Escola Politécnica da USP

Departamento de Engenharia de Computação e Sistemas Digitais

PCS3635 – Laboratório Digital 1

Turma 3 – Prof. Reginaldo Arakaki

Planejamento da Semana 1

Desenvolvimento de Projeto de Circuitos Digitais em FPGA

Lucas Parra Sgarbosa

Henrique de Andrade Assme

Bancada: A-01

Data: 13/03/2023

1. Escolha do tema, requisitos e cronograma

Para a escolha do tema do projeto da disciplina, a dupla pensou em algo que não fosse muito complicado, mas que ainda fosse desafiador o suficiente.

Escolhemos fazer algumas modificações no circuito que já existe:

- Sistema de vida: quando o jogador perder, seja por timeout ou por jogar errado, ele terá a opção de continuar jogando de onde parou e perdeu uma vida;
- Sistema de níveis: para tornar o jogo mais desafiador, o sistema de níveis conta com fácil, médio e difícil. O que muda de um nível para outro são

a quantidade de rodadas (4, 10 e 16) e quantidade de vidas disponíveis (4, 2 e 1);

- Efeitos estéticos e mais humanos ao jogo, como adicionar alguma maneira de ver quanto tempo falta para o jogador perder por tempo, escrever nos 7 segmentos as palavras ganhou, perdeu e erro, além de uma adição ou de um buzzer ou de leds coloridos para indicar esses cenários de resultado;

Além das modificações no jogo já existente, adicionaremos um novo modo de jogo. A ideia é que o jogador acione as jogadas na sequência que elas são mostradas (uma jogada, depois duas seguidas, depois 3, etc). Porém, a dificuldade desse modo se encontra no tempo, a cada nova rodada o tempo que o jogador tem para fazer a sequência correta é menor até que o menor tempo é atingido na última rodada do jogo.

Os requisitos pensados foram os seguintes:

Código: 0	■ Funcional □ Não Funcional
Requisito: Sistema de vida	
Descrição: Implementação de uma maneira do jogador continuar jogando mesmo que tenha perdido o jogo através de um sistema de vidas. Caso as vidas tenham acabado, o jogador perde de fato o jogo. Se não, perde uma vida e continua jogando do início da rodada perdida. A vida aparecerá em um display hexadecimal.	
Prioridade: ■ Alta □ Média □ Baixa	
Estabilidade: ■ Alta □ Média □ Baixa	
Requisitos associados:	

Código: 1	■ Funcional □ Não Funcional
Requisito: Sistema de nível	

Descrição: Implementação de um sistema de níveis no qual o jogador escolhe a dificuldade do jogo antes de iniciá-lo e a partir dela se definem o intervalo de tempo máximo pra cada jogada e o número de jogadas corretas a serem feitas para vencer o jogo.
Prioridade: ■ Alta □ Média □ Baixa
Estabilidade: ■ Alta □ Média □ Baixa
Requisitos associados: sistema de vida

Código: 2	■ Funcional □ Não Funcional
Requisito: Display hexadecimal para o timer da rodada	
Descrição: adicionar no display hexadecimal quanto tempo falta para terminar uma jogada, para que o jogador consiga ver na placa o tempo decaindo de 5 a 0 segundos.	
Prioridade: □ Alta □ Média ■ Baixa	
Estabilidade: □ Alta □ Média ■ Baixa	
Requisitos associados:	

Código: 3	□ Funcional ■ Não Funcional
Requisito: Memórias para modo novo	
Descrição: no novo modo do genius, a memória não possui escrita, logo, será necessário adicionar uma memória com os dados pré colocados para o jogador colocar a sequência correta. Além disso, uma memória guardará o tempo respectivo de cada rodada, uma vez que nesse novo modo o tempo será diferente para cada uma delas (diminuindo conforme o jogo se aproxima do final).	
Prioridade: ■ Alta □ Média □ Baixa	
Estabilidade: ■ Alta □ Média □ Baixa	
Requisitos associados:	

Código: 4	■ Funcional □ Não Funcional

Requisito: Chave para escolha entre os dois modos e chave para escolha de nível	
Descrição: uma das chaves da placa será usada para escolher qual modo de jogo será desejado e chaves também para escolher qual nível escolhido (fácil, médio ou difícil)	
Prioridade:	■ Alta □ Média □ Baixa
Estabilidade:	■ Alta □ Média □ Baixa
Requisitos associados:	

Código: 5	■ Funcional □ Não Funcional
Requisito: som/leds para indicar que o jogador errou (perde vida), ganhou e perdeu	
Descrição: pensamos em adicionar um sinal sonoro/visual para indicar quando o jogador perde uma vida, perde ou ganha o jogo.	
Prioridade:	■ Alta □ Média □ Baixa
Estabilidade:	■ Alta □ Média □ Baixa
Requisitos associados:	

As atividades necessárias foram separadas nas 4 semanas de desenvolvimento da seguinte forma:

Semana	Descrição
1	Implementação do sistema de vida, aparecer as mensagens no display 7 segmentos e começar a planejar a implementação do sistema de níveis
2	Implementação completa do sistema de níveis
3	Começo da implementação do novo modo de jogo e do sistema para escolher qual modo de jogo será usado

