Escola Politécnica da USP

Departamento de Engenharia de Computação e Sistemas Digitais

PCS3635 – Laboratório Digital 1

Turma 02 – Prof. Reginaldo Arakaki

Planejamento da Experiência 6

Projeto Base do Jogo do Desafio da Memória

Victor Pedreira Santos Pepe

Helena Bianchi Moyen

Gabriel Silva Vilas

Bancada: A-03

Data: 12/02/2024

Objetivos

Nessa atividade, o principal objetivo foi aprimorar o circuito feito na atividade 5 através da inclusão de novos componentes no fluxo de dados e de alterações na unidade de controle para simular o modo "Desafio da Memória" do jogo Genius. Assim, o circuito irá adquirir maior complexidade, possibilitando a melhora e a aplicação do conhecimento adquirido pelo grupo em Sistemas Digitais 1, 2 e no Labóratório Digital.

1.1. Atividade 1 – Projeto Lógico do Jogo Base do Desafio da Memória

Nessa atividade, daremos continuidade ao que foi desenvolvido até então, realizando novos incrementos para consolidar o jogo da memória. Até o desafio da experiência 5, o sistema digital apenas verificava se todas as 16 jogadas (tendo em vista que a ROM utilizada armazena 16 valores) feitas pelo usuário correspondiam aos valores guardados. Caso qualquer uma das jogadas fosse incorreta - ou então se o tempo limite estabelecido, o *timeout*, fosse atingido - o usuário automaticamente perderia o jogo. Caso acertasse as 16, ele venceria. Esse comportamento está bem sintetizado no diagrama de transição de estados, inserido no relatório anterior:

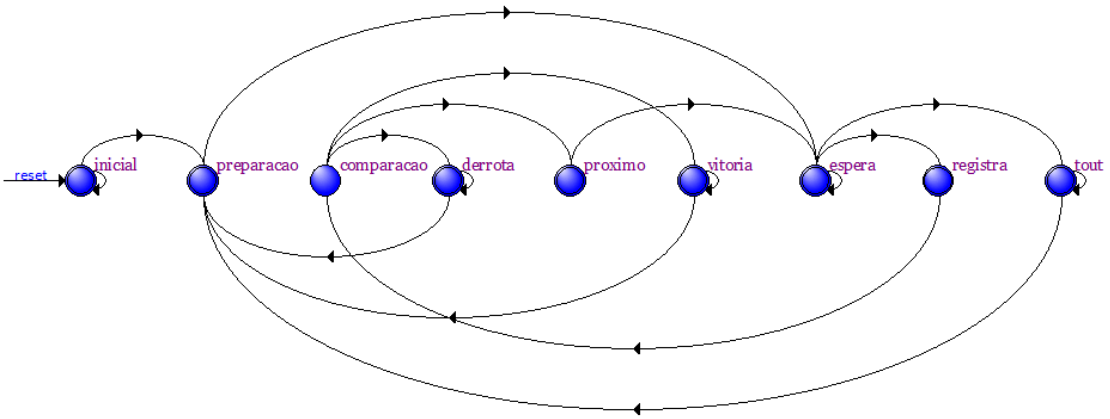


Figura 1; Visão da máquina de estados pela ferramenta STATE MACHINE VIEWER do Intel Quartus Prime do desafio da experiência 5

Note como todas as 16 jogadas são percorridas numa única rodada: a partir do momento que o jogo é iniciado, o jogador deve selecionar todos os 16 valores corretamente em sequência. No circuito atual, entretanto, o número de jogadas cresce gradativamente. Observe o pseudocódigo a seguir, fornecido no arquivo do planejamento:

```
Algoritmo: Jogo Base do Desafio da Memória
    entradas:   jogar, botoes
    saídas:     leds, ganhou, perdeu, pronto
    depuração (sugestão):   contagem, memoria, rodada, estado, jogada_feita,
                            enderecoIgualRodada, jogada_correta, timeout
1.  {
2.      while (verdadeiro) {
3.          espera acionamento do sinal JOGAR

4.          inicia circuito para as condições iniciais
5.          apresenta jogada inicial

6.          enquanto não atingir o final do jogo e não ocorrer uma jogada errada {

7.              reinicia rodada (sequência de jogadas)

8.              enquanto não atingir jogada final da sequência atual e jogada foi correta {

9.                  espera jogada
10.                 compara jogada efetuada com jogada armazenada
11.                 atualiza (incrementa) endereço da próxima jogada

12.             }

13.             vai para próxima rodada (próxima sequência)
14.         }

15.         se atingiu o final do jogo acertando todas as jogadas {
16.             então { // ganhou o jogo
17.                 ativa saída GANHOU
18.                 ativa saída PRONTO
19.                 espera acionamento do sinal JOGAR
20.                 reinicia o jogo
21.             }
22.         }
23.         se jogada errada ou demorou mais de 5 segundos para fazer a jogada {
24.             então { // perdeu o jogo
25.                 ativa saída PERDEU
26.                 ativa saída PRONTO
27.                 espera acionamento do sinal JOGAR
28.                 reinicia o jogo
29.             }
30.         }
31.     }
32. }
```

Figura 2: Pseudocódigo fornecido na apostila da experiência 6

Como podemos observar, na figura 2, agora o jogador precisa acertar a sequência de forma gradativa, ou seja, na primeira rodada o jogador faz apenas uma jogada, na segunda rodada, a mesma jogada da primeira rodada e mais uma jogada e assim sucessivamente até que se atinja a décima setxa rodada, ou que o jogador cometa algum erro. Repare que demorar mais de cinco segundos para realizar uma jogada é tratado como erro.

Agora que compreendemos a diferença entre as experiências anteriores e a atual, podemos prosseguir visualizando a figura 3 que representa as entradas e saídas para esta experiência.

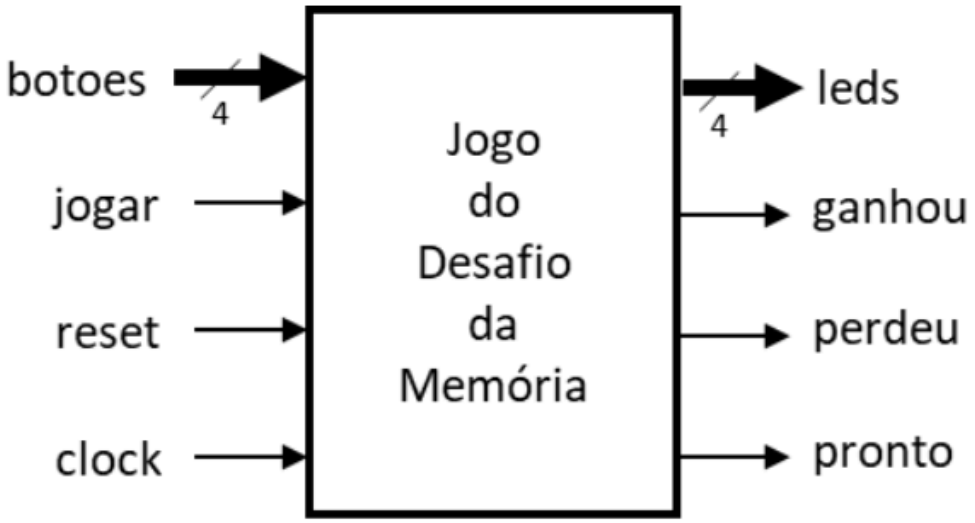


Figura 3: : Diagrama de Blocos da Interface Externa de Sinais do Circuito para a Atividade 1

As entradas dos botões são utilizadas para registrar a entrada das jogadas, que da mesma maneira que na experiência 5, assumem apenas 4 valores (1000, 0100, 0010 e 0001) e tem um dos leds da saída associados a cada uma delas. A entrada jogar inicia o jogo, a entrada reset é utilizada para reiniciar um jogo em andamento e o clock é utilizado na lógica digital interna do circuito.

O jogador pode apenas ganhar ou perder, e quando isso ocorrer a respectiva saída será acionada, juntamente com a saída pronto, que indica sempre que o jogo chegou a um estado final. Como descrito na figura 2, esses sinais de saída do estado final ficam ativos até que a entrada jogar seja acionada novamente.

Por fim, vale explicitar que a saída dos leds apresenta a jogada inicial e depois mostra apenas as jogadas feitas pelo jogador. Ou seja, é utilizado para verificar a entrada acionada pelos botões.

