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

Relato da Bancada B1 – Turma 3 – Prof. Antonio

Data de Emissão: 01 de Fevereiro de 2025

Nome: Ana Vitória Abreu Murad	**Número USP:** 14613160
Nome: Heitor Gama Ribeiro	**Número USP:** 14577494
Nome: Yasmin Francisquetti Barnes	**Número USP:** 13828230

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*.

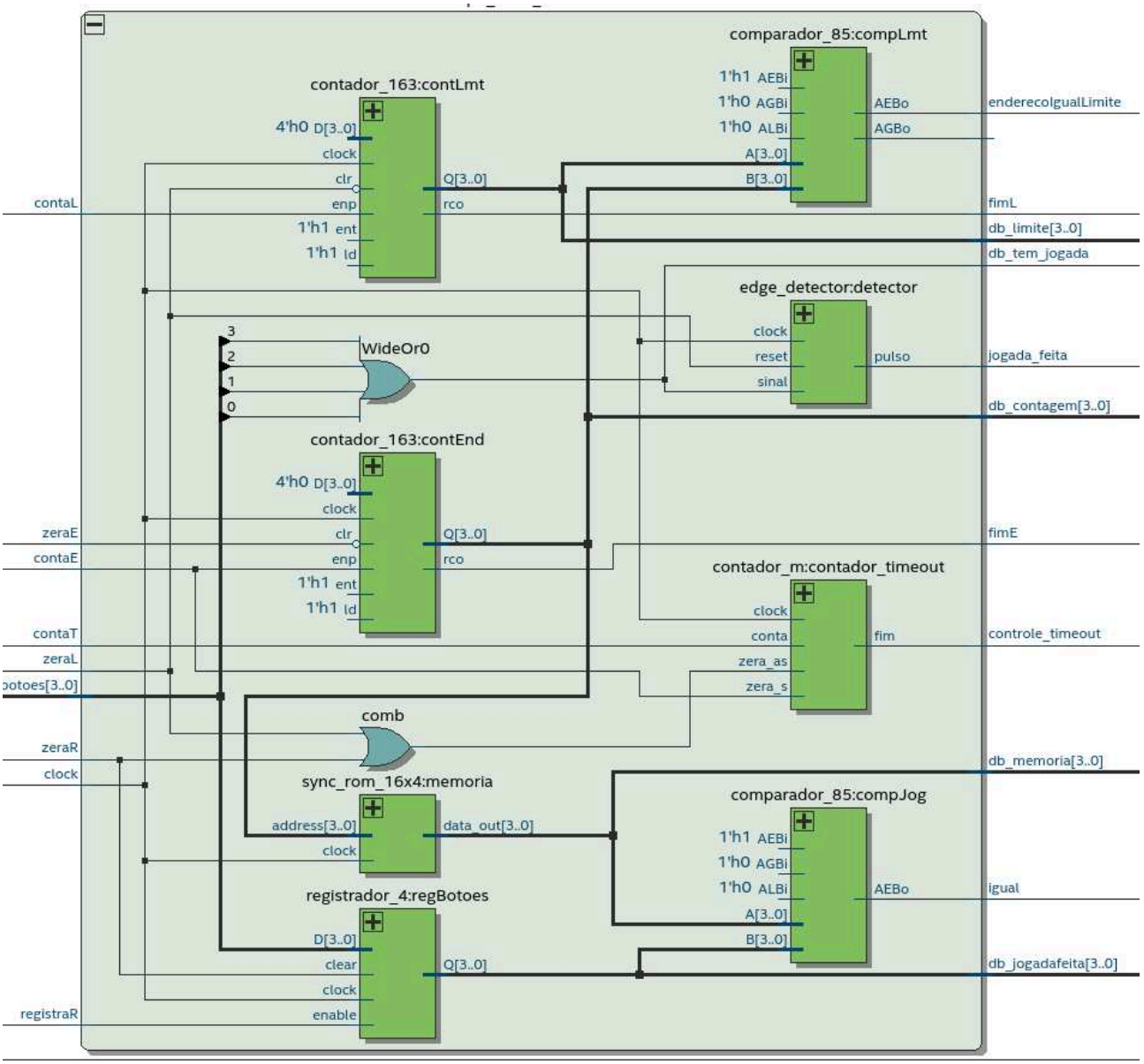


Figura 1 – Diagrama de Blocos do Fluxo de Dados do SIstema

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;
- *enderecoIgualLimite*: 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.

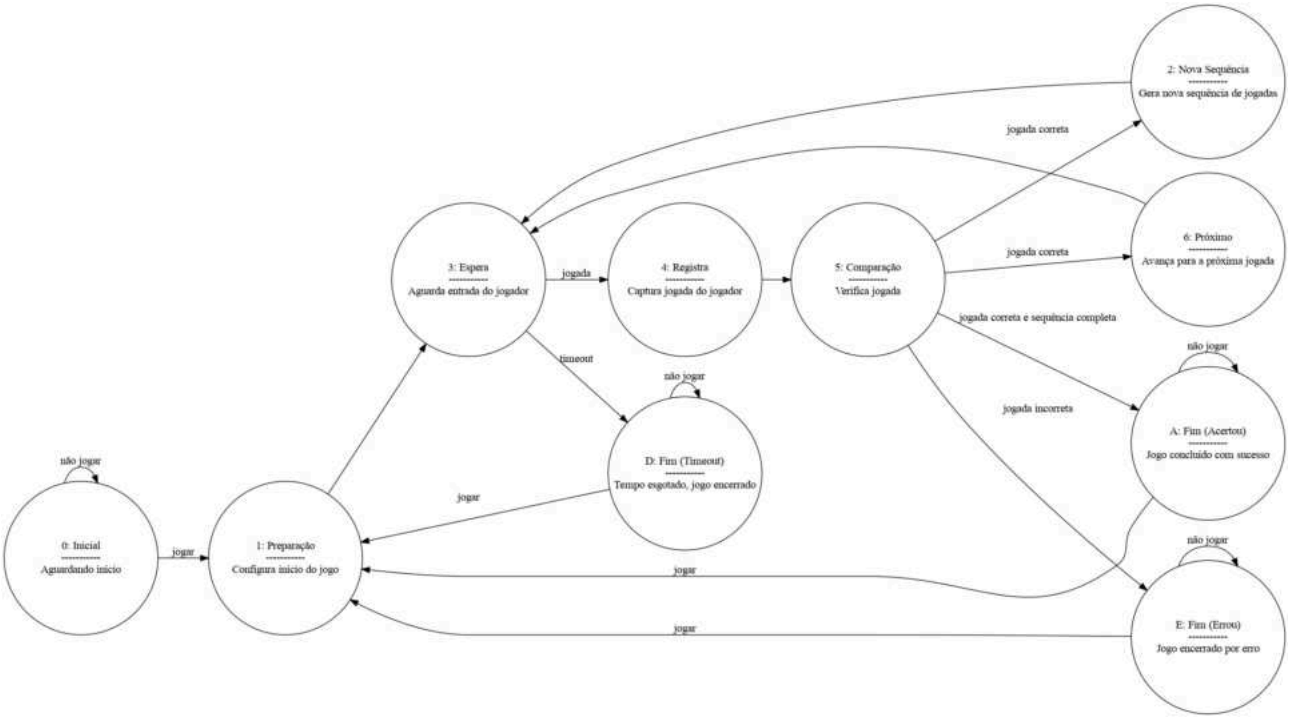


Figura 2 – Diagrama de Alto Nível - UC

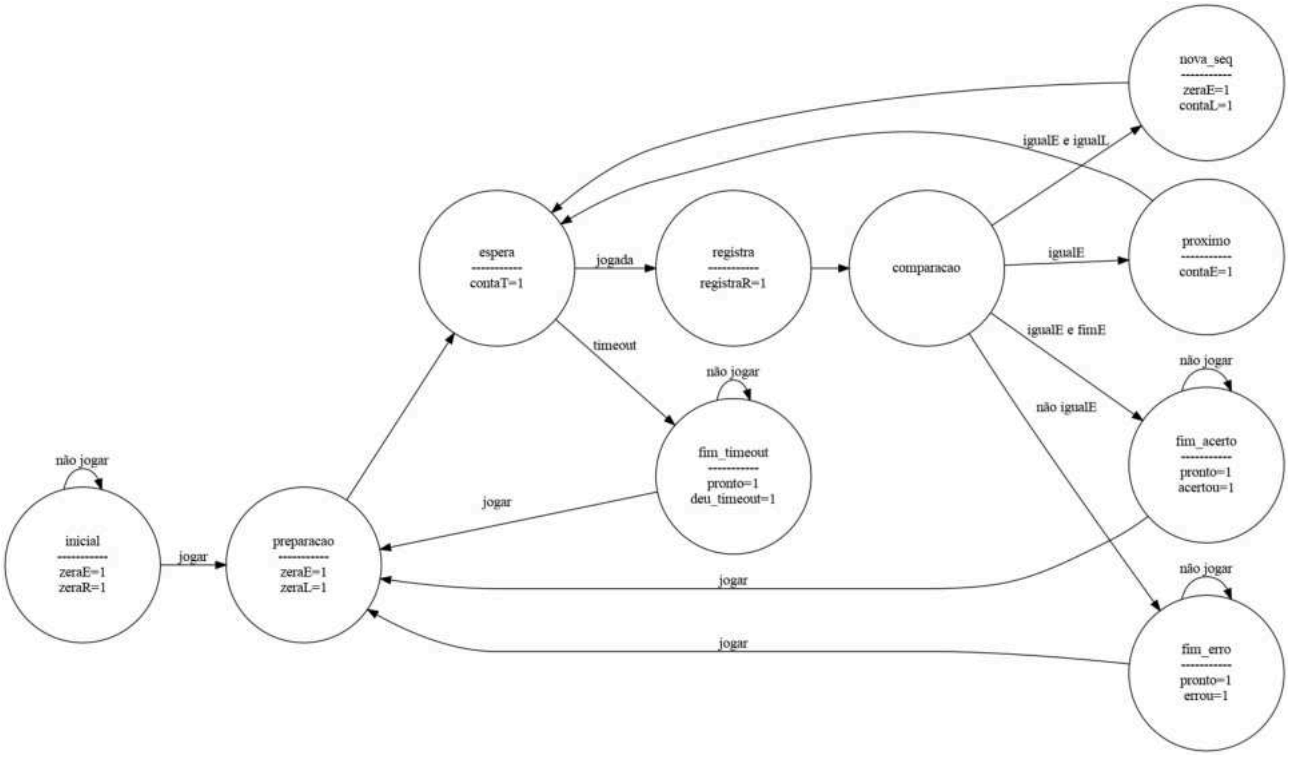


Figura 3 – Diagrama de Transição de Estados - UC

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. O sinal *db_contaL* monitora o sinal de *enable* do contador “contEnd”. 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.

