UNIVERSIDADE FEDERAL DO PARANÁ

DIEGO LUIZ MOLINA ERIK NAYAN ONORIO DOS SANTOS

TRABALHO DE MICROELETRÔNICA 1 PROJETO APLICATIVO

DIEGO LUIZ MOLINA - GRR20131145 ERIK NAYAN ONORIO DOS SANTOS - GRR20131349

TRABALHO DE MICROELETRÔNICA 1 PROJETO APLICATIVO

Trabalho de laboratório orientado pela professora Doutora Sibilla França, da disciplina de Microeletrônica I, do curso de Engenharia Elétrica da Universidade Federal do Paraná.

SUMÁRIO

INTRODUÇ	ÇÃO	4
OBJETIVO		4
1. ELABO	RAÇÃO DA MÁQUINA DE ESTADOS	5
2. DESEN	NVOLVIMENTO	6
2.1. JO	GO GENIUS	6
2.1.1.	Rom_characters.vhd	6
2.1.2.	Global_pack.vhd	7
2.1.3.	Genius.vhd	9
2.1.4.	Decoder.vhd	14
2.1.5.	VGA.vhd	17
2.2. AN	ÁLISE DOS RESULTADOS	20
2.2.1.	Testbench.vhd	20
2.2.2.	Simulações	21
2.2.3	Análise de utilização de recursos	23
CONCLUS	ÕES	24
REFERÊN	CIAS BIBLIOGRÁFICAS	25
ANEXOS		26

INTRODUÇÃO

Este projeto aplicativo fundamenta-se no conhecimento adquirido em sala de aula, na matéria de Microeletrônica 1, do curso de Engenharia Elétrica da Universidade Federal do Paraná, matéria ministrada pela Professora Doutora Sibilla Franca.

A proposta deste relatório é documentar e formalizar o projeto desenvolvido para a conclusão da matéria. O qual tem por objetivo aplicar todos os conceitos vistos e apresentados em sala de aula sobre VHDL. Serão realizadas simulações prévias para verificar o funcionamento do código, também será implementado o circuito e feito *upload* para o kit de desenvolvimento.

Para executar esta tarefa será utilizado um computador, cabo USB e uma placa FPGA Nexys 2 Spartan 3 e os softwares Xilinx ISE e Digilent Adept 2.

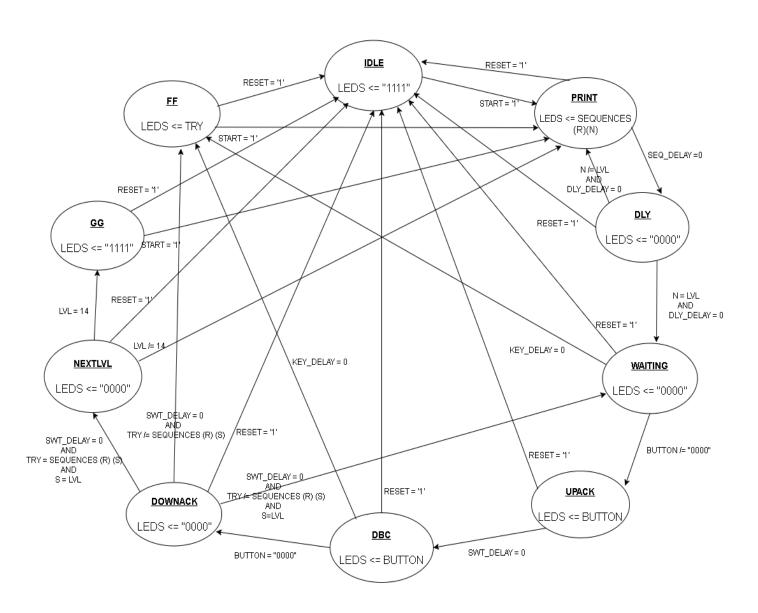
OBJETIVO

O objetivo deste projeto é utilizar o software Xilinx ISE para criar o código em VHDL que cumpra o desafio proposto.

Neste projeto a tarefa é implementar o jogo Genius em FPGA. O projeto deverá cumprir os seguintes requisitos: Ao ligar o dispositivo, uma mensagem de inicialização deverá ser apresentada nos displays de 7 segmentos: "PROJETO APLICATIVO - ED". Um rolamento das letras através dos 4 displays deverá garantir a visualização da mensagem inicial; O início do jogo se dará através do acionamento de um botão do kit; Para representar as quatro cores deverá ser utilizado os quatro leds do kit NEXYS2; Disponibilizar a primeira sequência e verificar se o jogador pressionou o botão correto (por exemplo, LED0 - SW0) dentro do tempo máximo de 5 segundos; Repetir o primeiro sinal e acrescentar mais um (utilizar 14 sequências). Verificar a sequência inserida pelo jogador; A cada acerto, o jogador soma 01 ponto. Esta pontuação deverá ser mostrada nos displays de 7 segmentos e em um monitor com resolução de 800x600 (usar saída VGA do kit); No monitor, também deverá ser exibida a seguência luminosa, utilizando as cores verde, amarelo, azul e vermelho; O acionamento de um mecanismo mecânico do kit deverá permitir a reinicialização do dispositivo de forma a iniciar-se novo; O botão INICIAR não reinicia o jogo; Utilizar dois switches para selecionar o nível de dificuldade do jogo (00 – fácil, 01 – médio, 10 – difícil); Se o jogador acertar todas as sequências, uma mensagem deve aparecer nos displays de 7 segmentos indicando que o jogador ganhou como, por exemplo, "PARABENS"; A mensagem também deve aparecer no monitor (ou as iniciais da mensagem); Se o jogador errar alguma das sequências, uma mensagem deve aparecer nos displays de 7 segmentos indicando que o jogador perdeu como, por exemplo, "GAME OVER". Um rolamento das letras através dos 4 displays deverá garantir a visualização da mensagem final.

1. ELABORAÇÃO DA MÁQUINA DE ESTADOS

A fim de definir os estados, condições de transição e valores de saída para cada momento do jogo, o diagrama a seguir foi desenvolvido exemplificando o funcionamento do projeto, buscando cumprir o objetivo proposto.



Buscando melhor organizar o código e a máquina de estados em si, foram criados um total de 10 estados responsáveis por controlar o andamento do jogo e decidir, mediante a alterações na entrada, qual será o próximo estado do sistema. Para não sobrecarregar um estado com muitas funções, tal divisão foi realizada de forma que cada estado possuisse um objetivo específico bem definido e assim pudesse ser melhor compreendido e testado durante as etapas de desenvolvimento e simulação.

2. DESENVOLVIMENTO

2.1. JOGO GENIUS

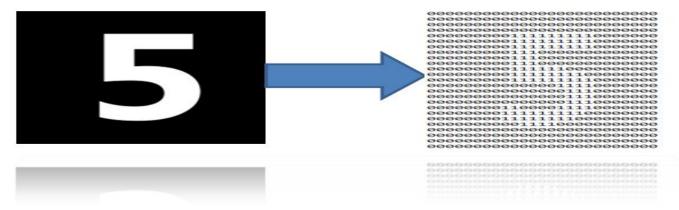
Para o desenvolvimento do projeto aplicativo do jogo Genius, foram implementados 2 packages, 3 modules, 1 testbench e 1 implementation constraints file. Os trechos mais relevantes de alguns desses códigos serão comentados neste capítulo. Todos estes códigos VHDL encontram-se na íntegra na sessão de anexos deste relatório.

2.1.1. Rom_characters.vhd

```
1 - Package que define as constantes com o desenho de cada um dos caracteres utilizados
  -- Números: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
3 -- Letras: F, G
 -- Constantes de 200x200 pixels
6 library IEEE;
 use IEEE.STD LOGIC 1164.all;
7
8
9 package rom characters is
10
   constant h: integer := 200;
11
   constant v: integer := 200;
12
   type char data is array (0 to v-1) of std logic vector (0 to h-1);
13
14
   constant zero: char data := (
1.5
```

Este *package* define um *type* chamado "char_data" que cria um arranjo de vetores, visando criar uma memória com o desenho de cada um dos caracteres utilizados na exibição do placar e de siglas no VGA (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, G, F). Cada caractere possui 200x200 *pixels* e foi primeiramente desenhado, para então ser convertido em um desenho em texto formado por bits '0' e '1', que representam as áreas em preto e em branco, respectivamente, a serem mostradas na tela de jogo.

O esquema a seguir exemplifica este processo, que foi realizado com auxílio do software Gimp^[4] e o site Dcode^[5].



2.1.2. Global_pack.vhd

Parte 1

```
1 -- Package global que implementa os tipos necessários no projeto, bem como procedures e constantes
                  library IEEE;
     3
                    use IEEE.STD LOGIC 1164.all:
                    use work.rom_characters.all;
                   package global pack is
    8
                                    type states is (idle, print, dly, waiting, upack, dbc, downack, nextlvl, gg, ff); -- Estados da máquina
     9
                                  type pa_ed is array (22 downto 0) of std_logic_vector (6 downto 0); -- Arranjo da mensagem inicial type score is array (3 downto 0) of std_logic_vector (6 downto 0); -- Arranjo da pontuação type enable is array (3 downto 0) of std_logic_vector (3 downto 0); -- Arranjo dos estados dos anodos type sequence is array (14 downto 0) of std_logic_vector (3 downto 0); -- Arranjo de uma sequência
 10
 11
 12
 13
 14
                                    -- Constantes que definem as sequências possiveis de 14 cores para o jogo
 15
                                  constant seq0: sequence := ("0001","1000","0001","0010","0100","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","100
 16
 17
 18
                                    constant seq1: sequence := ("0001","1000","0001","0010","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","010
 19
                                   "1000","0001","1000","0001","0100","0010","1000", "0000");
 20
 21
                                    constant seq2: sequence := ("0010","1000","0001","0010","0100","0010","1000",
 22
                                    "0100","1000","0001","0001","0010","0100","0000");
 23
 24
                                    constant seq3: sequence := ("0100","0010","0001","0010","1000","1000","0100",
"0001","0010","1000","0001","0010","0100","0100", "0000");
 25
 26
 27
                                   constant seq4: sequence := ("0001","0001","1000","0010","0100","1000","1000","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","010
 28
 29
30
                                    constant seq5: sequence := ("0100","1000","0010","0010","0100","1000","1000","1000","
31
                                     "0100", "0010", "0001", "1000", "0010", "0001", "00001", "0000");
 32
 33
                                    constant seq6: sequence := ("1000","0010","0010","0010","0100","1000","0001",
 34
                                       "0100","1000","0100","0001","0010","0100","0100", "0000");
35
36
                                    constant seq7: sequence := ("0001","1000","0010","0010","0100","1000","1000","0100","0001","0010","0001","1000","1000","1000","1000");
38
 39
                                    constant seq8: sequence := ("0010","0001","0100","0001","0010","1000","0100",
 40
                                    "0100", "1000", "0001", "1000", "0010", "0100", "0010", "0000");
 41
 42
                                    constant seq9: sequence := ("0100","1000","0001","0010","0010","1000","1000","1000","0001","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","01000","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","0100","01
 43
 44
 45
                                    type mult_seq is array (9 downto 0) of sequence; -- Arranjo das 10 sequências pré-definidas
 46
 47
 48
                                   constant sequences: mult_seq := (seq9, seq8, seq7, seq6, seq5, seq4, seq3, seq2, seq1, seq0);
 49
                                          - Declaração da procedure que imprime os caracteres na região superior do monitor
 50
 51
                                 procedure characters (h_count, v_count: in integer;
                                                                                                                                                 first_char, secon_char: in char_data;
 52
                                                                                                                                                red: out std_logic_vector (2 downto 0);
green: out std_logic_vector (2 downto 0);
 53
 54
                                                                                                                                                blue: out std_logic_vector (1 downto 0));
 55
 57 end global_pack;
```

O *global_pack* trata-se de um *package* criado com o intuito de definir alguns novos *types* necessários ao projeto, bem como reunir e organizar constants e disponibilizar uma procedure que é utilizada muitas vezes dentro do component responsável pela parte de VGA do projeto.

