



ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO  
Departamento de Engenharia de Computação e Sistemas Digitais

## PCS3635 – LABORATÓRIO DIGITAL I

### EXPERIÊNCIA 5 – Projeto de um jogo de sequências de jogadas

Planejamento da Bancada B1 – Turma 3 – Prof. Antonio

Data de Emissão: 01 de Fevereiro de 2025

<b>Nome:</b> Ana Vitória Abreu Murad	<b>Número USP:</b> 14613160
<b>Nome:</b> Heitor Gama Ribeiro	<b>Número USP:</b> 14577494
<b>Nome:</b> Yasmin Francisquetti Barnes	<b>Número USP:</b> 13828230

Tempo gasto no planejamento: 15h30min

#### 1 INTRODUÇÃO

Esta experiência tem como objetivo aprofundar o conhecimento dos alunos em circuitos digitais, retomando o circuito desenvolvido anteriormente e introduzindo um mecanismo de repetição de sequências a cada jogada. O experimento depende da implementação anterior e apresenta atualizações e modificações para aprimorar seu funcionamento. Entre as mudanças, destaca-se a substituição do sinal *iniciar* pelo sinal *jogar*, do sinal *chaves* pelo sinal *botoes* e dos sinais *acertou* e *errou* pelos sinais *ganhou* e *perdeu*. Além disso, o comportamento do sistema foi alterado para exigir que o jogador não apenas pressione um botão correspondente ao valor armazenado no endereço atual da memória, mas também reproduza corretamente toda a sequência de jogadas anteriores. Dessa forma, na primeira rodada, o jogador insere um valor correspondente ao primeiro endereço da memória; na segunda rodada, deve repetir o primeiro valor e, em

seguida, inserir o segundo; na terceira, repetir os dois primeiros valores e adicionar o terceiro, e assim sucessivamente, até alcançar toda a sequência de 16 posições. Os módulos serão projetados em Verilog, testados no software *ModelSim* e, posteriormente, compilados no *Intel Quartus Prime*. Nesse ambiente, também será realizada a pinagem da placa, e o circuito será sintetizado na FPGA DE0-CV.

## 2 DESCRIÇÃO DO PROJETO

O projeto consiste na adaptação do sistema digital desenvolvido na experiência anterior, incorporando novas funcionalidades que aproximam sua lógica de funcionamento ao jogo *Genius*. Além das funcionalidades já testadas, como o *timeout* da jogada e a introdução do estado de espera para permitir que o jogador tenha tempo de pressionar um botão, esta versão adiciona uma nova sequência de operações que depende não apenas do estado atual, mas também dos estados anteriores. À medida que o jogador avança no jogo, ele retorna ao início da memória e precisa repetir corretamente todas as posições anteriores antes de "adivinhar" a próxima. Dessa forma, o sistema exige a memorização progressiva da sequência, aumentando gradativamente a dificuldade.

O sistema digital sequencial desenvolvido consiste em um circuito que armazena um conjunto de 16 dados de 4 bits em uma memória interna. Esses dados são acessados sequencialmente por meio de um contador interno, responsável pelo endereçamento da memória. O funcionamento do circuito tem início com a ativação do sinal *reset*, após o qual ele aguarda o acionamento do sinal *jogar* para iniciar sua operação. Quando acionado, o sistema espera a entrada de uma jogada, que ocorre quando o jogador pressiona um dos botões de entrada. O valor inserido pelo jogador é então armazenado e exibido nos LEDs de saída e comparado com o dado correspondente na memória.

A lógica do jogo segue uma estrutura progressiva. Na primeira rodada, apenas o primeiro dado da memória é verificado. Na segunda rodada, os dois primeiros dados são comparados. Esse processo continua até que todas as 16 posições da memória tenham sido verificadas sequencialmente. Se todas as jogadas daquela sequência estiverem corretas, um contador interno de limites é incrementado, e a sequência se expande. Caso o jogador cometa um erro em qualquer etapa, o circuito interrompe a execução e ativa a saída *perdeu*. Caso o jogador demore mais que 3 segundos para pressionar o próximo botão, as saídas *timeout* e *errou* são ativadas. Se todas as jogadas forem acertadas até o final do jogo, a saída *ganhou* é acionada. Em todos os casos, a saída *pronto* é ativada para indicar o término da operação. O sistema permanece no estado final até que uma

nova ativação do sinal *jogar* reinicie o jogo.

### 3 DETALHAMENTO DO PROJETO LÓGICO

Este capítulo do relatório tem a finalidade de descrever detalhes do projeto lógico desenvolvido durante o planejamento da experiência.

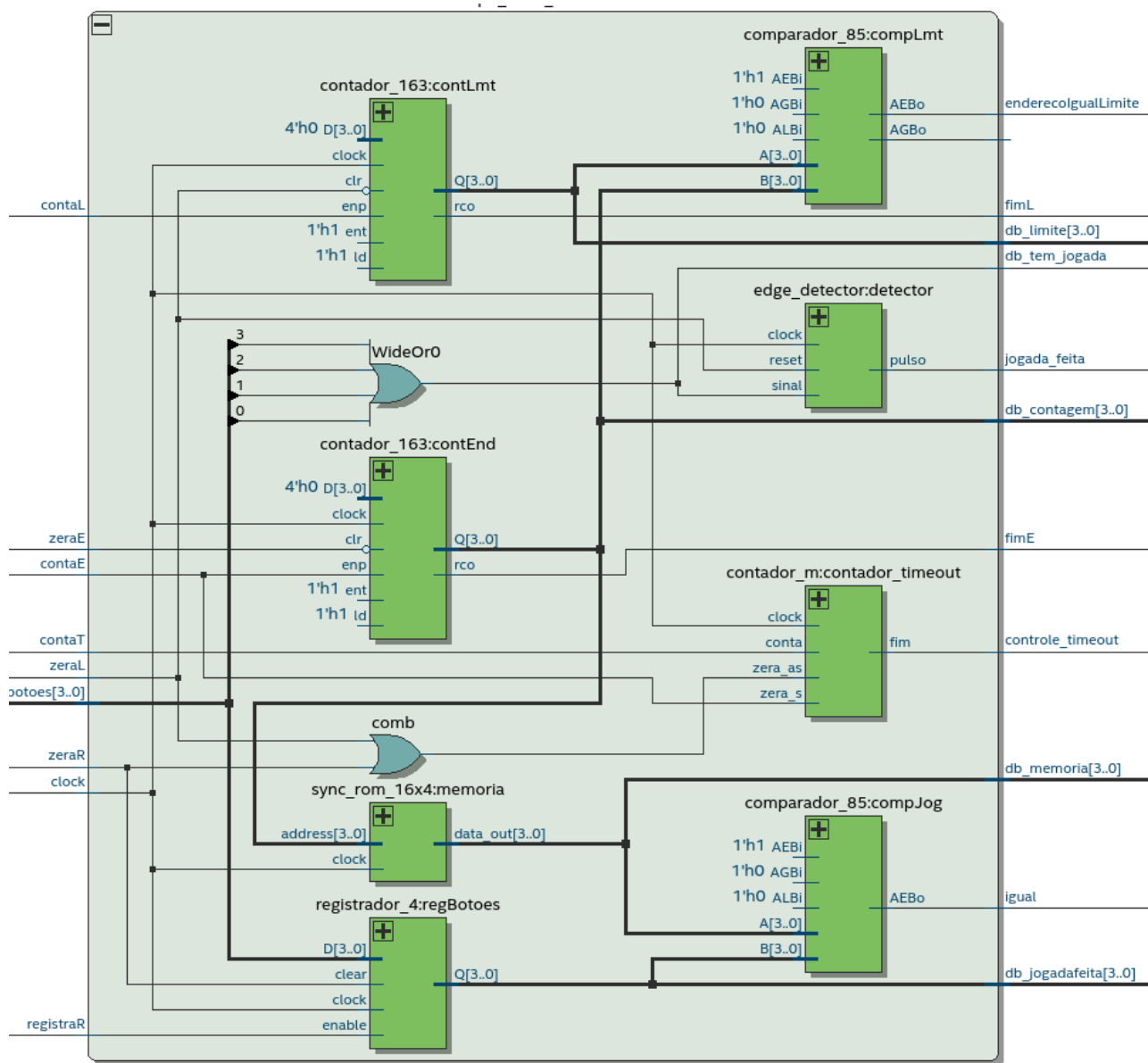
#### 3.1 PROJETO DO FLUXO DE DADOS

O Fluxo de Dados (FD) da experiência anterior foi projetado para incorporar as funcionalidades de detecção de borda (*edge detector*) e *timeout* da jogada após três segundos de espera. Para isso, contava com os seguintes componentes:

- *contador\_163*: responsável pela contagem das posições de memória a serem comparadas;
- *comparador\_85*: encarregado de verificar se o valor das chaves coincide com o armazenado na memória;
- *sync\_rom\_16x4*: memória ROM interna pré-programada com 16 dados de 4 bits;
- *registrador\_4*: registrador para armazenar o valor das chaves inseridas;
- *edge\_detector*: responsável por detectar mudanças no sinal e liberar um pulso entre dois ciclos de clock;
- *controle\_m*: responsável por contar os ciclos de clock durante o estado de espera de três segundos.

