Relatório do Projeto Final
Bomba Relógio

Pedro Henrique Alves Martos - 231026017
Daniel Rodrigues de Abreu - 241038540
Luca Barbosa Santos - 232028286
Grupo B6

Dep. Ciência da Computação – Universidade de Brasília (UnB)

CIC0231 - Laboratório de Circuitos Lógicos

Dezembro de 2025

Abstract

This report presents the procedures and results obtained in carrying out the Final Project of the Logic Circuits Laboratory course, focused on the implementation of a game simulating a safe with a time bomb, in which it was necessary to apply the content of combinational and sequential circuits, with emphasis on finite state machines and the Verilog Hardware Description Language (HDL). The main objectives were to understand and gain practical experience as a designer of complex digital circuits for structural and behavioral modeling, and to master the synthesis workflow on an FPGA platform.

Resumo

Este relatório apresenta os procedimentos e resultados adquiridos na realização do Projeto Final da disciplina Laboratório de Circuitos Lógicos, focado na implementação de um jogo simulador de um cofre com bomba relógio, no qual fez-se necessário utilizar o conteúdo de circuitos combinacionais e sequências, com ênfase em máquinas de estados finitos e da Linguagem de Descrição de Hardware (HDL) Verilog. Os objetivos centrais foram compreender e obter experiência prática como projetista de circuitos digitais complexos para modelagem estrutural e comportamental, e dominar o fluxo de trabalho de síntese em uma plataforma FPGA.

1 Introdução

O desenvolvimento de sistemas digitais permitiu que diversos avanços tecnológicos fossem feitos à humanidade, principalmente em relação com a segurança para armazenamento de pertences pessoais. Neste relatório, é detalhado sobre a implementação em hardware de um jogo digital que simulada o mecanismo de um cofre protegido por uma bomba relógio.

O sistema implementado consiste em uma arquitetura digital capaz de registrar duas senhas de acesso — uma de 4 bits (Senha A) e outra de 3 bits (Senha B) — definidas pelo usuário no momento da ativação. Ao iniciar o jogo, um contador regressivo de 2 minutos e 59 segundos é acionado, sendo exibido em displays de sete segmentos. O jogador deve inserir corretamente as senhas na ordem especificada, utilizando chaves (SW) e validando as tentativas por meio de botões (KEY). A lógica do sistema incorpora um bloco de comparação, geração de dicas, exibição de tentativas e controle de tempo, exigindo a implementação de múltiplos módulos combinacionais e sequenciais em HDL Verilog.

Para garantir o funcionamento adequado do jogo, o projeto também demanda o desenvolvimento de uma máquina de estados robusta, responsável por controlar as transições entre as etapas de configuração, tentativa da Senha A, tentativa da Senha B, sucesso, falha e explosão da bomba. A síntese e validação do sistema foram realizadas no ambiente Quartus-II, com testes práticos executados na placa FPGA DE2, utilizando LEDRs, LEDGs, displays hexadecimais e os periféricos disponíveis.

1.1 Objetivos

Este experimento tem como objetivos principais:

- Implementar um projeto prático temático que aborde todos os conhecimentos e práticas adquiridos ao longo da disciplina.
- Utilizar o software Quartus-II para criar diagramas esquemáticos, realizar simulações funcionais e temporais, e sintetizar circuitos digitais em FPGA.
- Validar o funcionamento de circuitos digitais implementados por meio de testes práticos na placa de desenvolvimento, utilizando suas chaves, botões e LEDs.

1.2 Materiais Utilizados

Os seguintes materiais foram utilizados para a realização do experimento:

- Computador;
- Software Altera Quartus-II Versão 13.0 SP1;
- Kit de desenvolvimento FPGA DE2.

2 Metodologia e Resultados

Nesta seção, é descrito o método e os passos detalhados de execução durante o desenvolvimento do projeto, bem como alguns resultados práticos obtidos. Para guiar a implementação, utilizou-se um esboço de máquina de estados geral do projeto, definindo cada etapa de execução para melhor divisão de tarefas. A separação foi feita em cinco estados principais, além dos módulos de dica, comparação e cronômetro que funcionam concomitantemente com os estados: **ESTADO_SETUP**, **ESTADO_JOGO_A**, **ESTADO_JOGO_B**, **ESTADO_VITORIA** e **ESTADO_GAMEOVER**, os quais serão descritos adiante.

Além disso, devido à complexidade do projeto, optou-se por realizar o sistema totalmente por meio de módulos HDL Verilog, visto que facilita a descrição do hardware. A Figura 1 remonta uma visualização do diagrama e logo abaixo há a definição dos estados na descrição.

