UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Departamento de Engenharia Eletrônica e da Computação
Escola Politécnica
EEL 480 – Sistemas Digitais

JOGO DA FORCA EM VHDL

Alunos: Bruno Dantas de Paiva, João Ricardo Magalhães e Guilherme Bergman

Professor: Luis Henrique Maciel Kosmalski Costa

Rio de Janeiro,
2019

I. Introdução:

A forca é um jogo onde uma palavra é escolhida e outra pessoa tenta adivinhar de letra em letra com um número específico de erros que a pessoa possa errar.

Neste trabalho, foi utilizado uma FPGA (que consiste em um arranjo de células lógicas ou blocos lógicos configuráveis contidos em um único circuito integrado) e a linguagem de descrição de hardware (VHDL) para implementar tais funções.

Foi projetado para que a pessoa tecle cada letra através de um teclado PS2; Foi pré alocado no código tanto a palavra quanto a quantidade de erros.

II. Desenvolvimento:

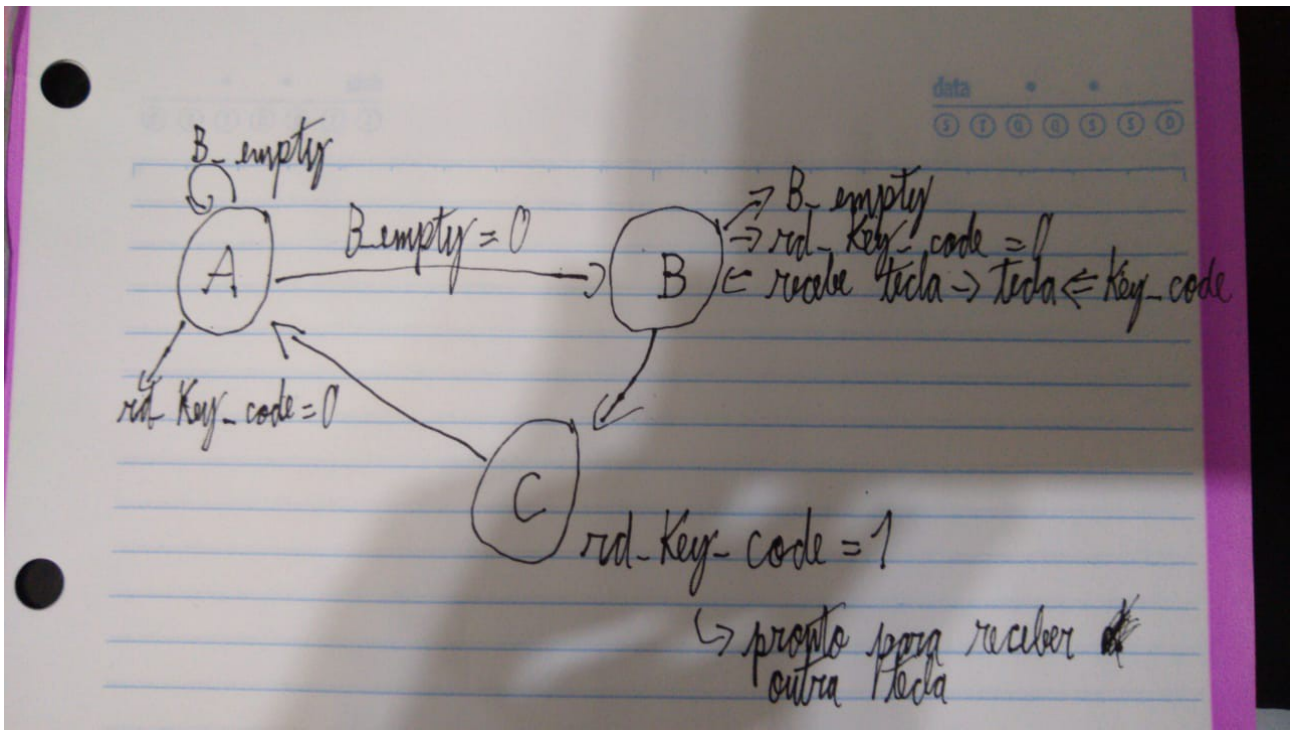
Como base do projeto, utilizamos o módulo Kb_code para tratar os dados recebidos pelo teclado, através do sinal key_code (onde fica armazenado a tecla do buffer), kb_buf_empty que indica se o buffer recebeu alguma informação, e rd_key_code, sinal que libera o buffer para receber nova tecla. Em relação ao display, foi aproveitado o módulo lcd, onde substituímos a frase exemplo que é impressa no display pelo conteúdo que nós desejamos.

