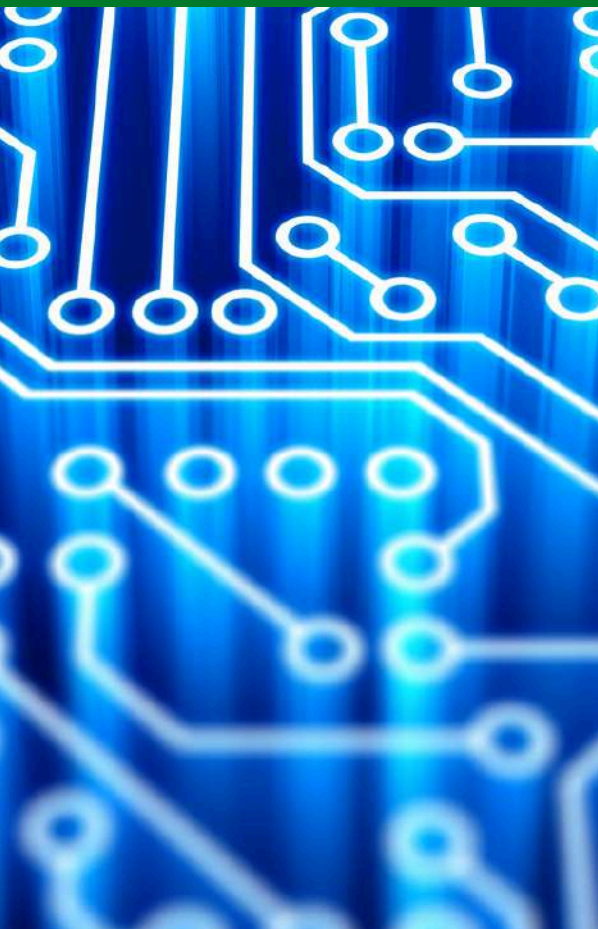




BOMBA RELÓGIO

Projeto Aplicativo



GRUPO C5 :

ANA LUIZA GOMES DE OLIVEIRA
JULIA PAULO AMORIM
LETÍCIA GONÇALVES BOMFIM



UnB | CIC

RESUMO

O projeto final da matéria “Laboratório de Circuitos Lógicos” consistiu em projetar, simular e implementar um sistema digital baseado em FPGA. Para que, assim, os alunos fizessem a implementação de um jogo digital de Bomba Relógio.

Essa implementação deveria ser feita utilizando as chaves (SW), botões (KEY), LEDRs, LEDGs e displays hexadecimais (HEX0 a HEX7).

O funcionamento do projeto aplicativo começa ao pressionar Start o sistema sorteia uma senha que fica oculta ao jogador, e inicializa um contador regressivo de 5 minutos.

O jogador deve entrar com uma tentativa de senha e pressionar o ENTER para validar. Se a senha registrada for a mesma da tentativa do jogador, o contador de tempo é congelado e a bomba foi desarmada e o cofre aberto. Se o jogador errar a senha, o sistema avisa-o fornecendo duas dicas, a primeira dica é o resultado de uma operação lógica sorteada pelo sistema, no qual a cada nova tentativa uma nova operação lógica é sorteada. A segunda dica é o número de dígitos corretos entre a senha real e a senha inserida pelo jogador.

O jogo prossegue até o jogador acertar ou o tempo se esgotar, quando ocorre a explosão da bomba.

ABSTRACT

The final project of the subject “Logical Circuits Laboratory” consisted of designing, simulating and implementing a digital system based on FPGA. So that the students could implement a digital Time Bomb game.

This implementation should be done using keys (SW), buttons (KEY), LEDRs, LEDGs and hexadecimal displays (HEX0 to HEX7).

The operation of the application project begins when pressing Start, the system draws a password that is hidden from the player, and starts a 5-minute countdown timer.

The player must enter a password attempt and press ENTER to validate. If the registered password is the same as the player's attempt, the time counter is frozen and the bomb has been defused and the safe opened. If the player gets the password wrong, the system warns him by providing two tips, the first tip is the result of a logical operation drawn by the system, in which with each new attempt a new logical operation is drawn. The second tip is the number of correct digits between the real password and the password entered by the player.

The game continues until the player gets it right or the time runs out, when the bomb explodes.

INTRODUÇÃO

Por fim, o projeto final. No decorrer da matéria de Laboratório de Circuitos Lógicos, os estudantes foram instruídos a projetar e implementar diversas funcionalidades que levaram ao conhecimento necessário para confecção do projeto final. No semestre 2 de 2024, o projeto foi um jogo digital de bomba relógio, sendo assim, seu objetivo era que os alunos projetassem, simulassem e implementassem um sistema digital baseado em FPGA, para a implementação do jogo.

Para sua implementação foram usadas as chaves (SW), botões(KEY), LEDRs, LEDGs e displays hexadecimal (HEX0 a HEX7), e os materiais: Kit de Desenvolvimento DE2 e Programa Quartus-II v.13.0. A história do jogo é: O jogador gostaria de abrir um cofre que está trancado por uma senha aleatória de 7 bits, assim que ele tentar a primeira senha, um contador se inicia até a explosão do cofre, dele e dos bens que o mesmo planejava se apossar

O jogo se inicia ao apertar Start, o sistema sorteia uma senha de 7 bits e o jogador possui 4 minutos e 59 segundos antes que a bomba exploda. Após isso, temos duas opções. A primeira, se a senha registrada pelo jogador estiver correta, o tempo é congelado e a primeira luz verde da direita para a esquerda se acende, indicando que a bomba foi desarmada.

Entretanto, se o jogador errar a senha, ele receberá duas dicas. Uma sendo uma dica aleatória utilizando uma porta lógica sorteada pelo sistema que aparece no display, e outra aparece em forma de LEDs vermelhos, que mostram a quantidade de dígitos corretos na senha. O jogo prossegue até o jogador acertar a senha ou até o tempo se esgotar. Se de tudo o jogador não acertar a senha dentro do tempo, a bomba explode e a segunda luz da direita para a esquerda se acende, indicando que a bomba explodiu.

PROCEDIMENTOS E RESULTADOS

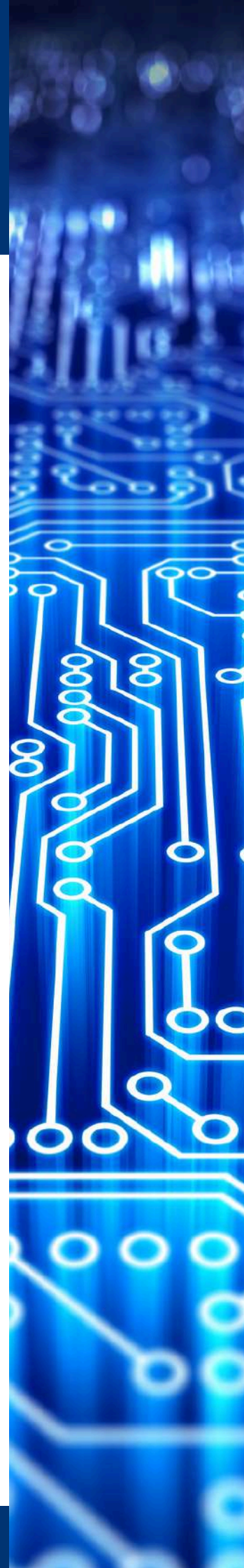
Chaves e módulos

No nosso projeto, o LEDR[1] na placa FPGA estava fraco, semi queimado, de modo que no vídeo não fica muito claro quando está ligado de fato. Além disso, o reset ficou definido como SW[0], o start como SW[1]. o enter como SW[2], as dicas aparecem no HEX5.

