

Relatório do 2º Projeto de Laboratório

Abraão Carvalho Gomes, Gabriel Rodrigues da Silva

¹Universidade Federal do Rio de Janeiro (UFRJ)

²EEL480 - Laboratório de Sistemas Digitais

abraao.cg@poli.ufrj.br, gabriel.rodrigues129@poli.ufrj.br

Resumo. Este relatório tem como objetivo mostrar o processo de construção de um jogo da forca numérico na linguagem VHDL, bem como demonstrar seu funcionamento na placa FPGA DE2-115.

1. Introdução

O projeto a ser implementado consiste em um jogo da forca número com as seguintes especificações:

- O sistema tem como entradas os switches da placa FPGA disponibilizada pelo LabsLand.
- Cada switch corresponde a um número decimal de 0 a 9;
- A palavra secreta consiste em um número composto de 6 decimais escolhidos pela dupla, não devendo ser escolhidas palavras secretas óbvias, como 000000 ou 123456;
- Deve-se exibir os erros restantes nos LEDs;
- Quando o jogador vencer, a placa FPGA mostra no display de 7 segmentos a letra G. Caso contrário, mostra-se a letra P;
- Inicialmente, os dígitos da palavra devem aparecer escondidos nos displays de 7 segmentos, representados por um traço. As vidas do jogador devem estar representadas por 3 LEDs acesos. A cada erro, um LED se apaga. A cada acerto, os dígitos corretos são exibidos nos displays de 7 segmentos correspondentes.

Dessa maneira, para atender os requisitos do projeto, deve-se desenvolver um ou mais módulos em VHDL que cumpram as funções acima. Nota-se que é preciso utilizar os conhecimentos teóricos sobre máquinas de estados em VHDL durante o projeto e, portanto, uma máquina de estados deve ser desenvolvida. Adianta-se que será utilizado apenas um módulo VHDL com todos os elementos necessários para o projeto¹.

¹O módulo do jogo da forca numérico em VHDL está disponível no GitHub *por meio deste link*.

2. Lógica Implementada

O esquema básico do jogo da força numérico é dado pela seguinte forma:

- O jogo tem 2 dificuldades: fácil e difícil. A palavra fácil escolhida foi 024689, por ser possível identificar um padrão simples. Já a palavra difícil escolhida foi 751394, por não ser um padrão facilmente identificável.
- As jogadas são feitas através de 10 switches, que representam os dígitos de 0 a 9. Para selecionar a dificuldade, deve-se ativar o switch 11 (modo fácil) ou o switch 12 (modo difícil). As vidas são representadas por três LEDs verdes no canto inferior direito da placa, e os dígitos da palavra secreta são mostrados em 6 displays, inicialmente ocultos por um traço.
- Cada tentativa do jogador será verificada e contabilizada como erro ou acerto em relação à palavra secreta de 6 dígitos. A cada jogada feita pelo jogador através dos switches, os acertos serão registrados em um vetor de 6 bits, e os erros serão registrados em um vetor de 4 bits.
- Em qualquer dificuldade, a cada tentativa correta do jogador, um bit do vetor de acertos correspondente à tentativa recebe nível alto ('1') e revela um dígito no display.
- Caso o jogador erre ao tentar adivinhar um dígito, o bit do vetor de erros correspondente à tentativa recebe nível alto ('1') e faz com que o jogador perca uma vida.
- Se pelo menos um bit do vetor de acertos for ('0') e o número de vidas do jogador chegar a zero, o jogador perde, e deve-se mostrar P no sétimo display da placa, contando da direita para a esquerda.
- Se todos os bits do vetor de acertos possuírem nível alto ('1') e a vida do jogador for maior que zero, então o jogador ganha, e deve-se mostrar G no sétimo display da placa, contando da direita para a esquerda.
- Em qualquer momento, o jogador pode resetar o jogo ao pressionar o primeiro botão à direita no simulador e escolher novamente a dificuldade.

A máquina de estados será representada por meio de um diagrama de estados.

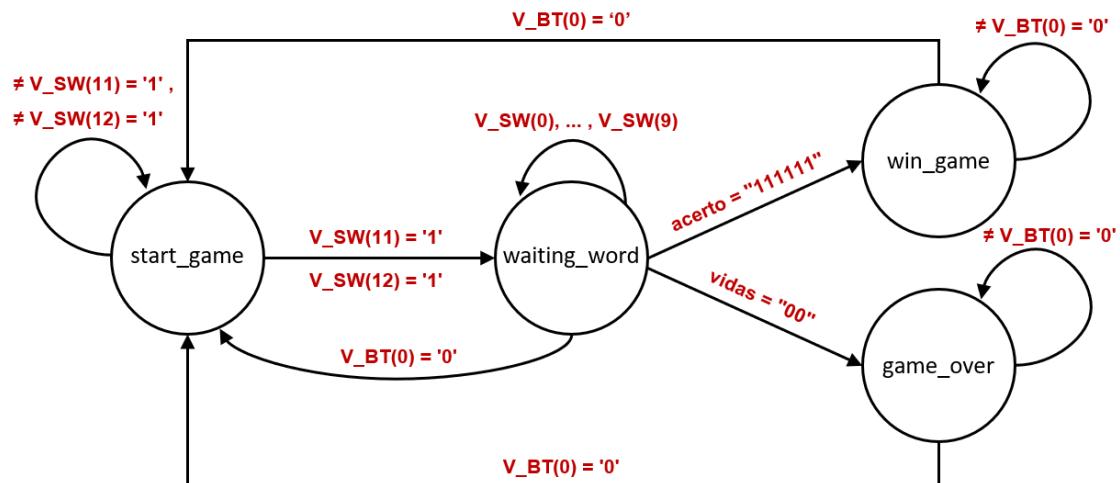


Figura 1. Diagrama de estados do jogo da força numérica a ser desenvolvido

3. Declaração de entradas, sinais e estados

Inicialmente, declaramos duas bibliotecas: ieee.std_logic_1164.all, necessária para usar os dados correspondentes à lógica padrão da biblioteca e que contém os tipos STD_LOGIC e STD_LOGIC_VECTOR; e declaramos a biblioteca ieee.std_logic_unsigned.all, necessária para realizar aritmética não sinalizada (sua aplicação será vista mais à frente quando for necessário subtrair uma vida do jogador).

Em seguida, declaramos as seguintes entradas e saídas:

- G_CLOCK_50 : representa o clock de 50MHz da placa FPGA;
- V_SW: representa os 13 switches da placa FPGA, sendo os switches V_SW(0), ..., V_SW(9) as entradas para as tentativas do jogador, V_SW(11) faz a seleção do modo fácil e V_SW(12) faz a seleção do modo difícil;
- V_BT : representa o botão da placa FPGA utilizado para resetar o jogo;
- G_LEDG: representam os 3 LEDs verdes da placa FPGA;
- G_LEDR: representam os 18 LEDs vermelhos da placa FPGA;
- G_HEX0, ..., G_HEX6: representam os sete displays de 7 segmentos da placa FPGA, sendo G_HEX6 o display que mostra P ou G, G_HEX5 é o primeiro dígito da palavra secreta, G_HEX4 é o segundo dígito da palavra secreta, G_HEX3 é o terceiro dígito da palavra secreta, G_HEX2 é o quarto dígito da palavra secreta, G_HEX1 é o quinto dígito da palavra secreta e, por fim, G_HEX0 é o sexto dígito da palavra secreta.

```
1 -- Declaração de bibliotecas:
2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_unsigned.all;
5
6 ENTITY jogo_forca IS
7   PORT(
8     G_CLOCK_50 : IN STD_LOGIC;          -- Clock de 50 MHz
9     V_SW       : IN STD_LOGIC_VECTOR(12 DOWNTO 0); -- Switches da placa FPGA
10    V_BT       : IN STD_LOGIC_VECTOR(0 DOWNTO 0); -- Botões da placa FPGA
11    G_LEDG     : OUT STD_LOGIC_VECTOR(2 DOWNTO 0); -- LEDs verdes
12    G_LEDR     : OUT STD_LOGIC_VECTOR(17 DOWNTO 0); -- LEDs vermelhos
13    G_HEX0, G_HEX1, G_HEX2, G_HEX3, G_HEX4, G_HEX5, -- Displays de
14    G_HEX6     : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)      -- 7 segmentos
15  );
16 END jogo_forca;
```