Na nova implementação, o módulo *edge\_detector* tornou-se um componente obrigatório para o funcionamento do sistema, pois o projeto também será alimentado por um clock de 1 kHz durante a execução prática. A funcionalidade de *timeout*, embora opcional nesta fase, foi mantida para garantir a integridade do experimento e para a integração com as etapas seguintes.

Para suportar a nova estrutura do sistema, foi necessário adicionar instâncias adicionais dos módulos “*contador\_163*” e “*comparador\_85*”, denominadas “*contLmt*” (contador de limite) e “*compLmt*” (comparador de limite), enquanto as versões originais passaram a ser chamadas de “*contEnd*” e “*compEnd*”. O componente “*contLmt*” controla a contagem do limite da sequência em uma determinada jogada e sinaliza à Unidade de Controle (UC) quando esse limite é atingido. Todos os novos componentes e ajustes foram integrados ao circuito e compilados em Verilog. A Figura 1 apresenta o diagrama de blocos do FD do *RTLViewer* gerado após a compilação no software *Quartus*.



**Figura 1 – Diagrama de Blocos do Fluxo de Dados do Sistema**

Os sinais de entrada e de saída do componente “contLmt”, como foram nomeados no Diagrama de Blocos estão descritos a seguir:

- *clock*: recebe o *clock* do sistema;
- *~zeraL*: sinal de controle que zera o limite ao reiniciar uma partida;
- *contaL*: acionado para incrementar o limite da sequência sempre que a próxima jogada começa;
- *fimL*: sinal de saída que informa que a última sequência está sendo comparada;
- *s\_limite*: sinal de saída que representa a sequência em execução (varia de 0 a F), e determina o sinal de depuração *db\_limite*.

O componente “compLmt” realiza a comparação entre o limite atual (*s\_limite*) e a posição de memória atualmente verificada (*s\_endereco*), tendo os seguintes sinais:

- *s\_limite*: entrada A do comparador, indica a sequência atual;
- *s\_endereco*: entrada B do comparador, representa a posição de memória sendo lida;
- *enderecoMenorOuIgualLimite*: saída que indica que o endereço ainda não atingiu o limite da sequência;
- *endereçoIgualLimite*: saída que indica que a sequência foi concluída.

Além disso, a escolha do sinal de entrada do *edge detector* pelos alunos (anteriormente chamado *zeraC*, agora renomeado para *zeraE*) precisou ser revisada. Isso ocorre porque o contador é reiniciado diversas vezes durante uma partida, sempre que uma nova sequência começa. Essa reinicialização comprometeria o funcionamento do *edge detector*, que poderia gerar mais de 1 pulso por jogada feita.

### 3.2 PROJETO DA UNIDADE DE CONTROLE

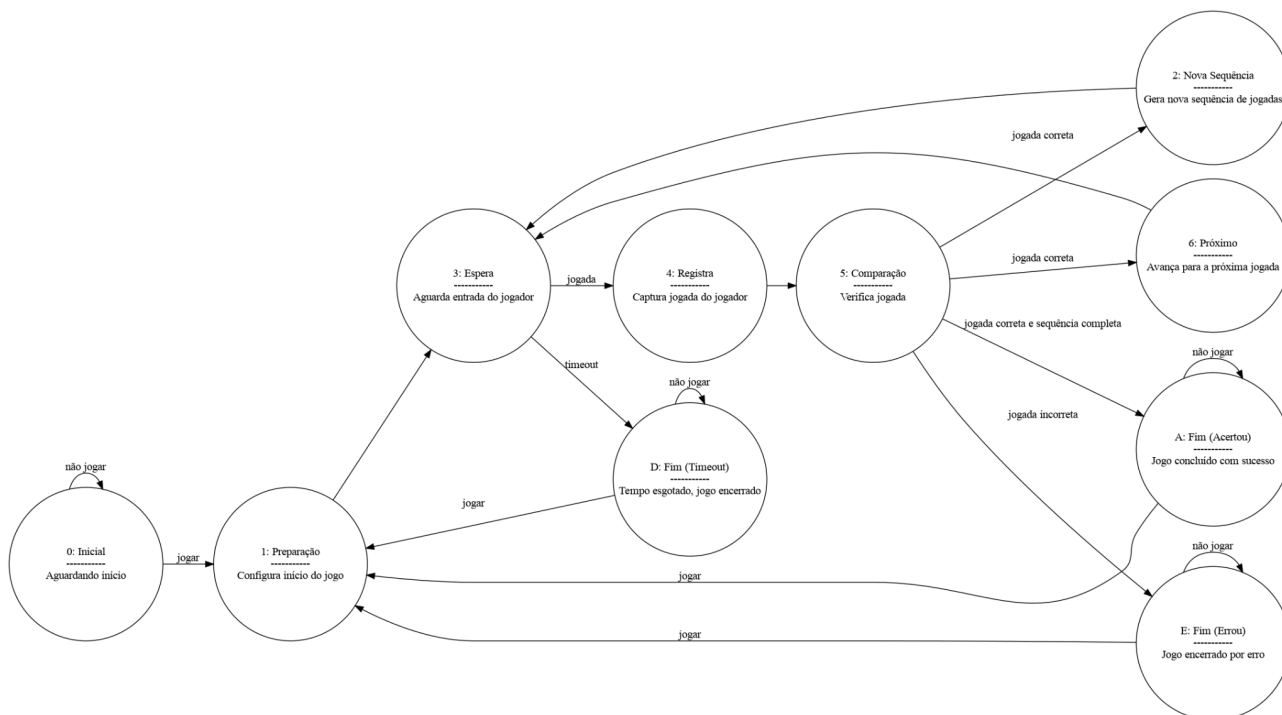
A Unidade de Controle (UC) foi implementada após a elaboração dos diagramas de alto nível e de transição de estados. Durante esse processo, identificou-se a necessidade de um novo estado para atender às funcionalidades exigidas pela atualização do projeto lógico. Assim, foi introduzido o estado “nova\_seq”, conforme ilustrado no diagrama de transição de estados da Figura 3. Esse novo estado funciona como um estágio intermediário, no qual:

- O contador é reinicializado (*zeraE* é ativado), permitindo o reinício da comparação e do ciclo de contagem;
- O sinal *contaL* é acionado, incrementando o contador “contLmt” e aumentando em uma unidade o limite da próxima sequência.

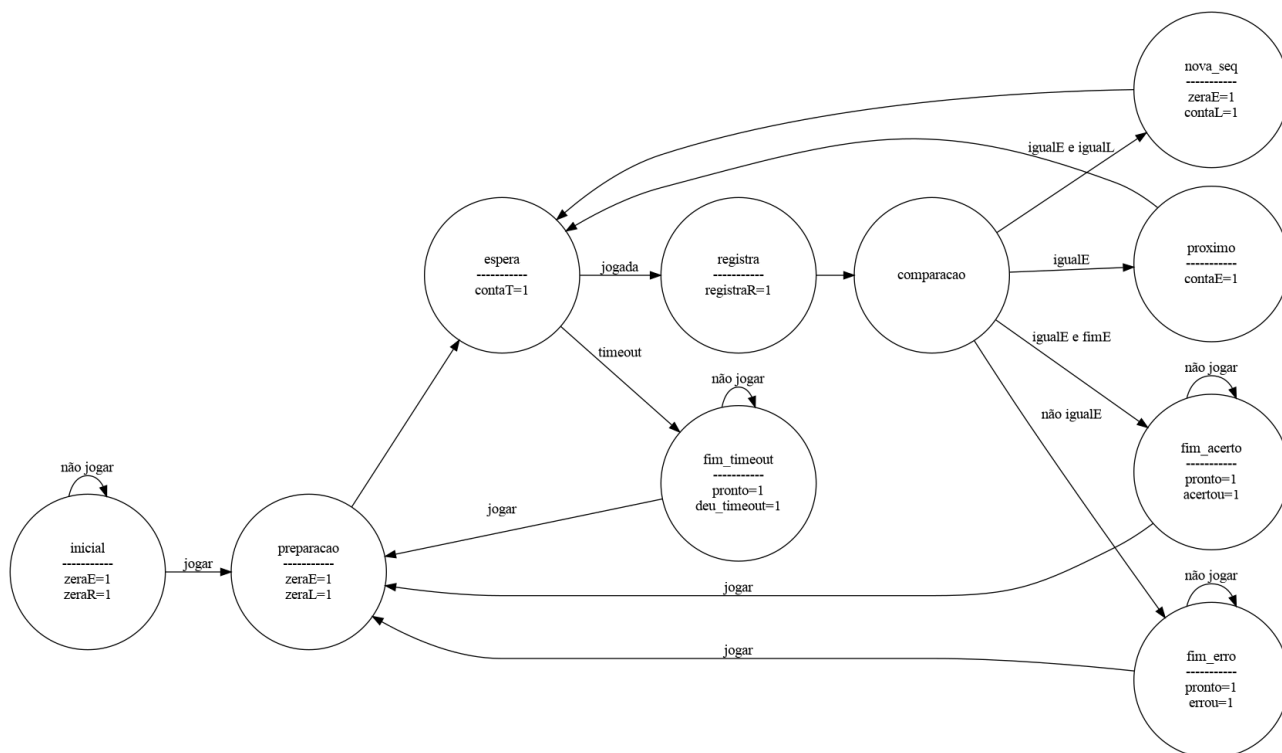
A transição para o estado “nova\_seq” ocorre após o estado de comparação, desde que as seguintes condições sejam atendidas:

- Os sinais *igualL* e *igualE* estejam ativados, indicando o término da sequência atual e a igualdade entre os dados comparados, permitindo a continuidade do jogo;
- O sinal *fimE* esteja desativado, indicando que o jogo ainda não terminou, ou seja, que não se trata da última jogada global.

O uso dos diagramas das Figuras 2 e 3 auxiliou na compreensão dessa dinâmica. Inicialmente, discutiu-se a possibilidade de adicionar mais estados intermediários às transições. No entanto, após análises e testes simulados, concluiu-se que essa adição era desnecessária, e a solução pôde ser simplificada para um único estado, como planejado.



**Figura 2 – Diagrama de Alto Nível - UC**



**Figura 3 – Diagrama de Transição de Estados - UC**

Esses diagramas foram gerados usando uma biblioteca de visualização de grafos do python chamada [Graphviz](#), que gera a imagem a partir das definições em nós e arestas. Para melhor visualização dos diagramas, eles também foram incluídos ao final do documento de forma vetorizada.

A Tabela 1 apresenta todas as transições de estado, incluindo o novo estado adicionado, enquanto as Figuras 4 e 5 ilustram a conversão do diagrama de transição de estados para Verilog com os sinais de controle envolvidos e a transição para o estado novo, respectivamente.

**Tabela 1 – Descrição da Unidade de Controle do Sistema**

Nome do Estado	Descrição do Estado	Próximo Estado	Condições e Justificativas para a Transição entre Estados
inicial	Estado inicial. Zera o contador de endereço (zeraE) e o registrador (zeraR).	preparacao	Transita para preparacao quando o sinal jogar está ativo. Mantém-se em inicial se jogar não estiver ativo.
preparacao	Prepara o sistema para aguardar uma jogada. Zera o contador de sequência (zeraL).	espera	Transita para espera automaticamente após executar as ações do estado preparacao.
nova_seq	Gera nova sequência de jogadas. Zera o contador de endereço (zeraE) e incrementa o contador de sequência (contaL).	espera	Transita para espera automaticamente após executar as ações do estado nova_seq.
espera	Aguardando uma jogada do jogador.	registra ou fim_timeout	Transita para registra quando o sinal jogada está ativo. Transita para fim_timeout se o tempo esgotar.
registra	Registra o valor da jogada no registrador (registraR).	comparacao	Transita para comparacao automaticamente após executar as ações do estado registra.
comparacao	Compara a jogada com o valor esperado na memória.	fim_acerto ou proximo ou fim_erro	- Vai para fim_acerto se igualE e fimE estiverem ativos. - Vai para proximo se apenas igualE estiver ativo. - Vai para fim_erro se igualE estiver inativo.
proximo	Avança o contador de endereço (contaE) para processar a próxima jogada.	espera	Transita para espera automaticamente após executar as ações do estado proximo.
fim_acerto	Indica que todas as comparações foram bem-sucedidas. Sinaliza pronto e acertou.	preparacao	Transita para preparacao se o sinal jogar estiver ativo. Mantém-se em fim_acerto enquanto jogar não estiver ativo.
fim_erro	Indica que houve um erro em alguma comparação. Sinaliza pronto e errou.	preparacao	Transita para preparacao se o sinal jogar estiver ativo. Mantém-se em fim_erro enquanto jogar não estiver ativo.
fim_timeout	Indica que o tempo esgotou antes de uma jogada ser feita. Sinaliza pronto e deu_timeout.	preparacao	Transita para preparacao se o sinal jogar estiver ativo. Mantém-se em fim_timeout enquanto jogar não estiver ativo.

```
// Logica de saida (maquina Moore)
always @* begin
    contaL      = (Eatual == nova_seq) ? 1'b1 : 1'b0;
    zeraE       = (Eatual == inicial || Eatual == preparacao || Eatual == nova_seq) ? 1'b1 : 1'b0;
```

Figura 4 – Tradução em Verilog dos Sinais de Controle

```
comparacao: Eprox = igualE ? (fimE ? fim_acerto : (iguall ? nova_seq : proximo)) : fim_erro;
nova_seq:    Eprox = espera;
```

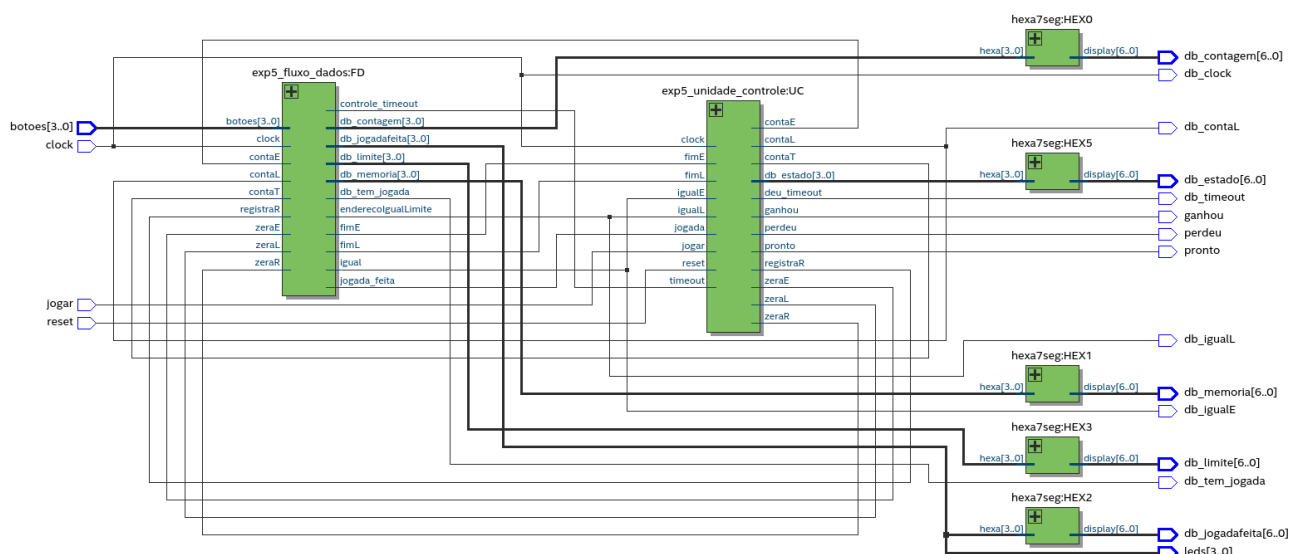
Figura 5 – Tradução em Verilog da Transição de Estados

### 3.3 PROJETO DO SISTEMA DIGITAL

A integração entre o fluxo de dados e a unidade de controle no circuito desenvolvido garante a comunicação necessária para coordenar corretamente as operações do sistema. A unidade de controle recebe sinais de entrada, como o *clock*, *reset* e comandos do usuário, e gera sinais de controle que comandam o fluxo de dados. Entre esses sinais de controle, destacam-se *zeraR* e *registraR*, que resetam e registram as jogadas nos registradores internos, os sinais *contaE*, *contaL* e *contaT*, utilizados para incrementar os contadores de endereço, de limite e de *timeout*, respectivamente, e os sinais *zeraE* e *zeraL*, que reinicializam os contadores de endereços e de limite. Os sinais de condição *igualE* (endereço == contagem) e *iguall* (limite == contagem) indicam quando os valores comparados são iguais, permitindo a transição de estado. Além disso, *fimE*, *fimL* e *timeout* sinalizam o término da contagem ou um estouro de tempo, permitindo que a unidade de controle reaja adequadamente a essas condições.