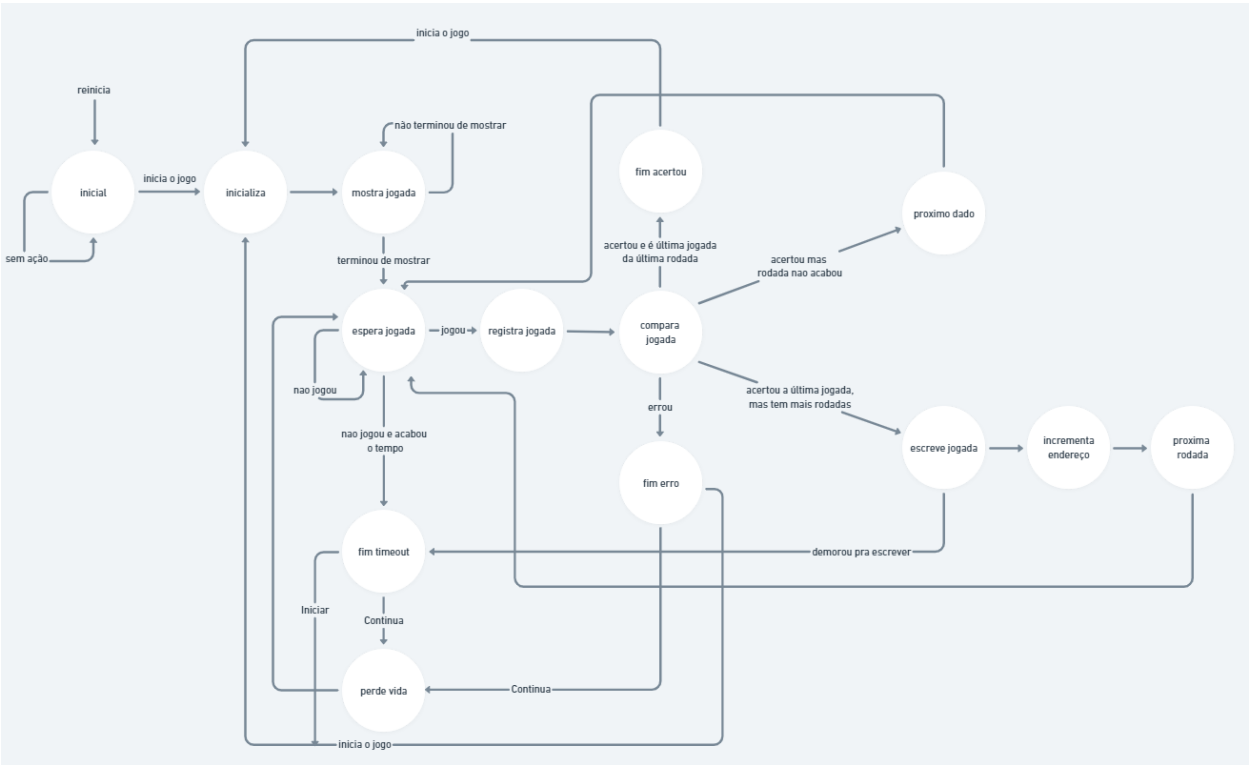
4	Implementação completa do modo de jogo e última bateria de testes no circuito antes da conclusão do projeto

2. Semana 1

Para desenvolver o sistema de vida foi necessário adicionar algumas modificações no DF, na UC e no código principal do jogo.

Na UC foram adicionados os sinais continuar (in), vidaZerada (in) e perderVida (out). O sinal continuar serve para que o jogador, ao perder, possa continuar a jogar do ponto em que perdeu desde que tenha uma vida para perder, aí entra o sinal vidaZerada. Além disso, cada vez que perde uma vida, o sinal perderVida é ativado e enviado para o DF.

Além disso, foi necessário adicionar o estado perde vida para fazer o sistema de vida. Assim ficou o novo diagrama da unidade de controle:



No DF foram adicionados os sinais perderVida (in), vizaZerada (out) e vidas (out). Esses sinais são interligados na UC e no circuito principal do jogo para realizar a lógida de vida. A lógica de perder vidas foi feita da seguinte maneira:

```
    process(perderVida, zeraCR) is
      begin
        if zeraCR='1' then
          s_vidas <= "0101";
          vidaZerada <= '0';
        elsif(perderVida = '1' and s_vidas /= "0000") then
          s_vidas <= std_logic_vector(s_vidas(3 downto 0) - "0001");
        elsif s_vidas="0001" then
          vidaZerada <= '1';
        end if;
        vidas(3 downto 0) <= s_vidas(3 downto 0);
    end process;
end architecture estrutural;
```

Inicialmente o sinal auxiliar s_vidas recebe 5, porém, depois de adicionar o sistema de dificuldade para o jogo, o valor inserido será o respectivo de cada dificuldade.

No circuito principal foram adicionados os sinais continuar (in) e db_vidas (out) e as modificações na UC e no DF foram adicionadas também, bem como suas ligações.

Para a semana 1 também foram adicionados no display de 7 segmentos as palavras “ganhou” e “perdeu” quando o circuito estava nos estados de fimA e fimE.