O projeto funciona a partir de duas máquinas de estado: uma serve para receber a tecla digitada pelo usuário, e a ultima verifica se a tecla é correta.

A maquina que lê a tecla possui 3 estados, o primeiro espera o kb_buf_empty ficar em 0, movendo para o estado intermediario. Neste, ela avisa para a outra maquina que recebeu uma tecla através do sinal 'teclou', e armazena a tecla no sinal keybuffer. Ela prossegue para o estado final quando for avisada através de 'leu' que a outra maquina já tratou a tecla - essa parte é apenas por segurança, já que a tecla é tratada em apenas um período de clock poderiamos descartar a variavel 'leu'.

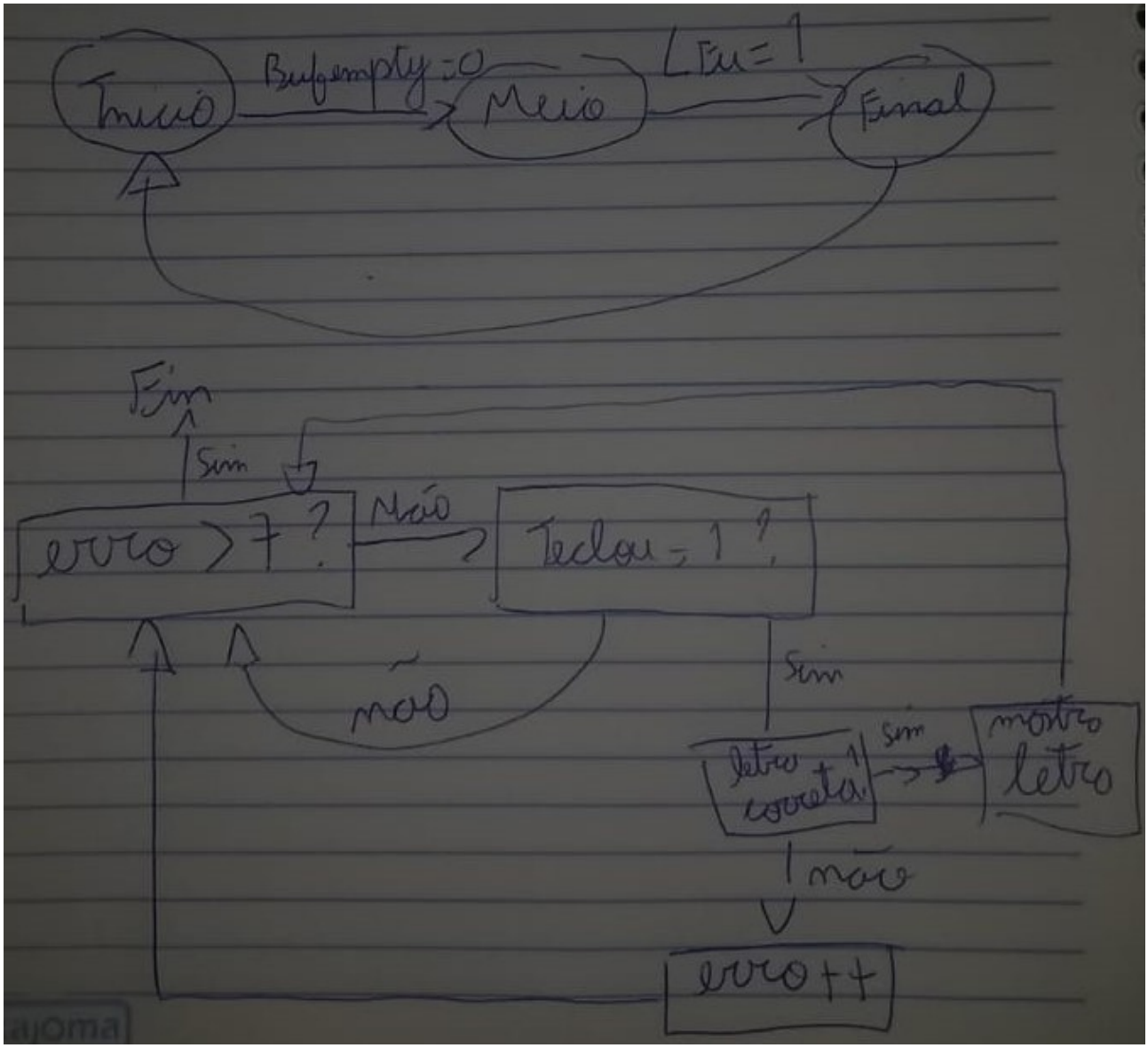
No estado final, 'teclou' e 'leu' vão para 0, e volta-se para o estado inicial.