Para facilitar a depuração e a análise do funcionamento do circuito, foram adicionados sinais de monitoramento que permitem acompanhar internamente o estado do sistema. Os sinais *db\_estado*, *db\_contagem*, *db\_memoria* e *db\_jogadafeita* fornecem informações detalhadas sobre a execução, exibindo o estado atual da unidade de controle, a contagem dos ciclos, o conteúdo da memória e a jogada realizada. O sinal *db\_limite* exibe o limite da sequência atual, enquanto *db\_iguall* e *db\_igualE* indicam o resultado das comparações realizadas. Os sinais *db\_tem\_jogada* e *db\_timeout* monitoram se há uma jogada em andamento e se ocorreu um estouro de tempo. Além disso, *db\_clock* permite acompanhar o *clock* do sistema, garantindo a sincronização das operações. Essa estrutura de depuração possibilita verificar o comportamento do circuito em tempo real, facilitando a validação e o ajuste da lógica implementada. Todos os sinais de depuração do circuito completo podem ser vistos com clareza no Diagrama de Blocos gerado pela ferramenta RTLViewer, como mostra a Figura 6.





**Figura 6 – Diagrama de Blocos do Circuito Completo**

#### 4 PLANO DE TESTES DO SISTEMA E SIMULAÇÕES

Para validar de forma abrangente os novos estados do circuito, foram elaborados três casos de teste distintos. O primeiro visa verificar o cenário em que todas as jogadas são realizadas corretamente. O segundo avalia diferentes tipos de erro, incluindo falhas na quarta jogada, reinício do jogo e erros na segunda jogada, este avalia, além do caso de erro, o funcionamento do circuito ao reiniciar a partir de um estado final. O terceiro caso testa o funcionamento do sistema com o mecanismo de *timeout*. É importante ressaltar que nem todos os sinais de saída estão especificados detalhadamente nos resultados esperados. Assim, para os sinais que não forem explicitamente mencionados em uma célula do plano de testes, deve-se assumir que seu valor será 0.

##### 4.1 CENÁRIO DE TESTE 1 – SIMULAÇÃO DE ACERTO DO CIRCUITO COMPLETO - GANHOU

Validação do resultado de acerto das 16 sequências de dados da memória, utilizando a testbench “circuito\_exp5\_tb1.v” e o software *ModelSim* para simulação.

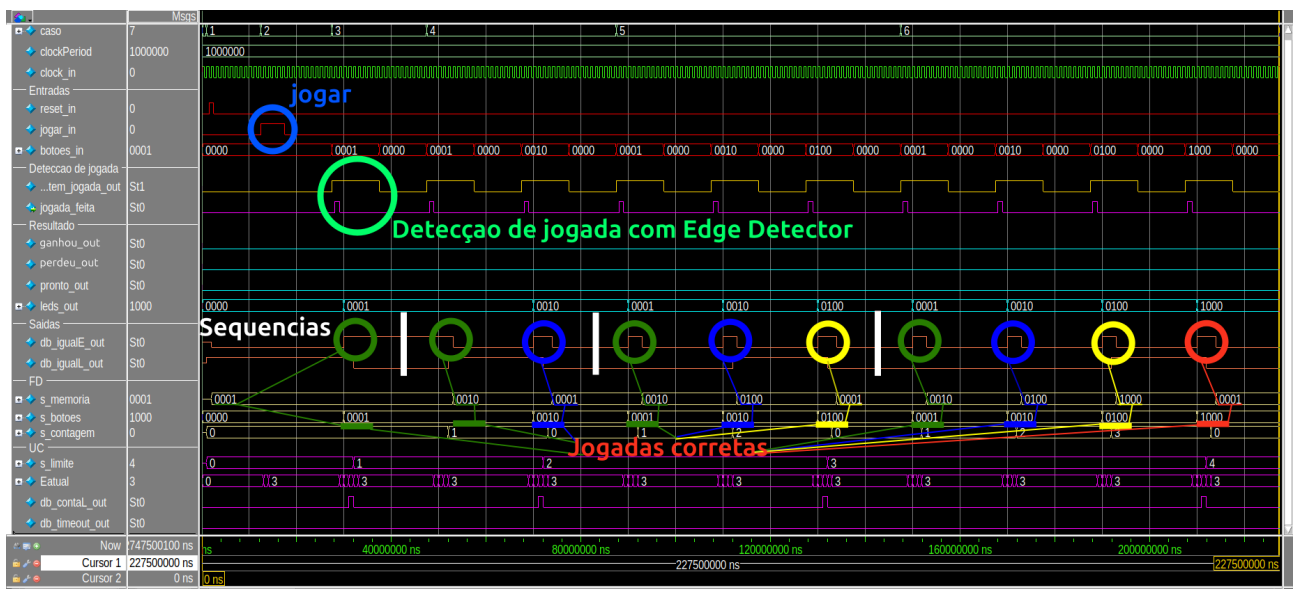
**Tabela 2 – Descrição e Resultados Simulados do Cenário de Teste 1**

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?
c.i.	Condições iniciais	-	jogar=0, reset=0	Sim
1	“Resetar” circuito e aguardar alguns segundos	acionar reset	jogar=0, reset=1	Sim
2	Acionar sinal jogar	acionar jogar	jogar=1	Sim

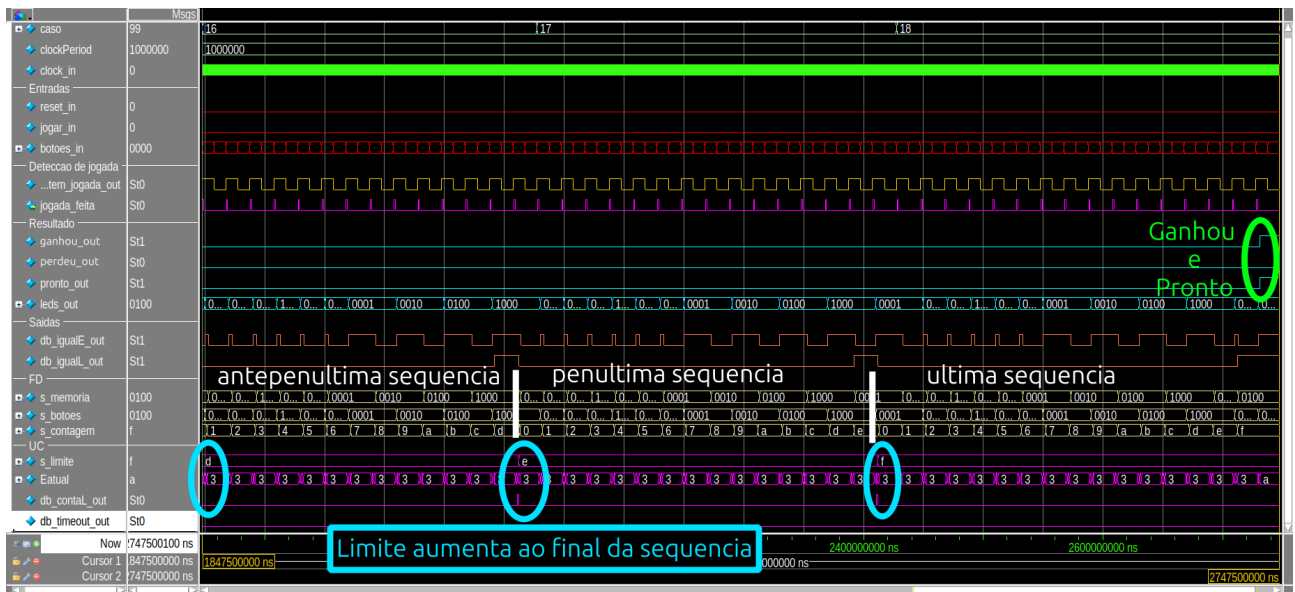
#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?
3	Acionar primeira entrada (jogada 1)	acionar botoes (0)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=0, db_igualL=1	Sim
4	Acionar segunda entrada (jogada 2)	acionar botoes (0) (1)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=1,db_igualL=1	Sim
5	Acionar terceira entrada (jogada 3)	acionar botoes (0) (1) (2)	db_memoria=8, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=4, db_limite=2, db_igualL=1	Sim
6	Acionar quarta entrada (jogada 4)	acionar botoes (0) (1) (2) (3)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=8, db_limite=3, db_igualL=1	Sim
7	Acionar quinta entrada (jogada 5)	acionar botoes (0) (1) (2) (3) (2)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=4, db_limite=4, db_igualL=1	Sim
8	Acionar sexta entrada (jogada 6)	acionar botoes (0) (1) (2) (3) (2) (1)	db_memoria=1, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=5, db_igualL=1	Sim
9	Acionar sétima entrada (jogada 7)	acionar botoes (0) (1) (2) (3) (2) (1) (0)	db_memoria=1, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=6, db_igualL=1	Sim
10	Acionar oitava entrada (jogada 8)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=7, db_igualL=1	Sim
11	Acionar nona entrada (jogada 9)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=8, db_igualL=1	Sim
12	Acionar décima entrada (jogada 10)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=9, db_igualL=1	Sim
13	Acionar décima primeira entrada	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2,	Sim