(continuação)

- O type "states" define os estados da máquina que controlaram o andamento do jogo.
- O *type* "pa_ed" define um arranjo que corresponde ao código para mostrar a mensagem inicial "Projeto Aplicativo ED" nos displays de sete segmentos.
- O *type* "score" é um arranjo para exibição da pontuação do jogador nos displays de sete segmentos.
- O *type* "enable" é um arranjo para multiplexação dos anodos dos displays, já que os catodos dos mesmos são curto-circuitados.
- O *type* "sequence" é um arranjo que cria um vetor em que cada posição é um nibble, correspondente a uma das cores e ao estado dos LEDs na saída. Este type implementa as sequências do jogo.

Por último, o *type* "sequences" é um arranjo do *type* "sequence" criado anteriormente, o que cria uma matriz com dez possibilidades de sequências, que foram criadas e definidas como constants. Sendo assim, ganha-se uma maior dinâmica no jogo já que o jogador não ficará preso a uma única sequência de cores que seria facilmente memorizada.

Parte 2

```
59 package body global pack is
60
        -- Procedure que recebe os valores de caracteres e estabelece a lógica para impressão na tela
61
        procedure characters (h_count, v_count: in integer;
62
                                 first_char, secon_char: in char_data;
red: out std_logic_vector (2 downto 0);
green: out std_logic_vector (2 downto 0);
63
64
65
                                 blue: out std_logic_vector (1 downto 0)) is
66
       begin
68
69
           if (h_count>250 and h_count<400 and v_count>50 and v_count<250) then
70
               if (first_char(v_count-50)(h_count-225)='1') then
71
                  red := (others => '1'):
72
                  green := (others => '1');
73
                  blue := (others => '1');
74
75
               else
                  red := (others => '0');
76
                  green := (others => '0');
77
                  blue := (others => '0');
78
               end if;
           elsif (h count>400 and h count<550 and v count>50 and v count<250) then
               if (secon_char(v_count-50)(h_count-375)='1') then
                  red := (others => '1');
green := (others => '1');
82
83
                  blue := (others => '1');
84
               else
85
                  red := (others => '0');
86
                  green := (others => '0');
87
                  blue := (others => '0');
88
               end if;
89
           end if:
90
91
        end characters;
94 end package body;
```

A procedure descrita no package body acima realiza a função de posicionar números e letras no monitor, para exibição do placar e das siglas de vitória e derrota. Para tal, ela requer quais serão os dois caracteres e qual é contagem horizontal e vertical do VGA.

2.1.3. Genius.vhd

Parte 1 6 library ieee; use ieee.std logic 1164.all; 8 use work.global pack.all; entity Genius is 10 11 generic (d: integer := 5000000; -- Tempo de 100 ms 12 p: integer := 500000; -- Tempo de 10 ms 13 e: integer := 50000000; -- Tempo de 1 s 14 m: integer := 25000000; -- Tempo de 500 ms 15 h: integer := 10000000; -- Tempo de 200 ms 16 T: integer := 250000000); -- Tempo de 5 s 17 port (clk, start, reset: in std_logic; -- Clock, Start e Reset 18 diff: in std_logic_vector (1 downto 0); -- Chaves seletoras de dificuldade 19 button: in std logic vector (3 downto 0); -- Push-buttons do jogo 20 display: out std_logic_vector (6 downto 0); -- Segmentos dos displays
anode: out std_logic_vector (3 downto 0); -- Anodos dos displays 21 22 leds: out std_logic_vector (3 downto 0); -- Leds de visualização 23 h_sync: out std_logic; -- Sinal de sincronização horizontal v_sync: out std_logic; -- Sinal de sincronização vertical 24 25 red: out std logic vector (2 downto 0); -- Bits vermelho do VGA 26 27 green: out std logic vector (2 downto 0); -- Bits verde do VGA blue: out std_logic_vector (1 downto 0)); -- Bits azul do VGA 28 29 30 end Genius;

Neste trecho inicial do código principal, são declarados alguns valores do tipo GENERIC, para definição de constantes de tempo a serem utilizadas no projeto. São eles:

- d Tempo utilizado para promover um delay entre a exibição de uma cor e outra (100ms):
- p Tempo de debouncing no reconhecimento das transições dos push-buttons (10ms);
- e Tempo de exibição de cada cor para a dificuldade fácil (1s);
- m Tempo de exibição de cada cor para a dificuldade média (0,5s);
- h Tempo de exibição de cada cor para a dificuldade difícil (0,2s);
- T Tempo limite de espera por uma entrada do jogador (5s).

Além disso, as entradas e saídas do jogo são declaradas, de acordo com as especificações do kit de desenvolvimento e do projeto:

ENTRADAS: Clock, chave de start, chave de reset, chaves seletoras da dificuldade do jogo e botões para repetição da sequência por parte do jogador (1 botão para cada cor).

SAÍDAS: Catodos dos segmentos dos displays, anodos dos displays, LEDs (1 para cada cor), sincronização horizontal do VGA, sincronização vertical do VGA e os bits para cada uma das cores do RGB (3 para vermelho e verde, 2 para o azul, cujo o olho humano é menos sensível).

Parte 2

```
69 begin
70
       -- Processo que implementa a lógica de estados do jogo
71
       process(clk, pr state, reset, start, seq delay, diff)
72
73
       begin
74
       if (reset = '1') then
75
76
          pr state<=idle; -- Estado inicial em espera do jogo
77
78
       elsif (rising edge(clk)) then
79
80
          -- Verificação da posição das chaves de dificuldade e atribuição ao sinal
81
82
          if ( diff="00" ) then
             seq_delay<=e;
83
          elsif ( diff="01" ) then
84
85
             seq_delay<=m;
          elsif ( diff="10" ) then
86
             seq_delay<=h;
87
          else
88
89
            seq delay<=e;
          end if;
91
92
          -- Case dos estados da máquina
93
          case pr state is
```

Devido a dificuldades encontradas na implementação prática de um código de máquinas de estado com dois process no kit de desenvolvimento, decidiu-se por implementar a máquina de estados utilizando somente um process.

No trecho em destaque, encontra-se a lógica que verifica se o reset do jogo foi acionado e, em caso, positivo, passa o estado da máquina para idle. Caso o reset não esteja pressionado e ocorra um pulso de subida do clock, verifica-se o estado das chaves seletoras de dificuldade e dependendo do seu estado o valor de tempo adequado é repassado para o signal responsável pela contagem. Isto permite que a dificuldade do jogo seja alterada a qualquer momento, caso o jogador julgue necessário.

```
Estado idle
            -- Estado inicial de espera da máquina
 9.5
            when idle => leds<="1111";
 96
                           nibble<="11111";
 97
                           lv1<=1;
 98
 99
                           n <= 0;
                           s<=1;
100
                           r<=0;
101
102
                           if ( start='1' ) then
103
104
                             pr state<=print;
105
                           end if;
```

O estado *idle* é o estado inicial do jogo, no qual se aguarda que o usuário mude o modo da chave start para '1'. Enquanto em *idle*, os ponteiros relacionados a posição das sequências e chaves esperadas são mantidos em seus valores iniciais, e os LEDs permanecem todos acesos.

```
Estado print
116
           -- Estado que imprime as posições das sequências pré-definidas
117
           when print => leds<=sequences(r)(n);
118
                          nibble<=sequences(r)(n);
119
                          dly delay<=d;
120
                          try<="00000";
121
122
                          if ( seq delay>0 ) then
123
                              seq delay<=seq delay-1;
124
125
                             pr state<=dly;
126
                          end if;
```

O estado *print* é responsável pela exibição da sequência do jogo nos LEDs do kit, através dos ponteiros "r" e "n", que definem qual sequência e qual posição dela deverá ser mostrada, respectivamente. Através do *signal* "nibble", este valor é repassado para o componente do VGA, que pode então decidir qual cor deverá ser exibida na tela, durante o tempo definido pela posição das chaves de dificuldade.

```
Estado dly
119
           -- Estado de delay entre a impressão de cada cor
           when dly => leds<="0000";
120
                        nibble<="0000";
121
122
                        if ( dly_delay>0 ) then
123
                           dly_delay<=dly_delay-1;
124
125
                        else
                           if ( n=lvl ) then
126
127
                              pr state<=waiting;
128
129
                              pr state<=print;
130
                              n \le n+1;
131
                           end if:
132
                        end if:
```

O estado *dly* serve para inserir um pequeno *delay* entre a exibição de cada cor, permitindo que o jogador perceba caso ocorra de uma mesma cor se repetir. Além disso, este estado verifica se já foram impressas todas as cores devidas de acordo com o nível atual ou se o jogo deve voltar a imprimir mais cores.

Estado waiting 153 -- Estado que aguarda que o jogador pressione um botão when waiting => leds<="0000"; 154 nibble<="0000"; 155 swt delay<=p; 156 157 if (key_delay>0) then 158 159 key_delay<=key_delay-1; 160 161 pr_state<=ff; end if; 162 163 if (button/="0000") then 164 165 try<=button; 166 pr_state<=upack; end if;

Após mostrar todas as cores necessárias, o jogo entra no estado *waiting* e aguarda que o jogador pressione algum botão, buscando repetir a sequência mostrada. Caso nenhum *push-button* seja pressionado dentro de 5s, o jogo vai para o estado de derrota, denominado *ff.* Se algum botão for pressionado, o estado dos *push-buttons* é armazenado no *signal try* e a máquina avança para o próximo estado.

```
Estado upack
169
           -- Estado de debouncing para o reconhecimento da subida do botão
           when upack => leds<=button:
170
                         nibble<=button:
171
172
173
                          if (swt delay>0) then
174
                             swt delay<=swt delay-1;
175
                              pr state<=dbc;
176
                          end if;
177
```

No estado *upack*, o jogo aguarda que uma contagem de *debouncing* seja concluída, evitando que possíveis "repiques" sejam identificados como várias pressionadas do botão. Terminado esta contagem de 10ms, o jogo avança para o estado seguinte.

```
Estado dbc
179
           -- Estado que aguarda que o jogador solte o botão pressionado
           when dbc => leds<=button;
180
                        nibble<=button;
181
                        swt delay<=p;
182
183
                        if ( key delay>0 ) then
184
                           key_delay<=key_delay-1;
185
                        else
186
187
                           pr_state<=ff;
188
                        end if:
189
                        if (button="0000") then
190
                           pr_state<=downack;
191
192
                        end if:
```

Em *dbc*, espera-se que o jogador solte o botão pressionado anteriormente, para que então o jogo avance. Caso o *push-button* pressionado não seja solto dentro do tempo de 5s (contado a partir do momento em que o jogo entra em *waiting*), o jogo irá para o estado de derrota. Se o botão seja liberado dentro do tempo, a máquina de estados avança.

