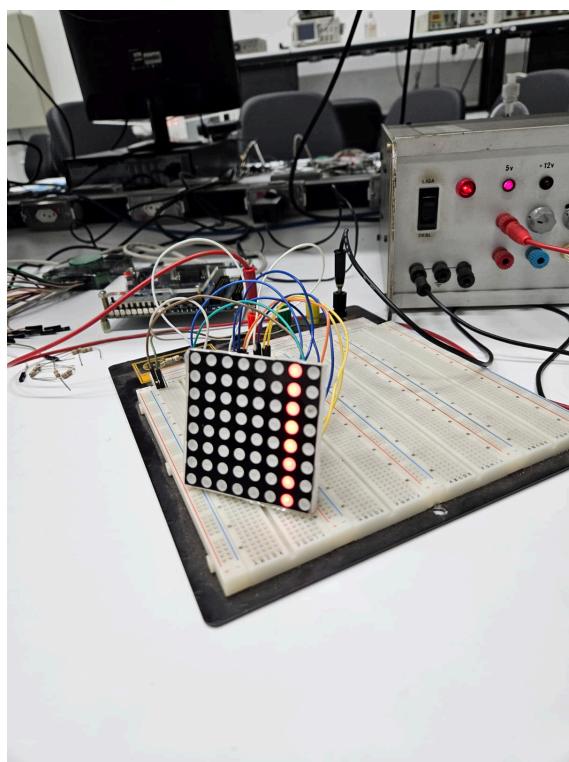


Figura 9: Montagem e teste da Matriz de LEDs na *protoboard*

7. SEMANA 2

7.1. Objetivos da semana

Entre o final da semana 1 e o início da semana 2, foram realizados testes para exibir um mapa simplificado na matriz de LEDs no software *Digital*. Para isso, foi desenvolvido um protótipo utilizando alguns componentes, como um contador, um multiplexador e dois geradores de *clock*, permitindo a simulação do funcionamento do sistema.

A utilização de um segundo gerador de *clock*, em conjunto com o contador, foi essencial para garantir que a matriz de LEDs parecesse sempre acesa ao exibir o mapa. No software *Digital*, quando os LEDs são acionados de forma sequencial sem uma taxa de atualização adequada, pode ocorrer um efeito de piscamento (flickering), no qual os LEDs aparentam acender e apagar rapidamente em vez de permanecerem visivelmente estáveis. Ao sincronizar o contador com um *clock* adicional, foi possível minimizar esse efeito, proporcionando uma exibição mais contínua e fluida do mapa.

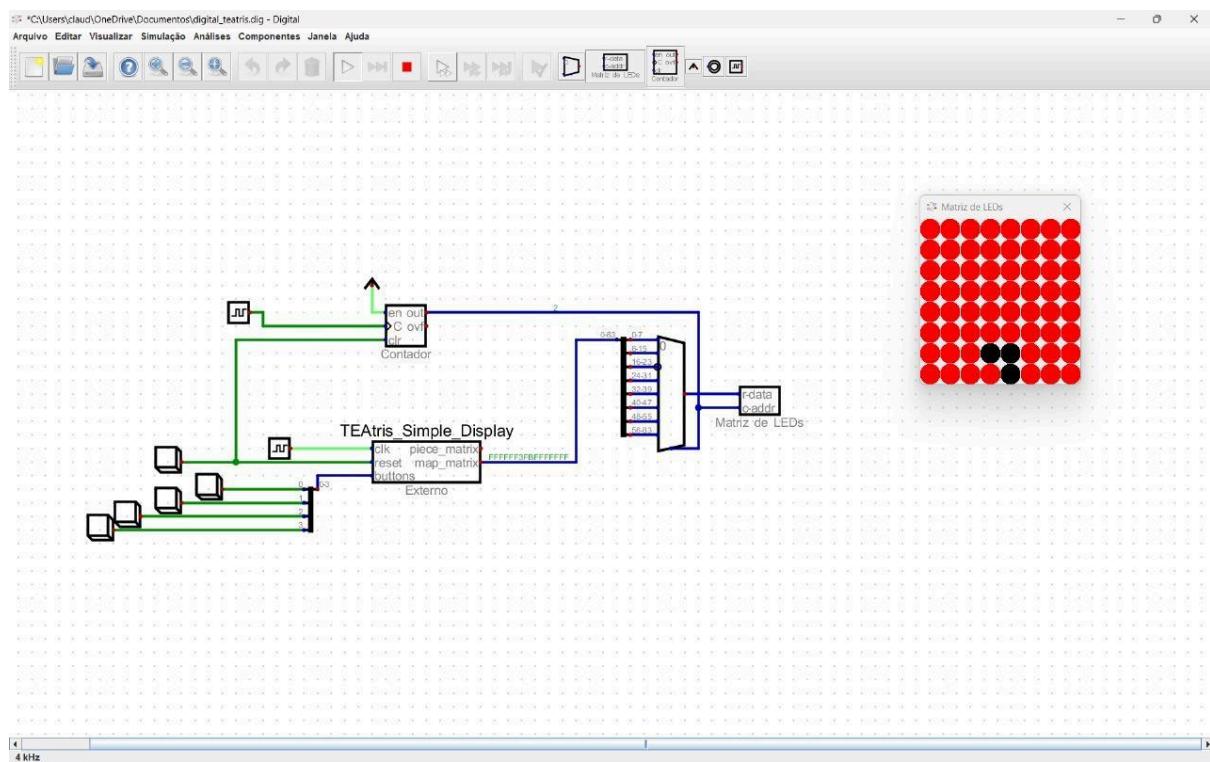


Figura 10: Simulação da exibição do mapa na Matriz de LEDs do *Digital*

A figura acima ilustra um possível mapa do jogo TEAtris. Nesta matriz, serão exibidos os mapas para que o jogador possa analisar e decidir em qual coluna posicionar a peça, garantindo um encaixe correto na matriz inferior.

É importante destacar que, no software Digital, a matriz de LEDs está rotacionada em 90° no sentido anti-horário. Dessa forma, as colunas da imagem abaixo correspondem às linhas de uma Matriz de LEDs montada em uma *protboard* e vice-versa, exigindo que o jogador leve essa diferença em consideração ao interpretar o mapa e tomar suas decisões.

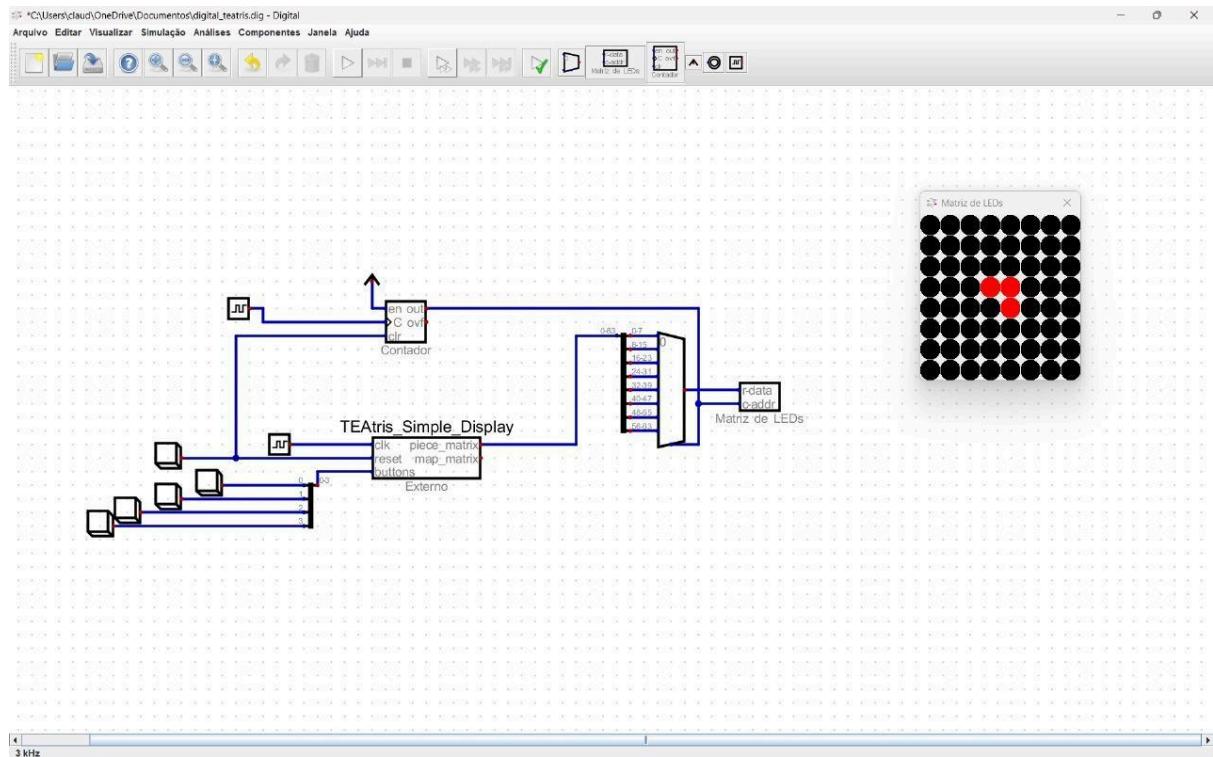


Figura 11: Simulação de exibição de peça que será encaixada na matriz de LEDS do *Digital*.

A figura 11 apresenta uma peça que será mostrada ao jogador. A partir disso, o jogador deverá selecionar a posição correta pelos botões para que o encaixe seja feito.

O uso de duas matrizes de LEDs no software Digital apresentou instabilidades, dificultando a exibição simultânea do mapa do jogo e da peça a ser encaixada. Para contornar esse problema, foram realizados dois testes separados: um para exibir o mapa e outro para exibir a peça.

A imagem abaixo ilustra o objetivo final do sistema, no qual a matriz inferior representa o mapa do jogo, enquanto a matriz superior exibe a próxima peça. A ideia é que o jogador utilize essa visualização para escolher a melhor posição de encaixe da peça no mapa, garantindo um funcionamento fluido e intuitivo do jogo. Como ainda é um protótipo, o nível de dificuldade e as jogadas esperadas são muito simples, porém a equipe busca melhorar isso no decorrer do projeto.

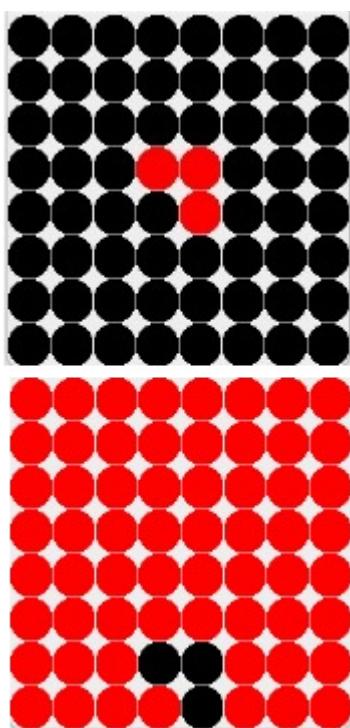


Figura 12: Representação esquemática do funcionamento do jogo. A matriz superior exibe a peça que será jogada, enquanto a matriz inferior mostra o mapa do jogo, indicando os possíveis locais de encaixe.

7.2. Requisitos da semana

Na semana 1, foi possível realizar os testes preliminares na matriz de LEDS 2088BS e identificar o mapeamento das suas posições, conforme a seção 6.2. Realizou-se um teste para acender os LEDs da matriz com dois resistores de 390

Ohms e verificou-se que a intensidade luminosa dos LEDs foi satisfatória. Fizemos também as simulações de exibição de mapas e peças no Digital, o que nos ajudou a concretizar as ideias de como será o circuito em verilog.

Na semana 2, desenvolveremos os códigos verilog necessários para controle da matriz com intuito de cumprir o requisito 006 (Interface de LEDS) e faremos a dinâmica de encaixe das peças. Além disso, conforme planejamento preliminar, faremos as memórias de mapas e testaremos a exibição de cada um deles na matriz, requisito 005 (Memória de Mapas).

O grupo planejou também adquirir nesta semana os 4 botões, resistores de 390 ohms, fios e o protoboard necessário para implementar o TeaTris.

7.3. Projeto lógico

7.3.1. Fluxo de dados

```
// Módulo Datapath
module TEAtris_Datapath (
    input clk,
    input reset,
    input next_piece,
    input move_left,
    input move_right,
    input reset_position,
    input update_display,
    input check_collision,
    output collision_detected,
    output correct_position,
    output [63:0] piece_matrix,
    output [63:0] map_matrix
);
    reg [2:0] piece_index;
    reg [2:0] map_index;
    reg [2:0] piece_position;
    wire [23:0] piece_data;
    wire [7:0] piece_row_0, piece_row_1, piece_row_2;
    wire [63:0] map_data;

    // Instanciação das ROMs
    TEAtris_PieceROM piece_rom (
        .clk(clk),
        .reset(reset),
        .next_piece(next_piece),
        .move_left(move_left),
        .move_right(move_right),
        .reset_position(reset_position),
        .update_display(update_display),
        .check_collision(check_collision),
        .collision_detected(collision_detected),
        .correct_position(correct_position),
        .piece_matrix(piece_matrix),
        .map_matrix(map_matrix)
    );
    piece_rom.piece_rom(piece_index, piece_data, piece_position, piece_row_0, piece_row_1, piece_row_2);
    piece_rom.map_rom(map_index, map_data);
endmodule
```

```

    .piece_index(piece_index),
    .piece_memory(piece_data)
);

TEAtris_MapROM map_rom (
    .map_index(map_index),
    .map_memory(map_data)
);

assign piece_row_0 = piece_data[23:16]; // Primeira linha
assign piece_row_1 = piece_data[15:8]; // Segunda linha
assign piece_row_2 = piece_data[7:0]; // Terceira linha

always @(posedge clk) begin
    if (reset) begin
        piece_index <= 0;
        map_index <= 0;
        piece_position <= 0;
    end else begin
        if (next_piece) begin
            piece_index <= (piece_index + 1) % 6;
            map_index <= (map_index + 1) % 6;
        end
        if (move_left && piece_position > 0) begin
            piece_position <= piece_position - 1;
        end
        if (move_right && piece_position < 3) begin
            piece_position <= piece_position + 1;
        end
        if (reset_position) begin
            piece_position <= 0;
        end
    end
end

// Geração das saídas
generate
    genvar j;
    for (j = 0; j < 8; j = j + 1) begin : matrix_assignments
        assign piece_matrix[j*8 +: 8] = (j < 3) ? (piece_row_0 >>
(piece_position * 2)) : 8'b00000000;
        assign map_matrix[j*8 +: 8] = map_data[j*8 +: 8];
    end

```

```

endgenerate

assign collision_detected = 0; // Implementação futura
assign correct_position = (piece_index == map_index &&
piece_position == map_index);
endmodule

end
else begin
    // Detectar mudanças nos botões
    if (buttons != prev_buttons) begin
        if (buttons[0]) begin
            // Botão 0: Próxima peça
            piece_index <= (piece_index + 1) % 6;
            map_index <= (map_index + 1) % 6; // Manter mapa
            sincronizado com tipo de peça
        end
        else if (buttons[1]) begin
            // Botão 1: Move peça para esquerda (uma posição =
2 colunas)
            if (piece_position > 0)
                piece_position <= piece_position - 1;
        end
        else if (buttons[2]) begin
            // Botão 2: Move peça para direita (uma posição = 2
colunas)
            if (piece_position < 3) // Agora só temos 4
            posições (0-3)
                piece_position <= piece_position + 1;
        end
        else if (buttons[3]) begin
            // Botão 3: Resetar posição
            piece_position <= 0;
        end
    end
    // Atualizar estado dos botões
    prev_buttons <= buttons;

    // Atualizar displays com base nos índices atuais
    // Display de peça - centralizada na matriz
    for (integer i = 0; i < 8; i = i + 1) begin

```

```

        if (i < 3) begin
            // Centralizar peça na matriz (linha 2-4)
            // Cada posição desloca 2 colunas
            piece_display[i+2] <= piece_memory[piece_index][i]
>> (piece_position * 2);
        end
        else if (i < 5) begin
            // Manter linhas 0-1 e 5-7 vazias
            piece_display[i-3] <= 8'b00000000; // Linhas 0-1
            piece_display[i+3] <= 8'b00000000; // Linhas 5-7
        end
    end

    // Display de mapa
    for (integer i = 0; i < 8; i = i + 1) begin
        map_display[i] <= map_memory[map_index][i];
    end
end

// Converter displays para formato de saída
assign piece_matrix = {
    piece_display[0],
    piece_display[1],
    piece_display[2],
    piece_display[3],
    piece_display[4],
    piece_display[5],
    piece_display[6],
    piece_display[7]
};

assign map_matrix = {
    map_display[0],
    map_display[1],
    map_display[2],
    map_display[3],
    map_display[4],
    map_display[5],
    map_display[6],
    map_display[7]
};
endmodule

```

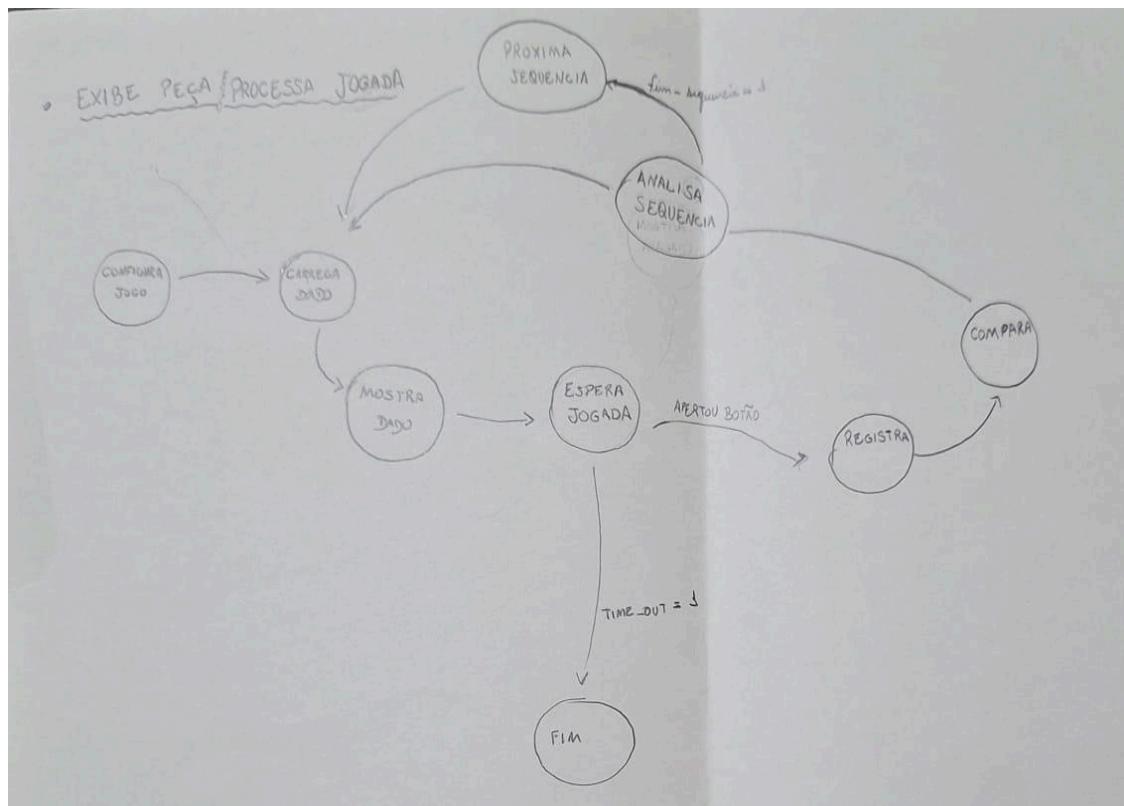


Figura 13: Projeto da unidade de controle do circuito

A figura acima mostra o projeto da Máquina de Transição de Estados desenvolvida pelo grupo. Foram desenvolvidos 9 estados, a saber, Inicial, Configura Jogo, Carrega Dado, Mostra Dado, Espera Jogada, Registra, Compara, Analisa Sequência, Próxima Sequência e Fim. Abaixo há uma descrição sucinta de cada estado:

Tabela 1 – Descrição da Unidade de Controle do Sistema

Nome do Estado	Descrição do Estado	Próximo Estado	Condições e Justificativas para a Transição entre Estados
Inicial	Estado inicial de reset do sistema.	Configura Jogo	Transita para Configura Jogo quando o sinal <i>iniciar</i> é ativado.
Configura Jogo	Prepara o sistema para começar uma	Carrega Dado	

Nome do Estado	Descrição do Estado	Próximo Estado	Condições e Justificativas para a Transição entre Estados
	nova operação		Transição após finalizar a seleção de nível.
Carrega Dado	Carrega o mapa e a peça a ser exibida	Mostra Dado	Transita automaticamente para o estado Mostra Dado.
Mostra Dado	Mostra qual a peça que o jogador irá encaixar	Espera Jogada	Transita automaticamente para Espera Jogada
Espera Jogada	Espera jogador selecionar um dos 4 botões	Registra ou Timeout	Transita automaticamente após armazenar a jogada.
Registra	Registra a posição em que o jogador fez sua jogada	Compara	Transições ocorre após o jogador pressionar um dos botões
Compara	Comparação para analisar se a jogada foi na posição esperada	Analisa Sequencia	Transita automaticamente para Analisa Sequencia
Analisa Sequencia	Verifica se o jogador terminou	Proxima Sequencia	Transita para Proxima sequencia se as jogadas acabaram.

