Projeto Final
BOMBA RELÓGIO

Grupo B4
Francisco Ribeiro de Souza Campos, 222014590
Vinicius Dos Santos Tomé, 232006780

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0231 - Laboratório de Circuitos Lógicos
22 de fevereiro de 2025

222014590@aluno.unb.br, 232006780@aluno.unb.br

Abstract. *This report describes the development and implementation of a bomb's simulation system using FPGA. The system randomly generates a 7-bit password, which must be entered correctly before the countdown timer reaches zero. The user interacts with the system via switches (SW) and buttons (KEY) to input their password attempts. If the password is correct, the timer stops, and the bomb is deactivated. Otherwise, the system provides two hints: a random logical operation result displayed on HEX6 and the number of correct bits indicated by LEDGs. If the user fails to enter the correct password before the time expires, the system simulates an explosion using LEDRs effects. The experiment was successfully implemented in SystemVerilog and tested on FPGA.*

Resumo. *Este relatório descreve o desenvolvimento e implementação de um sistema de simulação de bomba utilizando FPGA [Corporation 2013]. O sistema gera aleatoriamente uma senha de 7 bits, que deve ser inserida corretamente antes que o tempo do contador regressivo se esgote. O usuário interage com o sistema por meio de chaves (SW) e botões (KEY) para inserir suas tentativas de senha. Caso a senha esteja correta, o tempo é congelado e a bomba é desativada. Caso contrário, o sistema fornece duas dicas: o resultado de uma operação lógica aleatória exibida no display de sete segmentos HEX6 e a quantidade de bits corretos indicada pelos LEDGs. Se o tempo se esgotar sem que a senha correta seja inserida, a bomba "explode"por meio de efeito luminoso nos LEDRs. O experimento foi implementado com sucesso em SystemVerilog e testado na FPGA.*

1. Introdução

Neste experimento, foi desenvolvido um sistema de proteção de bomba, no qual o usuário deve inserir uma senha correta antes que o tempo se esgote. O projeto exigiu a revisão e aplicação de conceitos fundamentais de circuitos lógicos, incluindo contadores, divisores de frequência, operações lógicas e manipulação de displays e LEDs.

Para garantir o domínio dos conceitos necessários, foram refeitos todos os experimentos anteriores, incluindo a utilização de displays hexadecimais, manipulação de

LEDRs e LEDGs, e implementação de contadores assíncronos. Essa preparação foi essencial para a realização do projeto, permitindo o desenvolvimento de um código eficiente e funcional em SystemVerilog [Systemverilog 2016]. O sistema foi testado na FPGA, validando seu comportamento e garantindo sua fidelidade ao objetivo proposto.

1.1. Objetivos

O objetivo do experimento foi implementar um jogo com um sistema de segurança e uma bomba-relógio em FPGA, utilizando chaves (SW), botões (KEY), LEDs vermelhos (LEDR), LEDs verdes (LEDG) e displays de sete segmentos (HEX0 a HEX6) para criar uma interface interativa. O sistema deveria ser capaz de sortear uma senha aleatória de 7 bits e permitir que o usuário inserisse tentativas de senha dentro de um tempo limite. Além disso, caso a senha fosse inserida incorretamente, o sistema deveria fornecer dicas ao jogador e manter a contagem regressiva até o acerto ou a explosão da bomba.

1.2. Materiais

Neste projeto, foram utilizados os seguintes materiais e equipamentos:

- Software Quartus-II versão 13.0 SP1
- Pendrive
- FPGA DE2 ALTERA

2. Procedimentos e Resultados

A implementação do sistema foi dividida em módulos interconectados descritos na descrição de linguagem SystemVerilog, cada um desempenhando uma função específica dentro do jogo. Foram três módulos, os quais foram divididos e denominados:

1. Recorder
2. Tips
3. Timer

2.1. Recorder

Primeiramente, foi criado o módulo mais simples dentre os três, o qual tinha como função apenas registrar no display HEX4 a tentativa do usuário após a ativação da chave (KEY3). Além de repassar essa tentativa para o próximo módulo, onde ocorreria a verificação da tentativa com a senha aleatória gerada. Conforme a descrição, observa-se a seguir.

```
    module recorder(
    input reg clk,
    input reg key0,
    input reg key3,
    input reg [6:0] S,
    output reg [6:0] H4
);
    always_ff @(posedge clk or negedge key0) begin
        if (~key0) begin
            H4 <= 7'b0000000;
        end else if (~key3) begin
```

```
            H4 <= S;
        end
    end

endmodule
```

Como houve uma modularização da máquina de estados e o clock utilizado é o de 50MHz, foi possível o teste de funcionamento exclusivo do módulo, o qual pode ser visto pelo vídeo Recorder e pela simulação em ondas, conforme é demonstrado na figura 3.

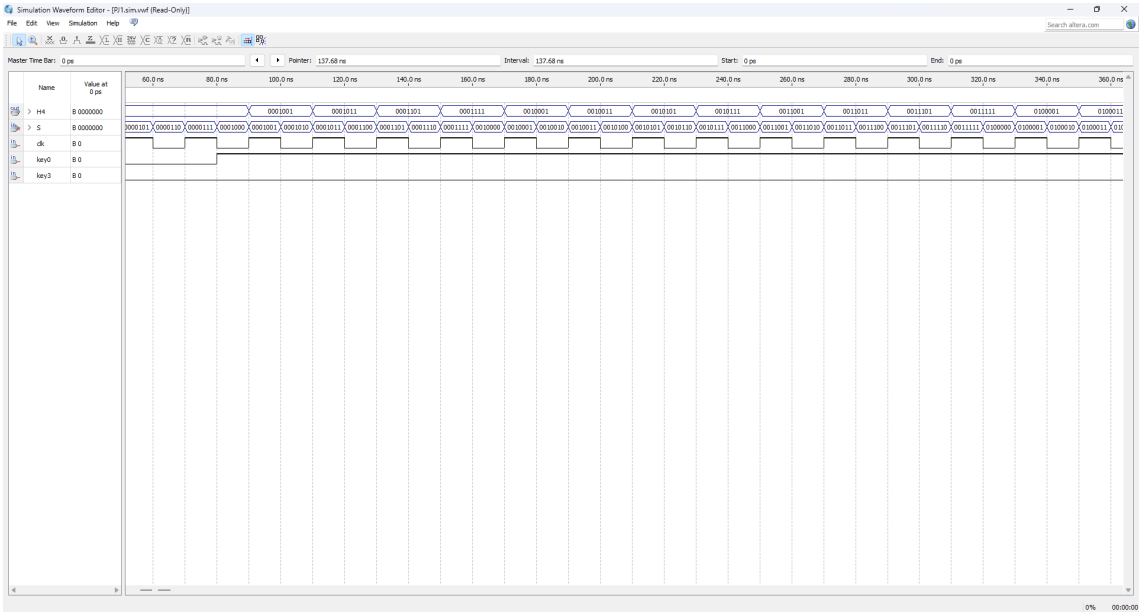

Figura 1. Recorder Waveform

2.2. Tips

Posteriormente, foi criado um módulo que realizava quatro necessidades básicas do funcionamento da máquina:

1. A geração de uma **senha aleatória**
2. A conferência de **igualdade** entre a tentativa do usuário e a senha aleatória
3. A formulação da **dica 1**
4. A formulação da **dica 2**

A geração da senha aleatória foi baseada em um registrador de 7 bits com clock de 50MHz, próprio da placa, garantindo que a senha fosse imprevisível a cada nova execução do sistema. Mesmo em consideração à velocidade do clock, a variável ”senha”definida para ser a senha sorteada foi pré-definida igual à permutação entre 0’s e 1’s para não existir uma inicialização no total de 0’s. A senha aleatória era armazenada toda vez que a chave (KEY0) era ativada.

