Universidade Federal
do Rio de Janeiro
Escola Politécnica

EEL480 - Laboratório de Sistemas Digitais
Prof. Luís Henrique Maciel

Hangman Game em VHDL

Christian Marques de Oliveira Silva
DRE: 117.214.742

Rio de Janeiro
2021

Sumário

1 Introdução

O objetivo deste projeto foi criar um jogo da forca (Hangman Game) descrito em VHDL[1]. A diferença é que ao invés de letras, serão utilizados número de 0 a 15. Ainda, caso o jogador erre 3 letras, o mesmo perde o jogo.

Figura 1: Diagrama lógico do jogo

Como ilustrado na imagem acima, existem algumas entradas e saídas do jogo que é importante pontuar:

- **Botões de Letras**: São os 16 botões que representam as "palavras" que o jogador tentará adivinhar. O botão fica pressionado, então não permite esquecer o que já foi inserido anteriormente;
- **Letras Ocultas**: É a "palavra" que o jogador tentará adivinhar. O que ficou configurado no código foi "FOCADA";
- **Reinício**: É o botão que é usado para iniciar o jogo, sendo importante no momento em que inicia a primeira partida e, também, quando o jogador perde ou ganha;
- **Vida Restante**: Indica a quantidade de vidas que o jogador ainda tem. Ao todo são 3, logo, essa é a quantidade de LEDs de sinalização. Quando todos se apagarem o jogador perdeu;
- **Clock**: É um sinal interno para sincronizar os processos e possui frequência de 50MHz;

2 Desenvolvimento

2.1 Estados

Para o jogo realizar suas funções, foi necessária a criação dos seguintes estados: *Waiting, Char0, Char1, Char2, Char3, Char4, Win, Lose*. Cada estado "CharX" guarda

[1]Very High Speed Integrated Circuit

a entrada correta de letras pelo usuário. Deste modo, como a palavra oculta é “FOCADA”, o 5º caractere certo equivale à vitória (já que a letra “A” se repete). O estado “Waiting” é basicamente o estado inicial e o “Win” e “Lose” representam os estados finais de vitória e derrota, respectivamente.

Abaixo segue o diagrama de estados implementado no jogo.

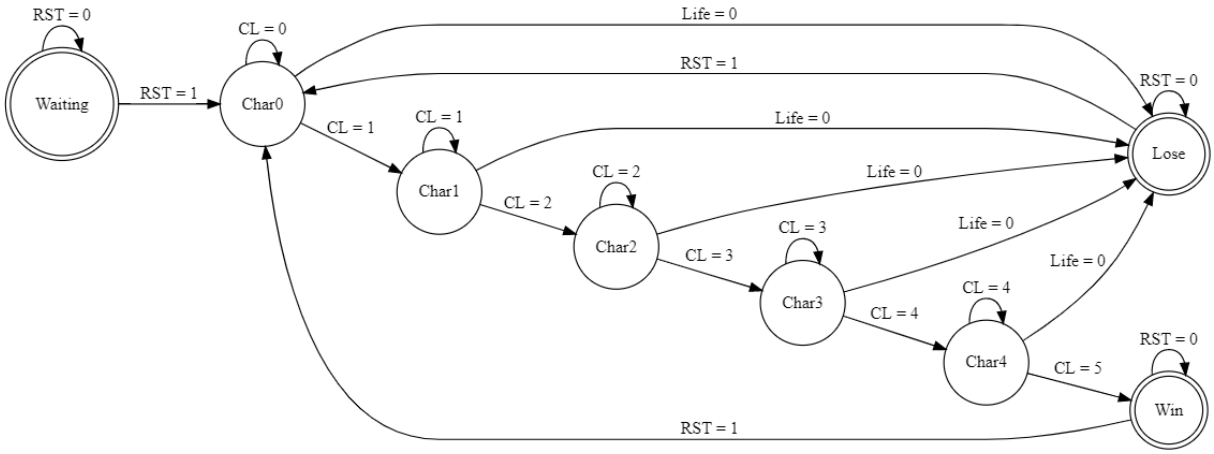


Figura 2: Diagrama de estados

2.2 Entradas e Saídas

Abaixo segue a forma encontrada para representar as entradas do jogo situadas no LabsLand: botões de cada letra e o botão de reinício/start.

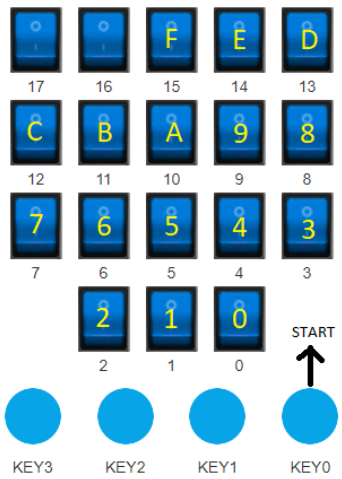


Figura 3: Indicação de entradas do jogo

Abaixo segue a forma encontrada para representar as saídas do jogo no LabsLand: Palavra Oculta, Status do Jogo e Vidas Restantes.

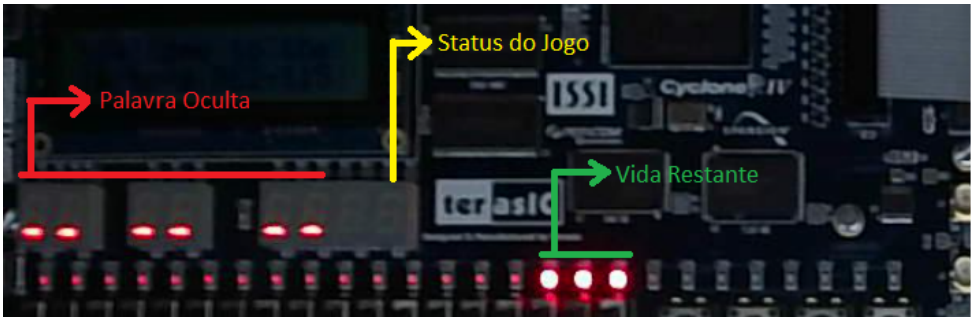


Figura 4: Indicação de saídas do jogo

2.3 Forma de projeção

A implementação no Quartus[2] foi realizada com um pensamento abstrato dos componentes a serem utilizados, não levando em conta os componentes físicos que existem em CI's[3] comerciais. Essa forma de abstração de funcionamento permitiu sua rápida criação.

Abaixo segue uma representação da hierarquia de componentes, em que o bloco que está no início da seta envia os sinais para o bloco que está no fim da mesma com o propósito de obter sua saída.

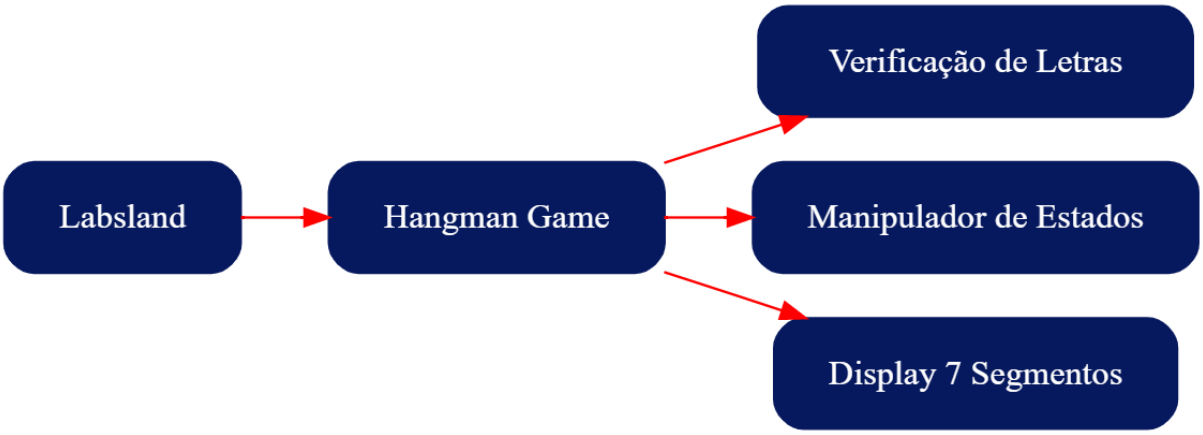


Figura 5: Diagrama geral de componentes

2.4 Módulos

Os módulos serão abordados do menor nível de hierarquia até o maior para facilitar a compreensão, uma vez que os mesmos não dependem de nenhum componente para a compreensão de seu funcionamento.