Nome do Estado	Descrição do Estado	Próximo Estado	Condições e Justificativas para a Transição entre Estados
	as jogadas e se os mapas acabaram.		
Proxima Sequencia	Inicia um novo mapa	Carrega Dado	Transita para Carrega Dado automaticamente.
Timeout	Indica que o jogador esgotou seu tempo para jogada.	Inicial	Transita para Inicial se <i>iniciar</i> for ativado
Fim	Indica que todas as jogadas foram realizadas corretamente.	Inicial	Transita para Inicial se <i>iniciar</i> for ativado

7.3.2. Unidade de Controle

```
module TEAtris_Control_Unit (
    input clk,                      // Clock do sistema
    input reset,                     // Sinal de reset
    input [3:0] buttons,             // Botões de entrada (um para cada par de colunas)
    input timeout,                  // Sinal de timeout do temporizador

    // Sinais de controle para o fluxo de dados
    output reg next_piece,          // Sinal para avançar para próxima peça
    output reg move_left,           // Sinal para mover peça para esquerda
    output reg move_right,          // Sinal para mover peça para direita
    output reg reset_position,      // Sinal para resetar posição
    output reg update_display,       // Sinal para atualizar display
    output reg check_collision,     // Verificar colisões

    // Sinais de estado do jogo
```

```

        output reg [3:0] game_state,      // Estado atual do jogo
        output reg timer_restart,       // Reiniciar o temporizador
        output reg game_over           // Sinal de fim de jogo
    );

    // Definição dos estados
    localparam STATE_INIT = 4'b0000;
    localparam STATE_CONFIG = 4'b0001;
    localparam STATE_LOAD_DATA = 4'b0010;
    localparam STATE_SHOW_DATA = 4'b0011;
    localparam STATE_WAIT_MOVE = 4'b0100;
    localparam STATE_REGISTER = 4'b0101;
    localparam STATE_COMPARE = 4'b0110;
    localparam STATE_ANALYZE_SEQ = 4'b0111;
    localparam STATE_NEXT_SEQ = 4'b1000;
    localparam STATE_SUCCESS = 4'b1001;
    localparam STATE_ERROR = 4'b1010;
    localparam STATE_TIMEOUT = 4'b1011;

    // Registrador para armazenar o estado anterior dos botões
    reg [3:0] prev_buttons;

    always @ (posedge clk) begin
        if (reset) begin
            game_state <= STATE_INIT;
            next_piece <= 0;
            move_left <= 0;
            move_right <= 0;
            reset_position <= 0;
            update_display <= 0;
            check_collision <= 0;
            timer_restart <= 1;
            game_over <= 0;
            prev_buttons <= 4'b0000;
        end
        else begin
            // Resetar sinais de controle
            next_piece <= 0;
            move_left <= 0;
            move_right <= 0;
            reset_position <= 0;
            check_collision <= 0;
            timer_restart <= 0;
        end
    end

```

```

update_display <= 1;

case (game_state)
    STATE_INIT: begin
        game_state <= STATE_CONFIG;
    end

    STATE_CONFIG: begin
        game_state <= STATE_LOAD_DATA;
    end

    STATE_LOAD_DATA: begin
        game_state <= STATE_SHOW_DATA;
    end

    STATE_SHOW_DATA: begin
        game_state <= STATE_WAIT_MOVE;
    end

    STATE_WAIT_MOVE: begin
        if (buttons != prev_buttons) begin
            if (buttons[0] && !prev_buttons[0]) begin
                game_state <= STATE_REGISTER;
            end
            else if (buttons[1] && !prev_buttons[1]) begin
                move_left <= 1;
            end
            else if (buttons[2] && !prev_buttons[2]) begin
                move_right <= 1;
            end
            else if (buttons[3] && !prev_buttons[3]) begin
                reset_position <= 1;
            end
        end
    end

    STATE_REGISTER: begin
        game_state <= STATE_COMPARE;
    end

    STATE_COMPARE: begin
        if (timeout) begin
            game_state <= STATE_TIMEOUT;
        end
    end

```

```

        end else begin
            game_state <= STATE_ANALYZE_SEQ;
        end
    end

    STATE_ANALYZE_SEQ: begin
        if /* condição de acerto */) begin
            game_state <= STATE_SUCCESS;
        end else if /* condição de erro */) begin
            game_state <= STATE_ERROR;
        end else begin
            game_state <= STATE_NEXT_SEQ;
        end
    end

    STATE_NEXT_SEQ: begin
        game_state <= STATE_LOAD_DATA;
    end

    STATE_SUCCESS: begin
        game_state <= STATE_CONFIG;
    end

    STATE_ERROR: begin
        game_over <= 1;
        game_state <= STATE_CONFIG;
    end

    STATE_TIMEOUT: begin
        game_over <= 1;
        game_state <= STATE_CONFIG;
    end
endcase

prev_buttons <= buttons;
end
end

endmodule

```

Obs: Tanto a Unidade de Controle quanto o fluxo de dados ainda não estão em suas versões finais.

7.4. Componentes testados

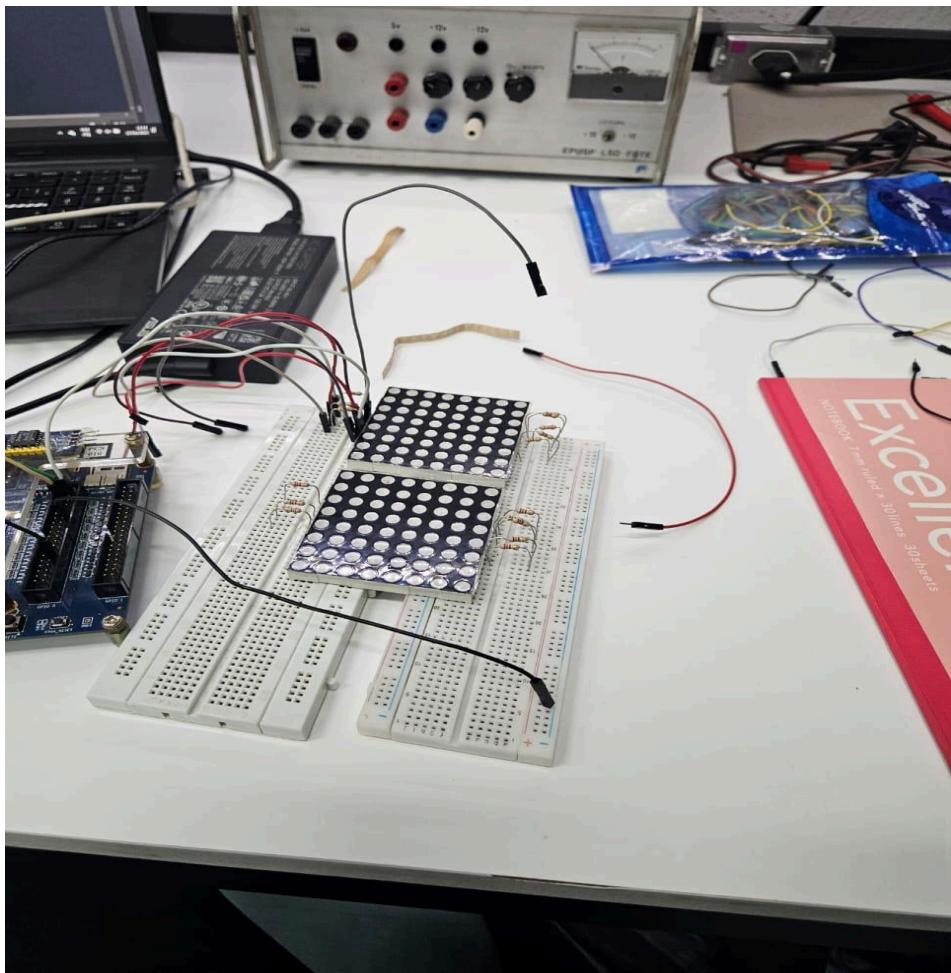


Figura 14: Início da montagem dos *displays*

O grupo foi capaz de iniciar a montagem dos displays do jogo (Matrizes de Led). A matriz superior será utilizada para mostrar as peças e a inferior para encaixá-las. Vale ressaltar que, diferentemente da semana 1, foram utilizados resistores de 220 ohms, ao invés de 390 ohms, o que garantiu uma maior corrente e um brilho mais satisfatório para os leds em condição de tensões mais próximas à da saída da placa *FPGA DE0-CV* (~3.3V).

7.5. Componentes Adquiridos

Nessa semana foram adquiridos os botões (*push buttons*) e capas para eles visando melhorar a experiência de usuário com o jogo. Foram comprados 10 *push*

buttons (4x para seleção de coluna da jogada, 1x para *Start*, 1x para *Reset*, 4x sobressalentes) e 4 capas coloridas (1x vermelho, 1x verde, 1x azul, 1x amarelo) para os botões de seleção das colunas da jogada.



Especificações técnicas

TENSÃO MÁXIMA	250V
CORRENTE MÁXIMA	50mA
DIMENSÕES	12 x 12 x 8,5mm

Figura 15: Especificações dos botões adquiridos

Descrição

Estas capas permitirão uma enorme variedade de combinações para incrementar o visual do seu projeto!



Especificações técnicas

DIÂMETRO	11mm
APLICAÇÕES	Em botões tipo push buttons que possuem o botão quadrado Compatível com Push Button (Chave Táctil) 12x12x7,3mm Não compatível com PS-22E85L
ENCAIXE	Na protoboard

Figura 16: Capas coloridas para identificação de cada botão-coluna

8. SEMANA 3

8.1. Objetivos da semana

O foco desta semana é a implementação da lógica de fixação das peças na matriz de jogo, etapa essencial para o andamento da partida. O primeiro objetivo é garantir que, ao detectar o encaixe de uma peça — seja ele correto ou incorreto — o sistema seja capaz de fixá-la na matriz correspondente, refletindo visualmente a nova configuração do jogo.

A partir dessa fixação, o sistema deverá gerar um novo mapa de jogo, atualizando a matriz de acordo com o resultado do encaixe anterior. Esse processo é importante tanto para o controle de fluxo quanto para a experiência do jogador, permitindo uma transição clara entre as rodadas.

Outro objetivo importante desta etapa é a implementação da detecção de fim de jogo (*game over*), assegurando que, ao atingir determinada condição de término (como a impossibilidade de novos encaixes), o sistema sinalize o encerramento da partida de forma adequada.

Ao final da semana, espera-se que o jogo esteja apto a processar os encaixes das peças, atualizar dinamicamente os mapas e identificar corretamente o fim da partida.

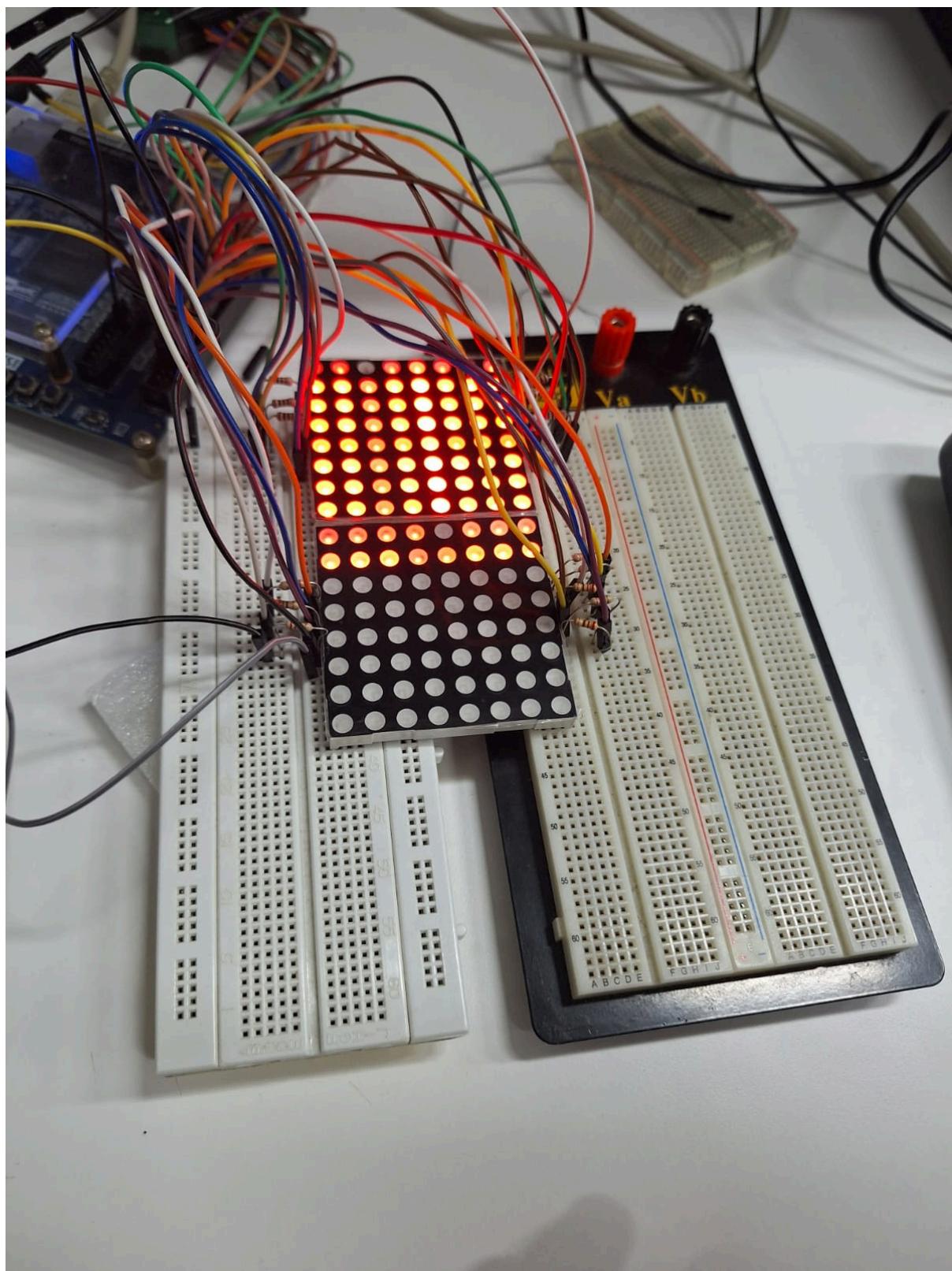


Figura 17: Ilustração da montagem do projeto na *protoboard*

8.2. Requisitos da semana

Nas últimas semanas os requisitos trabalhados foram o 5 e 6, que abordaram a Memória de mapas e a Interface de *LEDs*, respectivamente. Ambos estão bem encaminhados e só precisam de refinamento, que será feito no futuro. A partir da semana 3, a equipe trabalhará nos requisitos 2 e 4, que envolvem o *Timeout* e Atualizar o *Display*. Com a conclusão do requisito 2, torna-se mais viável a implementação do requisito 1, que trata da seleção de dificuldade, já que esse processo depende diretamente do controle de *Timeout*. Além disso, ao avançar com o requisito 4, a estrutura central (*core*) do projeto se tornará mais clara e consolidada.

8.3. Projeto lógico

8.3.1. Unidade de Controle

Após alguns testes no *protoboard*, a equipe identificou que a unidade de controle ainda não tinha uma máquina de estados bem definida. Dessa forma, foi desenvolvida uma máquina de estados baseada no projeto do jogo *Genius*, que apresenta uma estrutura similar. Essa implementação ainda será testada e, caso necessário, será refinada para assegurar o funcionamento adequado do sistema.

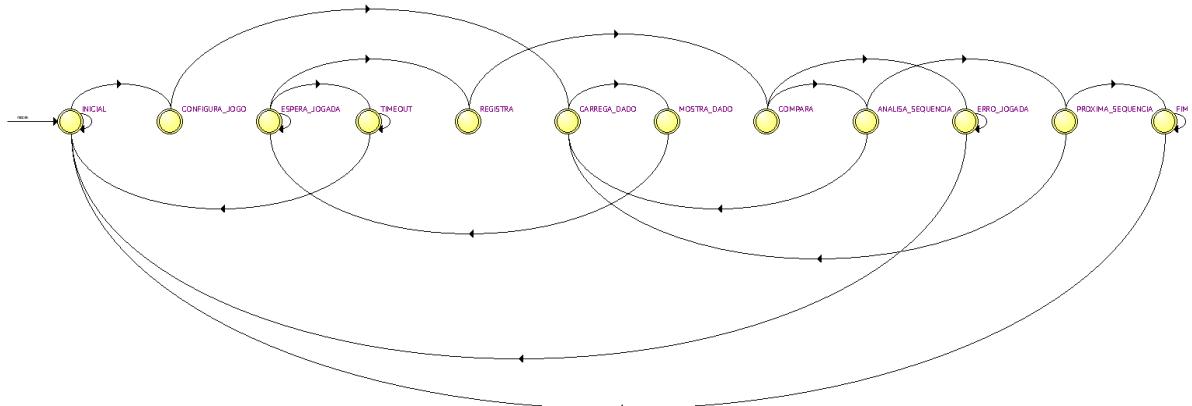


Figura 18: State Machine Viewer gerada no Quartus

	Source State	Destination State	Condition
1	ANALISA_SEQUENCIA	CARREGA_DADO	{!fim_sequencia}
2	ANALISA_SEQUENCIA	PROXIMA_SEQUENCIA	{fim_sequencia}
3	CARREGA_DADO	MOSTRA_DADO	
4	COMPARA	ANALISA_SEQUENCIA	{jogada_ok}
5	COMPARA	ERRO_JOGADA	{!jogada_ok}
6	CONFIGURA_JOGO	CARREGA_DADO	
7	ERRO_JOGADA	ERRO_JOGADA	{!start}
8	ERRO_JOGADA	INICIAL	{start}
9	ESPERA_JOGADA	ESPERA_JOGADA	{!item_jogada}. {!timeout}
10	ESPERA_JOGADA	REGISTRA	{item_jogada}. {!timeout}
11	ESPERA_JOGADA	TIMEOUT	{timeout}
12	FIM	FIM	{!start}
13	FIM	INICIAL	{start}
14	INICIAL	CONFIGURA_JOGO	{start}
15	INICIAL	INICIAL	{!start}
16	MOSTRA_DADO	ESPERA_JOGADA	
17	PROXIMA_SEQUEN...	CARREGA_DADO	{!dificuldade}
18	PROXIMA_SEQUEN...	FIM	{dificuldade}
19	REGISTRA	COMPARA	
20	TIMEOUT	INICIAL	{start}
21	TIMEOUT	TIMEOUT	{!start}

