

GRUPO 5 - TURMA A

Projeto Final
de Circuitos Digitais

MC 613 - Primeiro Semestre de 2010

PROFESSOR: GUIDO ARAÚJO

HENRIQUE SERAPIÃO GONZALES	RA: 083636
MARCELO GALVÃO PÓVOA	RA: 082115
TIAGO CHEDRAOUI SILVA	RA: 082941

29 de junho de 2010

1 Introdução

Nosso projeto foi criar em VHDL um jogo inspirado no clássico original PACMAN, de autoria de Tohru Iwatani em 1980 pela NAMCO. Devido às dificuldades inerentes em se projetar um jogo relativamente complexo com lógica de *hardware* ao invés de usar um processador com barramento de memória para tal, algumas partes foram simplificadas em relação ao jogo original.

Os componentes de hardware usados foram, além da placa Altera DE1, um monitor VGA e um teclado padrão PS/2. O jogo é projetado para dois jogadores (ao contrário de apenas um no jogo original) que disputam a vitória em um labirinto, um controla dois fantasmas e o outro, o Pacman. Para mais detalhes das regras, consulte o jogo original em <http://en.wikipedia.org/wiki/Pac-Man>. O desenvolvimento do projeto está hospedado em <http://code.google.com/p/cmp613/>

O código resultou relativamente extenso, portanto o funcionamento em detalhes de suas partes não está descrita nesse texto, mas ao longo do código extensivamente comentado. Esse relatório limita-se a discutir a estrutura geral do design, as considerações tomadas para o projeto, as dificuldades encontradas e as possíveis soluções.

2 Descrição estrutural

A organização de componentes do projeto é simples. Consta de um *top-level* que instancia e comunica alguns controladores de lógica de jogo e controladores de dispositivos independentes. Uma máquina de estados principal controla o andamento do jogo, gerando sinais para os componentes trocarem informações. Algumas operações são realizadas no *top-level*, entre elas o controle de parâmetros globais do jogo - como vidas e pontuação - que determinam o fim do jogo (o pacman ou os fantasmas vencem). Devido ao fato dos sinais necessários para o desenho dos personagens na tela depender de vários outros sinais, a geração destes primeiros também está feita em um grande PROCESS no *top-level*.

Em resumo, os componentes principais são os seguintes (veja mais detalhes nas respectivas arquiteturas e instâncias).

2.1 Controladores de lógica do jogo

- CTRL_PACMAN: Componente que gera a nova posição do pacman baseado no mapa ao redor dele que recebe.
- CTRL_FANS: Componente que gera as novas posições de múltiplos fantasmas do jogo baseado nos mapas ao redor deles que recebe. Também gerencia e informa os estados atuais dos fantasmas com máquinas de estado.
- CTRL_FRUTAS: Componente que gera, "aleatoriamente", sinais que informam que uma fruta apareceu temporariamente no mapa (elas representam um bônus para o pacman). Ao capturar a fruta, esse componente é resetado.

2.2 Controladores de dispositivos

- VGACON: Faz a varredura dos pixels da tela na frequência apropriada, gerando também todos os sinais para a interface VGA. Contém internamente duas memórias RAM que representam em blocos lógicos (não foram usados pixels, veja a seção 3 para mais detalhes) o estado atual da tela

em duas camadas: cenário e *overlay* (para desenho dos personagens). *Sprites* são usados para mapear esses blocos em pixels de profundidade 3-bits.

- DISP: Controla os quatro displays de 7-segmentos da placa, usados para complementar o HUD¹ com a pontuação atual e mensagens de final de jogo. Essa parte não foi implementada na tela devido a complexidade relacionada a escrever texto na mesma.
- KBD_KEY: Realiza comunicação com o teclado, gerando sinais quando são pressionadas teclas de interesse para o jogo (cursos de movimento). Leia sobre os problemas desse componente na seção 3.