A conferência de igualdade entre as tentativas do usuário e a senha aleatória gerada foi elaborada de forma simples. Apenas foi estabelecido um algoritmo condicional, em que conferia se a tentativa era igual à senha. Nisso, era gerado um sinal de 1 bit, onde 1’b0 era a saída negativamente e 1’b1 positivamente.

A função da **dica 1** era sortear uma das seis funções booleanas: AND, OR, NAND, NOR, XOR e XNOR e exibir o resultado no display HEX6. Para isso, **a formulação da dica 1** foi a utilização de dois registradores, em que o primeiro era um registrador de 3 bits, onde era armazenado um valor toda vez que a chave (KEY3) era ativada. O princípio dessa formulação foi a utilização da ULA (Unidade de Lógica e Aritmétrica), onde cada função foi designada por:

Binário	Função Boolena
000	AND
001	OR
010	NAND
011	NOR
100	XOR
101	XNOR

Tabela 1. Designação de Binário para Função

E o segundo registrador de 7 bits, onde era armazenado o resultado da operação booleana e exibido no display HEX6.

A função da **dica 2** era exibir nos LEDGs a quantidade de bits acertados na atual tentativa do usuário. Para isso, **a formulação da dica 2** foi a utilização de um registrador de 4 bits, que funcionava como uma espécie de contador. Nisso, usando um algoritmo de iteração, a senha e a tentativa eram comparadas bit a bit e a soma das suas igualdades era armazenada no registrador de 4 bits, sendo a quantidade exibida nos LEDGs.

Dessa forma, a função geral do módulo **Tips** era gerar a senha aleatória e fazer a comparação das tentativas. Caso a tentativa atual do usuário estivesse errada, o sistema fornecia as duas dicas. Além disso, era enviada a saída ”0”para o próximo módulo. Para a situação de igualdade entre a tentativa atual do usuário, era enviada a saída ”1”. Conforme a descrição, observa-se a seguir:

```
    module tips(
    input reg clk,
    input reg key0,
    input reg key3,
    input reg [6:0] S,
    output reg [6:0] LD,
    output reg [6:0] H6,
    output logic match_signal
);

    reg [6:0] senha = 7'b1010101;
    reg [3:0] acertos;
    reg [2:0] op;
    reg [6:0] resultado;Des

    always_ff @(posedge clk) begin
```

```
        if (~key0)
            senha <= (senha + 7'd3) & 7'b1111111;
    end

  always_ff @(posedge clk) begin
        if (S == senha)
            match_signal <= 1'b1;
        else
            match_signal <= 1'b0;
    end

    always_ff @(posedge clk) begin
        if (~key3) begin
            acertos = 4'd0;
            for (int i = 0; i < 7; i++) begin
                if (S[i] == senha[i])
                    acertos = acertos + 4'd1;
            end

            LD = (7'b0000001 << acertos) - 7'b0000001;

            op = (op + 3'd1) % 3'd6;

            case (op)
                3'b000: resultado = senha & S;
                3'b001: resultado = senha | S;
                3'b010: resultado = ~(senha & S);
                3'b011: resultado = ~(senha | S);
                3'b100: resultado = senha ^ S;
                3'b101: resultado = ~(senha ^ S);
                default: resultado = 7'b0000000;
            endcase

            H6 = resultado;

            if (S == senha) begin
                LD = 7'b1111111;
                H6 = 7'b1111111;
            end
        end
    end

endmodule
```

Tendo em vista a modularização da máquina de estados, foi possível o teste de funcionamento exclusivo do módulo, o qual pode ser vistos pelo vídeo Dica 1 e 2.

2.3. Timer

Para **a temporização**, foi implementado um contador decrescente de 4 minutos, 59 segundos e 9 décimos de segundo, utilizando um divisor de frequência de tal forma que o clock padrão da placa de 50MHz fosse divido e fosse fornecido o clock necessário para o contador padrão dos minutos,das dezenas de segundos, dos segundos e dos milésimos de segundos.