Abaixo, na figura 4, podemos observar um diagrama de alto nível do circuito desta experiência:

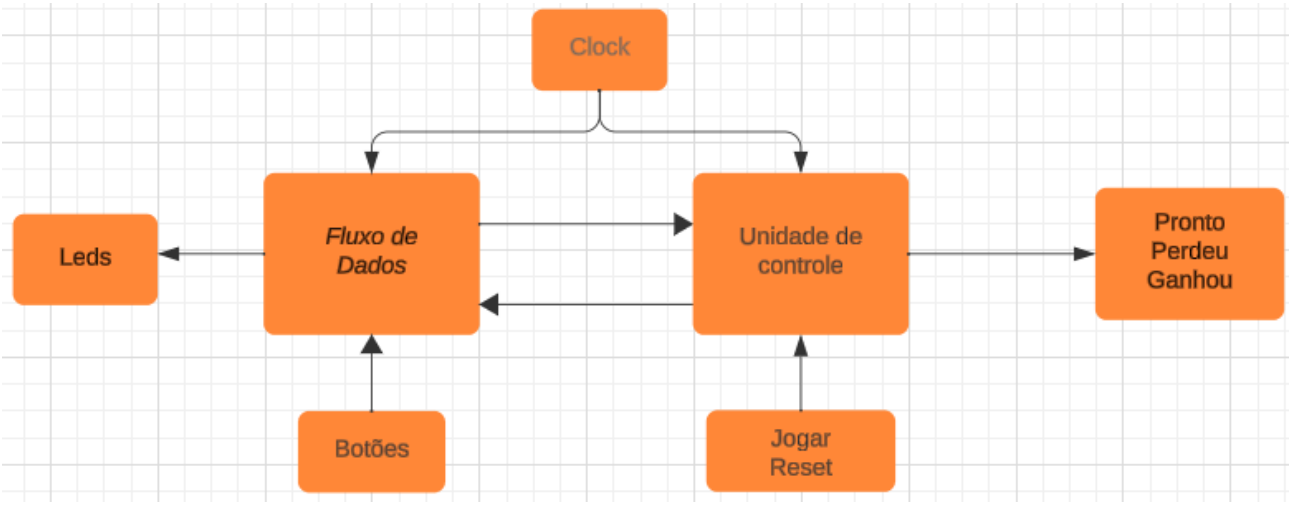


Figura 4: Diagrama de alto nível do projeto

Com base no exposto, podemos identificar os elementos do fluxo de dados do sistema. Primeiramente, como é possível observar pela figura 5, o fluxo de dados tem como entrada os sinais de controle, os botões e o clock, e como saída os sinais de condição destinados à unidade de controle e os leds.

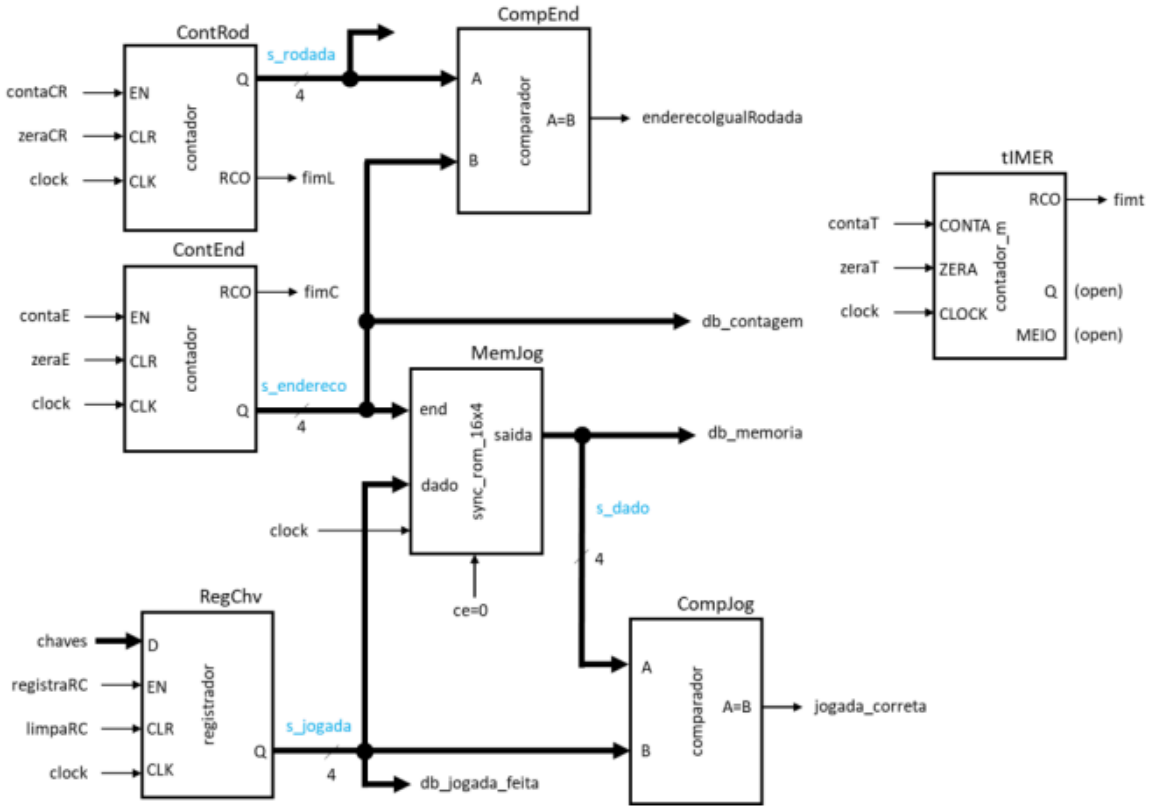


Figura 5: Diagrama do fluxo de dados da atividade 1

Pela figura x, podemos observar que os sinais de controle provenientes da UC são: contaE, zeraE, contaCR, zeraCR, limpaRC, registraRC, contaT e zeraT. Enquanto isso, os

sinais de condição enviados à UC são: jogada_correta, fimT, enderecoIgualRodada, fimC, fimL e db_jogada_feita. A existência desses sinais poderá ser verificada novamente mais a frente neste relatório, na seção "Descrição do Circuito - Módulos em Verilog", com algumas pequenas alterações de nomenclatura, porém com as funções lógicas mantidas.

Este fluxo de dados difere dos utilizados nas experiências anteriores, justamente pelo fato de termos, agora, jogadas dentro de rodadas. Para contemplar esta nova funcionalidade, foi necessário adicionar um contador de 4 bits ContRod e um novo comparador de 4 bits CompEnd. Como o nome sugere, o ContRod conta as rodadas do jogo que variam de 0 a 15 (ou seja, são 16 rodadas). Já o CompEnd desempenha o papel de limitar o número de jogadas por rodada, uma vez que o número máximo de jogadas é igual ao número da rodada presente. Assim, o comparador dispara o sinal, para a UC, de que as jogadas atingiram o seu valor máximo na rodada atual.

Além disso, temos um contador TIMER, acrescentado no circuito pela primeira vez no desafio da experiência 5 e tem a função de inserir um timeout para as jogadas. Sua tarefa é contar o sinal de clock até um valor pré-definido. Por exemplo, se tivermos um clock de 1 kHz e quisermos que a saída deste contador seja acionada após 3 segundos, devemos pré definir que a saída fimT deverá ser ativada quando o contador chegar ao valor 3000. Repare que este é um contador significativamente maior que os outros contadores utilizados neste circuito.

Por fim, todos os outros elementos já foram explorados nas experiências anteriores, mas, de forma resumida, temos uma ROM 16x4 que armazena 16 valores de 4 bits, um registrador que guarda a entrada dos botões, um comparador que compara o valor contido no endereço de memória da ROM escolhido naquele momento com o valor armazenado no registrador. O endereço de memória da ROM é escolhido pelo contador de 4 bits ContEnd. Assim, esta parte do circuito tem a capacidade de comparar todos os valores contidos na ROM com valores inseridos pelo usuário.

Um último comentário sobre o fluxo de dados diz respeito ao módulo edge_detector, que não está representado na figura 5, mas se faz presente no circuito, como poderá ser observado posteriormente neste documento. Sua função é detectar a borda de subida da ativação de uma das chaves de entrada (ou botões de entrada) e enviar um sinal de condição db_jogada_feita para a UC. Este módulo também foi utilizado na experiência 5.