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?
	(jogada 11)		db_jogada=4, db_limite=10, db_igualL=1	
14	Acionar décima segunda entrada (jogada 12)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2)	db_memoria=8, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=4, db_limite=11, db_igualL=1	Sim
15	Acionar décima terceira entrada (jogada 13)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2) (3)	db_memoria=8, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=8, db_limite=12, db_igualL=1	Sim
16	Acionar décima quarta entrada (jogada 14)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2) (3) (3)	db_memoria=1, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=8, db_limite=13, db_igualL=1	Sim
17	Acionar décima quinta entrada (jogada 15)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2) (3) (3) (0)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=14, db_igualL=1	Sim
18	Acionar décima sexta entrada (jogada 16)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2) (3) (3) (0) (2)	db_memoria=4, db_estado= ordem 4, 5 e A, db_jogada=4, db_limite=15, pronto=1, ganhou=1	Sim

- Sequências 1, 2, 3 e 4, com 4 jogadas diferentes (verde, azul, amarelo e vermelho)



- Tela final com o resultado *ganhou* e visão geral das três últimas sequências



## 4.2 CENÁRIO DE TESTE 2 – SIMULAÇÃO DE ERRO DO CIRCUITO COMPLETO - PERDEU

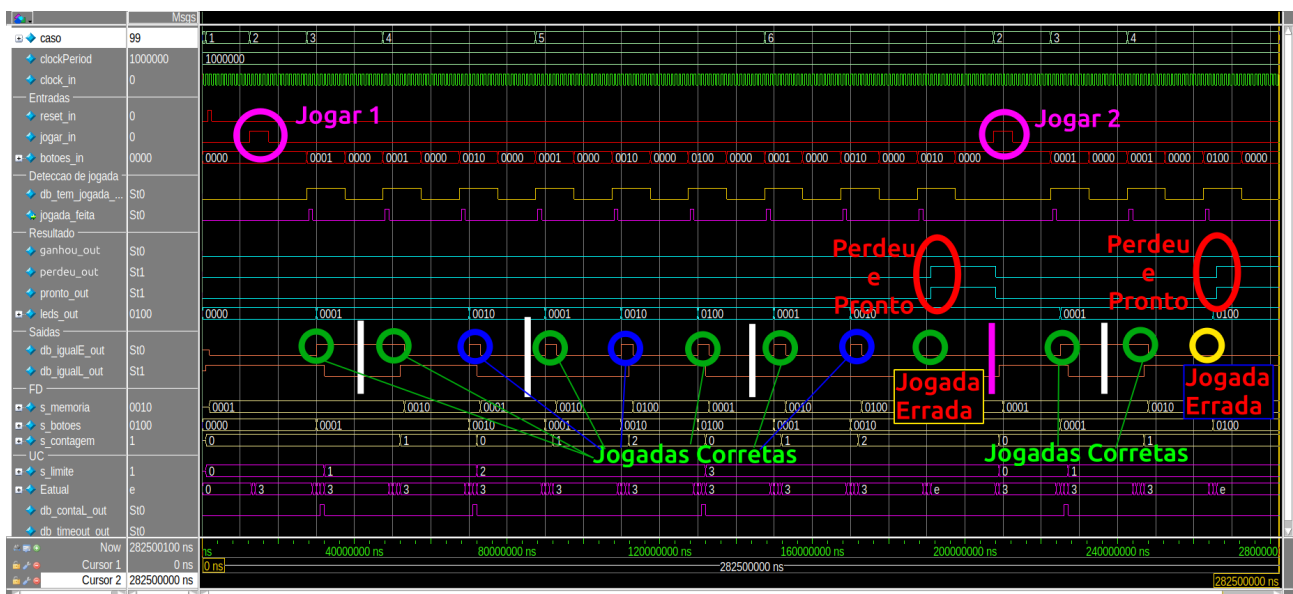
Validação do cenário de erro no 3º dado da quarta sequência, reinício do sistema com a ativação de *jogar* e erro no 2º dado da segunda sequência, utilizando a testbench “circuito\_exp5\_tb2.v” e o software *ModelSim* para simulação.

**Tabela 3 – Descrição e Resultados Simulados do Cenário de Teste 2**

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?
c.i.	Condições iniciais	-	jogar=0, reset=0	Sim
1	“Resetar” circuito e aguardar alguns segundos	acionar reset	jogar=0, reset=1	Sim
2	Acionar sinal jogar	acionar jogar	jogar=1	Sim
3	Acionar primeira entrada (jogada 1)	acionar botoes (0)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=0, db_igualL=1	Sim
4	Acionar segunda entrada (jogada 2)	acionar botoes (0) (1)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=1, db_igualL=1	Sim
5	Acionar terceira entrada (jogada 3)	acionar botoes (0) (1) (2)	db_memoria=8, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=4, db_limite=2, db_igualL=1	Sim
6	Acionar quarta entrada (jogada 4)	acionar botoes (0) (1) (1)	db_memoria=4, db_estado= ordem 4, 5 e E, db_jogada=2, db_limite=3, db_igualL=0, pronto=1, errou=1	Sim

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?
7	Acionar sinal jogar	acionar jogar	jogar=1	Sim
8	Acionar primeira entrada (jogada 1)	acionar botoes (0)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=0, db_igualL=1	Sim
9	Acionar segunda entrada (jogada 2)	acionar botoes (0) (2)	db_memoria=4, db_estado= ordem 4, 5 e E, db_jogada=2, db_limite=1, db_igualL=0, pronto=1, errou=1	Sim

- Visão Geral dos testes com duas jogadas erradas indicadas



#### 4.3 CENÁRIO DE TESTE 3 – SIMULAÇÃO DE TIMEOUT DO CIRCUITO COMPLETO - PERDEU

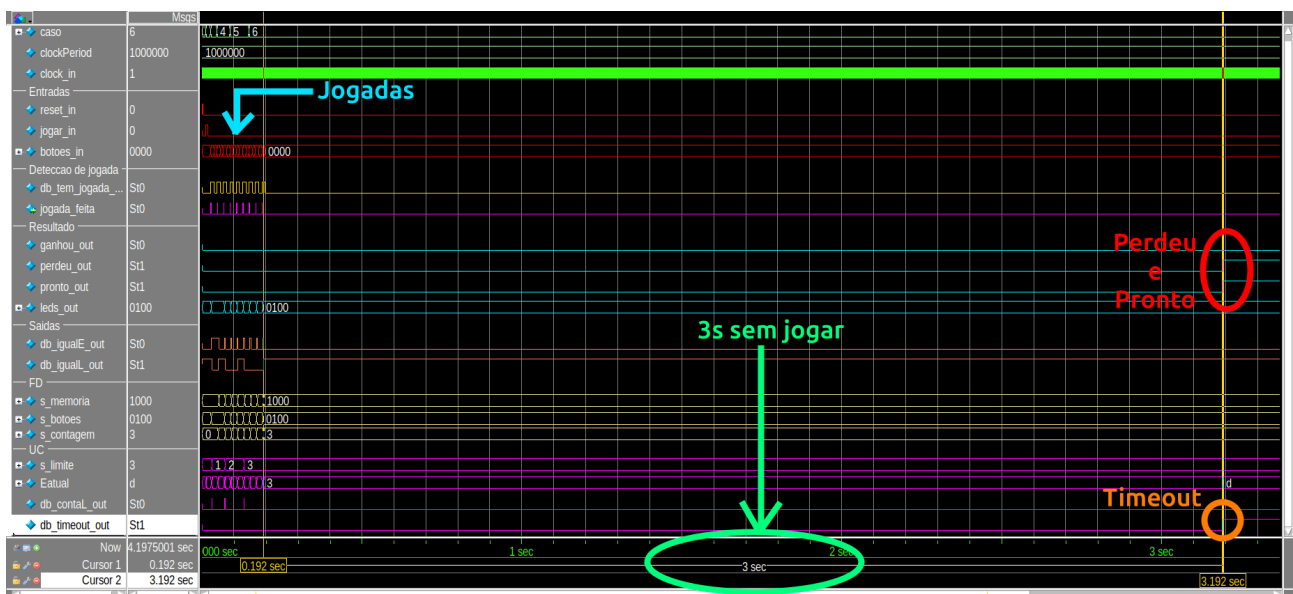
Validação do cenário de timeout do 3º dado da quarta sequência, utilizando a testbench “circuito\_exp5\_tb3.v” e o software *ModelSim* para simulação.