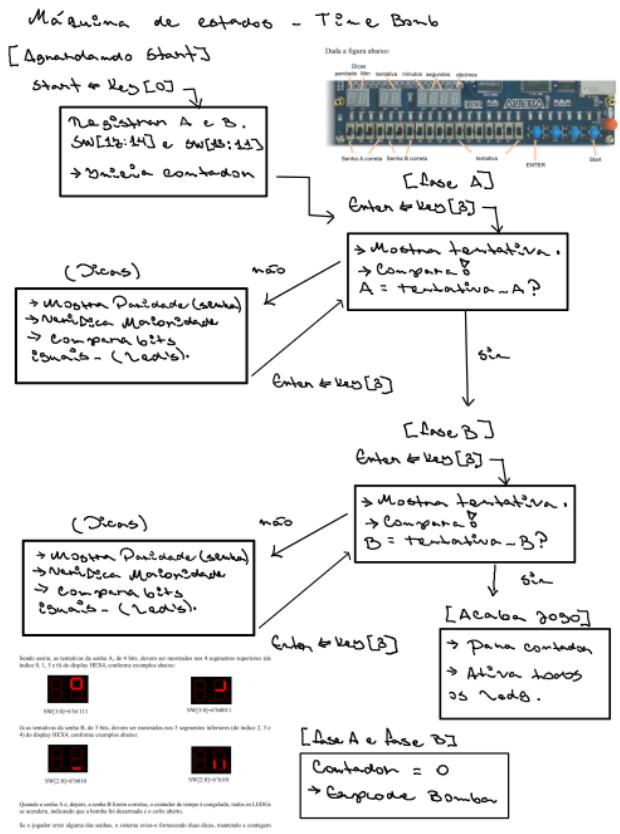

Figura 1: Diagrama Algorítmico dos Estados Presentes no Projeto.

a) Descrição da Definição dos Estados do Jogo

```
1       // Definição de Estados.
2       parameter ESTADO_SETUP      = 3'd0;
3       parameter ESTADO_JOGO_A     = 3'd1;
4       parameter ESTADO_JOGO_B     = 3'd2;
5       parameter ESTADO_VITORIA    = 3'd3;
6       parameter ESTADO_GAMEOVER   = 3'd4;
7
8       reg [2:0] estado_atual;
```

2.1 Estado Inicial: Aguardando Início do Jogo e Registro de Senhas

Primeiramente, foi construída no programa a definição de variáveis compostas pelos botões de **Start**, **Enter** e **Reset**, as chaves para atribuição das senhas e **Clock** para operação síncrona de borda positiva ao sistema.

Além disso, dois registradores de 4 bits e 3 bits, respectivamente, foram utilizados para armazenar as senhas do cofre e um registrador de 3 bits foi atribuído para mudança de estados. O **Reset** é utilizado para reiniciar o jogo, independentemente do estado e o **Start** registra as senhas na borda positiva de ativação (Na placa de desenvolvimento os botões são ativos em nível baixo, então inverteu-se a lógica no programa por conveniência).

Uma dificuldade encontrada durante a utilização dos botões é que a ativação e desativação ocorrem somente em nível. isso implica que, durante o registro das senhas, pode haver diversos registros no mesmo ciclo de **Clock**, causando uma instabilidade no sistema. Para contornar a situação, implementou-se registradores que fazem a leitura do nível anterior, o que permite que a ativação só ocorra em borda positiva e elimina tais **Bugs** de ocorrerem.

a) Descrição da Definição de Variáveis e Captura de Senha

```
    input CLOCK_50,
    input [3:0] KEY,  // KEY[0] = Start, KEY[1] = Reset, KEY[3] = Enter.
    input [17:0] SW,  // Configuração e Tentativas.

     // Registradores de Senha e Tentativa.
    reg [3:0] senha_a_secreta;
    reg [2:0] senha_b_secreta;
    reg [3:0] tentativa_a_registrada;
    reg [2:0] tentativa_b_registrada;

    // Tratamento de Botões.
    // Inverte a lógica: (Botão da FPGA é Ativa em Nível Baixo).
    wire bt_start_raw = ~KEY[0];
    wire bt_reset     = ~KEY[1];
    wire bt_enter_raw = ~KEY[3];

    // Registradores para Armazenar o Estado Anterior do Botão.
    reg bt_start_ant;
    reg bt_enter_ant;

    // Lógica de Detecção de Borda de Subida (0 -> 1).
    // O Sinal Só Fica Alto por 1 Ciclo de Clock.
    wire start_posedge = bt_start_raw && !bt_start_ant;
    wire enter_posedge = bt_enter_raw && !bt_enter_ant;

    always @(posedge CLOCK_50) begin
        bt_start_ant <= bt_start_raw;
        bt_enter_ant <= bt_enter_raw;
    end

        always @(posedge CLOCK_50 or posedge bt_reset) begin
        if (bt_reset) begin
            estado_atual <= ESTADO_SETUP;
            senha_a_secreta <= 0;
            senha_b_secreta <= 0;
            tentativa_a_registrada <= 0;
            tentativa_b_registrada <= 0;
        end else begin
            case (estado_atual) // Escolha dos Estados.
                ESTADO_SETUP: begin
                    if (start_posedge) begin
                        senha_a_secreta <= SW[17:14];
                        senha_b_secreta <= SW[13:11];
                        estado_atual <= ESTADO_JOGO_A;
                    end
                end
```

2.2 Módulo do Cronômetro: Contagem Regressiva e Divisor de Clock

Para o controle temporal do jogo, foi desenvolvido o módulo `Cronometer`. Como o clock base da placa FPGA DE2 opera a 50 MHz, foi necessário implementar um divisor de frequência, denominado `gerador_pulso_ms`, que converte o clock rápido em um sinal de habilitação (*tick*) de 1 milissegundo.