Tendo ainda em vista a figura 4, podemos analisar de forma mais profunda a Unidade de Controle. Segue o diagrama de estados da UC para a implementação desta experiência:

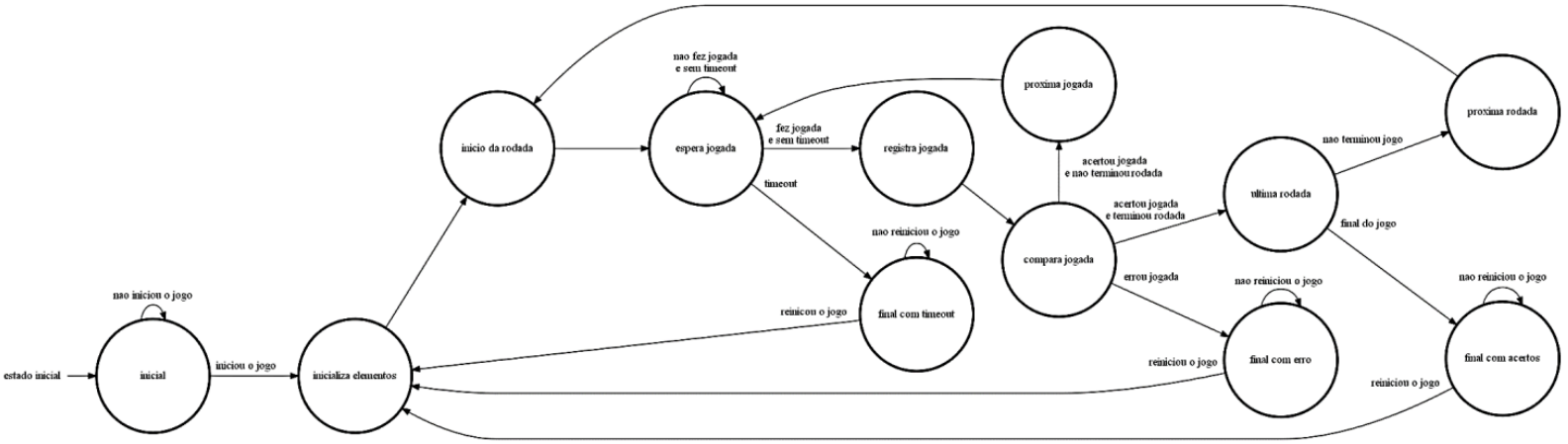


Figura 6: Diagrama de transição de estados da atividade 1

Como é possível observar, a complexidade aumentou consideravelmente comparando-se com a experiência anterior. Para compreender melhor a transição desses estados e os sinais que promovem estas transições, iremos analisar a figura 7 que contém o código verilog responsável pela lógica de próximo estado.

```
// Logica de proximo estado
always @* begin
    case (Eatual)
        inicial:     Eprox = iniciar ? preparacao : inicial;
        preparacao:  Eprox = inicio_rodada;
        inicio_rodada: Eprox =  espera;
        espera:      Eprox = timeout ? tout :
                                    jogada ? registra: espera;
        registra:    Eprox = comparacao;
        comparacao:  Eprox = (~jogada_correta) ? derrota :
                             (enderecoIgualRodada) ? ultima_rodada:
                                        proxima_jogada;
        proxima_jogada: Eprox = espera;
        ultima_rodada:     Eprox = fimCR ? vitoria : proxima_rodada;
        proxima_rodada:     Eprox = inicio_rodada;
        derrota:     Eprox = (iniciar) ? preparacao : derrota;
        vitoria:     Eprox = (iniciar) ? preparacao : vitoria;
        tout:            Eprox = (iniciar) ? preparacao : tout;
        default:     Eprox = inicial;
    endcase
end
```

Figura 7: Código verilog responsável pela lógica de próximo estado para esta experiência

Primeiramente, observemos que existem três estados finais, um para o caso de erro do jogador, um para o caso de vitória do jogador e outro para o caso de

timeout (também considerado derrota), que são, respectivamente, derrota, vitória e tout, e possuem como próximo estado o estado preparação, que indicia um reinício de jogo. Isso segue a mesma lógica utilizada no desafio da experiência 5.

Repare que as diferenças entre esta experiência e a anterior estão mais pronunciadas a partir do estado comparação e estão melhor discutidas adiante na seção Descrição do Circuito- Módulos em Verilog.

Assim, pudemos concluir a projeto ao conectar o fluxo de dados com a UC, e descrever nossas alterações no código verilog utilizado na experiência anterior. Juntamos tudo no módulo verilog circuito_jogo_base, cujas entradas e saídas correspondem às descritas na figura 8, com a adição de sinais de depuração.

```
module circuito_jogo_base (
input clock,
input reset,
input iniciar,
input [3:0] chaves,
output ganhou,
output perdeu,
output pronto,
output [3:0] leds,
output [6:0] db_contagem,
output [6:0] db_memoria,
output [6:0] db_estado,
output [6:0] db_jogadafeita,
output [6:0] db_rodada,
output db_clock,
output db_tem_jogada,
output db_timeout,
output db_jogada_correta,
output db_enderecoIgualRodada,
output [11:0] db_Q
);
```

Figura 8: Entradas e saídas no código verilog do circuito da atividade 1

Como pode ser visualizado na figura acima, adicionamos 11 sinais de depuração, ou seja, todos os sinais de output descritos no código após o sinal de saída dos leds. O input chaves corresponde aos botões utilizados pelo jogador para realizar as jogadas.

Descrição do Circuito - Módulos em Verilog

Para montar o circuito, primeiramente precisamos listar os componentes que serão utilizados. No fluxo de dados, precisaremos de dois contadores 163: um para prover o endereço que será lido na ROM (como antes) e outro para contar em qual rodada estamos. Será necessário também um contador de M bits, como visto anteriormente, para verificar o timeout. Além disso, utilizaremos dois comparadores 85: um para comparar o valor da memória com a jogada realizada e outro para verificar se o fim da rodada foi atingido. Outro componente adicionado nesse circuito é uma ROM 16x4 extra para sempre mostrar qual é o próximo valor a ser adicionado na sequência, além da outra ROM 16x4 que é usada para comparar seus dados com as chaves. Por fim, no fluxo de dados, ainda utilizamos um registrador de 4 bits para armazenar os valores das chaves.

Na unidade de controle, foram feitas algumas alterações. Primeiramente, precisaremos de novos estados, como teremos várias jogadas dentro de cada rodada. Ou seja, precisaremos de estados que diferenciam o fim de uma jogada e o fim de uma rodada. Afinal, estar na última rodada é um estado que leva a vitória ou a derrota, diferentemente de outras rodadas. Assim, é necessário ter um sinal que indique se atingimos a última rodada. Esse sinal vem de um dos contadores (*contador_rodada*) e é identificado como *fimCR*. Ele vai determinar se, estando numa última jogada correta de uma rodada N (**estado decodificado aqui como ultima_rodada**), iremos para um estado de vitória. Se *fimCR* for alto, significa que estamos na última rodada e, dado que acertamos a comparação, ganhamos o jogo. Caso contrário, apenas seguimos para a próxima rodada do jogo.

```
// Logica de proximo estado
always @* begin
    case (Eatual)
        inicial:     Eprox = iniciar ? preparacao : inicial;
        preparacao:  Eprox = inicio_rodada;
        inicio_rodada: Eprox =  espera;
        espera:      Eprox = timeout ? tout :
                                    jogada ? registra: espera;
        registra:    Eprox = comparacao;
        comparacao:  Eprox = (~jogada_correta) ? derrota :
                             (enderecoIgualRodada) ? ultima_rodada:
                                        proxima_jogada;
        proxima_jogada: Eprox = espera;
        ultima_rodada:     Eprox = fimCR ? vitoria : proxima_rodada;
        proxima_rodada:     Eprox = inicio_rodada;
        derrota:     Eprox = (iniciar) ? preparacao : derrota;
        vitoria:     Eprox = (iniciar) ? preparacao : vitoria;
        tout:           Eprox = (iniciar) ? preparacao : tout;
        default:     Eprox = inicial;
    endcase
end
```

Figura 9: : Lógica de transição de estados da Unidade de Controle

Segundamente, veja que no estado *comparação* temos uma condição nova: se o número contado dos dois contadores for o mesmo, segue-se para o estado *última_rodada*. Ou seja, se o número do endereço da memória for o mesmo que o índice da rodada atual, estamos na última jogada. Isso faz sentido, pois, para a n-ésima rodada, termina-se a sequência no n-ésimo dado da memória.