2.3 Pacotes de Definições

- PAC_DEFS: Contém todas as constantes e tipos principais usados globalmente no jogo. Note que algumas constantes específicas estão localizadas nos respectivos controladores. Armazena visualmente a estrutura do labirinto do jogo carregada em RAM. Esse pacote é destinado a ser facilmente customizável e entendido por usuários.
- PAC_SPRITES: Coleção (extensa) de constantes usadas para *sprites* e mapeamento de *sprites* enviados à tela. A maior parte deste pacote foi gerada automaticamente e não é aconselhável editá-lo manualmente.

3 Problemas e Soluções

3.1 Dificuldade de instanciar uma RAM (resolvido)

Nos estágios iniciais do projeto, houve bastante dificuldade em fazer o compilador inferir uma RAM através do *template* do fabricante. Queríamos uma RAM *Dual Port Dual Clock* que fosse inicializada com o mapa do jogo, para isso alteramos a **função de inicialização** do *template* para copiar para a RAM um array constante que continha blocos **enumerados**. Após compilar, a RAM não era reconhecida.

Solução: mudar o tipo do conteúdo da RAM de uma enumeração para um STD_LOGIC_VECTOR e, adicionalmente, fazer a carga do conteúdo inicial manualmente e externamente (sem a função supracitada) varrendo-a com contadores. Essas alterações fizeram a RAM ser reconhecida e sintetizada, porém para uma das RAMs do projeto, estranhamente, foi gerado um warning de "*Uninferred RAM due to asynchronous read logic*", que não foi resolvido mas parece não afetar o circuito.

3.2 Problemas com o componente do teclado (warnings e falhas)

O componente do teclado possui um critical warning relacionado ao timing, porém ele não afeta o jogo.

Além disso, a inicialização do componente, posterior ao carregamento do jogo na placa, pode apresentar um comportamento indevido. Esse comportamento que impossibilita o recebimento de três teclas simultâneas do teclado (poderia nesse caso receber duas, uma ou nenhuma).

Solução: Ao reiniciar o jogo manualmente, o problema é resolvido.

¹ *Head-Up Display*: conjunto de símbolos exibidos para informar ao jogador as condições atuais do jogo

3.3 Limitação de 3 teclas simultâneas no teclado (não resolvido)

Quando três teclas já estão pressionadas no teclado, teclas adicionais são ignoradas até que solte-se uma das teclas originais. Assim, um jogador mal-intencionado pode impedir o outro de jogar, conseguindo vencer desonestamente. Problema provavelmente causado por uma limitação do *hardware* (teclado) ou pela interface PS/2.

Solução: usar dois teclados (um por jogador) conectados em duas placas, sendo que a segunda apenas envia para a primeira informações do seu teclado durante o jogo. Porém, não se justifica adicionar uma interface, uma placa e um teclado só para corrigir esse problema. Ademais, o uso da segunda placa seria subaproveitado.

3.4 Dificuldade em fazer o som funcionar. Complicado pela NIOS e mal documentado em VHDL.

A proposta inicial de colocar o som, era desenvolver um som do jogo pacman próximo ao original. Havia duas possibilidades para sua implementação, uma usando o codec de audio documentado no site do altera, ou a implementação da Nios2. Optou-se pelo segundo método que consistia em usar o SOPC Builder (programa integrado ao quartus) que geraria uma NIOS com os componentes especificados pelo usuário.

Os componentes mínimos eram uma Cpu, um componente de memória e o componente do som fornecido no site do altera. Para testes iniciais da NIOS usamos componentes de saída para os leds e de entrada para os switches. Implementado a NIOS devíamos instanciá-la em um top-level e piná-lo de acordo com os manuais do altera.

Após isso, devíamos utilizar o programa NIOS II EDS, que utilizaria a NIOS instanciada e geraria um projeto em C/C++ para a criação de um software. Ou seja, a NIOS atua teoricamente como um computador e o código em C passaria a ser o software.

O passo iniciais de teste foi ligar os endereços dos switches ao endereços dos leds. Assim, os leds receberiam o valor dos switches. Foi um suceso a instancição NIOS. Bastava portanto, utilizar o componente do som para gerar a música.