Declaração de entradas e saídas

O jogo da força projetado tem 4 estados, sendo eles:

- start_game : estado inicial, onde a dificuldade é selecionada;
- waiting_word : estado que lida com acertos e erros, e faz a transição quando o jogador ganhar ou perder;
- win_game : quando o jogador está nesse estado, significa que o jogador adivinhou a palavra secreta, e permite que o display mostre G;
- game_over : quando o jogador está nesse estado, significa que as vidas do jogador acabaram, e permite que o display mostre P.

```
18 ARCHITECTURE behavioral OF jogo_forca IS
19
20 TYPE states IS (start_game, waiting_word, win_game, game_over);
```

Declaração de estados

Em seguida, são declarados os seguintes sinais:

- `current_state`: estado atual da máquina de estados;
- `next_state`: próximo estado da máquina de estados;
- `acerto`: vetor de 6 bits que guarda os acertos do jogador, sendo um bit para cada dígito correto;
- `erro`: vetor de 4 bits que guarda os erros do jogador, sendo um bit para cada dígito incorreto;
- `dificuldade`: vetor de 2 bits que guarda a dificuldade escolhida. Inicialmente, seu valor padrão é "00". Quando tem valor "01", indica que o modo fácil foi selecionado, e quando tem valor "10", indica que o modo difícil foi selecionado;
- `vidas`: vetor de 2 bits que guarda as vidas do jogador. Inicialmente, recebe "11", e a cada erro, subtrai-se 1. Ou seja, a cada erro, o seu valor pode mudar de "11" para "10", "01" ou até "00";
- `win`: sinal de 1 bit que só recebe nível alto quando o jogador está no estado `win_game`, e que faz acender G no display.
- `lose`: sinal de 1 bit que só recebe nível alto quando o jogador está no estado `game_over`, e que faz acender P no display.

```
21 SIGNAL current_state, next_state : states;
22
23 SIGNAL acerto : STD_LOGIC_VECTOR(5 DOWNTO 0);
24 SIGNAL erro : STD_LOGIC_VECTOR(3 DOWNTO 0);
25 SIGNAL dificuldade, vidas : STD_LOGIC_VECTOR(1 DOWNTO 0);
26 SIGNAL win, lose : STD_LOGIC;
```

Declaração de sinais

Para realizar a transição de sinal², foi utilizado um process, em que o estado atual realiza a transição para o próximo estado quando há subida de clock. Adicionalmente, foi configurado que, quando o primeiro botão da placa for pressionado (`V_BT` = '`0`', lógica negativa), o próximo estado será o `start_game`, possibilitando resetar o jogo.

```
28 BEGIN
29
30 PROCESS(G_CLOCK_50, V_BT)
31 BEGIN
32 -----
33 --- Processo que realiza a transição do estado atual para o próximo estado
34
35 IF V_BT(0) = '0' THEN
36     current_state <= start_game; -- Estado inicial
37
38 ELSIF rising_edge(G_CLOCK_50) THEN
39     current_state <= next_state; -- Mudança de estado
40
41 END IF;
42 END PROCESS;
43
44 --- Fim do processo
45 -----
```

Processo para transição de estados

Agora, inicia-se o projeto da máquina de estados². Em outro process, foi utilizada a estrutura CASE / WHEN para cada estado, em função do estado atual. Tal estrutura possibilita a separação do jogo por estados, de forma que cada bloco WHEN determina o que deve acontecer quando se está em um estado. Nesse sentido, é viável analisar cada estado individualmente a seguir.

²Com base no livro "Eletrônica Digital Moderna e VHDL", de Volnei A.Pedroni, Elsevier, 1ª Ed., 2010, cap. 23, págs. 527 e 528.

3.1. Start Game

Sempre que o estado atual for `start_game`, os sinais `acerto`, `erro`, `win` e `lose` serão zerados, para garantir que o jogo esteja com todos os valores em default quando esta for a primeira vez ou quando o jogador复位 o jogo apertando o botão.

Para iniciar o jogo, o jogador deve selecionar a dificuldade com os switches 11 ou 12 (fácil e difícil, respectivamente). No código, utilizamos 024689 como palavra fácil e 751394 como palavra difícil.

Se o jogador escolheu a palavra fácil, o vetor `dificuldade` recebe "01". Caso contrário, se o jogador escolheu a palavra difícil, o vetor `dificuldade` recebe "10". Independentemente da escolha, o jogador recebe três vidas. Assim que algum desses switches apresentem nível lógico alto, o próximo estado será `waiting_word`, indicando que o jogo está em andamento.

A estrutura `IF` / `ELSIF` / `ELSE` é utilizada para filtrar as condições acima, controlando a mudança de estados conforme a dificuldade é ou não selecionada utilizando os switches 11 e 12. O `ELSE` garante que, se nenhuma dificuldade for selecionada, o próximo estado continuará sendo `start_game`.

```
47 -----  
48 --- Início da máquina de estados  
49  
50 PROCESS(current_state, dificuldade, acerto, erro, V_SW, vidas)  
51 BEGIN  
52  
53 CASE current_state IS  
54  
55 WHEN start_game => -- Sinais são zerados ao iniciar ou reiniciar o jogo  
56     acerto <= "000000";  
57     erro <= "0000";  
58     win <= '0';  
59     lose <= '0';  
60  
61     IF (V_SW(11) = '1') THEN -- Escolhendo a palavra secreta fácil: 024689  
62         dificuldade <= "01";  
63         vidas <= "11";  
64         next_state <= waiting_word;  
65  
66     ELSIF (V_SW(12) = '1') THEN -- Escolhendo a palavra secreta difícil: 751394  
67         dificuldade <= "10";  
68         vidas <= "11";  
69         next_state <= waiting_word;  
70  
71     ELSE  
72         next_state <= start_game;  
73  
74     END IF;  
75 ----- Lógica do estado start_game -----
```

3.2. Waiting Word

Inicialmente, um filtro é introduzido com uma declaração condicional (IF) para impedir que seja possível jogar sem vidas. Desse modo, se o número de vidas chegar a "00", o próximo estado será game_over.