Apesar da ideia ter sido voltada para a inicialização do temporizador e da máquina de estados em si após a ativação da chave (KEY0), por alguma razão não fundamentada, o temporizador era o único que não obedecia a essa inicialização. Por isso, assim que a placa recebia o programa .sof, o temporizador era inicializado.

Basicamente, com o contador de 32 bits que englobava em decimal o valor de ciclos, os minutos começavam pelo valor de 4 minutos, 59 segundos e 9 décimos de segundos e eram decrementados a cada contagem de 5.000.000 de pulsos do clock padrão da placa. Os valores atuais dos tempos eram armazenados em quatro registradores de 4 bits, pois os seus valores eram mudados periodicamente. Ademais, foi feito um registrador de 1 bit chamado **running**, que a depender de duas situações: o tempo chegasse a 0:00:0 ou o sinal recebido do módulo **Tips** fosse "1", o temporizador era parado.

A exibição do tempo foi feita por meio de um encoder, em que era recebido o valor em decimal de cada tempo e convertido para o equivalente binário para o display de 7 segmentos. Apesar do código responder corretamente aos minutos, às dezenas de segundos e aos décimos de segundos, por alguma falha da placa, o segmento 6 do display HEX1, correspondente às unidades de segundos, falhava quando era necessário. Conforme a descrição, observa-se a seguir:

```
    module timer (
    input wire clk,
    input wire reset,
    input wire stop_signal,
    output reg [6:0] H3,
    output reg [6:0] H2,
    output reg [6:0] H1,
    output reg [6:0] H0
);


    reg [31:0] counter;
    reg [3:0] min, sec_high, sec_low, dec;
    reg running;

    always_ff @(posedge clk or negedge reset) begin
        if (!reset) begin
            counter <= 32'd0;
            min <= 4'd4;
            sec_high <= 4'd5;
            sec_low <= 4'd9;
            dec <= 4'd9;
```

```
        running <= 1'b1;
    end else if (!stop_signal && running) begin
        counter <= counter + 1;
        if (counter >= 32'd5000000) begin
            counter <= 32'd0;
            if (min==4'd0 && sec_high==4'd0 && sec_low==4'd0 &&
            dec==4'd0)
            begin
                running <= 1'b0;
            end else begin
                if (dec == 4'd0) begin
                    dec <= 4'd9;
                    if (sec_low == 4'd0) begin
                        sec_low <= 4'd9;
                        if (sec_high == 4'd0) begin
                            sec_high <= 4'd5;
                            if (min != 4'd0)
                                min <= min - 4'd1;
                        end else begin
                            sec_high <= sec_high - 4'd1;
                        end
                    end else begin
                        sec_low <= sec_low - 4'd1;
                    end
                end else begin
                    dec <= dec - 4'd1;
                end
            end
        end
    end
end

function automatic [6:0] dec_to_7seg(input reg [3:0] val);
    case (val)
4'd0: dec_to_7seg = ~7'b0111111;
        4'd1: dec_to_7seg = ~7'b0000110;
        4'd2: dec_to_7seg = ~7'b1011011;
        4'd3: dec_to_7seg = ~7'b1001111;
        4'd4: dec_to_7seg = ~7'b1100110;
        4'd5: dec_to_7seg = ~7'b1101101;
        4'd6: dec_to_7seg = ~7'b1111101;
        4'd7: dec_to_7seg = ~7'b0000111;
        4'd8: dec_to_7seg = ~7'b1111111;
        4'd9: dec_to_7seg = ~7'b1101111;
        default: dec_to_7seg = ~7'b0000000;
    endcase
```

```
    endfunction

    always_ff @(posedge clk) begin
        if (!running) begin
            H3 <= 7'b1111111;
            H2 <= 7'b1111111;
            H1 <= 7'b1111111;
            H0 <= 7'b1111111;
        end else begin
            H3 <= dec_to_7seg(min);
            H2 <= dec_to_7seg(sec_high);
            H1 <= dec_to_7seg(sec_low);
            H0 <= dec_to_7seg(dec);
        end
    end

endmodule
```

Tendo em vista a modularização da máquina de estados, foi possível o teste de funcionamento exclusivo do módulo, o qual pode ser visto pelo vídeo Timer .