Estado downack

```
194
            -- Estado de debouncing para o reconhecimento da descida do botão
           -- Este estado também verifica se o botão correto foi pressionado
195
           when downack => leds<="0000";
196
                            nibble<="0000";
197
198
                            if (swt delay>0) then
199
200
                              swt_delay<=swt_delay-1;
                            else
201
202
                              key_delay<=T;
203
                              if (try=sequences(r)(s)) then
                                 if ( s=lvl ) then
204
205
                                    pr state<=nextlvl;
206
                                    s<=s+1;
207
                                    pr_state<=waiting;
208
209
                                 end if;
                              else
210
211
                                 pr_state<=ff;
212
                              end if;
                            end if:
213
```

O estado *downack*, semelhante ao estado *upack*, realiza uma contagem de debouncing de 10ms após detectado que o botão voltou ao normal. Feito isso, é verificado se a tentativa do jogador, armazenada em *try*, está de acordo com o que fora impresso. Um ponteiro diferente, denominado "s", é utilizado para controlar quantos botões espera-se que o jogador pressione a fim de repetir a sequência apresentada. Se a sequência estiver correta e for a última cor esperada, o jogo segue para *nextlvl*. Caso o botão pressionado pelo jogador esteja correto mas ainda não for o último esperado, o jogo volta para o estado *waiting*. Por fim, se o botão pressionado não for aquele da cor correspondente, o jogo vai para o estado de derrota.

```
Estado nextlvl
            -- Estado que incrementa o nível atual do jogo
215
            when nextlvl => leds<="0000";
216
                             nibble<="0000";
217
218
                             if (lvl=14) then
219
                               pr state<=gg;
220
                             else
221
                               pr state<=print;
222
                               lv1<=1v1+1;
223
224
                               n < = 0:
                               s<=1;
225
226
                             end if;
```

O estado *nextlvl* verifica se o nível atual é o último e, em caso afirmativo, atualiza o estado da máquina para o estado *gg* de vitória. Caso o jogador tenha repetido a sequência com sucesso mas ainda existam níveis a serem jogados, o jogo volta para o estado *print* e o nível do jogo é incrementado, assim como os ponteiros voltam ao valor inicial.

```
Estado gg
219
            -- Estado final de vitória
            when gg => leds<="1111";
220
                        nibble<="1111":
221
222
                          if ( start='1' ) then
223
                             pr_state<=print;
224
                             key_delay<=T;
225
                             lv1<=1;
226
227
                             n <= 0;
                             s<=1;
228
229
                             r<=r+1;
                            if (r=9) then
230
                                r<=0;
231
                            end if;
232
                          end if;
233
```

Quando em *gg*, o jogo encontra-se no estado de vitória e aguarda que o jogador mude o estado da chave *start* para '1' de modo a recomeçar o jogo. Se isso acontecer, o nível do jogo volta para 1 e a sequência a ser jogada a seguir muda, pois o ponteiro "r" é incrementado.

```
Estado ff
              -- Estado final de derrota
when ff => leds<=try;</pre>
235
236
237
                            nibble<=try;
238
                               if ( start='1' ) then
239
                                 pr_state<=print;
240
                                 key_delay<=T;
lvl<=1;</pre>
241
242
                                 n<=0;
243
                                 s<=1;
244
                                 r <= r+1;
246
                                 if (r=9) then
                                     r<=0;
247
                                  end if;
248
                               end if;
250
              end case;
251
```

O estado ff é o estado de derrota do jogo, quando o jogador erra uma cor da sequência ou demora muito tempo para repeti-la. Da mesma forma como em gg, esperase que *start* vá para '1' para reinicializar o jogo e mudar a sequência a ser exibida nesta próxima rodada.

2.1.4. Decoder.vhd

```
Parte 1
1 library IEEE;
2 use IEEE.STD LOGIC 1164.ALL;
3 use work.global pack.all;
5 entity Decoder is
 6
      generic (f: integer := 4000; -- Tempo de multiplexação
                1: integer := 25000000); -- Clock de rolagem do display
 8
 9
      port ( clk, reset: in std_logic; -- Clock, Reset
             display: out std_logic_vector (6 downto 0); -- Segmentos dos displays (catodos)
10
             anode: out std_logic_vector (3 downto 0); -- Anodos dos displays
11
             pr state: in states; -- Estado atual da máquina
12
             lvl: in integer range 1 to 14); -- Nível atual do jogo
13
14
15 end Decoder;
```

No início do código do decoder para os displays de sete segmentos, tem-se a declaração das entradas e saídas do component. Cabe destacar os tempos de rolagem do display (0,5s) e o tempo para multiplexação dos anodos (80us).

(continuação...)

Os catodos e os anodos também são declarados, juntamente com as entradas oriundas do código principal, "pr_state" e "lvl" que repassarão o nível e estado atuais do jogo.

```
Parte 2
31 begin
32
         - Processo que realização a multiplexação e impressão dos valores nos displays
33
       process(clk, lvl, reset, pr_state)
34
       begin
35
36
37
           if ( reset='1' ) then
38
              j<=19;
39
              count_1<=1;
40
41
              count_f<=f;
42
           elsif ( rising_edge(clk) ) then
43
44
             count_f<=count_f-1;
45
              count_1<=count_1-1;
              if (count_f=0) then
  count_f<=f;
  i<=i-1;</pre>
46
47
48
49
                 if (i=0) then
                    i<=3;
50
                 end if;
51
52
              end if;
              if (count_l=0) then
53
                 count_1<=1;
54
                 j<=j-1;
56
                 if (j=0) then
57
                    j<=19;
58
                 end if;
59
              end if;
60
           end if;
```

Este *process* contém a lógica para multiplexação e rolamento da mensagem inicial nos displays de sete segmentos. Utilizando dois contadores, a multiplexação é realizada percorrendo as posições do vetor de enables que habilita cada um dos anodos por vez. Já a rolagem é feita somando-se um o signal "j" a variável "i", o que promove um deslocamento dentro do vetor da mensagem das quatro posições que são exibidas nos displays.

Parte 3

```
- Case que verifica o nível atual e repassa o valor adequado de pontuação para exibição
          case lvl is
63
64
                when 1 => pont <= ("11111111", "0000001", "0000001", "11111111");
65
66
                when 2 => pont <= ("11111111", "0000001", "1001111", "11111111");
67
68
                when 3 => pont <= ("11111111", "0000001", "0010010", "11111111");
69
70
                when 4 => pont <= ("11111111", "0000001", "0000110", "11111111");
71
72
73
                when 5 => pont <= ("11111111", "0000001", "1001100", "11111111");
74
                when 6 => pont <= ("11111111", "0000001", "0100100", "11111111");
76
                77
78
                when 8 => pont <= ("1111111", "0000001", "0001111", "11111111");
79
80
                when 9 => pont <= ("1111111", "0000001", "0000000", "11111111");
81
82
                when 10 => pont <= ("11111111", "0000001", "0000100", "11111111");
83
84
                when 11 => pont <= ("11111111", "1001111", "0000001", "1111111");
85
86
                when 12 => pont <= ("11111111", "1001111", "1001111", "11111111");
87
88
                when 13 => pont <= ("11111111", "1001111", "0010010", "11111111");
90
```

No case acima, tem-se a definição do que o signal "pont" irá receber com base no nível atual do jogo correspondente a pontuação, a fim de decodificar os números decimais para exibição nos displays. Este sinal será transmitido aos catodos caso se esteja num estado onde a pontuação deve ser exibida.

Parte 4

```
- Case que vericia o estado atual da máquina e o que deve ser exibido nos displays
95
 96
           case pr state is
 97
                 when idle => display <= projeto_aplic(i+j);
 98
 99
                 when ff => display <= gameover(i);
100
101
                 when gg => display <= win(i);
102
103
104
                 when others => display <= pont(i);
105
           end case:
106
107
           -- Atribuição aos anodos do modo atual para multiplexação
108
           anode <= anodos(i);
109
110
111
        end process;
112
113
114 end Behavioral;
```

Neste *case*, sensível ao estado presente da máquina, verifica-se o que os displays devem exibir no momento (mensagem inicial, pontuação, mensagem "GG" ou mensagem "FF").

2.1.5. VGA.vhd

Parte 1

```
1 library ieee;
                    e.std logic 1164.all;
       use work.global_pack.all;
       use work.rom_characters.all;
      entity VGA is
           generic(
                                   integer := 120; -- Largura do pulso de sincronização horizonta
integer := 64; -- Largura do back porch horizontal em pixels
integer := 800; -- Largura de display horizontal em pixels
integer := 56; -- Largura do front porch horizontal em pixels
               h_pulse
                                                    := 120; -- Largura do pulso de sincronização horizontal em pixels
 10
               h bp
               h_pixels :
 11
 12
               h_fp
                                   std logic := '1'; -- Polaridade da sincronização horizontal (1 = positiva, 0 = negativa)
 13
               h pol
                                   integer := 6; -- Largura do pulso de sincronização vertical em linhas
integer := 23; -- Largura do back porch vertical em linhas
               v_pulse :
 14
 15
                                                    := 600; -- Largura de display vertical em linhas
 16
17
               v_pixels :
                                   integer
                            : integer := 37; -- Largura do front porch vertical em linhas

: std_logic := '1'); --Polaridade da sincronização vertical (1 = positiva, 0 = negativa)
               v fp
 18
 19
20
                              : in std_logic; -- Pixel clock
: in std_logic; -- Reset
 21
 22
23
               reset
                               : in std_logic; -- Reset
: in std_logic_vector (3 downto 0); -- Estado dos leds
: in integer range 1 to 14; -- Nível atual do jogo
: in states; -- Estado da máquina
: out std_logic; -- Pulso de sincronização horizontal
: out std_logic; -- Pulso de sincronização vertical
: out std_logic_vector (2 downto 0); -- Bits do vermelho do VGA
: out std_logic_vector (2 downto 0); -- Bits do verde do VGA
               nibble
 24
               pr_state :
 25
 26
               h_sync
 27
               v sync
               red
               green
 29
                                     out std_logic_vector (1 downto 0)); -- Bits do azul do VGA
 31
 32 end VGA:
```

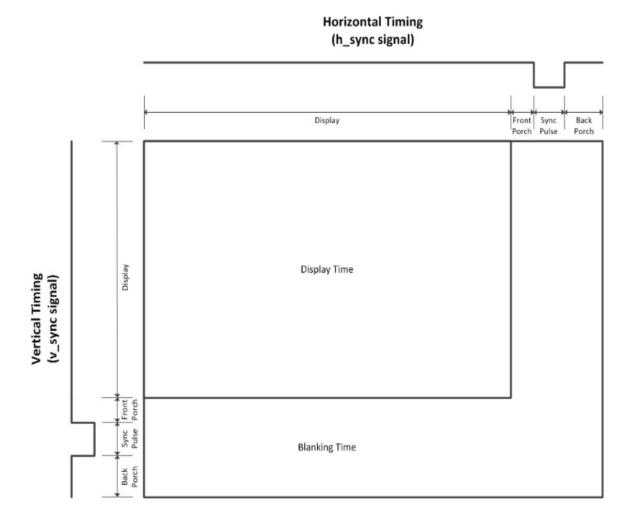
Tomando como base o controlador VGA em VHDL disponibilizado no site eewiki^[3], e também o datasheet do kit de desenvolvimento nexys 2^[6], desenvolveu-se um código para exibir as sequências de cores e pontuações do jogo através do VGA.

Foi decidido utilizar a resolução 800x600 (72Hz) para implementação do VGA já que tal resolução necessita de um *pixel clock* de 50Mhz, que coincide com o clock disponível no kit de desenvolvimento.

O VGA requer, fora os bits do RGB para definir cor do *pixel*, dois sinais de sincronização que permitem a correta atualização do monitor e varredura das linhas. O VGA funciona percorrendo uma linha completa e então avançando verticalmente para baixo, sendo assim, o sinal de sincronização horizontal ocorre com muito mais frequência que o vertical.

Fora o pulso de sincronização, existem dois intervalos de tempo, denominados de *back porch* e *front porch*, antes e depois do pulso de sincronização, em que nenhum *pixel* é atualizado. Todos esses tempos variam de resolução a resolução, bem como a polaridade do pulso. Sendo assim, os valores desses intervalos de tempo para a resolução escolhida, em função dos *pixels* e linhas, foram devidamente declarados como *generic*, tanto para a horizontal, como vertical.

Além disso, as entradas de clock e reset também foram declaradas, bem como as saídas (RGB e *sync*) e as entradas referente ao estado atual e ao nível do jogo, para permitir a implementação da lógica do que deve ser impresso na tela.



