Universidade Federal do Rio de Janeiro

Universidade Federal
do Rio de Janeiro
Escola Politécnica

Relatório: Jogo da Forca em FPGA

Alunos	Erik Branco Queiroz
	Paulo Vitor Couto Doederlein
	Arthur Freitas Ramos
Professor	Pedro Cruz
Horário	Ter - 13:00-15:00

Rio de Janeiro, dezembro de 2025

Conteúdo

1 Objetivo

O objetivo deste trabalho prático é projetar e implementar um sistema digital na FPGA que integre o uso de periféricos de entrada e saída, especificamente um teclado PS/2 e um display LCD, para criar um "Jogo da Forca" funcional. O sistema deve ser capaz de processar entradas do usuário, manter o estado do jogo (vidas, acertos, erros) e exibir as informações pertinentes visualmente.

2 Introdução

Este relatório descreve o desenvolvimento de um jogo da forca utilizando a linguagem de descrição de hardware VHDL. O projeto é composto por módulos que realizam a interface de baixo nível com o teclado, a decodificação de teclas pressionadas e a lógica central de controle do jogo que gerencia o display LCD.

A lógica do jogo consiste em permitir que o usuário selecione uma palavra secreta de um banco de dados pré-definido através de chaves (switches) da placa, e tente adivinhar a palavra digitando letras no teclado. O sistema oferece feedback visual imediato sobre letras corretas, letras erradas e quantidade de tentativas restantes (vidas).

3 Descrição do Projeto

O sistema foi dividido em blocos funcionais hierárquicos para facilitar o desenvolvimento e a depuração. Abaixo descreve-se o funcionamento de cada módulo principal.

3.1 Interface com Teclado (Entrada)

A captura e processamento dos dados do teclado foram divididos em três entidades principais:

- **PS2_RX**: Este módulo é responsável pela camada física da comunicação. Ele recebe os sinais elétricos brutos `ps2c` (clock) e `ps2d` (dados) vindos da porta PS/2. O módulo realiza a serialização dos bits recebidos para montar um vetor de 8 bits (`dout`) contendo o *scancode* da tecla, além de gerar um sinal de `rx_done_tick` quando um dado completo é recebido .

- **KB_CODE**: Atua como um controlador intermediário. Ele instancia o `PS2_RX` e utiliza um buffer (FIFO) para armazenar os códigos recebidos. Sua função principal é filtrar o código de interrupção (`F0` ou *break code*), garantindo que a tecla seja registrada apenas no evento correto de soltura ou pressão, evitando leituras duplicadas indesejadas.

- **KEY2ASCII**: Este módulo funciona como uma tabela de consulta (Look-Up Table - ROM). Ele recebe o *scancode* bruto do teclado (vetor de 8 bits) e o mapeia para o seu correspondente na tabela ASCII padrão. Por exemplo, converte o código hexadecimal `1C` para a letra 'A' (`01000001` em binário) e `1B` para 'S', permitindo que a lógica do jogo trabalhe com caracteres compreensíveis .

3.2 Controlador do Jogo e LCD (LcdProposta)

O módulo `lcd` (implementado no arquivo `LcdProposta2.vhd`) é a entidade de topo (*top-level*) que "comanda tudo". Ele integra a leitura do teclado, a lógica do jogo e a atualização do display LCD.

3.2.1 Banco de Palavras e Seleção

O jogo possui um banco de palavras armazenado internamente. O usuário utiliza os switches da placa (`sw`) para selecionar um código binário de 0 a 9. Cada código corresponde a uma palavra secreta pré-definida, como "UFRJ", "FPGA", "VHDL", "DIGITAIS", entre outras .

3.2.2 Lógica de Execução

O funcionamento do jogo segue a seguinte lógica:

1. **Estado Inicial:** O display solicita que o usuário escolha uma palavra (exibindo o número selecionado nos switches) e pressione ESPAÇO para iniciar.

Figura 1: Funcionamento do Jogo da Forca no Display LCD da FPGA

2. **Durante o Jogo:**
   - O display mostra, na primeira linha, a palavra secreta oculta por *underscores* (_). As letras são reveladas conforme o usuário acerta .
   - Na segunda linha, é exibido o contador de vidas (inicializado em 6) e as letras erradas que já foram digitadas.
   - Ao pressionar uma tecla, o sistema varre a palavra secreta. Se a letra existir, o vetor `certo` é atualizado. Se não existir, o contador de vidas é decrementado e a letra é adicionada ao vetor de erros .