2.4.1 Display

2.4.1.1 Descrição

[2]Software utilizado para a criação do código e teste do mesmo pela funcionalidade de formas de onda.
[3]Circuitos Integrados

O objetivo deste módulo é facilitar a visualização na tela dos números exibidos, tanto as letras ocultas (números em hexadecimal de 0 a F), como os caracteres de controle: letra oculta (“-”), ganhou o jogo (“G”) e perdeu o jogo (“P”).

Figura 6: Diagrama de entradas e saídas do Módulo Display

2.4.1.2 Código

Como os displays do LabsLand 2021 possuem Anodo Comum[4], o nível lógico de ativação é baixo (0).

```vhdl
-- UNIVERSIDADE FEDERAL DO RIO DE JANEIRO - UFRJ
-- Autor: Christian Marques de Oliveira Silva
-- Disciplina: Laboratório de Sistemas Digitais
--
--  Objetivo: Exibir valor em um display de 7 segmentos
--
-- Entradas:    VALUE_IN, CLK
-- Saidas:      DISPLAY_OUT
----------------------------------------------------------------------------------------------------

-----------------------------------
-- Bibliotecas
-----------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
-----------------------------------

-----------------------------------
-- Entidade Principal
-----------------------------------
entity display_7seg is
    port(
        VALUE_IN    :   in  std_logic_vector(4 downto 0);   -- Codido de entrada para visualizacao (4=Oculto, 3-0=valores)
        CLK         :   in  std_logic;                      -- Sinal de atualizacao dos estados
        DISPLAY_OUT :   out std_logic_vector(6 downto 0));  -- Segmentos a serem acesos
end display_7seg;
-----------------------------------

-----------------------------------
-- Arquitetura Principal
-----------------------------------
architecture hardware of display_7seg is

begin
    process(CLK)
    begin

        if (rising_edge(CLK)) then
            case VALUE_IN is
                --                          6543210
                when "00000" => DISPLAY_OUT <= "1000000"; -- "0"
                when "00001" => DISPLAY_OUT <= "1111001"; -- "1"
                when "00010" => DISPLAY_OUT <= "0100100"; -- "2"
                when "00011" => DISPLAY_OUT <= "0110000"; -- "3"
                when "00100" => DISPLAY_OUT <= "0011001"; -- "4"
                when "00101" => DISPLAY_OUT <= "0010010"; -- "5"
                when "00110" => DISPLAY_OUT <= "0000011"; -- "6"
                when "00111" => DISPLAY_OUT <= "1111000"; -- "7"
                when "01000" => DISPLAY_OUT <= "0000000"; -- "8"
                when "01001" => DISPLAY_OUT <= "0011000"; -- "9"
                when "01010" => DISPLAY_OUT <= "0001000"; -- "A"
                when "01011" => DISPLAY_OUT <= "0000011"; -- "B"
                when "01100" => DISPLAY_OUT <= "1000110"; -- "C"
                when "01101" => DISPLAY_OUT <= "0100001"; -- "D"
                when "01110" => DISPLAY_OUT <= "0000110"; -- "E"
                when "01111" => DISPLAY_OUT <= "0001110"; -- "F"
                when "10000" => DISPLAY_OUT <= "1110111"; -- "-"
                when "10001" => DISPLAY_OUT <= "1000010"; -- "G"
                when "10010" => DISPLAY_OUT <= "0001100"; -- "P"
                when others => DISPLAY_OUT <= "1111111"; -- "apagado"
            end case;
        end if;
    end process;

end hardware;
-----------------------------------
```

Figura 7: Código em VHDL do Display

[4]Todos os pinos positivos dos segmentos conectados pelo positivo

2.4.2 Verificação de Letra

2.4.2.1 Descrição

O objetivo deste módulo é receber as entradas do jogador e detectar se a letra clicada é uma das corretas. Caso seja, atualiza a máscara[5], do contrário, diminui a quantidade de vida do participante.

Figura 8: Diagrama de entradas e saídas do Módulo de Verificação de Letras

2.4.2.2 Código

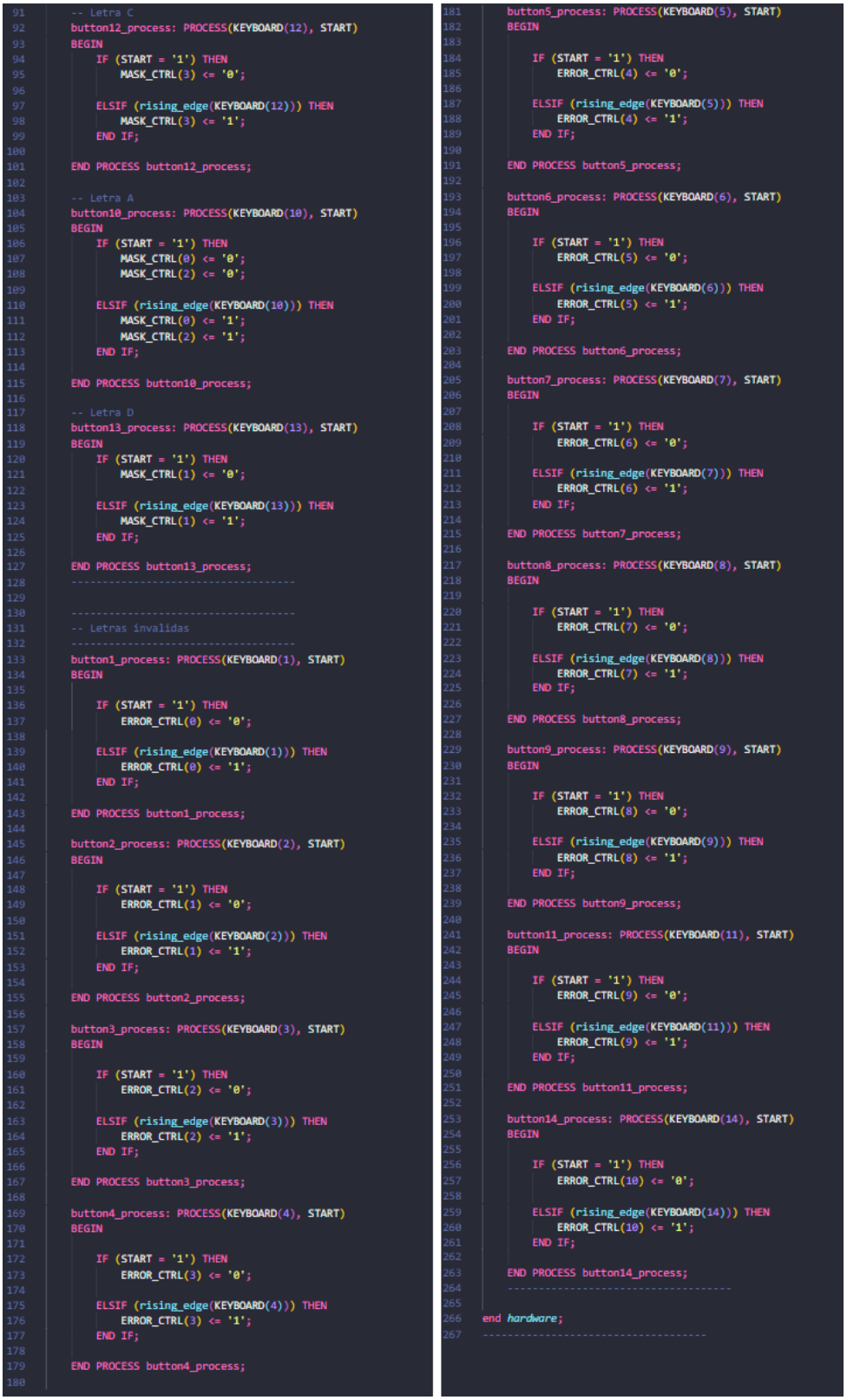
O desenvolvimento contou bastante com o recurso "process", sendo utilizado um para cada botão de modo que cada um deles ou atualiza a máscara (sendo um botão correto) ou insere a posição de erro em um vetor para que a vida seja calculada posteriormente. Para isso que existe o processo com o loop.