Figura 19: Tabela de transição de estados produzida pelo *RTL Viewer*

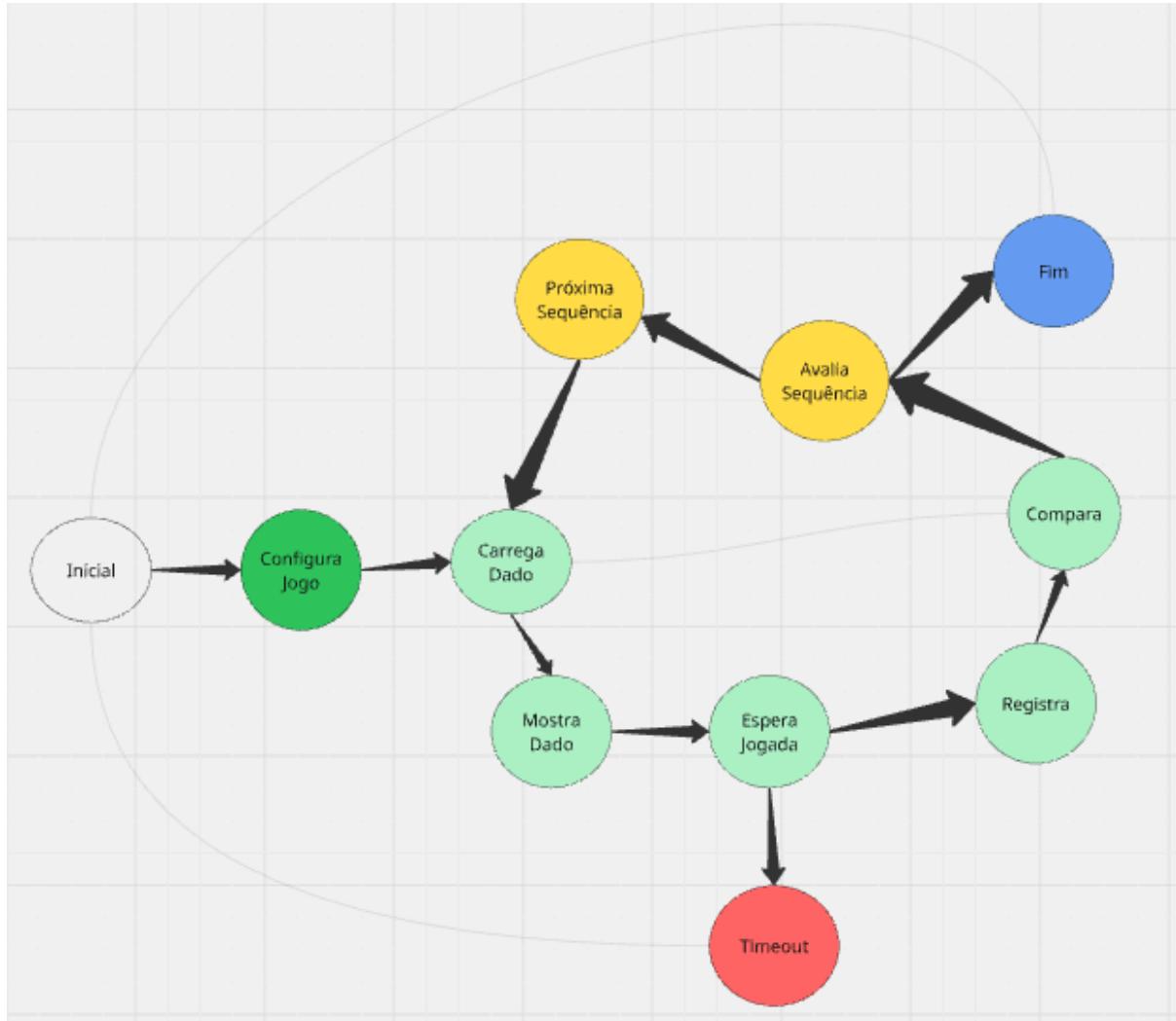


Figura 20: Diagrama de transição de estados feita com o *Miro*

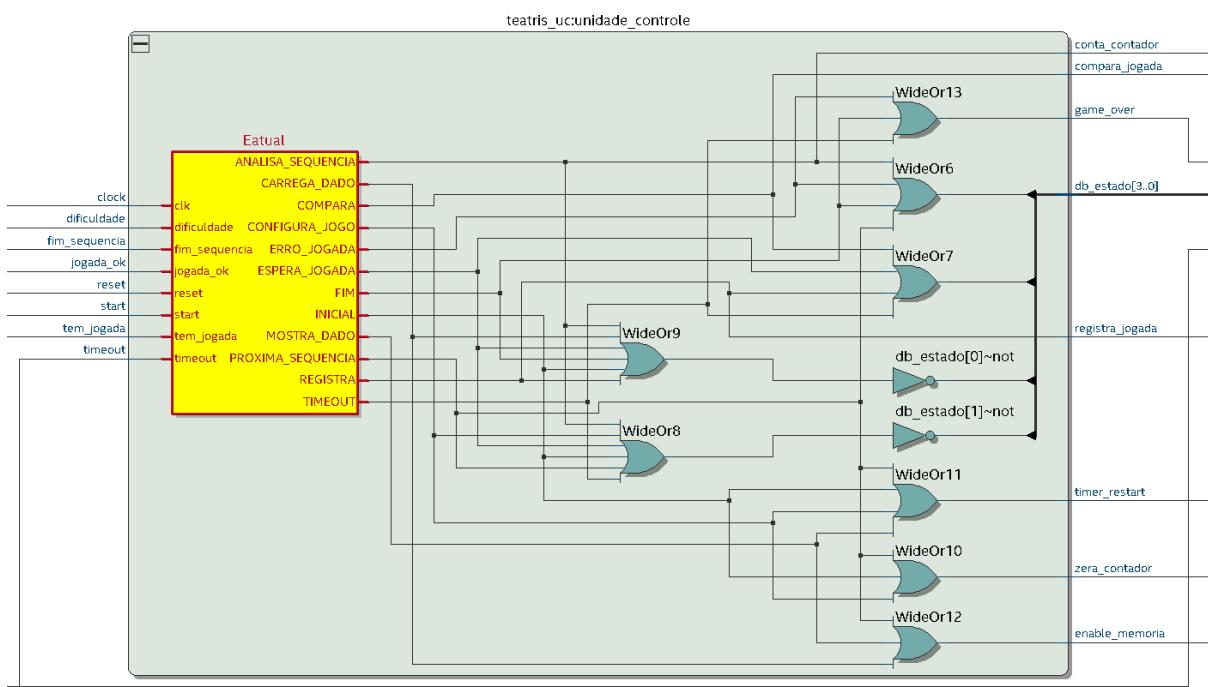


Figura 21: Arquitetura da unidade de controle no *RTL Viewer*

```
module teatris_uc (
    input clock,           // Clock do sistema
    input reset,            // Sinal de reset
    input timeout,          // Sinal de timeout do temporizador
    input start,             // Sinal para iniciar jogo
    input dificuldade,       // Nível de dificuldade
    input fim_sequencia,     // Indicador de fim da sequência
    input tem_jogada,        // Indicador de jogada realizada
    input jogada_ok,         // Indicador de jogada correta

    // Sinais de controle para o fluxo de dados
    output reg zera_contador,   // Zera o contador de endereços
    output reg conta_contador,  // Incrementa o contador de endereços
    output reg enable_memoria, // Habilita registrador de memória
    output reg registra_jogada, // Habilita registro da jogada
    output reg compara_jogada, // Habilita comparação da jogada
    output reg timer_restart,   // Reiniciar o temporizador
    output reg game_over,        // Sinal de fim de jogo

    // Saída de depuração
    output reg [3:0] db_estado // Estado atual do jogo
);

// Estados do jogo
localparam INICIAL = 4'b0000;
localparam CONFIGURA_JOGO = 4'b0001;
localparam CARREGA_DADO = 4'b0010;
localparam MOSTRA_DADO = 4'b0011;
localparam ESPERA_JOGADA = 4'b0100;
localparam TIMEOUT = 4'b0101;
localparam REGISTRA = 4'b0110;
localparam COMPARA = 4'b0111;
localparam ANALISA_SEQUENCIA = 4'b1000;
localparam PROXIMA_SEQUENCIA = 4'b1001;
localparam FIM = 4'b1010;
localparam ERRO_JOGADA = 4'b1011;

reg [3:0] Eactual, Eprox;

// Memória de Estado
always @(posedge clock or posedge reset) begin
```

```

if (reset)
    Eatual <= INICIAL;
else
    Eatual <= Eprox;
end

// Lógica para determinar próximo estado
always @(*) begin
    case (Eatual)
        INICIAL:
            Eprox = start ? CONFIGURA_JOGO : INICIAL;

        CONFIGURA_JOGO:
            Eprox = CARREGA_DADO;

        CARREGA_DADO:
            Eprox = MOSTRA_DADO;

        MOSTRA_DADO:
            Eprox = ESPERA_JOGADA;

        ESPERA_JOGADA:
            if (timeout)
                Eprox = TIMEOUT;
            else if (tem_jogada)
                Eprox = REGISTRA;
            else
                Eprox = ESPERA_JOGADA;

        TIMEOUT:
            Eprox = start ? INICIAL : TIMEOUT;

        REGISTRA:
            Eprox = COMPARA;

        COMPARA:
            if (jogada_ok)
                Eprox = ANALISA_SEQUENCIA;
            else
                Eprox = ERRO_JOGADA;

        ANALISA_SEQUENCIA:

```

```

    Eprox = fim_sequencia ? PROXIMA_SEQUENCIA :
CARREGA_DADO;

    PROXIMA_SEQUENCIA:
        Eprox = (dificuldade) ? FIM : CARREGA_DADO;

    ERRO_JOGADA:
        Eprox = start ? INICIAL : ERRO_JOGADA;

    FIM:
        Eprox = start ? INICIAL : FIM;

    default:
        Eprox = INICIAL;
endcase
end

// Lógica de saída (máquina Moore - saídas dependem apenas do
estado atual)
always @(*) begin
    // Valores default
    zera_contador = 1'b0;
    conta_contador = 1'b0;
    enable_memoria = 1'b0;
    registra_jogada = 1'b0;
    compara_jogada = 1'b0;
    timer_restart = 1'b0;
    game_over = 1'b0;
    db_estado = Eatual; // Sempre mostra o estado atual

    case (Eatual)
        INICIAL: begin
            zera_contador = 1'b1;
            timer_restart = 1'b1;
        end

        CONFIGURA_JOGO: begin
            zera_contador = 1'b1;
            timer_restart = 1'b1;
        end

        CARREGA_DADO: begin
            enable_memoria = 1'b1;

```

```
    end

    MOSTRA_DADO: begin
        enable_memoria = 1'b1;
            timer_restart = 1'b1;
        end

    ESPERA_JOGADA: begin
        // Aguarda entrada
    end

    TIMEOUT: begin
        game_over = 1'b1;
    end

    REGISTRA: begin
        registra_jogada = 1'b1;
    end

    COMPARA: begin
        compara_jogada = 1'b1;
    end

    ANALISA_SEQUENCIA: begin
        conta_contador = 1'b1;
    end

    PROXIMA_SEQUENCIA: begin
        zera_contador = 1'b1;
        enable_memoria = 1'b1;
        timer_restart = 1'b1;
    end

    ERRO_JOGADA: begin
        game_over = 1'b1;
    end

    FIM: begin
        game_over = 1'b1;
    end

    default: begin
```

```

        zera_contador = 1'b1;
    end
endcase
end

endmodule

```

8.3.2. Fluxo de Dados

O fluxo de dados vem sendo implementado de forma a se adequar à lógica do projeto. Recentemente, foram realizadas algumas modificações, como a adição do tratamento para detecção de jogadas. Inspirando-se na abordagem utilizada no projeto do jogo Genius, foi integrado um componente *edge_detector* para capturar as jogadas realizadas, de maneira semelhante àquela já empregada anteriormente.

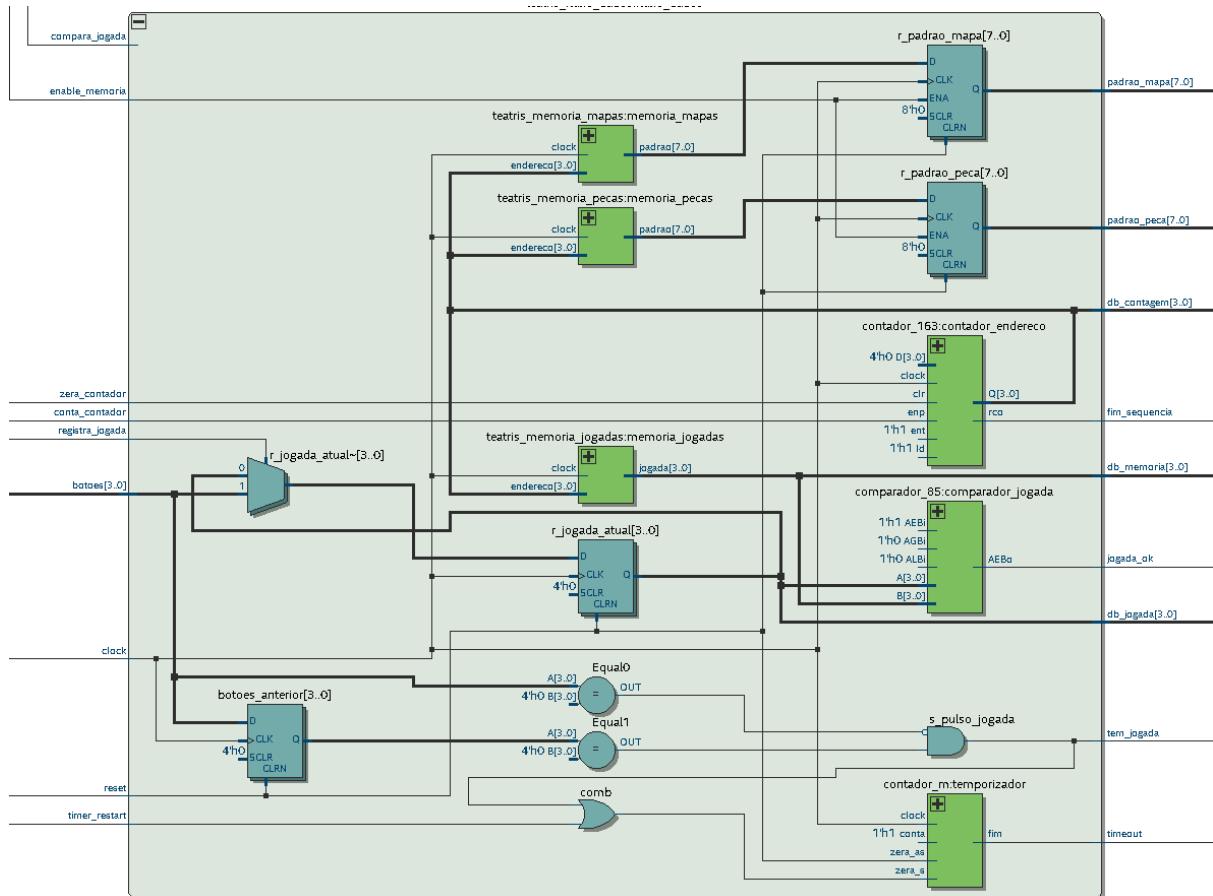


Figura 22: Arquitetura do fluxo de dados no *RTL Viewer*

```

module teatris_fluxo_dados (
    // Entradas básicas
    input clock,                      // Clock do sistema
    input reset,                       // Sinal de reset
    input [3:0] botoes,                // Botões de entrada

    // Sinais de controle da UC
    input zera_contador,              // Zera o contador de endereços
    input conta_contador,             // Incrementa o contador
    input enable_memoria,             // Habilita leitura/registro da memória
    input registra_jogada,            // Registra a jogada atual
    input compara_jogada,             // Habilita comparação de jogada
    input timer_restart,               // Reinicia o temporizador

    // Sinais de estado para a UC
    output tem_jogada,                // Indica que um botão foi pressionado
    output jogada_ok,                 // Indica que a jogada está correta
    output fim_sequencia,              // Indica fim da sequência atual
    output timeout,                   // Indica que o tempo se esgotou

    // Saídas para exibição
    output [7:0] padrao_peca,          // Padrão para matriz de peça
    output [7:0] padrao_mapa,           // Padrão para matriz de mapa

    // Saídas de depuração
    output [3:0] db_contagem,           // Valor atual do contador
    output [3:0] db_jogada,              // Jogada atual registrada
    output [3:0] db_memoria,             // Valor esperado da memória
);

    // Sinais internos
    wire [3:0] s_endereco;             // Endereço atual da memória
    wire [7:0] s_padrao_peca;           // Padrão de peça da memória
    wire [7:0] s_padrao_mapa;           // Padrão de mapa da memória
    wire [3:0] s_jogada_correta;        // Jogada correta para o padrão atual
    reg [3:0] r_jogada_atual;           // Jogada atual registrada
    reg [7:0] r_padrao_peca;            // Registrador para padrão de peça
    reg [7:0] r_padrao_mapa;            // Registrador para padrão de mapa

    // Detector de borda para identificar jogadas
    reg [3:0] botoes_anterior;          // Registrador para detectar mudança
nos botões

```

```

    wire s_pulso_jogada;           // Pulso quando um botão é
pressionado

    // Detecta quando qualquer botão foi pressionado
    always @(posedge clock or posedge reset) begin
        if (reset)
            botoes_anterior <= 4'b0000;
        else
            botoes_anterior <= botoes;
    end

    // Gera pulso quando um novo botão é pressionado (borda de subida)
    assign s_pulso_jogada = (botoes != 4'b0000) & (botoes_anterior ==
4'b0000);
    assign tem_jogada = s_pulso_jogada;

    // Contador de endereços para acessar a memória
    contador_163 contador_endereco (
        .clock(clock),
        .clr(zera_contador),
        .ld(1'b1),                  // Load ativo alto
        .ent(1'b1),                  // Enable de contagem
        .enp(conta_contador),       // Enable de incremento
        .D(4'b0000),                // Valor inicial
        .Q(s_endereco),             // Saída do contador
        .rco(fim_sequencia)        // Ripple Carry Out como fim de
sequência
    );

```

// Memórias ROM

```

    teatris_memoria_pecas memoria_pecas (
        .clock(clock),
        .endereco(s_endereco),
        .padrao(s_padrao_peca)
    );

```

teatris_memoria_mapas memoria_mapas (

```

        .clock(clock),
        .endereco(s_endereco),
        .padrao(s_padrao_mapa)
    );

```

teatris_memoria_jogadas memoria_jogadas (

```

    .clock(clock),
    .endereco(s_endereco),
    .jogada(s_jogada_correta)
);

// Registrador para armazenar botão pressionado
always @(posedge clock or posedge reset) begin
    if (reset)
        r_jogada_atual <= 4'b0000;
    else if (registra_jogada)
        r_jogada_atual <= botoes;
end

// Registradores para armazenar padrões atuais
always @(posedge clock or posedge reset) begin
    if (reset) begin
        r_padrao_peca <= 8'b00000000;
        r_padrao_mapa <= 8'b00000000;
    end
    else if (enable_memoria) begin
        r_padrao_peca <= s_padrao_peca;
        r_padrao_mapa <= s_padrao_mapa;
    end
end

// Comparador para verificar jogada
comparador_85 comparador_jogada (
    .A(r_jogada_atual),           // Jogada registrada
    .B(s_jogada_correta),         // Jogada esperada da memória
    .ALBi(1'b0),
    .AGBi(1'b0),
    .AEBi(1'b1),
    .ALBo(),
    .AGBo(),
    .AEBo(jogada_ok)            // Saída: jogada correta ou não
);

// Temporizador para timeout
contador_m temporizador (
    .clock(clock),
    .zera_as(reset),
    .zera_s(timer_restart || s_pulso_jogada),
    .conta(1'b1),                // Sempre contando

```

```

        .Q(),
        // Valor do contador (não usado)
        .fim(timeout)
        // Timeout quando chega ao fim
    );

    // Atribuição das saídas
    assign padrao_peca = r_padrao_peca;
    assign padrao_mapa = r_padrao_mapa;

    // Saídas de depuração
    assign db_contagem = s_endereco;
    assign db_jogada = r_jogada_atual;
    assign db_memoria = s_jogada_correta;

endmodule