Para passar a música para a placa, poderíamos usar uma das seguintes memória: SD Card, SRAM, RAM, ROM, FLASH. Como SRAM e RAM são voláteis, ao desligar a placa, perderíamos seus dados. Além disso, como não possuíamos um cartão de memória compatível com a entrada do SD Card, sua utilização também foi descartada. Restou portanto a utilização da memória flash. O site do altera possui pouca documentação prática sobre a utilização de memória flash, apresentado apenas uma grande documentação teórica, logo perdemos muito tempo aprendendo a instanciá-la na NIOS.

Contudo, não foi possível realizar uma comunicação com o dispositivo em flash para sua leitura, pois a NIOS não encontrava o dispositivo através da função “alt_flash_open_dev(nome da flash)”. Esse problema foi presenciado por diversos usuários no fórum do site do altera, e não existia uma solução, sendo um problema relacionado à ligação de pinos no código da NIOS citado como problema, mas não foi demonstrado sua solução.

Apesar de não conseguir ler, foi possível escrever na flash através de outro programa fornecido pelo altera “Altera Monitor Program”, com o qual pode-se apagar a memória flash e Sram, assim como, colocar arquivos inteiros, ou apenas alterar alguns bits de determinado endereço.

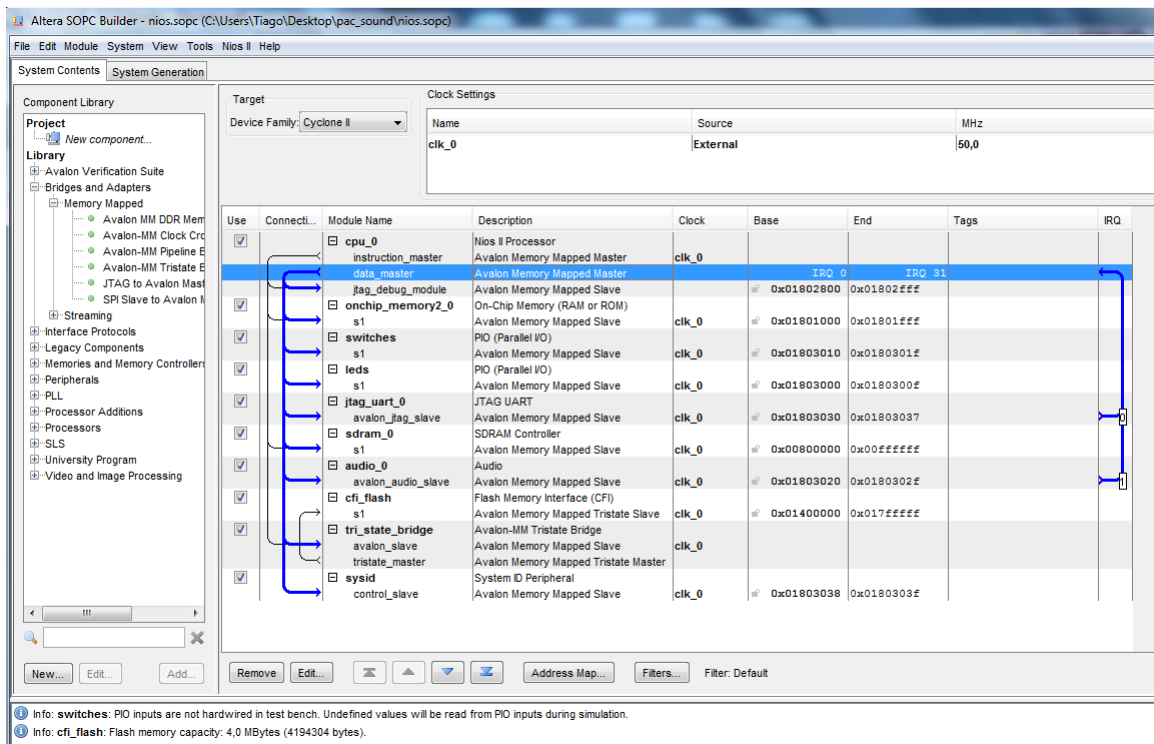


Figura 3.1: NIOS implementada para o laboratório

3.5 Projeto da inteligência artificial dos fantasmas (resolvido)

Esse não chegou a ser um problema de fato, mas implementar uma movimentação "inteligente" para pegar o pacman em hardware seria, potencialmente, bastante trabalhoso. Além disso, o mau funcionamento dessa parte poderia prejudicar a qualidade do jogo, então ela seria uma parte muito laboriosa do projeto. No jogo original, cada fantasma funciona com uma IA diferente, porém simples. Essa implementação seria muito mais confortável e natural programando um algoritmo.