Caso o jogador ainda tenha vidas, o próximo passo é verificar se o jogador já conseguiu acertar todos os números necessários. Com isso, a condicional anterior prossegue para o ELSIF, que verifica se o vetor de acertos é igual a "111111". Se sim, então o próximo estado será win_game.

```
76 WHEN waiting_word =>
77     IF (vidas = "00") THEN -- Se as vidas acabarem:
78         next_state <= game_over;
79
80     ELSIF (acerto = "111111") THEN -- Se todas as palavras foram descobertas:
81         next_state <= win_game;
82
```

Lógica inicial do estado waiting_word

Se nenhuma das condições anteriores for satisfeita, então o jogo ainda está em andamento e inicia-se um novo bloco condicional com ELSE. Nesse ponto do código, é onde será verificado se o jogador acertou ou não um número da palavra secreta. Para tanto, cada algarismo/switch é verificado individualmente com uma sequência IF/ELSIF/ELSIF/.../ELSIF, sendo que 6 deles são corretos e 4 errados.

Para cada verificação, o próximo estado será waiting_word. É preciso ressaltar que as 10 verificações são feitas tanto para a dificuldade fácil quanto para a difícil, ficando essa separação por dificuldade dividida por uma estrutura IF/ELSE.

Em cada ELSIF, é necessário verificar se o switch já foi acionado anteriormente, visto que uma palavra não pode ser acertada ou errada duas vezes no mesmo jogo. Para evitar isso, a cada vez que uma palavra é acertada ou errada ao passar por um ELSIF, o bit do sinal acerto ou do sinal erro correspondente àquele algarismo recebe '1'. Desse modo, usando uma porta AND para verificar se o bit do sinal ainda não foi ativo, só será considerado um acerto ou erro se for a primeira vez em que o interruptor é acionado. Por exemplo:

```
p_fácil = 024689
acerto = "000000"
Supondo que V_SW(0) é ativado: acerto(5) = '1'
p_fácil = 024689
acerto = 100000
Supondo que V_SW(4) é ativado: acerto(3) = '1'
p_fácil = 024689
acerto = 101000
Supondo que V_SW(0) é ativado novamente:
acerto(5) já está em nível alto. Não haverá mudança.
p_fácil = 024689
acerto = 101000
```

A cada erro constatado, a vida total do jogador será reduzida em 1 e o próximo estado será `waiting_word`, até que o jogador acerte a palavra secreta ou perca todas as vidas. Continuando o exemplo anterior:

```
p_fácil = 024689  
acerto = 101000  
erro = 0000  
vidas = 11
```

Supondo que `V_SW(1)` é ativado:
1 não está na palavra fácil, então
`erro(0) = '1'` e `vidas <= vidas - 1`
p_fácil = 024689
acerto = 101000
erro = 0001
vidas = 10

Supondo que `V_SW(7)` é ativado:
7 não está na palavra fácil, então
`erro(3) = '1'` e `vidas <= vidas - 1`
p_fácil = 024689
acerto = 101000
erro = 1001
vidas = 01

Trecho de código para o estado `waiting_word` quando o modo fácil é selecionado:

```
76 WHEN waiting_word =>
77     IF (vidas = "00") THEN -- Se as vidas acabarem:
78         next_state <= game_over;
79
80     ELSIF (acerto = "111111") THEN -- Se todas as palavras foram descobertas:
81         next_state <= win_game;
82
83     ELSE
84         -- Modo fácil: 024689
85         IF (dificuldade = "01") THEN
86
87             IF (V_SW(0) = '1' AND acerto(5) = '0') THEN
88                 acerto(5) <= '1';
89                 next_state <= waiting_word;
90
91             ELSIF (V_SW(2) = '1' AND acerto(4) = '0') THEN
92                 acerto(4) <= '1';
93                 next_state <= waiting_word;
94
95             ELSIF (V_SW(4) = '1' AND acerto(3) = '0') THEN
96                 acerto(3) <= '1';
97                 next_state <= waiting_word;
98
99             ELSIF (V_SW(6) = '1' AND acerto(2) = '0') THEN
100                acerto(2) <= '1';
101                next_state <= waiting_word;
102
103            ELSIF (V_SW(8) = '1' AND acerto(1) = '0') THEN
104                acerto(1) <= '1';
105                next_state <= waiting_word;
106
107            ELSIF (V_SW(9) = '1' AND acerto(0) = '0') THEN
108                acerto(0) <= '1';
109                next_state <= waiting_word;
110
111            ELSIF (V_SW(1) = '1' AND erro(0) = '0') THEN
112                erro(0) <= '1';
113                vidas <= vidas - 1;
114                next_state <= waiting_word;
115
116            ELSIF (V_SW(3) = '1' AND erro(1) = '0') THEN
117                erro(1) <= '1';
118                vidas <= vidas - 1;
119                next_state <= waiting_word;
120
121            ELSIF (V_SW(5) = '1' AND erro(2) = '0') THEN
122                erro(2) <= '1';
123                vidas <= vidas - 1;
124                next_state <= waiting_word;
125
126            ELSIF (V_SW(7) = '1' AND erro(3) = '0') THEN
127                erro(3) <= '1';
128                vidas <= vidas - 1;
129                next_state <= waiting_word;
130
131        ELSE
132            next_state <= waiting_word;
133        END IF;
```

Lógica do estado `waiting_word` para o modo fácil

Trecho de código para o estado `waiting_word` quando o modo difícil é selecionado:

```
135      -- Modo difícil: 751394
136      ELSIF (dificuldade = "10") THEN
137
138          IF (V_SW(7) = '1' AND acerto(5) = '0') THEN
139              acerto(5) <= '1';
140              next_state <= waiting_word;
141
142          ELSIF (V_SW(5) = '1' AND acerto(4) = '0') THEN
143              acerto(4) <= '1';
144              next_state <= waiting_word;
145
146          ELSIF (V_SW(1) = '1' AND acerto(3) = '0') THEN
147              acerto(3) <= '1';
148              next_state <= waiting_word;
149
150          ELSIF (V_SW(3) = '1' AND acerto(2) = '0') THEN
151              acerto(2) <= '1';
152              next_state <= waiting_word;
153
154          ELSIF (V_SW(9) = '1' AND acerto(1) = '0') THEN
155              acerto(1) <= '1';
156              next_state <= waiting_word;
157
158          ELSIF (V_SW(4) = '1' AND acerto(0) = '0') THEN
159              acerto(0) <= '1';
160              next_state <= waiting_word;
161
162          ELSIF (V_SW(0) = '1' AND erro(0) = '0') THEN
163              erro(0) <= '1';
164              vidas <= vidas - 1;
165              next_state <= waiting_word;
166
167          ELSIF (V_SW(2) = '1' AND erro(1) = '0') THEN
168              erro(1) <= '1';
169              vidas <= vidas - 1;
170              next_state <= waiting_word;
171
172          ELSIF (V_SW(6) = '1' AND erro(2) = '0') THEN
173              erro(2) <= '1';
174              vidas <= vidas - 1;
175              next_state <= waiting_word;
176
177          ELSIF (V_SW(8) = '1' AND erro(3) = '0') THEN
178              erro(3) <= '1';
179              vidas <= vidas - 1;
180              next_state <= waiting_word;
181
182      ELSE
183          next_state <= waiting_word;
184      END IF;
185  END IF;
186 END IF;
```

Lógica do estado `waiting_word` para o modo difícil

3.3. Win Game

Quando o jogador descobre a palavra secreta, o próximo estado será `win_game`. Nesse estado, o sinal `win` recebe '1' e o próximo estado será o próprio `win_game`, estabelecendo um loop que só é interrompido quando o botão 0 de reset é pressionado. Por fim, quando `win = '1'`, deve-se acender G no sétimo display da placa, contando da direita para a esquerda.

```
188 WHEN win_game =>
189     win <= '1';
190     next_state <= win_game;
191
```