O cronômetro recebe este sinal e realiza o decremento encadeado de milésimos, segundos e minutos, começando de 2 minutos e 59 segundos. Caso o tempo atinja zero (00:00:000), o

módulo ativa o sinal `time_over`, informando à máquina de estados que o jogo acabou.

a) Integração do Cronômetro

A instância do módulo no Top-Level conecta o sinal de reset, os estados de jogo e as saídas para os displays de 7 segmentos (HEX3 a HEX0), garantindo que o tempo só corra durante as etapas ativas do jogo (A e B).

2.3 Estados de Jogo (A e B) e Visualização Customizada no Display

A lógica central do jogo ocorre nos estados **ESTADO_JOGO_A** e **ESTADO_JOGO_B**. Neles, o sistema aguarda que o usuário configure a tentativa nas chaves (SW) e pressione **Enter**.

Para facilitar a interface com o usuário, implementou-se uma visualização gráfica no display **HEX4**. Ao invés de números, o display desenha "quadrados"que representam a tentativa atual:

- **Senha A:** Utiliza os segmentos superiores (A, B, F, G) do display.
- **Senha B:** Utiliza os segmentos inferiores (C, D, E) do display.

Essa separação visual permite que o jogador saiba exatamente qual senha está inserindo antes de confirmar.

a) Descrição da Definição dos Estados e HEX4

```
    // Lógica Visual HEX4 (Quadrado Superior para A, Inferior para B)
    assign hex4_signals[0] = ~tentativa_a_registrada[3]; // Seg A
    assign hex4_signals[1] = ~tentativa_a_registrada[2]; // Seg B
    assign hex4_signals[5] = ~tentativa_a_registrada[1]; // Seg F
    assign hex4_signals[6] = ~tentativa_a_registrada[0]; // Seg G

    // ... Lógica similar para Senha B nos segmentos C, D, E ...

    ESTADO_JOGO_A: begin
        if (time_over) estado_atual <= ESTADO_GAMEOVER;

        if (enter_posedge) begin
            tentativa_a_registrada <= SW[3:0]; // Grava tentativa A
            // Verifica se acertou a senha A
            if (SW[3:0] == senha_a_secreta) begin
                estado_atual <= ESTADO_JOGO_B;
            end
        end
    end
```

2.4 Módulo de Comparação de Senhas

O módulo de comparação foi desenvolvido para verificar se a tentativa inserida pelo jogador corresponde à senha correta armazenada internamente no sistema. Como o projeto utiliza duas senhas — Senha A (4 bits) e Senha B (3 bits) — adotou-se um mecanismo seletivo controlado pelo sinal `modoB`. Quando `modoB = 0`, o sistema compara a tentativa da Senha A; quando `modoB = 1`, compara a tentativa da Senha B.

O módulo é totalmente combinacional, garantindo resposta imediata sem depender do clock. A saída `resultado` utiliza a seguinte codificação:

00 – Tentativa correta

01 – Tentativa incorreta

Esse valor é utilizado pela máquina de estados para decidir se o jogo deve avançar para a próxima etapa (Senha B) ou permanecer exibindo as dicas ao jogador.

O descrição completo do módulo é apresentado abaixo:

a) Módulo Comparador de Senhas e Tentativas

```
module comparador(
    input tentativaA,
    input A_Pin,
    input tentativaB,
    input B_Pin,
    input modoB, // 0 = comparando A, 1 = comparando B

    output reg [1:0] resultado
    );

    always @(*) begin
        if (modoB == 1'b0) begin
            // Comparação da Senha A
            if (tentativaA == A_Pin)
                resultado = 2'b00; // acerto
            else
                resultado = 2'b01; // erro
        end else begin
            // Comparação da Senha B
            if (tentativaB == B_Pin)
                resultado = 2'b00; // acerto
            else
                resultado = 2'b01; // erro
        end
    end
endmodule
```

A utilização de um único módulo para ambas as senhas reduz o uso de lógica na FPGA e simplifica significativamente a integração com a máquina de estados.

2.5 Módulo de Dicas: Paridade e Comparação Numérica

O módulo de dicas tem como objetivo fornecer ao jogador duas informações auxiliares após uma tentativa incorreta: o bit de paridade das senhas e a comparação numérica entre a tentativa atual e a senha correta.

As duas saídas geradas são:

HEX7 — Paridade: indica se o número total de bits '1' nas duas senhas é par ou ímpar.

HEX6 — Comparação: indica se a tentativa é **menor**, **maior** ou **igual** à senha correta.