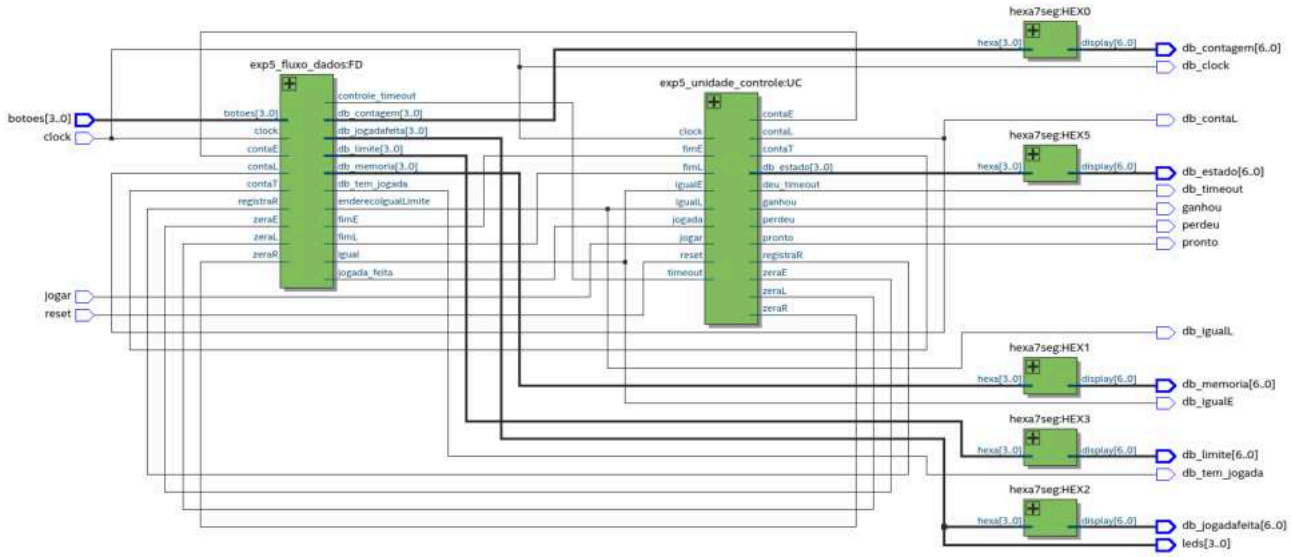


Figura 6 – Diagrama de Blocos do Circuito Completo

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 rodadas, utilizando a testbench “circuito_exp5_tb1.v” e o software *ModelSim* para simulação.

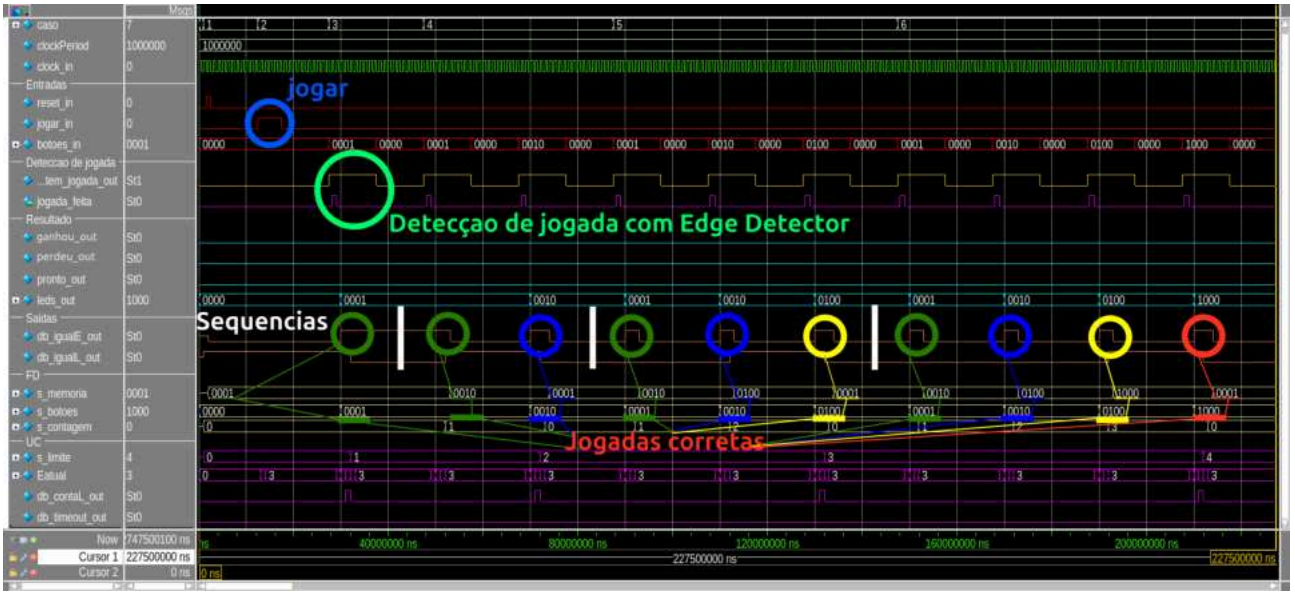
Tabela 2 – Descrição e Resultados Simulados do Cenário de Teste 1

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

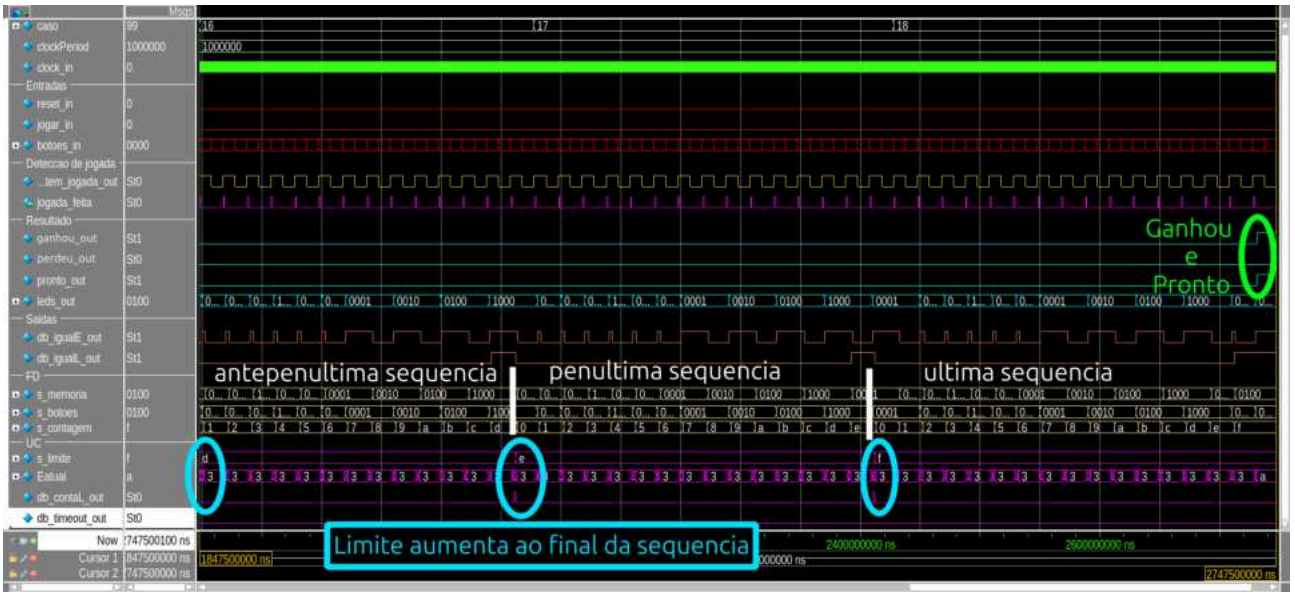
#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?
	segundos			
2	Acionar sinal jogar	acionar jogar	jogar=1	Sim
3	Jogar a primeira rodada	acionar botoes (0)	db_memoria=2, db_estado= ordem 4, 5, 6 e 2, db_jogada=1, db_limite=0, db_igualL=1	Sim
4	Jogar a segunda rodada	acionar botoes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