Tabela 4 – Descrição e Resultados Simulados do Cenário de Teste 3

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?
c.i.	Condições iniciais	-	jogar=0, reset=0	Sim
1	“Resetar” circuito e aguardar alguns segundos	acionar reset	jogar=0, reset=1	Sim
2	Acionar sinal jogar	acionar jogar	jogar=1	Sim
3	Acionar primeira entrada (jogada 1)	acionar botoes (0)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=0, db_igualL=1	Sim

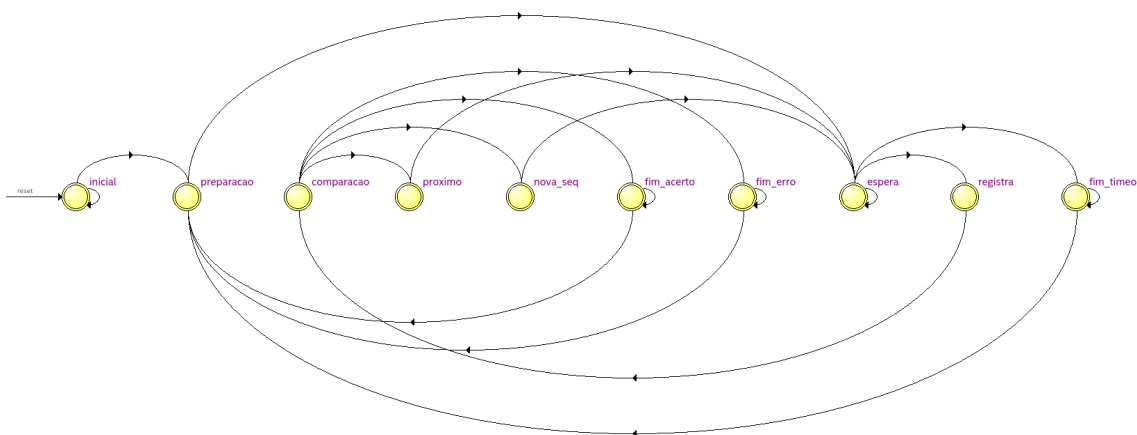
#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?
4	Acionar segunda entrada (jogada 2)	acionar botoes (0) (1)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=1, db_igualL=1	Sim
5	Acionar terceira entrada (jogada 3)	acionar botoes (0) (1) (2)	db_memoria=8, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=4, db_limite=2, db_igualL=1	Sim
6	Acionar quarta entrada (jogada 4)	acionar botoes (0) (1) (2) e aguardar	db_memoria=4, db_estado= ordem 4, 5 e D, db_jogada=4, db_limite=3, db_igualL=0, pronto=1, db_timeout=1	Sim

- Visão geral dos testes



## 5 IMPLANTAÇÃO DO PROJETO

O projeto foi compilado no software *Quartus* e as respectivas imagens de RTL Viewer do sistema digital completo e do fluxo de dados foram apresentadas nas seções 3.3 e 3.1, respectivamente. Para visualização da máquina de estados criada, foi utilizada a ferramenta *StateMachineViewer*, que gerou o diagrama da Figura 7.



**Figura 7 – Diagrama gerado pelo *StateMachineViewer* do Quartus**

Vale ressaltar que inicialmente o *StateMachineViewer* gerou um diagrama vazio, ou seja, ele não reconheceu a máquina de estados da UC. A razão disso é devido à seguinte linha de código da lógica de controle:

```
zeraL      = (Eatual == jogar || Eatual == preparacao) ? 1'b1 : 1'b0;
```

**Figura 8 – Tradução em Verilog da Transição de Estados**

O erro é dado pela condição *Eatual == jogar* que, apesar de não fazer sentido algum logicamente, é inofensiva do ponto de vista operacional do circuito digital. Isso é verdade pois *jogar* é um sinal binário de 1 bit, então essa condição faz o limite zerar caso o estado atual coincida com esse sinal que só pode assumir 0 ou 1 e, no código, esses estados foram definidos como *inicial* e *preparação* respectivamente. Como o estado *preparação* já zeraria o limite de qualquer forma, e o *inicial* transita para o *preparação* antes de toda partida, esse erro não afeta de maneira alguma o funcionamento do circuito e, por isso, é praticamente impossível detectá-lo via simulação. A única maneira seria perceber que o sinal *zeraL* ativa no estado *inicial* quando o *jogar* está desligado, o que é algo difícil de se observar pois só acontece uma vez por acionamento do circuito. Porém, como *Eatual* é comparado a um sinal de entrada, algo não esperado por uma variável de estado, o Quartus interpreta ela como uma variável combinatória qualquer e não detecta a FSM.

Para depurar esse erro, foi adotada a seguinte estratégia:

1. Restaurar a UC para um ponto em que a funcionalidade do *StateMachineViewer* ainda funcionava. O ponto escolhido foi a UC do experimento 4.
2. Trocar pedaços do código da UC antiga por pedaços da UC nova problemática, monitorando a funcionalidade do *StateMachineViewer* a cada iteração, até que ele

pare de funcionar.

- Quando o problema reaparece, o último pedaço substituído é muito provavelmente a causa dele. Por isso, repetimos o processo iterativo neste novo pedaço de código, até que ele se reduza a uma única linha ou a uma porção pequena o suficiente para que se perceba a raiz do erro.

Após a simples remoção da condição *Eatual == jogar* para ativar *zeraL*, mantendo apenas *Eatual == preparacao*, a geração da FSM pelo Quartus ficou novamente disponível.

## 5.1 PINAGEM DA PLACA FPGA

A configuração da pinagem da placa FPGA foi realizada levando em conta os novos sinais de depuração definidos pelos alunos. Durante a montagem, identificou-se uma quantidade insuficiente de LEDs na placa para exibir todos os sinais de depuração desejados. Para contornar essa limitação, os LEDs serão monitorados por meio do *Analog Discovery*, utilizando o software *WaveForms* para visualização.

**Tabela 5 – Designação de pinos na FPGA**

Sinal	Pino na Placa DE0-CV	Pino no FPGA	Analog Discovery
BOTOES(0)	GPIO_0_D19	PIN_P17	StaticIO – Button 0/1 – DIO3
BOTOES(1)	GPIO_0_D21	PIN_M18	StaticIO – Button 0/1 – DIO4
BOTOES(2)	GPIO_0_D23	PIN_L17	StaticIO – Button 0/1 – DIO5
BOTOES(3)	GPIO_0_D25	PIN_K17	StaticIO – Button 0/1 – DIO6
CLOCK	GPIO_0_D13	PIN_T22	StaticIO – LED – DIO0 Patterns – Clock – 1KHz
DB_CLOCK	Led LEDR5	PIN_N1	-
DB_CONTAGEM (0)	Display HEX00	PIN_U21	-
DB_CONTAGEM (1)	Display HEX01	PIN_V21	-
DB_CONTAGEM (2)	Display HEX02	PIN_W22	-
DB_CONTAGEM (3)	Display HEX03	PIN_W21	-
DB_CONTAGEM (4)	Display HEX04	PIN_Y22	-
DB_CONTAGEM (5)	Display HEX05	PIN_Y21	-
DB_CONTAGEM (6)	Display HEX06	PIN_AA22	-
DB_CONTAL	GPIO_0_D31	PIN_T20	StaticIO – LED – DIO8
DB_ESTADO (0)	Display HEX50	PIN_N9	-
DB_ESTADO (1)	Display HEX51	PIN_M8	-



Sinal	Pino na Placa DE0-CV	Pino no FPGA	Analog Discovery
DB_ESTADO (2)	Display HEX52	PIN_T14	-
DB_ESTADO (3)	Display HEX53	PIN_P14	-
DB_ESTADO (4)	Display HEX54	PIN_C1	-
DB_ESTADO (5)	Display HEX55	PIN_C2	-
DB_ESTADO (6)	Display HEX56	PIN_W19	-
DB_IGUALE	Led LEDR4	PIN_N2	-
DB_IGUALL	GPIO_0_D27	PIN_P18	StaticIO – LED – DIO9
DB_JOGADA (0)	Display HEX20	PIN_Y19	-
DB_JOGADA (1)	Display HEX21	PIN_AB17	-
DB_JOGADA (2)	Display HEX22	PIN_AA10	-
DB_JOGADA (3)	Display HEX23	PIN_Y14	-
DB_JOGADA (4)	Display HEX24	PIN_V14	-
DB_JOGADA (5)	Display HEX25	PIN_AB22	-
DB_JOGADA (6)	Display HEX26	PIN_AB21	-
DB_LIMITE (0)	Display HEX30	PIN_Y16	-
DB_LIMITE (1)	Display HEX31	PIN_W16	-
DB_LIMITE (2)	Display HEX32	PIN_Y17	-
DB_LIMITE (3)	Display HEX33	PIN_V16	-
DB_LIMITE (4)	Display HEX34	PIN_U17	-
DB_LIMITE (5)	Display HEX35	PIN_V18	-
DB_LIMITE (6)	Display HEX36	PIN_V19	-
DB_MEMORIA (0)	Display HEX10	PIN_AA20	-
DB_MEMORIA (1)	Display HEX11	PIN_AB20	-
DB_MEMORIA (2)	Display HEX12	PIN_AA19	-
DB_MEMORIA (3)	Display HEX13	PIN_AA18	-
DB_MEMORIA (4)	Display HEX14	PIN_AB18	-
DB_MEMORIA (5)	Display HEX15	PIN_AA17	-
DB_MEMORIA (6)	Display HEX16	PIN_U22	-
DB_TEM_JOGADA	Led LEDR6	PIN_U2	-
DB_TIMEOUT	GPIO_0_D29	PIN_R17	StaticIO – LED – DIO7
GANHOU	Led LEDR8	PIN_L2	-
JOGAR	GPIO_0_D17	PIN_P19	StaticIO – Button 0/1 – DIO2
LEDS(0)	Led LEDR0	PIN_AA2	-
LEDS(1)	Led LEDR1	PIN_AA1	-