Este esquema^[3] exemplifica melhor o funcionamento do sistema VGA descrito anteriormente, mostrando os tempos de display, *front porch*, pulso de *sync* e *back porch*. Quando estamos no "*Display Time*", estamos no período de tempo em que os *pixels* recebem os dados RGB para formação da imagem. Em "*Blanking Time*", nada é enviado para tela pois nos encontramos no tempo de atualização, onde é preciso manter os bits RGB em '0'.

```
Parte 2
34 architecture behavior of VGA is
35
       constant h_period : integer := h_pulse + h_bp + h_pixels + h_fp;
constant v_period : integer := v_pulse + v_bp + v_pixels + v_fp;
37
      shared variable h_count : integer range 0 to h_period - 1 := 0;
shared variable v_count : integer range 0 to v_period - 1 := 0;
38
39
      shared variable red_var : std_logic_vector (2 downto 0) := ( others => '0');
40
      shared variable green_var : std_logic_vector (2 downto 0) := ( others => '0');
41
      shared variable blue var : std logic vector (1 downto 0) := ( others => '0');
42
43
44 begin
45
       process(clk, reset, pr_state)
46
47
       begin
48
49
         -- Reinicia o VGA em caso de reset
         if(reset = '1') then
50
           h count := 0;
51
           v_count := 0;
52
           h sync <= not h pol;
53
           v sync <= not v pol;
54
55
         elsif(rising_edge(clk)) then
56
57
58
            -- Contadores
           if(h_count < h_period - 1) then</pre>
59
             h count := h count + 1;
60
61
              h count := 0;
62
             if(v\_count < v\_period - 1) then
63
               v_count := v_count + 1;
64
65
              else
               v_count := 0;
66
67
              end if:
           end if;
68
69
70
            -- Sincronização Horizontal
71
           if(h_count < h_pixels + h_fp or h_count > h_pixels + h_fp + h_pulse) then
             h sync <= not h pol;
72
73
           else
74
             h sync <= h pol;
75
           end if;
76
            -- Sincronização Vertical
77
78
           if(v_count < v_pixels + v_fp or v_count > v_pixels + v_fp + v_pulse) then
              v sync <= not v_pol;</pre>
79
80
            else
              v_sync <= v_pol;
81
82
            end if:
```

O process do VGA verifica a presença do reset e, em caso afirmativo, mantém a tela apagada, sem envio de nenhum sinal. Caso contrário, a contagem horizontal é atualizada a cada subida do clock e a contagem vertical a cada vez que se chega no fim da linha. Utilizando esses contadores, é possível determinar em que posição da área de display o sistema encontra-se e definir qual deve ser o valor do pulsos de sincronização e dos bits RGB.

Parte 3

```
Verifica se está dentro da área de impressao do display e implementa a lógica para cada estado
 84
             if(h_count < h_pixels and v_count < v_pixels) then</pre>
 87
                if (v_count>300) then
                    if (nibble="1000") then
 88
                       if (h_count<200) then
                           red <= "111";
green <= "000";
 91
                           blue <= "00";
 92
 94
                           red <= (others => '0');
                           green <= (others => '0');
blue <= (others => '0');
 95
 96
                        end if;
                    elsif (nibble="0100") then
 98
                       if (h_count>200 and h_count<400) then
 99
100
                           green <= "000";
blue <= "11";
101
102
                        else
103
104
                          red <= (others => '0');
                           green <= (others => '0');
blue <= (others => '0');
105
106
107
```

Este trecho exemplifica como foi implementada a lógica para saber o que deve ser impresso na tela de acordo com a posição que estamos e a cor atual da sequência. Para isso, o *signal* "nibble" é utilizado, repassando o estado dos LEDs e por consequência qual cor deve ser impressa. Cada uma das cores possui uma região retangular 200x300 na parte inferior da tela.

2.2. ANÁLISE DOS RESULTADOS

2.2.1. Testbench.vhd

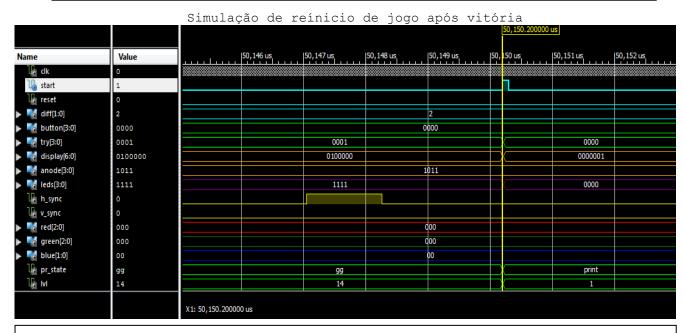
```
reset<='1';
           button<="0000";
115
           diff<="00";
116
           start<='0';
117
           wait for 100 ns;
118
           reset<='0';
119
           diff<="10";
120
           start<='1';
121
122
           wait for 100 ns;
           start<='0';
123
124
           wait for 600 us;
          button<="1000";
125
           wait for 50 us;
126
          button<="0000";
127
           wait for 900 us;
128
           button<="1000";
129
           wait for 50 us;
130
          button<="0000";
131
           wait for 100 us;
132
           button<="0100":
133
           wait for 50 us;
134
135
           button<="0000";
           wait for 1200 us;
136
           button<="1000";
137
138
           wait for 50 us;
           button<="0000";
139
           wait for 100 us;
140
           button<="0100";
141
```

O testbench do projeto completo foi implementado de modo a simular o pressionamento dos botões e das chaves de start e dificiculdade, verificando o comportamento do sistema de acordo com as mudanças de entrada.

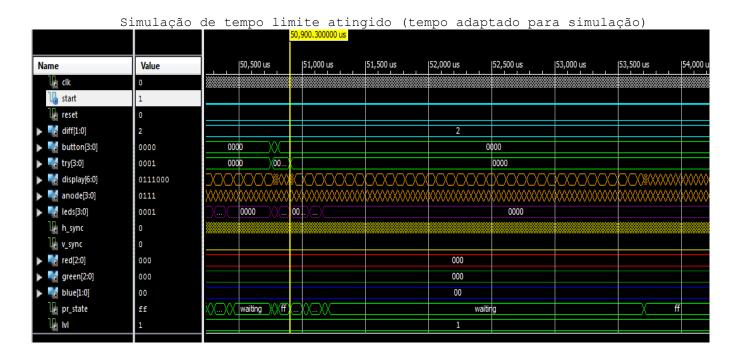
2.2.2. Simulações



Esta primeira simulação verifica se o jogo evolui de maneira adequada ao receber corretamente as repetições das sequências de cores. Podemos verificar no último sinal (*IvI*) que o jogo avançou até o nível de dificuldade 14 e após se repetir a última sequência corretamente o jogo avançou e permaneceu no estado de vitória "gg". É possível notar também que o tempo para avançar de nível aumenta progressivamente, já que mais tempo é despendido imprimindo a sequência de cores e mais tempo faz-se necessário para que o jogador repita a sequência utilizando os *push-buttons*. Um detalhe é a velocidade da ocorrência da sincronização horizontal do VGA em relação a vertical, conforme esperado.



Nesta simulação verificou-se o reinício adequado do jogo após a vitória, onde a chave *start* tem seu estado alterado e então o jogo vai para o estado *print* conforme esperado. Pode-se ver também que o display zera a pontuação e notamos o pulso se sincronização horizontal ocorrendo. Além disso, vemos que os LEDs vão para "0000", pois no ínicio de todas as sequências existe uma etapa a mais em que os LEDs permancem apagados, gerando um tempo para o jogador preparar-se para jogar equivalente ao de impressão de uma cor para o nível selecionado.



Nesta simulação, verificou-se que após imprimir uma cor e aguardar o jogador por um tempo maior que o limite, o jogo automaticamente vai para o estado de derrota, pois nenhum botão foi pressionado dentro do intervalo especificado.



Nesta última etapa de testes, vericou-se que após um erro na repetição da sequência o jogo vai para o estado de derrota "ff". No começo da simulação a esquerda, a sequência "1000" é mostrada pelos LEDs e após algum tempo o jogador pressiona o botão errado, enviando a tentativa como "0001", que também é impressa, o que permite ao jogador visualizar que cor inseriu. Após o *debouncing* do botão, verificou-se o erro e mudou-se para o estado "ff".

2.2.3 Análise de utilização de recursos

Genius Project Status (11/29/2016 - 00:44:08)						
Project File:	Genius.xise	Parser Errors:	No Errors			
Module Name:	Genius	Implementation State:	Programming File Generated			
Target Device:	xc3s500e-5fg320	• Errors:	No Errors			
Product Version:	ISE 14.7	• Warnings:	No Warnings			
Design Goal:	Timing Performance	• Routing Results:	All Signals Completely Routed			
Design Strategy:	Performance without IOB Packing	Timing Constraints:	All Constraints Met			
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)			

Device Utilization Summary							
Logic Utilization	Used	Available	Utilization	Note(s)			
Number of Slice Flip Flops	1,355	9,312	14%				
Number of 4 input LUTs	5,254	9,312	56%				
Number of occupied Slices	2,743	4,656	58%				
Number of Slices containing only related logic	2,743	2,743	100%				
Number of Slices containing unrelated logic	0	2,743	0%				
Total Number of 4 input LUTs	5,280	9,312	56%				
Number used as logic	5,254						
Number used as a route-thru	26						
Number of bonded <u>IOBs</u>	34	232	14%				
Number of BUFGMUXs	1	24	4%				
Average Fanout of Non-Clock Nets	3.67						

Performance Summary					
Final Timing Score:	0 (Setup: 0, Hold: 0)	Pinout Data:	Pinout Report		
Routing Results:	All Signals Completely Routed	Clock Data:	Clock Report		
Timing Constraints:	All Constraints Met				

Após compilado e implementado o *design* do projeto, o relatório de consumo acima foi gerado permitindo a análise da alocação de recursos. Em comparação com os outros projetos já realizados, o projeto aplicativo do jogo Genius teve um consumo muito maior. Tal consumo de recursos deu-se, sobretudo, graças a exigência para implementação da impressão de caracteres 200x200 *pixels* no VGA, e também pelo tamanho e quantidade dos contadores (e consequentemente comparadores) para gerar as bases de tempo necessárias ao jogo. Ao se retirar o trecho de pontuação do VGA e diminuir as constantes de tempo, o consumo de recursos da VGA caiu para em torno de 3 vezes do que o consumo aqui apresentado.

CONCLUSÕES

Com este trabalho pudemos comprovar o funcionamento prático do código de descrição de hardware transferindo o arquivo.bit para a FPGA Nexys 2 Spartan 3.

Pudemos praticar e aprimorar os conhecimentos de linguagem VHDL, de utilização dos softwares Xilinx ISE e Digilent Adept 2 e também verificar a robustez do software ao gerar automaticamente arquivos de simulação e permitir que pudéssemos analisar os resultados gerados através do gráfico da simulação, antes mesmo de sua implementação prática.

O conceito de máquina de estados, demonstrou-se importantíssimo para o desenvolvimento de soluções onde temos diferentes estados possíveis para o sistema, e variáveis de entrada que coordenam a alteração dos mesmos. Através do diagrama de estados, foi possível obter uma noção geral do projeto e definir todas as possíveis transações, permitindo então realizar a implementação do projeto em VHDL.

As funções de bibliotecas do sistema, package, component e procedure também foram extremamente úteis do ponto de vista de permitir distribuir-se melhor os componentes do jogo e criar trechos de lógica e tipos reutilizáveis em todos os outros códigos do projeto.

Não menos importante, o kit de desenvolvimento mostrou toda a capacidade que possui ao suportar a implementação de tal projeto, que demandou uma quantidade razoável de recursos da FPGA.

Por fim, ao término deste projeto aplicativo, comprova-se o quão rica de possibilidades e útil é a linguagem VHDL, permitindo que engenheiros e demais entusiastas possam descrever *hardwares* para operar nas mais diversas funcões e finalidades, sendo este um dos pilares da área de sistemas embarcados.