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

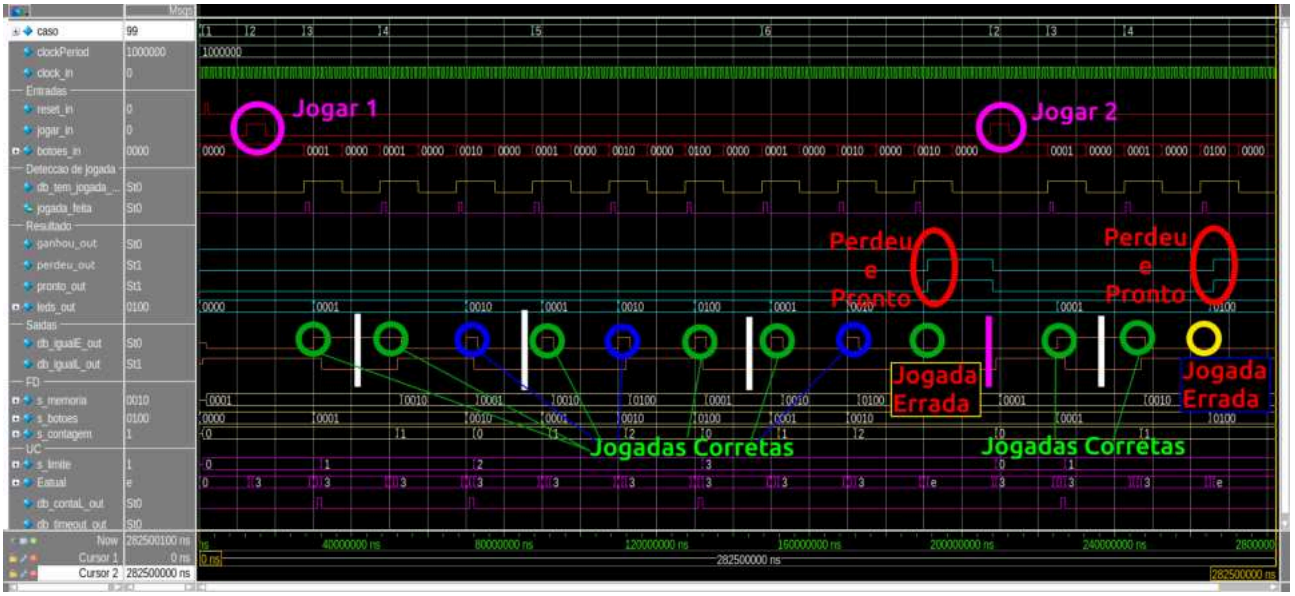
Validação do cenário de erro na 3º jogada da quarta rodada, reinício do sistema com a ativação de *jogar* e erro no 2º jogada da segunda rodada, utilizando a testbench “circuito_exp5_tb2.v” e o software *ModelSim* para simulação.

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	Jogar a primeira rodada	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	Jogar a segunda rodada	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	Jogar a terceira rodada	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	Jogar a quarta rodada e errar	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	Jogar a primeira rodada	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	Jogar a segunda rodada e errar	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

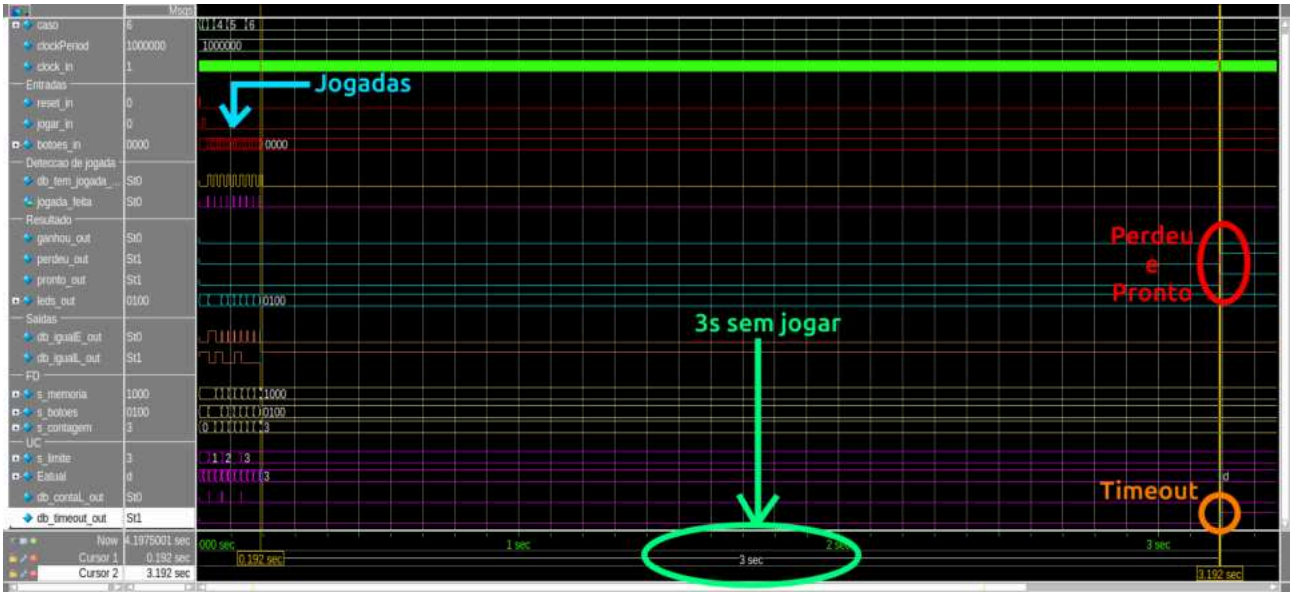
Validação do cenário de timeout da 3º jogada da quarta rodada, utilizando a testbench “circuito_exp5_tb3.v” e o software *ModelSim* para simulação.

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	Jogar a primeira rodada	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	Jogar a segunda rodada	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	Jogar a terceira rodada	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	Jogar a quarta rodada e aguardar timeout	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.

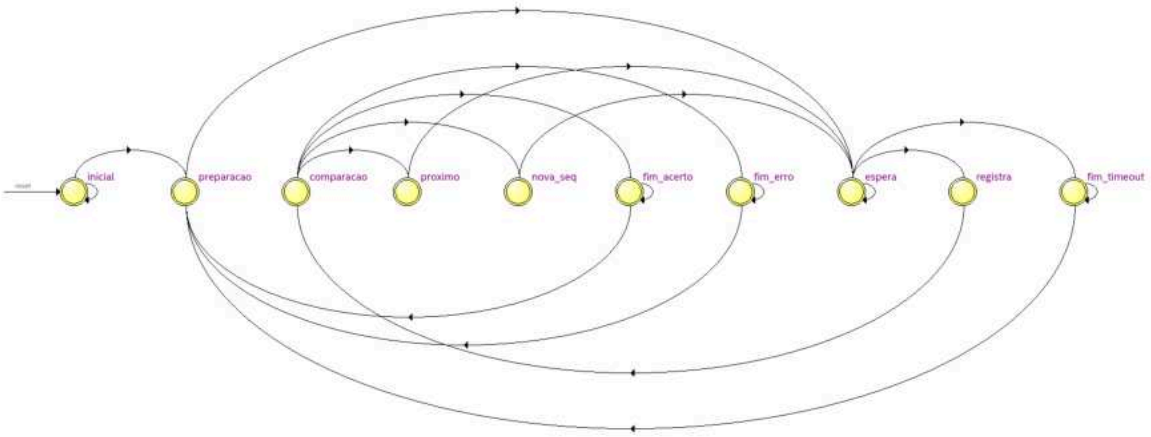


Figura 7 – Diagrama gerado pelo *StateMachineViewer* do Quartus

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.