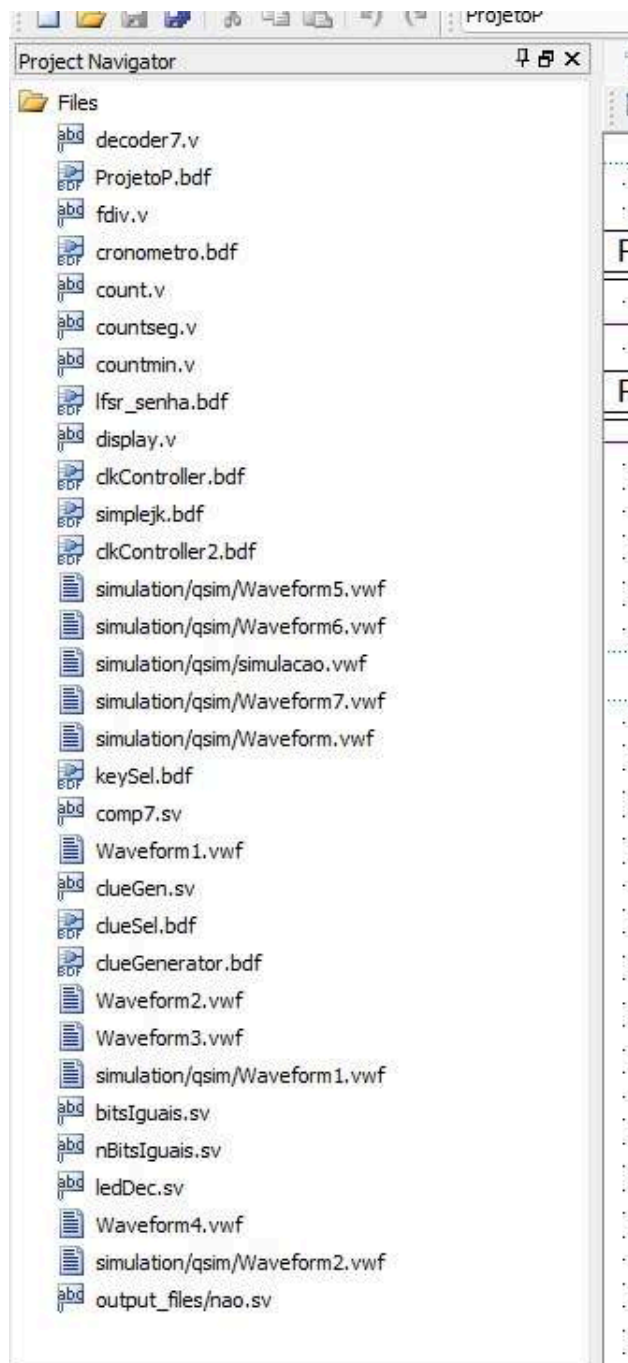
Quando o jogador acerta a senha o LEDG[0] acende indicando que a bomba foi desarmada. Em contrapartida, quando a bomba explode, o LEDG[1] acende. O cronômetro ficou definido nos HEX[3..0] e a dica de quantos números estão certos aparece nos LEDR[6..0], quanto mais números certos, mais LEDs ficam acesos até que a senha seja acertada.

Abaixo explicaremos a funcionalidade de cada uma das “caixinhas” criadas para o projeto. A primeira imagem dessa lista que aparece mostra todos os circuitos projetados para que o circuito principal obtivesse o resultado esperado do jogo da bomba relógio. O último a ser explicado será o ProjetoP.bdf, pois ele contém a versão “final” do jogo.

As funcionalidades relacionadas ao cronômetro e às dicas para as senhas serão explicadas junto com o circuito do projeto final, então serão puladas na explicação que vem logo em sequência.



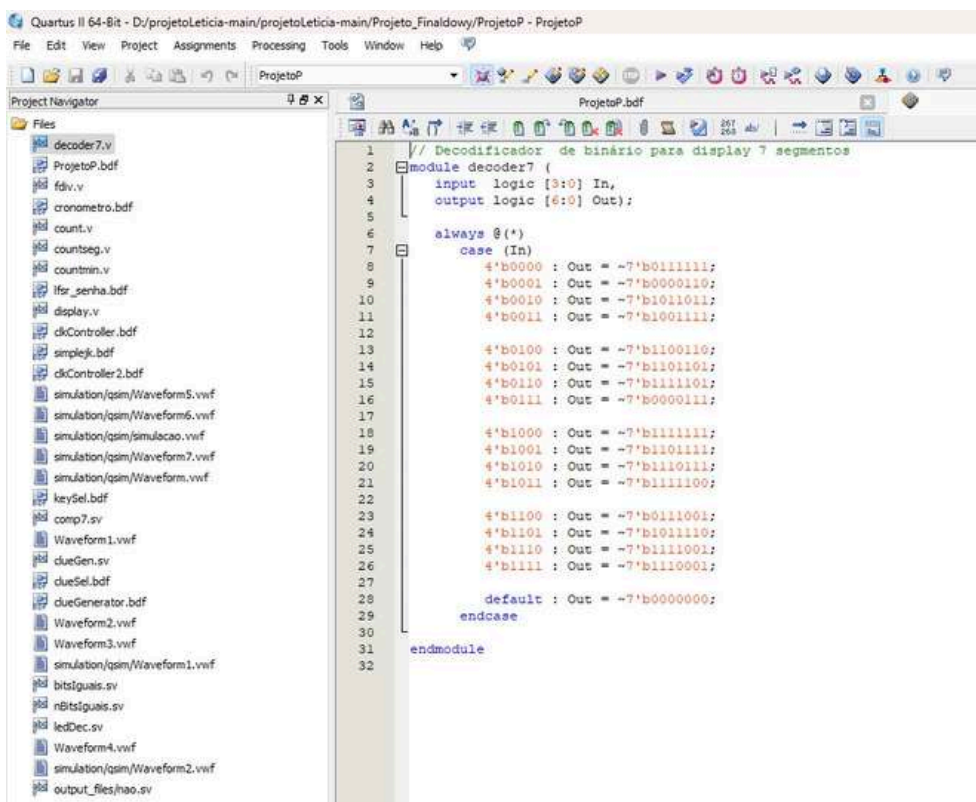
PROCEDIMENTOS E RESULTADOS



PROCEDIMENTOS E RESULTADOS

Em ordem, a primeira “caixinha” foi chamada de decoder7. Ela é um decodificador feito em system verilog, convertendo de binário para o display de 7 segmentos. A entrada (In) é uma variável de 4 bits ([3:0]) que representa um número binário, variando de 0 a 15. Já a saída (out) é uma variável de 7 bits ([6:0]) que controla o estado de cada um dos 7 segmentos de um display.

O bloco always @(*) define a lógica combinacional, ou seja, a saída depende apenas da entrada, sem qualquer dependência de sinais de clock. Assim, o código vai verificar o valor de In e, de acordo com esse valor, atribuir um valor correspondente à variável Out. Já os cases definem qual entrada vai gerar a saída correspondente no display.



```
1 // Decodificador de binário para display 7 segmentos
2 module decoder7 (
3     input logic [3:0] In,
4     output logic [6:0] Out);
5
6     always @(*)
7     case (In)
8         4'b0000 : Out = ~7'b0111111;
9         4'b0001 : Out = ~7'b0000110;
10        4'b0010 : Out = ~7'b1011011;
11        4'b0011 : Out = ~7'b1001111;
12
13        4'b0100 : Out = ~7'b1100110;
14        4'b0101 : Out = ~7'b1101101;
15        4'b0110 : Out = ~7'b1111101;
16        4'b0111 : Out = ~7'b0000111;
17
18        4'b1000 : Out = ~7'b1111111;
19        4'b1001 : Out = ~7'b1101111;
20        4'b1010 : Out = ~7'b1111011;
21        4'b1011 : Out = ~7'b1111100;
22
23        4'b1100 : Out = ~7'b0111001;
24        4'b1101 : Out = ~7'b1011110;
25        4'b1110 : Out = ~7'b1111001;
26        4'b1111 : Out = ~7'b1111001;
27
28        default : Out = ~7'b0000000;
29    endcase
30 endmodule
31
32
```


PROCEDIMENTOS E RESULTADOS

A próxima é a fdiv, seu código Verilog implementa um módulo que tem como objetivo gerar um sinal de clock (clkout) com uma frequência mais baixa a partir de um sinal de clock de entrada (clkin). Isso é feito utilizando um contador (cont) para dividir a frequência de clkin.

Assim o count é implementado +1 a cada borda de subida do clock. Quando o contador atinge o valor de 2.500.000, o sinal de clock de saída é invertido e o contador é reiniciado para 0.

```
1  module fdiv(  
2      input clkin,  
3      output reg clkout  
4  );  
5    
6      integer cont;  
7      initial cont=0;  
8    
9      always @ (posedge clkin)  
10     begin  
11         if(cont==2500000)  
12             begin  
13                 cont<=0;  
14                 clkout<=~clkout;  
15             end  
16         else  
17             begin  
18                 cont<=cont+1;  
19             end  
20         end  
21     end  
22 endmodule
```

PROCEDIMENTOS E RESULTADOS

As próximas três caixinhas são os contadores de minutos, segundos e décimos de segundos, que juntos formam o cronômetro decrescente da bomba. Todos são implementados em verilog e possuem a estrutura muito parecida. Para a explicação, utilizarei apenas o contador de minutos.