```

8.3.3. Circuito Digital

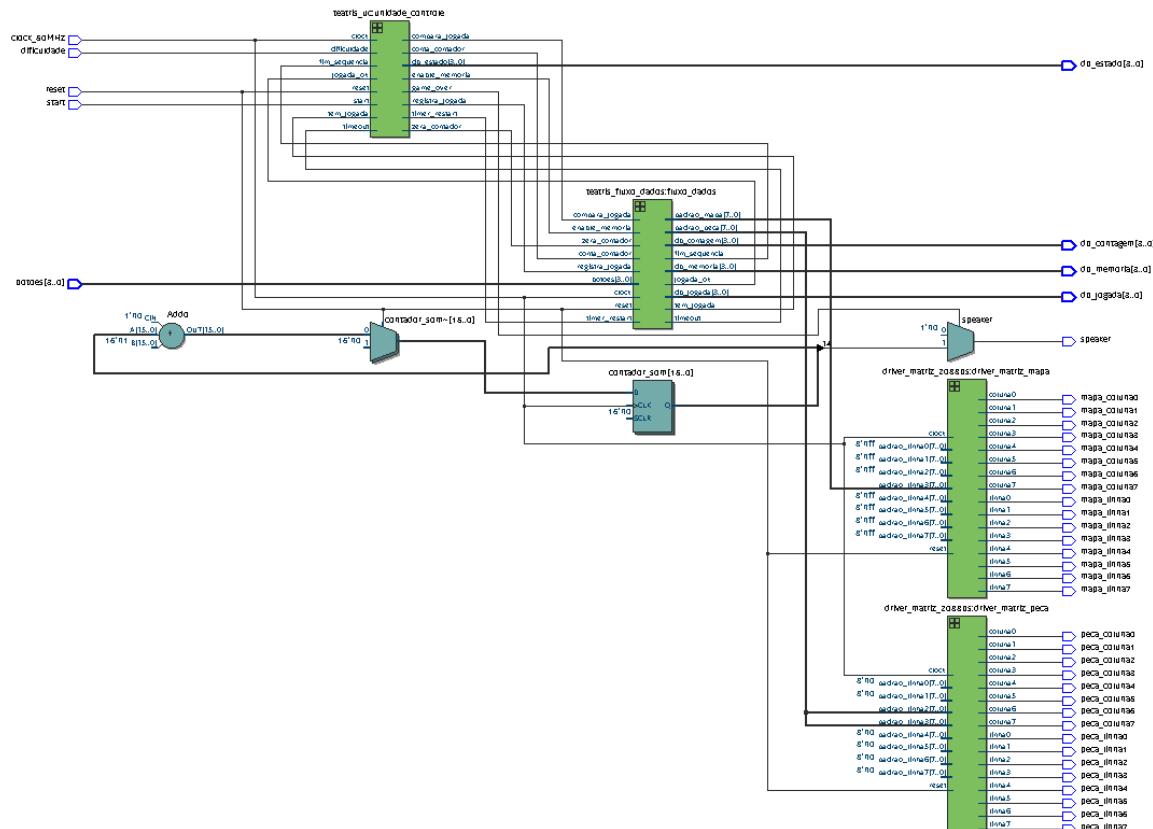


Figura 23: Arquitetura do circuito digital no *RTL Viewer*

```
module TEAtris_Top (
    // Entradas principais
    input clock_50MHz,           // Clock principal (tipicamente
50MHz)
    input reset,                 // Reset global (ativo alto)
    input start,                 // Botão para iniciar o jogo
    input [3:0] botoes,          // 4 botões para as jogadas
    input dificuldade,          // Seletor de nível de dificuldade

    // Saídas para matriz de peça (linhas e colunas individuais)
    output peca_linha0,
    output peca_linha1,
    output peca_linha2,
    output peca_linha3,
    output peca_linha4,
    output peca_linha5,
    output peca_linha6,
    output peca_linha7,
    output peca_coluna0,
    output peca_coluna1,
    output peca_coluna2,
    output peca_coluna3,
    output peca_coluna4,
    output peca_coluna5,
    output peca_coluna6,
    output peca_coluna7,

    // Saídas para matriz de mapa (linhas e colunas individuais)
    output mapa_linha0,
    output mapa_linha1,
    output mapa_linha2,
    output mapa_linha3,
    output mapa_linha4,
    output mapa_linha5,
    output mapa_linha6,
    output mapa_linha7,
    output mapa_coluna0,
    output mapa_coluna1,
    output mapa_coluna2,
    output mapa_coluna3,
    output mapa_coluna4,
    output mapa_coluna5,
    output mapa_coluna6,
```

```

        output mapa_coluna7,
        // Saída para som (opcional)
        output speaker,                                // Saída para
speaker/buzzer

        // Saídas de depuração
        output [3:0] db_estado,                         // Estado atual da UC
        output [3:0] db_contagem,                       // Contador atual
        output [3:0] db_jogada,                          // Jogada atual
        output [3:0] db_memoria                         // Valor da memória
    );

        // Sinais entre UC e FD
    wire s_zera_contador;
    wire sConta_contador;
    wire s_enable_memoria;
    wire s_registra_jogada;
    wire s_compara_jogada;
    wire s_timer_restart;
    wire s_tem_jogada;
    wire s_jogada_ok;
    wire s_fim_sequencia;
    wire s_timeout;
    wire s_game_over;

        // Sinais para matrizes de LED
    wire [7:0] s_padrao_peca;
    wire [7:0] s_padrao_mapa;

        // Sinais para exibição nas matrizes
    wire [7:0] s_peca_linhas [0:7];
    wire [7:0] s_mapa_linhas [0:7];

        // Instanciação da UNIDADE DE CONTROLE
    teatris_uc unidade_controle (
        .clock(clock_50MHz),
        .reset(reset),
        .timeout(s_timeout),
        .start(start),
        .dificuldade(dificuldade),
        .fim_sequencia(s_fim_sequencia),
        .tem_jogada(s_tem_jogada),

```

```

.jogada_ok(s_jogada_ok),
.zera_contador(s_zera_contador),
.conta_contador(sConta_contador),
.enable_memoria(s_enable_memoria),
.registra_jogada(s_registra_jogada),
.compara_jogada(s_compara_jogada),
.timer_restart(s_timer_restart),
.game_over(s_game_over),
.db_estado(db_estado)
);

// Instanciação do FLUXO DE DADOS
teatris_fluxo_dados fluxo_dados (
.clock(clock_50MHz),
.reset(reset),
.botoes(botoes),

.zera_contador(s_zera_contador),
.conta_contador(sConta_contador),
.enable_memoria(s_enable_memoria),
.registra_jogada(s_registra_jogada),
.compara_jogada(s_compara_jogada),
.timer_restart(s_timer_restart),

.tem_jogada(s_tem_jogada),
.jogada_ok(s_jogada_ok),
.fim_sequencia(s_fim_sequencia),
.timeout(s_timeout),

.padrao_peca(s_padrao_peca),
.padrao_mapa(s_padrao_mapa),

.db_contagem(db_contagem),
.db_jogada(db_jogada),
.db_memoria(db_memoria)
);

// Configuração dos padrões para as matrizes de LEDs
// Configuração para matriz de peça (centralizada nas linhas
2 e 3)
assign s_peca_linhas[0] = 8'b00000000;
assign s_peca_linhas[1] = 8'b00000000;

```

```

assign s_peca_linhas[2] = s_padrao_peca;
assign s_peca_linhas[3] = s_padrao_peca;
assign s_peca_linhas[4] = 8'b00000000;
assign s_peca_linhas[5] = 8'b00000000;
assign s_peca_linhas[6] = 8'b00000000;
assign s_peca_linhas[7] = 8'b00000000;

// Configuração para matriz de mapa (padrão na linha 3)
assign s_mapa_linhas[0] = 8'b11111111;
assign s_mapa_linhas[1] = 8'b11111111;
assign s_mapa_linhas[2] = 8'b11111111;
assign s_mapa_linhas[3] = s_padrao_mapa;
assign s_mapa_linhas[4] = 8'b11111111;
assign s_mapa_linhas[5] = 8'b11111111;
assign s_mapa_linhas[6] = 8'b11111111;
assign s_mapa_linhas[7] = 8'b11111111;

// Instanciação dos DRIVERS PARA MATRIZES 2088BS com pinos
individuais
driver_matrix_2088bs driver_matrix_peca (
    .clock(clock_50MHz),
    .reset(reset),
    .padrao_linha0(s_peca_linhas[0]),
    .padrao_linha1(s_peca_linhas[1]),
    .padrao_linha2(s_peca_linhas[2]),
    .padrao_linha3(s_peca_linhas[3]),
    .padrao_linha4(s_peca_linhas[4]),
    .padrao_linha5(s_peca_linhas[5]),
    .padrao_linha6(s_peca_linhas[6]),
    .padrao_linha7(s_peca_linhas[7]),

    // Linhas para matriz de peça
    .linha0(peca_linha0),
    .linha1(peca_linha1),
    .linha2(peca_linha2),
    .linha3(peca_linha3),
    .linha4(peca_linha4),
    .linha5(peca_linha5),
    .linha6(peca_linha6),
    .linha7(peca_linha7),

    // Colunas para matriz de peça
    .coluna0(peca_coluna0),

```

```

.coluna1(peca_coluna1),
.coluna2(peca_coluna2),
.coluna3(peca_coluna3),
.coluna4(peca_coluna4),
.coluna5(peca_coluna5),
.coluna6(peca_coluna6),
.coluna7(peca_coluna7)
);

driver_matriz_2088bs driver_matriz_mapa (
.clock(clock_50MHz),
.reset(reset),
.padrao_linha0(s_mapa_linhas[0]),
.padrao_linha1(s_mapa_linhas[1]),
.padrao_linha2(s_mapa_linhas[2]),
.padrao_linha3(s_mapa_linhas[3]),
.padrao_linha4(s_mapa_linhas[4]),
.padrao_linha5(s_mapa_linhas[5]),
.padrao_linha6(s_mapa_linhas[6]),
.padrao_linha7(s_mapa_linhas[7]),

// Linhas para matriz de mapa
.linha0(mapa_linha0),
.linha1(mapa_linha1),
.linha2(mapa_linha2),
.linha3(mapa_linha3),
.linha4(mapa_linha4),
.linha5(mapa_linha5),
.linha6(mapa_linha6),
.linha7(mapa_linha7),

// Colunas para matriz de mapa
.coluna0(mapa_coluna0),
.coluna1(mapa_coluna1),
.coluna2(mapa_coluna2),
.coluna3(mapa_coluna3),
.coluna4(mapa_coluna4),
.coluna5(mapa_coluna5),
.coluna6(mapa_coluna6),
.coluna7(mapa_coluna7)
);

// Gerador de som simplificado

```

```

    reg [15:0] contador_som;
    always @ (posedge clock_50MHz) begin
        if (reset)
            contador_som <= 0;
        else
            contador_som <= contador_som + 1;
    end

    // Som ativo durante game_over (pode ser modificado conforme
necessário)
    assign speaker = s_game_over ? contador_som[14] : 1'b0;

endmodule

```

8.4. Layout Proposto do Dispositivo

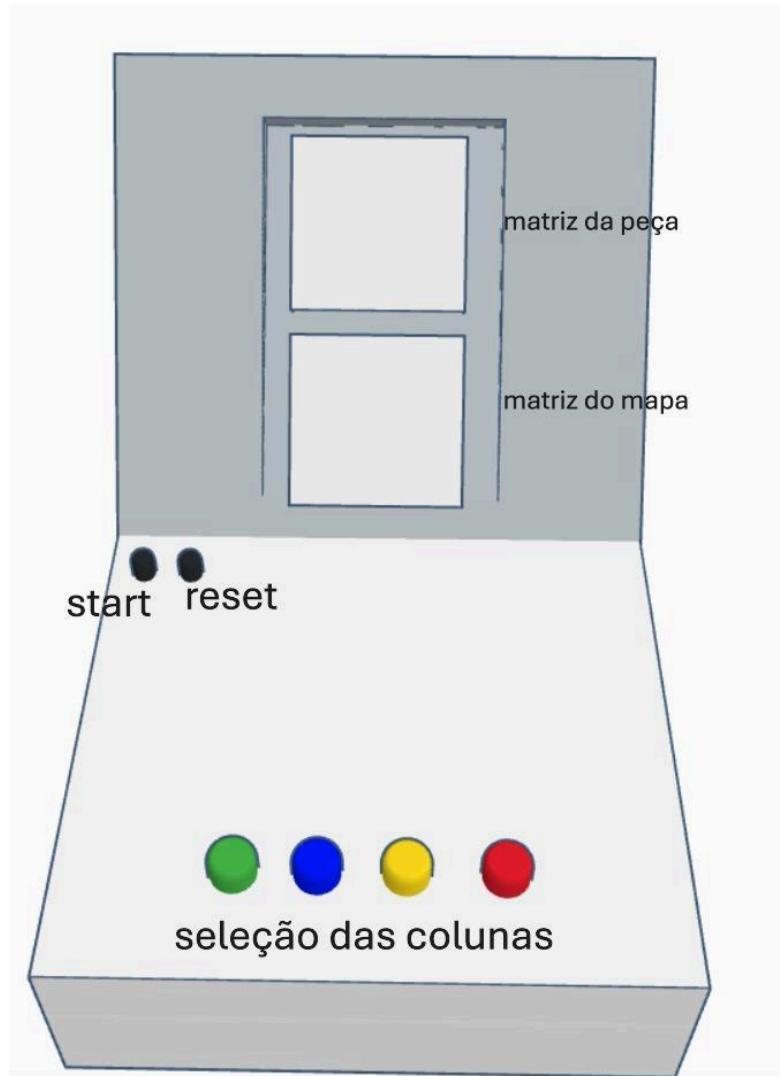


figura 24: Vista superior do jogo

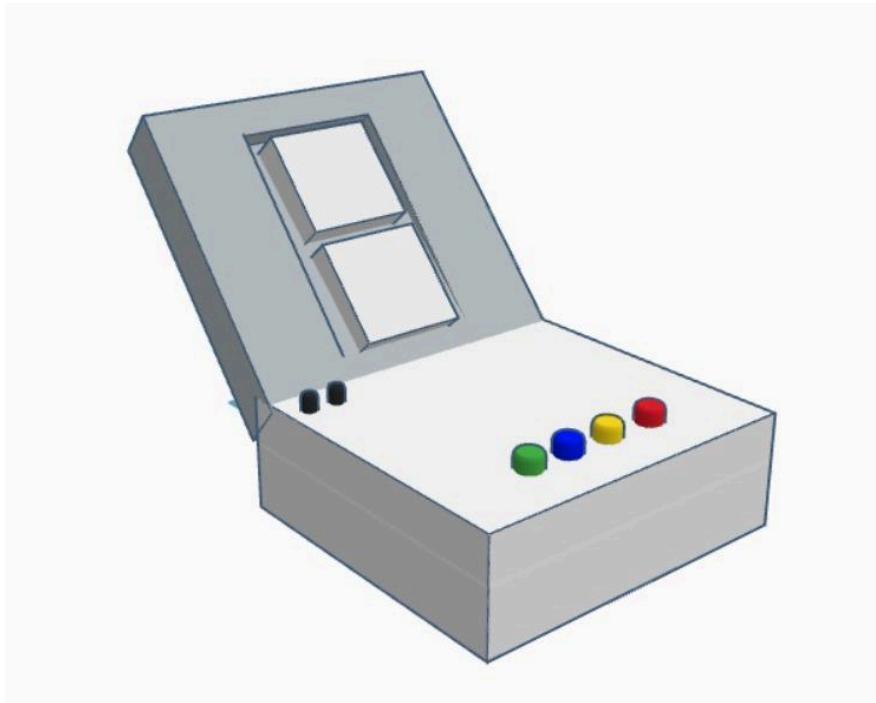


Figura 25: Visão isométrica do protótipo

8.5. Componentes Adquiridos

Na semana 2 foram adquiridos os botões (*push buttons*) e capas. Ainda nessa mesma semana foram comprados 50 resistores de 100 Ohms e 3 *protoboards* de 830 pontos (1 de reserva), que possuem previsão de chegada para o início da semana 3.

8.6. Componentes Testados

Nesta semana, o grupo conseguiu testar as duas matrizes de led simultaneamente. Foi exibida uma peça modelo na matriz superior e um tabuleiro na matriz inferior com o buraco para o respectivo encaixe. Abaixo, segue uma ilustração dos testes.

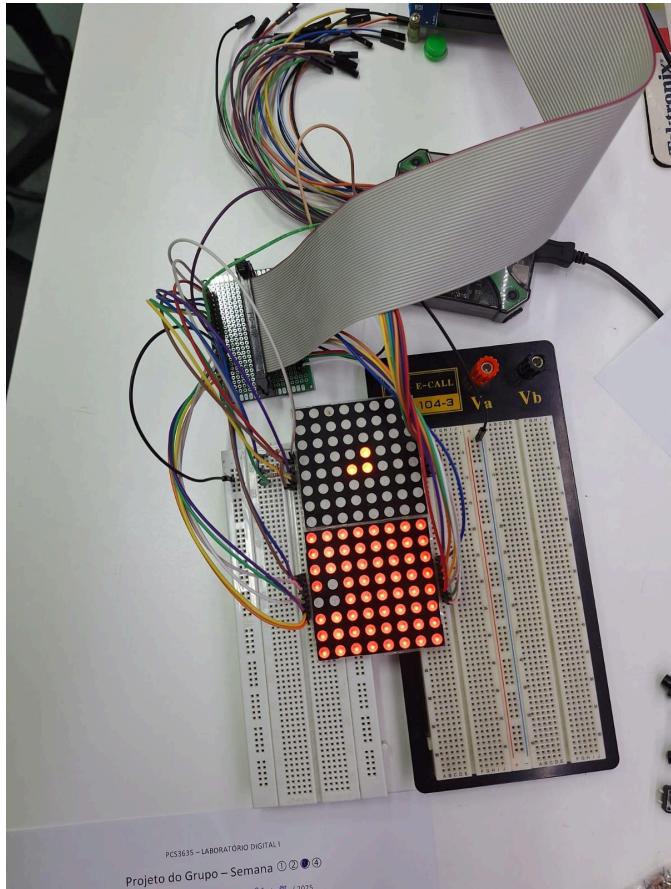


Figura 26: Teste realizado em sala da semana 3

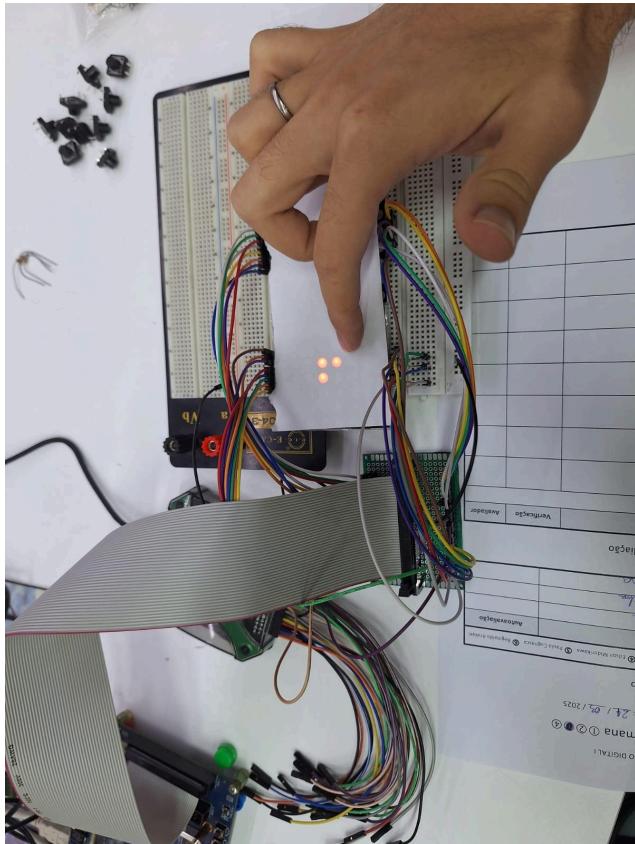


Figura 27: teste de exibição da matriz com tela

Dessa forma, o grupo planeja terminar a lógica de encaixe e o projeto funcional do jogo ainda na semana 3.

9. SEMANA 4

9.1. Objetivos da semana

Na última semana, o grupo conseguiu finalizar a lógica de encaixe, o que exigiu algumas alterações na Unidade de controle. Durante os testes, foi constatado que o uso temporário dos botões da *FPGA* não seria viável, pois eles não estavam sendo acionados corretamente, mesmo após serem pressionados. Ao substituir pelos botões do *Analog Discovery*, o funcionamento ocorreu como esperado.

Como não é possível incluir vídeos aqui, serão comentados apenas os itens que já foram realizados e o que ainda falta ser feito. Os requisitos 2 (*Timeout*), 4 (Atualização do *Display*), 5 (Memória de Mapas) e 6 (Interface de *LEDs*) já foram

implementados com sucesso. Restam apenas os requisitos 1 (Seletor de Nível) e 3 (Contador de Erros), que podem ser concluídos rapidamente. Caso seja necessário eliminar algum deles, o Seletor de nível é que tem menor prioridade, bastando, nesse caso, definir um tempo fixo para a espera das jogadas.

A maquete do projeto ainda não começou a ser construída, pois o grupo precisa buscar os materiais necessários. Também está nos planos adicionar animações para os seguintes casos: cenário de acerto de todas as jogadas, cenário com acertos e erros, e cenário inicial do jogo antes do acionamento do botão *start*.