3. 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

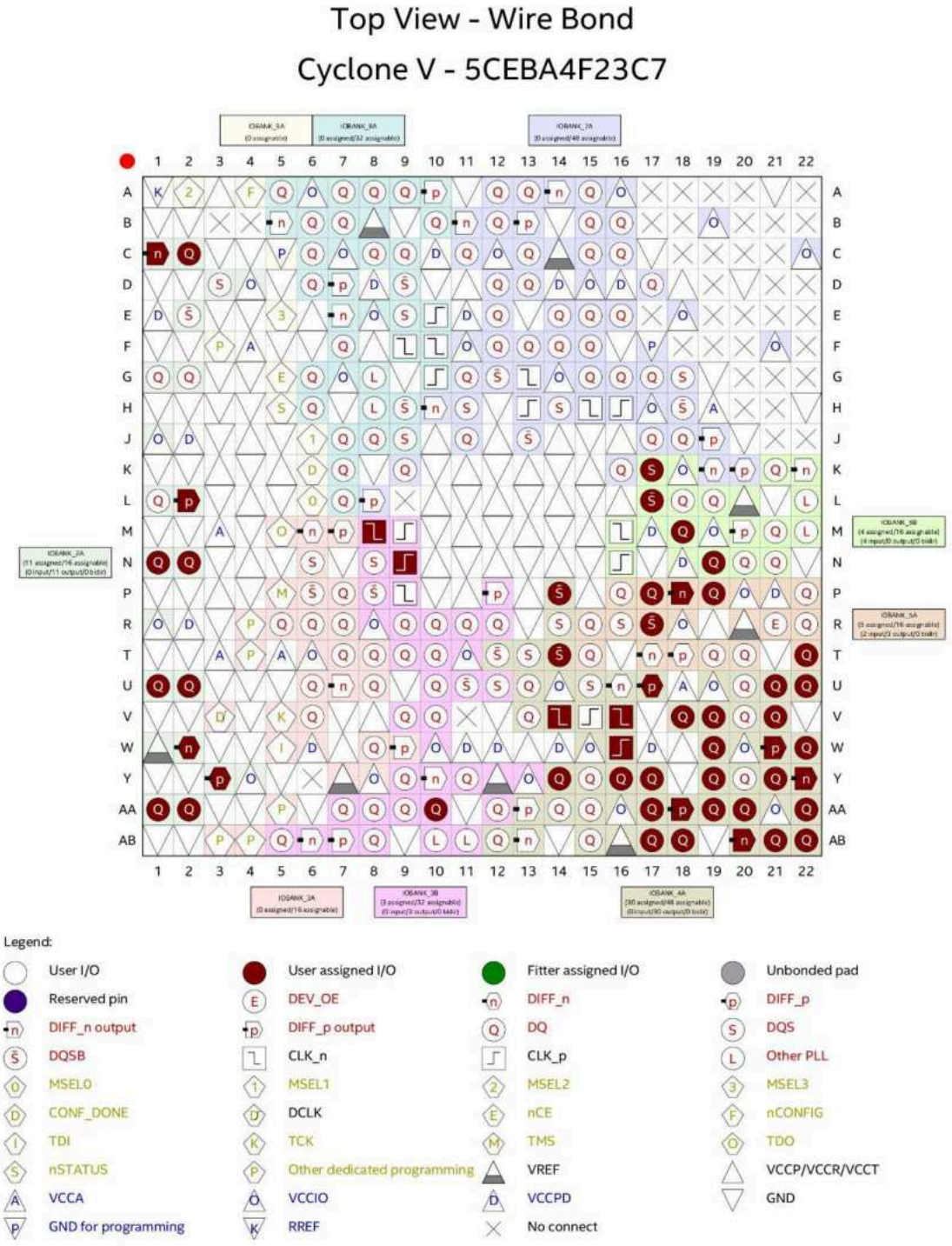


Figura 9 – Top-View da placa FPGA

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

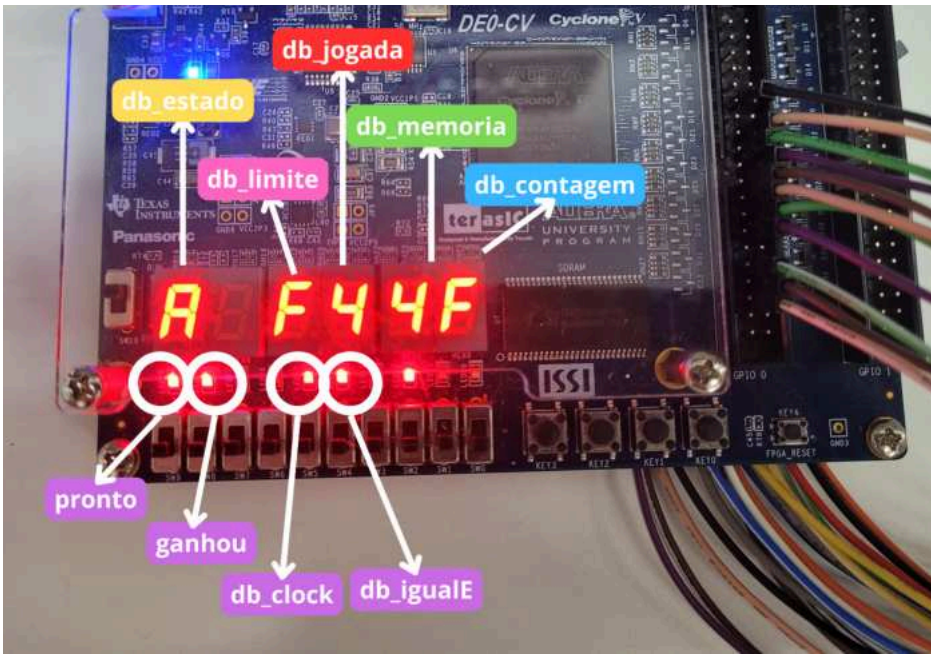
Validação do resultado de acerto das 16 rodada, utilizando o clock de 1KHz no circuito sintetizado na placa FPGA.

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	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) (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	acionar botoes (0) (1)	db_memoria=1, db_estado= ordem	Sim

#	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	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 (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	Sim
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

- Teste 18: Acionar décima sexta entrada



5.5 Execução Prática do Cenário de Teste 2 - Teste de Perder na FPGA

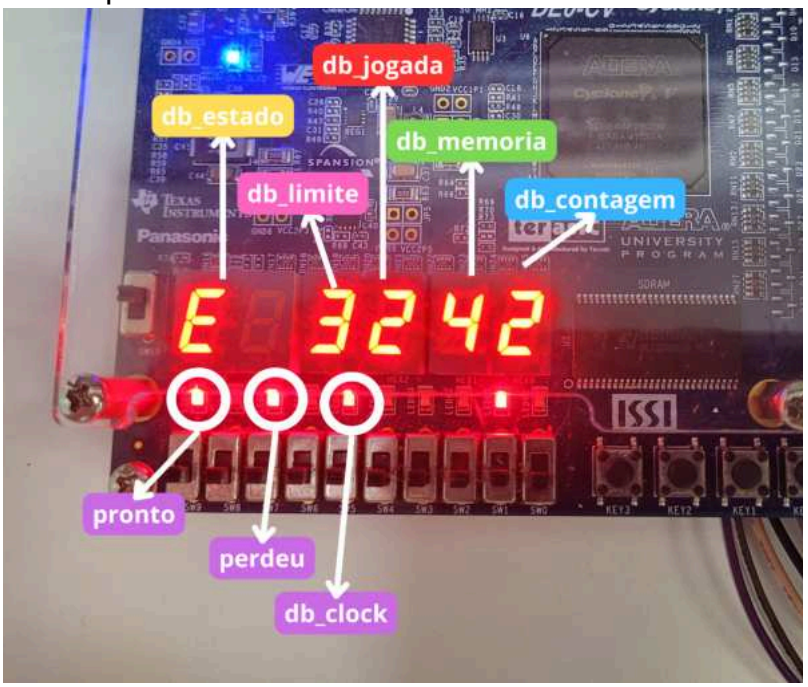
Teste na FPGA do cenário de erro na 3º jogada da quarta rodada, reinício do sistema com a ativação de *jogar* e erro na 2º jogada da segunda rodada.

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

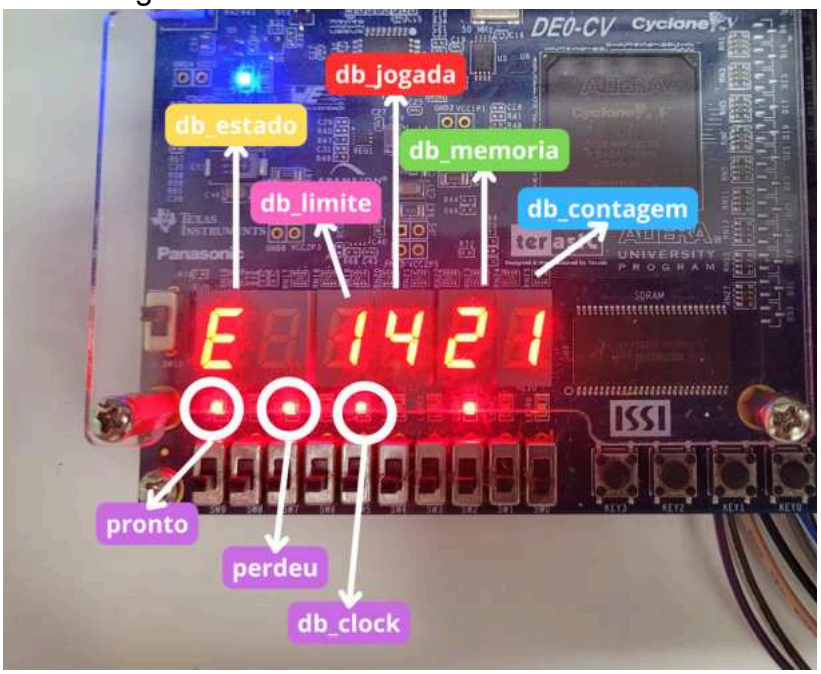
#	Operação	Sinais de Entrada	Resultado Esperado	Resultado prático ok?
c.i.	Condições iniciais	-	jogar=0, reset=0	Sim
1	“Resetar” circuito e aguardar alguns segundos	acionar reset	jogar=0, reset=1	Sim
2	Acionar sinal jogar	acionar jogar	jogar=1	Sim
3	Acionar primeira entrada (jogada 1)	acionar botoes (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

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

- Teste 6: Acionar quarta entrada e errar



- Teste 9: Acionar segunda entrada e errar



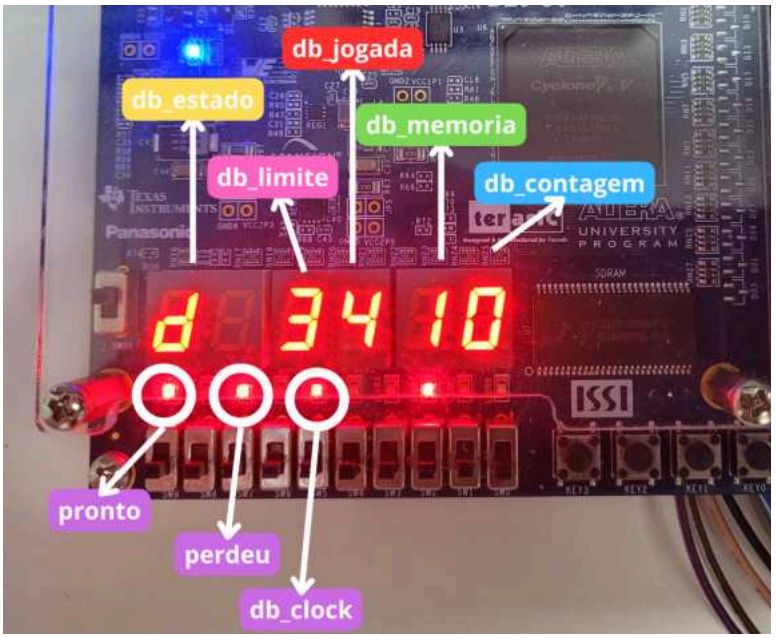
5.6 Execução Prática do Cenário de Teste 3 - Teste de Timeout na FPGA

Teste na FPGA do cenário de *Timeout* no 3º jogada da quarta rodada.