Figura 2: Funcionamento durante o jogo no Display

3. **Condições de Fim de Jogo:**

   - **Vitória:** Se todas as letras da palavra forem descobertas antes das vidas acabarem, o sistema exibe "VOCE GANHOU".

   - **Derrota:** Se o contador de vidas chegar a zero, o sistema exibe "VOCE PERDEU".

3.2.3 Controle do Display LCD

O módulo implementa uma máquina de estados finitos (FSM) para controlar a inicialização e escrita no LCD. Os estados incluem `stPowerOn_Delay`, `stFunctionSet`, `stDisplayControl`, entre outros, respeitando os tempos de atraso (*delays*) exigidos pelo hardware do display.

4 Pinagem e Hardware

A implementação física na FPGA Spartan-3A foi definida pelo arquivo de restrições (`.ucf`).

5 Diagrama de Blocos

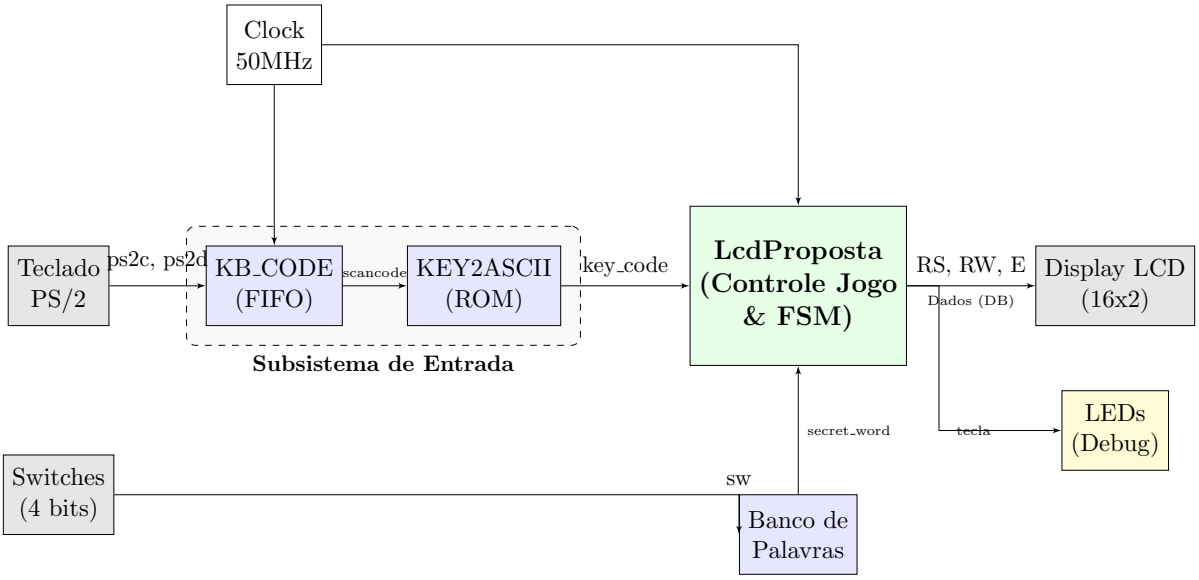

Figura 3: Diagrama de Blocos do Sistema Jogo da Forca

6 Diagrama de Estados

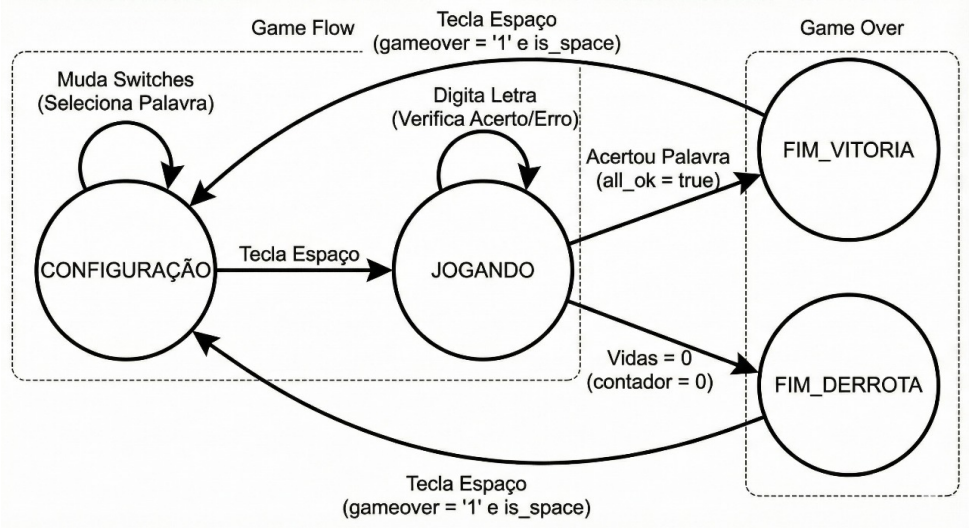

Figura 4: Diagrama de Estados (FSM) do Sistema Jogo da Forca

7 Conclusão

O projeto permitiu a aplicação prática de conceitos fundamentais de sistemas digitais, como máquinas de estados, serialização de dados e integração de módulos. O sistema desenvolvido atendeu aos requisitos de diferenciar as teclas pressionadas, exibir a saída no display LCD e proporcionar interatividade completa com o usuário através da lógica do jogo da forca..

8 Código

A seguir são apresentados os códigos VHDL desenvolvidos e o arquivo de restrições UCF utilizado no projeto.

8.1 PS2_RX.vhd

Responsável pela recepção serial dos dados do teclado.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ps2_rx is
port (
clk, reset: in std_logic;
ps2d, ps2c: in std_logic; -- key data, key clock
rx_en: in std_logic;