Solução: Resolvemos criar um segundo jogador que controla os fantasmas. Apesar disso, eles ainda voltam sozinho para a "jaula"², algo mais simples de projetar. Resultado: menos dor-de-cabeça no hardware e uma jogabilidade diferente.

3.6 Arrays usados como constantes (não resolvido)

Há dois grandes mapas constantes no jogo: o mapa inicial e o de percurso automático dos fantasmas. Além disso, há três vetores enormes de *sprites* para cada um dos mais de 300 tipos de blocos existentes no jogo. Apesar de isso facilitar muito a edição dos mapas diretamente no código, usar constantes faz com que o conteúdo desses vetores seja sintetizado através de células lógicas, enquanto na verdade eles funcionam como memória ROM.

²Trata-se do percurso automático do fantasma ao centro do mapa quando este é "morto" pelo pacman

Solução: Armazenar esses dados em memória ROM, reduzindo drasticamente o número de células lógicas. Isso não foi feito pelos seguintes motivos:

- Não havia praticamente mais blocos de memória on-chip (M4K) disponíveis, muitos foram usados para as RAMs.
- Essas ROMs seriam inicializadas de forma binária através de um arquivo .MIF. Além do trabalho em converter enumerações em binário e endereçar corretamente, não seria mais tão simples manipular e modificar a longa lista de sprites³. Idealmente, deveríamos trabalhar com um programa que permitisse editar um .MIF de pixels RGB organizados na forma de uma matriz.

3.7 Memória de vídeo "falsa" causa bordas indesejadas (não resolvido)

A escassez de memória *on-chip* e a necessidade de se usar uma resolução relativamente alta nos levou a criar uma memória de vídeo "falsa" para o projeto. A lógica dos elementos visuais do jogo foi dividida em um conjunto de tipos enumerados de blocos lógicos. Conseguimos distribuir esses elementos na tela em uma área de 128x96 blocos. Logo, fizemos a memória de vídeo armazenar ao invés da cor dos 640x480 pixels reais da tela, o tipo do bloco em cada posição. Dessa forma, cada bloco consiste numa área de 5x5 pixels da tela.

Apesar de economizar memória (já que não há um número tão grande de blocos diferentes), esse método faz aparecer uma borda preta em volta de blocos que estão na mesma memória cujos desenhos não ocupam toda a área do bloco. Isso ocorre pois não é possível sobrepor dois blocos num mesmo bloco, pois isso geraria um tipo de bloco indefinido. Esse é um problema essencialmente estético.

Solução: O uso de uma memória de vídeo com 640x480 pixels resolveria o problema. Isso poderia ser implementado em memória off-chip (SRAM ou SDRAM), mas a interface seria mais complicada.

3.8 Pequenas falhas visuais na imagem enviada ao monitor (resolvido)

A lógica seletora dos pixels dos sprites está localizada entre a leitura da memória de vídeo e os pinos de saída da VGA. Devido ao grande número de sprites, essa lógica consiste em enormes multiplexadores que, naturalmente, geram atrasos para a estabilização do sinal. Esses atrasos provavelmente eram a causa do surgimento de pequenas faixas (menores que um pixel) coloridas na tela. No entanto, nenhum *warning* ou problema de temporização era reportado pelo compilador.

Solução: Ao se registrar em *flip-flop* os dados de pixel antes de enviar para os pinos da VGA, o problema foi resolvido. Acreditamos que assim o compilador reconheceu um caminho crítico do circuito e ajustou os parâmetros de *timing* para satisfazer a lógica desejada. Sem o registrador, a estabilidade do caminho em questão não era considerada pelo compilador.