Segue a imagem da máquina de estados da tecla digitada:



A máquina que verifica a tecla, na verdade, possui apenas um estado, onde verificamos se o limite de erros já foi atingido, caso contrário, se 'teclou' = 1, é checado se a tecla é correta, se for, a letra é mostrada no display, se não, adiciona-se 1 ao contador de erros, ao final 'leu' = 1.

Segue a imagem da máquina de estados de verificação de tecla:



Segue os códigos utilizados no projeto:

1. Para a leitura de tecla digitada:

ps2_rx:

```
-- Listing 8.1

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
   --==================================================
   -- filter and falling edge tick generation for ps2c
   --==================================================
   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);

   --==================================================
   -- fsmd to extract the 8-bit data
   --==================================================
   -- 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;
```

kb_code:

```
-- Listing 8.3
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;

begin
   --=====================================================
   -- instantiation
   --=====================================================
   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, W=>W_SIZE)
```

```
      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);

   --=========================================================
   -- FSM to get the scan code after F0 received
   --=========================================================
   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;
   end process;
end arch;
```

fifo:

```
-- Listing 4.20

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity fifo is
   generic(
      B: natural:=8; -- number of bits
      W: natural:=4 -- number of address bits
   );
   port(
      clk, reset: in std_logic;
      rd, wr: in std_logic;
      w_data: in std_logic_vector (B-1 downto 0);
      empty, full: out std_logic;
      r_data: out std_logic_vector (B-1 downto 0)
   );
end fifo;

architecture arch of fifo is
   type reg_file_type is array (2**W-1 downto 0) of
        std_logic_vector(B-1 downto 0);
   signal array_reg: reg_file_type;
   signal w_ptr_reg, w_ptr_next, w_ptr_succ:
      std_logic_vector(W-1 downto 0);
   signal r_ptr_reg, r_ptr_next, r_ptr_succ:
      std_logic_vector(W-1 downto 0);
```

```
   signal full_reg, empty_reg, full_next, empty_next:
          std_logic;
   signal wr_op: std_logic_vector(1 downto 0);
   signal wr_en: std_logic;
begin
   --=================================================
   -- register file
   --=================================================
   process(clk,reset)
   begin
     if (reset='1') then
        array_reg <= (others=>(others=>'0'));
     elsif (clk'event and clk='1') then
        if wr_en='1' then
           array_reg(to_integer(unsigned(w_ptr_reg)))
                 <= w_data;
        end if;
     end if;
   end process;
   -- read port
   r_data <= array_reg(to_integer(unsigned(r_ptr_reg)));
   -- write enabled only when FIFO is not full
   wr_en <= wr and (not full_reg);

   --=================================================
   -- fifo control logic
   --=================================================
   -- register for read and write pointers
   process(clk,reset)
   begin
      if (reset='1') then
         w_ptr_reg <= (others=>'0');
         r_ptr_reg <= (others=>'0');
         full_reg <= '0';
         empty_reg <= '1';
      elsif (clk'event and clk='1') then
         w_ptr_reg <= w_ptr_next;
         r_ptr_reg <= r_ptr_next;
         full_reg <= full_next;
         empty_reg <= empty_next;
      end if;
   end process;

   -- successive pointer values
   w_ptr_succ <= std_logic_vector(unsigned(w_ptr_reg)+1);
   r_ptr_succ <= std_logic_vector(unsigned(r_ptr_reg)+1);

   -- next-state logic for read and write pointers
   wr_op <= wr & rd;
   process(w_ptr_reg,w_ptr_succ,r_ptr_reg,r_ptr_succ,wr_op,
           empty_reg,full_reg)
   begin
      w_ptr_next <= w_ptr_reg;
      r_ptr_next <= r_ptr_reg;
      full_next <= full_reg;
      empty_next <= empty_reg;
      case wr_op is
         when "00" => -- no op
         when "01" => -- read
            if (empty_reg /= '1') then -- not empty
               r_ptr_next <= r_ptr_succ;
               full_next <= '0';
               if (r_ptr_succ=w_ptr_reg) then
                  empty_next <='1';
```

```
               end if;
            end if;
         when "10" => -- write
            if (full_reg /= '1') then -- not full
               w_ptr_next <= w_ptr_succ;
               empty_next <= '0';
               if (w_ptr_succ=r_ptr_reg) then
                  full_next <='1';
               end if;
            end if;
         when others => -- write/read;
            w_ptr_next <= w_ptr_succ;
            r_ptr_next <= r_ptr_succ;
      end case;
   end process;
   -- output
   full <= full_reg;
   empty <= empty_reg;
end arch;
```