Veja que nos estados essa diferença entre "rodada" e "jogada" é bem delimitada com os estados *próxima_jogada* e *próxima_rodada*. Em cada um deles, há sinais diferentes a serem mandados, os quais analisaremos a seguir.

```
// Logica de saida (maquina Moore)
always @* begin
    zeraCE     = (Eatual == inicial || Eatual == preparacao || Eatual == inicio_rodada ) ? 1'b1 : 1'b0;
    contaCE    = (Eatual == proxima_jogada) ? 1'b1 : 1'b0;
    zeraCR     = (Eatual == inicial || Eatual == preparacao) ? 1'b1 : 1'b0;
    contaCR    = (Eatual == proxima_rodada) ? 1'b1 : 1'b0;
    zeraR      = (Eatual == inicial || Eatual == preparacao) ? 1'b1 : 1'b0;
    registraR  = (Eatual == registra) ? 1'b1 : 1'b0;
    contaT     = (Eatual == espera) ? 1'b1 : 1'b0;
    zeraT      = (Eatual == inicial || Eatual == preparacao || Eatual == inicio_rodada || Eatual == proxima_jogada) ? 1'b1 : 1'b0;
    pronto     = (Eatual == derrota || Eatual == vitoria || Eatual == tout) ? 1'b1 : 1'b0;
    errou      = (Eatual == derrota || Eatual == tout) ? 1'b1: 1'b0;
    acertou    = (Eatual == vitoria) ? 1'b1 : 1'b0;
```

Figura 10: : Lógica de transição de estados da Unidade de Controle

Note que o contador de endereço é zerado todo início de rodada (*zeraCE*), o que é congruente, afinal, recomeçamos a sequência toda rodada. Ele é habilitado para contar apenas no estado *próxima_jogada,* o que também faz sentido, afinal, a cada jogada, lemos um dado diferente da memória. Já o contador de rodada é zerado apenas quando o jogo é recomeçado (*zeraCR*) e habilitado no estado *próxima_rodada,* contabilizando cada nova rodada. De resto, os sinais se mantém os mesmos do desafio da experiência 5.

Todos os códigos verilog desta experiência seguem em anexo ao documento, e algumas fotos extras que podem auxiliar em uma visualização rápida do sistema e ajudar no laboratório (parâmetro dos estados) estão dispostas abaixo:

```
module unidade_controle (
    input       clock,
    input       reset,
    input       iniciar,
    input       fimCE,
    input       fimCR,
    input       jogada,
    input       enderecoIgualRodada,
    input       jogada_correta,
    input       timeout,
    output reg zeraCE,
    output reg contaCE,
    output reg zeraCR,
    output reg contaCR,
    output reg zeraR,
    output reg registraR,
    output reg zeraT,
    output reg contaT,
    output reg pronto,
    output reg errou,
    output reg acertou,
    output reg [3:0] db_estado
);
```

Figura 11: Entradas e saídas da UC do circuito

```
module contador_m #(parameter M=3000, N=12)
  (
   input  wire          clock,
   input  wire          zera_as,
   input  wire          zera_s,
   input  wire          conta,
   output reg  [N-1:0]  Q,
   output reg           fim,
   output reg           meio
  );
```

Figura 12: Entradas e saídas do código verilog do contador_m

```
// Saida de depuracao (estado)
case (Eatual)
    inicial:     db_estado = 4'b0000;  // 0
    preparacao:  db_estado = 4'b0011;  // 3
    inicio_rodada: db_estado = 4'b0010;  // 2
    espera:      db_estado = 4'b0001;  // 1
    registra:    db_estado = 4'b0100;  // 4
    comparacao:  db_estado = 4'b0101;  // 5
    proxima_jogada:     db_estado = 4'b0110;  // 6
    ultima_rodada: db_estado = 4'b0111;  // 7
    proxima_rodada: db_estado = 4'b1000;  // 8
    derrota:     db_estado = 4'b1110;  // E
    vitoria:     db_estado = 4'b1101;  // D
    tout:            db_estado = 4'b1011;  // B

    default:     db_estado = 4'b1111;  // F
endcase
```

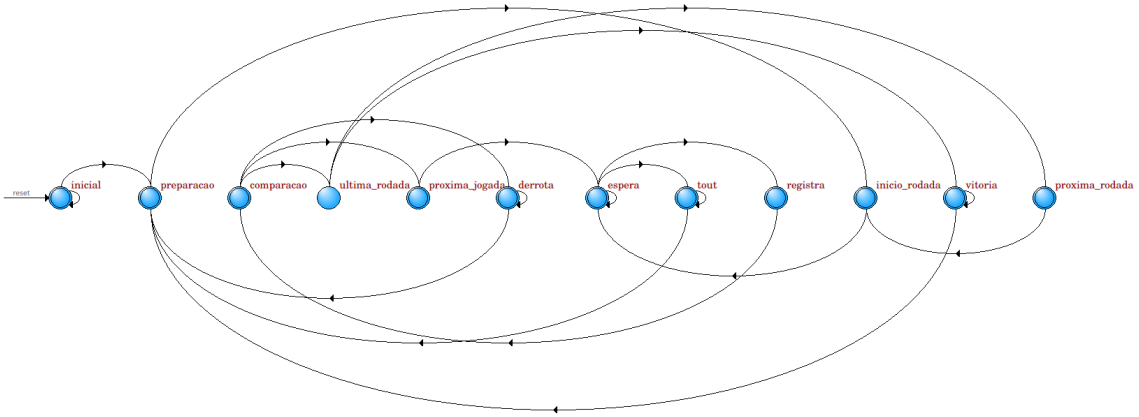
```
// Define estados
parameter inicial    = 4'b0000;  // 0
parameter preparacao = 4'b0011;  // 3
parameter inicio_rodada = 4'b0010; // 2
parameter espera     = 4'b0001; // 1
parameter registra   = 4'b0100;  // 4
parameter comparacao = 4'b0101;  // 5
parameter proxima_jogada    = 4'b0110;  // 6
parameter ultima_rodada = 4'b0111; // 7
parameter proxima_rodada = 4'b1000; // 8
parameter derrota    = 4'b1110; // E
parameter vitoria    = 4'b1101; // D
parameter tout       = 4'b1011; //B
```

Figuras 13 e 14: Numeração dos estados e saídas definição do valor de suas saídas de depuração.

Análise do Circuito - Ferramentas do Quartus

Com os módulos em Verilog prontos, pudemos compilar o circuito no Intel Quartus e analisar seus componentes com as ferramentas de Netlist Viewers. Primeiramente, usamos a State Machine Viewer para observar a máquina de Moore resultante do circuito. Note a diferença em relação à Figura 1 do circuito do Desafio

da experiência anterior (5). Aqui temos 3 novos estados: *última rodada, próxima_rodada e início_rodada.*



	Source State	Destination State	Condition
1	comparacao	derrota	(!jogada_correta)
2	comparacao	ultima_rodada	(enderecoIgualRodada).(jogada_correta)
3	comparacao	proxima_jogada	(!enderecoIgualRodada).(jogada_correta)
4	derrota	derrota	(!iniciar)
5	derrota	preparacao	(iniciar)
6	espera	espera	(!jogada).(!timeout)
7	espera	tout	(timeout)
8	espera	registra	(jogada).(!timeout)
9	inicial	inicial	(!iniciar)
10	inicial	preparacao	(iniciar)
11	inicio_roda...	espera	
12	preparacao	inicio_rodada	
13	proxima_jo...	espera	
14	proxima_ro...	inicio_rodada	
15	registra	comparacao	
16	tout	tout	(!iniciar)
17	tout	preparacao	(iniciar)
18	ultima_rod...	vitoria	(fimCR)
19	ultima_rod...	proxima_rodada	(!fimCR)
20	vitoria	vitoria	(!iniciar)
21	vitoria	preparacao	(iniciar)

Figura 15: State Machine Viewer da nova unidade de controle do circuito.