3.9 Desperdício de memória na RAM de *overlay* (não resolvido)

Todos os desenhos de overlay são formados por um grupo de 5x5 blocos juntos (25x25 pixels). Então, faria mais sentido haver um único bloco lógico no lugar de 25 blocos distintos, pois eles sempre aparecem juntos. Porém, não adianta simplesmente aumentar em cinco vezes o tamanho do bloco, pois seu posicionamento deve ter uma resolução de um bloco pequeno, senão a movimentação seria muito grosseira. Esse problema não estaria presente em uma memória de vídeo "real".

³Na verdade, a lista foi gerada automaticamente aos poucos a partir de um programa em C feito para exportar uma planilha para a lista de constantes binárias

Solução: Projetar alguma maneira avançada de salvar apenas um bloco central em uma região 5x5 mas fazer com que a leitura da memória descubra que, em volta dele, há um mapeamento de sprites relativos ao bloco central. Por exemplo, quando for requisitado o bloco localizado em uma certa posição (x, y) , se não houver um bloco ali, um componente deve verificar se existe algum bloco ao redor dessa posição (no retângulo limitado por $(x - 2, y - 2)$ e $(x + 2, y + 2)$). O problema é que isso deve ser feito em um ciclo de *clock* preferencialmente, o que dificulta o projeto.

4 Descrição básica de E/S

Teclado:

[W][A][S][D]	Controle na forma de cursor para o fantasma verde
[I][J][K][L]	Controle na forma de cursor para o fantasma vermelho
[Num 8][Num 4][Num 5][Num 6]	Controle na forma de cursor para o pacman

Placa:

KEY3	Reset assíncrono para todo o circuito
LEDG7	Indicador de <i>Game Over</i>
LEDG2..LEDG0	Número de teclas lidas no teclado
HEX3..HEX0	Mostrador de pontos e mensagens
VGA	Saída de vídeo para monitor (640x480 @60Hz)

5 Códigos em VHDL

Por simplicidade, foram omitidos no relatório o arquivo de sprites e os das componentes da interface com o teclado.

5.1 Top-Level

Listing 1: Top Level

```

1 -- Toplevel do jogo inspirado no PACMAN original
2 -- Disciplina MC613 1s/2010
3 -- 26 de Junho de 2010
4 -- Design em VHDL para placa Altera DE1
5 -- Compilado usando Quartus II 9.1 SP1 Web Edition
6
7 LIBRARY ieee;
8 USE ieee.STD_LOGIC_1164.all;
9 USE ieee.NUMERIC_STD.all;
10 USE work.PAC_DEFS.all;
11 USE work.PAC_SPRITES.all;
12
13 ENTITY pacman IS
14     PORT (
15         clk27M, clk24M          : IN STD_LOGIC;
16         reset_button            : IN STD_LOGIC;
17         red, green, blue        : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
18         hsync, vsync            : OUT STD_LOGIC;
19         LEDG                     : BUFFER STD_LOGIC_VECTOR(7 DOWNTO 5);
20         PS2_DAT                 : INOUT STD_LOGIC;
21         PS2_CLK                 : INOUT STD_LOGIC;
22         SEG0, SEG1, SEG2, SEG3   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
23         endgame                 : OUT STD_LOGIC;
24     );
25 END pacman;
26
27 ARCHITECTURE comportamento OF pacman IS
28     SIGNAL rstn: STD_LOGIC;          -- reset active low