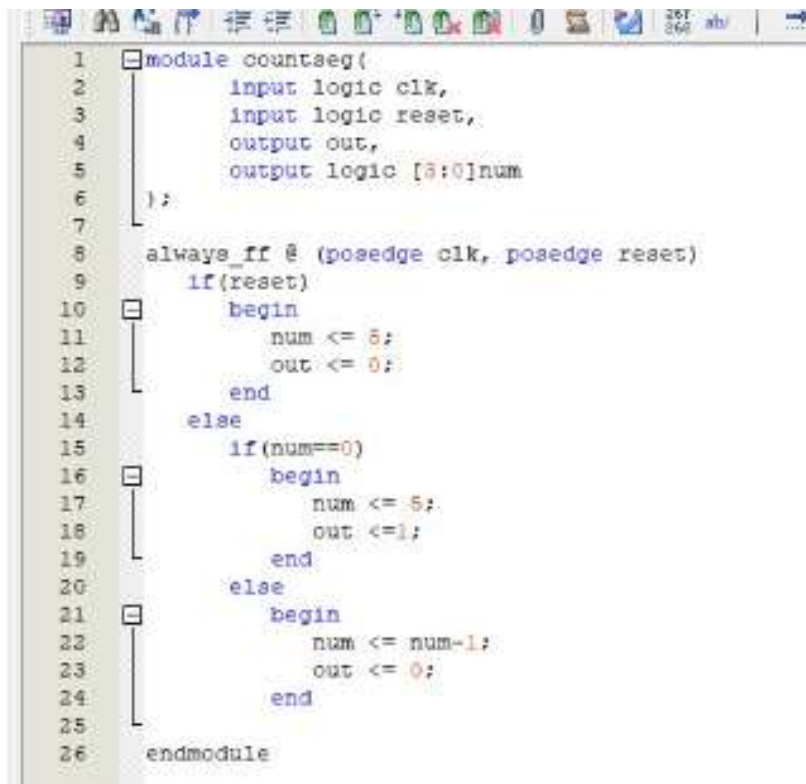
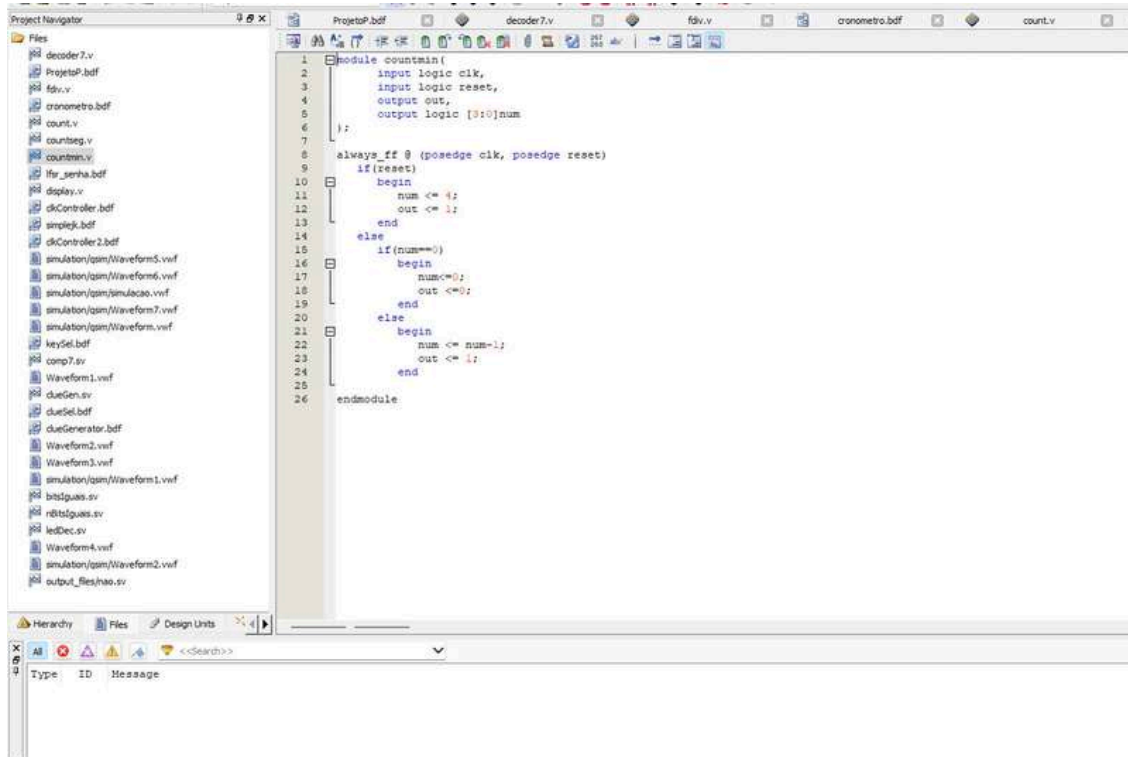
Esse contador tem o objetivo de contar de forma decrescente um número de 4 bits (representado por num) e gerar um sinal de saída (out) baseado nesse contador. As entradas são o clock (clk) que controla quando as operações de contagem e controle acontecem e reset que reinicia o contador quando acionado.

Enquanto isso, as saídas são o out que indica quando a contagem foi concluída ou não e o num, que armazena o valor da contagem.

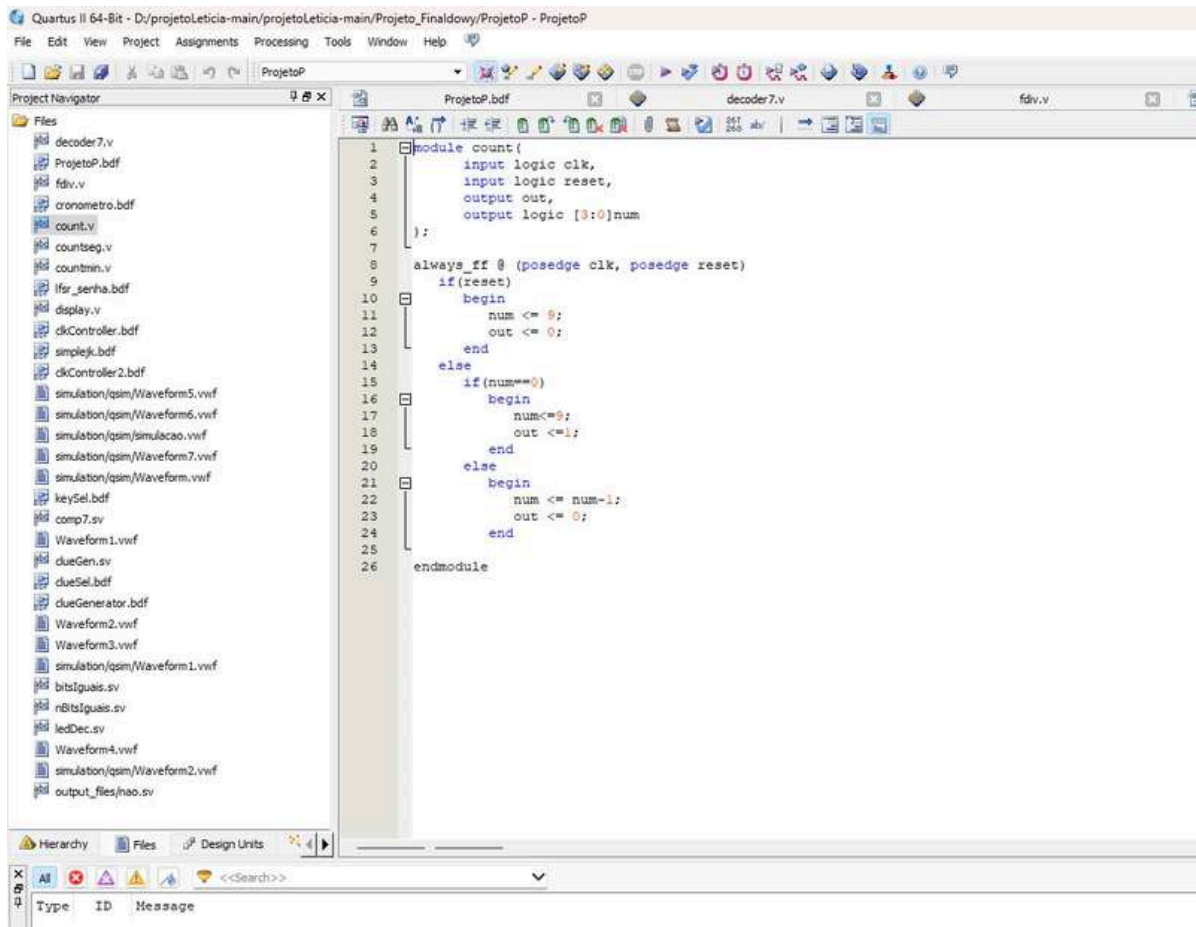
O módulo tem como objetivo contar de forma decrescente o valor armazenado em num a cada ciclo de clock. Quando o contador atinge zero, o sinal out é alterado para indicar que a contagem terminou.