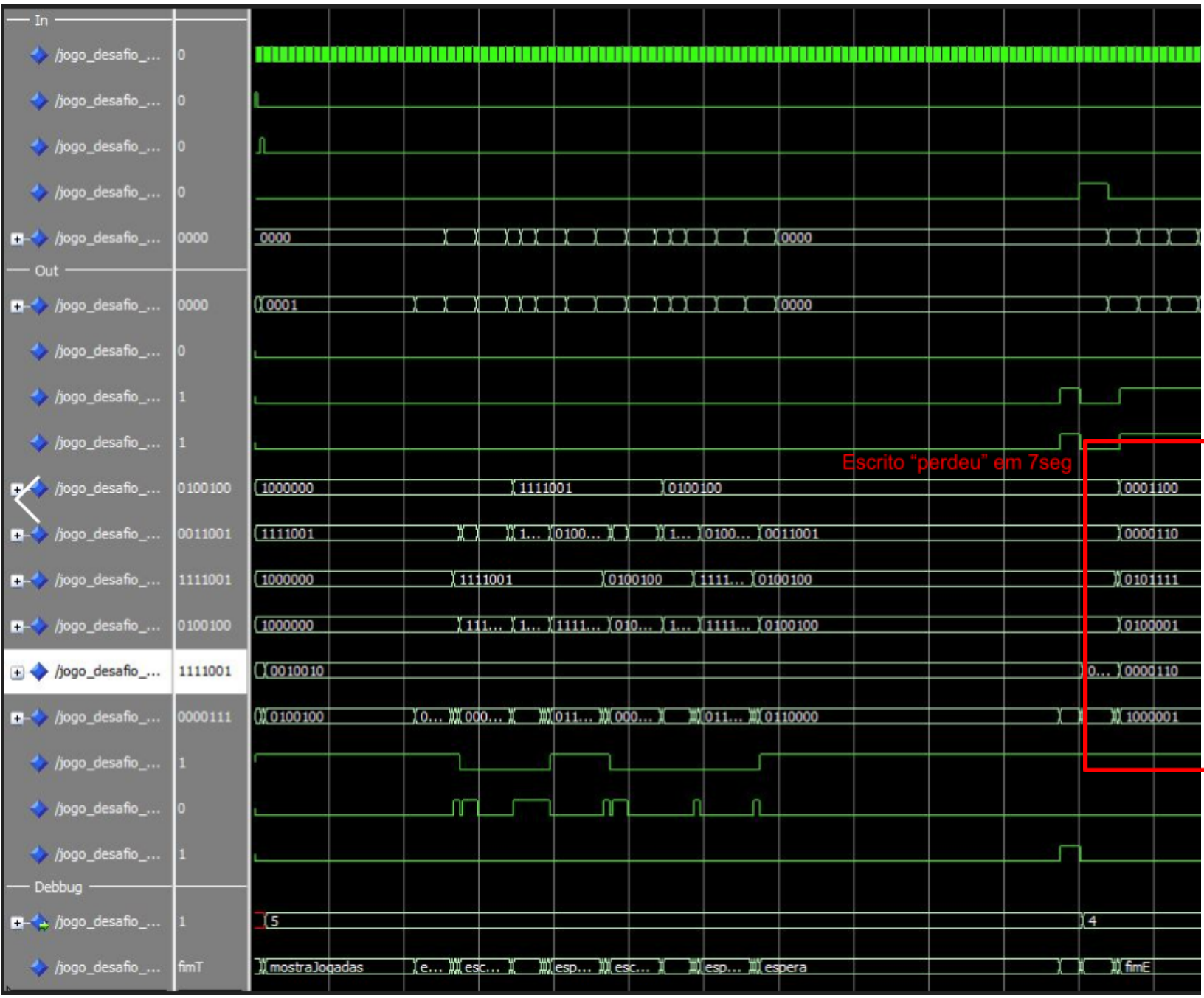
Para isso, mudanças foram feitas no hexa7seg, mudando o tamanho do sinal de entrada de 4 bits para 5 bits, o que permitiu que mais 16 opções fossem possíveis no display de sete segmentos. Para permitir a diferenciação entre as letras ou os sinais de debbug, um sinal auxiliar tipo foi criado no circuito do jogo principal que permitia diferenciar quando o sistema se encontra nos estados de fimA e fimE.