Assim como no comparador, o módulo utiliza `modoB` para decidir se deve operar sobre a Senha A ou sobre a Senha B.

a) Cálculo do Bit de Paridade

A paridade é o XOR de todos os bits presentes nas duas senhas. Caso o número de '1's seja ímpar, o resultado será 1; caso seja par, será 0.

```
    assign paridade = senhaA[3] ^ senhaA[2] ^ senhaA[1] ^ senhaA[0] ^
senhaB[2] ^ senhaB[1] ^ senhaB[0];
```

Essa implementação utiliza a propriedade natural do XOR como soma módulo 2, resultando em lógica simples e eficiente.

b) Comparação Numérica da Tentativa

Esta dica informa ao jogador se a tentativa atual está acima, abaixo ou exatamente igual à senha correta:

00 – Tentativa menor
01 – Tentativa maior
10 – Tentativa igual

```
always @(*) begin
    if (!modoB) begin
        // Comparação com Senha A (4 bits)
        if (tentativaA > senhaA) comp = 2'b01;
        else if (tentativaA < senhaA) comp = 2'b00;
        else comp = 2'b10;
    end else begin
        // Comparação com Senha B (3 bits)
        if (tentativaB > senhaB) comp = 2'b01;
        else if (tentativaB < senhaB) comp = 2'b00;
        else comp = 2'b10;
    end
end
```

c) Integração com o Sistema Principal

O módulo `dicas` opera em conjunto com o módulo `comparador`. Após pressionado `ENTER`:

- A tentativa é registrada e o comparador determina acerto ou erro.
- Em caso de erro: O bit de paridade é atualizado em **(HEX7)** e a comparação numérica é exibida em **(HEX6)**,
- A máquina de estados permanece na etapa de tentativa atual.

Esse mecanismo fornece ao jogador um retorno suficiente para ajustar suas próximas tentativas, conforme exigido pela especificação do projeto.

2.6 Estados de Fim de Jogo: Bomba Desarmada e Explosão da Bomba

Finalmente, há dois casos finais para o encerramento do jogo. O estado de **bomba desarmada** (Vitória) ocorre quando a Senha B é acertada. O sistema trava, exibe a senha no HEX4 e acende os LEDs verdes.

O estado de **explosão da bomba** (Game Over) ocorre se o tempo acabar. Foi implementado um efeito visual onde um contador de 25 bits gera um sinal de clock lento (`pisca_lento`). Esse sinal faz com que todos os LEDs vermelhos (LEDR) pisquem intermitentemente, alertando o jogador sobre a explosão.

a) Descrição da Explosão e Vitória

```
// Contador para efeito de pisca (Explosão)
reg [24:0] contador_pisca;
always @(posedge CLOCK_50) contador_pisca <= contador_pisca + 1;
wire pisca_lento = contador_pisca[24];

// LEDS VERMELHOS (Explosão se Game Over e pisca_lento ativo)
```

```
7       assign LEDR = (estado_atual == ESTADO_GAMEOVER && pisca_lento) ? 18'
            b111111111111111111 : ...;
8
9       // LEDS VERDES (Vitória)
10      assign LEDG = (estado_atual == ESTADO_VITORIA) ? 9'b111111111 : 9'
            b000000000;
```

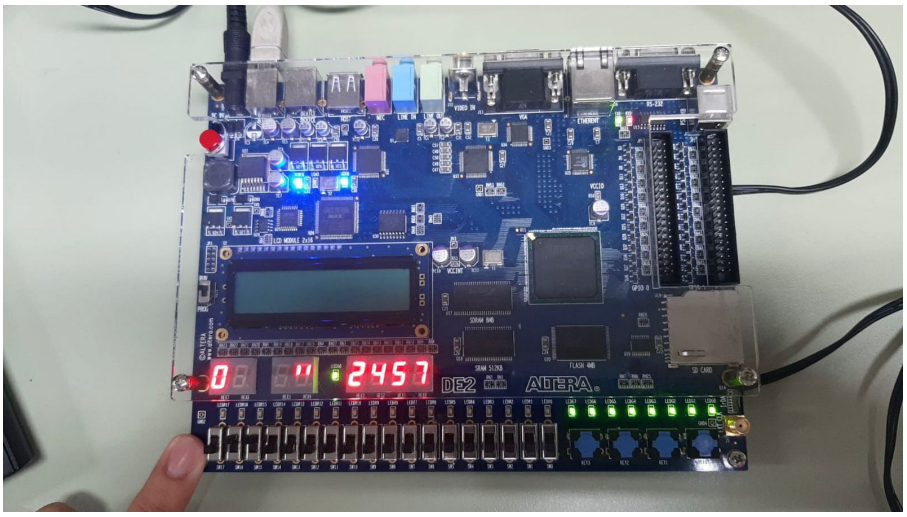

Figura 2: Imagem da Bomba Desarmada.