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	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) (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

- Teste 6: Acionar quarta entrada e esperar o *timeout*



6 Projeto do Desafio da Experiência

O desafio proposto para a experiência consistiu em projetar o timeout de cinco segundos para a jogada, dando continuidade ao sistema desenvolvido na experiência anterior e mantendo o planejamento estabelecido. Além disso, buscou-se habilitar os níveis de dificuldade da partida, um conceito já contemplado em experiências anteriores, mas ainda não implementado até este momento. Para este desafio, foram definidos dois modos de jogo: o *modo fácil* (nível 0), no qual a partida termina após as primeiras oito rodadas, e o *modo normal* (nível 1), onde o jogador deve acertar todas as jogadas até a décima sexta rodada.

6.1 Descrição do Desafio

O principal objetivo do desafio foi implementar os dois níveis de dificuldade e ajustar o tempo limite da jogada para cinco segundos. Durante o experimento, os alunos concentraram maior parte do tempo na implementação dos níveis de dificuldade, uma vez que o sistema de timeout já estava funcional com um tempo de três segundos, exigindo apenas um ajuste para cinco segundos.

Para a implementação dos níveis de dificuldade, foi proposta uma alteração no fluxo de dados, de modo que um sinal de condição fosse enviado para a unidade de controle de acordo com o nível de dificuldade escolhido pelo jogador (0 ou 1). Esse sinal determinaria o fim das comparações e encerraria o jogo no momento adequado. Além

disso, foi considerada uma solução escalável, que permitisse a inclusão de novos níveis de dificuldade com diferentes quantidades de rodadas, garantindo que a escolha do jogador fosse fixa e não pudesse ser alterada durante a partida.

6.2 Descrição do Projeto Lógico

Para a implementação do *timeout* desejado, foi necessária uma pequena mudança no fluxo de dados, na instanciação do componente “contador_m” e para a implementação dos níveis de dificuldade foram feitas mudanças tanto no FD quanto na UC. Essas alterações são abordadas com mais detalhes nas próximas seções.

6.3 Alterações do Fluxo de Dados

Para ajustar o timeout do planejamento para o valor testado em sala durante o desafio, foi modificada a instanciação do módulo “contador_m”. Anteriormente, os parâmetros M e N estavam configurados como 3000 e 12, respectivamente, permitindo a contagem de 3000 clocks a uma frequência de 1 kHz em um registrador de 12 bits, resultando em um limite de três segundos por jogada. Na nova configuração, M e N foram ajustados para 5000 e 13, permitindo a contagem de 5000 clocks, o que corresponde a um *timeout* de cinco segundos.

Além disso, para implementar as mudanças relacionadas ao modo de dificuldade, o componente “mux2x1” foi disponibilizado e utilizado pelos alunos para definir o valor do sinal que encerra a partida (sinal *fimE*). Inicialmente, esse sinal estava conectado ao *rco* do contador de endereços da memória, sendo enviado à UC para finalizar a jogada. Na nova implementação, o “mux2x1” foi configurado com um seletor de dificuldade, que alterna entre dois valores possíveis: o *rco* do contador, para nível de dificuldade igual a 0, e o resultado da operação *AND* entre os três bits menos significativos do endereço (*s_endereco*). Nesse segundo caso, a operação se encerra quando os três bits são iguais a 1, ou seja, quando o contador atinge o valor 7, marcando o fim da oitava rodada.

A implementação foi previamente verificada por meio do RTL Viewer, gerado pelo software Quartus, conforme ilustrado na Figura 10.

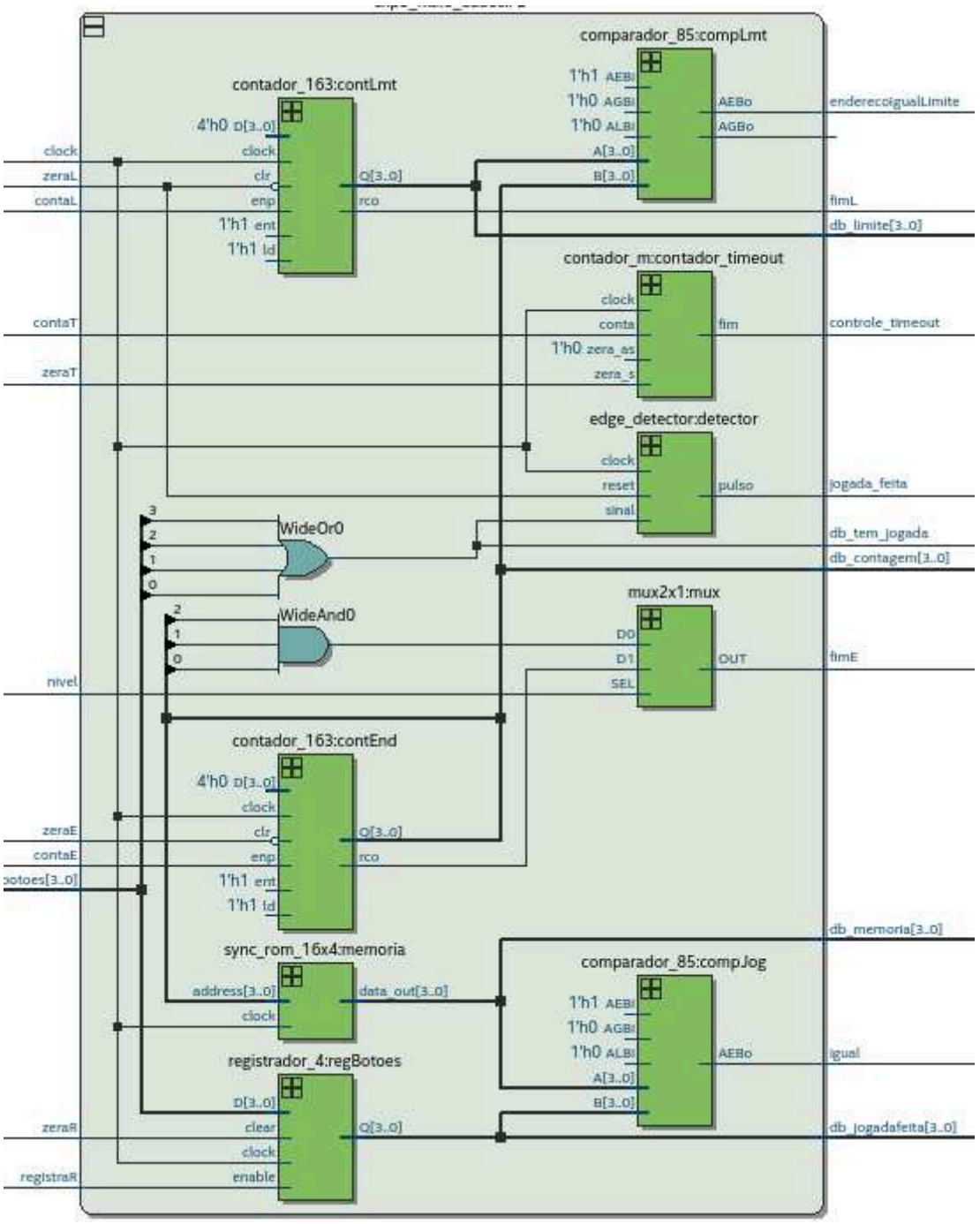


Figura 10 - RTL Viewer do Fluxo de Dados do Desafio

6.4 Alterações da Unidade de Controle

As alterações na unidade de controle foram realizadas para o desafio do modo de dificuldade do jogo. Para impedir que o jogador alterasse o nível durante a partida, permitindo a mudança apenas no início de um novo jogo, foi introduzido um sinal intermediário chamado *nivel_uc*, responsável por informar o nível de dificuldade ao FD. Esse novo sinal assume o valor do sinal de entrada *nivel*, definido pelo jogador no estado

de *preparação*. Dessa forma, *nivel_uc* é atualizado apenas no início de um novo jogo e permanece constante durante o restante da partida.

Além das alterações planejadas para o segundo desafio, foram necessárias modificações na unidade de controle relacionadas ao funcionamento do componente “contador_m”, responsável pelo timeout. Com a introdução do estado “nova_seq” na Experiência 5, a forma como o contador era inicializado tornou-se inválida. Anteriormente, o *reset* era ativado sempre que o contador de endereços era incrementado, ou seja, com a ativação do sinal *contaE*. No entanto, com a implementação das rodadas, ao atingir o estado de “nova_seq”, o contador de endereços era inicializado em vez de ser incrementado, e o sinal *contaE* permanecia desativado por duas jogadas consecutivas.

Essa falha foi identificada durante o teste prático na FPGA, quando os alunos, com o auxílio dos docentes presentes, perceberam que a perda do jogo devido ao *timeout* era acionado em um intervalo muito curto após uma jogada específica. A partir dessa observação, foram analisados os sinais de controle do contador, revelando a inconsistência. Para corrigir o problema, os alunos implementaram um novo sinal de controle, *zeraT*, que é ativado sempre que o sistema transita pelo estado “proximo”. Dessa forma, a contagem é corretamente reiniciada ao avançar para a próxima jogada.