Lógica do estado win-game

3.4. Game Over

No estado `game_over`, o sinal `lose` recebe '1' e o próximo estado será `game_over`, estabelecendo um loop que só é interrompido quando o botão 0 (reset) é pressionado.

```
192 WHEN game_over =>
193     lose <= '1';
194     next_state <= game_over;
195
196     END CASE;
197 END PROCESS;
198
199 --- Fim da máquina de estados
200
```

Lógica do estado game-over

4. Outputs na placa FPGA

Com a lógica do jogo da forca já implementada, é preciso que as informações referentes ao jogo sejam exibidas nos displays e LEDs da placa. Isso é feito com os seguintes parâmetros:

- São utilizados os displays de 0 a 6, sendo que os displays de 0 a 5 exibem o jogo da forca contendo a palavra secreta e utilizando traços para dígitos ocultos. O display 6 exibe 'P' para perdeu ou 'G' para ganhou, quando o jogo é encerrado;
- Os 3 primeiros LEDs verdes representam a vida do jogador, de modo que cada led aceso representa uma vida;
- Todos os LEDs vermelhos são ativados como efeito visual quando o jogador perde.

Para mostrar os traços das palavras ocultas, os dígitos a serem revelados e as letras P e G, é necessário utilizar uma codificação para representar os dígitos e símbolos nos displays de 7 segmentos. Vale ressaltar que, para acender um segmento, é necessário aplicar um nível lógico baixo ('0'). Prosseguindo, utilizou-se a seguinte tabela:

Dígitos	Saídas para o display de 7 segmentos						
	g	f	e	d	c	b	a
0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	1
2	0	1	0	0	1	0	0
3	0	1	1	0	0	0	0
4	0	0	1	1	0	0	1
5	0	0	1	0	0	1	0
6	0	0	0	0	0	1	0
7	1	1	1	1	0	0	0
8	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0
Símbolos	Saídas para o display de 7 segmentos						
	g	f	e	d	c	b	a
_	1	1	1	0	1	1	1
P	0	0	0	1	1	0	0
G	0	0	0	0	0	1	0

Figura 2. Dígitos, símbolos e seus respectivos segmentos de saída para os displays de 7 segmentos

Com base na tabela, criou-se outro process, que determina as condições para acender ou apagar os LEDs e displays através de estruturas IF/CASE/WHEN. Primeiramente, espera-se que nenhum LED ou display acenda antes do jogador selecionar alguma dificuldade. Sendo assim, quando o estado atual for start_game, nada deve acender.

```

202 -----
203 --- Processo que configura as condições para acender os displays e leds:
204
205 PROCESS(current_state, dificuldade, acerto, vidas, win, lose)
206 BEGIN
207
208     IF (current_state = start_game) THEN -- Não deve acender nada antes de
209         G_HEX6 <= "1111111";           -- selecionar a dificuldade
210         G_HEX5 <= "1111111";
211         G_HEX4 <= "1111111";
212         G_HEX3 <= "1111111";
213         G_HEX2 <= "1111111";
214         G_HEX1 <= "1111111";
215         G_HEX0 <= "1111111";
216         G_LEDG <= "000";
217         G_LEDR <= "00000000000000000000000000000000";
218
219     Início do processo de configuração dos displays e LEDs

```

Depois que o jogador selecionar alguma dificuldade, primeiramente, deve-se verificar se o jogador ganhou ou perdeu quando os sinais `win` ou `lose` forem '1'. Se isso não ocorrer, então o jogo ainda não terminou, e o display que mostra 'G' ou 'P' deve permanecer apagado.

```

219 ELSE
220     -- Para qualquer outro estado que não seja o start_game:
221
222     -- Mostrando G no display G_HEX6 para as duas dificuldades
223     IF win = '1' THEN
224
225         CASE win IS
226             WHEN '1' => G_HEX6 <= "0000010"; -- mostra G
227             WHEN OTHERS => G_HEX6 <= "1111111"; -- display apagado
228         END CASE;
229
230     -- Mostrando P no display G_HEX6 para as duas dificuldades
231     ELSIF lose = '1' THEN
232
233         CASE lose IS
234             WHEN '1' =>
235                 G_HEX6 <= "0001100"; -- mostra P
236                 G_LEDR <= "111111111111111111111111"; -- leds vermelhos acesos
237
238             WHEN OTHERS =>
239                 G_HEX6 <= "1111111"; -- display apagado
240                 G_LEDR <= "00000000000000000000000000000000"; -- leds vermelhos apagados
241         END CASE;
242
243     -- Se o jogo não tiver terminado, o display G_HEX6 permanece apagado.
244     ELSE
245         G_HEX6 <= "1111111"; -- display apagado
246         G_LEDR <= "00000000000000000000000000000000"; -- leds vermelhos apagados
247     END IF;
248
249     Configuração para acender ou não as letras P e G

```

Agora, cada mudança do sinal `vidas` será mostrado nos LEDs, bem como cada mudança do sinal `acerto`. É importante relembrar que, cada bit do sinal `acerto` que estiver em nível alto, faz com que o respectivo dígito acenda no display de 7 segmentos. O trecho de código abaixo determina o que será mostrado em cada display quando o modo fácil é selecionado.

```

249    -- Mostrando as vidas nos leds verdes para as duas dificuldades
250    CASE vidas IS
251        WHEN "11" => G_LEDG <= "111";
252        WHEN "10" => G_LEDG <= "011";
253        WHEN "01" => G_LEDG <= "001";
254        WHEN OTHERS => G_LEDG <= "000";
255    END CASE;
256
257    -- Mostrando palavras corretas nos displays para o modo fácil: 024689
258    IF (dificuldade = "01") THEN
259
260        CASE acerto(5) IS
261            WHEN '1' => G_HEX5 <= "1000000"; -- mostra 0
262            WHEN OTHERS => G_HEX5 <= "1110111"; -- palavra oculta
263        END CASE;
264
265        CASE acerto(4) IS
266            WHEN '1' => G_HEX4 <= "0100100"; -- mostra 2
267            WHEN OTHERS => G_HEX4 <= "1110111"; -- palavra oculta
268        END CASE;
269
270        CASE acerto(3) IS
271            WHEN '1' => G_HEX3 <= "0011001"; -- mostra 4
272            WHEN OTHERS => G_HEX3 <= "1110111"; -- palavra oculta
273        END CASE;
274
275        CASE acerto(2) IS
276            WHEN '1' => G_HEX2 <= "0000010"; -- mostra 6
277            WHEN OTHERS => G_HEX2 <= "1110111"; -- palavra oculta
278        END CASE;
279
280        CASE acerto(1) IS
281            WHEN '1' => G_HEX1 <= "0000000"; -- mostra 8
282            WHEN OTHERS => G_HEX1 <= "1110111"; -- palavra oculta
283        END CASE;
284
285        CASE acerto(0) IS
286            WHEN '1' => G_HEX0 <= "0010000"; -- mostra 9
287            WHEN OTHERS => G_HEX0 <= "1110111"; -- palavra oculta
288        END CASE;
289