9.2. Requisitos da semana

Nesta semana, o grupo visa terminar todos os requisitos que ainda não foram cumpridos, a saber, o requisito 1 (Seletor de Nível) e o requisito 3 (Contador de Erros). Além disso, o grupo percebeu que, caso haja tempo disponível, poderá fazer mudanças que tornarão o jogo mais lúdico ao usuário, como por exemplo as animações de fim de jogo e de acerto de jogadas. Essas animações serão de desenhos que aparecerão após o usuário acertar todas as jogadas (um sorriso) e de leds da matriz piscando após o jogador acertar uma única jogada. É possível também fazer, se houver disponibilidade de tempo, sinais sonoros durante esses acertos com o uso de “buzzers”.

9.3. Projeto lógico

Em relação à semana anterior, adicionamos novos estados na unidade de controle e novos componentes no fluxo de dados, com o intuito de cumprir os requisitos de encaixe e de atualização de display. Tais componentes serão discutidos nos próximos tópicos.

9.3.1. Fluxo de dados

O grupo introduziu no Fluxo de Dados uma memória de jogadas certas e uma memória de colunas corretas para que, assim como no projeto do Genius, pudesse comparar a todo momento, quando a jogada está certa ou errada. Além disso, o intuito dessa semana é de adicionar mais um contador que armazene o número de erros para que a evolução cognitiva do jogador possa ser monitorada a cada partida.

```
module teatris_fluxo_dados (

    // Entradas básicas

    input clock,                      // Clock do sistema
    input reset,                       // Sinal de reset
    input [3:0] botoes,                // Botões de entrada

    // Sinais de controle da UC

    input zera_contador,              // Zera o contador de endereços
    input conta_contador,             // Incrementa o contador
    input enable_memoria,             // Habilita leitura/registro da
memória

    input registra_jogada,            // Registra a jogada atual
    input zera_jogada,
    input timer_restart,               // Reinicia o temporizador
    input timer_animacao_restart,
```

```

    input conta_timer_animacao,
         

    // Sinais de estado para a UC

        output tem_jogada,           // Indica que um botão foi
pressionado

        output jogada_ok,           // Indica que a jogada está
correta

        output fim_sequencia,       // Indica fim da sequência atual

        output timeout,             // Indica que o tempo se esgotou

        output fim_timer_animacao,


    // Saídas para exibição

        output [15:0] padrao_peca,   // Padrão para matriz de peça

        output [63:0] padrao_mapa,   // Padrão para matriz de mapa

        output [63:0] padrao_coluna, // Padrão para matriz de
colunas certas


    // Saídas de depuração

        output [3:0] db_contagem,    // Valor atual do contador

        output [3:0] db_jogada,      // Jogada atual registrada

        output [3:0] db_memoria,     // Valor esperado da memória

        output db_tem_jogada


);

// Sinais internos

wire [3:0] s_endereco;           // Endereço atual da memória

```

```

    wire [15:0] s_padrao_peca;      // Padrão de peça da memória

    wire [63:0] s_padrao_mapa;      // Padrão de mapa da memória

    wire [63:0] s_padrao_coluna;

    wire [3:0] s_jogada_correta;   // Jogada correta para o padrão
atual

    wire s_pulso_jogada;

    wire [3:0] s_jogada_atual;     // Jogada atual registrada

    reg [15:0] r_padrao_peca;      // Registrador para padrão de
peça

    reg [63:0] r_padrao_mapa;      // Registrador para padrão de
mapa

    reg [63:0] r_padrao_coluna;    // Registrador para padrão de
colunas corretas

edge_detector detector_jogada (
    .clock(clock),
    .reset(~botoes[3] & ~botoes[2] & ~botoes[1] & ~botoes[0]),
    .sinal(botoes[3] | batoes[2] | batoes[1] | batoes[0]),
    .pulso(s_pulso_jogada)
);

// Contador de endereços para acessar a memória

contador_163 contador_endereco (
    .clock(clock),
    .clr(~zera_contador),
    .ld(1'b1),                  // Load ativo alto
    .ent(1'b1),                  // Enable de contagem
    .enp(conta_contador),        // Enable de incremento
);

```

```
        .D(4'b0000),           // Valor inicial
        .Q(s_endereco),       // Saída do contador
        .rco(fim_sequencia)   // Ripple Carry Out como fim de
sequência
    );
}

// Memórias ROM

teatris_memoria_pecas memoria_pecas (
    .clock(clock),
    .endereco(s_endereco),
    .padrao(s_padrao_peca)
);

teatris_memoria_mapas memoria_mapas (
    .clock(clock),
    .endereco(s_endereco),
    .padrao(s_padrao_mapa)
);

teatris_memoria_jogadas memoria_jogadas (
    .clock(clock),
    .endereco(s_endereco),
    .jogada(s_jogada_correta)
);
```

```

teatris_memoria_colunas_certas memoria_colunas_certas(
    .clock(clock),
    .endereco(s_endereco),
    .coluna(s_padrao_coluna)
);

registrador_4 r_jogada(
    .clock(clock),
    .clear(zera_jogada),
    .enable(registra_jogada),
    .D(botoes),
    .Q(s_jogada_atual)
);

// Registradores para armazenar padrões atuais

always @(posedge clock or posedge reset) begin

    if (reset) begin

        r_padrao_peca <= 64'b00000000;
        r_padrao_mapa <= 64'b00000000;
        r_padrao_coluna <= 64'b0;

    end

    else if (enable_memoria) begin

        r_padrao_peca <= s_padrao_peca;
        r_padrao_mapa <= s_padrao_mapa;
        r_padrao_coluna <= s_padrao_coluna;

    end
end

```

```

    end

end


// Comparador para verificar jogada

comparador_85 comparador_jogada (
    .A(s_jogada_atual),           // Jogada registrada
    .B(s_jogada_correta),         // Jogada esperada da memória
    .ALBi(1'b0),
    .AGBi(1'b0),
    .AEBi(1'b1),
    .ALBo(),
    .AGBo(),
    .AEBo(jogada_ok)            // Saída: jogada correta ou não
);

// Temporizador para timeout

contador_m #( .M(100_000_000), .N(27)) temporizador (
    .clock(clock),
    .zera_as(reset),
    .zera_s(timer_restart || s_pulso_jogada),
    .conta(1'b1),                // Sempre contando
    .Q(),                         // Valor do contador (não usado)
    .fim(timeout)                 // Timeout quando chega ao fim
);

// Temporizador para tempo de animacao

```

```
contador_m #( .M(100_000_000) , .N(27)) temporizador_animacao (
    .clock(clock),
    .zera_as(reset),
    .zera_s(timer_animacao_restart),
    .conta(conta_timer_animacao),
    .Q(),
    .fim(fim_timer_animacao)
);

// Atribuição das saídas

assign padrao_peca = r_padrao_peca;
assign padrao_mapa = r_padrao_mapa;
assign padrao_coluna = r_padrao_coluna;

// Saídas de depuração

assign db_contagem = s_endereco;
assign db_jogada = s_jogada_atual;
assign db_memoria = s_jogada_correta;
assign tem_jogada = s_pulso_jogada;
assign db_tem_jogada = botoes[3] | botoes[2] | botoes[1] | 
botoes[0];

endmodule
```

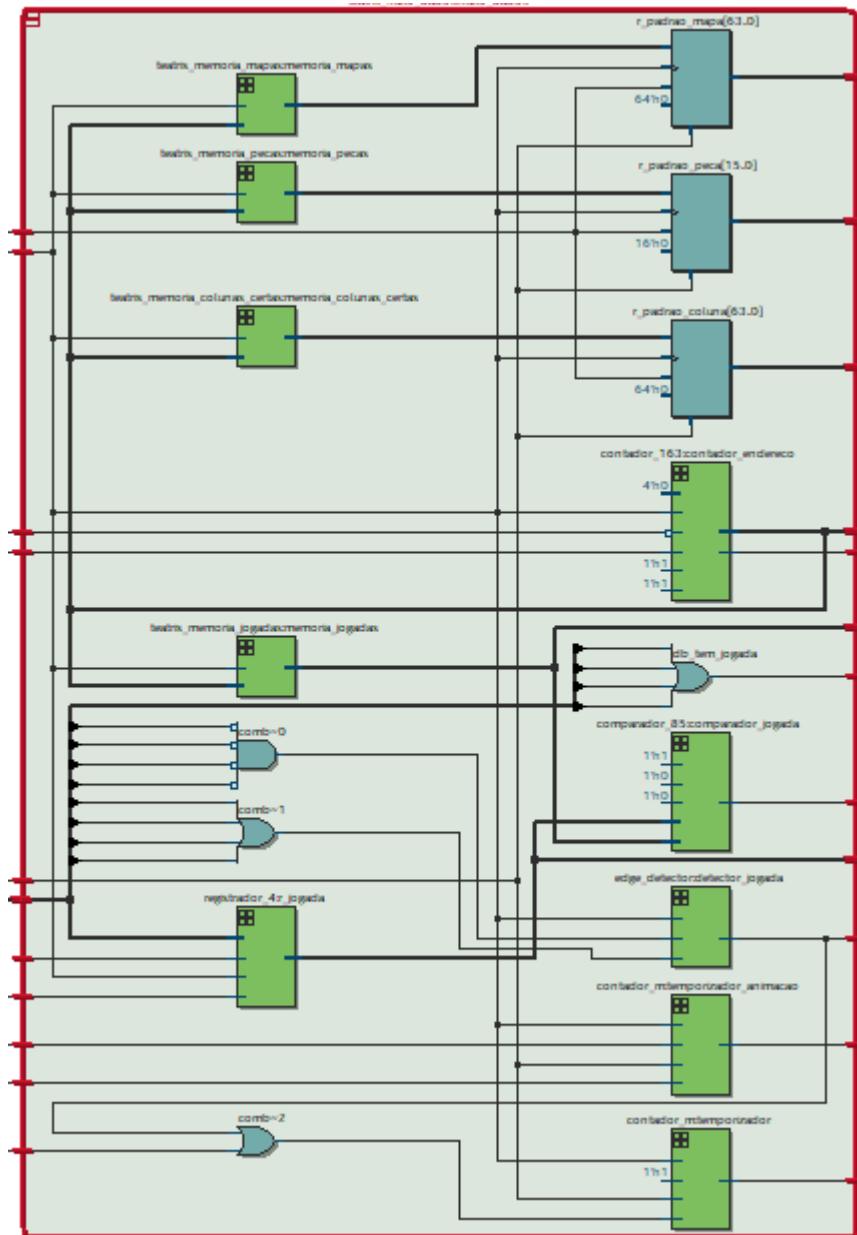


Figura 29: *RTL viewer* do Fluxo de Dados

9.3.2. Unidade de Controle

Perceba que foram adicionados mais estados, quando comparamos com a Unidade de Controle da semana 3. Esses estados são responsáveis pela transição das animações na hora do acerto de uma jogada.



Figura 28: Diagrama de Transição de Estados atualizado

```
module TEAtris_Control_Unit (
    input clock,           // Clock do sistema
    input reset,            // Sinal de reset
    input timeout,          // Sinal de timeout do temporizador
    input start,             // Sinal para iniciar jogo
    input dificuldade,       // Nível de dificuldade
    ...);
```

```




















```

);

// Estados do jogo

```

localparam INICIAL          = 4'b0000;
localparam CONFIGURA_JOGO  = 4'b0001;

```

```

localparam CARREGA_DADO      = 4'b0010;
localparam MOSTRA_DADO       = 4'b0011;
localparam ESPERA_JOGADA     = 4'b0100;
localparam TIMEOUT           = 4'b0101;
localparam REGISTRA          = 4'b0110;
localparam COMPARA            = 4'b0111;
localparam ANALISA_SEQUENCIA = 4'b1000;
localparam PROXIMA_SEQUENCIA = 4'b1001;
localparam FIM                = 4'b1010;
localparam ERRO_JOGADA        = 4'b1011;
localparam ACERTA_JOGADA      = 4'b1100;
localparam MOSTRA_ACERTO      = 4'b1101;
localparam MOSTRA_ORIGINAL   = 4'b1110;

reg [3:0] Eatual, Eprox;

// Memória de Estado

always @(posedge clock or posedge reset) begin
    if (reset)
        Eatual <= INICIAL;
    else
        Eatual <= Eprox;
end

```

```
// Lógica para determinar próximo estado

always @(*) begin

    case (Eatual)

        INICIAL:

            Eprox = start ? CONFIGURA_JOGO : INICIAL;

        CONFIGURA_JOGO:

            Eprox = CARREGA_DADO;

        CARREGA_DADO:

            Eprox = MOSTRA_DADO;

        MOSTRA_DADO:

            Eprox = ESPERA_JOGADA;

        ESPERA_JOGADA:

            Eprox = timeout ? TIMEOUT : tem_jogada ? REGISTRA
: ESPERA_JOGADA;

        TIMEOUT:

            Eprox = start ? INICIAL : TIMEOUT;

        REGISTRA:

            Eprox = COMPARA;
```

COMPARA:

```
Eprox = jogada_ok ? ACERTA_JOGADA : ERRO_JOGADA;
```

ACERTA_JOGADA:

```
Eprox = MOSTRA_ACERTO;
```

MOSTRA_ACERTO:

```
Eprox = MOSTRA_ORIGINAL;
```

MOSTRA_ORIGINAL:

```
Eprox = ANALISA_SEQUENCIA;
```

ANALISA_SEQUENCIA:

```
Eprox = fim_sequencia ? PROXIMA_SEQUENCIA :  
CARREGA_DADO;
```

PROXIMA_SEQUENCIA:

```
Eprox = (dificuldade) ? FIM : CARREGA_DADO;
```

ERRO_JOGADA:

```
Eprox = start ? INICIAL : ERRO_JOGADA;
```

FIM:

```
Eprox = start ? INICIAL : FIM;
```

default:

```
Eprox = INICIAL;
```

endcase

end

```

    // Lógica de saída (máquina Moore - saídas dependem apenas do
estado atual)

    always @(*) begin

        // Valores default

        zera_contador = 1'b0;
        conta_contador = 1'b0;
        enable_memoria = 1'b0;
        registra_jogada = 1'b0;
        compara_jogada = 1'b0;
        timer_restart = 1'b0;
        game_over = 1'b0;

        db_estado = Eatual; // Sempre mostra o estado atual

        timer_animacao_restart = 1'b0;
        sel_mapa = 2'b00;
        conta_timer_animacao = 1'b0;

    end

    case (Eatual)

        INICIAL: begin

            zera_contador = 1'b1;
            timer_restart = 1'b1;
            zera_jogada = 1'b0;

        end

        CONFIGURA_JOGO: begin


```

```
    zera_contador = 1'b1;

    timer_restart = 1'b1;

    zera_jogada = 1'b0;

end

CARREGA_DADO: begin

    enable_memoria = 1'b1;

end

MOSTRA_DADO: begin

    enable_memoria = 1'b1;

    timer_restart = 1'b1;

end

ESPERA_JOGADA: begin

    // Aguarda entrada

end

ACERTA_JOGADA:begin

    sel_mapa = 2'b00;

    conta_timer_animacao = 1'b1;

end

MOSTRA_ACERTO:begin

    sel_mapa = 2'b01;

    timer_animacao_restart = 1'b1;

    conta_timer_animacao = 1'b1;

end
```

```
MOSTRA_ORIGINAL:begin

    // espera timer

    timer_animacao_restart = 1'b1;

end

TIMEOUT: begin

    game_over = 1'b1;

end

REGISTRA: begin

    registra_jogada = 1'b1;

end

COMPARA: begin

end

ANALISA_SEQUENCIA: begin

    conta_contador = 1'b1;

    timer_animacao_restart = 1'b1;

end

PROXIMA_SEQUENCIA: begin

    zera_contador = 1'b1;

    enable_memoria = 1'b1;

    timer_restart = 1'b1;

end

ERRO_JOGADA: begin

    game_over = 1'b1;

end
```

```
FIM: begin  
  
    game_over = 1'b1;  
  
end  
  
default: begin  
  
    zera_contador = 1'b1;  
  
end  
  
endcase  
  
end  
  
endmodule
```

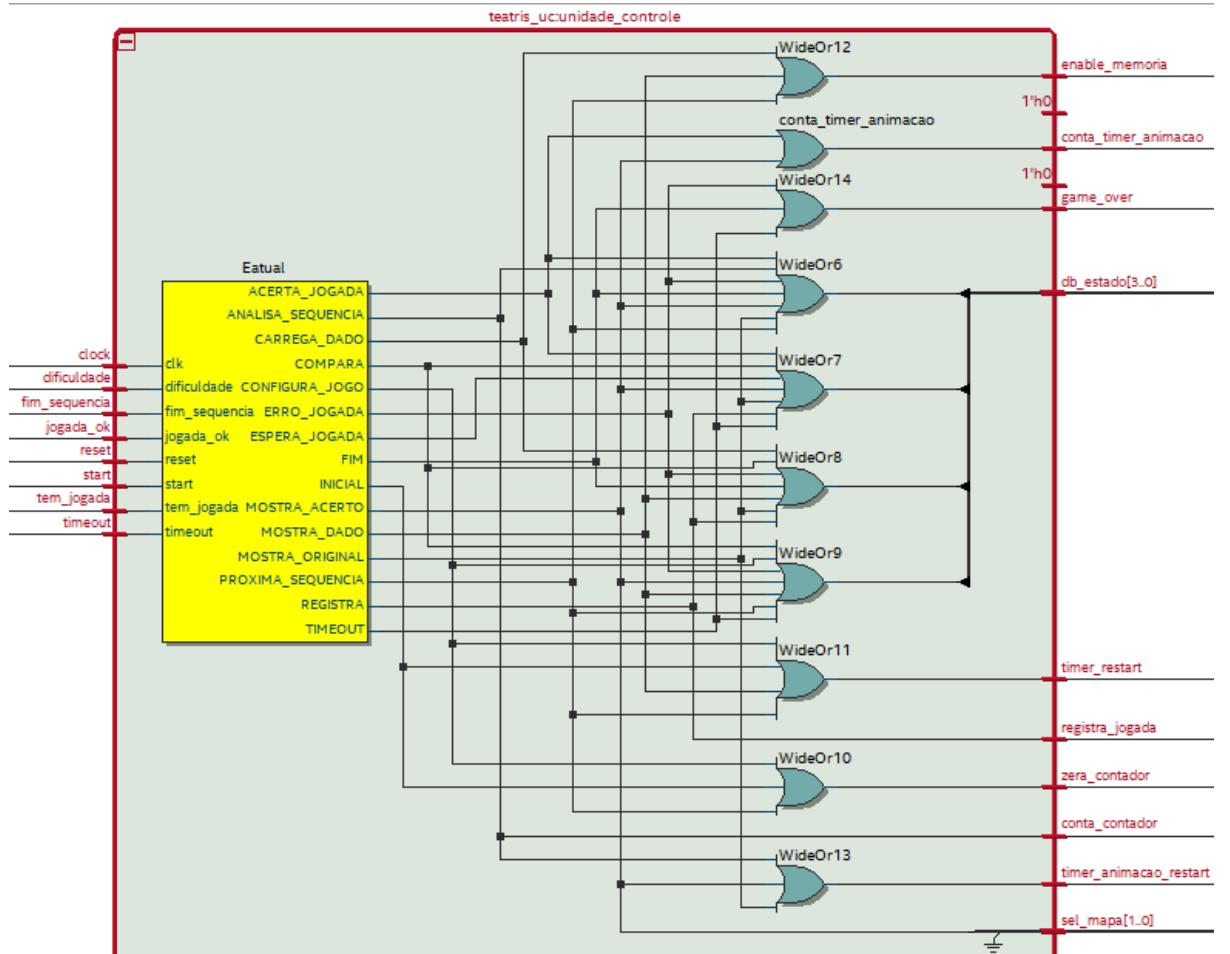


Figura 29: *RTL viewer* da Unidade de Controle

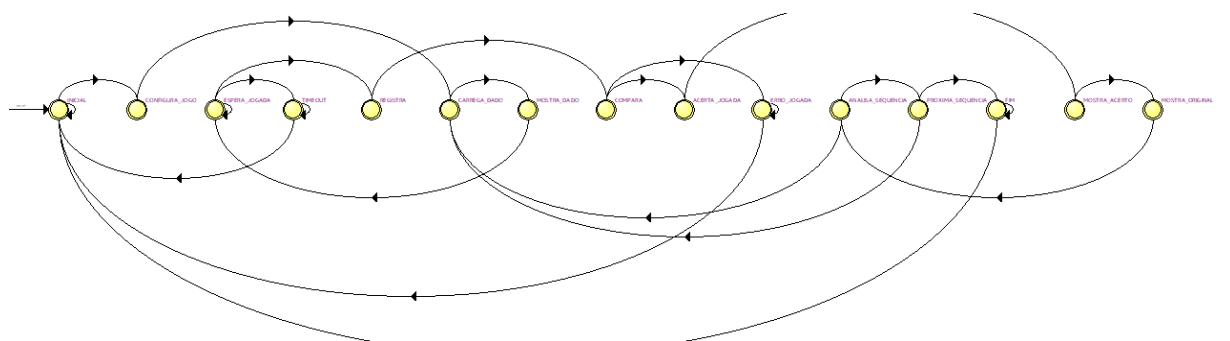


Figura 30: *State Machine Viewer* da Unidade de Controle

	Source State	Destination State	Condition
1	ACERTA_JOGADA	MOSTRA_ACERTO	
2	ANALISA_SEQUENCIA	PROXIMA_SEQUENCIA	(fim_sequencia)
3	ANALISA_SEQUENCIA	CARREGA_DADO	(!fim_sequencia)
4	CARREGA_DADO	MOSTRA_DADO	
5	COMPARA	ERRO_JOGADA	(!jogada_ok)
6	COMPARA	ACERTA_JOGADA	(jogada_ok)
7	CONFIGURA_JOGO	CARREGA_DADO	
8	ERRO_JOGADA	ERRO_JOGADA	(!start)
9	ERRO_JOGADA	INICIAL	(start)
10	ESPERA_JOGADA	TIMEOUT	(timeout)
11	ESPERA_JOGADA	REGISTRA	(tem_jogada).(!timeout)
12	ESPERA_JOGADA	ESPERA_JOGADA	(!tem_jogada).(!timeout)
13	FIM	FIM	(!start)
14	FIM	INICIAL	(start)
15	INICIAL	CONFIGURA_JOGO	(start)
16	INICIAL	INICIAL	(!start)
17	MOSTRA_ACERTO	MOSTRA_ORIGINAL	
18	MOSTRA_DADO	ESPERA_JOGADA	