REFERÊNCIAS BIBLIOGRÁFICAS

- 1 UFPR. Orientação para normalização de trabalhos acadêmicos: exemplos de referências. Curitiba, 2013. Disponível em: http://www.portal.ufpr.br/tutoriais_normaliza/referencia_exemplo.pdf. Acesso em 14 de junho 2016.
- 2 PEDRONI, Volnei A.: Eletronica digital moderna e VHDL 1ª Edição, 2010, Elsevier. (PEDRONI, 2010).
- 3 LARSON, Scott. VGA Controller (VHDL). Disponível em: https://eewiki.net/pages/viewpage.action?pageId=15925278. Acesso em 15 de novembro de 2016.
- 4 GIMP Genuine Image Manipulation Program. Disponível em: https://www.gimp.org/. Acesso em 17 de novembro de 2016.
- 5 DCODE Image in Binary 01. Disponível em: http://www.dcode.fr/binaryimage. Acesso em 17 de novembro de 2016.
- 6 DIGILENT Digilent Nexys2 Board Reference Manual. Disponível em: https://www.ece.umd.edu/class/enee245.F2016/nexys2_reference_manual.pdf.

 Acesso em 10 de novembro de 2016.

ANEXOS

Genius.vhd

```
-- Código Principal do Projeto Aplicativo de Microeletrônica - Jogo Genius
-- Prof: Dr. Sibilla Batista da Luz França
-- Alunos: Erik Nayan e Diego Molina
-- Versão final: 29/11/2016
library ieee;
use ieee.std logic 1164.all;
use work.global_pack.all;
entity Genius is
          generic (d: integer := 5000000; -- Tempo de 100 ms
                                            p: integer := 500000; -- Tempo de 10 ms
                                             e: integer := 50000000; -- Tempo de 1 s
                                            m: integer := 25000000; -- Tempo de 500 ms
                                            h: integer := 10000000; -- Tempo de 200 ms
                                            T: integer := 250000000); -- Tempo de 5 s
          port (clk, start, reset: in std_logic; -- Clock, Start e Reset
                                 diff: in std logic vector (1 downto 0); -- Chaves seletoras de
dificuldade
                                 button: in std logic vector (3 downto 0); -- Push-buttons do
jogo
                                 display: out std logic vector (6 downto 0); -- Segmentos dos
displays
                                 anode: out std_logic_vector (3 downto 0); -- Anodos dos displays
leds: out std_logic_vector (3 downto 0); -- Leds de visualização
                                 h_sync: out std_logic; -- Sinal de sincronização horizontal v sync: out std_logic; -- Sinal de sincronização vertical
                                 red: out std logic vector (2 downto 0); -- Bits vermelho do VGA
                                 green: out std_logic_vector (2 downto 0); -- Bits verde do VGA
                                 blue: out std_logic_vector (1 downto 0) ); -- Bits azul do VGA
end Genius;
architecture Behavioral of Genius is
          signal key delay: integer range 0 to T := T; -- Delay máximo entre botões
          signal dly_delay: integer range 0 to d := d; -- Delay entre impressão de cores signal swt_delay: integer range 0 to p := p; -- Tempo de debouncing para os push-
buttons
          signal seq delay: integer range 0 to e := e; -- Tempo de impressão de cada cor
          signal r: integer range 0 to 9:= 0; -- Ponteiro da matriz de sequências
          signal s: integer range 1 to 14 := 1; -- Ponteiro de espera de botões signal n: integer range 0 to 14 := 0; -- Ponteiro de impressao de sequência
          signal lvl: integer range 1 to 14 := 1; -- Nível atual do jogo signal nibble: std_logic_vector (3 downto 0) := "0000"; -- Repassa o estado dos leds
          signal try: std_logic_vector (3 downto 0) := "0000"; -- Armazena a tentativa do
iogador
           signal pr state: states := idle; -- Estado da máquina
           -- Declaração do componente Decoder que controla os displays de 7 segmentos
          component Decoder is
                     port ( clk, reset: in std logic;
                                           display: out std logic vector (6 downto 0);
                                           anode: out std_logic_vector (3 downto 0);
                                           pr_state: in states;
                                           lvl: in integer range 1 to 14);
          end component;
           -- Declaração do componente VGA que controla a exibição no monitor
          component VGA is
                                clk: in std logic;
                     port (
                                           reset: in std logic;
                                           nibble: in std_logic_vector (3 downto 0);
                                           lvl: in integer range 1 to 14;
                                           pr state: in states;
                                           h sync: out std logic;
                                           v sync: out std logic;
                                           red: out std_logic_vector (2 downto 0);
                                           green: out std logic vector (2 downto 0);
                                           blue: out std_logic_vector (1 downto 0));
```

```
end component;
begin
          -- Processo que implementa a lógica de estados do jogo
          process(clk, pr state, reset, start, seq delay, diff)
          begin
          if (reset = '1') then
                     pr_state<=idle; -- Estado inicial em espera do jogo</pre>
          elsif (rising edge(clk)) then
                     -- Verificação da posição das chaves de dificuldade e atribuição ao sinal
                     if (diff="00") then
                               seq_delay<=e;
                     elsif (diff="01") then
                    seq_delay<=m;
elsif ( diff="10" ) then</pre>
                               seq_delay<=h;
                     else
                               seq delay<=e;
                     end if;
                     -- Case dos estados da máquina
                     case pr state is
                     -- Estado inicial de espera da máquina
                    when idle => leds<="1111";
                                                                nibble<="1111";
                                                                lv1<=1;
                                                                n <= 0;
                                                                s<=1;
                                                                r <= 0;
                                                                if ( start='1' ) then
                                                                         pr_state<=print;</pre>
                                                                end if;
                    -- Estado que imprime as posições das sequências pré-definidas
              when print => leds<=sequences(r)(n);
                                                                 nibble <= sequences(r)(n);
                                                                 dly delay<=d;</pre>
                                                                 try<="0000";
                                                                 if ( seq_delay>0 ) then
          seq delay<=seq delay-1;
                                                                 else
                                                                                    pr_state<=dly;</pre>
                                                                 end if;
                     -- Estado de delay entre a impressão de cada cor when dly => leds<="0000";
                                                               nibble<="0000";
                                                               if (dly delay>0) then
                                                                         dly_delay<=dly_delay-1;</pre>
                                                               else
                                                                         if (n=lvl) then
                                                                                    pr state<=waiting;</pre>
                                                                         else
                                                                                    pr_state<=print;</pre>
                                                                                    n \le n+1;
                                                                         end if;
                     -- Estado que aguarda que o jogador pressione um botão
                     when waiting => leds<="0000";
                                                                   nibble<="0000";
                                                                           swt delay<=p;</pre>
                                                                           if ( key_delay>0 ) then
```

key delay<=key delay-1;

```
pr_state<=ff;</pre>
                                                                         end if;
                                                                         if (button/="0000") then
                                                                                  try<=button;
                                                                                  pr_state<=upack;
                                                                         end if;
                    -- Estado de debouncing para o reconhecimento da subida do botão
                    when upack => leds<=button;
                                                                nibble <= button;
                                                                if (swt delay>0) then
          swt delay<=swt delay-1;
                                                                else
                                                                                  pr_state<=dbc;
                                                                end if;
                    -- Estado que aguarda que o jogador solte o botão pressionado
                    when dbc => leds<=button;
                                                              nibble<=button;
                                                              swt_delay<=p;</pre>
                                                              if ( key_delay>0 ) then
                                                                        key delay<=key delay-1;
                                                              else
                                                                        pr_state<=ff;</pre>
                                                              end if;
                                                              if (button="0000") then
                                                                        pr_state<=downack;</pre>
                                                              end if;
                    -- Estado de debouncing para o reconhecimento da descida do botão
                    -- Este estado também verifica se o botão correto foi pressionado
                    when downack => leds<="0000";
                                                                         nibble<="0000";
                                                                         if ( swt_delay>0 ) then
          swt_delay<=swt_delay-1;</pre>
                                                                         else
                                                                                  key_delay<=T;
if (</pre>
try=sequences(r)(s)) then
                                                                                             if (
s=lvl ) then
                                                                                                       pr state<=r
                                                                                             else
                                                                                                       s<=s+1;
                                                                                                       pr_state<=w
                                                                                             end if;
                                                                                  else
          pr state<=ff;</pre>
                                                                                  end if;
                                                                         end if;
                    -- Estado que incrementa o nível atual do jogo
                    when nextlvl => leds<="0000";
                                                                         nibble<="0000";
                                                                         if ( lvl=14 ) then
                                                                                  pr_state<=gg;</pre>
                                                                                  pr_state<=print;</pre>
                                                                                  lvl<=lvl+1;
                                                                                  n \le 0;
                                                                                  s<=1;
                                                                         end if;
                    -- Estado final de vitória
                    when gg \Rightarrow leds<="1111";
                                                     nibble<="1111";
```