```

Acendendo LEDs das vidas e displays correspondentes aos dígitos do modo fácil

Por fim, o trecho de código abaixo determina o que será mostrado em cada display quando o modo difícil é selecionado. Após termos definido todos os casos, o process é encerrado e terminamos o código.

```
249      -- Mostrando palavras corretas nos displays para o modo difícil: 751394
250      ELSIF (dificuldade = "10") THEN
251
252          CASE acerto(5) IS
253              WHEN '1' => G_HEX5 <= "1111000"; -- mostra 7
254              WHEN OTHERS => G_HEX5 <= "1110111";   -- palavra oculta
255          END CASE;
256
257          CASE acerto(4) IS
258              WHEN '1' => G_HEX4 <= "0010010"; -- mostra 5
259              WHEN OTHERS => G_HEX4 <= "1110111";   -- palavra oculta
260          END CASE;
261
262          CASE acerto(3) IS
263              WHEN '1' => G_HEX3 <= "1111001"; -- mostra 1
264              WHEN OTHERS => G_HEX3 <= "1110111";   -- palavra oculta
265          END CASE;
266
267          CASE acerto(2) IS
268              WHEN '1' => G_HEX2 <= "0110000"; -- mostra 3
269              WHEN OTHERS => G_HEX2 <= "1110111";   -- palavra oculta
270          END CASE;
271
272          CASE acerto(1) IS
273              WHEN '1' => G_HEX1 <= "0010000"; -- mostra 9
274              WHEN OTHERS => G_HEX1 <= "1110111";   -- palavra oculta
275          END CASE;
276
277          CASE acerto(0) IS
278              WHEN '1' => G_HEX0 <= "0011001"; -- mostra 4
279              WHEN OTHERS => G_HEX0 <= "1110111";   -- palavra oculta
280          END CASE;
281      END IF;
282  END IF;
283 END PROCESS;
284
285 --- Fim do processo
286 -----
287
288 END behavioral;
```

Acendendo displays correspondentes aos dígitos do modo difícil

5. Funcionamento no simulador LabsLand

Para demonstrar o funcionamento na prática, foram simulados dois jogos na plataforma LabsLand.

5.1. Primeiro jogo

No primeiro jogo, será selecionado o modo fácil (palavra secreta = 024689) ao ativar o switch 11, e serão efetuadas jogadas até vencer. Como dito anteriormente, os displays e LEDs acenderão somente depois de escolher uma dificuldade.

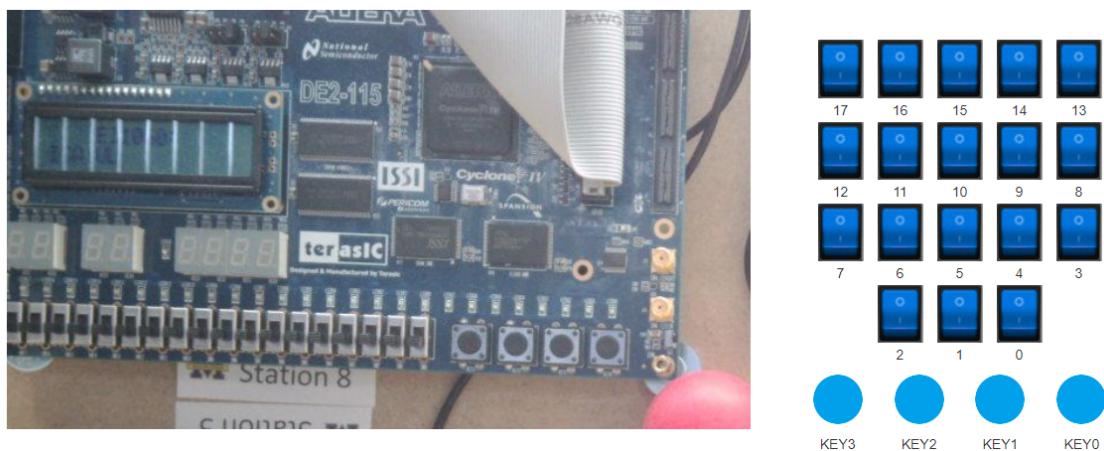


Figura 3. Início

Com o modo fácil selecionado, os displays mostram, inicialmente, um traço em cada palavra secreta, e os LEDs verdes indicam as três vidas do jogador.

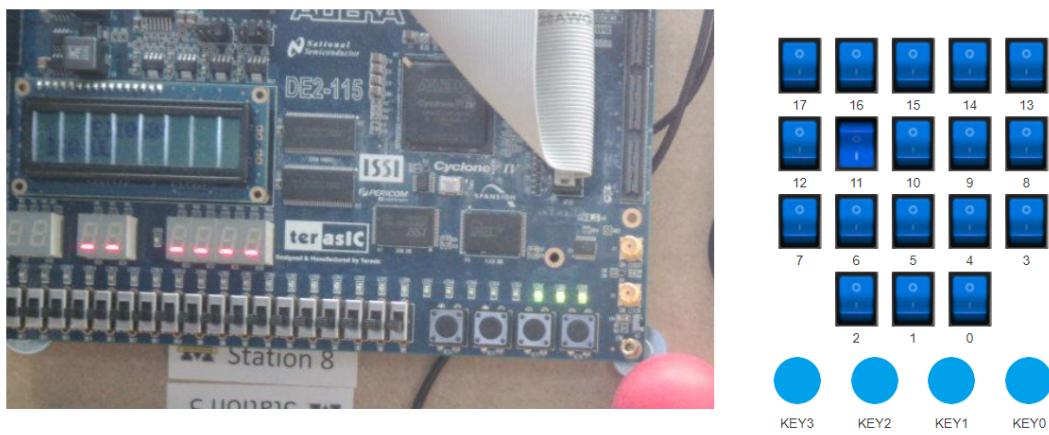


Figura 4. Modo fácil selecionado (switch 11).

Agora, o jogador deve tentar adivinhar a palavra secreta selecionando qualquer um dos switches de 0 a 9. Nesse caso, o switch 0 será pressionado.

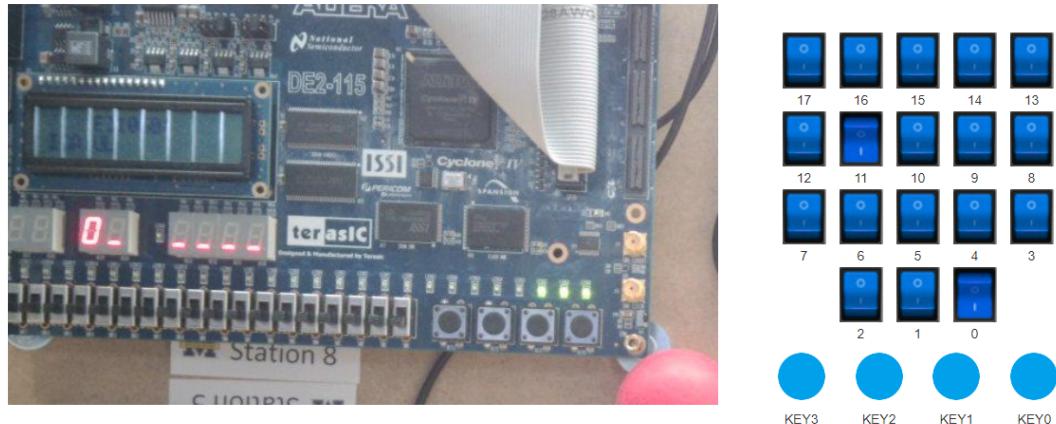


Figura 5. Tentativa realizada : '0'; o '0' aparece no display.

O palpite estava correto: o dígito 0 faz parte da palavra secreta. Nesse caso, nenhuma vida é perdida, e o jogador pode tentar adivinhar novamente os outros dígitos escondidos. Pressiona-se, então, o switch 5.