18	MOSTRA_DADO	ESPERA_JOGADA	
19	MOSTRA_ORIGINAL	ANALISA_SEQUENCIA	
20	PROXIMA_SEQUEN...	FIM	(dificuldade)
21	PROXIMA_SEQUEN...	CARREGA_DADO	(!dificuldade)
22	REGISTRA	COMPARA	
23	TIMEOUT	INICIAL	(start)
24	TIMEOUT	TIMEOUT	(!start)

Figuras 31: Tabela de Transição de Estados

9.3.3. Circuito digital

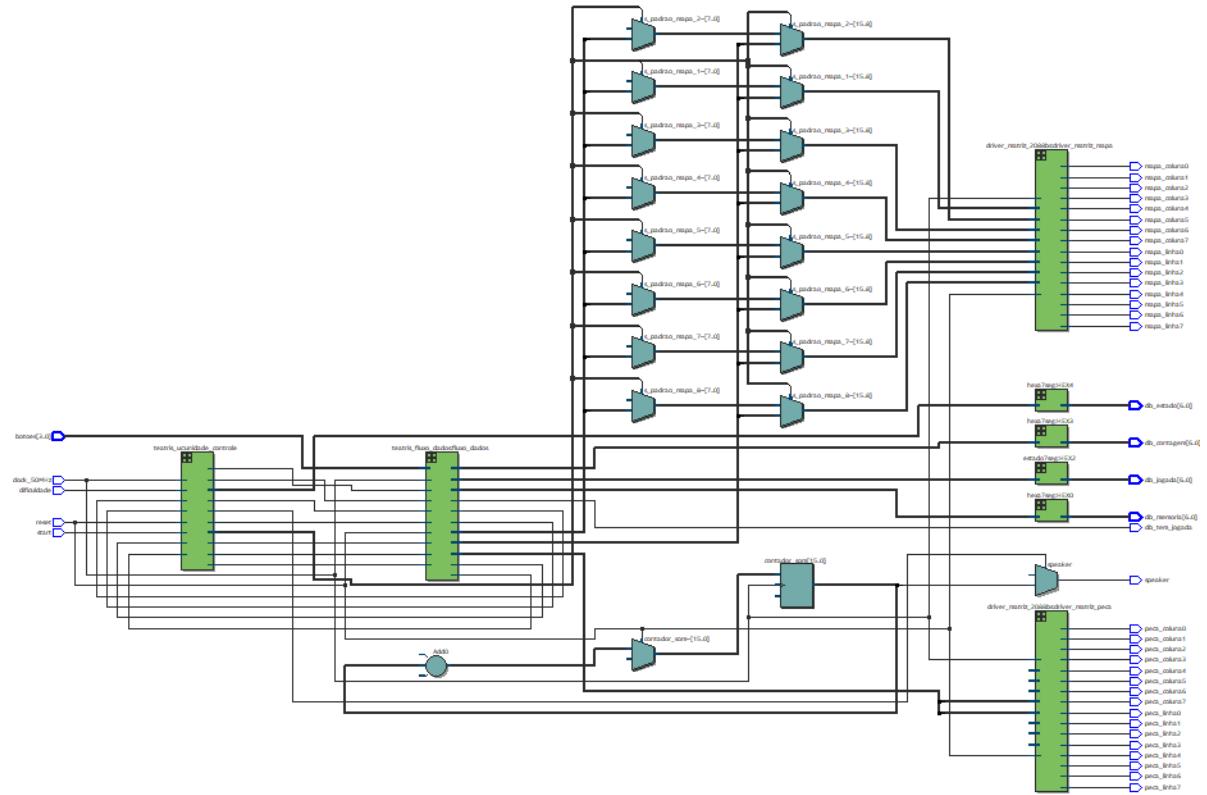


Figura 32: *RTL Viewer Top Level*

```
module TEAtris_Top (
    // Entradas principais
    input clock_50MHz,           // Clock principal (tipicamente
50MHz)
    input reset,                // Reset global (ativo alto)
    input start,                // Botão para iniciar o jogo
    input [3:0] botoes,          // 4 botões para as jogadas
    input dificuldade,          // Seletor de nível de dificuldade

    // Saídas para matriz de peça (linhas e colunas individuais)
    output peca_linha0,
    output peca_linha1,
```

```
    output peca_linha2,
    output peca_linha3,
    output peca_linha4,
    output peca_linha5,
    output peca_linha6,
    output peca_linha7,
    output peca_coluna0,
    output peca_coluna1,
    output peca_coluna2,
    output peca_coluna3,
    output peca_coluna4,
    output peca_coluna5,
    output peca_coluna6,
    output peca_coluna7,
    // Saídas para matriz de mapa (linhas e colunas individuais)
    output mapa_linha0,
    output mapa_linha1,
    output mapa_linha2,
    output mapa_linha3,
    output mapa_linha4,
    output mapa_linha5,
    output mapa_linha6,
    output mapa_linha7,
    output mapa_coluna0,
```

```

    output mapa_coluna1,
    output mapa_coluna2,
    output mapa_coluna3,
    output mapa_coluna4,
    output mapa_coluna5,
    output mapa_coluna6,
    output mapa_coluna7,

    // Saída para som (opcional)

    output speaker,                                // Saída para
speaker/buzzer

    // Saídas de depuração

    output [6:0] db_estado,                         // Estado atual da UC
    output [6:0] db_contagem,                       // Contador atual
    output [6:0] db_jogada,                          // Jogada atual
    output [6:0] db_memoria,                         // Valor da memória
    output db_tem_jogada

);

// Sinais entre UC e FD

wire s_zera_contador;
wire sContaContador;
wire s_enable_memoria;
wire s_RegistraJogada;

```

```
wire s_compara_jogada;
wire s_timer_restart;
wire s_timer_animacao_restart;
wire s_conta_timer_animacao;
wire s_tem_jogada;
wire s_jogada_ok;
wire s_fim_sequencia;
wire s_timeout;
wire s_game_over;
wire [1:0] sel_mapa;
wire [3:0] db_jogada_fio;
wire [3:0] db_estado_fio;
wire [3:0] db_memoria_fio;
wire [3:0] db_contagem_fio;

// Sinais para matrizes de LED
wire [15:0] s_padrao_peca;
wire [7:0] s_padrao_peca_sup;
wire [7:0] s_padrao_peca_inf;
wire [63:0] s_padrao_mapa;
wire [7:0] s_padrao_mapa_1;
wire [7:0] s_padrao_mapa_2;
wire [7:0] s_padrao_mapa_3;
wire [7:0] s_padrao_mapa_4;
```

```
    wire [7:0] s_padrao_mapa_5;

    wire [7:0] s_padrao_mapa_6;

    wire [7:0] s_padrao_mapa_7;

    wire [7:0] s_padrao_mapa_8;

    wire [63:0] s_padrao_coluna;

// Sinais para exibição nas matrizes

    wire [7:0] s_peca_linhas [0:7];

    wire [7:0] s_mapa_linhas [0:7];

// Instanciação da UNIDADE DE CONTROLE

teatris_uc unidade_controle (
    .clock(clock_50MHz),
    .reset(reset),
    .timeout(s_timeout),
    .start(start),
    .dificuldade(dificuldade),
    .fim_sequencia(s_fim_sequencia),
    .tem_jogada(s_tem_jogada),
    .jogada_ok(s_jogada_ok),
    .sel_mapa(sel_mapa),
    .zera_contador(s_zera_contador),
    .conta_contador(sContaContador),
```

```
.enable_memoria(s_enable_memoria) ,  
  
.registra_jogada(s_registra_jogada) ,  
  
.timer_restart(s_timer_restart) ,  
  
.timer_animacao_restart (s_timer_animacao_restart) ,  
  
.conta_timer_animacao (s_conta_timer_animacao) ,  
  
.game_over(s_game_over) ,  
  
.db_estado(db_estado_fio)  
);  
  
  
// Instanciação do FLUXO DE DADOS  
  
teatris_fluxo_dados fluxo_dados (   
  
.clock(clock_50MHz) ,  
  
.reset(reset) ,  
  
.botoes(botoes) ,  
  
.zera_contador(s_zera_contador) ,  
  
.conta_contador(s_conta_contador) ,  
  
.enable_memoria(s_enable_memoria) ,  
  
.registra_jogada(s_registra_jogada) ,  
  
.timer_restart(s_timer_restart) ,  
  
.timer_animacao_restart (s_timer_animacao_restart) ,  
  
.conta_timer_animacao (s_conta_timer_animacao) ,  
  
.tem_jogada(s_tem_jogada) ,  
  
.jogada_ok(s_jogada_ok) ,
```

```

    .fim_sequencia(s_fim_sequencia) ,
    .timeout(s_timeout) ,
    .padrao_peca(s_padrao_peca) ,
    .padrao_mapa(s_padrao_mapa) ,
    .padrao_coluna(s_padrao_coluna) ,
    .db_contagem(db_contagem_fio) ,
    .db_jogada(db_jogada_fio) ,
    .db_memoria(db_memoria_fio) ,
    .db_tem_jogada(db_tem_jogada)
);

// Configuração dos padrões para as matrizes de LEDs

assign s_padrao_peca_sup = s_padrao_peca [7:0];
assign s_padrao_peca_inf = s_padrao_peca [15:8];
assign s_padrao_mapa_8 = ~sel_mapa[0] ? s_padrao_mapa [7 : 0] : sel_mapa[1] ? s_padrao_coluna [7:0] : 8'b0;
assign s_padrao_mapa_7 = ~sel_mapa[0] ? s_padrao_mapa [15:8] : sel_mapa[1] ? s_padrao_coluna [15:8] : 8'b0;
assign s_padrao_mapa_6 = ~sel_mapa[0] ? s_padrao_mapa [23:16] : sel_mapa[1] ? s_padrao_coluna [23:16] : 8'b0;
assign s_padrao_mapa_5 = ~sel_mapa[0] ? s_padrao_mapa [31:24] : sel_mapa[1] ? s_padrao_coluna [31:24] : 8'b0;
assign s_padrao_mapa_4 = ~sel_mapa[0] ? s_padrao_mapa [39:32] : sel_mapa[1] ? s_padrao_coluna [39:32] : 8'b0;

```

```

        assign s_padrao_mapa_3 = ~sel_mapa[0] ? s_padrao_mapa
[47:40] : sel_mapa[1] ? s_padrao_coluna [47:40] : 8'b0;

        assign s_padrao_mapa_2 = ~sel_mapa[0] ? s_padrao_mapa
[55:48] : sel_mapa[1] ? s_padrao_coluna [55:48] : 8'b0;

        assign s_padrao_mapa_1 = ~sel_mapa[0] ? s_padrao_mapa
[63:56] : sel_mapa[1] ? s_padrao_coluna [63:56] : 8'b0;

// Configuração para matriz de peça (centralizada nas linhas
3 e 4)

assign s_peca_linhas[0] = 8'b11111111;
assign s_peca_linhas[1] = 8'b11111111;
assign s_peca_linhas[2] = 8'b11111111;
assign s_peca_linhas[3] = s_padrao_peca_sup;
assign s_peca_linhas[4] = s_padrao_peca_inf;
assign s_peca_linhas[5] = 8'b11111111;
assign s_peca_linhas[6] = 8'b11111111;
assign s_peca_linhas[7] = 8'b11111111;

// Configuração para matriz de mapa (padrão na linha 3)

assign s_mapa_linhas[0] = s_padrao_mapa_1;
assign s_mapa_linhas[1] = s_padrao_mapa_2;
assign s_mapa_linhas[2] = s_padrao_mapa_3;
assign s_mapa_linhas[3] = s_padrao_mapa_4;
assign s_mapa_linhas[4] = s_padrao_mapa_5;
assign s_mapa_linhas[5] = s_padrao_mapa_6;
assign s_mapa_linhas[6] = s_padrao_mapa_7;
assign s_mapa_linhas[7] = s_padrao_mapa_8;

```

```
// Instanciação dos DRIVERS PARA MATRIZES 2088BS com pinos  
individuais  
  
driver_matrix_2088bs driver_matrix_peca ( // Linhas para matriz de peça  
    .clock(clock_50MHz),  
    .reset(reset),  
    .padrao_linha0(s_peca_linhas[0]),  
    .padrao_linha1(s_peca_linhas[1]),  
    .padrao_linha2(s_peca_linhas[2]),  
    .padrao_linha3(s_peca_linhas[3]),  
    .padrao_linha4(s_peca_linhas[4]),  
    .padrao_linha5(s_peca_linhas[5]),  
    .padrao_linha6(s_peca_linhas[6]),  
    .padrao_linha7(s_peca_linhas[7]),  
);
```

```
// Colunas para matriz de peça

.coluna0(peca_coluna0) ,
.coluna1(peca_coluna1) ,
.coluna2(peca_coluna2) ,
.coluna3(peca_coluna3) ,
.coluna4(peca_coluna4) ,
.coluna5(peca_coluna5) ,
.coluna6(peca_coluna6) ,
.coluna7(peca_coluna7)

);

driver_matriz_2088bs driver_matriz_mapa (
    .clock(clock_50MHz) ,
    .reset(reset) ,
    .padrao_linha0(s_mapa_linhas[0]) ,
    .padrao_linha1(s_mapa_linhas[1]) ,
    .padrao_linha2(s_mapa_linhas[2]) ,
    .padrao_linha3(s_mapa_linhas[3]) ,
    .padrao_linha4(s_mapa_linhas[4]) ,
    .padrao_linha5(s_mapa_linhas[5]) ,
    .padrao_linha6(s_mapa_linhas[6]) ,
    .padrao_linha7(s_mapa_linhas[7]) ,

// Linhas para matriz de mapa

.linha0(mapa_linha0) ,
```

```
.linha1(mapa_linha1) ,  
  
.linha2(mapa_linha2) ,  
  
.linha3(mapa_linha3) ,  
  
.linha4(mapa_linha4) ,  
  
.linha5(mapa_linha5) ,  
  
.linha6(mapa_linha6) ,  
  
.linha7(mapa_linha7) ,  
  
  
// Colunas para matriz de mapa  
  
.coluna0(mapa_coluna0) ,  
  
.coluna1(mapa_coluna1) ,  
  
.coluna2(mapa_coluna2) ,  
  
.coluna3(mapa_coluna3) ,  
  
.coluna4(mapa_coluna4) ,  
  
.coluna5(mapa_coluna5) ,  
  
.coluna6(mapa_coluna6) ,  
  
.coluna7(mapa_coluna7)  
);  
  
  
estado7seg HEX2(  
  
.estado( db_jogada_fio ) ,  
  
.display( db_jogada)  
);  
  
  
hexa7seg HEX3(
```

```

        .hexa( db_contagem_fio ) ,
        .display( db_contagem )
    ) ;

hexa7seg HEX0(
    .hexa( db_memoria_fio ) ,
    .display( db_memoria )
) ;

hexa7seg HEX4(
    .hexa( db_estado_fio ) ,
    .display( db_estado )
) ;

// Gerador de som simplificado

reg [15:0] contador_som;

always @ (posedge clock_50MHz) begin
    if (reset)
        contador_som <= 0;
    else
        contador_som <= contador_som + 1;
end

```

```
// Som ativo durante game_over (pode ser modificado conforme  
necessário)  
  
assign speaker = s_game_over ? contador_som[14] : 1'b0;  
  
endmodule
```

9.4. Componentes adquiridos

O grupo planeja adquirir na semana 4 botões novos que se encaixem nas capinhas coloridas compradas na semana 2 e 1 buzzer para sinalização das jogadas erradas, caso haja tempo disponível. Vale destacar que já foram adquiridos, na semana 3, resistores de 100 ohms para utilização na matriz de led e nos botões, 3 protoboards e fios para otimização do espaço que o circuito ocupa.

10. RELATO SEMANA 4

No laboratório da última semana, o grupo conseguiu finalizar a maior parte das pendências, incluindo a integração do circuito com os botões adquiridos, a conclusão do Requisito 3 (Contador de Erros) e a adição de animações para os casos de acertos e erros durante as jogadas.