Sinal	Pino na Placa DE0-CV	Pino no FPGA	Analog Discovery
LEDS(2)	Led LEDR2	PIN_W2	-
LEDS(3)	Led LEDR3	PIN_Y3	-
PERDEU	Led LEDR7	PIN_U1	-
PRONTO	Led LEDR9	PIN_L1	-
RESET	GPIO_0_D15	PIN_N19	StaticIO – Button 0/1 – DIO1

Top View - Wire Bond  
Cyclone V - 5CEBA4F23C7

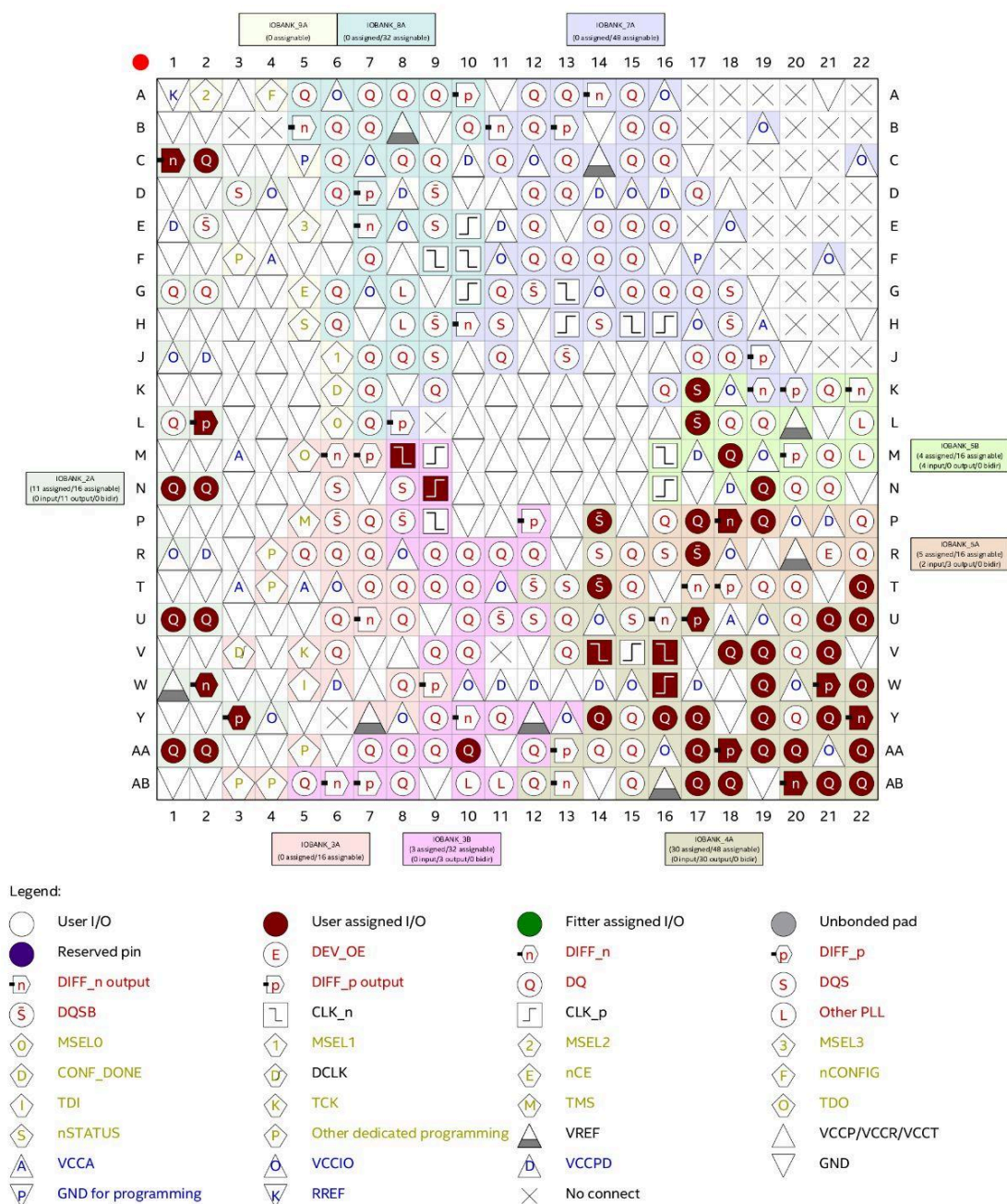


Figura 9 – Top-View da placa FPGA

## 5.2 ESTRATÉGIA DE MONTAGEM

A montagem experimental do circuito digital na FPGA DE0-CV seguirá os seguintes passos:

1. Configuração do Ambiente:
  - Certificar-se de que o software *Intel Quartus Prime* está instalado e configurado corretamente.
  - Verificar que a placa FPGA DE0-CV está conectada ao computador via cabo USB e garantir que o driver esteja instalado.
  - Abrir o projeto no Quartus Prime e verificar a integridade do código Verilog.
2. Geração e Programação do Arquivo de Configuração:
  - Compilar o código Verilog no Quartus Prime para gerar o arquivo .sof correspondente à programação da FPGA.
  - Utilizar o Quartus Programmer para carregar o arquivo .sof na placa FPGA.
3. Configuração dos Pinos da FPGA:
  - Configurar as conexões dos sinais físicos da placa conforme a tabela de pinagem definida no planejamento.
  - Certificar-se de que os botões, LEDs e saídas digitais estão corretamente atribuídos.
4. Ligação de Terminais e Fios:
  - Conectar os sinais de entrada, como botões e clock, aos pinos da FPGA.
  - Conectar os sinais de saída aos LEDs e displays da placa para visualização das respostas.
5. Monitoramento dos Sinais Físicos:
  - Verificar os estados dos LEDs para acompanhar a execução do circuito.
  - Utilizar um osciloscópio ou analisador lógico para verificar os sinais internos críticos, como clock e reset.
6. Sinais de Depuração:
  - Monitorar sinais internos através do RTL Viewer e do *StateMachineViewer* no Quartus Prime.
  - Utilizar displays de 7 segmentos e LEDs para indicar os estados da máquina de controle.
  - Observar saídas de depuração (*db\_estado*, *db\_memoria*, *db\_limite*, etc.) para facilitar a verificação do funcionamento.

### 5.3 ESTRATÉGIA DE DEPURAÇÃO

A depuração será realizada de forma sistemática, garantindo que os sinais monitorados estejam coerentes com o comportamento esperado do circuito. As etapas incluem:

1. Monitoramento dos sinais principais de depuração:
  - a. *db\_contagem* e *db\_memoria* – Verificar se o contador de sequência está avançando corretamente e se os valores da memória estão sendo acessados na ordem correta.
  - b. *db\_igualE* e *db\_igualL* – Confirmar se o sistema está comparando corretamente as jogadas do usuário com a sequência armazenada na memória.
  - c. *db\_estado* – Garantir que a máquina de estados transite adequadamente entre os estados conforme o esperado.
  - d. *db\_limite* – Garantir que os limites estão de acordo com as sequências correspondentes ou se há algum comportamento inesperado.
  - e. *pronto*, *ganhou*, *perdeu* e *db\_timeout* – Analisar os LEDs correspondentes para validar os resultados finais do circuito.
2. Utilização do Analog Discovery:
  - a. Conectar o *Analog Discovery* ao circuito para capturar e observar os sinais internos, incluindo bordas de clock e transições de estado.
  - b. Utilizar o *WaveForms* para monitorar a frequência e estabilidade do sinal de clock e validar os tempos de resposta dos sinais de controle.
3. Caso os resultados esperados não sejam obtidos, serão adotadas as seguintes estratégias:
  - a. Validação no *ModelSim*: Executar as simulações detalhadas para identificar possíveis falhas na lógica do circuito (já feito no planejamento, mas pode ocorrer algum erro com um sinal de depuração não verificado). Comparar os sinais simulados com os sinais capturados na FPGA.
  - b. Verificação no *Quartus Prime*: Revisar os sinais no RTL Viewer e no *StateMachineViewer* para confirmar a lógica implementada. Examinar o Pin Planner para garantir que as atribuições dos pinos estão corretas.
4. Correção na Montagem Física:
  - a. Verificar se os botões de entrada e LEDs de saída estão corretamente

conectados.

b. Testar a continuidade dos cabos e eliminar possíveis mau contatos.