Abaixo seguem os códigos que foram divididos em 2 imagens para ser possível visualizar o conteúdo, uma vez que são muitas linhas.

[5]Máscara: É um indicador de qual letra já foi descoberta, onde as que não foram possuem valor "0" em sua posição e as que já foram recebem "1". Essa esstrutura facilita a análise de mudança de estados, além da verificação de qual letra pode ser exibida na tela.

```vhdl
-- UNIVERSIDADE FEDERAL DO RIO DE JANEIRO - UFRJ
-- Autor: Christian Marques de Oliveira Silva
-- Disciplina: Laboratório de Sistemas Digitais
--
-- Objetivo: Detectar se a letra inserida eh valida e fazer a mascara do que deve ser exibido
--
--  Entradas:   KEYBOARD, START, CLK
--  Saidas:     MASK, LIFE
----------------------------------------------------------------------------------------------------

-------------------------------------
-- Bibliotecas
-------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
-------------------------------------

-------------------------------------
-- Entidade Principal
-------------------------------------
entity letter_verification is
    port(
        KEYBOARD            :   in  std_logic_vector(15 downto 0);  -- Teclado com todas as "letras" de 4 bits possiveis (de 0 a 15)
        START               :   in  std_logic;                      -- Sinal de atualizacao dos estados
        CLK                 :   in  std_logic;                      -- Sinal de atualizacao dos estados
        MASK                :   out std_logic_vector(5 downto 0);   -- Mascara que determina quais letras foram acertadas e serao exibidas
        LIFE                :   out std_logic_vector(1 downto 0)    -- Vida restante: 3 (completa) a 0 (perdeu o jogo)
    );
end letter_verification;
-------------------------------------

-------------------------------------
-- Arquitetura Principal
-------------------------------------
architecture hardware of letter_verification is
    SIGNAL MASK_CTRL    : std_logic_vector(5 downto 0)  := (others => '0'); -- Controle da mascara de ativacao (letra correta)
    SIGNAL ERROR_CTRL   : std_logic_vector(10 downto 0) := (others => '0'); -- Controle do subtrator de vidas (letra errada)

    -- Sinal de Controle da vida restante
    SIGNAL LIFE_CTRL    : std_logic_vector(1 downto 0) := "11"; -- Controle das vidas restantes
begin

    MASK <= MASK_CTRL;

    life_process: PROCESS(CLK)
        VARIABLE life_counter : std_logic_vector(1 downto 0) := "11"; -- Vidas restantes
    BEGIN

        IF (rising_edge(CLK)) THEN
            life_counter := "11";
            FOR index IN 0 to 10 LOOP
                IF ((ERROR_CTRL(index) = '1') and (life_counter > 0)) THEN
                    life_counter := life_counter - 1;
                END IF;
            END LOOP;
        END IF;

        LIFE_CTRL <= life_counter;
        LIFE <= LIFE_CTRL;
    END PROCESS life_process;

    -------------------------------------
    -- Letras corretas
    -------------------------------------
    -- Letra F
    button15_process: PROCESS(KEYBOARD(10), START)
    BEGIN
        IF (START = '1') THEN
            MASK_CTRL(5) <= '0';

        ELSIF (rising_edge(KEYBOARD(15))) THEN
            MASK_CTRL(5) <= '1';
        END IF;

    END PROCESS button15_process;

    -- Letra O
    button0_process: PROCESS(KEYBOARD(0), START)
    BEGIN
        IF (START = '1') THEN
            MASK_CTRL(4) <= '0';

        ELSIF (rising_edge(KEYBOARD(0))) THEN
            MASK_CTRL(4) <= '1';
        END IF;

    END PROCESS button0_process;

```

Figura 9: Código em VHDL da Verificação de Letras - Parte 1



```
    -- Letra C
    button12_process: PROCESS(KEYBOARD(12), START)
    BEGIN
        IF (START = '1') THEN
            MASK_CTRL(3) <= '0';

        ELSIF (rising_edge(KEYBOARD(12))) THEN
            MASK_CTRL(3) <= '1';
        END IF;

    END PROCESS button12_process;

    -- Letra A
    button10_process: PROCESS(KEYBOARD(10), START)
    BEGIN
        IF (START = '1') THEN
            MASK_CTRL(0) <= '0';
            MASK_CTRL(2) <= '0';

        ELSIF (rising_edge(KEYBOARD(10))) THEN
            MASK_CTRL(0) <= '1';
            MASK_CTRL(2) <= '1';
        END IF;

    END PROCESS button10_process;

    -- Letra D
    button13_process: PROCESS(KEYBOARD(13), START)
    BEGIN
        IF (START = '1') THEN
            MASK_CTRL(1) <= '0';

        ELSIF (rising_edge(KEYBOARD(13))) THEN
            MASK_CTRL(1) <= '1';
        END IF;

    END PROCESS button13_process;
    -----------------------------------

    -----------------------------------
    -- Letras invalidas
    -----------------------------------
    button1_process: PROCESS(KEYBOARD(1), START)
    BEGIN

        IF (START = '1') THEN
            ERROR_CTRL(0) <= '0';

        ELSIF (rising_edge(KEYBOARD(1))) THEN
            ERROR_CTRL(0) <= '1';
        END IF;

    END PROCESS button1_process;

    button2_process: PROCESS(KEYBOARD(2), START)
    BEGIN

        IF (START = '1') THEN
            ERROR_CTRL(1) <= '0';

        ELSIF (rising_edge(KEYBOARD(2))) THEN
            ERROR_CTRL(1) <= '1';
        END IF;

    END PROCESS button2_process;

    button3_process: PROCESS(KEYBOARD(3), START)
    BEGIN

        IF (START = '1') THEN
            ERROR_CTRL(2) <= '0';

        ELSIF (rising_edge(KEYBOARD(3))) THEN
            ERROR_CTRL(2) <= '1';
        END IF;

    END PROCESS button3_process;

    button4_process: PROCESS(KEYBOARD(4), START)
    BEGIN

        IF (START = '1') THEN
            ERROR_CTRL(3) <= '0';

        ELSIF (rising_edge(KEYBOARD(4))) THEN
            ERROR_CTRL(3) <= '1';
        END IF;

    END PROCESS button4_process;

    button5_process: PROCESS(KEYBOARD(5), START)
    BEGIN

        IF (START = '1') THEN
            ERROR_CTRL(4) <= '0';

        ELSIF (rising_edge(KEYBOARD(5))) THEN
            ERROR_CTRL(4) <= '1';
        END IF;

    END PROCESS button5_process;

    button6_process: PROCESS(KEYBOARD(6), START)
    BEGIN

        IF (START = '1') THEN
            ERROR_CTRL(5) <= '0';

        ELSIF (rising_edge(KEYBOARD(6))) THEN
            ERROR_CTRL(5) <= '1';
        END IF;

    END PROCESS button6_process;

    button7_process: PROCESS(KEYBOARD(7), START)
    BEGIN

        IF (START = '1') THEN
            ERROR_CTRL(6) <= '0';

        ELSIF (rising_edge(KEYBOARD(7))) THEN
            ERROR_CTRL(6) <= '1';
        END IF;

    END PROCESS button7_process;

    button8_process: PROCESS(KEYBOARD(8), START)
    BEGIN

        IF (START = '1') THEN
            ERROR_CTRL(7) <= '0';

        ELSIF (rising_edge(KEYBOARD(8))) THEN
            ERROR_CTRL(7) <= '1';
        END IF;

    END PROCESS button8_process;

    button9_process: PROCESS(KEYBOARD(9), START)
    BEGIN

        IF (START = '1') THEN
            ERROR_CTRL(8) <= '0';

        ELSIF (rising_edge(KEYBOARD(9))) THEN
            ERROR_CTRL(8) <= '1';
        END IF;

    END PROCESS button9_process;

    button11_process: PROCESS(KEYBOARD(11), START)
    BEGIN

        IF (START = '1') THEN
            ERROR_CTRL(9) <= '0';

        ELSIF (rising_edge(KEYBOARD(11))) THEN
            ERROR_CTRL(9) <= '1';
        END IF;

    END PROCESS button11_process;

    button14_process: PROCESS(KEYBOARD(14), START)
    BEGIN

        IF (START = '1') THEN
            ERROR_CTRL(10) <= '0';

        ELSIF (rising_edge(KEYBOARD(14))) THEN
            ERROR_CTRL(10) <= '1';
        END IF;

    END PROCESS button14_process;
    -----------------------------------

end hardware;
-----------------------------------
```