```

```

29 SIGNAL restartn: STD_LOGIC; -- Usado quando o pacman morre (active low)
30 SIGNAL le_cenario: STD_LOGIC; -- Informa quando o cenário está sendo recarregado
31
32 -- Interface com a memória de vídeo do controlador
33 SIGNAL we: STD_LOGIC; -- write enable ('1' p/ escrita)
34 SIGNAL cen_addr: INTEGER
35     range 0 to SCR_HGT*SCR_WDT-1; -- ENDereco mem. vga
36 SIGNAL block_in, block_out: t_blk_sym; -- dados trocados com a mem. vga
37 SIGNAL vga_pixel_out: t_color_3b;
38
39 -- Sinais dos contadores de linhas e colunas utilizados para percorrer
40 -- as posições da memória de vídeo (pixels) no momento de construir um quadro.
41 SIGNAL line: INTEGER range 0 to SCR_HGT-1; -- linha atual
42 SIGNAL col: INTEGER range 0 to SCR_WDT-1; -- coluna atual
43 SIGNAL col_rstn: STD_LOGIC; -- reset do contador de colunas
44 SIGNAL col_enable: STD_LOGIC; -- enable do contador de colunas
45 SIGNAL line_rstn: STD_LOGIC; -- reset do contador de linhas
46 SIGNAL line_enable, line_inc: STD_LOGIC; -- enable do contador de linhas
47 SIGNAL fim_varredura: STD_LOGIC; -- '1' quando um quadro terminou de ser
48 -- escrito na memória de vídeo
49
50 -- Especificação dos tipos e sinais da máquina de estados de controle
51 TYPE estado_t IS (POWERUP, CARREGA_MAPA, ESTADO_INICIAL, PERCORRE_QUADRO,
52     ATUALIZA_LOGICA_1, ATUALIZA_LOGICA_2, MEMORIA_WR,
53     REINICIO, FIM_JOGO);
54 SIGNAL estado: estado_t := POWERUP;
55 SIGNAL pr_estado: estado_t := POWERUP;
56
57 -- sinais que servem como enable de várias velocidades
58 SIGNAL atua_en: STD_LOGIC_VECTOR(0 to VEL_NO-1);
59 SIGNAL display_en: STD_LOGIC;
60 SIGNAL disp_count: INTEGER range 0 to 27000000;
61 SIGNAL sig_blink: UNSIGNED(6 downto 0); -- enables com duty de 50%
62
63 -- Sinais de desenho em overlay sobre o cenário do jogo
64 SIGNAL varre_tela: STD_LOGIC;
65 SIGNAL ovl_blk_in: t_ovl_blk_sym;
66
67 -- Sinais para um contador utilizado para atrasar
68 -- a frequência da atualização
69 SIGNAL contador, long_cont: INTEGER range 0 to DIV_FACT-1;
70 SIGNAL timer, long_timer: STD_LOGIC; -- vale '1' quando o contador chegar ao fim
71 SIGNAL timer_rstn, timer_enable: STD_LOGIC;
72
73 COMPONENT counter IS
74 PORT (clk, rstn, en: IN STD_LOGIC;
75     max: IN INTEGER;
76     q: OUT INTEGER);
77 END COMPONENT counter;
78
79
80 -- Sinais de controle da lógica do jogo
81
82 SIGNAL got_coin, got_spc_coin: STD_LOGIC; -- informa se obteve moeda no ultimo movimento
83 SIGNAL reg_coin_we: STD_LOGIC;
84 -- quantidade de moedas restantes normais para vencer o jogo
85 SIGNAL q_rem_moedas: INTEGER range 0 to 255 := 240;
86 SIGNAL q_vidas: INTEGER range 0 to 5 := 3;
87 SIGNAL q_pontos: INTEGER range 0 to 9999 := 0;
88 SIGNAL vidas_arv: STD_LOGIC_VECTOR(2 downto 0);
89 SIGNAL update_info: STD_LOGIC;
90 SIGNAL fruta_id: t_fruta_id;
91 SIGNAL got_fruta, fruta_rstn: STD_LOGIC;
92 SIGNAL q_fruta_com: INTEGER range 0 to MAX_FRUTA_COM;
93 SIGNAL frutas_com: t_fruta_vet;
94 SIGNAL nwc: STD_LOGIC := '0';
95
96 -- Controle do pacman
97 SIGNAL pac_pos_x: t_pos;
98 SIGNAL pac_pos_y: t_pos;
99 SIGNAL pac_cur_dir: t_direcao;
100 SIGNAL pac_area: t_blk_sym_3x3;
101 SIGNAL pacman_dead: STD_LOGIC;
102 SIGNAL pac_fans_hit: UNSIGNED(0 to FAN_NO-1);
103 SIGNAL pac_atua, pac_move: STD_LOGIC;
104
105 -- Controle dos fantasmas
106 SIGNAL fan_pos_x: t_fans_pos;
107 SIGNAL fan_pos_y: t_fans_pos;
108 SIGNAL fan_cur_dir: t_fans_dirs;
109 SIGNAL fan_state: t_fans_states;
110 SIGNAL fan_area: t_fans_blk_sym_3x3;
111 SIGNAL fan_atua: STD_LOGIC;
112 SIGNAL fan_died: STD_LOGIC;
113
114 SIGNAL pac_key_dir: t_direcao; -- sinais lidos pelo teclado
115 SIGNAL fan_key_dir: t_fans_dirs;
116 BEGIN
117 -- Controlador VGA com duas camadas (RAMs) de blocos:
118 -- cenário e overlay, isto é, o pacman e os fantasmas
119 -- Devolve os pixels convertidos pelos sprites e os
120 -- sinais de controle do monitor