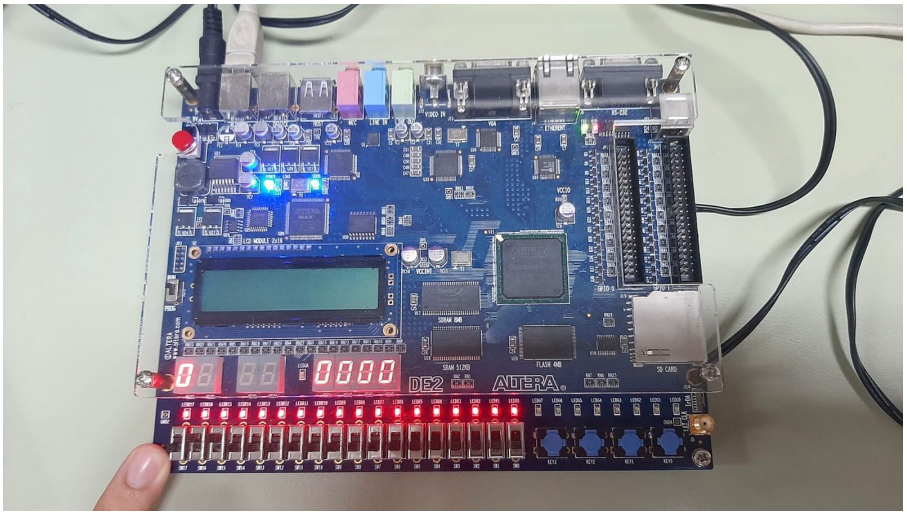

Figura 3: Imagem da Explosão.

Vídeo de Demonstração

O vídeo a seguir demonstra o funcionamento do projeto completo estruturado e implementado na placa FPGA Altera DE2.

Link: Clique aqui para assistir ao vídeo no YouTube.

3 Conclusão

A implementação do projeto de bomba relógio possibilitou consolidar os principais conceitos de circuitos digitais, integrando lógica combinacional, sequencial, máquinas de estados finitos

e modelagem em Verilog. A construção da arquitetura, o controle do contador regressivo, a lógica de comparação de senhas e o fornecimento de dicas exigiram planejamento, validação incremental e domínio do fluxo de síntese em FPGA.

Os testes em hardware demonstraram que o sistema atende aos requisitos funcionais, reproduzindo com precisão o comportamento esperado do jogo. A realização deste trabalho reforçou a importância do aprendizado prático em projetos digitais, ampliando a experiência na implementação, depuração e integração de módulos em plataformas reconfiguráveis.

Referências

[1] Universidade de Brasília. Departamento de Ciência da Computação. (2025). *Especificações do Projeto Final da Disciplina: Bomba Relógio*. Laboratório de Circuitos Lógicos.

[2] TOCCI, Ronald J.; WIDMER, Neal S.; MOSS, Gregory L. Sistemas digitais: princípios e aplicações. 11. ed. São Paulo: Pearson, 2016. Cap. 10 – Projeto de Sistema Digital Usando HDL.

Apêndice A — Descrição Verilog Completa do Projeto

Este apêndice apresenta a Descrição-fonte completa implementada em Verilog para o projeto *Bomba Relógio*. Os módulos do projeto estão organizados a seguir.