Essas alterações podem ser verificadas diretamente na ferramenta *StateMachineViewer*, conforme ilustrado na Figura 11.

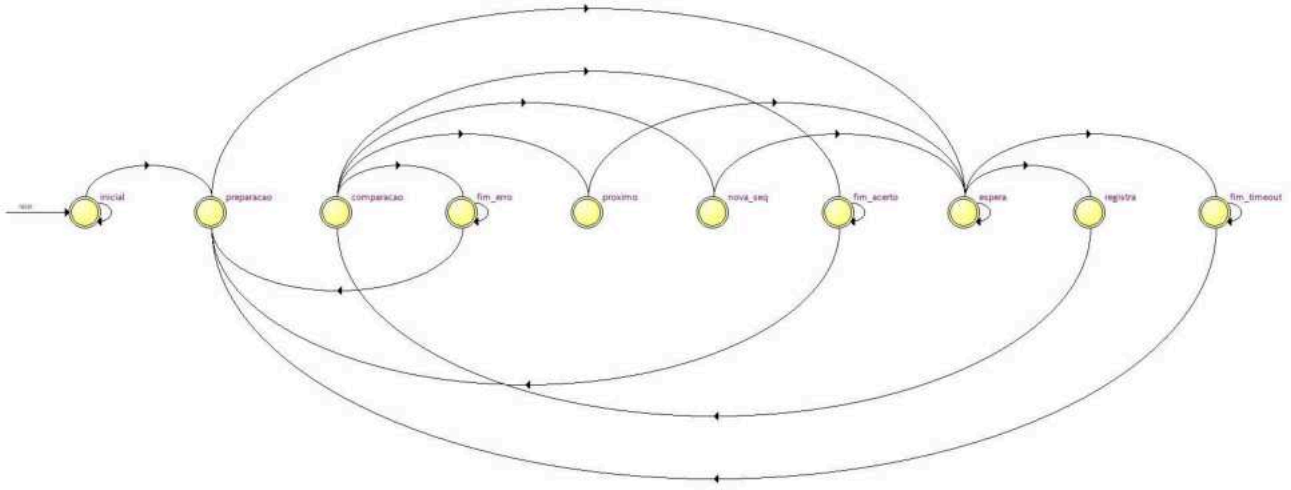


Figura 11 - State Machine Viewer do Desafio

6.5 Alterações do Sistema Digital

O circuito completo passou por poucas modificações com as novas adaptações. Foi necessário adicionar a entrada binária *nivel* e estabelecer a conexão entre a UC e o FD por meio do fio *s_nivel*. Além disso, para facilitar a depuração, foi introduzido o sinal *db_nivel*, permitindo verificar que o nível de dificuldade permanece inalterado durante o jogo, mesmo com a mudança das chaves. A Figura 12 apresenta o *RTL Viewer* do circuito completo, evidenciando essas alterações e proporcionando uma visão detalhada da implementação final.

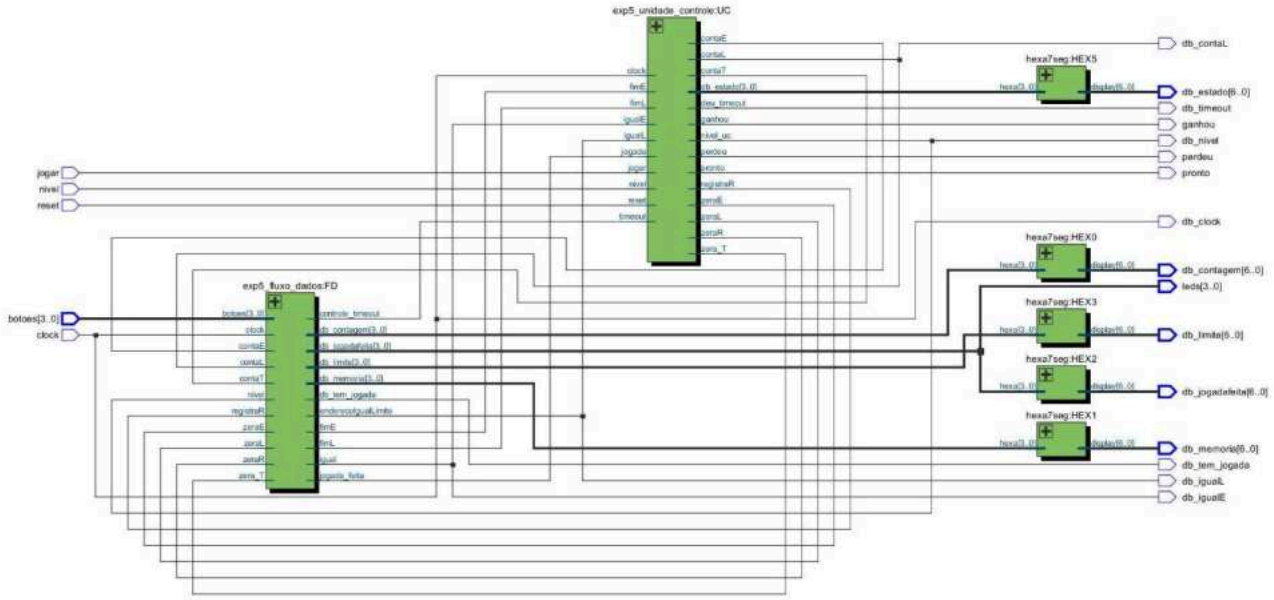


Figura 12 - RTL Viewer do Circuito Completo do Desafio

6.6 Verificação e Validação do Desafio

Foi feita a pinagem da placa FPGA, levando em conta o sinal de depuração novo *db_nivel* e a chave de entrada *nivel*. Elas são identificadas na Tabela 9 e na Figura 13.

Tabela 9 – Designação de pinos na FPGA do Desafio

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

Sinal	Pino na Placa DE0-CV	Pino no FPGA	Analog Discovery
CLOCK	GPIO_0_D13	PIN_T22	StaticIO – LED – DIO0 Patterns – Clock – 1KHz
DB_CLOCK	Led LEDR5	PIN_N1	-
DB_CONTAL	GPIO_0_D31	PIN_T20	StaticIO – LED – DIO8
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_ESTADO (0)	Display HEX50	PIN_N9	-
DB_ESTADO (1)	Display HEX51	PIN_M8	-
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	-

Sinal	Pino na Placa DE0-CV	Pino no FPGA	Analog Discovery
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_NIVEL	GPIO_0_D33	PIN_T18	
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	-
LEDS(2)	Led LEDR2	PIN_W2	-
LEDS(3)	Led LEDR3	PIN_Y3	-
NIVEL	Chave SW9	PIN_AB12	-
PERDEU	Led LEDR7	PIN_U1	-
PRONTO	Led LEDR9	PIN_L1	-
RESET	GPIO_0_D15	PIN_N19	StaticIO – Button 0/1 – DIO1

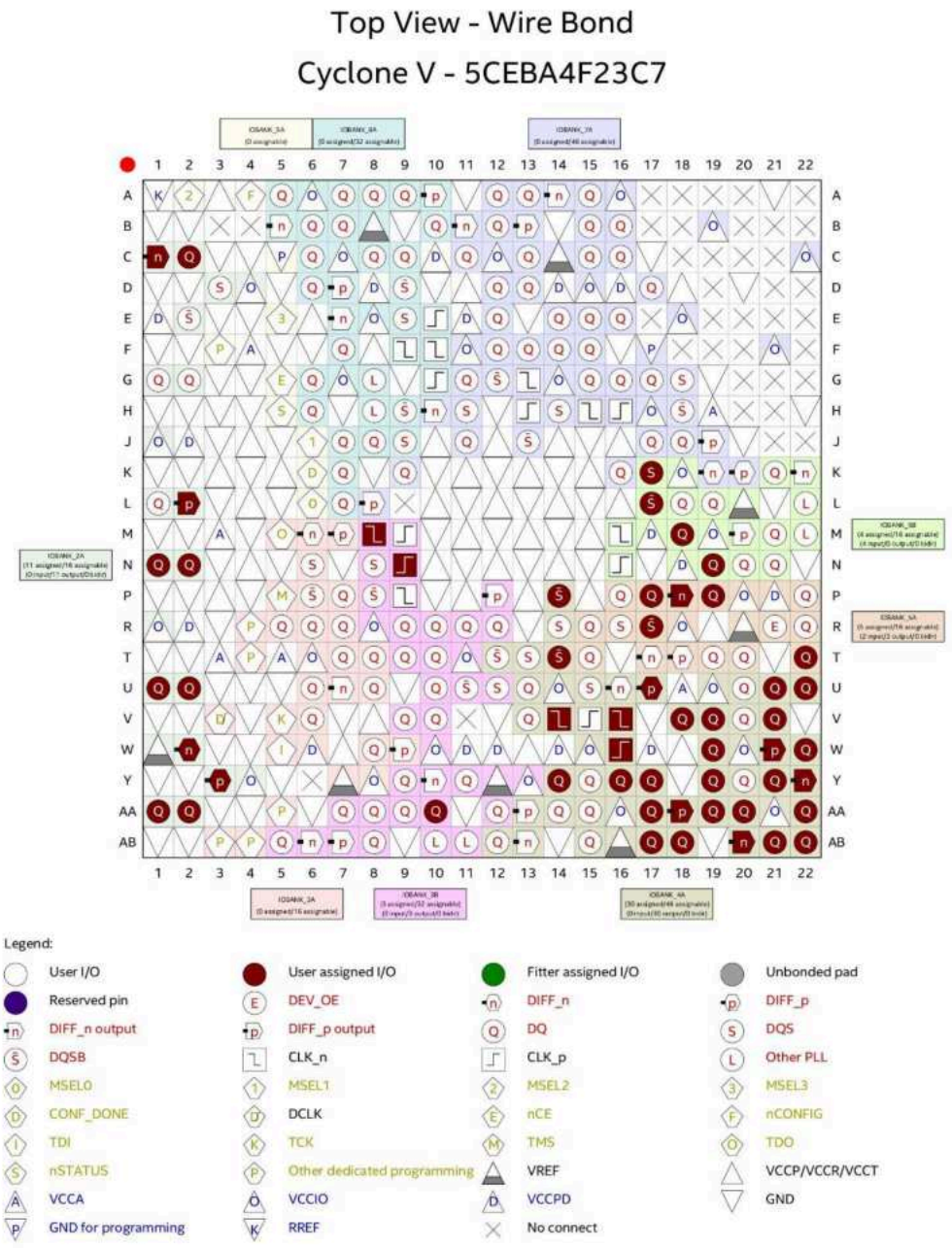


Figura 13 – Top View da FPGA para o Desafio

6.7 Cenário de Teste 1 – Vitória no nível difícil ou normal

Validação do resultado de acerto das 16 rodadas, no modo normal e mudando o nível no meio da partida, utilizando a testbench “circuito_exp5_tb1.v”, o software *ModelSim* para simulação e testes práticos na FPGA.