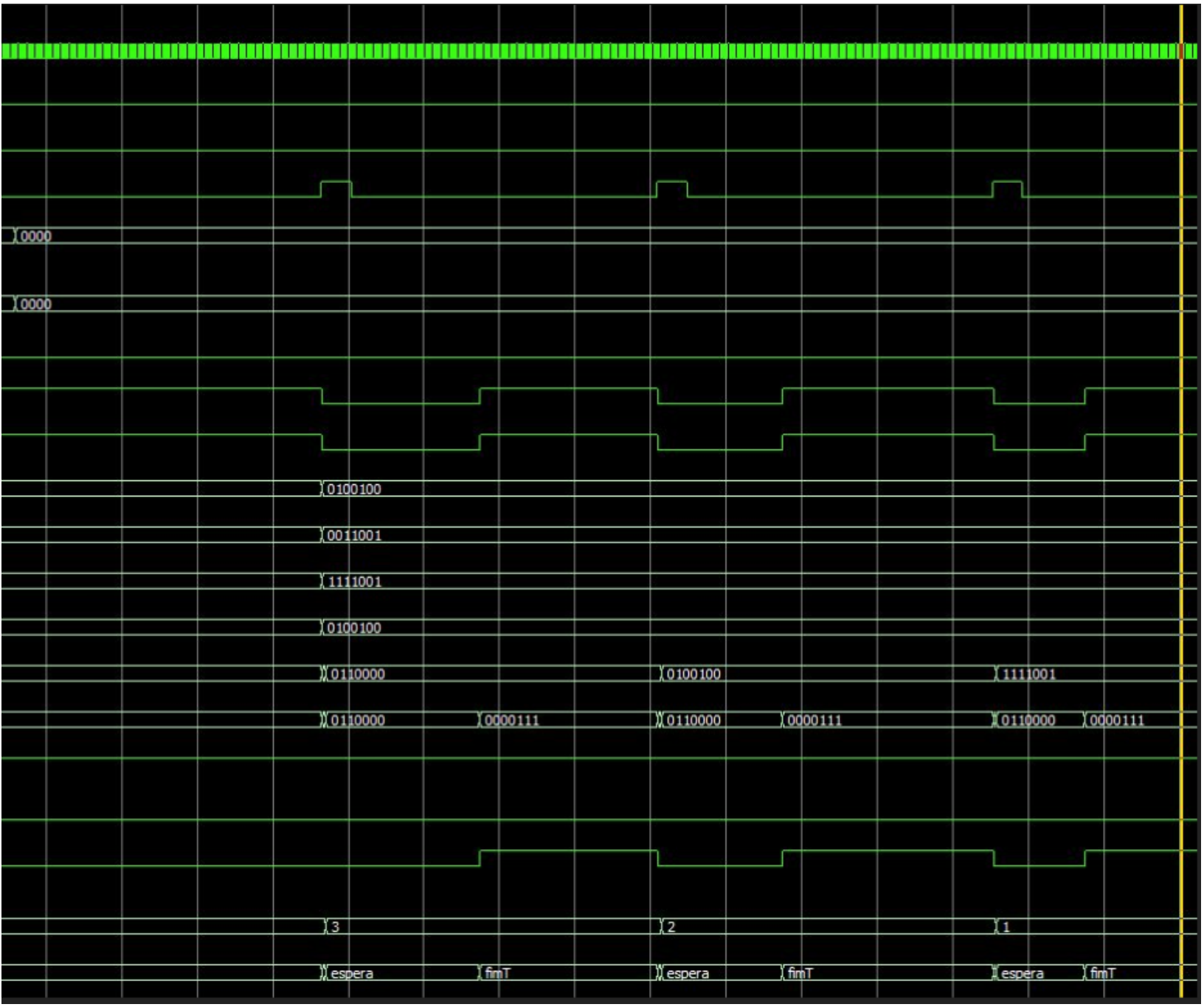
```
tipo <= "00" when uc_acertou='1' and uc_df_errou='0' else
        "01" when uc_acertou='0' and uc_df_errou='1' and uc_timeout='0' else
        "10" when uc_acertou='0' and uc_df_errou='1' and uc_timeout='1' else
        "11";

with tipo select
    entradaHEX0 <= "10000" when "00", --G
                   "10101" when "01", --P
                   "0" & df_hex_contagem when others;
with tipo select
    entradaHEX1 <= "01010" when "00", --A
                   "01110" when "01", --E
                   "0" & df_hex_memoria when others;
with tipo select
    entradaHEX2 <= "10001" when "00", --N
                   "10110" when "01", --R
                   "0" & df_hex_jogada when others;
with tipo select
    entradaHEX3 <= "10010" when "00", --H
                   "01101" when "01", --D
                   "0" & df_hex_rodada when others;
with tipo select
    entradaHEX4 <= "10011" when "00", --O
                   "01110" when "01", --E
                   "0" & df_hex_vidas when others;
with tipo select
    entradaHEX5 <= "10100" when "00", --U
                   "10100" when "01", --U
                   "11001" when "10", --t
                   "0" & uc_hex_estado when others;

db_contagem <= saidaHEX0;
db_memoria <= saidaHEX1;
db_jogadafeita <= saidaHEX2;
db_rodada <= saidaHEX3;
db_vidas <= saidaHEX4;
db_estado <= saidaHEX5;
```

a. Testes das novas funcionalidades

Para os testes, foram adaptados os testbenchs já existentes para testar o escrito nos displays e um testbench foi criado para verificar se o sistema de vidas está funcionando corretamente.