2.4. Projeto Final

Embora o funcionamento dos três módulos fosse executado separadamente, a condição de parada do sistema era única: **a tentativa do usuário ser igual a senha aleatória**. Por conta disso, a sequência de entradas e saídas dos módulos é voltada para a condição da saída da variável expressa em 1 bit conectada ao temporizador. Segue a sequência da ilustração do circuito em Block Diagram/Schematic File:

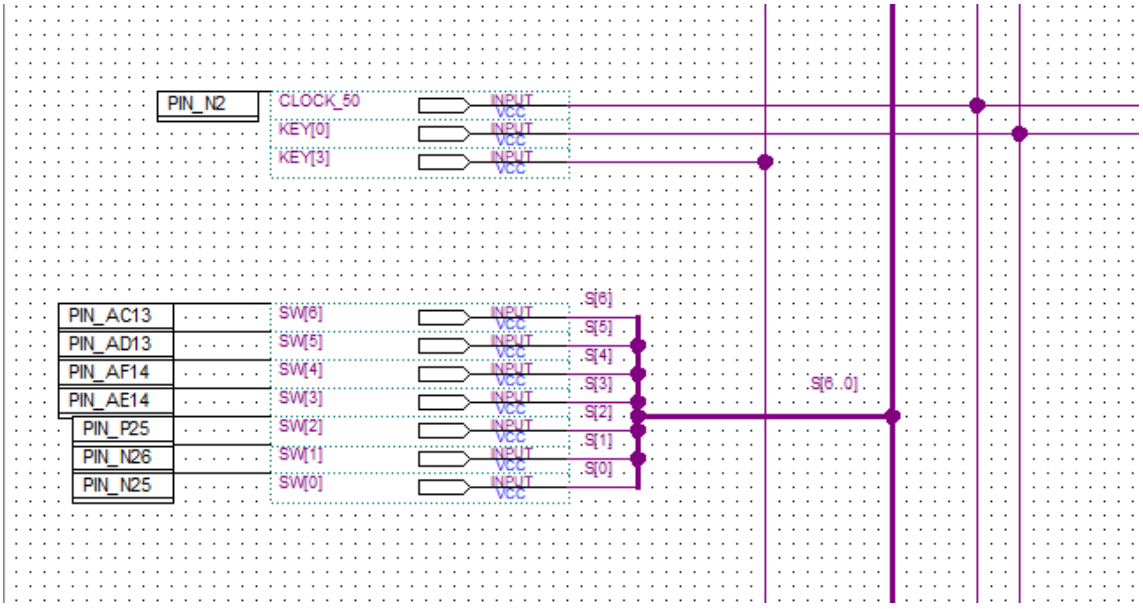

Figura 2. Entradas, Chaves e Clock do Sistema

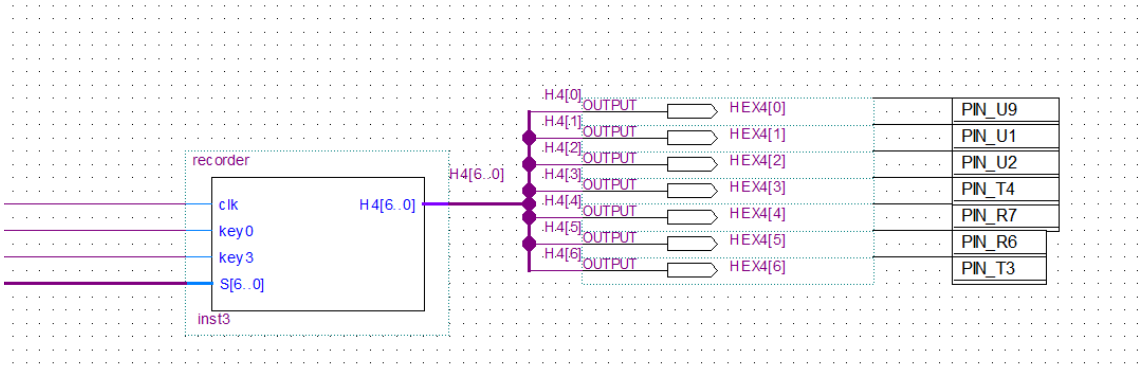

Figura 3. Módulo Recorder

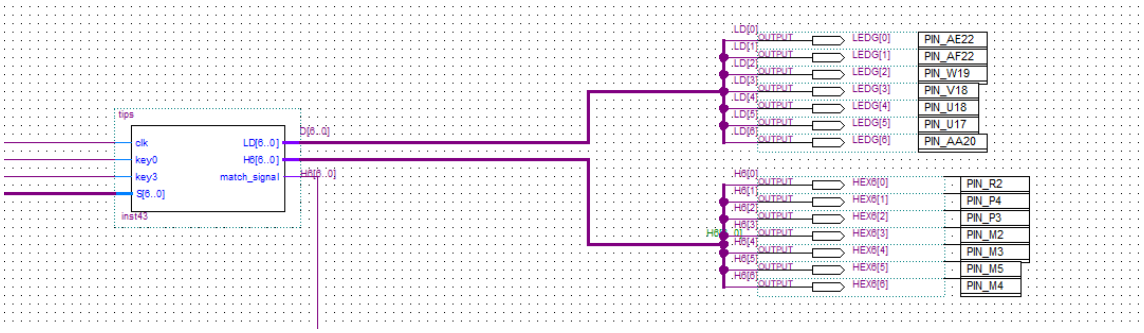

Figura 4. Módulo Tips

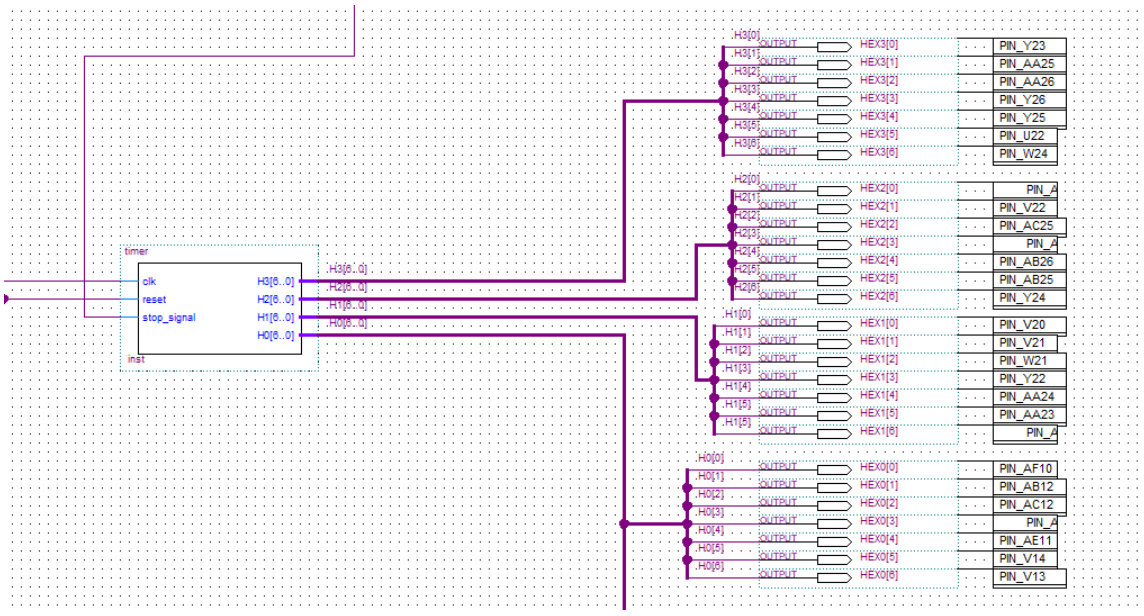

Figura 5. Módulo Timer

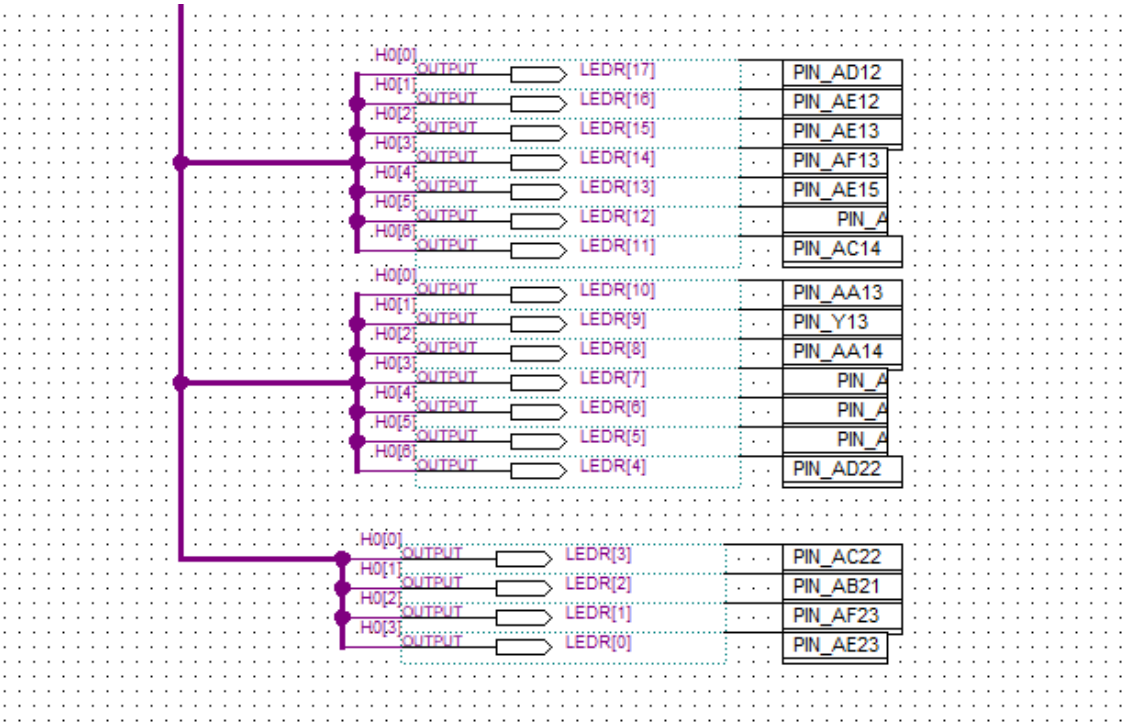