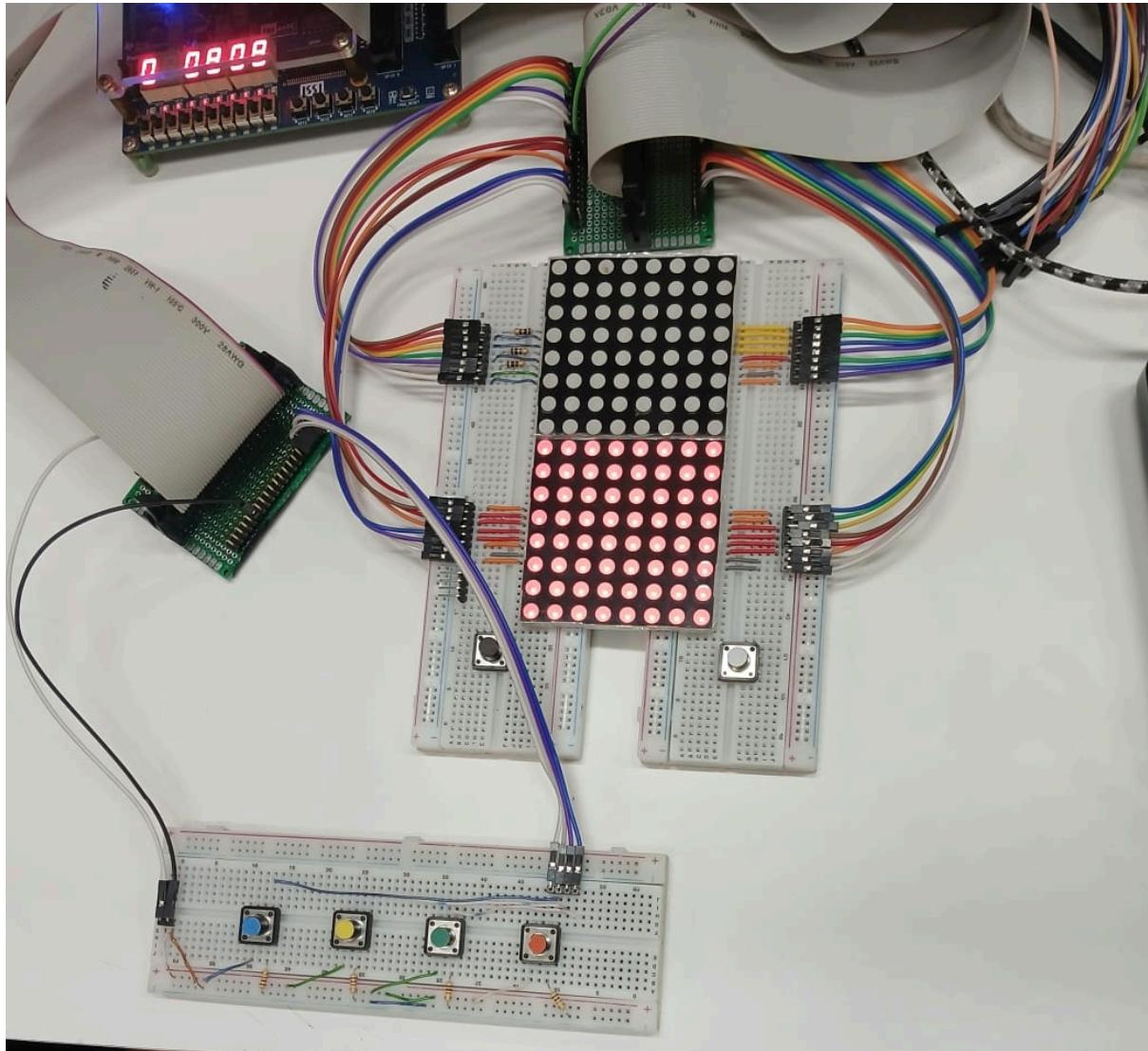


Figura 33: Montagem do circuito com os botões físicos e funcionais

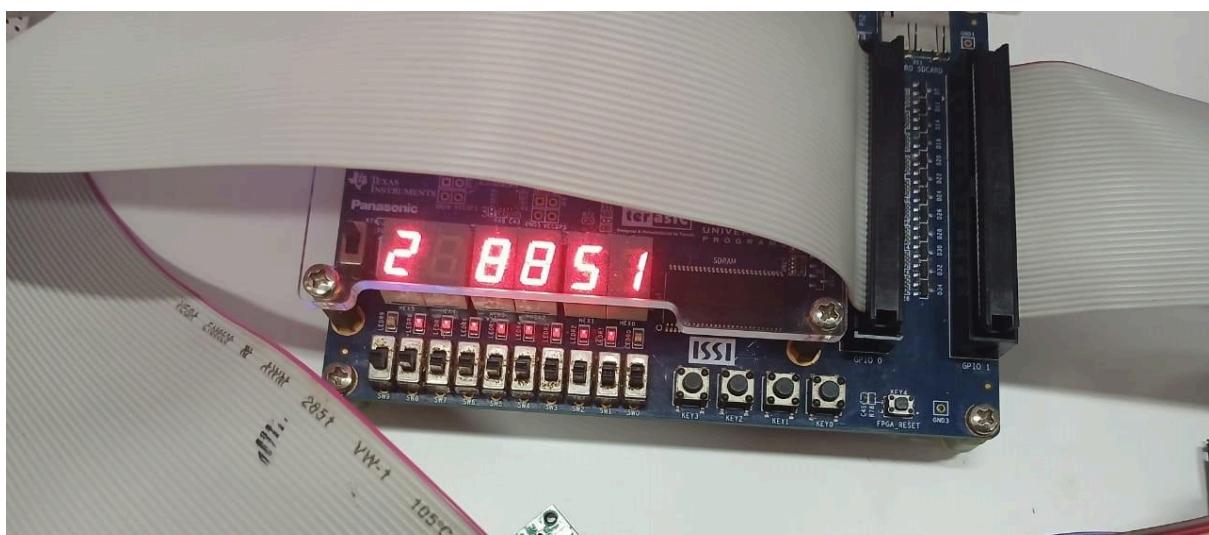


Figura 34: Sinais de depuração conectados na *FPGA*

Abaixo segue o link do *Google Drive* com os vídeos de demonstração de funcionamento do projeto.

[https://drive.google.com/drive/folders/1SMAHWOApo3qX_CyWhfU0r5izFDG5fPnL?
usp=sharing](https://drive.google.com/drive/folders/1SMAHWOApo3qX_CyWhfU0r5izFDG5fPnL?usp=sharing)

11. RELATO FINAL

Abaixo, segue a versão final da especificação do projeto apresentado pelo grupo no dia 07/04/2025.

Todos os requisitos (funcionais e não funcionais), a saber, Seletor de Nível, Timeout, Contador de Erros, Atualiza Display, Memória de Mapas, Controle de Leds, foram cumpridos. O grupo adicionou na última semana mais melhorias ao projeto, pois julgou que elas seriam úteis para melhorar a experiência do usuário com o Teatris. Entre essas melhorias, destacam-se: chave seletora de dificuldades, mapas com mais de um buraco para encaixe (no modo de dificuldade difícil), animação no início do jogo e para jogadas erradas (indicando a coluna correta que deveria ter sido escolhida).

Além disso, o grupo também finalizou a montagem para apresentação. Para a interface de exibição do jogo, foi utilizado papel vegetal sobre as matrizes de *LEDs*, uma caixa de sapato para acomodação das matrizes e da *FPGA*, papel cartão preto fosco para revestimento da caixa, panfletos ilustrativos com o manual de instruções do jogo e cartaz ilustrativo.



Figura 35: Versão física do jogo TEAtris com comandos interativos e display em matriz de LEDs



Figura 35: Montagem do Projeto para apresentação

11.1. Projeto lógico

Abaixo está especificado como o projeto lógico final foi desenvolvido. Vale destacar que o grupo precisou realizar algumas alterações em relação ao que foi proposto originalmente na semana 4 para conseguir acomodar novas implementações, como a chave seletora de nível, a animação no início do jogo e os novos mapas do modo de jogo difícil, por exemplo.

11.1.1. Fluxo de dados

Em relação à semana anterior, não tivemos alterações no Fluxo de Dados. Conseguimos mantê-lo e implementar modificações apenas nas memórias de mapas e na Unidade de Controle.

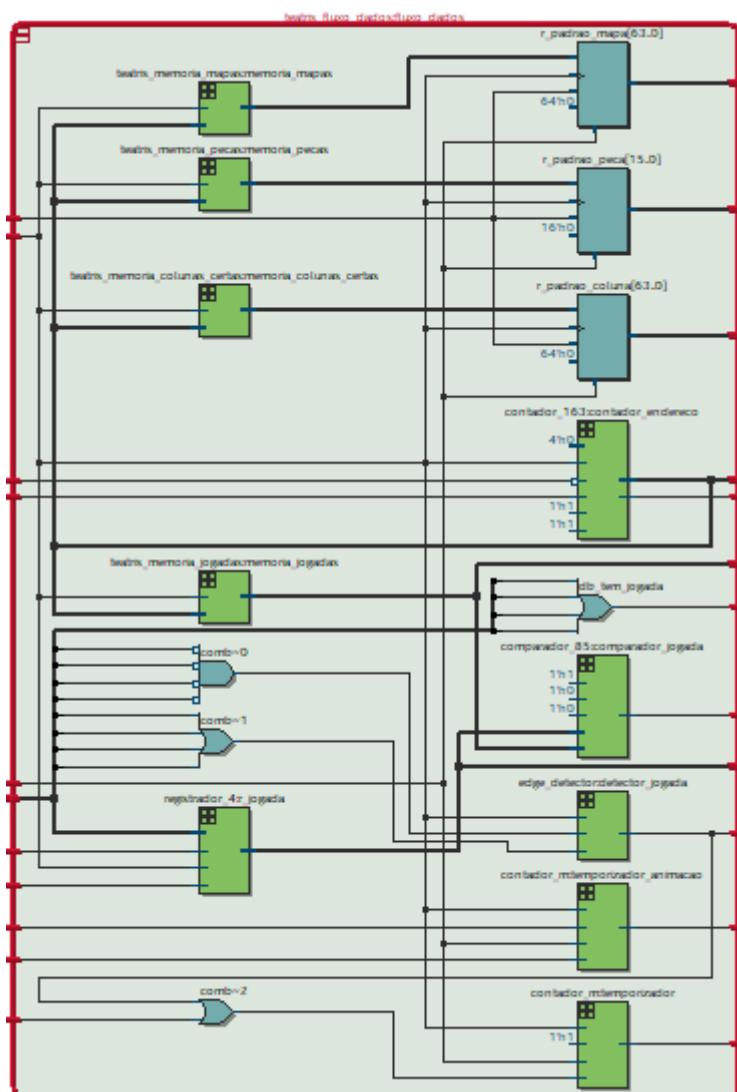


Figura 36: Fluxo de Dados final

```
module teatris_fluxo_dados (
    // Entradas básicas
    input clock,                      // Clock do sistema
    input reset,                       // Sinal de reset
    input [3:0] botoes,                // Botões de entrada

    // Sinais de controle da UC
```

```

    input zera_contador,           // Zera o contador de endereços
    input conta_contador,         // Incrementa o contador
    input enable_memoria,        // Habilita leitura/registro da memória
    input registra_jogada,       // Registra a jogada atual
    input zera_jogada,
    input timer_restart,          // Reinicia o temporizador
    input timer_animacao_restart,
    input conta_timer_animacao,

    // Sinais de estado para a UC
    output tem_jogada,           // Indica que um botão foi pressionado
    output jogada_ok,             // Indica que a jogada está correta
    output fim_sequencia,        // Indica fim da sequência atual
    output timeout,               // Indica que o tempo se esgotou
    output fim_timer_animacao,

    // Saídas para exibição
    output [15:0] padrao_peca,    // Padrão para matriz de peça
    output [63:0] padrao_mapa,    // Padrão para matriz de mapa
    output [63:0] padrao_coluna,  // Padrão para matriz de colunas
certas

    // Saídas de depuração
    output [3:0] db_contagem,     // Valor atual do contador
    output [3:0] db_jogada,       // Jogada atual registrada
    output [3:0] db_memoria,      // Valor esperado da memória
    output db_tem_jogada

);

    // Sinais internos
    wire [3:0] s_endereco;        // Endereço atual da memória
    wire [15:0] s_padrao_peca;    // Padrão de peça da memória
    wire [63:0] s_padrao_mapa;    // Padrão de mapa da memória
    wire [63:0] s_padrao_coluna;
    wire [3:0] s_jogada_correta;  // Jogada correta para o padrão atual
    wire s_pulso_jogada;
    wire [3:0] s_jogada_atual;    // Jogada atual registrada
    reg [15:0] r_padrao_peca;    // Registrador para padrão de peça
    reg [63:0] r_padrao_mapa;    // Registrador para padrão de mapa
    reg [63:0] r_padrao_coluna;  // Registrador para padrão de
colunas corretas

```

```

edge_detector detector_jogada (
    .clock(clock),
    .reset(~botoes[3] & ~botoes[2] & ~botoes[1] & ~botoes[0]),
    .sinal(botoes[3] | botoes[2] | botoes[1] | botoes[0]),
    .pulso(s_pulso_jogada)
);

// Contador de endereços para acessar a memória
contador_163 contador_endereco (
    .clock(clock),
    .clr(~zera_contador),
    .ld(1'b1), // Load ativo alto
    .ent(1'b1), // Enable de contagem
    .enp(conta_contador), // Enable de incremento
    .D(4'b0000), // Valor inicial
    .Q(s_endereco), // Saída do contador
    .rco(fim_sequencia) // Ripple Carry Out como fim de
sequência
);

// Memórias ROM
teatris_memoria_pecas memoria_pecas (
    .clock(clock),
    .endereco(s_endereco),
    .padrao(s_padrao_peca)
);

teatris_memoria_mapas memoria_mapas (
    .clock(clock),
    .endereco(s_endereco),
    .padrao(s_padrao_mapa)
);

teatris_memoria_jogadas memoria_jogadas (
    .clock(clock),
    .endereco(s_endereco),
    .jogada(s_jogada_correta)
);

teatris_memoria_colunas_certas memoria_colunas_certas(
    .clock(clock),

```

```

        .endereco(s_endereco),
        .coluna(s_padrao_coluna)
    );

    registrador_4 r_jogada(
        .clock(clock),
        .clear(zera_jogada),
        .enable(registra_jogada),
        .D(botoes),
        .Q(s_jogada_atual)
    );

    // Registradores para armazenar padrões atuais
    always @ (posedge clock or posedge reset) begin
        if (reset) begin
            r_padrao_peca <= 64'b00000000;
            r_padrao_mapa <= 64'b00000000;
            r_padrao_coluna <= 64'b0;
        end
        else if (enable_memoria) begin
            r_padrao_peca <= s_padrao_peca;
            r_padrao_mapa <= s_padrao_mapa;
            r_padrao_coluna <= s_padrao_coluna;
        end
    end
end

// Comparador para verificar jogada
comparador_85 comparador_jogada (
    .A(s_jogada_atual),      // Jogada registrada
    .B(s_jogada_correta),    // Jogada esperada da memória
    .ALBi(1'b0),
    .AGBi(1'b0),
    .AEBi(1'b1),
    .ALBo(),
    .AGBo(),
    .AEB0(jogada_ok)        // Saída: jogada correta ou não
);

// Temporizador para timeout
contador_m #(M(100_000_000), N(27)) temporizador (
    .clock(clock),

```

```

    .zera_as(reset),
    .zera_s(timer_restart || s_pulso_jogada),
    .conta(1'b1),           // Sempre contando
    .Q(),                  // Valor do contador (não usado)
    .fim(timeout)          // Timeout quando chega ao fim
);

// Temporizador para tempo de animacao
contador_m #( .M(100_000_000), .N(27)) temporizador_animacao (
    .clock(clock),
    .zera_as(reset),
    .zera_s(timer_animacao_restart),
    .conta(conta_timer_animacao),
    .Q(),
    .fim(fim_timer_animacao)
);

// Atribuição das saídas
assign padrao_peca = r_padrao_peca;
assign padrao_mapa = r_padrao_mapa;
assign padrao_coluna = r_padrao_coluna;

// Saídas de depuração
assign db_contagem = s_endereco;
assign db_jogada = s_jogada_atual;
assign db_memoria = s_jogada_correta;
assign tem_jogada = s_pulso_jogada;
assign db_tem_jogada = botoes[3] | botoes[2] | botoes[1] |
botoes[0];

endmodule

```

11.1.2. Unidade de Controle

Perceba que, em relação a semanas anteriores, o grupo precisou alterar a entrada da Unidade de Controle para acrescentar “dificuldade” ao jogo. Caso dificuldade seja “1”, o jogo irá para o modo difícil e, assim, o usuário precisará encaixar as peças no local mais apropriado.

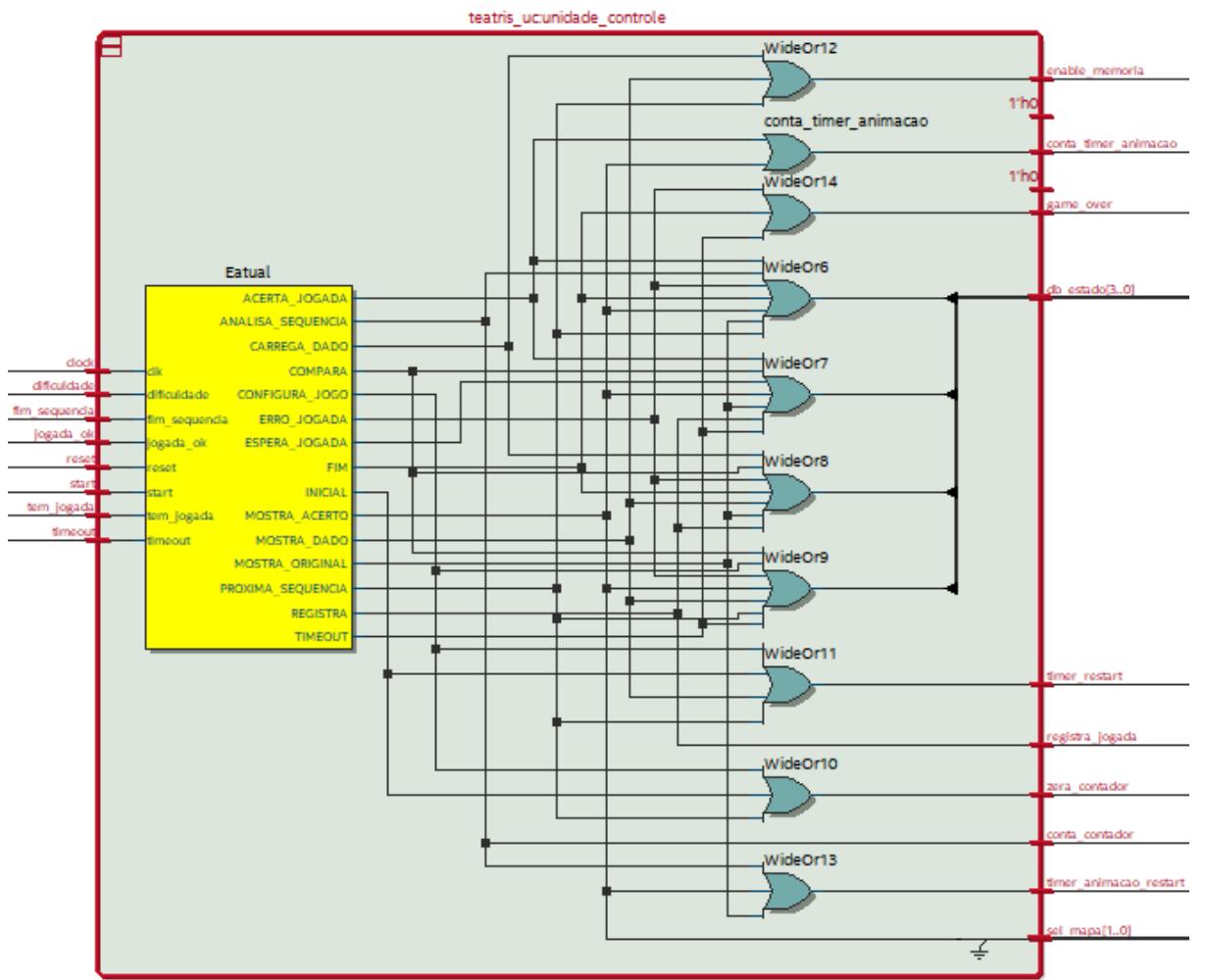


Figura 37: Versão final da Unidade de Controle



Figura 38: Diagrama de Transição de Estados

	Source State	Destination State	Condition
1	ACERTA_JOGADA	MOSTRA_ACERTO	
2	ANALISA_SEQUENCIA	PROXIMA_SEQUENCIA	(fim_sequencia)
3	ANALISA_SEQUENCIA	CARREGA_DADO	(!fim_sequencia)
4	CARREGA_DADO	MOSTRA_DADO	
5	COMPARA	ERRO_JOGADA	(!jogada_ok)
6	COMPARA	ACERTA_JOGADA	(jogada_ok)
7	CONFIGURA_JOGO	CARREGA_DADO	
8	ERRO_JOGADA	INICIAL	(start)
9	ERRO_JOGADA	ERRO_JOGADA	(!start)
10	ESPERA_JOGADA	ESPERA_JOGADA	(!item_jogada). (!timeout)
11	ESPERA_JOGADA	REGISTRA	(tem_jogada). (!timeout)
12	ESPERA_JOGADA	TIMEOUT	(timeout)
13	FIM	INICIAL	(start)
14	FIM	FIM	(!start)
15	INICIAL	INICIAL	(!start)
16	INICIAL	CONFIGURA_JOGO	(start)
17	MOSTRA_ACERTO	MOSTRA_ORIGINAL	
18	MOSTRA_DADO	ESPERA_JOGADA	
19	MOSTRA_ORIGINAL	ANALISA_SEQUENCIA	
20	PROXIMA_SEQUEN...	FIM	(dificuldade)
21	PROXIMA_SEQUEN...	CARREGA_DADO	(!dificuldade)
22	REGISTRA	COMPARA	
23	TIMEOUT	INICIAL	(start)
24	TIMEOUT	TIMEOUT	(!start)

Figura 40: State Machine Viewer

```
module teatris_uc (
    input clock,           // Clock do sistema
    input reset,           // Sinal de reset
    input timeout,          // Sinal de timeout do temporizador
    input start,            // Sinal para iniciar jogo
    input dificuldade,      // Nível de dificuldade
```

```

    input fim_sequencia,           // Indicador de fim da sequência
    input tem_jogada,              // Indicador de jogada realizada
    input jogada_ok,               // Indicador de jogada correta
    input fim_timer_animacao,

    // Sinais de controle para o fluxo de dados
    output reg zera_contador,      // Zera o contador de endereços
    output reg conta_contador,     // Incrementa o contador de endereços
    output reg enable_memoria,     // Habilita registrador de memória
    output reg registra_jogada,    // Habilita registro da jogada
    output reg compara_jogada,     // Habilita comparação da jogada
    output reg timer_restart,      // Reiniciar o temporizador
    output reg game_over,          // Sinal de fim de jogo
    output reg zera_jogada,
    output reg timer_animacao_restart,
    output reg [1:0] sel_mapa,
    output reg conta_timer_animacao,

    // Saída de depuração
    output reg [3:0] db_estado    // Estado atual do jogo
);

// Estados do jogo
localparam INICIAL            = 4'b0000;
localparam CONFIGURA_JOGO      = 4'b0001;
localparam CARREGA_DADO        = 4'b0010;
localparam MOSTRA_DADO         = 4'b0011;
localparam ESPERA_JOGADA       = 4'b0100;
localparam TIMEOUT             = 4'b0101;
localparam REGISTRA            = 4'b0110;
localparam COMPARA              = 4'b0111;
localparam ANALISA_SEQUENCIA   = 4'b1000;
localparam PROXIMA_SEQUENCIA   = 4'b1001;
localparam FIM                  = 4'b1010;
localparam ERRO_JOGADA         = 4'b1011;
localparam ACERTA_JOGADA        = 4'b1100;
localparam MOSTRA_ACERTO        = 4'b1101;
localparam MOSTRA_ORIGINAL     = 4'b1110;

reg [3:0] Eatual, Eprox;

// Memória de Estado
always @(posedge clock or posedge reset) begin