#### 5. Ajustes no Clock para Depuração Manual:

Se os resultados esperados (como os sinais *ganhou* ou *perdeu*) não forem apresentados corretamente, uma abordagem será alterar temporariamente o clock para um modo de operação manual ou de baixa frequência (1 Hz). Isso permitirá que cada transição de estado seja verificada individualmente, facilitando a depuração do circuito.

### 5.4 EXECUÇÃO PRÁTICA DO CENÁRIO DE TESTE 1 – TESTE DE GANHAR NA FPGA

Validação do resultado de acerto das 16 sequências de dados da memória, utilizando o clock de 1KHz no circuito sintetizado na placa FPGA.

**Tabela 6 – Descrição e Resultados Práticos do Cenário de Teste 1**

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado prático ok?
c.i.	Condições iniciais	-	jogar=0, reset=0	
1	“Resetar” circuito e aguardar alguns segundos	acionar reset	jogar=0, reset=1	
2	Acionar sinal jogar	acionar jogar	jogar=1	
3	Acionar primeira entrada (jogada 1)	acionar botoes (0)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=0, db_igualL=1	
4	Acionar segunda entrada (jogada 2)	acionar botoes (0) (1)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=1,db_igualL=1	
5	Acionar terceira entrada (jogada 3)	acionar botoes (0) (1) (2)	db_memoria=8, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=4, db_limite=2, db_igualL=1	
6	Acionar quarta entrada (jogada 4)	acionar botoes (0) (1) (2) (3)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=8, db_limite=3, db_igualL=1	
7	Acionar quinta entrada (jogada 5)	acionar botoes (0) (1) (2) (3) (2)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=4, db_limite=4, db_igualL=1	
8	Acionar sexta	acionar botoes (0) (1)	db_memoria=1, db_estado= ordem	

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado prático ok?
	entrada (jogada 6)	(2) (3) (2) (1)	4, 5, 6, 7 e 2, db_jogada=2, db_limite=5, db_igualL=1	
9	Acionar sétima entrada (jogada 7)	acionar botoes (0) (1) (2) (3) (2) (1) (0)	db_memoria=1, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=6, db_igualL=1	
10	Acionar oitava entrada (jogada 8)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=7, db_igualL=1	
11	Acionar nona entrada (jogada 9)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=8, db_igualL=1	
12	Acionar décima entrada (jogada 10)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=9, db_igualL=1	
13	Acionar décima primeira entrada (jogada 11)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=4, db_limite=10, db_igualL=1	
14	Acionar décima segunda entrada (jogada 12)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2)	db_memoria=8, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=4, db_limite=11, db_igualL=1	
15	Acionar décima terceira entrada (jogada 13)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2) (3)	db_memoria=8, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=8, db_limite=12, db_igualL=1	
16	Acionar décima quarta entrada (jogada 14)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2) (3) (3)	db_memoria=1, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=8, db_limite=13, db_igualL=1	
17	Acionar décima quinta entrada (jogada 15)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2) (3) (3) (0)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=14, db_igualL=1	
18	Acionar décima sexta entrada (jogada 16)	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2) (3) (3) (0) (2)	db_memoria=4, db_estado= ordem 4, 5 e A, db_jogada=4, db_limite=15, pronto=1, ganhou=1	

## 5.5 EXECUÇÃO PRÁTICA DO CENÁRIO DE TESTE 2 - TESTE DE PERDER NA FPGA

Teste na FPGA do cenário de erro no 3º dado da quarta sequência, reinício do sistema com a ativação de *jogar* e erro no 2º dado da segunda sequência.

Tabela 7 – Descrição e Resultados Práticos do Cenário de Teste 2

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado prático ok?
c.i.	Condições iniciais	-	jogar=0, reset=0	
1	“Resetar” circuito e aguardar alguns segundos	acionar reset	jogar=0, reset=1	
2	Acionar sinal jogar	acionar jogar	jogar=1	
3	Acionar primeira entrada (jogada 1)	acionar botoes (0)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=0, db_igualL=1	
4	Acionar segunda entrada (jogada 2)	acionar botoes (0) (1)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=1, db_igualL=1	
5	Acionar terceira entrada (jogada 3)	acionar botoes (0) (1) (2)	db_memoria=8, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=4, db_limite=2, db_igualL=1	
6	Acionar quarta entrada (jogada 4)	acionar botoes (0) (1) (1)	db_memoria=4, db_estado= ordem 4, 5 e E, db_jogada=2, db_limite=3, db_igualL=0, pronto=1, errou=1	
7	Acionar sinal jogar	acionar jogar	jogar=1	
8	Acionar primeira entrada (jogada 1)	acionar botoes (0)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=0, db_igualL=1	
9	Acionar segunda entrada (jogada 2)	acionar botoes (0) (2)	db_memoria=4, db_estado= ordem 4, 5 e E, db_jogada=2, db_limite=1, db_igualL=0, pronto=1, errou=1	

## 5.6 EXECUÇÃO PRÁTICA DO CENÁRIO DE TESTE 3 - TESTE DE TIMEOUT NA FPGA

Teste na FPGA do cenário de *Timeout* no 3º dado da quarta sequência.

Tabela 8 – Descrição e Resultados Práticos do Cenário de Teste 3

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado prático ok?
c.i.	Condições iniciais	-	jogar=0, reset=0	
1	“Resetar” circuito e aguardar alguns segundos	acionar reset	jogar=0, reset=1	
2	Acionar sinal jogar	acionar jogar	jogar=1	
3	Acionar primeira entrada (jogada 1)	acionar botoes (0)	db_memoria=2, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=1, db_limite=0, db_igualL=1	
4	Acionar segunda entrada (jogada 2)	acionar botoes (0) (1)	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=1,db_igualL=1	
5	Acionar terceira entrada (jogada 3)	acionar botoes (0) (1) (2)	db_memoria=8, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=4, db_limite=2, db_igualL=1	
6	Acionar quarta entrada (jogada 4)	acionar botoes (0) (1) (2) e aguardar	db_memoria=4, db_estado= ordem 4, 5 e D, db_jogada=4, db_limite=3, db_igualL=0, pronto=1, db_timeout=1	

## 6 PROJETO DO DESAFIO DA EXPERIÊNCIA

O enunciado para o projeto do Desafio da Experiência 5 será disponibilizado durante a aula do Laboratório Digital. Nessas condições, as próximas seções serão preenchidas somente no Relato Final.

### 6.1 Descrição do Desafio

### 6.2 Descrição do Projeto Lógico

### 6.3 Alterações do Fluxo de Dados

### 6.4 Alterações da Unidade de Controle

### 6.5 Alterações do Sistema Digital

### 6.6 Alterações do Sistema Digital

### 6.7 Verificação e Validação do Desafio

### 6.8 Cenário de Teste 1



## 6.9 Cenário de Teste 2 –

## 7 CONCLUSÕES

Até o momento do planejamento, todos os testes realizados cumpriram os objetivos definidos para a experiência. Foram executados testes que simularam diferentes cenários de funcionamento do circuito, abrangendo diversas situações para garantir que o sistema opere conforme o esperado. O novo mecanismo de repetição progressiva da sequência foi validado em simulações, demonstrando que o circuito é capaz de armazenar e comparar corretamente os valores inseridos pelo jogador, expandindo a sequência conforme o avanço no jogo.

Os testes permitiram verificar a integração entre o fluxo de dados e a unidade de controle, bem como a funcionalidade das saídas de depuração, como *db\_contagem*, *db\_memoria*, *db\_estado*, *db\_botoes*, *db\_limite* e *db\_jogada*. Os resultados obtidos validaram o comportamento do sistema em situações representativas, incluindo as transições entre estados e os sinais de finalização, como *pronto*, *ganhou* e *perdeu*. Além disso, a funcionalidade de timeout foi testada para garantir que a jogada seja encerrada corretamente caso o tempo máximo seja atingido.

A abordagem adotada assegurou a identificação e correção de possíveis inconsistências, deixando o grupo preparado para a etapa de implementação prática durante a aula. A expectativa é que, ao seguir a estratégia de montagem e depuração planejada, o circuito funcione corretamente na FPGA DE0-CV, permitindo a validação prática de todas as funcionalidades do sistema.