rx_done_tick: out std_logic;
dout: out std_logic_vector(7 downto 0)
);
end ps2_rx;

architecture arch of ps2_rx is
type statetype is (idle, dps, load);
signal state_reg, state_next: statetype;
signal filter_reg, filter_next:
std_logic_vector(7 downto 0);
signal f_ps2c_reg,f_ps2c_next: std_logic;
signal b_reg, b_next: std_logic_vector(10 downto 0);
signal n_reg,n_next: unsigned(3 downto 0);
signal fall_edge: std_logic;

begin

process (clk, reset)
begin
if reset='1' then
filter_reg <= (others=>'0');
f_ps2c_reg <= '0';
elsif (clk'event and clk='1') then
filter_reg <= filter_next;
f_ps2c_reg <= f_ps2c_next;
end if;
end process;

filter_next <= ps2c & filter_reg(7 downto 1);
f_ps2c_next <= '1' when filter_reg="11111111" else
'0' when filter_reg="00000000" else
f_ps2c_reg;
fall_edge <= f_ps2c_reg and (not f_ps2c_next);


-- registers
process (clk, reset)
begin
if reset='1' then
state_reg <= idle;
n_reg <= (others=>'0');
b_reg <= (others=>'0');
elsif (clk'event and clk='1') then
state_reg <= state_next;
n_reg <= n_next;
b_reg <= b_next;
end if;
end process;
-- next-state logic
process(state_reg,n_reg,b_reg,fall_edge,rx_en,ps2d)
begin
rx_done_tick <='0';
state_next <= state_reg;
```

```
n_next <= n_reg;
b_next <= b_reg;
case state_reg is
when idle =>
if fall_edge='1' and rx_en='1' then
-- shift in start bit
b_next <= ps2d & b_reg(10 downto 1);
n_next <= "1001";
state_next <= dps;
end if;
when dps => -- 8 data + 1 pairty + 1 stop
if fall_edge='1' then
b_next <= ps2d & b_reg(10 downto 1);
if n_reg = 0 then
state_next <=load;
else
n_next <= n_reg - 1;
end if;
end if;
when load =>
-- 1 extra clock to complete the last shift
state_next <= idle;
rx_done_tick <='1';
end case;
end process;
-- output