O bloco `always_ff` é usado para descrever a lógica sequencial, que é sensível à borda de subida do clock (`posedge clk`) ou ao sinal de reset assíncrono (`posedge reset`). Ou seja, o bloco será executado sempre que houver uma transição de borda de subida no clock ou quando o reset for ativado.

PROCEDIMENTOS E RESULTADOS



PROCEDIMENTOS E RESULTADOS



Em seguida, temos o display. O LFSR é um tipo de registrador de deslocamento onde a cada ciclo de clock, os bits do registro são deslocados e o novo bit de entrada é calculado usando uma função linear de feedback. Suas entradas são o clock e o reset, e a saída é o LFSR em si, Este módulo implementa um LFSR de 7 bits com feedback baseado na operação XOR entre os bits mais significativos. Ele gera uma sequência aleatória de 7 bits.

O valor do LFSR é atualizado a cada ciclo de clock, e o valor de saída é o valor atual do LFSR. Quando o sinal reset é ativado, o valor do LFSR é reiniciado. O comportamento de feedback garante que a sequência gerada seja aleatória, sendo utilizada como geradores de números aleatórios.

PROCEDIMENTOS E RESULTADOS

```
1 module display(  
2     input wire clock,  
3     input wire reset,  
4     output reg [6:0] lfsr  
5 );  
6  
7     wire feedback = lfsr[6] ^ lfsr[5];  
8  
9     always @ (posedge clock) begin  
10         if(reset)  
11             begin  
12                 lfsr <= 7'h1;  
13             end  
14         else  
15             begin  
16                 lfsr <= {lfsr[5:0], feedback};  
17             end  
18         end  
19     end  
20 endmodule
```

O próximo é o comp7, que consiste em um comparador simples de 7 bits. Ele verifica se os dois valores de entrada ans e usr são iguais e gera um sinal de saída eq que indica se a comparação é verdadeira (1) ou falsa (0). O código utiliza a operação de comparação == para fazer essa verificação de igualdade.

```
module comp7(  
    input logic [6:0] ans,  
    input logic [6:0] usr,  
    output logic eq  
);  
    assign eq = (ans == usr);  
endmodule
```


PROCEDIMENTOS E RESULTADOS

O módulo clue7 realiza diferentes operações lógicas entre key e usr, selecionadas pelo valor de clueSel. O valor resultante da operação é armazenado em dica. Dependendo de clueSel, ele pode realizar operações como AND, OR, NAND, NOR, XOR ou XNOR, com a lógica definida dentro do bloco case. O módulo utilizado em aplicações onde diferentes dicas precisam ser geradas com base em comparações lógicas.

```
module clue7(  
    input logic [6:0]key,  
    input logic [6:0]usr,  
    input logic [5:0]clueSel,  
    output logic [6:0]dica  
);  
always_comb begin  
    case (clueSel)  
        6'b000001: dica = key & usr;  
        6'b000010: dica = key | usr;  
        6'b000100: dica = ~(key & usr);  
        6'b001000: dica = ~(key | usr);  
        6'b010000: dica = key ^ usr;  
        default: dica = ~(key ^ usr);  
    endcase  
end  
endmodule
```

O módulo seguinte é o bitsIguais, que compara dois vetores de 7 bits (ans e usr) e gera um vetor de 7 bits chamado iguais. O vetor iguais terá valor 1 nas posições onde os bits de ans e usr são iguais, e 0 onde são diferentes.

Isso é feito utilizando uma operação XOR seguida de um complemento (~), o que inverte os resultados do XOR.

PROCEDIMENTOS E RESULTADOS

```
module bitsIguais(  
    input logic [6:0]ans,  
    input logic [6:0]usr,  
    output logic [6:0]iguais  
);  
    assign iguais = ~(ans ^ usr);  
endmodule
```

O módulo nBitsIguais conta o número de bits com valor 1 no vetor iguais. Em seguida, ele soma os bits de iguais, e o total é armazenado na saída nIguais. Por ter 3 bits, ele é permitido representar valores de 0 a 7, sendo essas as luzes vermelhas da placa.

```
module nBitsIguais(  
    input logic [6:0]iguais,  
    output logic [2:0]nIguais  
);  
    assign nIguais = iguais[0] + iguais[1] + iguais[2] + iguais[3] + iguais[4] + iguais[5] + iguais[6];  
endmodule
```

A caixinha ledDec converte um valor de 3 bits (num) para um valor de 7 bits (lds), onde cada bit de lds pode representar o estado de um LED. O mapeamento de num para lds é feito com uma estrutura case, que acende mais LEDs conforme o valor de num aumenta.

Quando num é 0, nenhum LED é aceso; quando num é 7, todos os LEDs são acesos. Ele usa um bloco always_comb para realizar uma operação combinacional, ou seja, a saída lds é calculada a partir de num sem depender de um sinal de clock. Dentro deste bloco, há uma estrutura case que mapeia o valor de num para um valor correspondente de lds. Cada valor de lds resulta em uma sequência de LEDs acesos.

PROCEDIMENTOS E RESULTADOS

```
module ledDec(  
    input logic [2:0]num,  
    output logic [6:0]lds  
);  
    always_comb begin  
        case (num)  
            0: lds = 0;  
            1: lds = 1;  
            2: lds = 3;  
            3: lds = 7;  
            4: lds = 15;  
            5: lds = 31;  
            6: lds = 63;  
            7: lds = 127;  
            default: lds = 0;  
        endcase  
    end  
endmodule
```

O módulo não recebe um vetor de 7 bits (in) e retorna o vetor out, que é o complemento de in. A operação de NOT é realizada bit a bit, invertendo os valores de cada bit de in.

```
module nao(  
    input logic [6:0]in,  
    output logic [6:0]out  
);  
    assign out = ~in;  
endmodule
```


PROCEDIMENTOS E RESULTADOS

Funcionamento do jogo

O funcionamento do projeto aplicativo começa ao pressionar Start (SW[1]) o sistema sorteia uma senha de 7 bits que fica oculta ao jogador, e inicializa um contador regressivo de 4 minutos e 59 segundos. O jogador deve entrar com uma tentativa de senha com as chaves da placa e pressionar o ENTER (SW[2]) para validar.

Se a senha registrada for a mesma da tentativa do jogador, o contador de tempo é congelado e o primeiro led verde acende (LEDG[0]), indicando que a bomba foi desarmada e o cofre aberto.

Se o jogador errar a senha, o sistema avisa-o fornecendo duas dicas, uma no display (HEX5) e outra nos leds vermelhos (LEDR[6..0]) mantendo a contagem regressiva. A primeira dica é o resultado de uma operação lógica sorteada pelo sistema (AND, OR, NAND, NOR, XOR ou XNOR), entre a senha real e a senha inserida pelo jogador, onde o número resultante é mostrado no display.

A segunda dica é o número de dígitos corretos entre a senha registrada e a tentativa do jogador, expressa como uma contagem progressiva em barra dos leds vermelhos. A cada nova tentativa uma nova operação lógica é sorteada entre as 6. O jogo prossegue até o jogador acertar ou o tempo se esgotar, quando ocorre a explosão da bomba. Para reiniciar o cronômetro e a tentativa, existe o Reset (SW[0]).