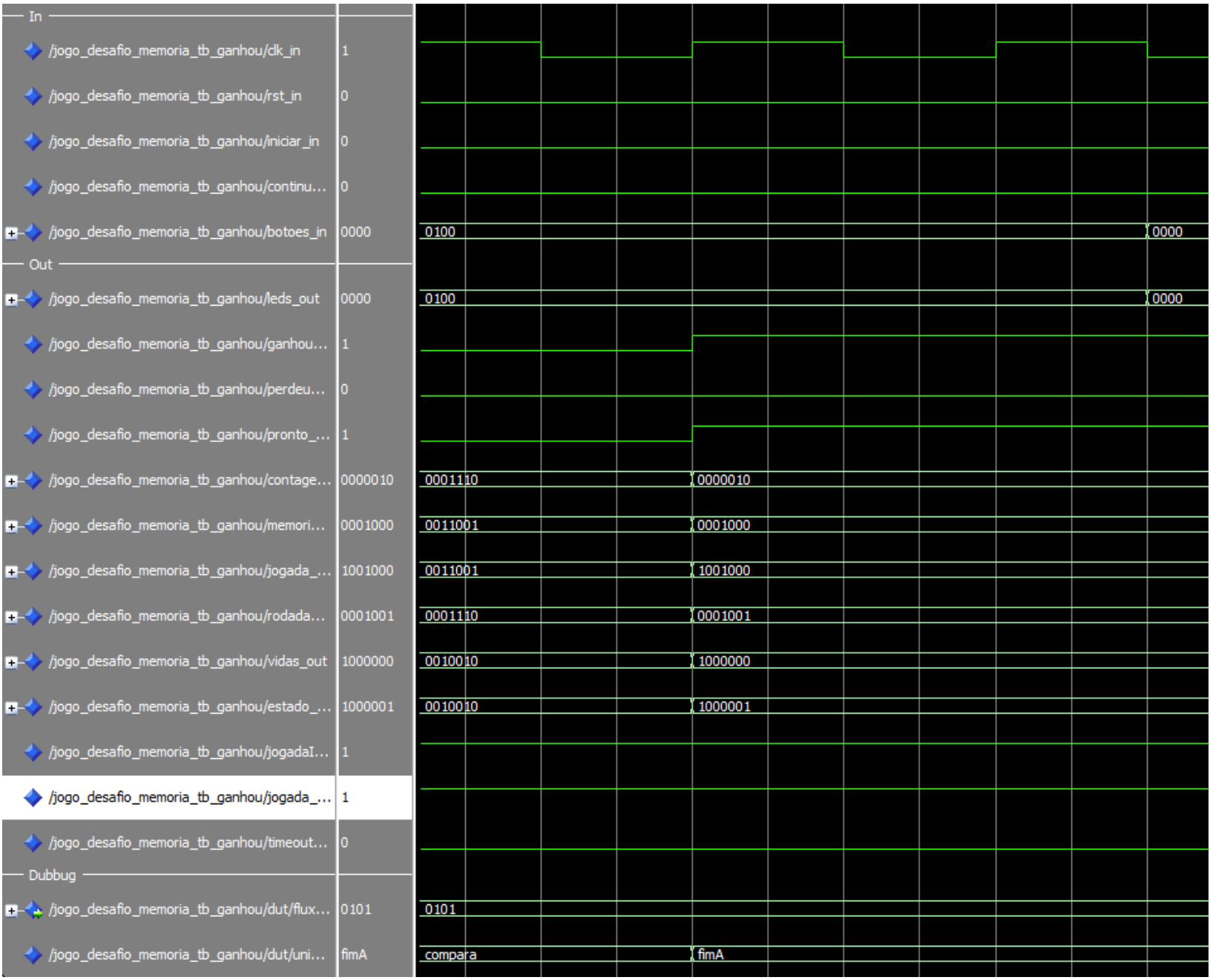
Cenários de testes 1 - vidas + display perdeu:


In
/jogo_desafio_... 0
/jogo_desafio_... 0
/jogo_desafio_... 0
/jogo_desafio_... 0
/jogo_desafio_... 0000
Out
/jogo_desafio_... 0000
/jogo_desafio_... 0
/jogo_desafio_... 1
/jogo_desafio_... 1
/jogo_desafio_... 0100100
/jogo_desafio_... 0011001
/jogo_desafio_... 1111001
/jogo_desafio_... 0100100
/jogo_desafio_... 1111001
/jogo_desafio_... 0000111
/jogo_desafio_... 1
/jogo_desafio_... 0
/jogo_desafio_... 1
Debbug
/jogo_desafio_... 1
/jogo_desafio_... fimT
Escrito "perdeu" em 7seg
mostraJogadas
espera



É possível ver que a vida do jogador decai de 5 à 1, como era esperado. Além disso, quando o jogo se encontra no estado fimE, todas as saídas para o display de 7 segmentos são mudados para “perdeu”.

Cenário de testes 2 - display “ganhou”:



Aqui suprimimos o resto do gtkwave que não nos importava, mostrando apenas que no estado fimA os sinais de debbug formam a palavra “ganhou”.

Por fim, seguem abaixo os esquemas gerados pelo Quartus da Visão RTL do circuito e fluxo de dados assim como o Diagrama de Transição de Estados:

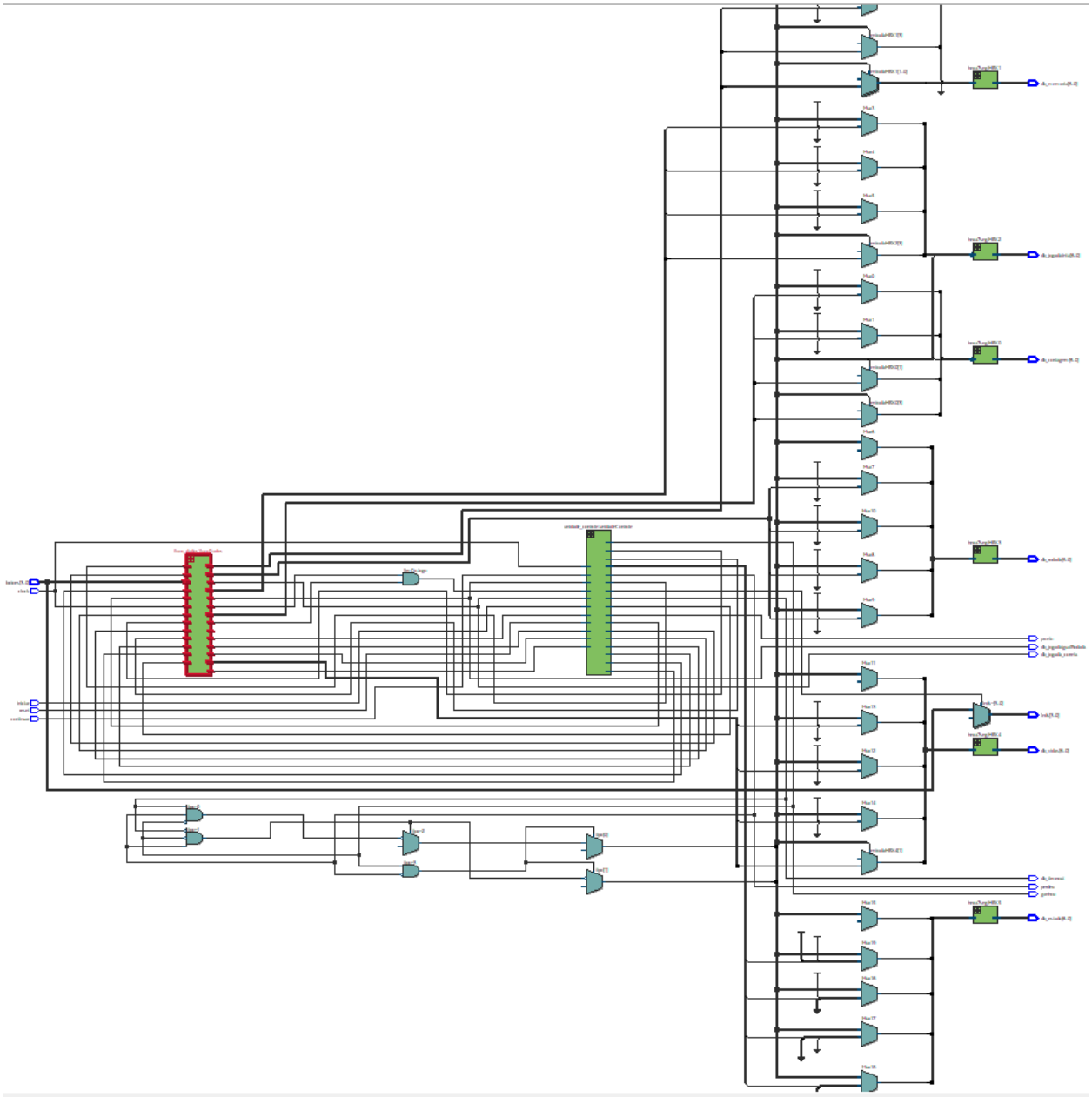


Figura 4: RTL viewer do circuito completo

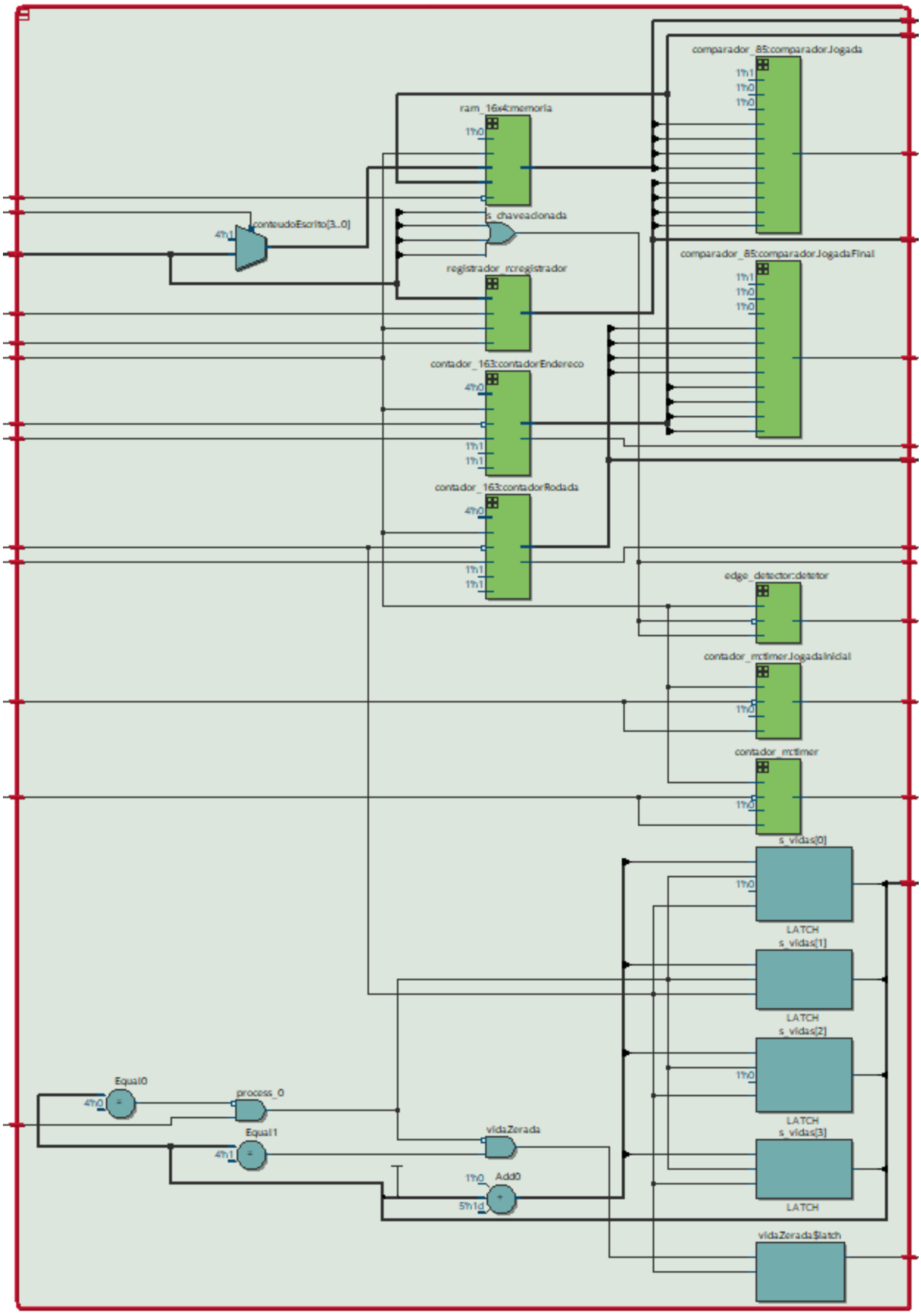

comparador_85:comparadorJogada
1'h1
1'h0
1'h0
ram_16x4:memoria
1'h0
s_chaveadonada
4'h1
conteudoEscrito[3..0]
registrador_rcregistrador
comparador_85:comparadorJogadaFinal
1'h1
1'h0
1'h0
contador_163:contadorEndereco
4'h0
1'h1
1'h1
contador_163:contadorRodada
4'h0
1'h1
1'h1
edge_detector:detetor
contador_rrctimer:JogadaInicial
1'h0
contador_rrctimer
1'h0
s_vidas[0]
1'h0
LATCH
s_vidas[1]
LATCH
s_vidas[2]
1'h0
LATCH
s_vidas[3]
LATCH
Equal0
4'h0
process_0
Equal1
4'h1
vidaZerada
1'h0
Add0
5'h1d
vidaZerada$latch