dout <= b_reg(8 downto 1); -- data bits
end arch;
```

8.2 KB_CODE.vhd

Controlador que gerencia a FIFO e filtra os códigos de break.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity kb_code is
generic(W_SIZE: integer:=2); -- 2^W_SIZE words in FIFO
port (
clk, reset: in std_logic;
ps2d, ps2c: in std_logic;
rd_key_code: in std_logic;
key_code: out std_logic_vector(7 downto 0);
kb_buf_empty: out std_logic
);
end kb_code;

architecture arch of kb_code is
constant BRK: std_logic_vector(7 downto 0):="11110000";
-- F0 (break code)
type statetype is (wait_brk, get_code);
signal state_reg, state_next: statetype;
signal scan_out, w_data: std_logic_vector(7 downto 0);
signal scan_done_tick, got_code_tick: std_logic;
signal ascii_code, key_code_2: std_logic_vector(7 downto 0);

begin


ps2_rx_unit: entity work.ps2_rx(arch)
port map(clk=>clk, reset=>reset, rx_en=>'1',
ps2d=>ps2d, ps2c=>ps2c,
rx_done_tick=>scan_done_tick,
dout=>scan_out);

fifo_key_unit: entity work.fifo(arch)
generic map(B=>8)
port map(clk=>clk, reset=>reset, rd=>rd_key_code,
wr=>got_code_tick, w_data=>scan_out,
empty=>kb_buf_empty, full=>open,
```

```
r_data=>key_code_2);

key2ascii_unit: entity work.key2ascii(arch)
port map (key_code=>key_code_2,
ascii_code=>ascii_code);


process (clk, reset)
begin
if reset='1' then
state_reg <= wait_brk;
elsif (clk'event and clk='1') then
state_reg <= state_next;
end if;
end process;

process(state_reg, scan_done_tick, scan_out)
begin
got_code_tick <='0';
state_next <= state_reg;
case state_reg is
when wait_brk => -- wait for F0 of break code
if scan_done_tick='1' and scan_out=BRK then
state_next <= get_code;
end if;
when get_code => -- get the following scan code
if scan_done_tick='1' then
got_code_tick <='1';
state_next <= wait_brk;
end if;
end case;
key_code <= ascii_code;
end process;
end arch;
```

8.3 KEY2ASCII.vhd

Converte scancodes PS/2 para ASCII.

```
-- Listing 8.4
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity key2ascii is
port (
key_code: in std_logic_vector(7 downto 0);
ascii_code: out std_logic_vector(7 downto 0)
);
end key2ascii;

architecture arch of key2ascii is
begin
with key_code select
ascii_code <=
"00110000" when "01000101", -- 0
"00110001" when "00010110", -- 1
"00110010" when "00011110", -- 2
"00110011" when "00100110", -- 3
"00110100" when "00100101", -- 4
"00110101" when "00101110", -- 5
"00110110" when "00110110", -- 6
"00110111" when "00111101", -- 7
"00111000" when "00111110", -- 8
"00111001" when "01000110", -- 9

"01000001" when "00011100", -- A
"01000010" when "00110010", -- B
"01000011" when "00100001", -- C
"01000100" when "00100011", -- D
"01000101" when "00100100", -- E
"01000110" when "00101011", -- F
```

```
"01000111" when "00110100", -- G
"01001000" when "00110011", -- H
"01001001" when "01000011", -- I
"01001010" when "00111011", -- J
"01001011" when "01000010", -- K
"01001100" when "01001011", -- L
"01001101" when "00111010", -- M
"01001110" when "00110001", -- N
"01001111" when "01000100", -- O
"01010000" when "01001101", -- P
"01010001" when "00010101", -- Q
"01010010" when "00101101", -- R
"01010011" when "00011011", -- S
"01010100" when "00101100", -- T
"01010101" when "00111100", -- U
"01010110" when "00101010", -- V
"01010111" when "00011101", -- W
"01011000" when "00100010", -- X
"01011001" when "00110101", -- Y
"01011010" when "00011010", -- Z

"01100000" when "00001110", -- `
"00101101" when "01001110", -- -
"00111101" when "01010101", -- =
"01011011" when "01010100", -- [

"01011101" when "01011011", -- ]
"01011100" when "01011101", -- \
"00111011" when "01001100", -- ;
"00100111" when "01010010", -- '
"00101100" when "01000001", -- ,
"00101110" when "01001001", -- .
"00101111" when "01001010", -- /

"00100000" when "00101001", -- (space)
"00001101" when "01011010", -- (enter, cr)
"00001000" when "01100110", -- (backspace)
"00101010" when others;
end arch;
```

8.4 LcdProposta2.vhd

Módulo principal que controla o jogo e o display LCD.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity lcd is
  generic (
    CLK_HZ : integer := 50_000_000
  );
  port (
    LCD_DB : out std_logic_vector(7 downto 0);
    RS     : out std_logic;
    RW     : out std_logic;
    E      : out std_logic;
    CLK    : in  std_logic;
    OE     : out std_logic;
    rst    : in  std_logic;
    ps2d   : in  std_logic;
    ps2c   : in  std_logic;
    sw     : in  std_logic_vector(3 downto 0);
    tecla  : out std_logic_vector(7 downto 0)
  );
end lcd;