2. Para o funcionamento do display LCD:

```
----------------------------------------------------------------------
--  lcd.vhd -- general LCD testing program
----------------------------------------------------------------------
--  Author -- Guilherme Bergman
--       -- Bruno Dantas
--       -- João Ricardo
----------------------------------------------------------------------
--  This module is a test module for implementing read/write and
--  initialization routines for an LCD on the Digilab boards
----------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity lcd is
    Port ( LCD_DB: out std_logic_vector(7 downto 0);       --DB( 7 through 0)
           RS:out std_logic;                           --WE
           RW:out std_logic;                   --ADR(0)
         CLK:in std_logic;                     --GCLK2
         --ADR1:out std_logic;                       --ADR(1)
         --ADR2:out std_logic;                       --ADR(2)
         --CS:out std_logic;                   --CSC
```

```
         OE:out std_logic;                      --OE
         rst:in std_logic;          --BTN
         --rdone: out std_logic);              --WriteDone output to work with
DI05 test
            leds : out std_logic_vector (7 downto 0);
            ps2d, ps2c: in std_logic
            );
end lcd;

architecture Behavioral of lcd is

------------------------------------------------------------------
--  Component Declarations
COMPONENT kb_code port (
                        clk, reset: in  std_logic; --clk da fpga
                        ps2d, ps2c: in  std_logic;
                        rd_key_code: in std_logic; -- libera o buffer
                        key_code: out std_logic_vector(7 downto 0);--tecla no
buffer
                        kb_buf_empty: out std_logic -- tecla foi escrita no
buffer


                  );
      END COMPONENT kb_code;
------------------------------------------------------------------

------------------------------------------------------------------
--  Local Type Declarations
------------------------------------------------------------------
--  Symbolic names for all possible states of the state machines.

      --LCD control state machine
      type mstate is (
            stFunctionSet,                          --Initialization states
            stDisplayCtrlSet,
            stDisplayClear,
            stPowerOn_Delay,                        --Delay states
            stFunctionSet_Delay,
            stDisplayCtrlSet_Delay,
            stDisplayClear_Delay,
            stInitDne,                        --Display charachters and perform
standard operations
```

```
            stActWr,
            stCharDelay                         --Write delay for operations
            --stWait                            --Idle state
      );

      --Write control state machine
      type wstate is (
            stRW,                               --set up RS and RW
            stEnable,                           --set up E
            stIdle                                    --Write data on DB(0)-DB(7)
      );
      type jestados is (
            jEspera,
            jAcerto,
            jErro,
            jPerde
      );
      type mleitor is (
            minicial,
            mmeio,
            mfinal
      );

-----------------------------------------------------------------------
--  Signal Declarations and Constants
-----------------------------------------------------------------------
      --These constants are used to initialize the LCD pannel.

      --FunctionSet:
            --Bit 0 and 1 are arbitrary
            --Bit 2:  Displays font type(0=5x8, 1=5x11)
            --Bit 3:  Numbers of display lines (0=1, 1=2)
            --Bit 4:  Data length (0=4 bit, 1=8 bit)
            --Bit 5-7 are set
      --DisplayCtrlSet:
            --Bit 0:  Blinking cursor control (0=off, 1=on)
            --Bit 1:  Cursor (0=off, 1=on)
            --Bit 2:  Display (0=off, 1=on)
            --Bit 3-7 are set
      --DisplayClear:
            --Bit 1-7 are set
      signal clkCount:std_logic_vector(5 downto 0);
      signal activateW:std_logic:= '0';                         --Activate
```

```
Write sequence
      signal count:std_logic_vector (16 downto 0):= "00000000000000000";     --
15 bit count variable for timing delays
      signal delayOK:std_logic:= '0';                                        --High
when count has reached the right delay time
      signal OneUSClk:std_logic;                                       --Signal is
treated as a 1 MHz clock
      signal stCur:mstate:= stPowerOn_Delay;    --LCD control state machine
      signal jAtual:jestados:= jEspera;
      signal jNext:jestados;
      signal stNext:mstate;
      signal matual: mleitor:= minicial;
      signal stCurW:wstate:= stIdle;                                         --Write
control state machine
      signal stNextW:wstate;
      signal writeDone:std_logic:= '0';         --Command set finish
      signal liberaBuf : std_logic := '0';
      signal keyRead : std_logic_vector (7 downto 0):= "00000000";
      signal keybuffer : std_logic_vector (7 downto 0);
      signal bufEmpty : std_logic ;
      signal errocount: UNSIGNED (3 DOWNTO 0):= "0000";
      signal teclou : std_logic := '0';
      signal leu : std_logic := '0';

      type SHOW_T is array(integer range 0 to 5) of std_logic_vector(9 downto
0);
      signal show : SHOW_T := (
            0 => "10"&X"2E",
            1 => "10"&X"2E",
            2 => "10"&X"2E",
            3 => "10"&X"2E",
            4 => "10"&X"2E",
            5 => "10"&X"2E"
            );

      --signal escritura: std_logic_vector (23 downto 0) := ""

      type LCD_CMDS_T is array(integer range 0 to 13) of std_logic_vector(9