```



```

121 vga_controller: entity work.vgacon port map (
122     clk27M    => clk27M,
123     rstn      => '1',
124     vga_pixel  => vga_pixel_out,
125     data_block => block_out,
126     hsync     => hsync,
127     vsync     => vsync,
128     write_clk  => clk27M,
129     write_enable => we,
130     write_addr => cen_addr,
131     data_in    => block_in,
132     ovl_in     => ovl_blk_in,
133     ovl_we     => varre_tela);
134
135 -- Atribuição capada das cores 3b -> 12b
136 red    <= (OTHERS => vga_pixel_out(0));
137 green  <= (OTHERS => vga_pixel_out(1));
138 blue   <= (OTHERS => vga_pixel_out(2));
139
140 -- Controlador do teclado. Devolve os sinais síncronos das teclas
141 -- de interesse pressionadas (arquivo player_dir.vhd).
142 --
143 -- Devido a limitações da interface, só são lidas no máximo 3
144 -- teclas simultâneas, as adicionais serão ignoradas. Uma solução
145 -- é duplicar o componente para funcionar em dois teclados separados,
146 -- através da comunicação entre duas placas, mas é trabalhoso.
147 --
148 -- Este componente apresenta um leve problema de timing
149 -- o que pode torná-lo irresponsivo em algumas ocasiões.
150 -- Ativando o reset geralmente resolve o problema. :)
151 kbd: ENTITY WORK.kbd_key PORT MAP (
152     CLOCK_24    => clk24M,
153     KEY         => reset_button,
154     LEDG        => LEDG(7 downto 5),
155     PS2_DAT     => PS2_DAT,
156     PS2_CLK     => PS2_CLK,
157     p1_dir      => pac_key_dir,
158     p2_dir      => fan_key_dir(0),
159     p3_dir      => fan_key_dir(1)
160 );
161
162 -- Módulo que controla os displays 7-seg imprimindo
163 -- mensagens no decorrer do jogo e a pontuação atual
164 display: ENTITY WORK.disp PORT MAP (
165     CLK         => clk27M,
166     EN         => display_en,
167     VIDAS       => q_vidas,
168     PNT         => q_pontos,
169     PEDRAS      => q_rem_moedas,
170     seg0        => SEG0,
171     seg1        => SEG1,
172     seg2        => SEG2,
173     seg3        => SEG3
174 );
175
176 -- Contador usado para gerar enable lento para o display
177 -- ser atualizado de forma humanamente legível.
178 disp_counter: COMPONENT counter
179     PORT MAP (clk    => clk27M,
180               rstn   => rstn,
181               en     => '1',
182               max    => DISP_DIV_FACT-1,
183               q      => disp_count);
184
185 display_en <= '1' WHEN (disp_count = DISP_DIV_FACT-1)
186             ELSE '0';
187
188 -- Controlador do gerador de frutas
189 frutas: ENTITY WORK.ctrl_frutas PORT MAP (
190     clk    => clk27M, rstn => rstn and restartn and fruta_rstn,
191     enable => update_info, fruta => fruta_id
192 );
193
194 -- Contadores de varredura da tela
195 conta_coluna: COMPONENT counter
196     PORT MAP (clk    => clk27M,
197               rstn   => col_rstn,
198               en     => col_enable,
199               max    => SCR_WDT-1,
200               q      => col);
201
202 -- o contador de linha só incrementa quando o contador de colunas
203 -- chegou ao fim
204 line_inc <= '1' WHEN (line_enable='1' and col = SCR_WDT-1)
205 ELSE '0';
206
207 conta_linha: COMPONENT counter
208     PORT MAP (clk    => clk27M,
209               rstn   => line_rstn,
210               en     => line_inc,
211               max    => SCR_HGT-1,
212               q      => line);