architecture Behavioral of lcd is

  component kb_code
    generic (W_SIZE: integer := 2);
```

```
    port (
      clk, reset   : in  std_logic;
      ps2d, ps2c   : in  std_logic;
      rd_key_code  : in  std_logic;
      key_code     : out std_logic_vector(7 downto 0);
      kb_buf_empty : out std_logic
    );
  end component;

  type mstate is (
    stFunctionSet, stDisplayCtrlSet, stDisplayClear,
    stPowerOn_Delay, stFunctionSet_Delay, stDisplayCtrlSet_Delay,
    stDisplayClear_Delay, stInitDne, stActWr, stCharDelay
  );
  type wstate is ( stRW, stEnable, stIdle );

  constant TICK_US : integer := integer(CLK_HZ / 1_000_000);
  signal   us_cnt  : integer range 0 to TICK_US-1 := 0;
  signal   oneUSClk : std_logic := '0';

  constant T_PWRON_US  : integer := 20_000;
  constant T_FUNSET_US : integer := 37;
  constant T_DCTRL_US  : integer := 37;
  constant T_CLEAR_US  : integer := 1_520;
  constant T_CHAR_US   : integer := 50;

  signal count     : integer := 0;
  signal delayOK   : std_logic := '0';
  signal stCur     : mstate := stPowerOn_Delay;
  signal stNext    : mstate := stPowerOn_Delay;
  signal stCurW    : wstate := stIdle;
  signal stNextW   : wstate := stIdle;
  signal activateW : std_logic := '0';
  signal writeDone : std_logic := '0';
  signal E_reg     : std_logic := '0';

  signal rd_key_code   : std_logic := '0';
  signal key_read      : std_logic_vector(7 downto 0) := (others=>'0');
  signal kb_empty      : std_logic := '1';
  signal kb_empty_prev : std_logic := '1';

  type word_t is array (0 to 9) of std_logic_vector(7 downto 0);
  signal secret_word : word_t := (others => x"20");
  signal secret_len  : integer range 0 to 10 := 0;
  signal certo       : std_logic_vector(9 downto 0) := (others=>'0');

  type errors_t is array (0 to 5) of std_logic_vector(7 downto 0);
  signal letras_erradas : errors_t := (others => x"20");
  signal indice_erro    : integer range 0 to 6 := 0;

  signal contador : integer range 0 to 6 := 6;
  signal venceu   : std_logic := '0';
  signal perdeu   : std_logic := '0';
  signal gameover : std_logic := '0';
  signal started  : std_logic := '0';

  type LCD_CMDS_T is array(integer range 0 to 39) of std_logic_vector(9 downto 0);
  signal LCD_CMDS : LCD_CMDS_T;
  signal lcd_cmd_ptr : integer range 0 to LCD_CMDS'high + 1 := 0;

begin

  E  <= E_reg;
  OE <= '1';

  leitura: kb_code
    generic map (W_SIZE => 2)
    port map (
      clk => CLK, reset => rst, ps2d => ps2d, ps2c => ps2c,
      rd_key_code => rd_key_code, key_code => key_read, kb_buf_empty => kb_empty
    );

  process(CLK, rst)
```

```
  begin
    if rst='1' then us_cnt <= 0; oneUSClk <= '0';
    elsif rising_edge(CLK) then
      if us_cnt = TICK_US-1 then us_cnt <= 0; oneUSClk <= '1';
      else us_cnt <= us_cnt + 1; oneUSClk <= '0';
      end if;
    end if;
  end process;

  process(secret_len, certo, contador)
    variable all_ok : boolean;
  begin
    all_ok := true;
    if secret_len > 0 then
        for i in 0 to 9 loop
            if i < secret_len then
                if certo(i) = '0' then all_ok := false; end if;
            end if;
        end loop;
    else
        all_ok := false;
    end if;

    if all_ok then venceu <= '1'; else venceu <= '0'; end if;
    if contador = 0 then perdeu <= '1'; else perdeu <= '0'; end if;
  end process;

  gameover <= venceu or perdeu;