downto 0);
      signal LCD_CMDS : LCD_CMDS_T := (
                                    0 => "00"&X"3C",            --Function Set
                                  1 => "00"&X"0C",              --Display ON,
Cursor OFF, Blink OFF
```

```
                                    2 => "00"&X"01",                --Clear Display
                                    3 => "00"&X"02",                --return home

                                    4 => "10"&X"48",                --H
                                    5 => "10"&X"65",                --e
                                    6 => "10"&X"6C",                --l
                                    7 => "10"&X"6C",                --l
                                    8 => "10"&X"6F",                --o
                                    9 => "10"&X"20",                --space
                                    10 => "10"&X"46",               --F
                                    11 => "10"&X"72",               --r
                                       12 => "10"&X"72",                  --r
                                       13 => "10"&X"72");

      signal lcd_cmd_ptr : integer range 0 to LCD_CMDS'HIGH + 1 := 0;
begin
       leds(0) <= keyRead(0);
       leds(1) <= keyRead(1);
       leds(2) <= keyRead(2);
       leds(3) <= keyRead(3);
       leds(4) <= keyRead(4);
       leds(5) <= keyRead(5);
       leds(6) <= keyRead(6);
       leds(7) <= keyRead(7);

       LCD_CMDS(0) <= "00"&X"3C";
       LCD_CMDS(1) <= "00"&X"0C";
       LCD_CMDS(2) <= "00"&X"01";
       LCD_CMDS(3) <= "00"&X"02";

       LCD_CMDS(4) <= show(0);
       LCD_CMDS(5) <= show(1);
       LCD_CMDS(6) <= show(2);
       LCD_CMDS(7) <= show(3);
       LCD_CMDS(8) <= show(4);
       LCD_CMDS(9) <= show(5);
       LCD_CMDS(10) <= show(3);



       LCD_CMDS(11) <= "1000100000";

       LCD_CMDS(12) <= "10"&"0011"&(std_logic_vector(errocount));
       LCD_CMDS(13) <= "00"&X"02";
```

```
kbc: kb_code port map (CLK, rst, ps2d, ps2c, liberaBuf, keybuffer, bufEmpty);

      --  This process counts to 50, and then resets.  It is used to divide the
clock signal time.
      process (CLK, oneUSClk)
            begin
                  if (CLK = '1' and CLK'event) then
                        clkCount <= clkCount + 1;
                  end if;
            end process;
      --  This makes oneUSClock peak once every 1 microsecond

      oneUSClk <= clkCount(5);
      --  This process incriments the count variable unless delayOK = 1.
      process (oneUSClk, delayOK)
            begin
                  if (oneUSClk = '1' and oneUSClk'event) then
                        if delayOK = '1' then
                              count <= "00000000000000000";
                        else
                              count <= count + 1;
                        end if;
                  end if;
            end process;

      --This goes high when all commands have been run
      writeDone <= '1' when (lcd_cmd_ptr = LCD_CMDS'HIGH)
            else '0';
      --rdone <= '1' when stCur = stWait else '0';
      --Increments the pointer so the statemachine goes through the commands
      process (lcd_cmd_ptr, oneUSClk)
            begin
                  if (oneUSClk = '1' and oneUSClk'event) 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 or stNext =
stPowerOn_Delay then
                              lcd_cmd_ptr <= 0;
                        elsif teclou = '1' then
                              lcd_cmd_ptr <= 3;
                        else
```

```
                        lcd_cmd_ptr <= lcd_cmd_ptr;
                  end if;
            end if;
      end process;

      --  Determines when count has gotten to the right number, depending on the
state.

      delayOK <= '1' when ((stCur = stPowerOn_Delay and count =
"00100111001010010") or                  --20050
                        (stCur = stFunctionSet_Delay and count =
"00000000000110010") or --50
                        (stCur = stDisplayCtrlSet_Delay and count =
"00000000000110010") or --50
                        (stCur = stDisplayClear_Delay and count =
"00000011001000000") or --1600
                        (stCur = stCharDelay and count =
"1111111111111111"))            --Max Delay for character writes and shifts
                        --(stCur = stCharDelay and count =
"00000000000100101"))         --37  This is proper delay between writes to ram.
      else  '0';

      -- This process runs the LCD status state machine
      process (oneUSClk, rst)
            begin
                  if oneUSClk = '1' and oneUSClk'Event then
                        if rst = '1' then
                              stCur <= stPowerOn_Delay;
                        else
                              stCur <= stNext;
                        end if;
                  end if;
            end process;


      --  This process generates the sequence of outputs needed to initialize
and write to the LCD screen
      process (stCur, delayOK, writeDone, lcd_cmd_ptr)
            begin

                  case stCur is

                        --  Delays the state machine for 20ms which is needed
```

```
for proper startup.
                        when stPowerOn_Delay =>
                              if delayOK = '1' then
                                    stNext <= stFunctionSet;
                              else
                                    stNext <= stPowerOn_Delay;
                              end if;
                              RS <= LCD_CMDS(lcd_cmd_ptr)(9);
                              RW <= LCD_CMDS(lcd_cmd_ptr)(8);
                              LCD_DB <= LCD_CMDS(lcd_cmd_ptr)(7 downto 0);
                              activateW <= '0';