Figura 10: Código em VHDL da Verificação de Letras - Parte 2

2.4.2.3 Simulação

Foi criada uma simulação no Quartus para testar o comportamento das vidas restantes e da máscara de acerto, obtendo-se o seguinte resultado:

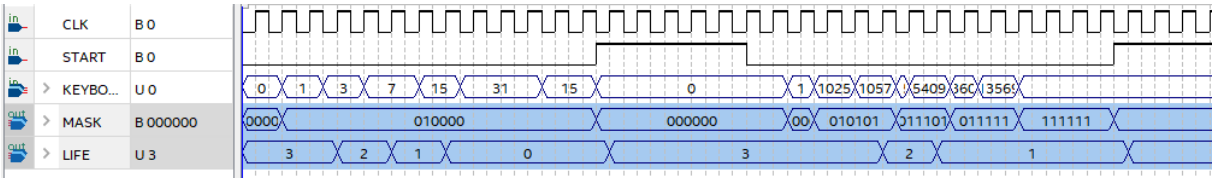


Figura 11: Simulação WF do módulo de Verificação de Letras

2.4.3 Manipulador de Estados

2.4.3.1 Descrição

O objetivo deste módulo é dada a situação atual da máscara, a quantidade de vida restante e o estado que se encontra, seguir para outro estado ou continuar. Obviamente que também reseta em caso de vitória ou derrota.

Figura 12: Diagrama de entradas e saídas do Módulo Manipulador de Estados

2.4.3.2 Código

Basicamente é dividido em 2 etapas. A primeira verifica e atualiza a mudança de estados. A segunda realiza a adequação do estado atual para a variável de saída que informa o estado através de uma codificação.

Abaixo seguem os códigos que foram divididos em 2 imagens para ser possível visualizar o conteúdo, uma vez que são muitas linhas.

```vhdl
-- UNIVERSIDADE FEDERAL DO RIO DE JANEIRO - UFRJ
-- Autor: Christian Marques de Oliveira Silva
-- Disciplina: Laboratório de Sistemas Digitais
--
-- Objetivo: Manipular os estados do sistema baseado no estado atual e nas entradas
--
--  Entradas:   MASK, LIFE, START, CLK
--  Saidas:     STATE_OUT
----------------------------------------------------------------------------------------------------

-----------------------------------
-- Bibliotecas
-----------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
-----------------------------------

-----------------------------------
-- Entidade Principal
-----------------------------------
entity state_handler is
    port(
        MASK            :   in  std_logic_vector(5 downto 0);   -- Mascara que determina quais letras foram acertadas e serao exibidas
        LIFE            :   in  std_logic_vector(1 downto 0);   -- Vida restante: 3 (completa) a 0 (perdeu o jogo)
        START           :   in  std_logic;                      -- Comando para comecar o jogo
        CLK             :   in  std_logic;                      -- Sinal de atualizacao dos estados
        STATE_OUT       :   out std_logic_vector(2 downto 0)    -- Estado das palavras: 0-4 (N letras certas), 5 (win), 6 (lose) e 7 (waiting)
    );
end state_handler;
-----------------------------------

-----------------------------------
-- Arquitetura Principal
-----------------------------------
architecture hardware of state_handler is
    -- Criacao dos estados
    TYPE GAME_STATE IS (Waiting, Char0, Char1, Char2, Char3, Char4, Win, Lose);
    SIGNAL current_state: GAME_STATE := Waiting;
    SIGNAL next_state: GAME_STATE := Char0;

    -- Funcao para obter a quantidade de letras diferentes situadas na mascara
    FUNCTION GET_NUMBER_VALID_LETTERS (MASK_INPUT : in std_logic_vector) return std_logic_vector is
        VARIABLE letters : std_logic_vector(2 downto 0) := "000";
    BEGIN
        FOR index IN 1 TO MASK_INPUT'length-1 LOOP
            IF (MASK_INPUT(index) = '1') THEN
                letters := letters + 1;
            END IF;
        END LOOP;

        RETURN letters;

    END FUNCTION GET_NUMBER_VALID_LETTERS;

begin

    -- Transicao de estados
    change_state: PROCESS (START, MASK, CLK)
    BEGIN

        -- Caso esteja em espera e receba o comando de start, o estado atual passa a ser o de CHAR_0 e o proximo CHAR_1
        IF (((current_state = Waiting) or (current_state = Win) or (current_state = Lose)) and (START = '1') and rising_edge(CLK)) THEN
            current_state   <= Char0;
            next_state      <= Char1;

        -- Caso esteja em CHAR_0 e possua 1 letra valida na mascara, o estado atual passa a ser o de CHAR_1 e o proximo CHAR_2
        ELSIF ((current_state = Char0) and (GET_NUMBER_VALID_LETTERS(MASK) = "001") and rising_edge(CLK)) THEN
            current_state   <= next_state;
            next_state      <= Char2;

        -- Caso esteja em CHAR_1 e possua 2 letras validas na mascara, o estado atual passa a ser o de CHAR_2 e o proximo CHAR_3
        ELSIF ((current_state = Char1) and (GET_NUMBER_VALID_LETTERS(MASK) = "010") and rising_edge(CLK)) THEN
            current_state   <= next_state;
            next_state      <= Char3;

        -- Caso esteja em CHAR_2 e possua 3 letras validas na mascara, o estado atual passa a ser o de CHAR_3 e o proximo CHAR_4
        ELSIF ((current_state = Char2) and (GET_NUMBER_VALID_LETTERS(MASK) = "011") and rising_edge(CLK)) THEN
            current_state   <= next_state;
            next_state      <= Char4;

        -- Caso esteja em CHAR_3 e possua 4 letras validas na mascara, o estado atual passa a ser o de CHAR_4 e o proximo WIN
        ELSIF ((current_state = Char3) and (GET_NUMBER_VALID_LETTERS(MASK) = "100") and rising_edge(CLK)) THEN
            current_state   <= next_state;
            next_state      <= Win;

        -- Caso esteja em CHAR_4 e possua 5 letras validas na mascara, o estado atual passa a ser o de WIN
        ELSIF ((current_state = Char4) and (GET_NUMBER_VALID_LETTERS(MASK) = "101") and rising_edge(CLK)) THEN
            current_state   <= next_state;
```

Figura 13: Código em VHDL do Manipulador de Estados - Parte 1

```
91
92          -- Caso nao esteja no modo espera e nao houver mais vida restante, o estado atual passa a ser o de LOSE
93          ELSIF ((current_state /= Waiting) and (LIFE = 0) and rising_edge(CLK)) THEN
94              current_state   <= Lose;
95
96          -- Mantem os estados atuais
97          ELSE
98              current_state   <= current_state;
99              next_state      <= next_state;
100
101         END IF;
102
103     END PROCESS change_state;
104
105     -- Atuacao das saidas a partir do estado
106     output_update: PROCESS (current_state)
107     BEGIN
108
109         -- 1 conjunto de letras diferentes descoberta
110         IF ((current_state = Char0)) THEN STATE_OUT <= "000";
111
112         -- 2 conjuntos de letras diferentes descobertas
113         ELSIF ((current_state = Char1)) THEN STATE_OUT <= "001";
114
115         -- 2 conjuntos de letras diferentes descobertas
116         ELSIF ((current_state = Char2)) THEN STATE_OUT <= "010";
117
118         -- 3 conjuntos de letras diferentes descobertas
119         ELSIF ((current_state = Char3)) THEN STATE_OUT <= "011";
120
121         -- 4 conjuntos de letras diferentes descobertas
122         ELSIF ((current_state = Char4)) THEN STATE_OUT <= "100";
123
124         -- 5 conjuntos de letras diferentes descobertas (Venceu o jogo)
125         ELSIF ((current_state = Win)) THEN STATE_OUT <= "101";
126
127         -- Perdeu o jogo
128         ELSIF ((current_state = Lose)) THEN STATE_OUT <= "110";
129
130         -- Aguardando o inicio da partida
131         ELSIF ((current_state = Waiting)) THEN STATE_OUT <= "111";
132
133         END IF;
134
135     END PROCESS output_update;
136
137 end hardware;
138 ---------------------------------
```