  -- ------------------------------------------------------------------------
  -- BANCO DE PALAVRAS (ROM) - CORRIGIDO COM VARI VEIS
  -- ------------------------------------------------------------------------
  process(sw)
    variable v_word : word_t;
    variable v_len  : integer range 0 to 10;
  begin
    -- Inicializa variaveis locais
    v_word := (others => x"20");
    v_len := 0;

    case sw is
        when "0000" => -- UFRJ
            v_word(0):=x"55"; v_word(1):=x"46"; v_word(2):=x"52"; v_word(3):=x"4A";
            v_len := 4;
        when "0001" => -- FPGA
            v_word(0):=x"46"; v_word(1):=x"50"; v_word(2):=x"47"; v_word(3):=x"41";
            v_len := 4;
        when "0010" => -- VHDL
            v_word(0):=x"56"; v_word(1):=x"48"; v_word(2):=x"44"; v_word(3):=x"4C";
            v_len := 4;
        when "0011" => -- DIGITAIS
            v_word(0):=x"44"; v_word(1):=x"49"; v_word(2):=x"47"; v_word(3):=x"49";
            v_word(4):=x"54"; v_word(5):=x"41"; v_word(6):=x"49"; v_word(7):=x"53";
            v_len := 8;
        when "0100" => -- SPARTAN
            v_word(0):=x"53"; v_word(1):=x"50"; v_word(2):=x"41"; v_word(3):=x"52";
            v_word(4):=x"54"; v_word(5):=x"41"; v_word(6):=x"4E";
            v_len := 7;
        when "0101" => -- XILINX
            v_word(0):=x"58"; v_word(1):=x"49"; v_word(2):=x"4C"; v_word(3):=x"49";
            v_word(4):=x"4E"; v_word(5):=x"58";
            v_len := 6;
        when "0110" => -- PROJETO
            v_word(0):=x"50"; v_word(1):=x"52"; v_word(2):=x"4F"; v_word(3):=x"4A";
            v_word(4):=x"45"; v_word(5):=x"54"; v_word(6):=x"4F";
            v_len := 7;
        when "0111" => -- LOGICA
            v_word(0):=x"4C"; v_word(1):=x"4F"; v_word(2):=x"47"; v_word(3):=x"49";
            v_word(4):=x"43"; v_word(5):=x"41";
            v_len := 6;
        when "1000" => -- ESTADOS
            v_word(0):=x"45"; v_word(1):=x"53"; v_word(2):=x"54"; v_word(3):=x"41";
            v_word(4):=x"44"; v_word(5):=x"4F"; v_word(6):=x"53";
```

```
            v_len := 7;
        when others => -- PROCESS
            v_word(0):=x"50"; v_word(1):=x"52"; v_word(2):=x"4F"; v_word(3):=x"43";
            v_word(4):=x"45"; v_word(5):=x"53"; v_word(6):=x"53";
            v_len := 7;
    end case;

    -- Atribui ao sinal apenas no final
    secret_word <= v_word;
    secret_len <= v_len;
  end process;

  -- ------------------------------------------------------------------------
  -- ATUALIZA  O DO LCD
  -- ------------------------------------------------------------------------
  process(started, venceu, perdeu, contador, secret_word, secret_len, certo,
    letras_erradas, sw)
    variable L1 : word_t;
  begin
    -- Inicializa buffer com espaços limpos
    L1 := (others => x"20");

    -- Comandos Fixos
    LCD_CMDS(0) <= "00"&x"38"; LCD_CMDS(1) <= "00"&x"0C";
    LCD_CMDS(2) <= "00"&x"01"; LCD_CMDS(3) <= "00"&x"80";

    -- Lógica LINHA 1
    if started = '0' then
        -- "ESCOLHA: X"
        L1(0):=x"45"; L1(1):=x"53"; L1(2):=x"43"; L1(3):=x"4F";
        L1(4):=x"4C"; L1(5):=x"48"; L1(6):=x"41"; L1(7):=x"3A"; L1(8):=x"20";
        if to_integer(unsigned(sw)) < 10 then
            L1(9) := std_logic_vector(to_unsigned(48 + to_integer(unsigned(sw)), 8));
        else
            L1(9) := x"3F";
        end if;

    elsif venceu = '1' then
        -- "VOCE GANHOU"
        L1(0):=x"56"; L1(1):=x"4F"; L1(2):=x"43"; L1(3):=x"45"; L1(4):=x"20";
        L1(5):=x"47"; L1(6):=x"41"; L1(7):=x"4E"; L1(8):=x"48"; L1(9):=x"4F"; L1(10) := x
    "55";
    elsif perdeu = '1' then
        -- "VOCE PERDEU"
        L1(0):=x"56"; L1(1):=x"4F"; L1(2):=x"43"; L1(3):=x"45"; L1(4):=x"20";
        L1(5):=x"50"; L1(6):=x"45"; L1(7):=x"52"; L1(8):=x"44"; L1(9):=x"45"; L1(10) := x
    "55";
    else
        -- JOGO: Palavra oculta ou revelada
        for i in 0 to 9 loop
            if i < secret_len then
                if certo(i) = '1' then L1(i) := secret_word(i);
                else L1(i) := x"5F"; -- Underline
                end if;
            else
                L1(i) := x"20"; -- Espaço para o resto
            end if;
        end loop;
    end if;