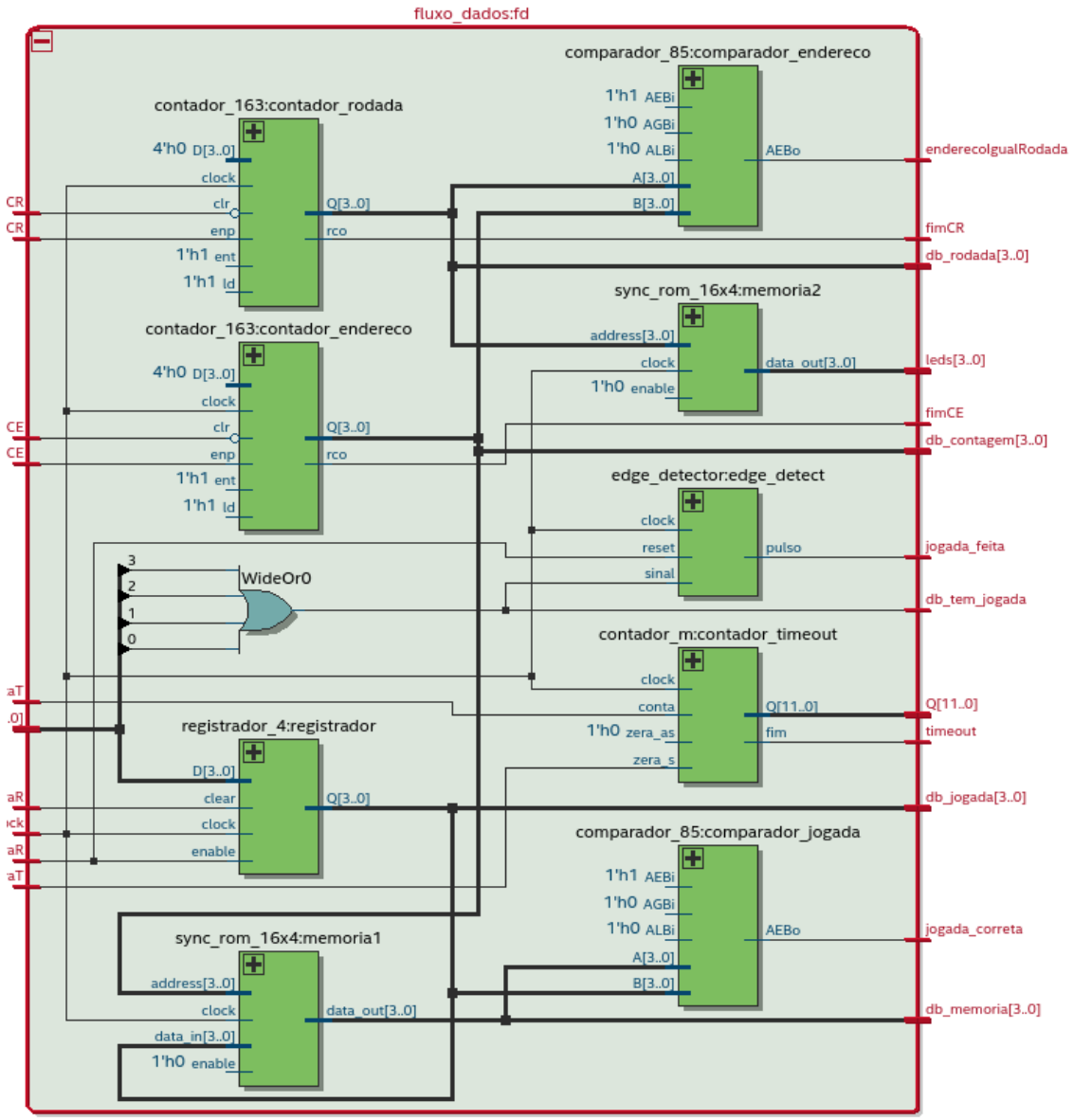


Figura 16: : RTL Viewer do fluxo de dados do circuito

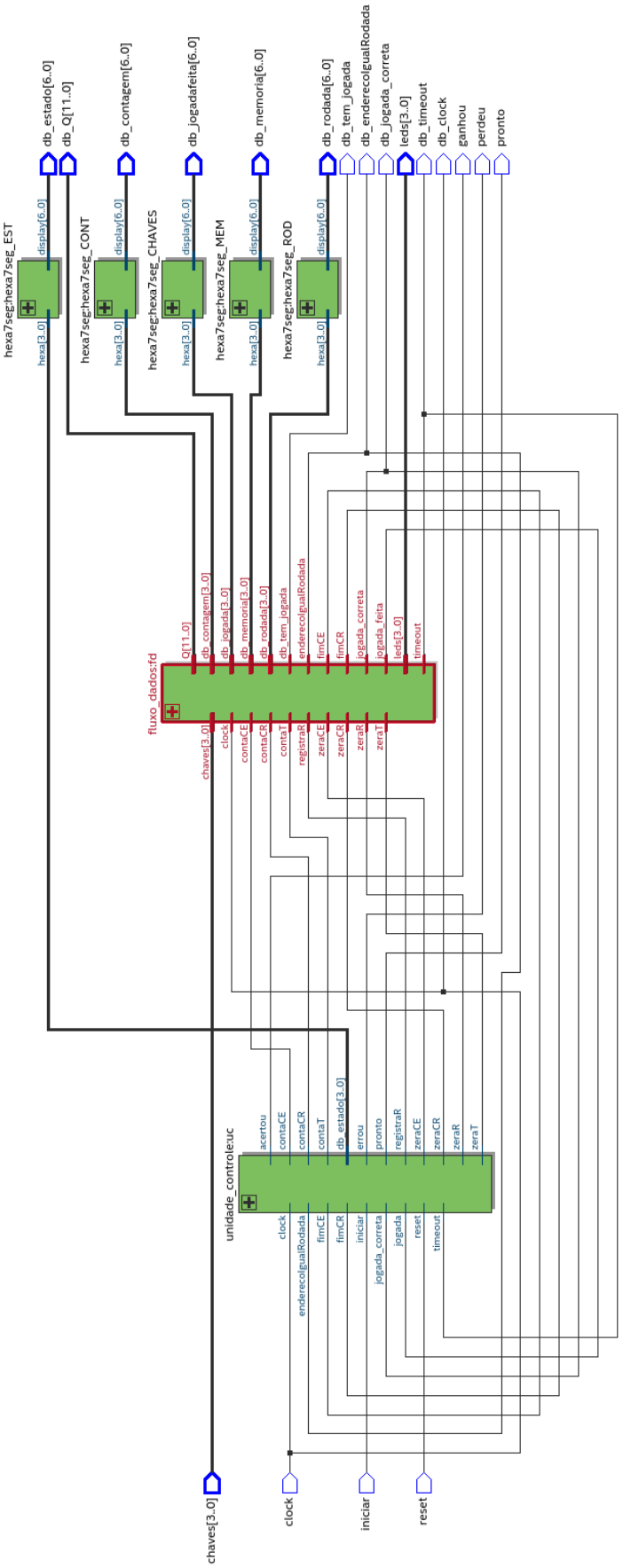


Figura 17: : RTL Viewer do circuito completo

Simulações no ModelSim dos Planos de Teste

Para testar as novas funcionalidades do circuito, implementamos três planos de testes descritos a seguir. No primeiro deles, o jogador consegue ser vitorioso e ganha o jogo, no segundo ele comete algum erro e no terceiro ele demora muito para realizar uma jogada e perde, novamente, mas pelo timeout e não por errar a sequência em si.

Todos os resultados esperados foram obtidos nas simulações no ModelSim, como pode ser visto parcialmente nas imagens abaixo.

#	Operação	Sinais de controle	Resultado esperado
c.i	Condições iniciais	reset=0 iniciar=0 chaves=0000	pronto=x db_contagem=xxxx db_memoria=xxxx=leds db_jogada =xxxx db_rodada = xxxx db_estado=xxxx
1	Resetar o circuito	reset=1 iniciar=0 chaves=0000	pronto=0 db_contagem=0000 db_memoria=0001=leds db_jogada =0000 db_rodada = 0000 db_estado=0000 (inicial) endereçoIgualRodada = 1 db_jogada_correta=0
2	Iniciar jogo por 5 períodos de clock e esperar	reset=0 iniciar=1 chaves=0000	pronto=0 db_contagem=0000 db_memoria=0001=leds db_jogada =0000 db_rodada = 0000 endereçoIgualRodada = 1 db_jogada_correta=0 db_estado=3 - 2- 1(preparação, início_rodada e espera)
3	Ajustar chaves para 0001 por 10 períodos de clock (Primeira rodada - 1 jogada)	reset=0 iniciar=0 chaves=0001	pronto=0 db_contagem=0000 db_memoria=0001 leds = 0001 db_jogada =0001 db_rodada = 0000 endereçoIgualRodada = 1 db_jogada_correta=1