Figura 14: Código em VHDL do Manipulador de Estados - Parte 2

2.4.3.3 Simulação

Foi criada uma simulação no Quartus para validar o comportamento do código de saída do estado a partir de todas as entradas, obtendo-se o seguinte resultado:

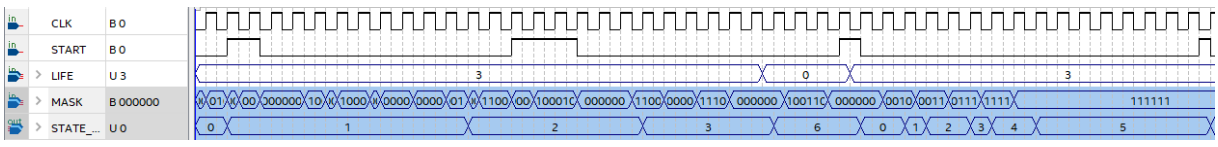


Figura 15: Simulação WF do módulo Manipulador de Estados

2.4.4 Hangman Game

2.4.4.1 Descrição

O objetivo deste módulo é gerenciar o funcionamento do jogo como um todo. Como os módulos de verificação de letras e manipulador de estados fizeram a maior parte do trabalho, sua função é basicamente controlar a forma de expor isso ao jogador.

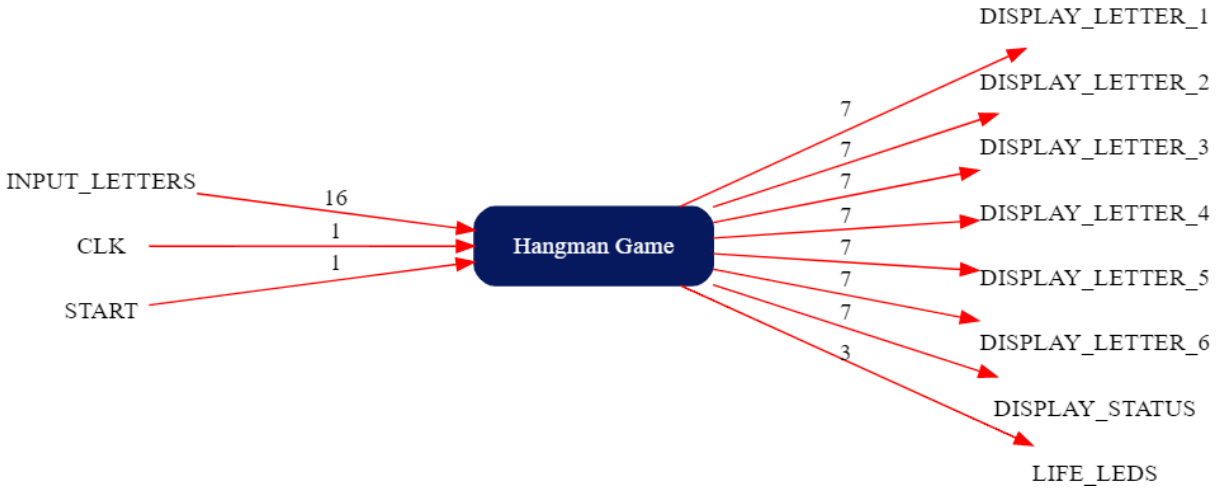


Figura 16: Diagrama de entradas e saídas do Módulo Geral do Jogo

2.4.4.2 Código

A lógica ficou bem simples e em um nível de abstração alto, graças às variáveis de máscara e código do estado. Desse modo foi necessário somente fazer a manipulação da saída.

Abaixo seguem os códigos que foram divididos em 2 imagens para ser possível visualizar o conteúdo, uma vez que são muitas linhas.

```vhdl
-- UNIVERSIDADE FEDERAL DO RIO DE JANEIRO - UFRJ
-- Autor: Christian Marques de Oliveira Silva
-- Disciplina: Laboratório de Sistemas Digitais
--
-- Objetivo: Controlar as acoes do jogo como um todo
--
--  Entradas:   START, CLK, INPUT_LETTERS
--  Saidas:     DISPLAY_LETTER(1-6), DISPLAY_STATUS, LIFE_LEDS
-------------------------------------------------------------------------------------------------

-----------------------------------
-- Bibliotecas
-----------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
-----------------------------------

-----------------------------------
-- Entidade Principal
-----------------------------------
entity hangman is
    port(
        START                :   in  std_logic;                     -- Comando para comecar o jogo
        CLK                  :   in  std_logic;                     -- Sinal de atualizacao dos estados
        INPUT_LETTERS        :   in  std_logic_vector(15 downto 0); -- Letras de entrada (0 a 15 => 0 a F)
        DISPLAY_LETTER1      :   out std_logic_vector(6 downto 0);  -- Display da 1a letra
        DISPLAY_LETTER2      :   out std_logic_vector(6 downto 0);  -- Display da 2a letra
        DISPLAY_LETTER3      :   out std_logic_vector(6 downto 0);  -- Display da 3a letra
        DISPLAY_LETTER4      :   out std_logic_vector(6 downto 0);  -- Display da 4a letra
        DISPLAY_LETTER5      :   out std_logic_vector(6 downto 0);  -- Display da 5a letra
        DISPLAY_LETTER6      :   out std_logic_vector(6 downto 0);  -- Display da 6a letra
        DISPLAY_STATUS       :   out std_logic_vector(6 downto 0);  -- Display do Status do JOGO ("", "P" ou "G")
        LIFE_LEDS            :   out std_logic_vector(2 downto 0)   -- LEDS que indicam a quantidade de vidas restante
    );
end hangman;
-----------------------------------

-----------------------------------
-- Arquitetura Principal
-----------------------------------
architecture hardware of hangman is

    -- Definicao do verificador de letras
    component letter_verification is
        port(
            KEYBOARD         :   in  std_logic_vector(15 downto 0);  -- Teclado com todas as "letras" de 4 bits possiveis (de 0 a 15)
            START            :   in  std_logic;                      -- Sinal de atualizacao dos estados
            CLK              :   in  std_logic;                      -- Sinal de atualizacao dos estados
            MASK             :   out std_logic_vector(5 downto 0);   -- Mascara que determina quais letras foram acertadas e serao exibidas
            LIFE             :   out std_logic_vector(1 downto 0)    -- Vida restante: 3 (completa) a 0 (perdeu o jogo)
        );
    end component;

    -- Definicao do manipulador de estados
    component state_handler is
        port(
            MASK             :   in  std_logic_vector(5 downto 0);   -- Mascara que determina quais letras foram acertadas e serao exibidas
            LIFE             :   in  std_logic_vector(1 downto 0);   -- Vida restante: 3 (completa) a 0 (perdeu o jogo)
            START            :   in  std_logic;                      -- Comando para comecar o jogo
            CLK              :   in  std_logic;                      -- Sinal de atualizacao dos estados
            STATE_OUT        :   out std_logic_vector(2 downto 0)    -- Estado das palavras: 0-4 (N letras certas), 5 (win), 6 (lose) e 7 (waiting)
        );
    end component;

    -- Definicao do display
    component display_7seg is
        port(
            VALUE_IN     :   in  std_logic_vector(4 downto 0);   -- Codido de entrada para visualizacao (4=Oculto, 3-0=valores)
            CLK          :   in  std_logic;                      -- Sinal de atualizacao dos estados
            DISPLAY_OUT  :   out std_logic_vector(6 downto 0)    -- Segmentos a serem acesos
        );
    end component;