a) Módulo Top-Level do Projeto

```
module Time_Bomb (
    input CLOCK_50,
    input [3:0] KEY,      // KEY[0]=Start, KEY[1]=Reset, KEY[3]=Enter
    input [17:0] SW,      // Configuração e Tentativas

    output [6:0] HEX7, HEX6, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0,
    output [8:0] LEDG,    // LEDs Verdes (Vitória)
    output [17:0] LEDR    // LEDs Vermelhos (Barra de Progresso)
);

    // --- ESTADOS ---
    parameter ESTADO_SETUP     = 3'd0;
    parameter ESTADO_JOGO_A    = 3'd1;
    parameter ESTADO_JOGO_B    = 3'd2;
    parameter ESTADO_VITORIA   = 3'd3;
    parameter ESTADO_GAMEOVER  = 3'd4;

    reg [2:0] estado_atual;

    // --- SINAIS INTERNOS ---
    wire tick_ms;
    wire time_over;

    // Registradores de Senha e Tentativa
    reg [3:0] senha_a_secreta;
    reg [2:0] senha_b_secreta;
    reg [3:0] tentativa_a_registrada;
    reg [2:0] tentativa_b_registrada;

    // Fios de conexão com o módulo de Dicas
    wire [6:0] w_hex7_paridade;
```

```
32      wire [6:0] w_hex6_dica;
33      wire [3:0] w_leds_a;
34      wire [2:0] w_leds_b;
35
36      // --- TRATAMENTO DOS BOTÕES (CORREÇÃO CRÍTICA) ---
37      // Inverte a lógica (Botão pressionado na DE2 gera 0)
38      wire bt_start_raw = ~KEY[0];
39      wire bt_reset     = ~KEY[1];
40      wire bt_enter_raw = ~KEY[3];
41
42      // Registradores para armazenar o estado anterior do botão
43      reg bt_start_ant;
44      reg bt_enter_ant;
45
46      // Lógica de detecção de borda de subida (0 -> 1)
47      // O sinal só fica alto por 1 ciclo de clock, evitando leituras mú
           ltiplas
48      wire start_posedge = bt_start_raw && !bt_start_ant;
49      wire enter_posedge = bt_enter_raw && !bt_enter_ant;
50
51      always @(posedge CLOCK_50) begin
52          bt_start_ant <= bt_start_raw;
53          bt_enter_ant <= bt_enter_raw;
54      end
55
56      // --- EFEITO PISCA-PISCA (Para Explosão) ---
57      reg [24:0] contador_pisca;
58      always @(posedge CLOCK_50) contador_pisca <= contador_pisca + 1;
59      wire pisca_lento = contador_pisca[24]; // Pisca a cada ~0.6 segundos
60
61      // --- MÁQUINA DE ESTADOS ---
62      always @(posedge CLOCK_50 or posedge bt_reset) begin
63          if (bt_reset) begin
64              estado_atual <= ESTADO_SETUP;
65              senha_a_secreta <= 0;
66              senha_b_secreta <= 0;
67              tentativa_a_registrada <= 0;
68              tentativa_b_registrada <= 0;
69          end else begin
70              case (estado_atual)
71                  ESTADO_SETUP: begin
72                      // Usa o pulso (posedge) ao invés do nível
73                      if (start_posedge) begin
74                          senha_a_secreta <= SW[17:14];
75                          senha_b_secreta <= SW[13:11];
76                          estado_atual <= ESTADO_JOGO_A;
77                      end
78                  end
79
80                  ESTADO_JOGO_A: begin
81                      if (time_over) estado_atual <= ESTADO_GAMEOVER;
82
83                      if (enter_posedge) begin
84                          tentativa_a_registrada <= SW[3:0]; // Grava
                               tentativa A
85
86                          // Verifica se acertou a senha A
87                          if (SW[3:0] == senha_a_secreta) begin
88                              estado_atual <= ESTADO_JOGO_B;
89                          end
```

```
90                      end
91                  end
92
93                  ESTADO_JOGO_B: begin
94                      if (time_over) estado_atual <= ESTADO_GAMEOVER;
95
96                      if (enter_posedge) begin
97                          tentativa_b_registrada <= SW[2:0]; // Grava
                               tentativa B
98
99                          // Verifica se acertou a senha B
100                         if (SW[2:0] == senha_b_secreta) begin
101                             estado_atual <= ESTADO_VITORIA;
102                         end
103                     end
104                 end
105
106                 ESTADO_VITORIA: begin
107                     // Trava aqui até resetar
108                 end
109
110                 ESTADO_GAMEOVER: begin
111                     // Trava aqui até resetar
112                 end
113             endcase
114         end
115     end
116
117     // --- INSTÂNCIAS ---
118
119     // 1. Gerador de Pulso de 1ms
120     // IMPORTANTE: Certifique-se que seu módulo gerador_pulso_ms gera um
            pulso de 1 ciclo (strobe),
121     // e não uma onda quadrada (50% duty cycle), para o cronômetro funcionar
             perfeitamente.
122     gerador_pulso_ms DIVISOR (
123         .clk(CLOCK_50),
124         .tick_1ms(tick_ms)
125     );
126
127     wire [3:0] wm, wd, wu, wdec;
128
129     // 2. Cronômetro Regressivo
130     Cronometer RELOGIO (
131         .clk(CLOCK_50),
132         .reset(bt_reset),
133         // O cronômetro conta apenas durante as fases de jogo
134         .start(estado_atual == ESTADO_JOGO_A || estado_atual ==
                ESTADO_JOGO_B),
135         .game_won(estado_atual == ESTADO_VITORIA),
136         .tick_1ms(tick_ms),
137         .min_unidade(wm),
138         .seg_dezena(wd),
139         .seg_unidade(wu),
140         .ms_decimos(wdec),
141         .time_over(time_over)
142     );
143
144     // 3. Módulo de Dicas
145     dicas DICAS (
```

```
146         .senha_a(senha_a_secreta),
147         .senha_b(senha_b_secreta),
148         .tentativa_a(tentativa_a_registrada),
149         .tentativa_b(tentativa_b_registrada),
150         .fase_b_ativa(estado_atual == ESTADO_JOGO_B),
151         .hex_paridade(w_hex7_paridade),
152         .hex_maior_menor(w_hex6_dica),
153         .leds_barra_a(w_leds_a),
154         .leds_barra_b(w_leds_b)
155     );
156 
157     // --- SAÍDAS VISUAIS ---
158 
159     // DISPLAYS TEMPO (Se explodir/GameOver, eles piscam "00:00")
160     wire mostrar_display = (estado_atual == ESTADO_GAMEOVER) ? pisca_lento :
            1'b1;
161 
162     wire [6:0] h3, h2, h1, h0;
163     Segment_Display_7 D3 (.valor(wm),   .hex(h3));
164     Segment_Display_7 D2 (.valor(wd),   .hex(h2));
165     Segment_Display_7 D1 (.valor(wu),   .hex(h1));
166     Segment_Display_7 D0 (.valor(wdec), .hex(h0));
167 
168     assign HEX3 = mostrar_display ? h3 : 7'b1111111;
169     assign HEX2 = mostrar_display ? h2 : 7'b1111111;
170     assign HEX1 = mostrar_display ? h1 : 7'b1111111;
171     assign HEX0 = mostrar_display ? h0 : 7'b1111111;
172 
173     // DISPLAY DICAS (HEX7 e HEX6)
174     assign HEX7 = w_hex7_paridade;
175     // HEX6 só acende durante o jogo
176     assign HEX6 = (estado_atual == ESTADO_JOGO_A || estado_atual ==
            ESTADO_JOGO_B) ? w_hex6_dica : 7'b1111111;
177     assign HEX5 = 7'b1111111; // Apagado conforme especificação
178 
179     // DISPLAY CUSTOMIZADO HEX4 (Mostra as tentativas inseridas)
180     // Se estiver no Game Over, pisca também!
181     wire mostrar_hex4 = (estado_atual == ESTADO_GAMEOVER) ? pisca_lento : 1'
            b1;
182 
183     wire [6:0] hex4_signals;
184 
185     // Mapeamento conforme PDF Página 2 (Quadrado Superior para A, Inferior
            para B)
186     // Lógica invertida para display 7 segmentos (0 = aceso, 1 = apagado)
187     // Segmentos: 0(A), 1(B), 2(C), 3(D), 4(E), 5(F), 6(G)
188 
189     // Senha A (4 bits): Segmentos 0, 1, 5, 6 (Topo, Sup Dir, Sup Esq, Meio)
190     assign hex4_signals[0] = ~tentativa_a_registrada[3]; // A
191     assign hex4_signals[1] = ~tentativa_a_registrada[2]; // B
192     assign hex4_signals[5] = ~tentativa_a_registrada[1]; // F
193     assign hex4_signals[6] = ~tentativa_a_registrada[0]; // G
194 
195     // Senha B (3 bits): Segmentos 2, 3, 4 (Inf Dir, Base, Inf Esq)
196     assign hex4_signals[2] = ~tentativa_b_registrada[2]; // C
197     assign hex4_signals[3] = ~tentativa_b_registrada[1]; // D
198     assign hex4_signals[4] = ~tentativa_b_registrada[0]; // E
199 
200     assign HEX4 = mostrar_hex4 ? hex4_signals : 7'b1111111;
201 
```

```
    // LEDS VERDES (Vitória)
    // Acendem todos se ganhar
    assign LEDG = (estado_atual == ESTADO_VITORIA) ? 9'b111111111 : 9'
        b000000000;

    // LEDS VERMELHOS (Barra de Progresso + Explosão)
    // Lógica ajustada para o PDF:
    // - Game Over: Pisca todos
    // - Jogo A: Mostra progresso de A nos LEDs [3:0]
    // - Jogo B: Mostra progresso de B nos LEDs [2:0]

    assign LEDR = (estado_atual == ESTADO_GAMEOVER && pisca_lento) ? 18'
        b111111111111111111 :
                  (estado_atual == ESTADO_JOGO_A) ? {14'b0, w_leds_a} :
                  (estado_atual == ESTADO_JOGO_B) ? {15'b0, w_leds_b} :
                  18'b0; // Apagados em Setup ou Vitória (padrão)

endmodule
```