    -- Transfere L1 para LCD_CMDS
    for i in 0 to 9 loop LCD_CMDS(4+i) <= "10" & L1(i); end loop;
    for i in 10 to 15 loop LCD_CMDS(4+i) <= "10" & x"20"; end loop;

    -- Pula para Linha 2
    LCD_CMDS(20) <= "00" & x"C0";

    -- Lógica LINHA 2
    if started = '0' then
        -- "PRESS ESPACO"
        LCD_CMDS(21) <= "10"&x"50"; LCD_CMDS(22) <= "10"&x"52"; LCD_CMDS(23) <= "10"&x
    "45";
```

```
242         LCD_CMDS(24) <= "10"&x"53"; LCD_CMDS(25) <= "10"&x"53"; LCD_CMDS(26) <= "10"&x
    "20";
243         LCD_CMDS(27) <= "10"&x"53"; LCD_CMDS(28) <= "10"&x"50"; LCD_CMDS(29) <= "10"&x
    "41";
244         LCD_CMDS(30) <= "10"&x"43"; LCD_CMDS(31) <= "10"&x"45";
245         for i in 32 to 36 loop LCD_CMDS(i) <= "10"&x"20"; end loop;
246       else
247         -- Status: V:6 Er:ABC...
248         LCD_CMDS(21) <= "10"&x"56"; -- V
249         LCD_CMDS(22) <= "10"&x"3A"; -- :
250         LCD_CMDS(23) <= "10" & std_logic_vector(to_unsigned(48 + contador, 8)); -- Num
251         LCD_CMDS(24) <= "10"&x"20";
252         LCD_CMDS(25) <= "10"&x"45"; -- E
253         LCD_CMDS(26) <= "10"&x"72"; -- r
254         LCD_CMDS(27) <= "10"&x"3A"; -- :
255         for i in 0 to 5 loop
256             LCD_CMDS(28+i) <= "10" & letras_erradas(i);
257         end loop;
258         LCD_CMDS(34) <= "10"&x"20"; LCD_CMDS(35) <= "10"&x"20"; LCD_CMDS(36) <= "10"&x
    "20";
259       end if;
260     end process;
261
262     process (CLK, rst)
263       variable is_space    : boolean;
264       variable is_letter   : boolean;
265       variable ascii       : std_logic_vector(7 downto 0);
266       variable match_found : boolean;
267       variable ja_errou    : boolean;
268     begin
269       if rst='1' then
270         rd_key_code    <= '0';
271         tecla          <= (others=>'0');
272         kb_empty_prev  <= '1';
273         started        <= '0';
274         contador       <= 6;
275         certo          <= (others=>'0');
276         letras_erradas <= (others => x"20");
277         indice_erro    <= 0;
278
279       elsif rising_edge(CLK) then
280         rd_key_code <= '0';
281         kb_empty_prev <= kb_empty;
282
283         if (kb_empty_prev = '1') and (kb_empty = '0') then
284           rd_key_code <= '1';
285           tecla       <= key_read;
286           ascii       := key_read;
287
288           if ascii = x"20" or ascii = x"29" then is_space := true; else is_space := false;
    end if;
289           is_letter := (ascii >= x"41" and ascii <= x"5A");
290
291           if gameover = '1' then
292             if is_space then
293               started        <= '0';
294               contador       <= 6;
295               certo          <= (others=>'0');
296               letras_erradas <= (others => x"20");
297               indice_erro    <= 0;
298             end if;
299
300           elsif started = '0' then
301             if is_space then started <= '1'; end if;
302
303           else
304             if is_letter then
305               match_found := false;
306               for i in 0 to 9 loop
307                 if i < secret_len then
308                   if ascii = secret_word(i) then
309                     match_found := true;
310                     certo(i)    <= '1';
```