                        -- This issuse the function set to the LCD as follows
                        -- 8 bit data length, 2 lines, font is 5x8.
                        when stFunctionSet =>
                              RS <= LCD_CMDS(lcd_cmd_ptr)(9);
                              RW <= LCD_CMDS(lcd_cmd_ptr)(8);
                              LCD_DB <= LCD_CMDS(lcd_cmd_ptr)(7 downto 0);
                              activateW <= '1';
                              stNext <= stFunctionSet_Delay;

                        --Gives the proper delay of 37us between the function
set and
                        --the display control set.
                        when stFunctionSet_Delay =>
                              RS <= LCD_CMDS(lcd_cmd_ptr)(9);
                              RW <= LCD_CMDS(lcd_cmd_ptr)(8);
                              LCD_DB <= LCD_CMDS(lcd_cmd_ptr)(7 downto 0);
                              activateW <= '0';
                              if delayOK = '1' then
                                    stNext <= stDisplayCtrlSet;
                              else
                                    stNext <= stFunctionSet_Delay;
                              end if;

                        --Issuse the display control set as follows
                        --Display ON,  Cursor OFF, Blinking Cursor OFF.
                        when stDisplayCtrlSet =>
                              RS <= LCD_CMDS(lcd_cmd_ptr)(9);
                              RW <= LCD_CMDS(lcd_cmd_ptr)(8);
                              LCD_DB <= LCD_CMDS(lcd_cmd_ptr)(7 downto 0);
                              activateW <= '1';
                              stNext <= stDisplayCtrlSet_Delay;
```

```
                        --Gives the proper delay of 37us between the display
control set
                        --and the Display Clear command.
                        when stDisplayCtrlSet_Delay =>
                              RS <= LCD_CMDS(lcd_cmd_ptr)(9);
                              RW <= LCD_CMDS(lcd_cmd_ptr)(8);
                              LCD_DB <= LCD_CMDS(lcd_cmd_ptr)(7 downto 0);
                              activateW <= '0';
                              if delayOK = '1' then
                                    stNext <= stDisplayClear;
                              else
                                    stNext <= stDisplayCtrlSet_Delay;
                              end if;

                        --Issues the display clear command.
                        when stDisplayClear     =>
                              RS <= LCD_CMDS(lcd_cmd_ptr)(9);
                              RW <= LCD_CMDS(lcd_cmd_ptr)(8);
                              LCD_DB <= LCD_CMDS(lcd_cmd_ptr)(7 downto 0);
                              activateW <= '1';
                              stNext <= stDisplayClear_Delay;

                        --Gives the proper delay of 1.52ms between the clear
command
                        --and the state where you are clear to do normal
operations.
                        when stDisplayClear_Delay =>
                              RS <= LCD_CMDS(lcd_cmd_ptr)(9);
                              RW <= LCD_CMDS(lcd_cmd_ptr)(8);
                              LCD_DB <= LCD_CMDS(lcd_cmd_ptr)(7 downto 0);
                              activateW <= '0';
                              if delayOK = '1' then
                                    stNext <= stInitDne;
                              else
                                    stNext <= stDisplayClear_Delay;
                              end if;

                        --State for normal operations for displaying characters,
changing the
                        --Cursor position etc.
                        when stInitDne =>
                              RS <= LCD_CMDS(lcd_cmd_ptr)(9);
```

```
                        RW <= LCD_CMDS(lcd_cmd_ptr)(8);
                        LCD_DB <= LCD_CMDS(lcd_cmd_ptr)(7 downto 0);
                        activateW <= '0';
                        stNext <= stActWr;

                  when stActWr =>
                        RS <= LCD_CMDS(lcd_cmd_ptr)(9);
                        RW <= LCD_CMDS(lcd_cmd_ptr)(8);
                        LCD_DB <= LCD_CMDS(lcd_cmd_ptr)(7 downto 0);
                        activateW <= '1';
                        stNext <= stCharDelay;

                  --Provides a max delay between instructions.
                  when stCharDelay =>
                        RS <= LCD_CMDS(lcd_cmd_ptr)(9);
                        RW <= LCD_CMDS(lcd_cmd_ptr)(8);
                        LCD_DB <= LCD_CMDS(lcd_cmd_ptr)(7 downto 0);
                        activateW <= '0';
                        if delayOK = '1' then
                              stNext <= stInitDne;
                        else
                              stNext <= stCharDelay;
                        end if;
            end case;

      end process;

--This process runs the write state machine
process (oneUSClk, rst)
      begin
            if oneUSClk = '1' and oneUSClk'Event then
                  if rst = '1' then
                        stCurW <= stIdle;
                  else
                        stCurW <= stNextW;
                  end if;
            end if;
      end process;

--This genearates the sequence of outputs needed to write to the LCD
screen
process (stCurW, activateW)
      begin
```

```
                case stCurW is
                        --This sends the address across the bus telling the DIO5
that we are
                        --writing to the LCD, in this configuration the
adr_lcd(2) controls the
                        --enable pin on the LCD
                        when stRw =>
                                OE <= '0';