PROCEDIMENTOS E RESULTADOS

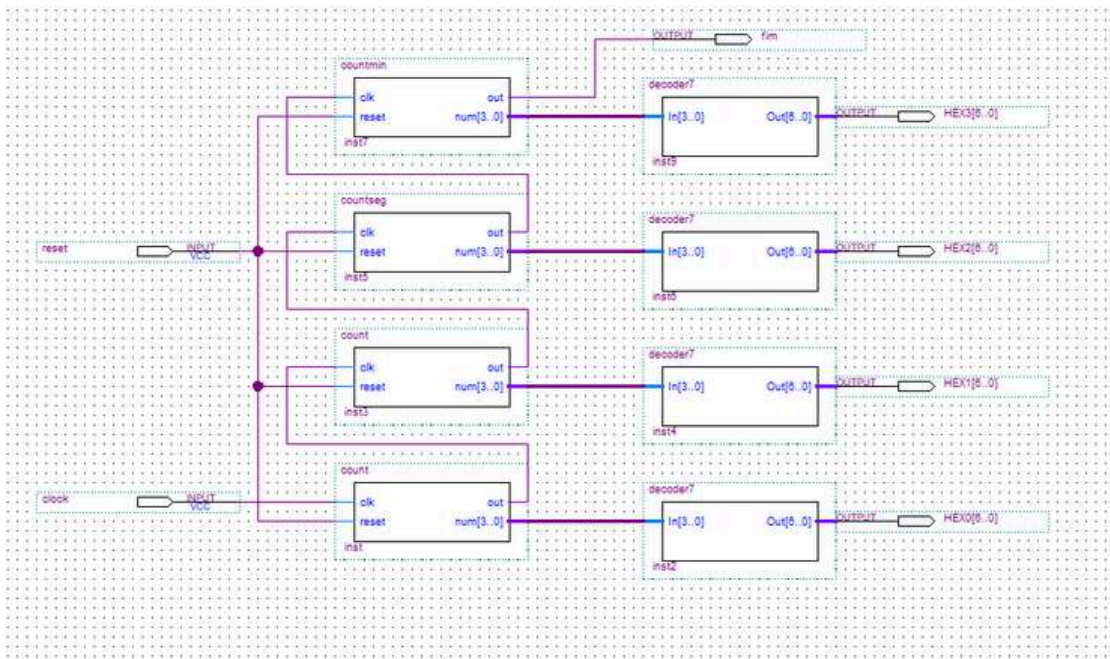
Cronômetro

A implementação do cronômetro começa com o uso da caixinha `clkController2` que serve para controlar o clock que chega até o cronômetro. O `clkFast` é o `CLOCK_50` que vem direto da placa por isso o nome fast, porque ele é o clock mais rápido; já o `clkOn` é o clock que inicia o funcionamento do cronômetro, sendo controlado pelo `SW[1](START)`, enquanto o `clkOff` é o fim ou pausa do cronômetro, sendo controlado pela operação or entre o `SW[0](RESET)` e o fim dos 5 minutos, por fim o `clkSlow` é o próprio `CLOCK_50` passado pelo `fdiv` de 2.500.000 Hz.

Todos esses sinais de clock entram no controller que tem o funcionamento de um flip flop JK. Onde se o sinal de `clockOn(start)` estiver em 1 e o `clockOff(reset/fim)` estiver 0, `Q` é 1. Já se o `Off` for 1 e `On=0`, `Q` é 0. Após isso o `Q` é conectado em um and junto com o `clkSlow`(passado pelo `fdiv`), que faz ele parar quando `Q=0` que é ligado o `reset/fim`, e finalmente temos o `output controlledClk`. o clock, que vai ser usado pelo cronômetro.

PROCEDIMENTOS E RESULTADOS

Dentro do cronômetro temos uma sequência de contadores, o primeiro que recebe o clock já controlado e um reset, sempre na subida de borda do clock ele diminui 1 da contagem que começa com 9 (décimos), quando ele atinge 0, ele manda um sinal de out que passa a ser o clock dos outros contadores da sequência. Durante isso, a cada número que ocorre a contagem, ele é jogado para uma saída de 4 bits que vai para o decoder7.v para sair nos displays de 7 segmentos. O reset faz com que eles voltem para o valor inicial. No último contador, o sinal de out indica quando acabou o cronômetro, o sinal sai negativo por isso colocamos o not no circuito para utilizar ele novamente depois.



PROCEDIMENTOS E RESULTADOS

Randomização de senha

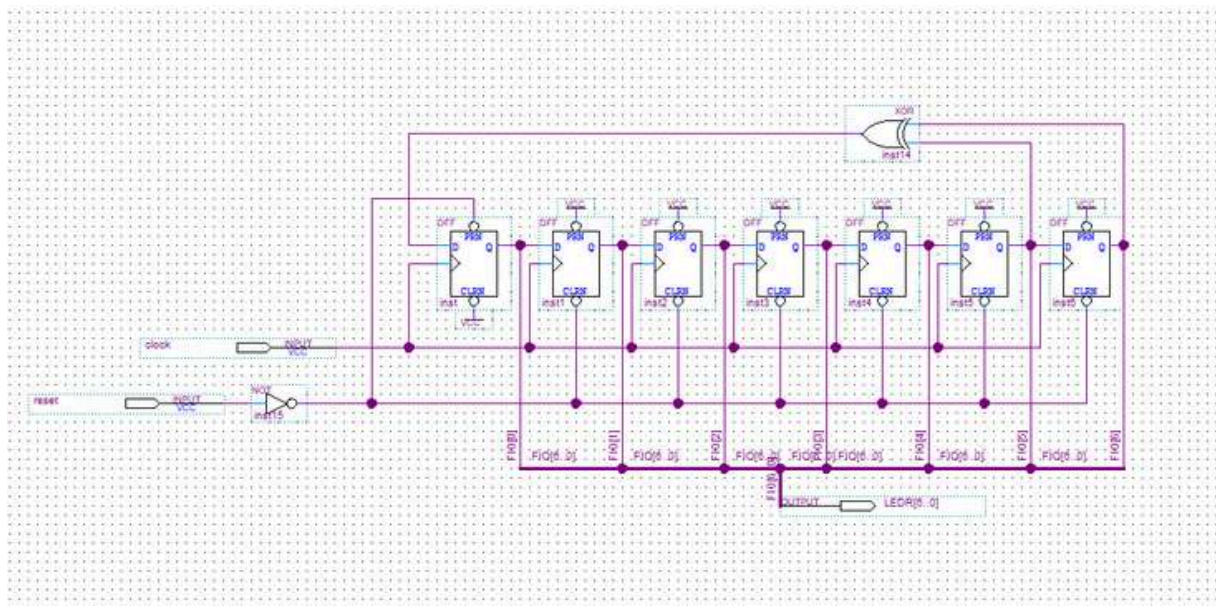
O primeiro passo da randomização da senha é a escolha da senha é a caixinha `lfsr_senha`. Ela recebe o `CLOCK_50` e o sinal de reset que servem para começar o processo do LFSR (Linear Feedback Shift-Register), registrador de deslocamento que possui realimentação entre as saídas de um conjunto de elementos de armazenamento e suas entradas, serve para gerar valores pseudo-aleatórios.

O que significa na prática que na primeira partida pelo reset não ter sido ligado, o valor da senha sempre vai ser o mesmo, já nas próximas partidas, ligando e desligando o reset, o LFSR a cada clock, troca o valor da senha, sendo um "contador" somando 1 bit ao valor de 1000000 em uma velocidade muito rápida e voltando.

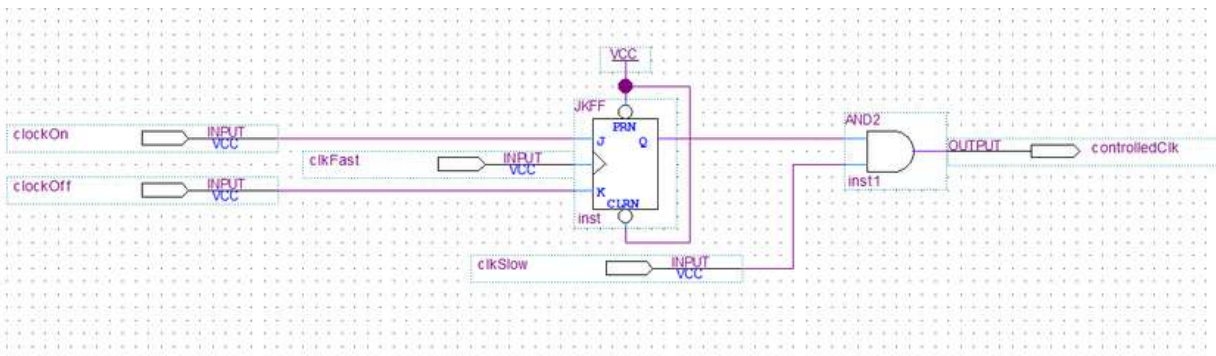
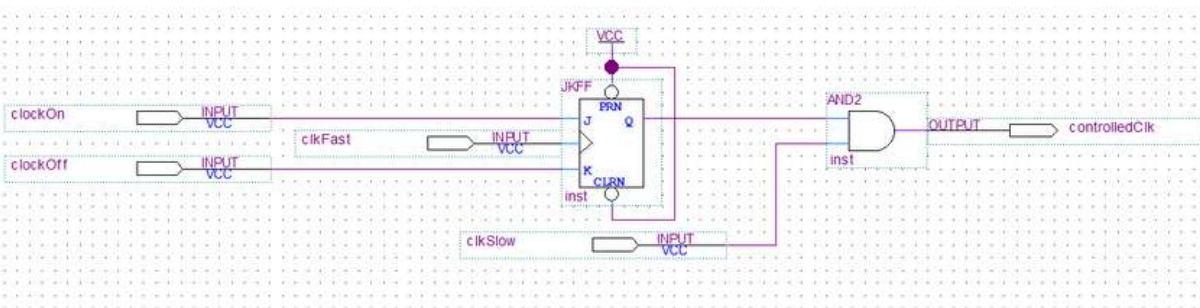
Foi implementado usando 7 flip flops D em sequência, sendo o Q de um, a entrada de D de outro, e para atingir o maior de nível de números possíveis foi colocado um xor entre o 6° e o 7° bit, de acordo com a tabela de registradores de shift. A saída é a senha de 7 bits, que é cada Q de cada flip flop.