```

Figura 17: Código em VHDL do Jogo - Parte 1

```
    -- Variaveis
    -- Letras da palavra oculta
    CONSTANT letter6 : std_logic_vector(4 downto 0) := "01111";   -- 16 = "F"
    CONSTANT letter5 : std_logic_vector(4 downto 0) := "00000";   -- 0 =  "O"
    CONSTANT letter4 : std_logic_vector(4 downto 0) := "01100";   -- 12 = "C"
    CONSTANT letter3 : std_logic_vector(4 downto 0) := "01010";   -- 10 = "A"
    CONSTANT letter2 : std_logic_vector(4 downto 0) := "01101";   -- 13 = "D"
    CONSTANT letter1 : std_logic_vector(4 downto 0) := "01010";   -- 10 = "A"

    -- Constantes de status
    CONSTANT HIDDEN_LETTER  : std_logic_vector(4 downto 0) := "10000"; -- "-"
    CONSTANT WIN_GAME       : std_logic_vector(4 downto 0) := "10001"; -- "G"
    CONSTANT LOSE_GAME      : std_logic_vector(4 downto 0) := "10010"; -- "P"
    CONSTANT NOTHING        : std_logic_vector(4 downto 0) := "11111"; -- ""

    SIGNAL MASK : std_logic_vector(5 downto 0);   -- Mascara que determina quais letras foram acertadas e serao exibidas
    SIGNAL LIFE : std_logic_vector(1 downto 0);    -- Vida restante: 3 (completa) a 0 (perdeu o jogo)
    SIGNAL STATE: std_logic_vector(2 downto 0);   -- Estado das palavras: 0-4 (N letras certas), 5 (win), 6 (lose) e 7 (waiting)

    SIGNAL DISPLAY_LETTER_1 : std_logic_vector(4 downto 0) := NOTHING;
    SIGNAL DISPLAY_LETTER_2 : std_logic_vector(4 downto 0) := NOTHING;
    SIGNAL DISPLAY_LETTER_3 : std_logic_vector(4 downto 0) := NOTHING;
    SIGNAL DISPLAY_LETTER_4 : std_logic_vector(4 downto 0) := NOTHING;
    SIGNAL DISPLAY_LETTER_5 : std_logic_vector(4 downto 0) := NOTHING;
    SIGNAL DISPLAY_LETTER_6 : std_logic_vector(4 downto 0) := NOTHING;
    SIGNAL DISPLAY_STAT     : std_logic_vector(4 downto 0) := NOTHING;
    SIGNAL DISPLAY_LIFE     : std_logic_vector(1 downto 0) := "11";

begin


    letter_verification_comp: letter_verification PORT MAP(INPUT_LETTERS, START, CLK, MASK, LIFE);

    state_handler_comp: state_handler PORT MAP(MASK, LIFE, START, CLK, STATE);

    state_proc: PROCESS (CLK, STATE, LIFE, MASK)
    BEGIN
        DISPLAY_LIFE <= LIFE;

        IF ((STATE = "110")) THEN
            DISPLAY_STAT <= LOSE_GAME;

        ELSIF((STATE = "101")) THEN
            DISPLAY_STAT <= WIN_GAME;

        ELSE
            DISPLAY_STAT <= NOTHING;

            -- Esperando
            IF((STATE = "111")) THEN
                DISPLAY_LETTER_1 <= NOTHING;
                DISPLAY_LETTER_2 <= NOTHING;
                DISPLAY_LETTER_3 <= NOTHING;
                DISPLAY_LETTER_4 <= NOTHING;
                DISPLAY_LETTER_5 <= NOTHING;
                DISPLAY_LETTER_6 <= NOTHING;

            ELSE
                IF (MASK(5) = '1') THEN DISPLAY_LETTER_6 <= letter6; ELSE DISPLAY_LETTER_6 <= HIDDEN_LETTER; END IF;

                IF (MASK(4) = '1') THEN DISPLAY_LETTER_5 <= letter5; ELSE DISPLAY_LETTER_5 <= HIDDEN_LETTER; END IF;

                IF (MASK(3) = '1') THEN DISPLAY_LETTER_4 <= letter4; ELSE DISPLAY_LETTER_4 <= HIDDEN_LETTER; END IF;

                IF (MASK(2) = '1') THEN DISPLAY_LETTER_3 <= letter3; ELSE DISPLAY_LETTER_3 <= HIDDEN_LETTER; END IF;

                IF (MASK(1) = '1') THEN DISPLAY_LETTER_2 <= letter2; ELSE DISPLAY_LETTER_2 <= HIDDEN_LETTER; END IF;

                IF (MASK(0) = '1') THEN DISPLAY_LETTER_1 <= letter1; ELSE DISPLAY_LETTER_1 <= HIDDEN_LETTER; END IF;
            END IF;
        END IF;
    END PROCESS state_proc;


    display6_comp: display_7seg PORT MAP(DISPLAY_LETTER_6, CLK, DISPLAY_LETTER6);
    display5_comp: display_7seg PORT MAP(DISPLAY_LETTER_5, CLK, DISPLAY_LETTER5);
    display4_comp: display_7seg PORT MAP(DISPLAY_LETTER_4, CLK, DISPLAY_LETTER4);
    display3_comp: display_7seg PORT MAP(DISPLAY_LETTER_3, CLK, DISPLAY_LETTER3);
    display2_comp: display_7seg PORT MAP(DISPLAY_LETTER_2, CLK, DISPLAY_LETTER2);
    display1_comp: display_7seg PORT MAP(DISPLAY_LETTER_1, CLK, DISPLAY_LETTER1);

    display_status_comp: display_7seg PORT MAP(DISPLAY_STAT, CLK, DISPLAY_STATUS);

    LIFE_LEDS <=    "000" when DISPLAY_LIFE = "00" else
                    "001" when DISPLAY_LIFE = "01" else
                    "011" when DISPLAY_LIFE = "10" else
                    "111" when DISPLAY_LIFE = "11";

end hardware;
---------------------------------
```

Figura 18: Código em VHDL do Jogo - Parte 2

2.4.4.3 Simulação

Foi criada uma simulação no Quartus para validar de modo geral funcionamento do jogo, obtendo-se o seguinte resultado:

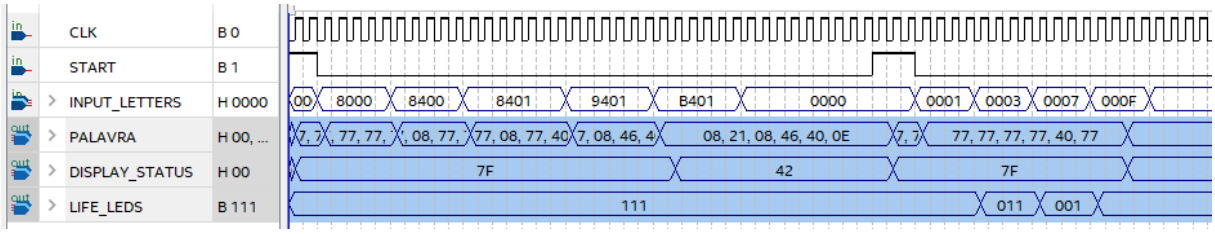


Figura 19: Simulação WF do módulo Geral do Jogo

2.4.5 Interface com o LabsLand

2.4.5.1 Descrição

O objetivo deste módulo é somente encapsular todos os demais componentes já criados para interagir com o LabsLand 2021. Sua função é facilitar a leitura do código, uma vez que os vetores do LabsLand 2021 não possuem nomes adequados com a lógica de funcionamento do jogo.