                                --CS <= '0';
                                --ADR2 <= '1';
                                --ADR1 <= '0';
                                stNextW <= stEnable;

                        --This adds another clock onto the wait to make sure
data is stable on
                        --the bus before enable goes low.  The lcd has an active
falling edge
                        --and will write on the fall of enable
                        when stEnable =>
                                OE <= '0';
                                --CS <= '0';
                                --ADR2 <= '0';
                                --ADR1 <= '0';
                                stNextW <= stIdle;

                        --Waiting for the write command from the instuction
state machine
                        when stIdle =>
                                --ADR2 <= '0';
                                --ADR1 <= '0';
                                --CS <= '1';
                                OE <= '1';
                                if activateW = '1' then
                                        stNextW <= stRw;
                                else
                                        stNextW <= stIdle;
                                end if;
                        end case;
        end process;
        process(rst,oneUSClk,teclou,keyread)
                begin
                        if rst = '1' then
```

```
                        show(0) <= "10"&X"2E";
                        show(1) <= "10"&X"2E";
                        show(2) <= "10"&X"2E";
                        show(3) <= "10"&X"2E";
                        show(4) <= "10"&X"2E";
                        show(5) <= "10"&X"2E";
                        errocount <= "0000";

                    elsif oneUSClk = '1' and oneUSClk'Event then
                                if errocount >= 7 then
                                      show(0) <= "10"&X"4B";
                                      show(1) <= "10"&X"4B";
                                      show(2) <= "10"&X"4B";
                                      show(3) <= "10"&X"4B";
                                      show(4) <= "10"&X"4B";
                                      show(5) <= "10"&X"4B";
                                elsif teclou = '1' then
                                      case keyread is
                                            when "00011100" =>
                                                  show(0) <=
"10"&X"41";
                                                  show(1 to 5) <=
show(1 to 5);
                                            when "00110010" =>
                                                  show(1) <=
"10"&X"42";
                                                  show(0) <= show(0);
                                                  show(2 to 5) <=
show(2 to 5);
                                            when "00100011" =>
                                                  show(2) <=
"10"&X"44";
                                                  show(0 to 1) <=
show(0 to 1);
                                                  show(3 to 5) <=
show(3 to 5);
                                            when "00111100" =>
                                                  show(3) <=
"10"&X"55";
                                                  show(0 to 2) <=
show(0 to 2);
                                                  show(4 to 5) <=
show(4 to 5);
```

```
                                                      when "00011010" =>
                                                            show(4) <=
"10"&X"5A";
                                                            show(0 to 3) <=
show(0 to 3);
                                                            show(5) <= show(5);
                                                      when "01000011" =>
                                                            show(5) <=
"10"&X"49";
                                                            show(0 to 4) <=
show(0 to 4);
                                                      when others =>
                                                            errocount <=
errocount + 1;
                                                            show(0 to 5)<= show(0
to 5);
                                                end case;
                                                leu <= '1';
                                          else
                                                show<=show;
                                          end if;
                        end if;
                  end process;
      process(oneUSClk)
      begin
      if oneUSClk = '1' and oneUSClk'Event then
            case matual is
                  when minicial=>
                        if bufempty = '0' then
                              matual <= mmeio;
                        end if;

                  when mmeio =>
                        if leu <= '1' then
                              matual <= mfinal;
                        end if;

                  when mfinal =>
                        matual <= minicial;
            end case;
      end if;
      end process;
```

```
      process(oneUSClk)
      begin
      if oneUSClk = '1' and oneUSClk'Event then
            case matual is
                  when minicial =>
                        liberaBuf <= '0';
                  when mmeio =>
                        teclou <= '1';
                        keyRead <= keybuffer;
                  when mfinal =>
                        teclou <= '0';
                        leu<= '0';
                        liberaBuf <= '1';
            end case;
      end if;
      end process;


end Behavioral;
```