```

```

    if (reset)
        Eatual <= INICIAL;
    else
        Eatual <= Eprox;
end

// Lógica para determinar próximo estado
always @(*) begin
    case (Eatual)
        INICIAL:
            Eprox = start ? CONFIGURA_JOGO : INICIAL;

        CONFIGURA_JOGO:
            Eprox = CARREGA_DADO;

        CARREGA_DADO:
            Eprox = MOSTRA_DADO;

        MOSTRA_DADO:
            Eprox = ESPERA_JOGADA;

        ESPERA_JOGADA:
            Eprox = timeout ? TIMEOUT : tem_jogada ? REGISTRA
: ESPERA_JOGADA;

        TIMEOUT:
            Eprox = start ? INICIAL : TIMEOUT;

        REGISTRA:
            Eprox = COMPARA;

        COMPARA:
            Eprox = jogada_ok ? ACERTA_JOGADA : ERRO_JOGADA;

        ACERTA_JOGADA:
            Eprox = MOSTRA_ACERTO;

        MOSTRA_ACERTO:
            Eprox = MOSTRA_ORIGINAL;

        MOSTRA_ORIGINAL:
            Eprox = ANALISA_SEQUENCIA;

```

```

ANALISA_SEQUENCIA:
    Eprox = fim_sequencia ? PROXIMA_SEQUENCIA :
CARREGA_DADO;

PROXIMA_SEQUENCIA:
    Eprox = (dificuldade) ? FIM : CARREGA_DADO;

ERRO_JOGADA:
    Eprox = start ? INICIAL : ERRO_JOGADA;

FIM:
    Eprox = start ? INICIAL : FIM;

default:
    Eprox = INICIAL;
endcase
end

// Lógica de saída (máquina Moore - saídas dependem apenas do
estado atual)
always @(*) begin
    // Valores default
    zera_contador = 1'b0;
    conta_contador = 1'b0;
    enable_memoria = 1'b0;
    registra_jogada = 1'b0;
    compara_jogada = 1'b0;
    timer_restart = 1'b0;
    game_over = 1'b0;
    db_estado = Eatual; // Sempre mostra o estado atual
    timer_animacao_restart = 1'b0;
    sel_mapa = 2'b00;
    conta_timer_animacao = 1'b0;

    case (Eatual)
        INICIAL: begin
            zera_contador = 1'b1;
            timer_restart = 1'b1;
            zera_jogada = 1'b0;
        end

```

```

CONFIGURA_JOGO: begin
    zera_contador = 1'b1;
    timer_restart = 1'b1;
    zera_jogada = 1'b0;
end

CARREGA_DADO: begin
    enable_memoria = 1'b1;
end

MOSTRA_DADO: begin
    enable_memoria = 1'b1;
    timer_restart = 1'b1;
end

ESPERA_JOGADA: begin
    // Aguarda entrada
end

ACERTA_JOGADA:begin
    sel_mapa = 2'b00;
    conta_timer_animacao = 1'b1;
end

MOSTRA_ACERTO:begin
    sel_mapa = 2'b01;
    timer_animacao_restart = 1'b1;
    conta_timer_animacao = 1'b1;
end

MOSTRA_ORIGINAL:begin
    // espera timer
    timer_animacao_restart = 1'b1;
end

TIMEOUT: begin
    game_over = 1'b1;
end

REGISTRA: begin
    registra_jogada = 1'b1;
end

```

```
COMPARA: begin
end

ANALISA_SEQUENCIA: begin
    conta_contador = 1'b1;
    timer_animacao_restart = 1'b1;
end

PROXIMA_SEQUENCIA: begin
    zera_contador = 1'b1;
    enable_memoria = 1'b1;
    timer_restart = 1'b1;
end

ERRO_JOGADA: begin
    game_over = 1'b1;
end

FIM: begin
    game_over = 1'b1;
end

default: begin
    zera_contador = 1'b1;
end
endcase
end

endmodule
```

11.4.3 Top Level

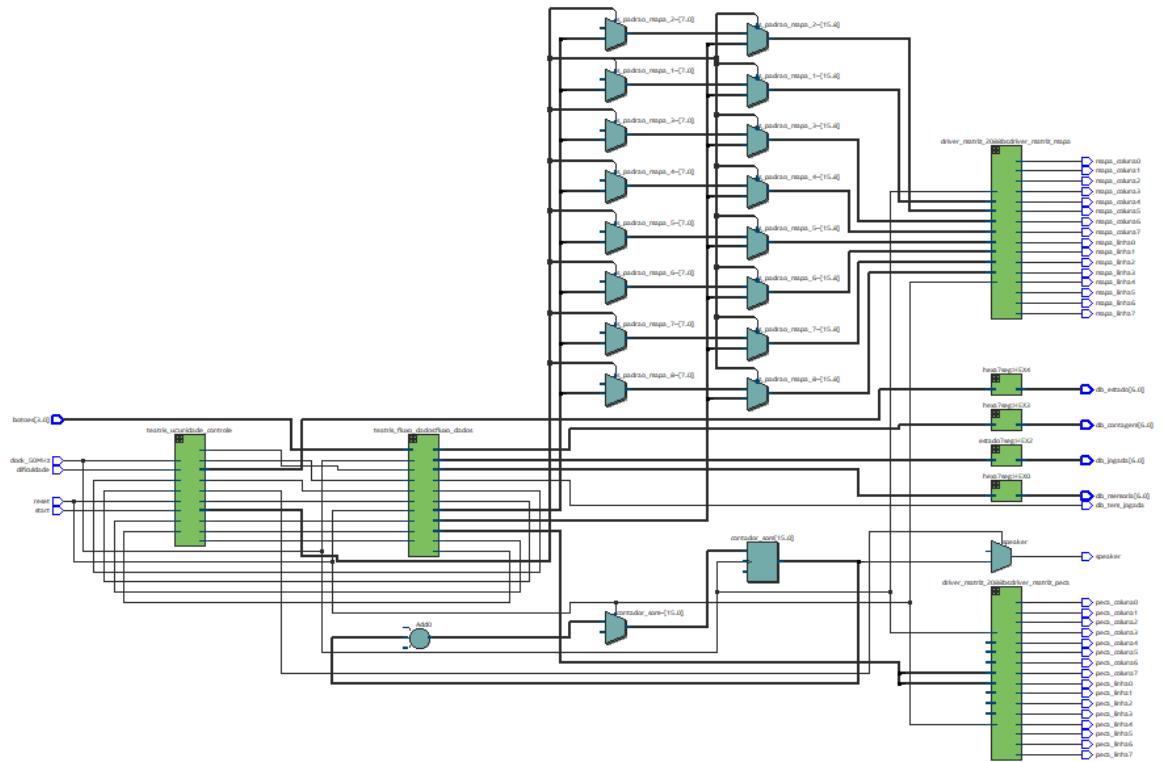


Figura 41: Versão final do Top Level

```
module TEAtris_Top (
    // Entradas principais
    input clock_50MHz,           // Clock principal (tipicamente 50MHz)
    input reset,                 // Reset global (ativo alto)
    input start,                 // Botão para iniciar o jogo
    input [3:0] botoes,          // 4 botões para as jogadas
    input dificuldade,          // Seletor de nível de dificuldade

    // Saídas para matriz de peça (linhas e colunas individuais)
    output peca_linha0,
    output peca_linha1,
    output peca_linha2,
    output peca_linha3,
    output peca_linha4,
    output peca_linha5,
    output peca_linha6,
    output peca_linha7,
    output peca_coluna0,
    output peca_coluna1,
    output peca_coluna2,
```

```

    output peca_coluna3,
    output peca_coluna4,
    output peca_coluna5,
    output peca_coluna6,
    output peca_coluna7,

    // Saídas para matriz de mapa (linhas e colunas individuais)
    output mapa_linha0,
    output mapa_linha1,
    output mapa_linha2,
    output mapa_linha3,
    output mapa_linha4,
    output mapa_linha5,
    output mapa_linha6,
    output mapa_linha7,
    output mapa_coluna0,
    output mapa_coluna1,
    output mapa_coluna2,
    output mapa_coluna3,
    output mapa_coluna4,
    output mapa_coluna5,
    output mapa_coluna6,
    output mapa_coluna7,

    // Saída para som (opcional)
    output speaker,                                // Saída para speaker/buzzer

    // Saídas de depuração
    output [6:0] db_estado,                         // Estado atual da UC
    output [6:0] db_contagem,                       // Contador atual
    output [6:0] db_jogada,                          // Jogada atual
    output [6:0] db_memoria,                         // Valor da memória
    output db_tem_jogada

);

// Sinais entre UC e FD
wire s_zera_contador;
wire sContaContador;
wire s_enable_memoria;
wire s_RegistraJogada;
wire s_compara_jogada;

```

```

wire s_timer_restart;
wire s_timer_animacao_restart;
wire s_conta_timer_animacao;
wire s_tem_jogada;
wire s_jogada_ok;
wire s_fim_sequencia;
wire s_timeout;
wire s_game_over;
wire [1:0] sel_mapa;
wire [3:0] db_jogada_fio;
wire [3:0] db_estado_fio;
wire [3:0] db_memoria_fio;
wire [3:0] db_contagem_fio;

// Sinais para matrizes de LED
wire [15:0] s_padrao_peca;
wire [7:0] s_padrao_peca_sup;
wire [7:0] s_padrao_peca_inf;

wire [63:0] s_padrao_mapa;
wire [7:0] s_padrao_mapa_1;
wire [7:0] s_padrao_mapa_2;
wire [7:0] s_padrao_mapa_3;
wire [7:0] s_padrao_mapa_4;
wire [7:0] s_padrao_mapa_5;
wire [7:0] s_padrao_mapa_6;
wire [7:0] s_padrao_mapa_7;
wire [7:0] s_padrao_mapa_8;

wire [63:0] s_padrao_coluna;

// Sinais para exibição nas matrizes
wire [7:0] s_peca_linhas [0:7];
wire [7:0] s_mapa_linhas [0:7];

// Instanciação da UNIDADE DE CONTROLE
teatris_uc unidade_controle (
    .clock(clock_50MHz),
    .reset(reset),
    .timeout(s_timeout),
    .start(start),
    .dificuldade(dificuldade),

```

```

.fim_sequencia(s_fim_sequencia),
.tem_jogada(s_tem_jogada),
.jogada_ok(s_jogada_ok),

.sel_mapa(sel_mapa),
.zera_contador(s_zera_contador),
.conta_contador(sContaContador),
.enable_memoria(s_enable_memoria),
.registra_jogada(s_registra_jogada),
.timer_restart(s_timer_restart),
.timer_animacao_restart (s_timer_animacao_restart),
.conta_timer_animacao (s_conta_timer_animacao),
.game_over(s_game_over),
.db_estado(db_estado_fio)

);

// Instanciação do FLUXO DE DADOS
teatris_fluxo_dados fluxo_dados (
    .clock(clock_50MHz),
    .reset(reset),
    .botoes(botoes),

    .zera_contador(s_zera_contador),
    .conta_contador(sContaContador),
    .enable_memoria(s_enable_memoria),
    .registra_jogada(s_registra_jogada),
    .timer_restart(s_timer_restart),
    .timer_animacao_restart (s_timer_animacao_restart),
    .conta_timer_animacao (s_conta_timer_animacao),

    .tem_jogada(s_tem_jogada),
    .jogada_ok(s_jogada_ok),
    .fim_sequencia(s_fim_sequencia),
    .timeout(s_timeout),

    .padrao_peca(s_padrao_peca),
    .padrao_mapa(s_padrao_mapa),
    .padrao_coluna(s_padrao_coluna),

    .db_contagem(db_contagem_fio),
    .db_jogada(db_jogada_fio),
    .db_memoria(db_memoria_fio),

```

```

    .db_tem_jogada(db_tem_jogada)
);

// Configuração dos padrões para as matrizes de LEDs
assign s_padrao_peca_sup = s_padrao_peca [7:0];
assign s_padrao_peca_inf = s_padrao_peca [15:8];
assign s_padrao_mapa_8 = ~sel_mapa[0] ? s_padrao_mapa [7 : 0] :
sel_mapa[1] ? s_padrao_coluna [7:0] : 8'b0;
assign s_padrao_mapa_7 = ~sel_mapa[0] ? s_padrao_mapa [15: 8] :
sel_mapa[1] ? s_padrao_coluna [15:8] : 8'b0;
assign s_padrao_mapa_6 = ~sel_mapa[0] ? s_padrao_mapa [23:16] :
sel_mapa[1] ? s_padrao_coluna [23:16] : 8'b0;
assign s_padrao_mapa_5 = ~sel_mapa[0] ? s_padrao_mapa [31:24] :
sel_mapa[1] ? s_padrao_coluna [31:24] : 8'b0;
assign s_padrao_mapa_4 = ~sel_mapa[0] ? s_padrao_mapa [39:32] :
sel_mapa[1] ? s_padrao_coluna [39:32] : 8'b0;
assign s_padrao_mapa_3 = ~sel_mapa[0] ? s_padrao_mapa [47:40] :
sel_mapa[1] ? s_padrao_coluna [47:40] : 8'b0;
assign s_padrao_mapa_2 = ~sel_mapa[0] ? s_padrao_mapa [55:48] :
sel_mapa[1] ? s_padrao_coluna [55:48] : 8'b0;
assign s_padrao_mapa_1 = ~sel_mapa[0] ? s_padrao_mapa [63:56] :
sel_mapa[1] ? s_padrao_coluna [63:56] : 8'b0;

// Configuração para matriz de peça (centralizada nas linhas 3 e 4)
assign s_peca_linhas[0] = 8'b11111111;
assign s_peca_linhas[1] = 8'b11111111;
assign s_peca_linhas[2] = 8'b11111111;
assign s_peca_linhas[3] = s_padrao_peca_sup;
assign s_peca_linhas[4] = s_padrao_peca_inf;
assign s_peca_linhas[5] = 8'b11111111;
assign s_peca_linhas[6] = 8'b11111111;
assign s_peca_linhas[7] = 8'b11111111;

// Configuração para matriz de mapa (padrão na linha 3)
assign s_mapa_linhas[0] = s_padrao_mapa_1;
assign s_mapa_linhas[1] = s_padrao_mapa_2;
assign s_mapa_linhas[2] = s_padrao_mapa_3;
assign s_mapa_linhas[3] = s_padrao_mapa_4;
assign s_mapa_linhas[4] = s_padrao_mapa_5;
assign s_mapa_linhas[5] = s_padrao_mapa_6;
assign s_mapa_linhas[6] = s_padrao_mapa_7;
assign s_mapa_linhas[7] = s_padrao_mapa_8;

```

```
// Instanciação dos DRIVERS PARA MATRIZES 2088BS com pinos individuais
driver_matrix_2088bs driver_matrix_peca (
    .clock(clock_50MHz),
    .reset(reset),
    .padrao_linha0(s_peca_linhas[0]),
    .padrao_linha1(s_peca_linhas[1]),
    .padrao_linha2(s_peca_linhas[2]),
    .padrao_linha3(s_peca_linhas[3]),
    .padrao_linha4(s_peca_linhas[4]),
    .padrao_linha5(s_peca_linhas[5]),
    .padrao_linha6(s_peca_linhas[6]),
    .padrao_linha7(s_peca_linhas[7]),

    // Linhas para matriz de peça
    .linha0(peca_linha0),
    .linha1(peca_linha1),
    .linha2(peca_linha2),
    .linha3(peca_linha3),
    .linha4(peca_linha4),
    .linha5(peca_linha5),
    .linha6(peca_linha6),
    .linha7(peca_linha7),

    // Colunas para matriz de peça
    .coluna0(peca_coluna0),
    .coluna1(peca_coluna1),
    .coluna2(peca_coluna2),
    .coluna3(peca_coluna3),
    .coluna4(peca_coluna4),
    .coluna5(peca_coluna5),
    .coluna6(peca_coluna6),
    .coluna7(peca_coluna7)
);

driver_matrix_2088bs driver_matrix_mapa (
    .clock(clock_50MHz),
    .reset(reset),
    .padrao_linha0(s_mapa_linhas[0]),
    .padrao_linha1(s_mapa_linhas[1]),
    .padrao_linha2(s_mapa_linhas[2]),
    .padrao_linha3(s_mapa_linhas[3]),
```

```

.padrao_linha4(s_mapa_linhas[4]),
.padrao_linha5(s_mapa_linhas[5]),
.padrao_linha6(s_mapa_linhas[6]),
.padrao_linha7(s_mapa_linhas[7]),

// Linhas para matriz de mapa
.linha0(mapa_linha0),
.linha1(mapa_linha1),
.linha2(mapa_linha2),
.linha3(mapa_linha3),
.linha4(mapa_linha4),
.linha5(mapa_linha5),
.linha6(mapa_linha6),
.linha7(mapa_linha7),

// Colunas para matriz de mapa
.coluna0(mapa_coluna0),
.coluna1(mapa_coluna1),
.coluna2(mapa_coluna2),
.coluna3(mapa_coluna3),
.coluna4(mapa_coluna4),
.coluna5(mapa_coluna5),
.coluna6(mapa_coluna6),
.coluna7(mapa_coluna7)

);

estado7seg HEX2(
.estado( db_jogada_fio ),
.display( db_jogada )
);

hexa7seg HEX3(
.hexa( db_contagem_fio ),
.display( db_contagem )
);

hexa7seg HEX0(
.hexa( db_memoria_fio ),
.display( db_memoria )
);

```

```

hexa7seg HEX4(
    .hexa( db_estado_fio ),
    .display( db_estado )
);

// Gerador de som simplificado
reg [15:0] contador_som;
always @(posedge clock_50MHz) begin
    if (reset)
        contador_som <= 0;
    else
        contador_som <= contador_som + 1;
end

// Som ativo durante game_over (pode ser modificado conforme necessário)
assign speaker = s_game_over ? contador_som[14] : 1'b0;

endmodule

```

12. CONCLUSÃO

Ao longo das últimas semanas de desenvolvimento, o TEAtris gradualmente foi se alinhando aos requisitos estabelecidos desde o início do projeto. Graças à análise, ao planejamento e, principalmente, à dedicação do grupo, todos os requisitos, funcionais e não funcionais, foram implementados e devidamente testados.

Além de atender às especificações iniciais, o projeto evoluiu para incluir recursos adicionais que não estavam previstos, como a adição de animações para acertos e erros, seleção de nível de dificuldade, e novas configurações de mapas no modo difícil. Essas melhorias contribuíram significativamente para tornar o jogo mais atrativo, acessível e lúdico.

A montagem física do protótipo foi finalizada com sucesso, utilizando materiais simples como papel cartão e uma caixa de sapato, mas entregando uma interface funcional, intuitiva e visualmente agradável. O uso de matrizes de LEDs e botões com capinhas coloridas colaborou para uma experiência interativa completa.

O TEAtris não apenas cumpre um papel de entretenimento, mas também apresenta grande potencial como ferramenta de apoio à análise cognitiva e motora de indivíduos com Transtorno do Espectro Autista (TEA), aproximando-se, assim, de um produto aplicável em contextos educacionais e terapêuticos.

Durante o desenvolvimento, o grupo consolidou conhecimentos teóricos e práticos relacionados à síntese de circuitos digitais em FPGA, criação de máquinas de estados, manipulação de sinais e fluxos de dados, além de aspectos importantes de projeto, como planejamento, testes, integração e documentação. Dessa forma, o projeto TEAtris se encerra com um resultado altamente satisfatório, evidenciando o cumprimento dos objetivos da disciplina Laboratório Digital I e demonstrando o potencial dos integrantes para desafios ainda mais complexos em etapas futuras da formação em engenharia.