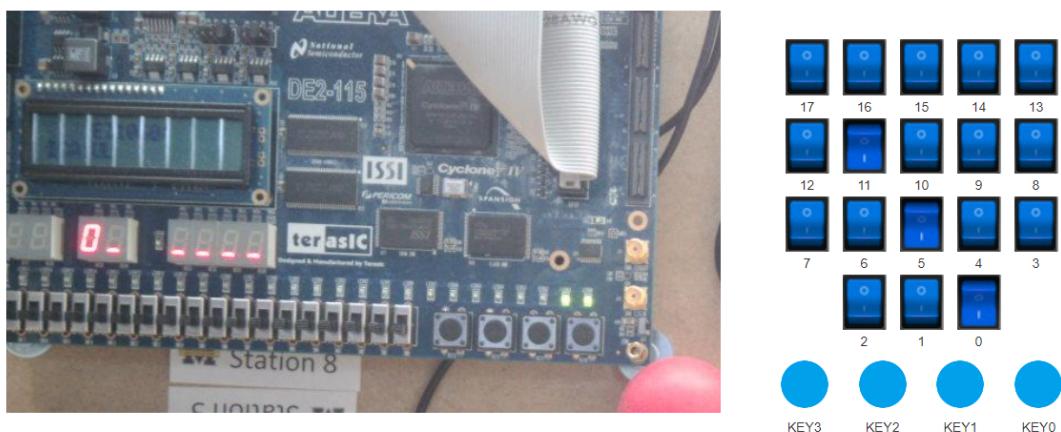


Figura 6. Tentativa realizada : '5' ; uma vida é perdida.

O palpite estava errado: o dígito 5 não faz parte da palavra secreta. Nesse caso, perde-se uma vida, restando apenas 2 LEDs acesos. Como as vidas não chegaram a zero, o jogador pode continuar as tentativas. Pressiona-se, então, o switch 4.

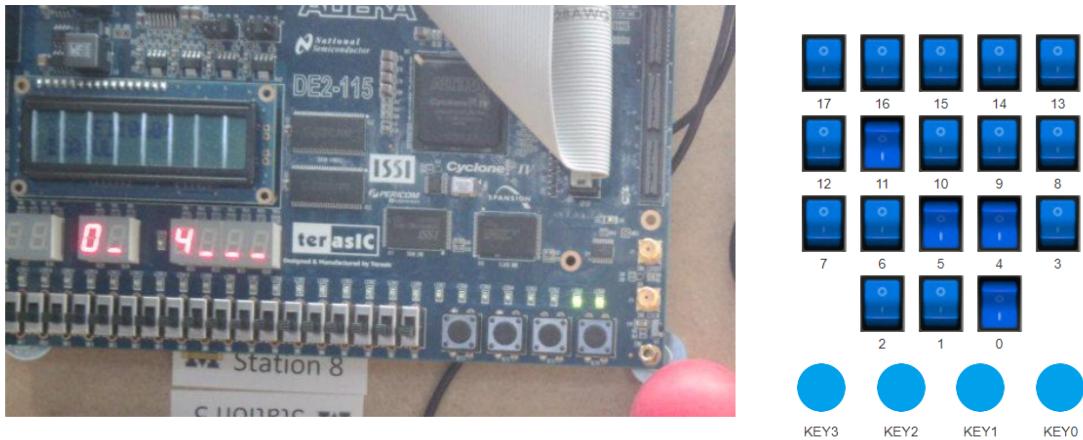


Figura 7. Tentativa realizada : '4'; o '4' aparece no display.

O palpite estava correto: o dígito 4 faz parte da palavra secreta. Nesse caso, as vidas são mantidas, e o jogador pode tentar adivinhar novamente os outros dígitos escondidos. Pressiona-se, então, o switch 2.

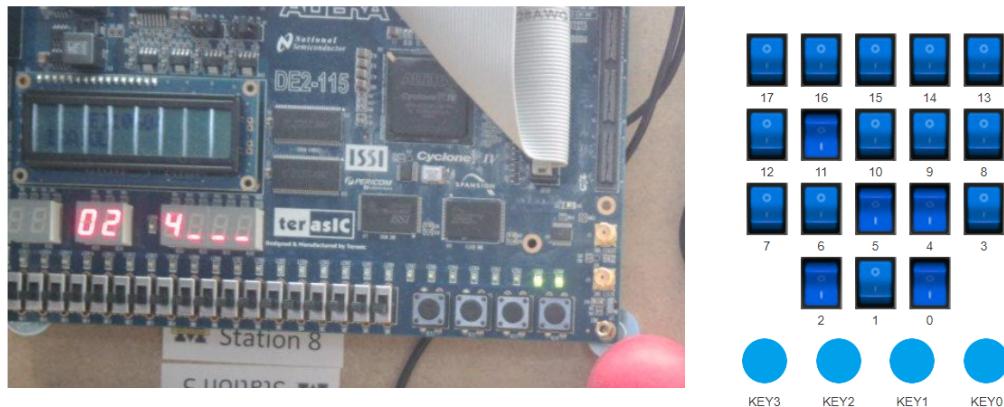


Figura 8. Tentativa realizada : '2'; o '2' aparece no display.

O palpite estava correto: o dígito 2 faz parte da palavra secreta. Nesse caso, as vidas são mantidas, e o jogador pode tentar adivinhar novamente os outros dígitos escondidos. Pressiona-se, então, o switch 9.

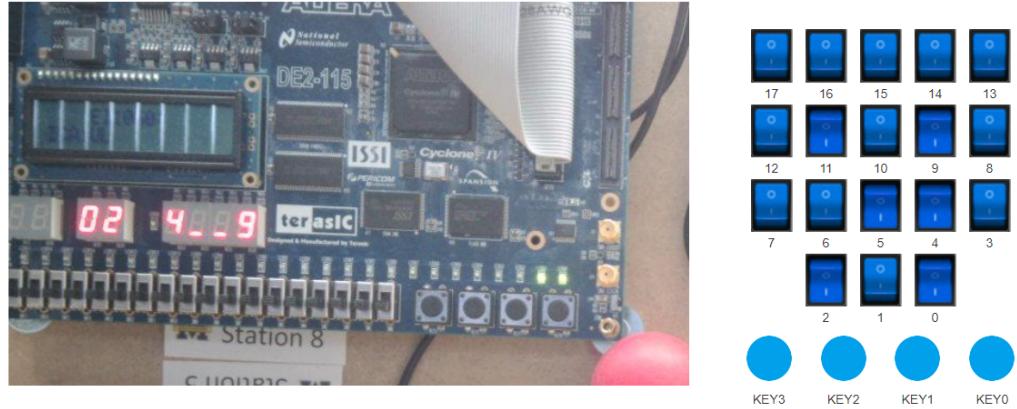


Figura 9. Tentativa realizada : '9'; o '9' aparece no display.

O palpite estava correto: o dígito 9 faz parte da palavra secreta. Nesse caso, as vidas são mantidas, e o jogador pode tentar adivinhar novamente os outros dígitos escondidos. Pressiona-se, então, o switch 6.