```
                end if;
              end if;
            end loop;

            if not match_found then
                ja_errou := false;
                for k in 0 to 5 loop
                    if letras_erradas(k) = ascii then ja_errou := true; end if;
                end loop;
                if (not ja_errou) and (contador > 0) then
                    contador <= contador - 1;
                    if indice_erro < 6 then
                        letras_erradas(indice_erro) <= ascii;
                        indice_erro <= indice_erro + 1;
                    end if;
                end if;
            end if;
          end if;
        end if;
      end if;
    end process;

    writeDone <= '1' when (lcd_cmd_ptr > 36) else '0';

    process (CLK, rst)
    begin
      if rst='1' then
        lcd_cmd_ptr <= 0;
      elsif rising_edge(CLK) then
        if oneUSClk='1' then
          if ((stNext = stInitDne or stNext = stDisplayCtrlSet or stNext = stDisplayClear)
      and writeDone='0') then
            lcd_cmd_ptr <= lcd_cmd_ptr + 1;
          elsif (stCur = stPowerOn_Delay) then
            lcd_cmd_ptr <= 0;
          elsif (writeDone='1') then
            lcd_cmd_ptr <= 3;
          end if;
        end if;
      end if;
    end process;

    process (CLK, rst)
    begin
      if rst='1' then stCur <= stPowerOn_Delay;
      elsif rising_edge(CLK) then
        if oneUSClk='1' then stCur <= stNext; end if;
      end if;
    end process;

    process (stCur, delayOK, writeDone, lcd_cmd_ptr, LCD_CMDS)
    begin
      if lcd_cmd_ptr <= 39 then
          RS <= LCD_CMDS(lcd_cmd_ptr)(9);
          RW <= LCD_CMDS(lcd_cmd_ptr)(8);
          LCD_DB <= LCD_CMDS(lcd_cmd_ptr)(7 downto 0);
      else
          RS<='0'; RW<='0'; LCD_DB<=x"00";
      end if;

      activateW <= '0';
      stNext <= stCur;

      case stCur is
        when stPowerOn_Delay =>
          if delayOK='1' then stNext <= stFunctionSet; else stNext <= stPowerOn_Delay; end
      if;
        when stFunctionSet =>
          activateW <= '1'; stNext <= stFunctionSet_Delay;
        when stFunctionSet_Delay =>
          if delayOK='1' then stNext <= stDisplayCtrlSet; else stNext <=
      stFunctionSet_Delay; end if;
```

```
      when stDisplayCtrlSet =>
        activateW <= '1'; stNext <= stDisplayCtrlSet_Delay;
      when stDisplayCtrlSet_Delay =>
        if delayOK='1' then stNext <= stDisplayClear; else stNext <=
    stDisplayCtrlSet_Delay; end if;
      when stDisplayClear =>
        activateW <= '1'; stNext <= stDisplayClear_Delay;
      when stDisplayClear_Delay =>
        if delayOK='1' then stNext <= stInitDne; else stNext <= stDisplayClear_Delay; end
     if;
      when stInitDne =>
        stNext <= stActWr;
      when stActWr =>
        activateW <= '1'; stNext <= stCharDelay;
      when stCharDelay =>
        if delayOK='1' then stNext <= stInitDne; else stNext <= stCharDelay; end if;
    end case;
  end process;

  process (CLK, rst)
  begin
    if rst='1' then count <= 0; delayOK <= '0';
    elsif rising_edge(CLK) then
      if oneUSClk='1' then
        if ((stCur = stPowerOn_Delay and count >= T_PWRON_US) or
            (stCur = stFunctionSet_Delay and count >= T_FUNSET_US) or
            (stCur = stDisplayCtrlSet_Delay and count >= T_DCTRL_US) or
            (stCur = stDisplayClear_Delay and count >= T_CLEAR_US) or
            (stCur = stCharDelay and count >= T_CHAR_US)) then
          delayOK <= '1'; count <= 0;
        else
          delayOK <= '0'; count <= count + 1;
        end if;
      end if;
    end if;
  end process;

  process (CLK, rst)
  begin
    if rst='1' then stCurW <= stIdle; E_reg <= '0';
    elsif rising_edge(CLK) then
      if oneUSClk='1' then
        stCurW <= stNextW;
        case stCurW is
          when stIdle => E_reg <= '0';
          when stRw   => E_reg <= '0';
          when stEnable => E_reg <= '1';
        end case;
      end if;
    end if;
  end process;

  process (stCurW, activateW)
  begin
    stNextW <= stCurW;
    case stCurW is
      when stIdle => if activateW='1' then stNextW <= stRw; end if;
      when stRw => stNextW <= stEnable;
      when stEnable => stNextW <= stIdle;
    end case;
  end process;

end Behavioral;
```

8.5 PinagemProposta.ucf

Arquivo de restrições de pinagem para a placa Spartan-3A.