Figura 5: RTL viewer do fluxo de dados

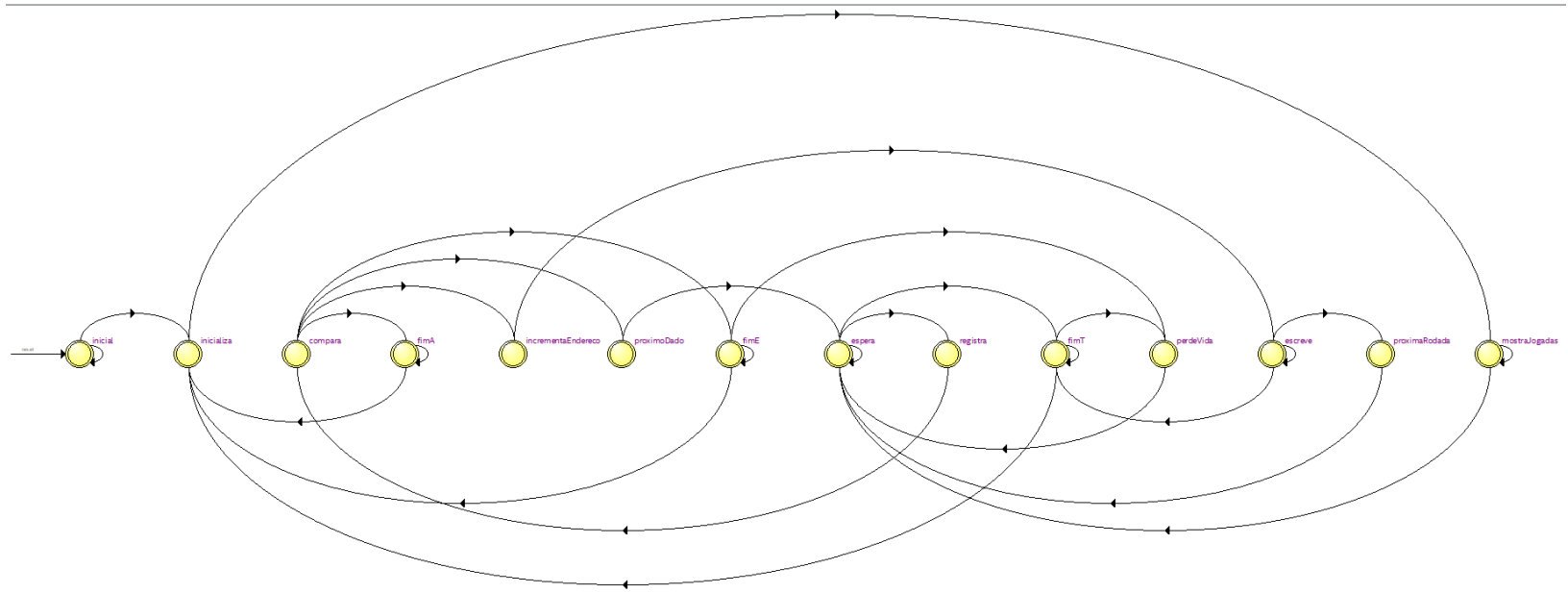


Figura 6: State Machine Viewer do circuito

	Source State	Destination State	Condition
1	compara	proximoDado	(!fim).(igual).(!enderecoFinal)
2	compara	fimE	(!igual)
3	compara	incrementaEndereco	(!fim).(igual).(enderecoFinal)
4	compara	fimA	(igual).(fim)
5	escreve	escreve	(!timeout).(!jogada)
6	escreve	fimT	(timeout)
7	escreve	proximaRodada	(!timeout).(jogada)
8	espera	fimT	(timeout)
9	espera	registra	(!timeout).(jogada)
10	espera	espera	(!timeout).(!jogada)
11	fimA	inicializa	(iniciar)
12	fimA	fimA	(!iniciar)
13	fimE	fimE	(!iniciar).(!continuar)
14	fimE	inicializa	(!iniciar).(continuar).(vidaZerada) + (iniciar).(!continuar) + (iniciar).(continuar).(vidaZerada)
15	fimE	perdeVida	(continuar).(!vidaZerada)
16	fimT	fimT	(!iniciar).(!continuar)
17	fimT	inicializa	(!iniciar).(continuar).(vidaZerada) + (iniciar)
18	fimT	perdeVida	(!iniciar).(continuar).(!vidaZerada)
19	incrementa...	escreve	
20	inicial	inicializa	(iniciar)
21	inicial	inicial	(!iniciar)
22	inicializa	mostraJogadas	
23	mostraJog...	espera	(timeoutJogadaInicial)
24	mostraJog...	mostraJogadas	(!timeoutJogadaInicial)
25	perdeVida	espera	
26	proximaRo...	espera	
27	proximoDa...	espera	

28	registra	compara	

Figura 7:Tabela de transição de estados do State Machine Viewer

A pinagem do circuito foi feita de acordo com a seguinte tabela:

	Sinal	Pino na Placa DE0-CV	Pino no FPGA	Analog Discovery
Entradas	CLOCK	GPIO_0_D0	PIN_N16	Patterns – Clock – 1 KHz – DIO
	RESET	GPIO_0_D1	PIN_B16	StaticIO – Button 0/1 – DIO1
	INICIAR	GPIO_0_D3	PIN_C16	StaticIO – Button 0/1 – DIO2
	CONTINUAR	GPIO_0_D5		StaticIO – Button 0/1 – DIO3
	BOTOES(0)	GPIO_0_D11	PIN_R22	StaticIO – Button 0/1 – DIO4
	BOTOES(1)	GPIO_0_D13	PIN_T22	StaticIO – Button 0/1 – DIO5
	BOTOES(2)	GPIO_0_D15	PIN_N19	StaticIO – Button 0/1 – DIO6
	BOTOES(3)	GPIO_0_D17	PIN_P19	StaticIO – Button 0/1 – DIO7
Saídas	PERDEU	GPIO_1_D11	PIN_J18	StaticIO – LED – DIO8
	GANHOU	GPIO_1_D13	PIN_G11	StaticIO – LED – DIO9
	PRONTO	GPIO_1_D15	PIN_J11	StaticIO – LED – DIO10
	LEDS(0)	GPIO_1_D17	PIN_A15	StaticIO – LED – DIO12
	LEDS(1)	GPIO_1_D19	PIN_L8	StaticIO – LED – DIO13
	LEDS(2)	GPIO_1_D21	PIN_B15	StaticIO – LED – DIO14
	LEDS(3)	GPIO_1_D23	PIN_E14	StaticIO – LED – DIO15
Depuração	DB_CONTAGEM	HEX0	PIN_AA22	

			PIN_Y21 PIN_Y22 PIN_W21 PIN_W22 PIN_V21 PIN_U21	
	DB_MEMORIA	HEX1	PIN_U22 PIN_AA17 PIN_AB18 PIN_AA18 PIN_AA19 PIN_AB20 PIN_AA20	
	DB_JOGADAFEITA	HEX2	PIN_AB21 PIN_AB22 PIN_V14 PIN_Y14 PIN_AA10 PIN_AB17 PIN_Y19	
	DB_RODADA	HEX3	PIN_V19 PIN_V18 PIN_U17 PIN_V16 PIN_Y17 PIN_W16 PIN_Y16	
	DB_VIDAS	HEX4	PIN_P9 PIN_Y15 PIN_U15 PIN_U16 PIN_V20 PIN_Y20 PIN_U20	
	DB_ESTADO	HEX5	PIN_W19 (6) PIN_C2	

			PIN_C1 PIN_P14 PIN_T14 PIN_M8 PIN_N9 (0)	
	DB_JOGADAIGUALRODADA	LED1	PIN_AA1	
	DB_JOGADACORRETA	LED0	PIN_AA2	
	DB_TIMEOUT	LED2	PIN_W2	

b. Parte prática em laboratório

O teste do circuito modificado será testado na aula e registrado aqui nessa seção.