else

```
if ( start='1' ) then
                                                                                      pr state<=print;</pre>
                                                                                      key delay<=T;</pre>
                                                                                      lv1<=1;
                                                                                      n \le 0;
                                                                                     s<=1;
                                                                                      r <= r + 1:
                                                                                      if (r=9) then
                                                                                                  r < = 0;
                                                                                      end if;
                                                                         end if:
                        -- Estado final de derrota
                        when ff => leds<=try;
                                                                nibble<=try;
                                                                           if (start='1') then
                                                                                     pr state<=print;</pre>
                                                                                     key_delay<=T;
lvl<=1;</pre>
                                                                                      n <= 0;
                                                                                      s<=1;
                                                                                      r <= r+1;
                                                                                     if (r=9) then
                                                                                                  r<=0:
                                                                                      end if;
                                                                           end if;
                        end case:
            end if;
            end process;
            -- Associação dos componentes com os respectivos E/S e sinais internos
           disp: Decoder port map(clk, reset, display, anode, pr_state, lvl);
monitor: VGA port map (clk, reset, nibble, lvl, pr_state, h_sync, v_sync, red,
green, blue);
end Behavioral;
Decoder.vhd
library IEEE;
use IEEE.STD LOGIC 1164.ALL;
use work.global pack.all;
entity Decoder is
            generic (f: integer := 4000; -- Tempo de multiplexação
           l: integer := 25000000); -- Clock de rolagem do display port (clk, reset: in std_logic; -- Clock, Reset
                     display: out std_logic_vector (6 downto 0); -- Segmentos dos displays
(catodos)
                                      anode: out std logic vector (3 downto 0); -- Anodos dos displays
                                     pr_state: in states; -- Estado atual da máquina
lvl: in integer range 1 to 14); -- Nível atual do jogo
end Decoder:
architecture Behavioral of Decoder is
            signal i: integer range 0 to 3 := 3; -- Ponteiro de multiplexação dos displays
            signal j: integer range 0 to 19 := 19; -- Ponteiro para rolagem da mensagem
            signal count_f: integer range 0 to f := f; -- Contador de rolagem
            signal count_1: integer range 0 to 1 := 1; -- Contador de Horagem
signal count_1: integer range 0 to 1 := 1; -- Contador de multiplexação
signal pont: score := ("11111111","11111111","1111111"); -- Sinal para
exibição da pontuação
            constant anodos: enable := ("0111","1011","1101","1110"); -- Modos dos anodos constant gameover: score := ("0111000","0001000","1001111","1110001"); -- Mensagem
de "GG"
            constant win: score := ("1111111","0100000","0100000","1111111"); -- Mensagem de
"FF"
constant projeto_aplic: pa_ed :=
("0011000","1111010","0000001","1000011","0110000","1110000","0000001",
```

```
"1111111","0001000","0011000","1110001","1001111","0110001","0001000","1110000","100
1111", "1000001",
           "0000001","1111110","0110000","1000010","1111111","1111111"); -- Mensagem "Projeto
Aplicativo - ED"
begin
           -- Processo que realização a multiplexação e impressão dos valores nos displays
          process(clk, lvl, reset, pr state)
                     if ( reset='1' ) then
                                 i <= 3:
                                 j<=19;
                                count_1<=1;
                                count f<=f;
                      elsif ( rising_edge(clk) ) then
                                count_f<=count_f-1;
count_l<=count_l-1;</pre>
                                 if (count_f=0) then
                                            count f<=f;
                                           i <= i - \overline{1};
                                           if (i=0) then
                                                       i<=3:
                                           end if:
                                 end if;
                                 if (count 1=0) then
                                           ___count_1<=1;
                                            j <= j - \overline{1};
                                            if (j=0) then
                                                      j<=19;
                                           end if;
                                end if;
                      end if;
                      -- Case que verifica o nível atual e repassa o valor adequado de pontuação
para exibição
                     case lvl is
                                           when 1 => pont <=
("1111111", "0000001", "0000001", "1111111");
                                            when 2 \Rightarrow pont <=
("1111111", "0000001", "1001111", "1111111");
                                            when 3 \Rightarrow pont <=
("1111111", "0000001", "0010010", "11111111");
                                            when 4 \Rightarrow pont <=
("1111111", "0000001", "0000110", "11111111");
                                            when 5 \Rightarrow pont \Leftarrow
("1111111", "0000001", "1001100", "11111111");
                                           when 6 => pont <=
("1111111", "0000001", "0100100", "11111111");
                                           when 7 \Rightarrow pont <=
("1111111", "0000001", "0100000", "1111111");
                                            when 8 => pont <=
("1111111", "0000001", "0001111", "1111111");
                                            when 9 \Rightarrow pont <=
("1111111", "0000001", "0000000", "11111111");
                                           when 10 => pont <=
("1111111", "0000001", "0000100", "11111111");
                                            when 11 => pont <=
("1111111", "1001111", "0000001", "11111111");
                                           when 12 \Rightarrow pont <=
("1111111", "1001111", "1001111", "11111111");
```

```
when 13 => pont <=
("1111111", "1001111", "0010010", "11111111");
                                             when 14 => pont <=
("1111111", "1001111", "0000110", "1111111");
                      end case;
                      -- Case que vericia o estado atual da máquina e o que deve ser exibido nos
displays
                      case pr state is
                                             when idle => display <= projeto aplic(i+j);
                                     when ff => display <= gameover(i);
                                             when gg => display <= win(i);
                                             when others => display <= pont(i);
                      end case;
                      -- Atribuição aos anodos do modo atual para multiplexação
                      anode <= anodos(i);</pre>
           end process;
end Behavioral;
VGA.vhd
library ieee;
use ieee.std logic 1164.all;
use work.global_pack.all;
use work.rom characters.all;
entity VGA is
  generic(
    h_pulse : integer := 120; -- Largura do pulso de sincronização horizontal em pixels
    h_bp : integer := 64; -- Largura do back porch horizontal em pixels
h_pixels : integer := 800; -- Largura de display horizontal em pixels
             : integer := 56; -- Largura do front porch horizontal em pixels
: std_logic := '1'; -- Polaridade da sincronização horizontal (1 = positiva, 0 =
    h_fp
    h pol
negativa)
    v pulse : integer
                              := 6; -- Largura do pulso de sincronização vertical em linhas
               : integer := 23; -- Largura do back porch vertical em linhas
    v bp
    v_pixels : integer := 600; -- Largura de display vertical em linhas
             : integer := 37; -- Largura do front porch vertical em linhas
: std logic := '1'); --Polaridade da sincronização vertical (1 = positiva, 0 =
    v_fp
    v pol
negativa)
  port(
                                 std logic; -- Pixel clock
    clk
                         : in
               : in std logic; -- Reset
    reset
            nibble : in std_logic_vector (3 downto 0); -- Estado dos leds lvl : in integer range 1 to 14; -- Nível atual do iogo
            pr_state : in
                                states; -- Estado da máquina
    h_sync : out std_logic; -- Pulso de sincronização horizontal v_sync : out std_logic; -- Pulso de sincronização vertical
              ed : out std_logic_vector (2 downto 0); -- Bits do vermelho do VGA : out std_logic_vector (2 downto 0); -- Bits do verde do VGA : out std_logic_vector (1 downto 0)); -- Bits do azul do VGA
           red
    blue
end VGA:
architecture behavior of VGA is
  constant h_period : integer := h_pulse + h_bp + h_pixels + h_fp; -- Número total de
pixels em uma linha
  constant v period : integer := v pulse + v bp + v pixels + v fp; -- Número total de
linhas em uma coluna
  shared variable h_count : integer range 0 to h_period - 1 := 0; -- Contador horizontal
  shared variable v count : integer range 0 to v period - 1 := 0; -- Contador vertical
```

```
shared variable red var : std logic vector (2 downto 0) := ( others => '0'); -- Variável
para procedure de caracteres
  shared variable green_var : std_logic_vector (2 downto 0) := ( others => '0'); -- Variável
para procedure de caracteres
 shared variable blue_var : std_logic_vector (1 downto 0) := ( others => '0'); -- Variável
para procedure de caracteres
begin
  process(clk, reset, pr state)
  begin
           -- Reinicia o VGA em caso de reset
    if(reset = '1') then
     h count := 0;
      v_count := 0;
      h_sync <= not h_pol;
      v_sync <= not v_pol;</pre>
    elsif(rising\_edge(clk)) then
      -- Contadores
      if(h count < h period - 1) then
        h = h = h = h = 1;
      else
        h count := 0;
        \overline{if} (v count < v period - 1) then
          v count := v count + 1;
        else
          v_count := 0;
        end if;
      end if:
      -- Sincronização Horizontal
      if(h_count < h_pixels + h_fp or h_count > h_pixels + h_fp + h_pulse) then
        h sync <= not h pol;
      else
       h_sync <= h_pol;
      end if;
      -- Sincronização Vertical
      \label{eq:count}  \mbox{if} \mbox{($v$\_count < $v$\_pixels + $v$\_fp or $v$\_count > $v$\_pixels + $v$\_fp + $v$\_pulse) then 
        v_sync <= not v_pol;</pre>
      else
        v sync <= v pol;
      end if;
      -- Verifica se está dentro da área de impressao do display e implementa a lógica para
      if (h count < h pixels and v count < v pixels) then
                               if (v_count>300) then
                                         if (nibble="1000") then
                                                   if (h count<200) then
                                                              red <= "111";
                                                              green <= "000";
                                                              blue <= "00";
                                                    else
                                                              red <= (others => '0');
                                                              green <= (others => '0');
blue <= (others => '0');
                                                    end if;
                                         elsif (nibble="0100") then
                                                    if (h_count>200 and h_count<400) then
                                                              red <= "000";
                                                              green <= "000";
                                                              blue <= "11";
                                                    else
                                                              red <= (others => '0');
                                                              green <= (others => '0');
blue <= (others => '0');
                                                    end if;
                                         elsif (nibble="0010") then
                                                    if (h count>400 and h count<600) then
                                                              red <= "111";
                                                              green <= "111";
```

```
else
                                                                red <= (others => '0');
                                                                green <= (others => '0');
blue <= (others => '0');
                                                     end if;
                                           elsif (nibble="0001") then
                                                     if (h_count>600 and h_count<800) then
                                                                red <= "000";
                                                                green <= "111";
                                                                blue <= "00";
                                                     else
                                                                red <= (others => '0');
                                                                green <= (others => '0');
blue <= (others => '0');
                                                     end if;
                                           elsif (nibble="1111") then
                                                     if (h_count<200) then
                                                                red <= "111";
                                                                green <= "000";
                                                                blue <= "00";
                                                      elsif (h count>200 and h count<400) then
                                                                red <= "000";
                                                                green <= "000";
                                                                blue <= "11";
                                                      elsif (h_count>400 and h_count<600) then
                                                                red <= "111";
                                                                green <= "111";
                                                                blue <= "00";
                                                      elsif (h_count>600 and h_count<800) then
                                                                red <= "000";
                                                                green <= "111";
                                                                blue <= "00";
                                                     end if;
                                           end if;
                                else
                                           if (pr_state=idle) then
                                                     if (h_count<200) then
    red <= "111";</pre>
                                                                green <= "000";
                                                                blue <= "00";
                                                      elsif (h_count>200 and h_count<400) then
                                                                red <= "000";
                                                                green <= "000";
                                                                blue <= "11";
                                                      elsif (h_count>400 and h_count<600) then
                                                                red <= "111";
                                                                green <= "111";
                                                                blue <= "00";
                                                      elsif (h_count>600 and h count<800) then
                                                                red <= "000";
                                                                green <= "111";
                                                                blue <= "00";
                                                     end if:
                                           \verb|elsif (pr_state=ff)| then \\
                                                     if (h count>250 and h count<400 and v count>50
and v count<250) then
                                                                if (F(v_count-50)(h_count-225)='1')
then
                                                                           red <= (others => '1');
                                                                           green <= (others => '1');
blue <= (others => '1');
                                                                else
                                                                           red <= (others => '0');
                                                                           green <= (others => '0');
                                                                           blue <= (others => '0');
                                                                end if;
                                                     elsif (h count>400 and h count<550 and
v_{count}>50 and v_{count}<250) then
                                                                if (F(v count-50) (h count-375) = '1')
then
                                                                           red <= (others => '1');
                                                                           green <= (others => '1');
blue <= (others => '1');
                                                                else
```

blue <= "00";

```
red <= (others => '0');
                                                                        green <= (others => '0');
blue <= (others => '0');
                                                             end if;
                                                   end if;
                                         elsif (pr state=gg) then
                                                   if (h_count>250 and h_count<400 and v_count>50
and v count<250) then
                                                             if (G(v count-50)(h count-225)='1')
then
                                                                        red <= (others => '1');
                                                                        green <= (others => '1');
blue <= (others => '1');
                                                             else
                                                                        red <= (others => '0');
                                                                        green <= (others => '0');
blue <= (others => '0');
                                                             end if:
                                                   elsif (h count>400 and h count<550 and
v count>50 and v count<250) then
                                                             if (G(v count-50)(h count-375)='1')
then
                                                                        red <= (others => '1');
                                                                        green <= (others => '1');
                                                                        blue <= (others => '1');
                                                             else
                                                                        red <= (others => '0');
                                                                        green <= (others => '0');
blue <= (others => '0');
                                                             end if:
                                                   end if;
                                         else
                                                   -- Case que verifica o level e chama procedure
para impressão dos devidos caracteres
                                                   case lvl is
                                                   when 1 => characters (h count, v count, zero,
zero, red var, green var, blue var);
                                                   when 2 => characters (h count, v count, zero,
one, red_var, green_var, blue_var);
                                                   when 3 => characters (h count, v count, zero,
two, red var, green var, blue var);
                                                   when 4 => characters (h_count, v_count, zero,
three, red var, green var, blue var);
                                                   when 5 => characters (h count, v count, zero,
four, red var, green var, blue var);
                                                   when 6 => characters (h count, v count, zero,
five, red var, green var, blue var);
                                                   when 7 => characters (h count, v count, zero,
six, red var, green var, blue var);
                                                   when 8 => characters (h_count, v_count, zero,
seven, red_var, green_var, blue_var);
                                                   when 9 => characters (h count, v count, zero,
eight, red var, green var, blue var);
                                                   when 10 => characters (h_count, v_count, zero,
nine, red var, green var, blue var);
                                                   when 11 => characters (h count, v count, one,
zero, red var, green var, blue var);
                                                   when 12 => characters (h count, v count, one,
one, red var, green var, blue var);
                                                   when 13 => characters (h count, v count, one,
```

two, red var, green var, blue var);

```
when 14 => characters (h count, v count, one,
three, red var, green var, blue var);
                                                        end case;
                                                        -- Associação das variáveis com as saidas RGB
                                                        red <= red var;</pre>
                                                        green <= green var;
                                                        blue <= blue var;
                                            end if;
                                 end if;
                      -- Tempo em que nada é enviado no RGB
       else
                                 red <= (others => '0');
                                 green <= (others => '0');
                                 blue <= (others => '0');
      end if;
    end if;
  end process;
end behavior;
Nexys2.ucf
NET "LEDs(0)" LOC = "J14";
NET "LEDs(1)" LOC = "J15";
NET "LEDs(2)" LOC = "K15";
NET "LEDs(3)" LOC = "K14";
NET "display(6)" LOC = "L18";
NET "display(5)" LOC = "F18";
NET "display(4)" LOC = "D17";
NET "display(3)" LOC = "D16";
NET "display(2)" LOC = "G14";
NET "display(1)" LOC = "J17";
NET "display(0)" LOC = "H14";
NET "anode(0)" LOC = "F17";
NET "anode(1)" LOC = "H17";
NET "anode(2)" LOC = "C18";
NET "anode(3)" LOC = "F15";
NET "red(0)" LOC = "R9";
NET "red(1)" LOC = "T8";
NET "red(2)" LOC = "R8";
NET "green(0)" LOC = "N8";
NET "green(1)" LOC = "P8";
NET "green(2)" LOC = "P6";
NET "blue(0)" LOC = "U5";
NET "blue(1)" LOC = "U4";
NET "h_sync" LOC = "T4";
NET "v_sync" LOC = "U3";
NET "clk" LOC = "B8";
NET "diff(0)" LOC = "K18";
NET "diff(1)" LOC = "K17";
NET "button(0)" LOC = "B18";
NET "button(1)" LOC = "D18";
NET "button(2)" LOC = "E18";
NET "button(3)" LOC = "H13";
NET "start" LOC = "G18";
NET "reset" LOC = "H18";
```