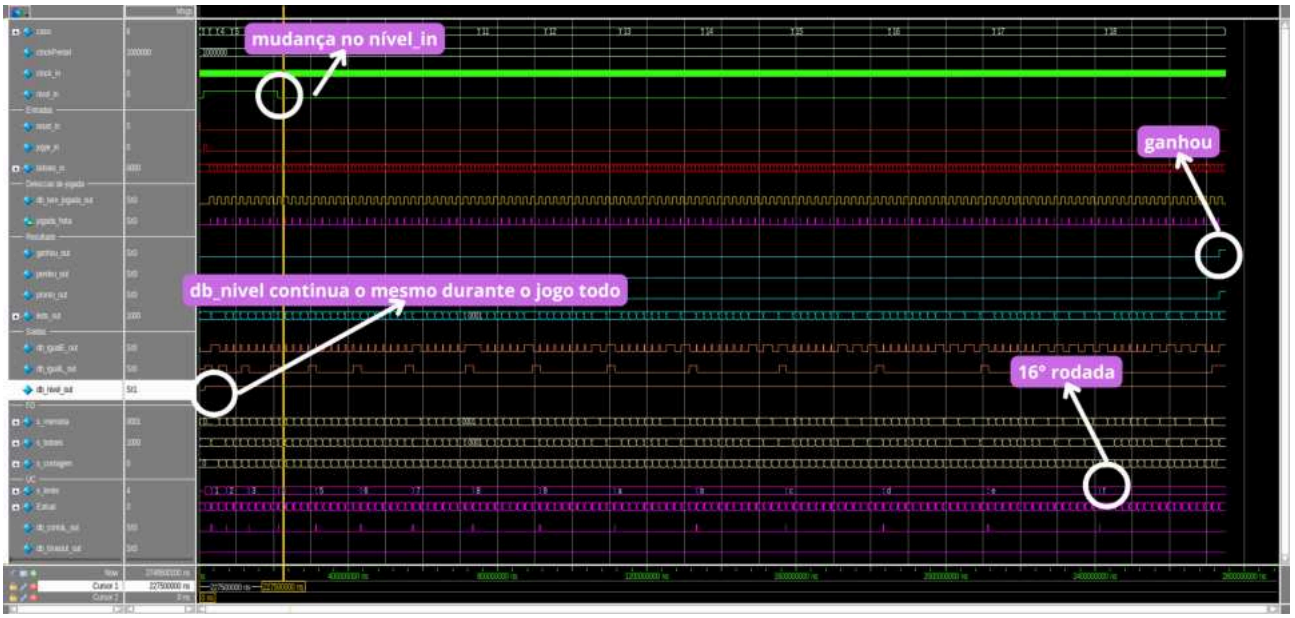
Tabela 10 – Descrição e Resultados Práticos e Simulados do Cenário de Teste 1

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?	Resultado prático ok?
c.i.	Condições	-	jogar=0, reset=0	Sim	Sim

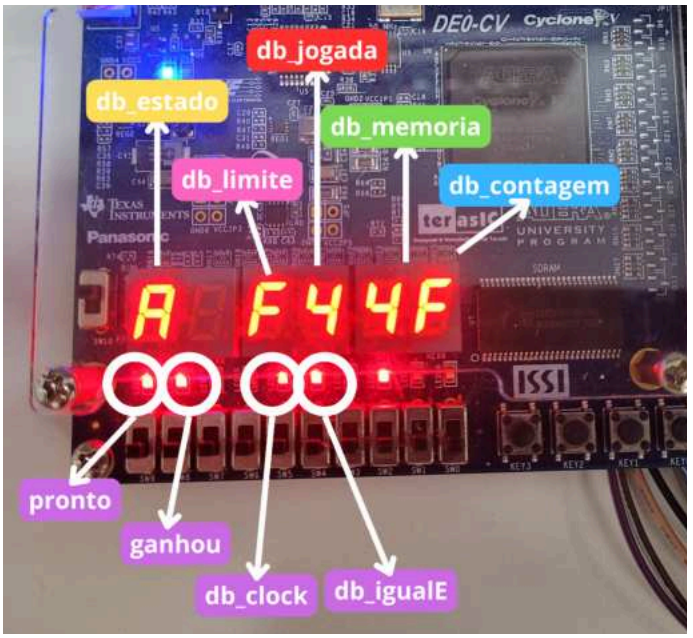
#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?	Resultado prático ok?
	iniciais				
1	“Resetar” circuito e aguardar alguns segundos	acionar reset	jogar=0, reset=1	Sim	Sim
2	Acionar sinal jogar	acionar jogar e db_nivel=1	jogar=1, db_nivel=1	Sim	Sim
3	Jogar a primeira rodada	acionar botoes (0)	db_memoria=2, db_estado= ordem 4, 5, 6 e 2, db_jogada=1, db_limite=0, db_igualL=1	Sim	Sim
4	Jogar a segunda rodada	acionar botoes (0) (1)	db_memoria=4, db_estado= ordem 4, 5, 6 e 2, db_jogada=2, db_limite=1,db_igualL=1	Sim	Sim
5	Jogar a terceira rodada	acionar botoes (0) (1) (2)	db_memoria=8, db_estado= ordem 4, 5, 6 e 2, db_jogada=4, db_limite=2, db_igualL=1	Sim	Sim
6	Jogar a quarta rodada	acionar botoes (0) (1) (2) (3) e nivel_in = 0	db_memoria=4, db_estado= ordem 4, 5, 6 e 2, db_jogada=8, db_limite=3, db_igualL=1, db_nivel=1	Sim	Sim
7	Jogar a quinta rodada	acionar botoes (0) (1) (2) (3) (2)	db_memoria=2, db_estado= ordem 4, 5, 6 e 2, db_jogada=4, db_limite=4, db_igualL=1	Sim	Sim
8	Jogar a sexta rodada	acionar botoes (0) (1) (2) (3) (2) (1)	db_memoria=1, db_estado= ordem 4, 5, 6 e 2, db_jogada=2, db_limite=5, db_igualL=1	Sim	Sim
9	Jogar a sétima rodada	acionar botoes (0) (1) (2) (3) (2) (1) (0)	db_memoria=1, db_estado= ordem 4, 5, 6 e 2, db_jogada=1, db_limite=6, db_igualL=1	Sim	Sim
10	Jogar a oitava rodada	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0)	db_memoria=2, db_estado= ordem 4, 5, 6 e 2, db_jogada=1, db_limite=7, db_igualL=1	Sim	Sim
11	Jogar a	acionar botoes (0) (1)	db_memoria=2, db_estado=	Sim	Sim

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?	Resultado prático ok?
	nona rodada	(2) (3) (2) (1) (0) (0) (1)	ordem 4, 5, 6 e 2, db_jogada=2, db_limite=8, db_igualL=1		
12	Jogar a décima rodada	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1)	db_memoria=4, db_estado= ordem 4, 5, 6 e 2, db_jogada=2, db_limite=9, db_igualL=1	Sim	Sim
13	Jogar a décima primeira rodada	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2)	db_memoria=4, db_estado= ordem 4, 5, 6 e 2, db_jogada=4, db_limite=10, db_igualL=1	Sim	Sim
14	Jogar a décima segunda rodada	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2)	db_memoria=8, db_estado= ordem 4, 5, 6 e 2, db_jogada=4, db_limite=11, db_igualL=1	Sim	Sim
15	Jogar a décima terceira rodada	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0) (1) (1) (2) (2) (3)	db_memoria=8, db_estado= ordem 4, 5, 6 e 2, db_jogada=8, db_limite=12, db_igualL=1	Sim	Sim
16	Jogar a décima quarta rodada	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 e 2, db_jogada=8, db_limite=13, db_igualL=1	Sim	Sim
17	Jogar a décima quinta rodada	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 e 2, db_jogada=1, db_limite=14, db_igualL=1	Sim	Sim
18	Jogar a décima sexta rodada	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	Sim

- Simulação no ModelSim



- Teste 18 - Acertar décima sexta jogada, modo difícil:



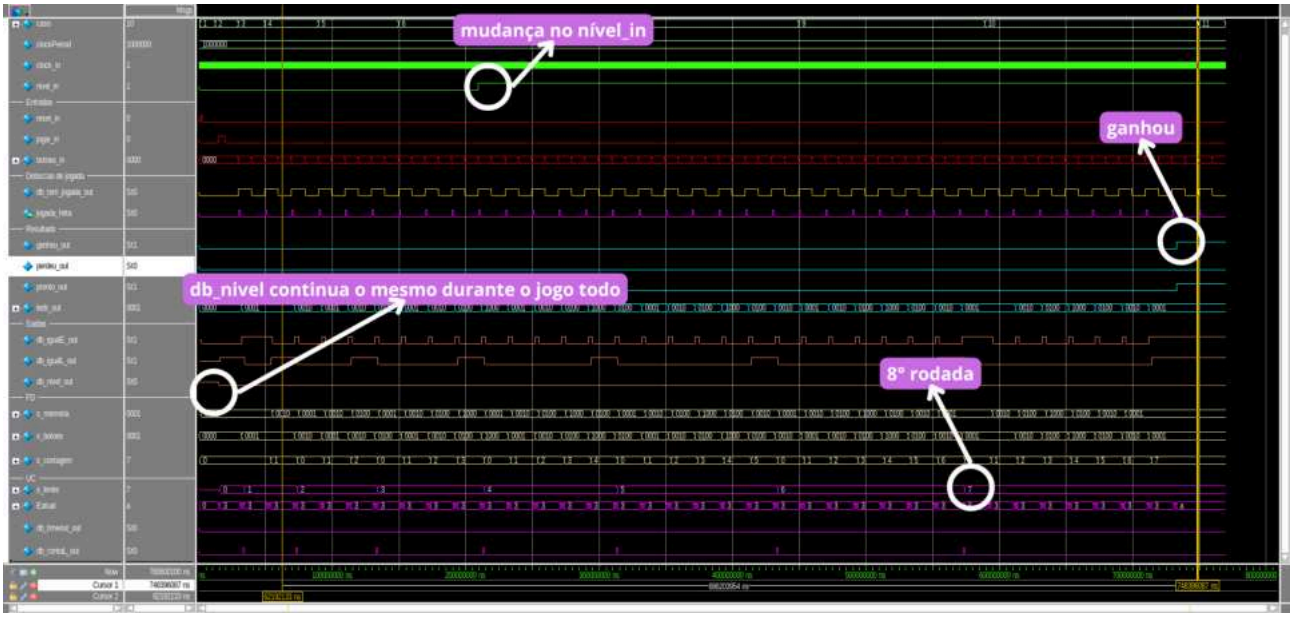
6.8 Cenário de Teste 2 – Vitória no nível fácil

Validação do resultado de acerto de 8 rodadas, no modo fácil e mudando o nível no meio da partida, utilizando a testbench “circuito_exp5_tb1.v”, o software *ModelSim* para simulação e testes práticos na FPGA.

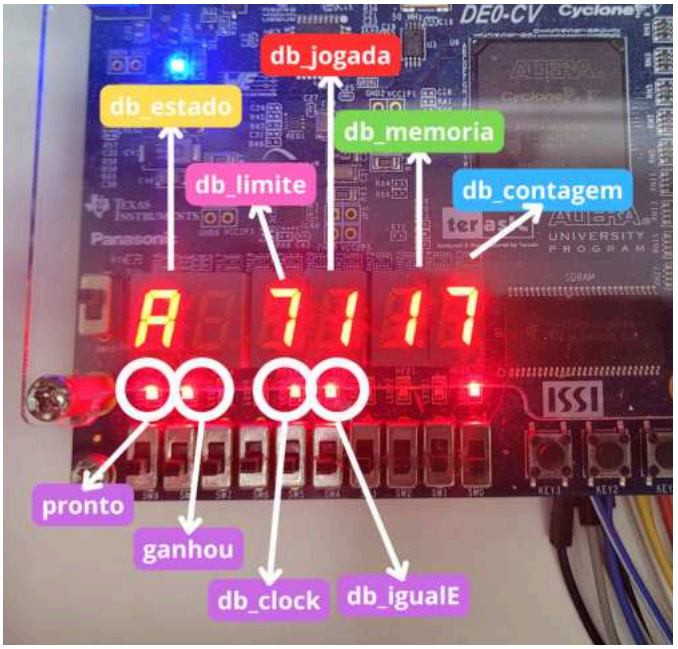
Tabela 11 – Descrição e Resultados Práticos e Simulados do Cenário de Teste 2

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?	Resultado prático ok?
c.i.	Condições iniciais	-	jogar=0, reset=0	Sim	Sim
1	“Resetar” circuito e aguardar alguns segundos	acionar reset	jogar=0, reset=1	Sim	Sim
2	Acionar sinal jogar	acionar jogar e nivel_in=0	jogar=1, db_nivel=0	Sim	Sim
3	Jogar a primeira rodada	acionar botoes (0)	db_memoria=2, db_estado= ordem 4, 5, 6 e 2, db_jogada=1, db_limite=0, db_igualL=1	Sim	Sim
4	Jogar a segunda rodada	acionar botoes (0) (1)	db_memoria=4, db_estado= ordem 4, 5, 6 e 2, db_jogada=2, db_limite=1,db_igualL=1	Sim	Sim
5	Jogar a terceira rodada	acionar botoes (0) (1) (2)	db_memoria=8, db_estado= ordem 4, 5, 6 e 2, db_jogada=4, db_limite=2, db_igualL=1	Sim	Sim
6	Jogar a quarta rodada	acionar botoes (0) (1) (2) (3), nivel_in=1	db_memoria=4, db_estado= ordem 4, 5, 6 e 2, db_jogada=8, db_limite=3, db_igualL=1, db_nivel=0	Sim	Sim
7	Jogar a quinta rodada	acionar botoes (0) (1) (2) (3) (2)	db_memoria=2, db_estado= ordem 4, 5, 6 e 2, db_jogada=4, db_limite=4, db_igualL=1	Sim	Sim
8	Jogar a sexta rodada	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	Sim
9	Jogar a sétima rodada	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	Sim
10	Jogar a oitava rodada	acionar botoes (0) (1) (2) (3) (2) (1) (0) (0)	db_memoria=2, db_estado= ordem 4, 5 e A, db_jogada=4, db_limite=7, pronto=1, ganhou=1	Sim	Sim

- Simulação no ModelSim



- Teste 10 - Oitava rodada



6.9 Cenário de teste 3 – Timeout no nível difícil

Simulação de *timeout* da primeira jogada da terceira rodada, após inserir a segunda jogada da segunda rodada com sucesso, com a testbench “circuito_exp5_tb1.v”, o software *ModelSim* para simulação e testes práticos na FPGA.

Tabela 12 – Descrição e Resultados Práticos e Simulados do Cenário de Teste 3

#	Operação	Sinais de Entrada	Resultado Esperado	Resultado simulado ok?	Resultado prático ok?
c.i.	Condições iniciais	-	jogar=0, reset=0	Sim	Sim
1	“Resetar” circuito e aguardar alguns segundos	acionar reset	jogar=0, reset=1	Sim	Sim
2	Acionar sinal jogar	acionar jogar e nível	jogar=1, nivel=1	Sim	Sim
3	Jogar a primeira rodada	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	Sim
4	Jogar a segunda rodada e esperar timeout	acionar botoes (0) (1) e aguardar	db_memoria=4, db_estado= ordem 4, 5, 6, 7 e 2, db_jogada=2, db_limite=1, perdeu=1, db_timeout=1, pronto=1	Sim	Sim

- Simulação no ModelSim

Como o desafio de *timeout* já tinha sido implementado, o docente responsável aconselhou o uso já do tempo de cinco segundos logo no primeiro teste em sala. Desse modo, a imagem da seção 5.6 já evidencia o sucesso do *timeout* feito em sala.

7 Conclusões

A implementação prática do experimento permitiu validar a funcionalidade do jogo de sequências de jogadas na FPGA DE0-CV, confirmando a operação dos estados de controle e a integração com o fluxo de dados. Os testes realizados em bancada demonstraram que o mecanismo de repetição progressiva da sequência foi corretamente implementado, garantindo que o jogador precisasse memorizar e repetir todas as jogadas anteriores antes de inserir a próxima.

Apesar de alguns erros durante a montagem e os testes iniciais, as estratégias de depuração adotadas pelo grupo foram eficazes na identificação e correção das inconsistências. O monitoramento dos sinais de depuração, tanto nos LEDs da FPGA quanto no Analog Discovery, possibilitou diagnosticar falhas de transição de estado e garantir que o comportamento da unidade de controle estivesse conforme o esperado. Além disso, ajustes na configuração da máquina de estados foram necessários para solucionar problemas na detecção de timeout, garantindo que a contagem do tempo fosse reiniciada corretamente após cada jogada.

Outro ponto importante foi a execução do desafio proposto, que envolveu a implementação de diferentes níveis de dificuldade e a expansão do timeout para cinco segundos. A adição do seletor de nível permitiu testar o sistema com uma versão mais curta do jogo, encerrando após oito rodadas no modo fácil e mantendo as dezesseis rodadas no modo normal. O grupo enfrentou desafios na configuração do mux2x1 para alternar corretamente entre os limites de cada nível, mas, após ajustes no fluxo de dados, o circuito operou conforme planejado.

Os resultados finais indicaram que o circuito foi capaz de processar corretamente todas as jogadas, detectar erros do jogador e encerrar o jogo quando necessário. As simulações no ModelSim foram compatíveis com os testes físicos na FPGA, validando a fidelidade do projeto. Com isso, a experiência foi concluída com sucesso, assegurando que os objetivos definidos foram atingidos e preparando o grupo para desafios mais avançados em projetos digitais.