			db_estado=4-5-7-8- 2-1 (registra, comparação, última_rodada,, próxima_rodada, início_rodada e espera)
4 e 5	Ajustar chaves para 0001 e 0010 por 10 períodos de clock cada (Segunda rodada - 2 jogadas)	reset=0 iniciar=0 chaves=0001-0010	pronto=0 db_contagem=0000-0001 db_memoria=0001-0010 leds = 0010 db_jogada =0001-0010 db_rodada = 0001 endereçolgualRodada = 0 - 1 db_jogada_correta=1 db_estado=4-5-6-1-4-5-7-8- 2-1
6, 7 e 8	Ajustar chaves para 0001, 0010 e 0100 por 10 períodos de clock cada (Terceira rodada - 3 jogadas)	reset=0 iniciar=0 chaves=0001-0010-0100	pronto=0 db_contagem=0000-0001-0010 db_memoria=0001-0010-0100 leds = 0100 db_jogada =0001-0010-0100 db_rodada = 0010 endereçolgualRodada = 0 –0- 1 db_jogada_correta=1 db_estado=4-5-6-1-4-5-6-1-4-5-7-8- 2-1
9,10,11 e 12	Ajustar chaves para 0001, 0010, 0100 e 1000 por 10 períodos de clock cada (Quarta rodada - 4 jogadas)	reset=0 iniciar=0 chaves=0001-0010-0100-1000	pronto=0 db_contagem=0000-0001-0010-0011 db_memoria=0001-0010-0100-1000 leds = 1000 db_jogada =0001-0010-0100-1000 db_rodada = 0011 endereçolgualRodada = 0-0-0- 1 db_jogada_correta=1 db_estado=4-5-6-1 (3x) e 4-5-7-8- 2-1
13,14,15, 16 e 17	Ajustar chaves para 0001, 0010, 0100, 1000 e 0100 por 10 períodos de clock cada (Quinta rodada - 5 jogadas)	reset=0 iniciar=0 chaves=0001-0010-0100-1000-0100	pronto=0 db_contagem=0000-0001-0010-0011-0100 db_memoria=0001-0010-0100-1000-0100 leds = 0100 db_jogada =0001-0010-0100-1000-0100 db_rodada = 0100 endereçolgualRodada = 0-0-0-0- 1 db_jogada_correta=1 db_estado=4-5-6-1 (4x) e 4-5-7-8- 2-1
18 a 23	Ajustar chaves para 0001, 0010, 0100, 1000, 0100 e 0010 por 10 períodos de clock cada	reset=0 iniciar=0 chaves=0001-0010-0100-1000-0100-0010	pronto=0 db_contagem=0000-0001-0010-0011-0100-0101 db_memoria=0001-0010-0100-1000-0100-0010 leds = 0010

	(Sexta rodada - 6 jogadas)		db_jogada =0001-0010-0100-1000-0100-0010 db_rodada = 0101 endereçoIgualRodada = 0-0-0–0-0- 1 db_jogada_correta=1 db_estado=4-5-6-1 (5x) e 4-5-7-8- 2-1
24 a 122	Continua acertando a sequência até o décimo quarto dado (0001,0010,0100,1000,0100,0010,0001,0001, 0010, 0010, 0100, 0100, 1000, 1000, 0001) - Sétima a décima quinta rodadas	reset=0 iniciar=0 chaves=0001-0010-0100-1000-0100-0010…0001	pronto=0 db_contagem=0000-0001-0010-0011-0100-0101-…-1110 db_memoria=0001-0010-0100-1000-0100-0010-...-0001 leds = 0001-...-0001 db_jogada =0001-0010-0100-1000-0100-0010-... -0001 db_rodada = 0110-...-1110 endereçoIgualRodada = 0-0-0–0-0…-1-0-0…-1… db_jogada_correta=1 db_estado=4-5-6-1 (n vezes) e 4-5-7-8- 2-1 (9 vezes)
123 a 138	Acertar a sequência completa e ganhar	reset = 0 iniciar = 0 chaves = 0001,0010,0100,1000,0100,0010, 0001,0001, 0010, 0010, 0100, 0100, 1000, 1000, 0001, 0100	pronto=0 db_contagem=0000-0001-0010-0011-0100-0101-…-1110-1111 db_memoria=0001-0010-0100-1000-0100-0010-...-0001-0100 leds = 0100 db_jogada =0001-0010-0100-1000-0100-0010-... -0001-0100 db_rodada = 1111 endereçoIgualRodada = 0-0-0–0-0-0…-1 db_jogada_correta=1 db_estado=4-5-6-1 (15 vezes) e 4-5-7-D (vitória) pronto = 1 ganhou = 1

Tabela 1: Plano de testes vitória

#	Operação	Sinais de controle	Resultado esperado
c.i	Condições iniciais	reset=0 iniciar=0 chaves=0000	pronto=x db_contagem=xxxx db_memoria=xxxx=leds db_jogada =xxxx db_rodada = xxxx db_estado=xxxx

1	Resetar o circuito	reset=1 iniciar=0 chaves=0000	pronto=0 db_contagem=0000 db_memoria=0001=leds db_jogada =0000 db_rodada = 0000 db_estado=0000 (inicial) endereçolgualRodada = 1 db_jogada_correta=0
2	Iniciar jogo por 5 períodos de clock e esperar	reset=0 iniciar=1 chaves=0000	pronto=0 db_contagem=0000 db_memoria=0001=leds db_jogada =0000 db_rodada = 0000 endereçolgualRodada = 1 db_jogada_correta=0 db_estado=3 - 2- 1(preparação, início_rodada e espera)
3	Ajustar chaves para 0001 por 10 períodos de clock (Primeira rodada - 1 jogada)	reset=0 iniciar=0 chaves=0001	pronto=0 db_contagem=0000 db_memoria=0001 leds = 0001 db_jogada =0001 db_rodada = 0000 endereçolgualRodada = 1 db_jogada_correta=1 db_estado=4-5-7-8- 2-1 (registra, comparação, última_rodada,, próxima_rodada, início_rodada e espera)
4 e 5	Ajustar chaves para 0001 e 0010 por 10 períodos de clock cada (Segunda rodada - 2 jogadas)	reset=0 iniciar=0 chaves=0001-0010	pronto=0 db_contagem=0000-0001 db_memoria=0001-0010 leds = 0010 db_jogada =0001-0010 db_rodada = 0001 endereçolgualRodada = 0 - 1 db_jogada_correta=1 db_estado=4-5-6-1-4-5-7-8- 2-1
6, 7 e 8	Ajustar chaves para 0001, 0010 e 0100 por 10 períodos de clock cada (Terceira rodada - 3 jogadas)	reset=0 iniciar=0 chaves=0001-0010-0100	pronto=0 db_contagem=0000-0001-0010 db_memoria=0001-0010-0100 leds = 0100 db_jogada =0001-0010-0100 db_rodada = 0010 endereçolgualRodada = 0 –0- 1 db_jogada_correta=1 db_estado=4-5-6-1-4-5-6-1-4-5-7-8-2-1
9,10,11 e	Ajustar chaves para	reset=0 iniciar=0	pronto=0 db_contagem=0000-0001-0010-0011

12	0001, 0010, 0100 e 1000 por 10 períodos de clock cada (Quarta rodada - 4 jogadas)	chaves=0001-0010-0100-1000	db_memoria=0001-0010-0100-1000 leds = 1000 db_jogada =0001-0010-0100-1000 db_rodada = 0011 endereçolgualRodada = 0-0-0- 1 db_jogada_correta=1 db_estado=4-5-6-1 (3x) e 4-5-7-8- 2-1
13,14,15, 16 e 17	Ajustar chaves para 0001, 0010, 0100, 0001 e 0100 por 10 períodos de clock cada (Quinta rodada - 5 jogadas, erro na quarta jogada)	reset=0 iniciar=0 chaves=0001-0010-0100-0001-0100	pronto=1 perdeu = 1 db_contagem=0000-0001-0010-0011 db_memoria=0001-0010-0100-1000- leds = 0100 db_jogada =0001-0010-0100-0001-0100 db_rodada = 0100 endereçolgualRodada = 0-0-0-0 db_jogada_correta=1-0 db_estado=4-5-6-1 (3x) e 4-5-E (derrota)