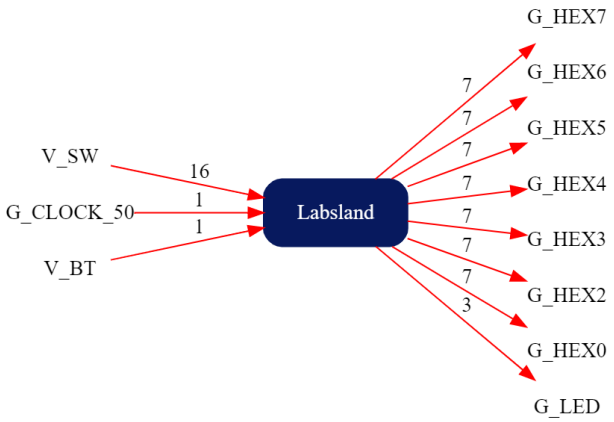


Figura 20: Diagrama de entradas e saídas do Módulo Interface com o LabsLand

2.4.5.2 Código

A implementação tornou-se bem simples com o uso de *signal's*, pois toda a lógica já estava implementada nos componentes criados neste módulo, restando somente o preenchimento correto e utilização para as saídas nas ocasiões devidas. Como o botão de reinício é do tipo normalmente alto, o nível lógico enviado para o componente é invertido.

```vhdl
-- UNIVERSIDADE FEDERAL DO RIO DE JANEIRO - UFRJ
-- Autor: Christian Marques de Oliveira Silva
-- Disciplina: Laboratório de Sistemas Digitais
--
--  Objetivo: Interfacear os pinos do LabsLand com o componente do jogo. Isso permite que as variáveis
--            do jogo sejam mais intuitivas.
--
--  Entradas:   V_SW, V_BT, G_CLOCK_50
--  Saídas:     G_HEX(7-2), G_HEX0, G_LED
--
------------------------------------------------------------------------------------------------

-----------------------------------
-- Bibliotecas
-----------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
-----------------------------------

-----------------------------------
-- Entidade Principal
-----------------------------------
entity labsland is port(
    V_SW        :   in  std_logic_vector(15 downto 0);  -- Entradas das letras
    V_BT        :   in  std_logic_vector(1 downto 0);   -- Botao de Start
    G_CLOCK_50  :   in  std_logic;                      -- Clock para o funcionamento do sistema
    G_HEX7      :   out std_logic_vector(6 downto 0);   -- Letra 6
    G_HEX6      :   out std_logic_vector(6 downto 0);   -- Letra 5
    G_HEX5      :   out std_logic_vector(6 downto 0);   -- Letra 4
    G_HEX4      :   out std_logic_vector(6 downto 0);   -- Letra 3
    G_HEX3      :   out std_logic_vector(6 downto 0);   -- Letra 2
    G_HEX2      :   out std_logic_vector(6 downto 0);   -- Letra 1
    G_HEX0      :   out std_logic_vector(6 downto 0);   -- Status do jogo
    G_LED       :   out std_logic_vector(2 downto 0)    -- Vidas restantes
    );
end labsland;

-----------------------------------
-- Arquitetura Principal
-----------------------------------
architecture hardware of labsland is

    -- Definicao do componente
    component hangman is
        port(
            START                   :   in  std_logic;                      -- Comando para comecar o jogo
            CLK                     :   in  std_logic;                      -- Sinal de atualizacao dos estados
            INPUT_LETTERS           :   in  std_logic_vector(15 downto 0);  -- Letras de entrada (0 a 15 => 0 a F)
            DISPLAY_LETTER1         :   out std_logic_vector(6 downto 0);   -- Display da 1a letra
            DISPLAY_LETTER2         :   out std_logic_vector(6 downto 0);   -- Display da 2a letra
            DISPLAY_LETTER3         :   out std_logic_vector(6 downto 0);   -- Display da 3a letra
            DISPLAY_LETTER4         :   out std_logic_vector(6 downto 0);   -- Display da 4a letra
            DISPLAY_LETTER5         :   out std_logic_vector(6 downto 0);   -- Display da 5a letra
            DISPLAY_LETTER6         :   out std_logic_vector(6 downto 0);   -- Display da 6a letra
            DISPLAY_STATUS          :   out std_logic_vector(6 downto 0);   -- Display do Status do JOGO ("", "P" ou "G")
            LIFE_LEDS               :   out std_logic_vector(2 downto 0)   -- LEDS que indicam a quantidade de vidas restante
        );
    end component;
    --
begin

    hangman_game: hangman PORT MAP(not V_BT(0), G_CLOCK_50, V_SW, G_HEX2, G_HEX3, G_HEX4, G_HEX5, G_HEX6, G_HEX7, G_HEX0, G_LED);

end hardware;
-----------------------------------
```

Figura 21: Código em VHDL da Interface com o LabsLand

3 Funcionamento no Labsland

3.1 Esperando início do jogo

Abaixo, como o display *G_HEX0* está apagado, todos os caracteres estão ocultos e não exite nenhum botão ativado, o jogo está aguardando alguma ação do jogado. Assim, está correto.

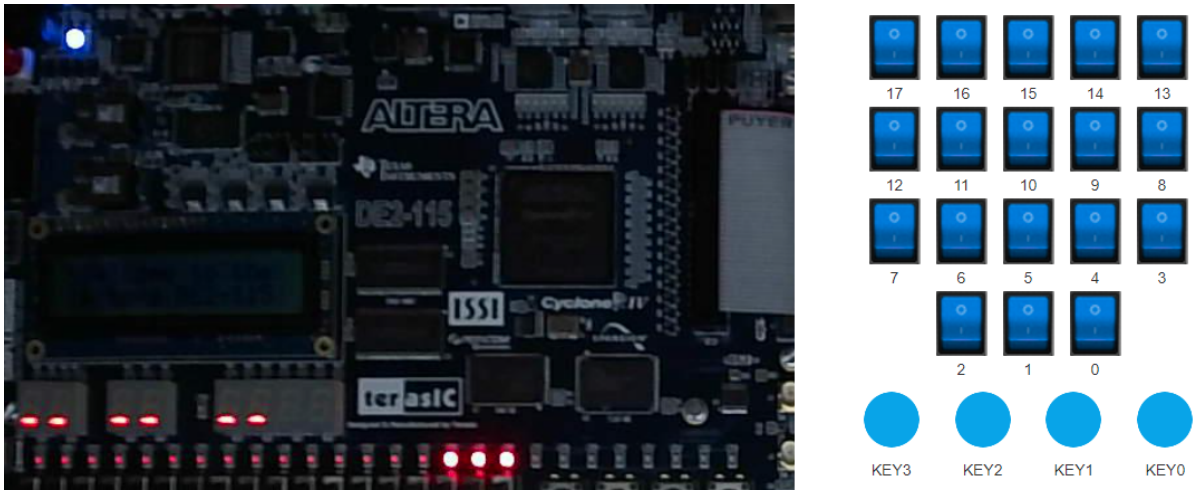


Figura 22: Esperando início do jogo

3.2 Jogando

Abaixo, como o display *G_HEX0* está apagado, exite ao menos um botão ativado e nem todos os caracteres foram descobertos, o jogador está no meio do jogo. E como ainda tem LED de indicação de vida aceso, o jogador realmente não perdeu. Assim, está correto.

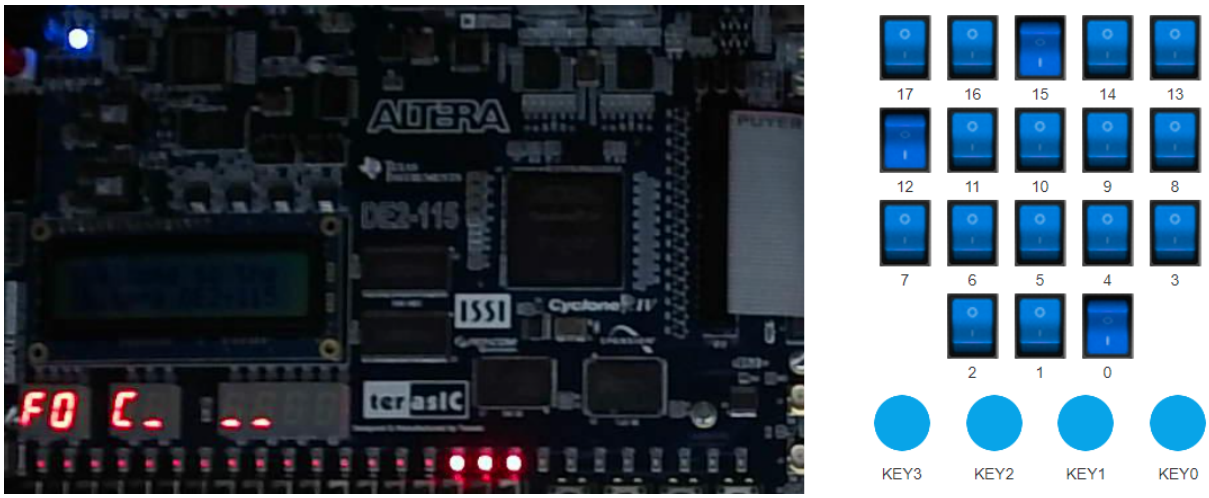


Figura 23: Partida em execução

3.3 Ganhou

Abaixo, com o display *G_HEX0* mostrando "G" e todos os caracteres foram desocultados (mostrando que foram descobertos), o jogador venceu a partida. E como ainda tem LED de indicação de vida aceso, o jogador realmente não perdeu. Assim, está correto.