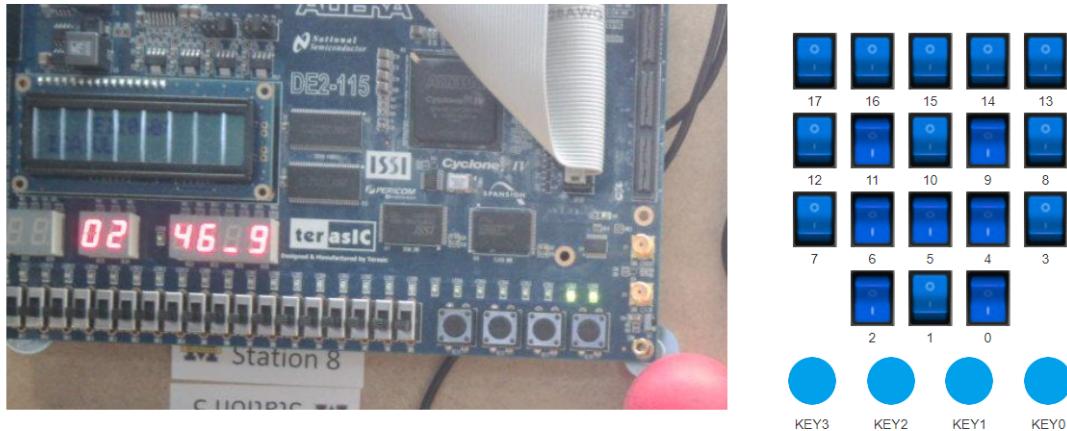


Figura 10. Tentativa realizada : '6'; o '6' aparece no display.

O palpite estava correto: o dígito 6 faz parte da palavra secreta. Nesse caso, as vidas são mantidas, e o jogador pode tentar adivinhar novamente os outros dígitos escondidos. Pressiona-se, então, o switch 8.

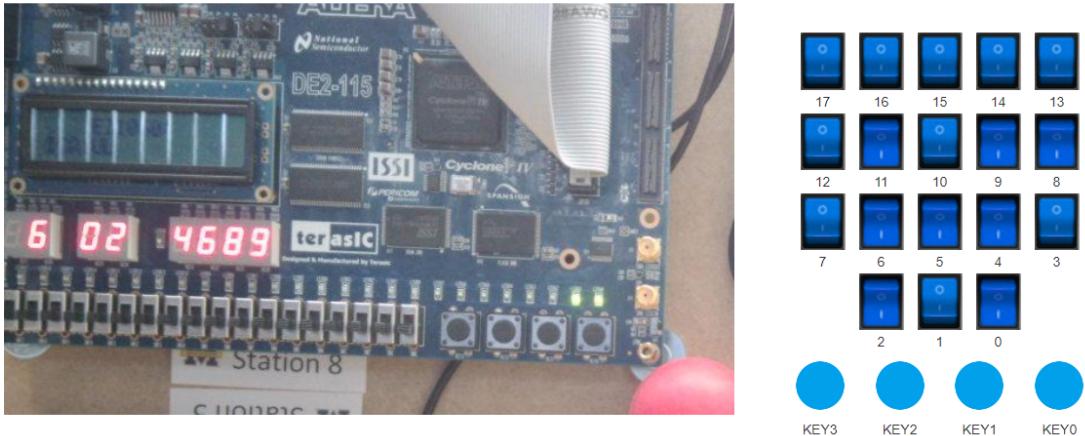


Figura 11. Tentativa realizada : '8'; o '8' aparece no display, assim como o 'G' no display mais à esquerda, indicando que o jogador venceu.

O palpite estava correto: o dígito 8 faz parte da palavra secreta. O jogador conseguiu identificar todos os dígitos da palavra secreta e ganhou o jogo. Sendo assim, mostra-se G no sétimo display, contando da direita para a esquerda.

Caso o jogador queira jogar novamente, é necessário desativar todos os switches, apertar o botão KEY0 para reiniciar o jogo e, por fim, escolher novamente a dificuldade ao ligar os switches 11 ou 12.

5.2. Segundo jogo

No segundo jogo, será selecionado o modo difícil (palavra secreta = 751394) ao ativar o switch 12, e serão efetuadas jogadas até perder. Novamente, os displays e LEDs acenderão somente depois de escolher uma dificuldade.

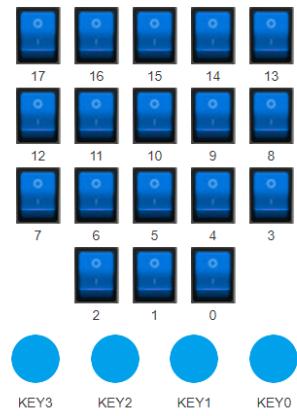
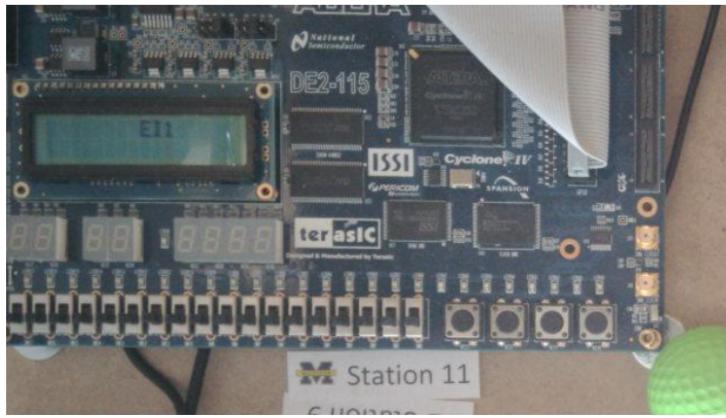


Figura 12. Inicio

Com o modo difícil selecionado, os displays mostram, inicialmente, um traço em cada palavra secreta, e os LEDs verdes indicam as três vidas do jogador.

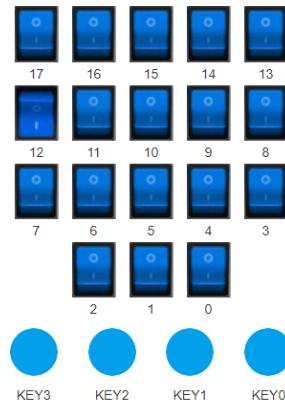


Figura 13. Modo difícil selecionado (Switch 12)

Agora, o jogador deve tentar adivinhar a palavra secreta selecionando qualquer um dos switches de 0 a 9. Nesse caso, o switch 6 será pressionado.

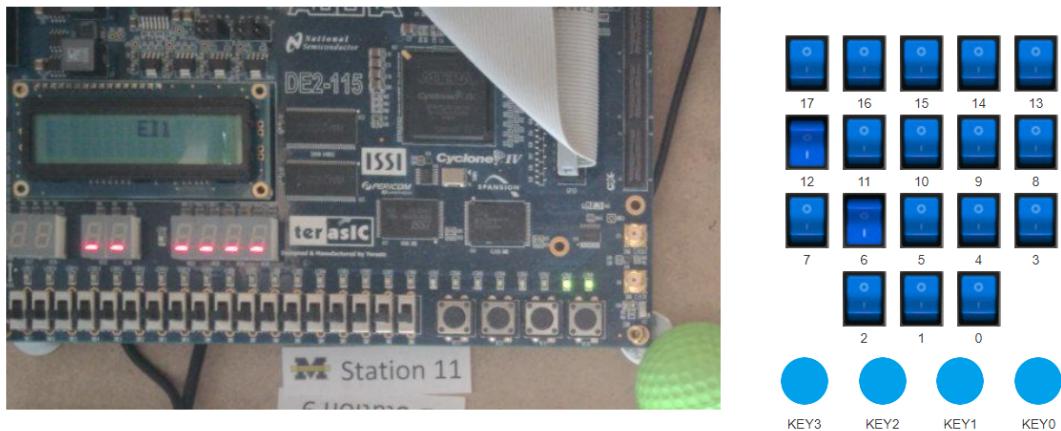


Figura 14. Tentativa realizada : '6'; uma vida é perdida.