b) Módulo de Cronômetro Regressivo

```
module Cronometer (
    input clk,
    input reset,
    input start,
    input game_won,
    input tick_1ms,      // Pulso de milissegundo

    output reg [3:0] min_unidade, // Minutos
    output reg [3:0] seg_dezena,  // Segundos (Dezena)
    output reg [3:0] seg_unidade, // Segundos (Unidade)
    output reg [3:0] ms_decimos,  // Décimos (para HEX0)
    output reg time_over          // Explodiu
);

    reg contando;
    // Contadores internos de precisão
    reg [3:0] ms_unidade;
    reg [3:0] ms_dezena;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            // COMEÇA EM 2:59 (Regressivo)
            min_unidade <= 2;
            seg_dezena  <= 5;
            seg_unidade <= 9;
            ms_decimos  <= 9;
            ms_dezena   <= 9;
            ms_unidade  <= 9;
            time_over   <= 0;
            contando    <= 0;
        end else begin
            if (start) contando <= 1;
            if (game_won) contando <= 0;

            // Se o tempo acabar (0:00:000)
            if (min_unidade == 0 && seg_dezena == 0 && seg_unidade == 0 &&
                ms_decimos == 0 && ms_dezena == 0 && ms_unidade == 0) begin
                  time_over <= 1;
```

```
                    contando <= 0;
                end
                else if (contando && tick_1ms && !time_over) begin
                    // Lógica de SUBTRAÇÃO (Decremento)
                    if (ms_unidade > 0) begin
                        ms_unidade <= ms_unidade - 1;
                    end else begin
                        ms_unidade <= 9;
                        if (ms_dezena > 0) begin
                            ms_dezena <= ms_dezena - 1;
                        end else begin
                            ms_dezena <= 9;
                            if (ms_decimos > 0) begin
                                ms_decimos <= ms_decimos - 1;
                            end else begin
                                ms_decimos <= 9;
                                if (seg_unidade > 0) begin
                                    seg_unidade <= seg_unidade - 1;
                                end else begin
                                    seg_unidade <= 9;
                                    if (seg_dezena > 0) begin
                                        seg_dezena <= seg_dezena - 1;
                                    end else begin
                                        seg_dezena <= 5; // Volta para 59s
                                        if (min_unidade > 0) begin
                                            min_unidade <= min_unidade - 1;
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
endmodule
```