3. O jogo própriamente dito:

```
----------------------------------------------------------------------------------
--
-- Company:
-- Engineer:
--
-- Create Date:    16:28:18 05/21/2019
-- Design Name:
-- Module Name:    Leitor_Tecla - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
```

```
--use UNISIM.VComponents.all;

entity Leitor_Tecla is
        port (
                        clk, reset: in  std_logic; --clk da fpga
                        ps2d, ps2c: in  std_logic;
                        leds : out std_logic_vector (7 downto 0)
                        teclou : out std_logic;
                );
end Leitor_Tecla;

architecture Behavioral of Leitor_Tecla is

        COMPONENT kb_code port (
                                clk, reset: in  std_logic; --clk da fpga
                                ps2d, ps2c: in  std_logic;
                                rd_key_code: in std_logic; -- libera o buffer
                                key_code: out std_logic_vector(7 downto 0);--
tecla no buffer
                                kb_buf_empty: out std_logic -- tecla foi escrita
no buffer
                        );
        END COMPONENT kb_code;

        type estados is (
        eInicial,
        eMeio,
        eFinal
        );

        signal eAtual : estados := eInicial;
        signal eProximo : estados;
        signal liberaBuf : std_logic := '0';
        signal keyRead : std_logic_vector (7 downto 0):= "00000000";
        signal keybuffer : std_logic_vector (7 downto 0);
        signal bufEmpty : std_logic ;

        signal clkReduzido : std_logic := '0';

begin
kbc: kb_code port map (clk, reset, ps2d, ps2c, liberaBuf, keybuffer, bufEmpty);
        leds <= keyRead;


        process(clk)
                variable contagem : UNSIGNED (5 downto 0) := "000000";
                BEGIN
                        if (clk = '1' and clk'event) then
                                if (contagem >= 9) then
                                        contagem := "000000";
                                        clkReduzido<= not clkReduzido;
                                else
                                        contagem := contagem + 1;
                                end if;
                        end if;
        end process;

        process(clkReduzido, eAtual, bufEmpty)
        begin
                if (clkReduzido = '1' and clkReduzido'event) then
                        if eAtual = eInicial then
                                if bufEmpty = '0' then
                                        eAtual <= eMeio;
                                end if;
```

```
                        end if;
                        if eAtual = eMeio then
                                eAtual <= efinal;
                        end if;
                        if eAtual = eFinal then
                                eAtual <= eInicial;
                        end if;
                end if;
        end process;
        process(clkReduzido)
        begin
                if eAtual = eInicial then
                        liberaBuf <= '0';
                end if;
                if eAtual = eMeio then
                        keyRead <= keybuffer;
                end if;
                if eAtual = eFinal then
                        liberaBuf <= '1';
                end if;
        end process;
end Behavioral;
```

Durante a execução do projeto, foi necessário ter o cuidado com o equipamento de entrada utilizado (teclado) e a sua interação com o desenvolvimento do código, pois, qualquer pequeno detalhe que não fosse tratado, poderia levar a entender que o código estava errado, e não o dispositivo. Para tal, utilizamos como debug os 8 led's existentes na placa da Xilinx, o que acabou levando o projeto a ser concluído sem nenhuma simulação usando o compilador. Desta forma, a seção que tratava da simulação de cada módulo e do circuito completo não foi preenchida.

III. Conclusão:

Concluímos que o trabalho é um exercício bem interessante sobre como pensar em uma máquina de estados pensando em nível de bits em conjunto da montagem de circuito em VHDL. Além disso, pôde-se notar que a codificação da utilização da interface, teve uma preocupação bastante básica comparado ao nível que complexidade que poderia ser remetida ao projeto, vide a implementação de um possível sistema de escolha de palavras.

IV. Referências bibliográficas:

Spartan-3AN User Guide

https://www.gta.ufrj.br/ensino/EEL480/index.html

https://github.com/DantasB/Hangman-Game-VHDL