O palpite estava errado: o dígito 6 não faz parte da palavra secreta. Nesse caso, perde-se uma vida, restando apenas 2 LEDs acesos. Como as vidas não chegaram a zero, o jogador pode continuar as tentativas. Pressiona-se, então, o switch 4.

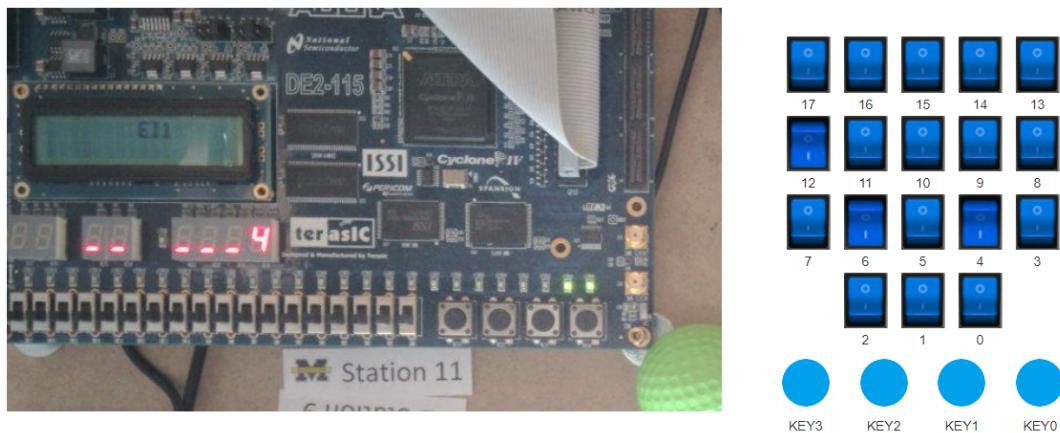


Figura 15. Tentativa realizada : '4' ; o '4' aparece no display

O palpite estava correto: o dígito 4 faz parte da palavra secreta. Nesse caso, as duas vidas restantes são mantidas, e o jogador pode tentar adivinhar novamente os outros dígitos escondidos. Pressiona-se, então, o switch 0.

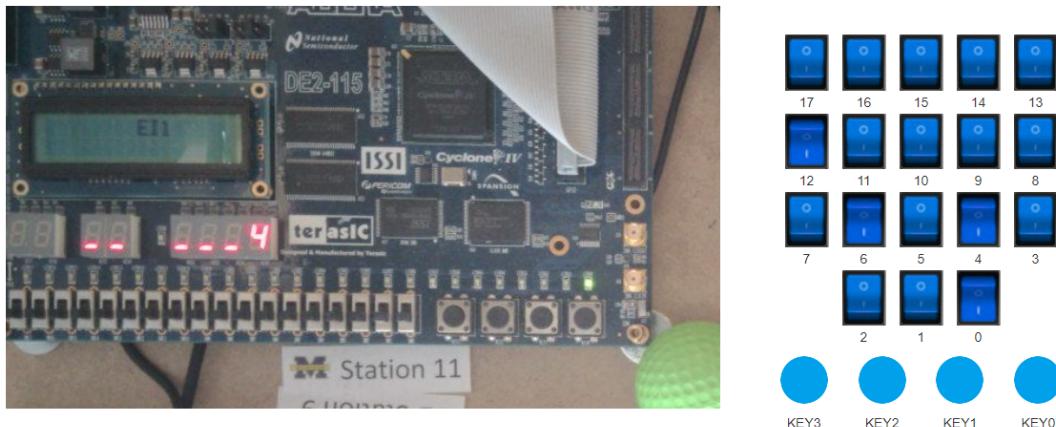


Figura 16. Tentativa realizada : '0'; uma vida é perdida.

O palpite estava errado: o dígito 0 não faz parte da palavra secreta. Nesse caso, perde-se outra vida, restando apenas 1 LED aceso. Como as vidas não chegaram a zero, o jogador pode continuar as tentativas. Pressiona-se, então, o switch 8.

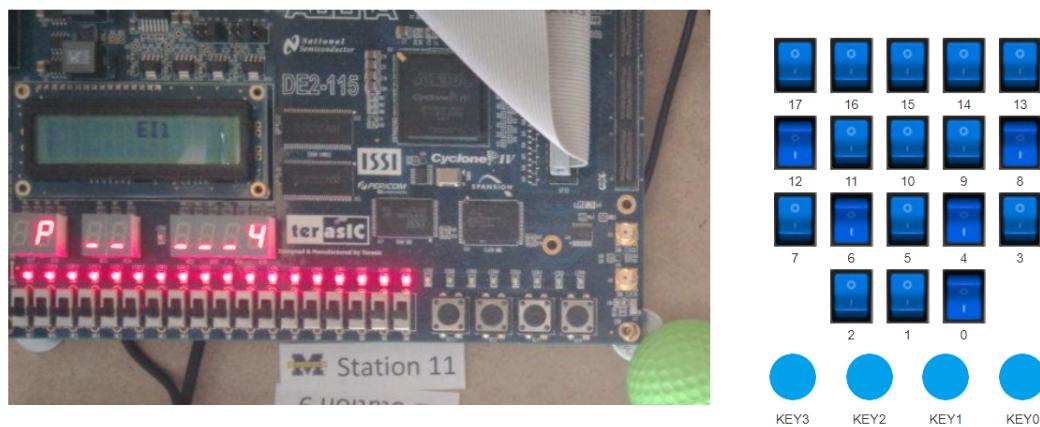


Figura 17. Tentativa realizada : '8'; A última vida é perdida, o display exibe 'P', indicando que o jogador perdeu. Os LEDs vermelhos se acendem.

O palpite estava errado: o dígito 8 não faz parte da palavra secreta. Nesse caso, o jogador perdeu a última vida, não conseguiu identificar todos os dígitos da palavra secreta e perdeu o jogo. Sendo assim, mostra-se P no sétimo display, contando da direita para a esquerda.

Caso o jogador queira jogar novamente, é necessário desativar todos os switches, apertar o botão KEY0 para reiniciar o jogo e, por fim, escolher novamente a dificuldade ao ligar os switches 11 ou 12.

6. Conclusão

O projeto da força numérica foi muito interessante de ser desenvolvido como uma aplicação prática da Eletrônica Digital em um jogo popular. Notou-se que a lógica de uma máquina de estados é muito mais fácil de ser implementada, em comparação a um circuito totalmente combinacional. Trata-se de um projeto muito mais simples que o primeiro, possuindo menos módulos e funções requeridas³.

Ainda assim, o desenvolvimento desse projeto foi de grande valia para aplicar conhecimentos sobre máquinas de estados em VHDL, indispensáveis no estudo da disciplina de Sistemas Digitais.

7. Referências Bibliográficas

Volnei A. Pedroni, *Eletrônica Digital Moderna e VHDL*. Capítulo 23: Projeto de Máquinas de Estados, págs. 526 a 547. Rio de Janeiro: Elsevier, 1^a edição, 2010.

³O módulo do jogo da força numérico em VHDL está disponível no GitHub *por meio deste link*.