Tabela 2: Plano de testes derrota

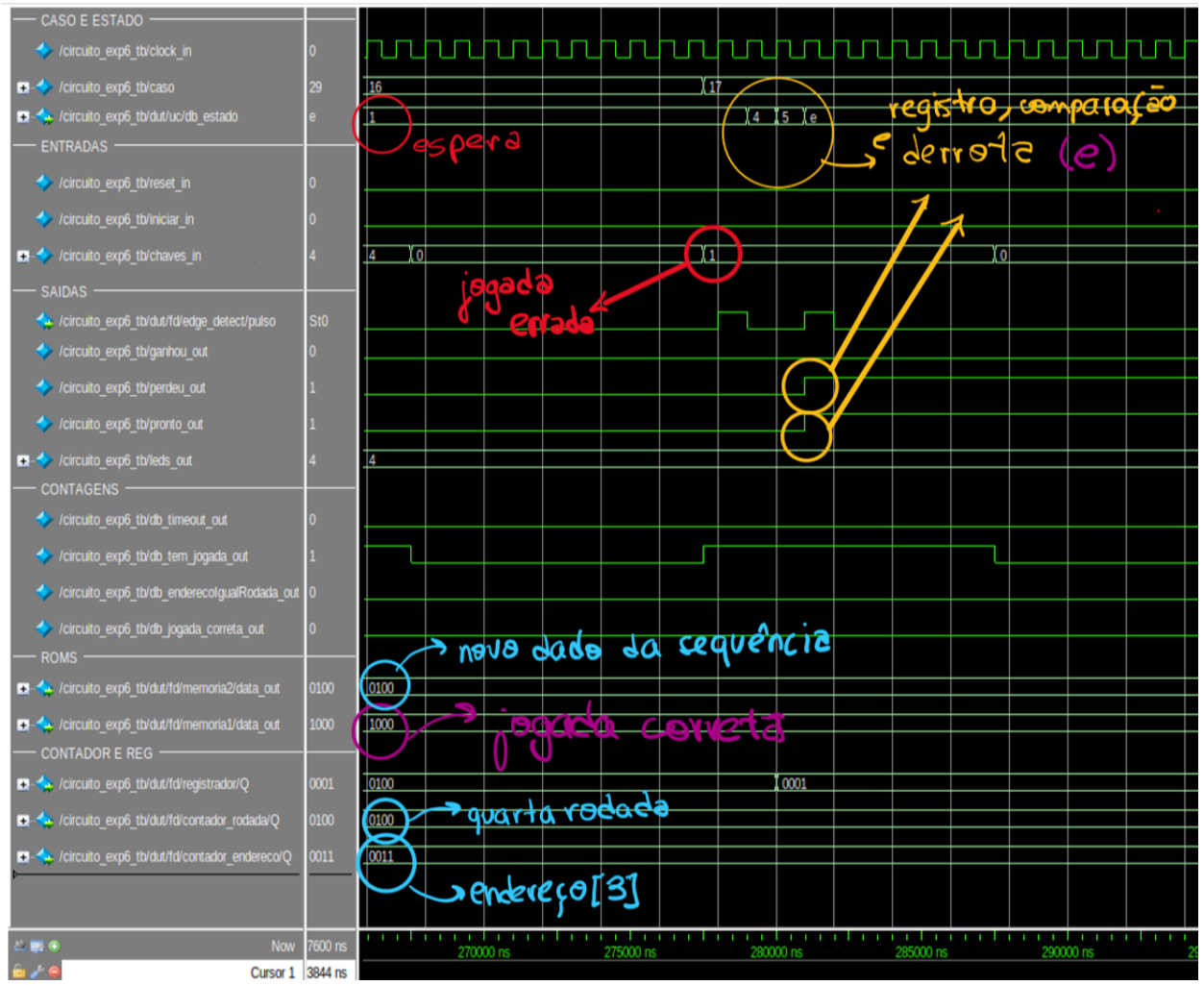


Figura 18: Simulação no Modelsim do cenário de derrota por erro das chaves

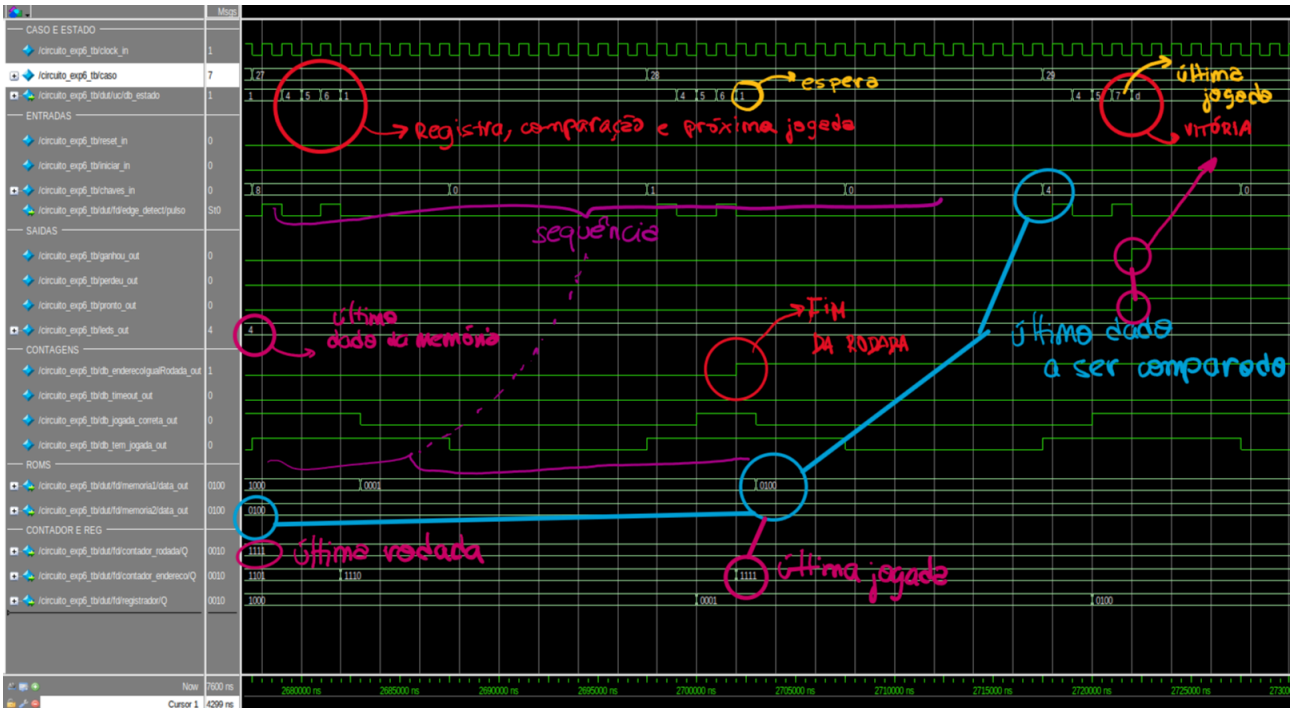


Figura 19: Simulação no Modelsim do cenário de vitória

#	Operação	Sinais de controle	Resultado esperado
c.i	Condições iniciais	reset=0 iniciar=0 chaves=0000	pronto=x db_contagem=xxxx db_memoria=xxxx=leds db_jogada =xxxx db_rodada = xxxx db_estado=xxxx
1	Resetar o circuito	reset=1 iniciar=0 chaves=0000	pronto=0 db_contagem=0000 db_memoria=0001=leds db_jogada =0000 db_rodada = 0000 db_estado=0000 (inicial) endereçolgualRodada = 1 db_jogada_correta=0
2	Iniciar jogo por 5 períodos de clock e esperar	reset=0 iniciar=1 chaves=0000	pronto=0 db_contagem=0000 db_memoria=0001=leds db_jogada =0000 db_rodada = 0000 endereçolgualRodada = 1 db_jogada_correta=0 db_estado=3 - 2- 1(preparação, início_rodada e espera)

3	Ajustar chaves para 0001 por 10 períodos de clock (Primeira rodada - 1 jogada)	reset=0 iniciar=0 chaves=0001	pronto=0 db_contagem=0000 db_memoria=0001 leds = 0001 db_jogada =0001 db_rodada = 0000 endereçolgualRodada = 1 db_jogada_correta=1 db_estado=4-5-7-8- 2-1 (registra, comparação, última_rodada,, próxima_rodada, início_rodada e espera)
4 e 5	Ajustar chaves para 0001 e 0010 por 10 períodos de clock cada (Segunda rodada - 2 jogadas)	reset=0 iniciar=0 chaves=0001-0010	pronto=0 db_contagem=0000-0001 db_memoria=0001-0010 leds = 0010 db_jogada =0001-0010 db_rodada = 0001 endereçolgualRodada = 0 - 1 db_jogada_correta=1 db_estado=4-5-6-1-4-5-7-8- 2-1
6, 7 e 8	Ajustar chaves para 0001, 0010 e 0100 por 10 períodos de clock cada (Terceira rodada - 3 jogadas)	reset=0 iniciar=0 chaves=0001-0010-0100	pronto=0 db_contagem=0000-0001-0010 db_memoria=0001-0010-0100 leds = 0100 db_jogada =0001-0010-0100 db_rodada = 0010 endereçolgualRodada = 0 –0- 1 db_jogada_correta=1 db_estado=4-5-6-1-4-5-6-1-4-5-7-8- 2-1
9 e 10	Ajustar chaves para 0001 e 0010, esperando 5000 períodos de clock para dar a segunda jogada (timeout)	reset=0 iniciar=0 chaves=0001-0010	pronto=0-1 db_timeout = 0-1 db_contagem=0000-0001 db_memoria=0001-0010 leds = 1000 db_jogada =0001 db_rodada = 0011 endereçolgualRodada = 0-1-0 db_jogada_correta=1 db_estado=4-5-6-1-B