Genius TB.vhd

```
LIBRARY ieee;
USE ieee.std logic 1164.ALL;
ENTITY Genius_TB IS
END Genius_TB;
ARCHITECTURE behavior OF Genius TB IS
    COMPONENT Genius
         clk: IN std logic;
         start : IN std_logic;
reset : IN std_logic;
          diff: IN std logic vector(1 downto 0);
         button: IN std logic vector(3 downto 0);
         display : OUT std_logic_vector(6 downto 0);
anode : OUT std_logic_vector(3 downto 0);
         leds : OUT std logic vector(3 downto 0);
         h_sync : OUT std_logic;
v_sync : OUT std_logic;
         red : OUT std_logic_vector(2 downto 0);
         green : OUT std_logic_vector(2 downto 0);
         blue : OUT std logic vector(1 downto 0)
    END COMPONENT;
   --Inputs
   signal clk : std logic := '0';
   signal start : std_logic := '0';
signal reset : std_logic := '0';
   signal diff : std\_\overline{logic\_vector}(1 downto 0) := (others => '0');
   signal button : std_logic_vector(3 downto 0) := (others => '0');
          --Outputs
   signal display : std_logic_vector(6 downto 0);
   signal anode : std logic vector(3 downto 0);
   signal leds : std logic vector(3 downto 0);
   signal h_sync : std_logic;
   signal v_sync : std_logic;
   signal red : std_logic_vector(2 downto 0);
   signal green : std logic vector(2 downto 0);
   signal blue : std logic vector(1 downto 0);
   -- Clock period definitions
   constant clk period : time := 10 ns;
BEGIN
          -- Instantiate the Unit Under Test (UUT)
   uut: Genius PORT MAP (
          clk => clk,
           start => start,
           reset => reset,
          diff => diff,
          button => button,
          display => display,
           anode => anode,
           leds => leds,
          h sync => h sync,
          v sync => v sync,
          red => red,
          green => green,
          blue => blue
        );
   -- Clock process definitions
   clk process :process
   begin
                    clk <= '0';
                    wait for clk period/2;
                    clk <= '1';
                    wait for clk period/2;
   end process;
```

```
-- Stimulus process
stim_proc: process
begin
                  reset<='1';
                  button<="0000";
                  diff<="00";
                  start<='0';
   wait for 100 ns;
                  reset<='0';
                  diff<="10";
                  start<='1';
                  wait for 100 ns;
                  start<='0';
                  wait for 600 us;
button<="1000";</pre>
                  wait for 50 us;
                  button<="0000";
                  wait for 900 us;
                  button<="1000";
                  wait for 50 us;
                  button<="0000";
                  wait for 100 us;
button<="0100";</pre>
                  wait for 50 us;
                  button<="0000";
                  wait for 1200 us;
                  button<="1000";
                  wait for 50 us;
                  button<="0000";
                  wait for 100 us;
                  button<="0100";
                  wait for 50 us;
button<="0000";
                  wait for 100 us;
                  button<="0010";
                  wait for 50 us;
                  button<="0000";
                  wait for 1500 us;
                  button<="1000";
                  wait for 50 us;
button<="0000";
                  wait for 100 us;
                  button<="0100";
                  wait for 50 us;
                  button<="0000";
                  wait for 100 us;
                  button<="0010";
                  wait for 50 us;
                  button<="0000";
                  wait for 100 us;
                  button<="0001";
                  wait for 50 us;
                  button<="0000";
                  wait for 1800 us;
                  button<="1000";
                  wait for 50 us;
                  button<="0000";
                  wait for 100 us;
button<="0100";</pre>
                  wait for 50 us;
                  button<="0000";
                  wait for 100 us;
                  button<="0010";
                  wait for 50 us;
                  button<="0000";
                  wait for 100 us;
button<="0001";</pre>
                  wait for 50 us;
                  button<="0000";
                  wait for 100 us;
                  button<="0001";
                  wait for 50 us;
                  button<="0000";
                  wait for 2100 us;
```

```
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 2400 us;
button<="1000";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 2700 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
```

```
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 3000 us;
button<="1000";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 3300 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
```

```
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 3600 us;
button<="1000";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 3900 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
```

```
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 4200 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 4500 us;
```

```
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";</pre>
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0100";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0010";
wait for 50 us;
button<="0000";</pre>
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="1000";
wait for 50 us;
button<="0000";
wait for 100 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
start<='1';
wait for 100 ns;
start<='0';
wait for 600 us;
button<="0001";
wait for 50 us;
button<="0000";
wait for 100 us;
start<='1';
wait for 100 ns;
start<='0';
wait for 5 ms;
```

end process;

wait;

END;

Global_pack.vhd

package body global pack is

```
constantes
library IEEE;
use IEEE.STD LOGIC 1164.all;
use work.rom_characters.all;
package global pack is
                                    type states is (idle, print, dly, waiting, upack, dbc, downack, nextlyl, qq, ff); --
Estados da máquina
                                    type pa_ed is array (22 downto 0) of std logic vector (6 downto 0); -- Arranjo da
mensagem inicial
                                    type score is array (3 downto 0) of std logic vector (6 downto 0); -- Arranjo da
pontuação
                                    type enable is array (3 downto 0) of std_logic_vector (3 downto 0); -- Arranjo dos
 estados dos anodos
                                    type sequence is array (14 downto 0) of std logic vector (3 downto 0); -- Arranjo de
uma seguência
                                    -- Constantes que definem as sequências possiveis de 14 cores para o jogo constant seq0: sequence := ("0001","1000","0010","0010","0100","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","100
                                    constant seq1: sequence := ("0001","1000","0001","0100","0100","0100","0100","1000","0001","1000","0100","0100","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","1000","100
                                    constant seq2: sequence := ("0010","1000","0001","0010","0100","0010","1000",
"0100","1000","0001","0001","0010","0100","0001", "0000");
                                    constant seq3: sequence := ("0100","0010","0001","0010","1000","1000","0100",
"0001","0010","1000","0001","0100","0100","0100", "0000");
                                    constant seq4: sequence := ("0001","0001","1000","0010","0100","1000","1000","1000","0100","0100","0100","0100","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","0010","001
                                    constant seq5: sequence := ("0100","1000","0010","0010","0100","1000","1000","1000","0100","0010","0001","0001","0001","0000");
                                     constant seq6: sequence := ("1000","0010","0010","0100","0100","1000","0001",
                                    "0100", "1000", "0100", "0001", "0100", "0100", "0100", "0000");
                                     constant seq7: sequence := ("0001","1000","0010","0010","0100","1000","1000",
                                     "0100", "0001", "0010", "0001", "1000", "1000", "0000");
                                    constant seq8: sequence := ("0010","0001","0100","0001","0010","1000","0100",
                                    "0100","1000","0001","1000","0010","0100","0010", "0000");
                                     constant seq9: sequence := ("0100","1000","0001","0010","0010","1000","0001",
                                      "0100","1000","0001","0010","0001","0100","0001", "0000");
                                    type mult seq is array (9 downto 0) of sequence; -- Arranjo das 10 sequências pré-
 definidas
                                     constant sequences: mult_seq := (seq9, seq8, seq7, seq6, seq5, seq4, seq3, seq2,
 seq1, seq0); -- Constante com as 10 sequências criadas
                                      -- Declaração da procedure que imprime os caracteres na região superior do monitor
                                    procedure characters (h_count, v_count: in integer;
                                                                                                                                                                                                                                                                                                            first char,
 secon char: in char data;
                                                                                                                                                                                                                                                                                                           red: out
 std logic vector (2 downto 0);
                                                                                                                                                                                                                                                                                                           green: out
 std logic vector (2 downto 0);
                                                                                                                                                                                                                                                                                                           blue: out
std logic vector (1 downto 0));
end global pack;
```

-- Package global que implementa os tipos necessários no projeto, bem como procedures e

```
-- Procedure que recebe os valores de caracteres e estabelece a lógica para
impressão na tela
         procedure characters (h_count, v_count: in integer;
                                                                             first char,
secon char: in char data;
                                                                             red: out
std logic vector (2 downto 0);
                                                                             green: out
std logic vector (2 downto 0);
                                                                             blue: out
std logic vector (1 downto 0)) is
         begin
                   if (h count>250 and h count<400 and v count>50 and v count<250) then
                            if (first_char(v_count-50)(h_count-225)='1') then
                                     red := (others => '1');
                                      green := (others => '1');
                                     blue := (others => '1');
                            else
                                     red := (others => '0');
                                      green := (others => '0');
                                     blue := (others => '0');
                            end if;
                   elsif (h_count>400 and h_count<550 and v_count>50 and v_count<250) then
                            red := (others => '1');
green := (others => '1');
                                     blue := (others => '1');
                            else
                                      red := (others => '0');
                                      green := (others => '0');
                                      blue := (others => '0');
                            end if:
                   end if:
         end characters;
end package body;
```