Figura 6. Saidas dos LEDRs

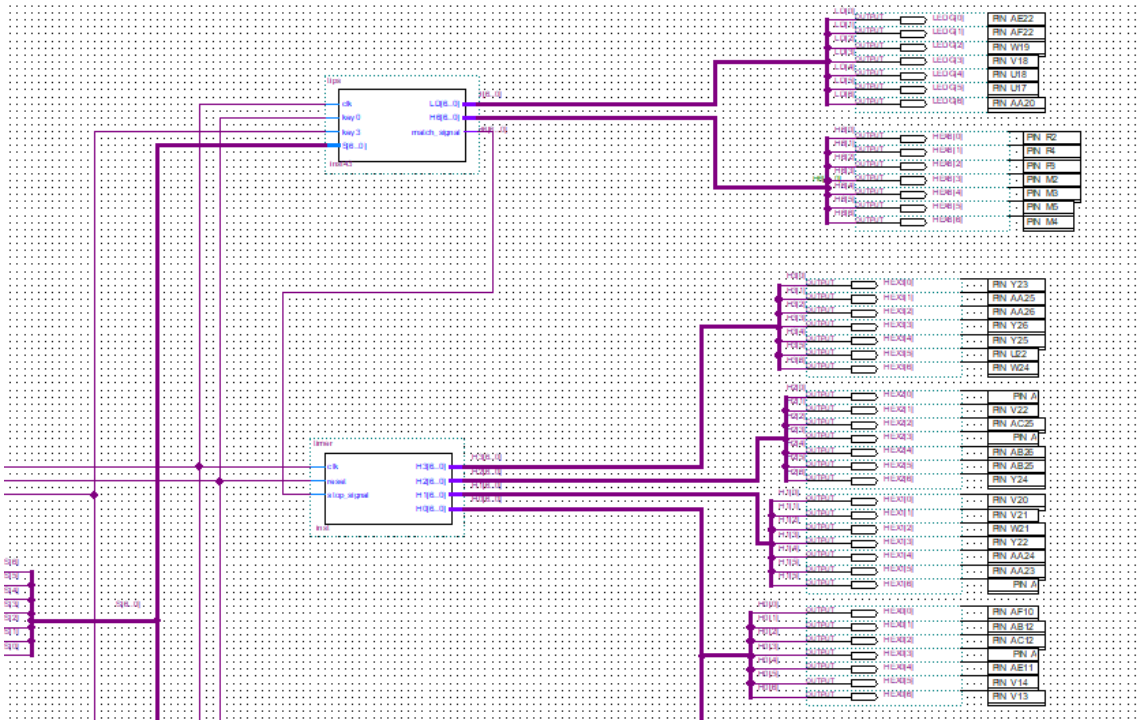

Figura 7. Ilustração da Conexão entre Tips e Timer

Com os módulos conectados, o jogo prosseguia até que a senha correta fosse inserida ou o tempo se esgotasse. No caso de sucesso, todos os LEDs verdes permaneciam

acendidos, o temporizador era congelado, o display HEX6, que fornecia a dica 1, era apagado e as LEDRs piscantes eram paradas. Caso o tempo terminasse sem a inserção correta da senha, a ”bomba”indicava a concretização da sua função, gerando a ativação sem pulso de todos os LEDs vermelhos e dos displays HEX.

A ideia do uso dos LEDRs era soar como se a ”bomba”estivesse emitindo dados e que a função do usuário fosse interromper essa transmissão, por isso os LEDRs paravam na posição