Tabela 3: Plano de testes do timeout

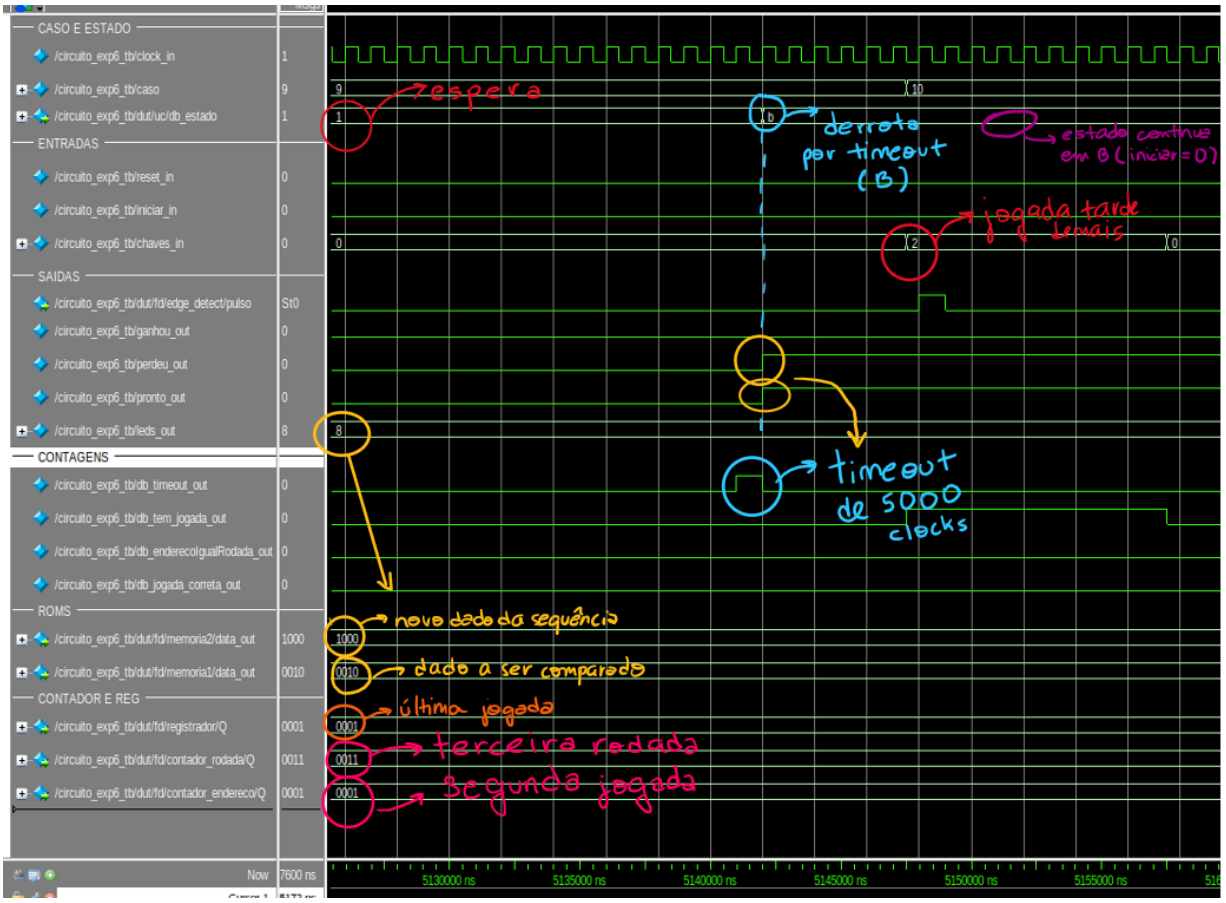


Figura 20: Simulação no Modelsim do cenário de derrota por timeout

1.2- Atividade 2 - Implementação e Síntese do Sistema Digital

Abaixo está a tabela com a pinagem que será utilizada para programar a placa DE0-CV, juntamente com os respectivos pinos do analog discovery. Repare que neste primeiro momento não designamos pinos para a saída de depuração db_Q, pois necessitaríamos de mais displays do que os disponíveis. Assim, utilizaremos esta saída apenas se isto se provar necessário durante o laboratório.

Sinal	**Pino na Placa DE0-CV**	**Pino no FPGA**	**Analog Discovery**
CLOCK	GPIO_0_D0	PIN_N16	StaticIO – LED – DIO0 e Patterns – Clock – 1 kHz
RESET	GPIO_0_D1	PIN_B16	StaticIO – Button 0/1
INICIAR	chave SW0	PIN_C16	StaticIO – Button 0/1 – DIO2
CHAVES(0)	GPIO_0_D11	PIN_R22	StaticIO – Button 0/1 – DIO4
CHAVES(1)	GPIO_0_D13	PIN_T22	StaticIO – Button 0/1 – DIO5
CHAVES(2)	GPIO_0_D15	PIN_N19	StaticIO – Button 0/1 – DIO6
CHAVES(3)	GPIO_0_D17	PIN_P19	StaticIO – Button 0/1 – DIO7

LEDS(0)	GPIO_1_D17	PIN_A15	StaticIO – LED– DIO12
LEDS(1)	GPIO_1_D19	PIN_L8	StaticIO – LED– DIO12
LEDS(2)	GPIO_1_D21	PIN_B15	StaticIO – LED– DIO14
LEDS(3)	GPIO_1_D23	PIN_E14	StaticIO – LED– DIO15
PRONTO	GPIO_1_D15	PIN_J11	StaticIO – LED– DIO10
PERDEU	GPIO_1_D11	PIN_J18	StaticIO – LED– DIO8
GANHOU	GPIO_1_D13	PIN_G11	StaticIO – LED– DIO9
DB_JOGADA_CORRETA	led LEDR2	PIN_W2	
DB_CLOCK	led LEDR0	PIN_AA2	
DB_TEM_JOGADA	led LEDR1	PIN_AA1	
db_enderecolgualRodada	led LEDR3	PIN_Y3	
DB_TIMEOUT	led LEDR4	PIN_N2	
DB_CONTAGEM	display HEX0	PIN_U21	
DB_MEMORIA	display HEX1	PIN_AA20	
DB_JOGADAFEITA	display HEX2	PIN_Y19	
DB_RODADA	display HEX3	PIN_Y16	
DB_ESTADO	display HEX5	PIN_N9	

Tabela 4: Designação dos pinos no pin planner

Em laboratório, após a programação pela funcionalidade programmer do Intel Quartus Prime na placa DE0-CV, iremos realizar os três planos de testes descritos anteriormente e verificar seus resultados no mundo real.

Abaixo seguem imagens dos GPIOs da placa DE0-CV e do Analog Discovery para facilitar sua identificação em laboratório.

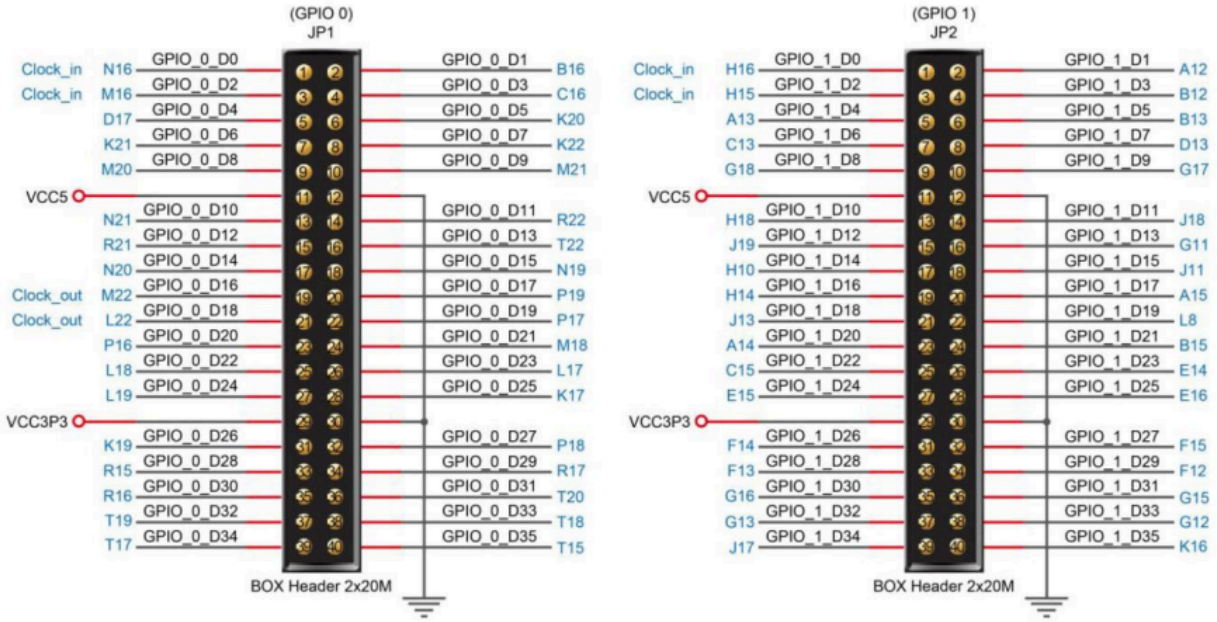


Figure 3-12 I/O distribution of the expansion headers