Rom characters.vhd

```
-- Package que define as constantes com o desenho de cada um dos caracteres utilizados
-- Números: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
-- Letras: F, G
-- Constantes de 200x200 pixels
library IEEE;
use IEEE.STD LOGIC 1164.all;
package rom_characters is
  constant h: integer := 200;
  constant v: integer := 200;
  type char data is array (0 to v-1) of std logic vector (0 to h-1);
  constant zero: char data := (
000000000000000".
00000000000000",
00000000000000,
0000000000000",
```

```
00000000000000"
000000000000000
",00000000000000
000000000000000".
"00000000000000"
000000000000000".
00000000000000
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
000000000000000
0000000000000
000000000000000".
0000000000000",
000000000000000".
00000000000000
",00000000000000
.,000000000000000
```

```
00000000000000"
000000000000000
",00000000000000
000000000000000.
0000000000000000",
.",00000000000000
00000000000000".
000000000000000".
000000000000000.
.",00000000000000
",00000000000000
000000000000000
00000000000000"
000000000000000.
0000000000000",
000000000000000".
00000000000000
",00000000000000
.,000000000000000
```

```
00000000000000"
000000000000000",
0000000000000000
.",00000000000000
0000000000000000",
.",00000000000000
0000000000000000.
000000000000000".
.",00000000000000
",00000000000000
00000000000000",
000000000000000",
0000000000000",
00000000000000
constant one: char data := (
000000000000000."
00000000000000".
00000000000000",
",00000000000000
```

```
00000000000000
000000000000000
",00000000000000
000000000000000".
0000000000000000
000000000000000".
000000000000000
000000000000000.
00000000000000.".
.",00000000000000
",00000000000000
000000000000000
0000000000000
000000000000000".
0000000000000",
000000000000000".
00000000000000
",00000000000000
.,000000000000000
```

```
00000000000000"
000000000000000",
0000000000000000
000000000000000.
0000000000000000",
.",00000000000000
0000000000000000.
000000000000000".
.",00000000000000
",00000000000000
00000000000000",
000000000000000",
0000000000000".
0000000000000",
.",00000000000000
00000000000000
);
constant two: char data := (
000000000000000
00000000000000".
\\
```

```
00000000000000"
00000000000000"
",00000000000000
00000000000000.
"00000000000000"
000000000000000".
00000000000000
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
000000000000000
0000000000000
000000000000000".
0000000000000",
000000000000000".
00000000000000
",00000000000000
.",0000000000000
```

```
00000000000000"
",00000000000000
.",00000000000000
0000000000000000",
.",00000000000000
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
000000000000000
00000000000000"
000000000000000.
0000000000000",
000000000000000".
00000000000000
",00000000000000
.",0000000000000
```

```
00000000000000"
000000000000000",
0000000000000000
.",00000000000000
0,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,0
0000000000000000",
.",00000000000000
0000000000000000.
00000000000000.
.",00000000000000
",00000000000000
00000000000000",
000000000000000",
0000000000000",
.",00000000000000
00000000000000"
00000000000000
    constant three: char data := (
0000000000000",
",00000000000000
0000000000000",
```

```
00000000000000"
000000000000000
",00000000000000
000000000000000.
00000000000000
000000000000000".
00000000000000
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
000000000000000
0000000000000
000000000000000".
0000000000000",
000000000000000".
00000000000000
",00000000000000
.",0000000000000
```

```
00000000000000"
00000000000000"
0000000000000000
000000000000000.
,"00000000000",
00000000000000.
000000000000000
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
000000000000000
00000000000000"
000000000000000".
0000000000000",
000000000000000".
00000000000000
",00000000000000
```

.,00000000000000

0000000000000",

```
00000000000000"
000000000000000",
0000000000000000
.",00000000000000
000000000000000",
.",00000000000000
0000000000000000.
000000000000000".
.",00000000000000
",00000000000000
00000000000000",
000000000000000",
0000000000000",
.",00000000000000
00000000000000"
0,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,0
000000000000000.
000000000000000
    constant four: char data := (
00000000000000".
```

```
00000000000000"
000000000000000
000000000000000
.",00000000000000
000000000000000",
.",00000000000000
00000000000000".
000000000000000".
000000000000000.
.",00000000000000
",00000000000000
000000000000000,
0000000000000",
.",00000000000000
00000000000000"
000000000000000.
",00000000000000
00000000000000".
```

```
00000000000000"
000000000000000
0000000000000000
000000000000000.
0000000000000000",
.",00000000000000
000000000000000.
000000000000000.
000000000000000.
.",00000000000000
",00000000000000
000000000000000
00000000000000"
000000000000000.
0000000000000",
000000000000000".
00000000000000
",00000000000000
.,00000000000000
```

```
00000000000000"
000000000000000",
0000000000000000
.",00000000000000
0000000000000000",
.",00000000000000
0000000000000000.
000000000000000".
.",00000000000000
000000000000000.
00000000000000",
00000000000000"
0000000000000",
.",00000000000000
",00000000000000
000000000000000.
",00000000000000
00000000000000"
constant five: char data := (
0000000000000",
```

```
00000000000000"
00000000000000"
",00000000000000
000000000000000.
0000000000000000",
.",00000000000000
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
000000000000000
00000000000000"
000000000000000".
0000000000000",
000000000000000".
00000000000000
",00000000000000
.,00000000000000
```

```
00000000000000
000000000000000
",00000000000000
000000000000000000
000000000000000".
.",00000000000000
.,00000000000000
000000000000000".
.",00000000000000
",00000000000000
00000000000000",
000000000000000",
0000000000000",
.",00000000000000
00000000000000"
000000000000000.
",00000000000000
00000000000000".
```

```
00000000000000"
00000000000000"
",00000000000000
.",00000000000000
0000000000000000",
.",00000000000000
000000000000000
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
000000000000000
00000000000000"
000000000000000".
0000000000000",
000000000000000".
00000000000000
",00000000000000
.,00000000000000
```

```
00000000000000"
000000000000000
0000000000000000
.",00000000000000
000000000000000",
.",00000000000000
.",0000000000000
.",0000000000000
.",00000000000000
",00000000000000
00000000000000"
000000000000000".
0000000000000",
.",00000000000000
",00000000000000
000000000000000.
",00000000000000
00000000000000".
\\
00000000000000
);
constant six: char data := (
```

```
00000000000000
000000000000000
",00000000000000
0000000000000000000
"00000000000000"
.",00000000000000
.",0000000000000
00000000000000.
000000000000000.
.",00000000000000
",00000000000000
00000000000000"
0000000000000",
.",00000000000000
00000000000000"
000000000000000.
",00000000000000
00000000000000".
```

```
00000000000000"
000000000000000
",00000000000000
000000000000000.
000000000000000",
.",00000000000000
",00000000000000
00000000000000.
.",00000000000000
",00000000000000
000000000000000
000000000000000.
0000000000000",
000000000000000".
00000000000000
",00000000000000
.",0000000000000
```

```
00000000000000"
000000000000000
0000000000000000
.",00000000000000
0000000000000000",
.",00000000000000
00000000000000000
000000000000000".
000000000000000.
.",00000000000000
",00000000000000
00000000000000",
0000000000000".
0000000000000",
.",00000000000000
00000000000000"
000000000000000.
",00000000000000
00000000000000".
```

```
00000000000000"
000000000000000,
0000000000000000
000000000000000.
000000000000000",
.",00000000000000
000000000000000.
000000000000000".
000000000000000.
.",00000000000000
",00000000000000
000000000000000,
0000000000000".
0000000000000",
00000000000000",
00000000000000
",00000000000000
.,00000000000000
```

```
constant seven: char data := (
```

```
00000000000000"
000000000000000
",00000000000000
000000000000000.
000000000000000",
.",00000000000000
000000000000000.
00000000000000.
.",00000000000000
",00000000000000
000000000000000
00000000000000"
000000000000000".
0000000000000",
000000000000000".
00000000000000
",00000000000000
.,00000000000000
```

```
00000000000000
000000000000000
",00000000000000
000000000000000".
00000000000000
000000000000000".
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
```

```
00000000000000"
00000000000000"
",00000000000000
000000000000000.
0000000000000000",
.",00000000000000
000000000000000
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
000000000000000
00000000000000"
000000000000000".
0000000000000",
000000000000000".
00000000000000
",00000000000000
.",0000000000000
```

```
00000000000000
000000000000000
0000000000000000
.",00000000000000
0000000000000000",
.",00000000000000
000000000000000.
000000000000000".
000000000000000.
.",00000000000000
",00000000000000
00000000000000",
0000000000000".
0000000000000",
.",00000000000000
00000000000000"
000000000000000.
",00000000000000
00000000000000".
```

```
00000000000000"
00000000000000"
",00000000000000
.",00000000000000
0000000000000000",
.",00000000000000
000000000000000".
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
000000000000000
00000000000000"
000000000000000".
0000000000000",
000000000000000".
00000000000000
",00000000000000
.,000000000000000
0000000000000",
```

```
00000000000000
000000000000000
",00000000000000
0000000000000000000
"00000000000000"
00000000000000.
00000000000000".
00000000000000.
000000000000000.
.",00000000000000
",00000000000000
00000000000000",
0000000000000".
0000000000000",
.",00000000000000
00000000000000"
000000000000000.
",00000000000000
00000000000000".
```

```
00000000000000"
000000000000000",
0000000000000000
000000000000000.
0000000000000000",
.",00000000000000
000000000000000.
000000000000000".
000000000000000.
.",00000000000000
",00000000000000
00000000000000",
0000000000000".
0000000000000",
.",00000000000000
",00000000000000
",00000000000000
.,000000000000000
```

);

```
00000000000000"
00000000000000"
",00000000000000
000000000000000.
000000000000000
00000000000000".
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
000000000000000
00000000000000"
000000000000000".
0000000000000",
000000000000000".
00000000000000
",00000000000000
```

.,000000000000000

```
00000000000000
000000000000000
",00000000000000
0000000000000000000
000000000000.
.",00000000000000
000000000000000.
000000000000000".
000000000000000.
.",00000000000000
",00000000000000
00000000000000",
0000000000000".
0000000000000",
.",00000000000000
00000000000000"
000000000000000.
",00000000000000
00000000000000".
```

```
00000000000000"
000000000000000",
0000000000000000
000000000000000.
0000000000000000",
.",00000000000000
000000000000000.
000000000000000".
000000000000000.
.",00000000000000
",00000000000000
00000000000000",
0000000000000".
0000000000000",
00000000000000",
",00000000000000
000000000000000.
",00000000000000
.",0000000000000
```

```
00000000000000
000000000000000
",00000000000000
000000000000000000
"00000000000000"
000000000000000".
000000000000000
00000000000000".
00000000000000.
.",00000000000000
",00000000000000
000000000000000
000000000000000".
0000000000000",
.",00000000000000
",00000000000000
000000000000000.
",00000000000000
00000000000000".
```

```
constant F: char_data := (
```

```
00000000000000
000000000000000
",00000000000000
000000000000000000
"00000000000000"
00000000000000.
00000000000000".
000000000000000.
000000000000000".
",00000000000000
00000000000000",
000000000000000,
0000000000000",
.",00000000000000
00000000000000"
000000000000000.
",00000000000000
00000000000000".
0000000000000",
```

```
00000000000000"
",00000000000000
000000000000000
.",00000000000000
000000000000000
00000000000000".
.",0000000000000
000000000000000".
.",00000000000000
",00000000000000
00000000000000",
000000000000000,
0000000000000",
.",00000000000000
00000000000000
",00000000000000
.",0000000000000
```

```
00000000000000
000000000000000
",00000000000000
000000000000000000
"00000000000000"
00000000000000.
00000000000000".
000000000000000.
.",00000000000000
",00000000000000
000000000000000
00000000000000"
0000000000000".
0000000000000",
.",00000000000000
",00000000000000
000000000000000.
",00000000000000
00000000000000".
0000000000000",
```

```
00000000000000"
",00000000000000
000000000000000
.",00000000000000
000000000000000
00000000000000".
.",0000000000000
000000000000000".
.",00000000000000
",00000000000000
00000000000000",
000000000000000,
0000000000000",
.",00000000000000
00000000000000"
000000000000000.
",00000000000000
00000000000000".
```

```
constant G: char_data := (
```

```
00000000000000
000000000000000
",00000000000000
000000000000000000
"00000000000000"
00000000000000.
00000000000000".
000000000000000.
.",00000000000000
",00000000000000
00000000000000"
0000000000000",
.",00000000000000
00000000000000"
000000000000000.
",00000000000000
00000000000000".
```

```
00000000000000
000000000000000",
",00000000000000
000000000000000"
"00000000000000"
00000000000000.
",00000000000000
000000000000000.
000000000000000".
",00000000000000
00000000000000"
00000000000000"
00000000000000.
0000000000000",
.",00000000000000
00000000000000"
000000000000000.
"00000000000000
00000000000000".
```

```
00000000000000"
"00000000000000
000000000000000.
0000000000000000",
.",00000000000000
00000000000000000
000000000000000".
000000000000000.
.",00000000000000
",00000000000000
00000000000000",
0000000000000".
0000000000000",
.",00000000000000
00000000000000"
",00000000000000
.,00000000000000
```

```
00000000000000
000000000000000
",00000000000000
000000000000000".
"00000000000000"
000000000000000".
00000000000000
00000000000000".
00000000000000.
000000000000000".
",00000000000000
000000000000000
00000000000000"
000000000000000.
0000000000000",
",00000000000000
"00000000000000
00000000000000".
\\
```

);

end rom_characters;