Após a randomização a cada ciclo de clock, a senha ainda mudando constantemente passa para o `keySel`, que é o seletor da chave. o `keySel` recebe a senha e o `enabler(start)`, sendo seu circuito uma sequência de 7 flip flops D, onde o clock dos flip flops é o `enabler`, o que significa que os flip flops só guardam uma senha quando o jogador começar a jogar. Tendo finalmente na saída do seletor a senha salva que estava quando o jogador acionou o start.

PROCEDIMENTOS E RESULTADOS



Acima temos o circuito de randomização de senhas e abaixo os dois circuitos de CLK controller.



PROCEDIMENTOS E RESULTADOS

Funcionamento dos LEDs

LEDG[1] - Explosão

A saída fim do cronômetro, indica quando acaba os 5 minutos, a saída é 0 e há um not logo após para servir como entrada do simplejk. Esse checa se o K(reset) for 0 e o J(fim negado) for 1, o Q=1 e o LEDG[1] é acesso indicando a explosão da bomba

LEDG[0] - Ganhou

Após a senha ser selecionada pelo keySel, ela vai para o comp7, que foi escrito em verilog que checa se a tentativa do jogador for igual a senha, caso sim a saída eq(igual) é 1, ela é conectada a um and com a chave SW[2] (enter) e se ambos forem 1, vai para outro simplejk que só vê se o K(reset) é 0 e deixa o Q=1, que é o LEDG[0], indicando a vitória do jogador

LEDR[6..0] - Dicas

Após a senha ser selecionada pelo keySel, ela vai para bitsIguais, que passa bit a bit comparando a senha e a tentativa fazendo um xor negado, que faz com que toda vez que eles forem iguais a saída seja 1.

A saída dessa caixinha vai para nBitsIguais que faz a soma dos bits iguais gerando a saída de 3 bits nIguais. A qual vai para ledDec, onde é feito um switch case para cada quantidade de bits iguais, a saída lds é feita de acordo com $(2^n)-1$, onde $n=nIguais$, para acender a quantidade certa de leds vermelhos

PROCEDIMENTOS E RESULTADOS

Display e dica dos HEX

HEX5 - Seleção randômica

Para gerar a primeira, no display HEX5 é feito primeiro o clueGenerator que é uma sequência de flip flops D, onde a saída Q vira a entrada D do outro. A saída PRN do primeiro ff e as saídas Q dos outros vão para um or, que gera uma saída de 5 bits, a cada ciclo do CLOCK_50, fazendo um LFSR, um deslocamento de bit extremamente rápido.

Após isso a saída clue, que fica mudando, vai para o clueSel, uma sequência de flip flops D que salva a seleção no momento que o clock dos flip flops, que é o enabler, for 1. O enabler é ligado ao sinal de enter, ou seja, toda vez que o jogador fizer uma tentativa e colocar enter é salvo nos flip flops D uma seleção entre as que estavam mudando no clueGenerator.

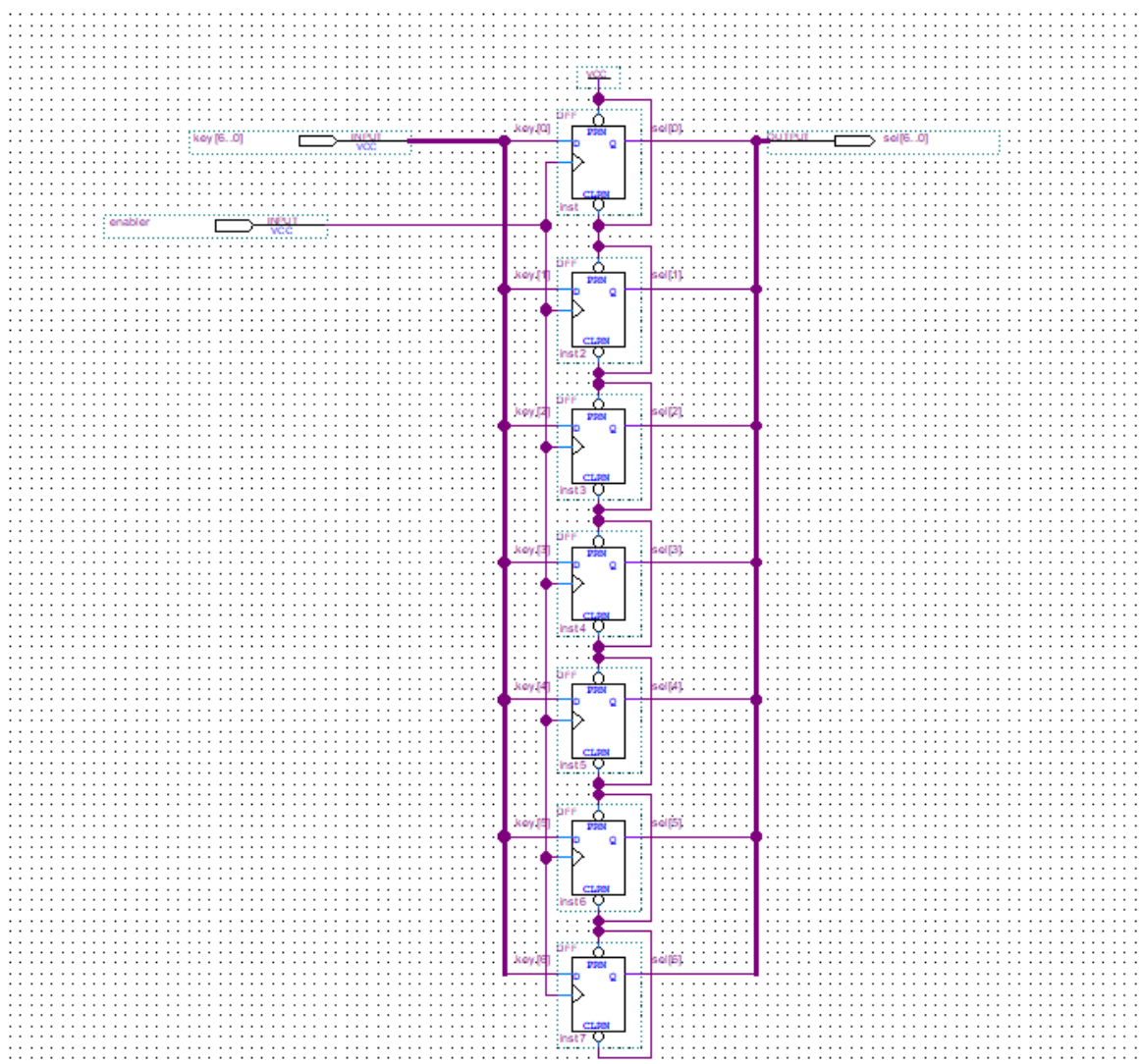
A saída sel de 5 bits vai para o clueGen que gera um switch case da seleção da dica, onde para cada opção é uma operação lógica diferente, já explicado anteriormente. Gerando assim a saída com a operação entre a senha e a tentativa e indo para o HEX5.

HEX4 - Tentativa

O HEX4 foi ligado diretamente as chaves SW[9..3], o que faz com que a cada mudança da tentativa do jogador, ela seja mostrada diretamente no display de 7 segmentos.