atual que estavam quando ocorria o acerte. Porém se o objetivo da ”bomba”fosse concretizado, os dados seriam todos passados, sendo o sucesso indicado pela presença de todos os LEDRs acesos e parados.

O funcionamento do projeto final pode ser visto em dois vídeos, em que o primeiro mostra a situação sem sucesso e o segundo a situação com sucesso.

1. Sem sucesso
2. Com sucesso

3. Conclusões

A implementação do sistema proposto permitiu consolidar diversos conceitos de lógica digital, incluindo temporização, operações booleanas e interação com periféricos. A divisão do projeto em módulos independentes possibilitou uma abordagem estruturada e facilitou a depuração do código.

Durante os testes, observou-se que o funcionamento do sistema estava de acordo com os requisitos gerais especificados, garantindo a interatividade esperada no jogo. A adição do contador e da geração de dicas enriqueceu a experiência do usuário, tornando o projeto mais desafiador. Por fim, a experiência proporcionou um aprofundamento significativo na utilização de FPGAs para aplicações práticas envolvendo lógica sequencial e combinacional, e principalmente a conquista de intermediária habilitação na linguagem de descrição SystemVerilog.

Referências

[Corporation 2013] Corporation, I. (2013). Quartus ii version 13.0 sp1. [Online; accessed 12-December-2024].

[Systemverilog 2016] Systemverilog (2016). Systemverilog — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=SystemVerilog&oldid=728456019. [Online; accessed 9-July-2016].