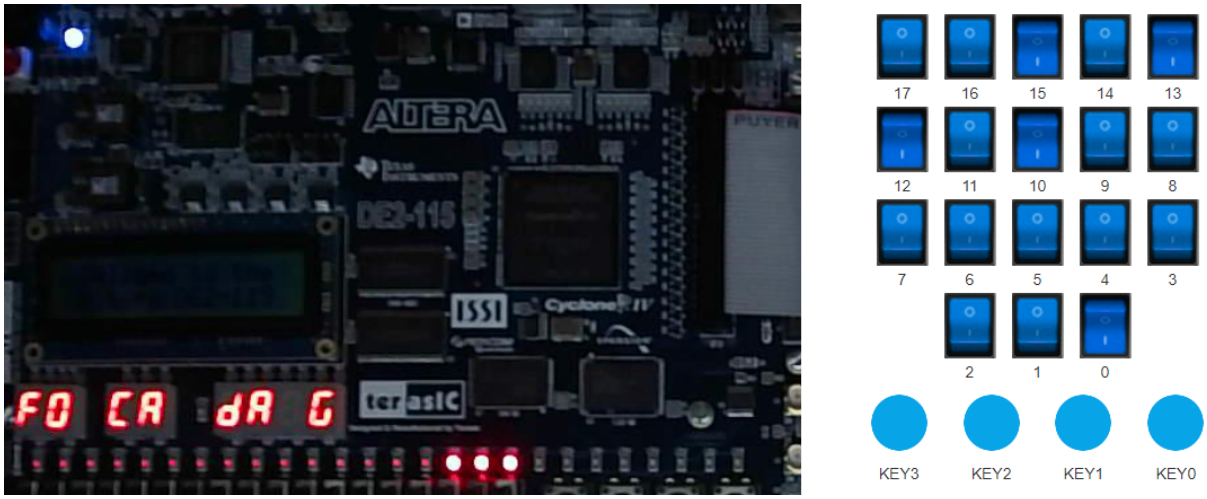


Figura 24: Ganhou a partida (acertou todas as letras ocultas)

3.4 Perdeu

Abaixo, com o display *G_HEX0* mostrando "P" e não existe mais nenhum LED de vida aceso, o jogador perdeu a partida. E como ainda existem caracteres ocultos (mostrando que não foram descobertos), o jogador realmente não venceu. Assim, está correto.

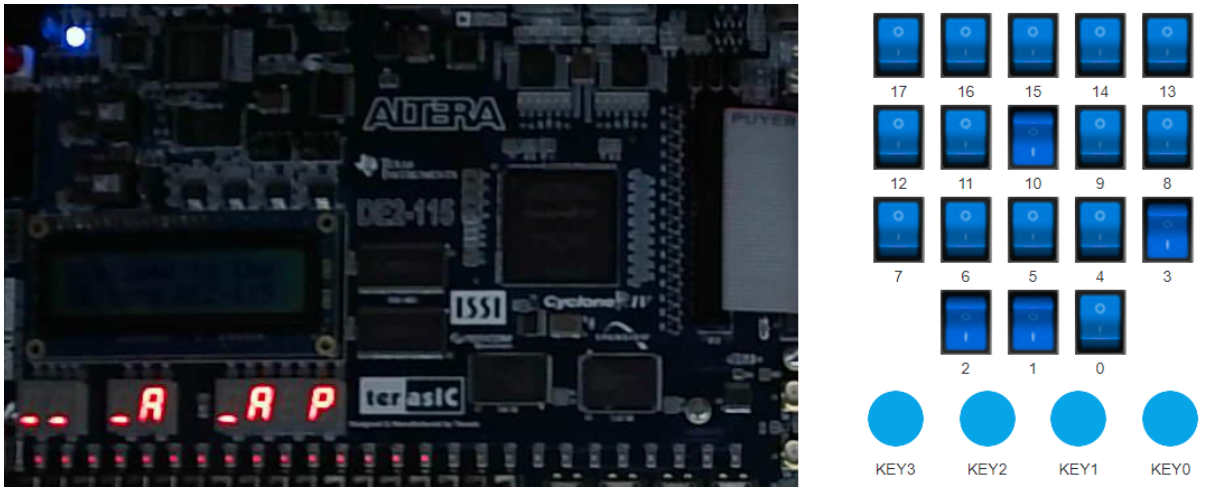


Figura 25: Perdeu a partida (errou 3 letras)

4 Conclusões

O código projetado não apresentou falhas após a conclusão de seu desenvolvimento, cumprindo exatamente o proposto para sua confecção.

Durante a apresentação do trabalho houveram 2 erros por falta de atenção no código (replicação da mesma letra na hora de sair e não segurada do estado de venceu ou perdeu ao final da partida), no entanto, ambos foram corrigidos logo em seguida.

O único momento em que houve um "erro" foi, em um dos testes, quando ao invés da letra F, apareceu um caractere inválido (não programado). A causa desse erro foi porque a FPGA do LabsLand 2021 estava com o segmento central do display mais a esquerda queimado, impossibilitando a visualização do mesmo aceso. A comprovação foi feita testando em outra placa, onde o código voltou a funcionar.

Ainda sobre erros do LabsLand 2021, a documentação do mesmo informa que a ordem dos segmentos dos displays é inversa ao funcionamento real, onde a ordem do MSB ao LSB está trocada.

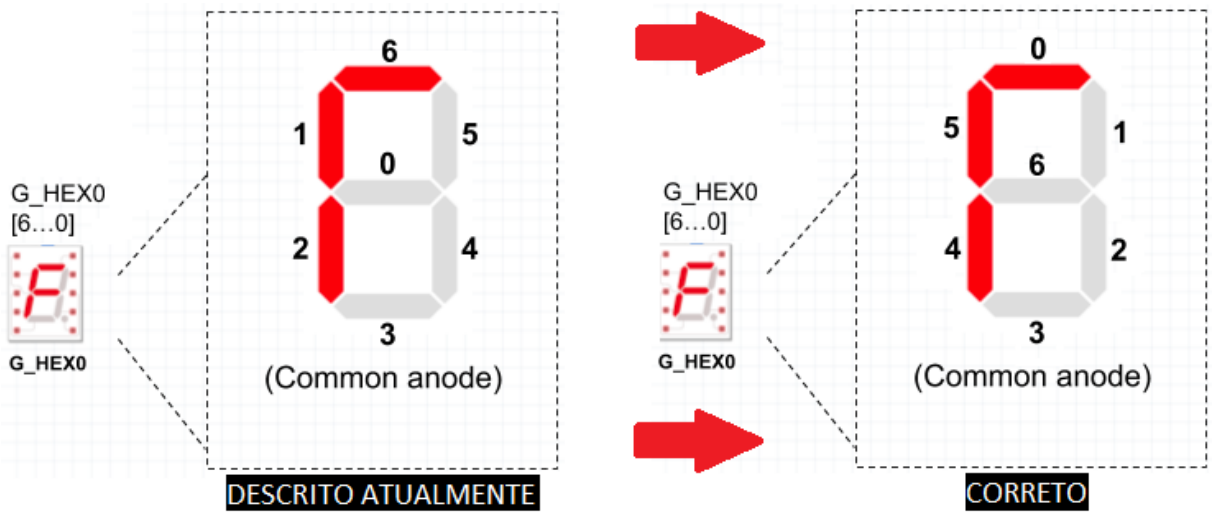


Figura 26: Correção da documentação do LabsLand 2021 para a ordem do display de 7 segmentos

A cima pode ser visto que o MSB do registrador *G_HEX0* (6) é o segmento descrito como 6, no entanto, o que realmente surte efeito no MSB é o segmento descrito inicialmente como 0. E o mesmo equivale para as outras alterações. O único que se manteve foi o segmento 3. Após realizar as alterações a interação funcionou corretamente.

5 Referências

LabsLand (2021). *Laboratório de FPGA's online*. URL: `https://altera-de2-115-vhdl.ide.labsland.com/` (acesso em 05/10/2021).

Luís Henrique Maciel (out. de 2021). *EEL480 - Lab. de Sistemas Digitais*. URL: `https://classroom.google.com/u/0/c/MzY4MDYzMzQ5MDc4` (acesso em 05/10/2021).

Wagner Rambo (2021). *Playlist de FPGA utilizando o Quartus no Youtube*. URL: `https://www.youtube.com/playlist?list=PLZ8dBTV2_5HS79fVexGTtCMDUp7kjnumS` (acesso em 05/10/2021).