PROCEDIMENTOS E RESULTADOS

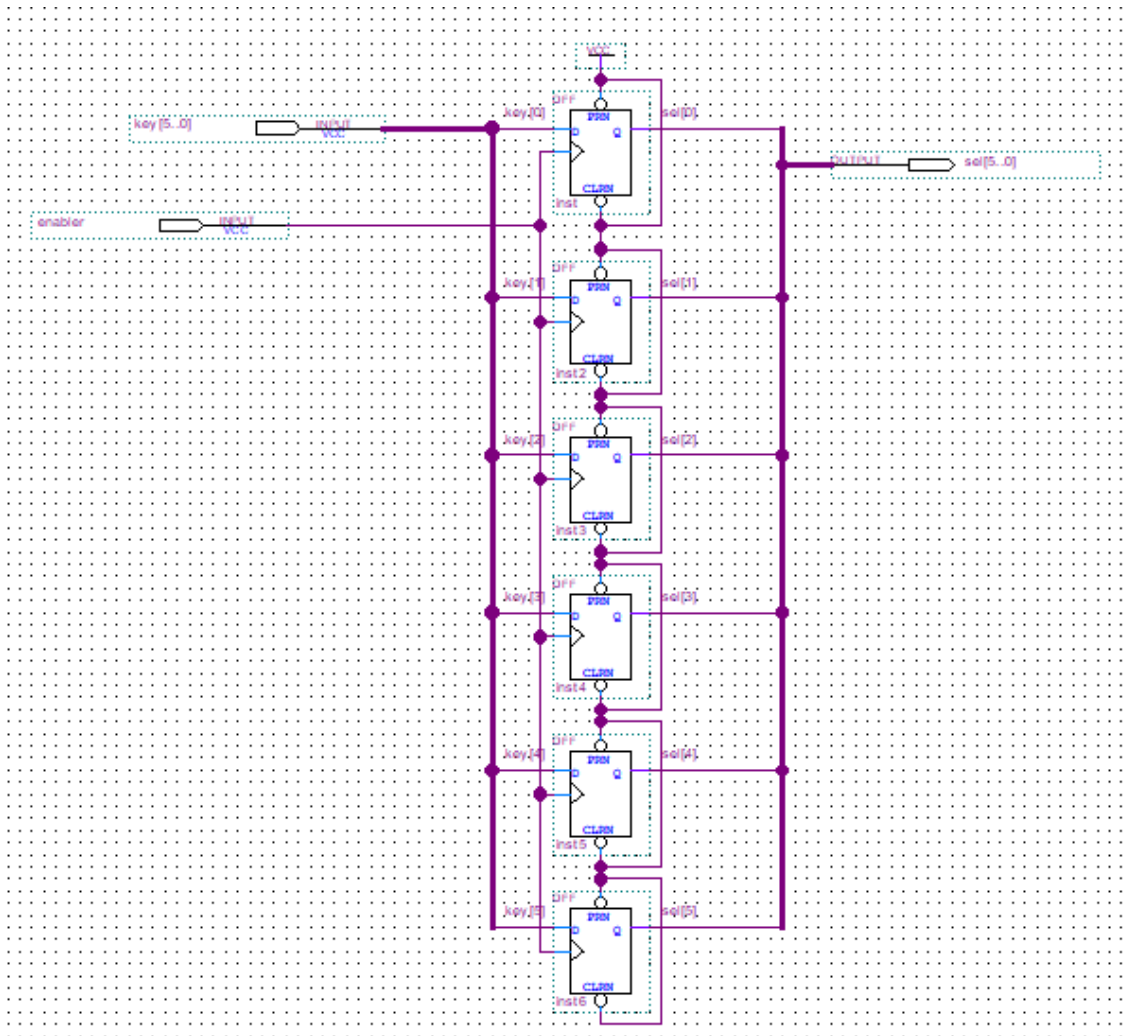
Implementação das dicas em circuito:



KeySel - seletor das chaves

PROCEDIMENTOS E RESULTADOS

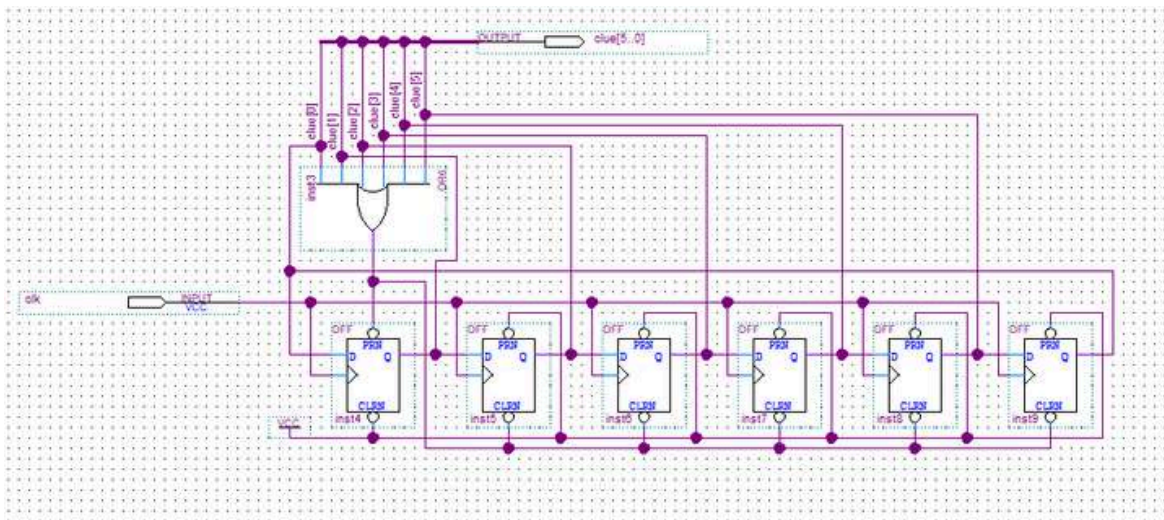
Implementação das dicas em circuito:



clueSel - seletor das dicas

PROCEDIMENTOS E RESULTADOS

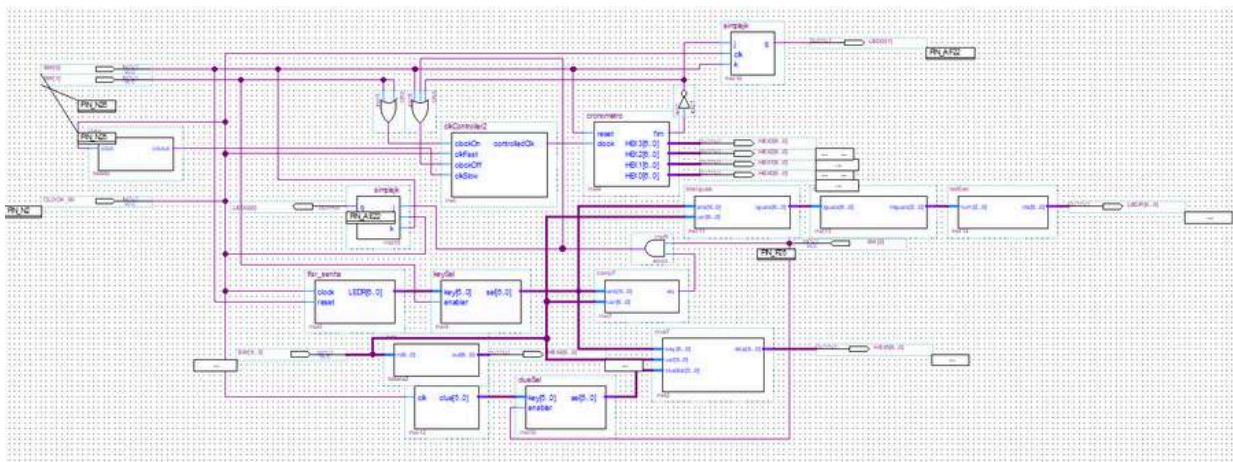
Implementação das dicas em circuito:



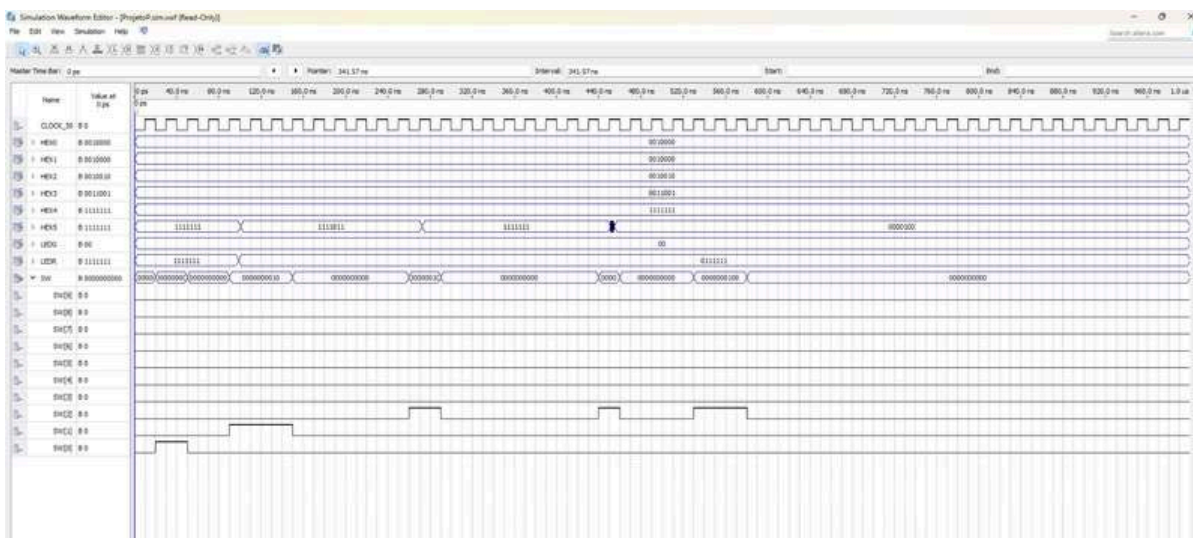
clueGenerator - gera as dicas

PROCEDIMENTOS E RESULTADOS

Circuito principal



O circuito apresentado acima é o “principal”, aquele que é definido como high level para ser passado para a placa e iniciar a jogatina da bomba relógio. Ele engloba todas as caixinhas e funcionalidades explicadas anteriormente. Ao rodarmos as simulações desses circuitos obtivemos, respectivamente, as simulações temporal e funcional:



PROCEDIMENTOS E RESULTADOS



Como citado anteriormente, SW[0] - reset, SW[1] - start, SW[2] - enter e HEX5 - dica. Dessa forma, inicializamos primeiro o sistema com um reset, depois com o start. Além disso, HEX[0-3] é o cronometro e o HEX[4] é o número digitado pelo jogador.

Testamos três vezes acionar o Enter pela subida da borda de clock, de modo que a dica, visível em HEX5 mudasse, testando sua aleatoriedade de escolha. Não é possível observar muita diferença entre as simulações funcional e temporal, entretanto, é possível notar uma inconsistência no HEX5 entre 111111 e 1111011.

PROCEDIMENTOS E RESULTADOS

Vídeos do funcionamento na placa

Para observar a especificação das chaves na placa FPGA clique: [AQUI](#)

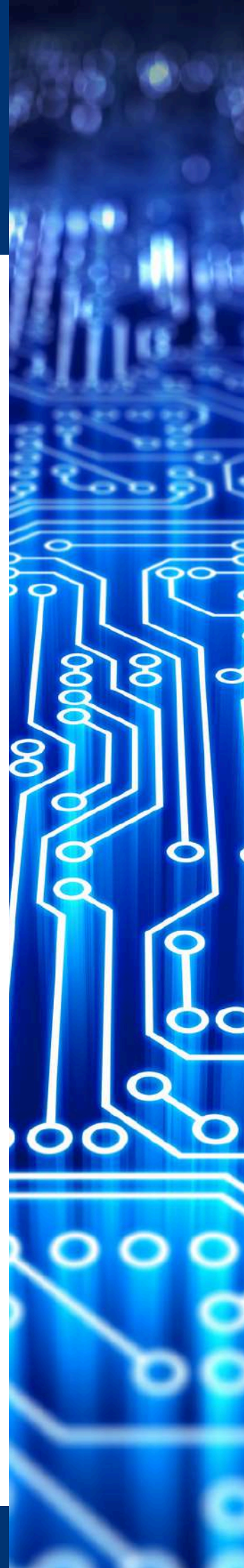
Para observar uma partida sendo vencida clique: [AQUI](#)

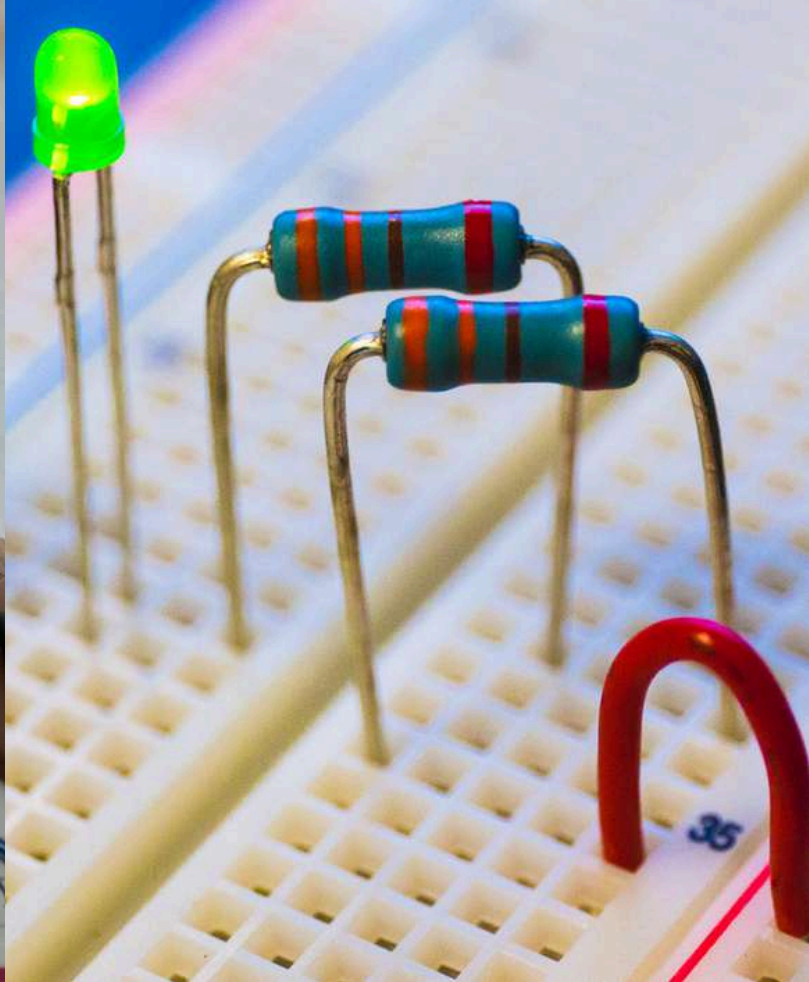
Como o cronômetro de 5 minutos demorava muito para passar, fizemos um timelapse de alguns minutos para demonstrar sua passagem, porém cortamos antes do final. E pode ser observado: [AQUI](#)

Para observar a senha sendo tentada incorretamente clique: [AQUI](#)

Para observar uma partida qualquer clique: [AQUI](#)

Por fim, para observar a bomba sendo explodida ao final do cronômetro, clique: [AQUI](#)





CONCLUSÕES

Com isso, foi possível observar que, neste projeto final utilizamos diversos conteúdos das aulas passadas, como: flip-flops, latches, decodificadores, systemVerilog, contadores, comparadores, clock, entre outros.

Dessa forma, ao projetarmos e implementarmos o jogo de desarmar uma bomba relógio, abarcamos grande parte das aulas de modo que colocamos em prática aquilo que aprendemos de forma lúdica e não convencional. Assim como esperado, a bomba apresentou a ligação de um LED e congelamento do cronômetro ao ser desarmada. Além de apresentar as dicas de forma coerente nos display HEX6, e “explodir” quando o cronômetro chega ao seu fim.

REFERÊNCIAS

- Instruções do Projeto dadas pelo Aprender3 da matéria de Laboratório de Circuitos Lógicos
- Verilogpro.com - systemverilog-always_comb-always_ff
- Asic-world.com - Verilog
- TECHNICAL REPORTS from the ELECTRONICS GROUP at the UNIVERSITY of OTAGO - Table of Linear Feedback Shift Registers by Roy Ward, Timothy C.A. Molteno

Referências Gerais da Matéria:

- Pedroni, V., Eletrônica Digital Moderna e VHDL, Campus, 2010
- Tocci, R. J. e Widmer, N. S, Sistemas digitais: princípios e aplicações, LTC, 2010
- Floyd, T., Sistemas digitais fundamentos e aplicações, Bookman, 2011
- Harris, D. M. e Harrys S. L. Digital design and computer architecture, Morgan Kaufmann, 2017