c) Módulo de Divisor de Frequência de 1000 Hz

```
module gerador_pulso_ms (
    input clk,
    output reg tick_1ms
);
    // 50 MHz = 50.000.000 ciclos/segundo
    // 1 ms = 1.000 Hz
    // Contagem = 50.000.000 / 1.000 = 50.000 ciclos

    reg [15:0] contador; // 16 bits cabem até 65535

    always @(posedge clk) begin
        if (contador == 50000 - 1) begin
            contador <= 0;
            tick_1ms <= 1; // Pulso de 1 ciclo!
        end else begin
            contador <= contador + 1;
            tick_1ms <= 0;
        end
    end
endmodule
```

d) Módulo de Dicas

```
module dicas (
    input [3:0] senha_a,         // Senha Correta A (4 bits)
    input [2:0] senha_b,         // Senha Correta B (3 bits)
    input [3:0] tentativa_a,     // Tentativa do Jogador A
    input [2:0] tentativa_b,     // Tentativa do Jogador B
    input fase_b_ativa,          // 0 = Jogando A, 1 = Jogando B

    output reg [6:0] hex_paridade,     // Para o HEX7
    output reg [6:0] hex_maior_menor, // Para o HEX6
    output reg [3:0] leds_barra_a,     // Barra de progresso A (LEDR 3-0)
    output reg [2:0] leds_barra_b      // Barra de progresso B (LEDR 2-0)
);

    // --- 1. Lógica da Paridade (HEX7) ---
    // Soma todos os bits das senhas corretas (XOR reduzido)
    // Se resultado 1 (Impar) mostra 1. Se 0 (Par) mostra 0.
    wire paridade_bit = ^ {senha_a, senha_b};

    always @(*) begin
        case (paridade_bit)
            1'b0: hex_paridade = 7'b1000000; // Mostra '0'
            1'b1: hex_paridade = 7'b1111001; // Mostra '1'
        endcase
    end

    // --- 2. Lógica Maior/Menor (HEX6) ---
    always @(*) begin
        hex_maior_menor = 7'b1111111; // Padrão: Apagado

        if (fase_b_ativa == 0) begin
            // Analisando Senha A
            if (tentativa_a > senha_a)
                hex_maior_menor = 7'b1001111; // Acende segmento 'a' (Topo/
                    Teto) -> MAIOR
            else if (tentativa_a < senha_a)
                hex_maior_menor = 7'b1111001; // Acende segmento 'd' (Base/
                    Chão) -> MENOR
        end else begin
            // Analisando Senha B
            if (tentativa_b > senha_b)
                hex_maior_menor = 7'b1001111; // MAIOR
            else if (tentativa_b < senha_b)
                hex_maior_menor = 7'b1111001; // MENOR
        end
    end

    // --- 3. Lógica da Barra de Progresso (LEDR) ---

    // Função auxiliar para contar bits iguais (XNOR)
    integer i;
    reg [2:0] contagem_a;
    reg [2:0] contagem_b;

    always @(*) begin
        // Conta acertos em A
        contagem_a = 0;
        for (i = 0; i < 4; i = i + 1) begin
            if (senha_a[i] == tentativa_a[i]) contagem_a = contagem_a + 1;
```

```
        end

        // Conta acertos em B
        contagem_b = 0;
        for (i = 0; i < 3; i = i + 1) begin
            if (senha_b[i] == tentativa_b[i]) contagem_b = contagem_b + 1;
        end

        // Converte contagem em Barra (Thermometer Code) para A
        case (contagem_a)
            3'd0: leds_barra_a = 4'b0000;
            3'd1: leds_barra_a = 4'b0001;
            3'd2: leds_barra_a = 4'b0011;
            3'd3: leds_barra_a = 4'b0111;
            3'd4: leds_barra_a = 4'b1111;
            default: leds_barra_a = 4'b0000;
        endcase

        // Converte contagem em Barra para B
        case (contagem_b)
            3'd0: leds_barra_b = 3'b000;
            3'd1: leds_barra_b = 3'b001;
            3'd2: leds_barra_b = 3'b011;
            3'd3: leds_barra_b = 3'b111;
            default: leds_barra_b = 3'b000;
        endcase
    end
endmodule
```

e) Módulo de Comparação

```
module comparador(
    input tentativaA,
    input A_Pin,
    input tentativaB,
    input B_Pin,
    input modoB,              // 0=A, 1=B
    output reg [1:0] resultado
);
    always @(*) begin
        if (modoB == 1'b0) begin
            // Comparação A
            if (tentativaA == A_Pin)
                resultado = 2'b00; // acerto
            else
                resultado = 2'b01; // erro
        end else begin
            // Comparação B
            if (tentativaB == B_Pin)
                resultado = 2'b00; // acerto
            else
                resultado = 2'b01; // erro
        end
    end
endmodule
```