```

```

213
214 -- podemos avançar para o próximo estado?
215 fim_varredura <= '1' WHEN (line = SCR_HGT-1) and (col = SCR_WDT-1)
216 ELSE '0';
217
218 -- Controlador dos fantasmas
219 -- Recebe um sinal de atualiza principal que é AND com uma das
220 -- três velocidades atua_en para mover o fantasma em cada estado
221 -- Gera sinais importantes de morte do pacman e dos fantasmas
222 ctrl_fans_inst: ENTITY work.ctrl_fans PORT MAP (
223     clk => clk27M,          rstn => rstn and restartn,
224     atualiza => fan_atua,      atua_en => atua_en (1 to 3),
225     keys_dir => fan_key_dir,   fan_died => fan_died,
226     fan_area => fan_area,      pacman_dead => pacman_dead,
227     spc_coin => got_spc_coin,   pac_fans_hit => pac_fans_hit,
228     fan_pos_x => fan_pos_x,     fan_pos_y => fan_pos_y,
229     fan_state => fan_state,     fan_cur_dir => fan_cur_dir
230 );
231
232 pac_move <= pac_atua and atua_en(0);
233 -- Controlador do pacman
234 -- Gera sinais quando as moedas são comidas
235 ctrl_pac_inst: ENTITY work.ctrl_pacman PORT MAP (
236     clk => clk27M,          rstn => rstn and restartn,
237     key_dir => pac_key_dir,   atualiza => pac_move,
238     pac_area => pac_area,     pac_cur_dir => pac_cur_dir,
239     pac_pos_x => pac_pos_x,    pac_pos_y => pac_pos_y,
240     got_coin => got_coin,     got_spc_coin => got_spc_coin
241 );
242
243 -- Preenche as matrizes 3x3 das vizinhanças pac_area
244 -- e fans_area durante PERCORRE_QUADRO
245 -- Essas matrizes informam aos controladores os blocos de cenário
246 -- no entorno dos personagens usados para definir o próximo movimento
247 -- type : sequential
248 p_fill_memarea: PROCESS (clk27M)
249     VARIABLE x_offset, y_offset: t_offset;
250     VARIABLE blk_out_sp: t_blk_sym;
251 BEGIN
252     IF (clk27M'event and clk27M='1') THEN
253         IF (varre_tela = '1') THEN
254             -- Parte "inútil" do jogo
255             -- Mais detalhes no final do código
256             IF (nwc = '1' and (not WALKABLE(block_out))) THEN
257                 blk_out_sp := BLK_PATH; -- always walking...
258             ELSE
259                 blk_out_sp := block_out;
260             END IF;
261
262             -- Leitura atrasada devido ao ciclo de clock da ram
263             y_offset := line - pac_pos_y;
264             x_offset := col - pac_pos_x;
265             IF (x_offset >= 0 and x_offset <= 2 and y_offset >= -1 and y_offset <= 1) THEN
266                 pac_area(y_offset, x_offset-1) <= blk_out_sp;
267             END IF;
268
269             FOR i in 0 to FAN_NO-1 LOOP
270                 y_offset := line - fan_pos_y(i);
271                 x_offset := col - fan_pos_x(i);
272                 IF (x_offset >= 0 and x_offset <= 2 and y_offset >= -1 and y_offset <= 1) THEN
273                     fan_area(i)(y_offset, x_offset-1) <= blk_out_sp;
274                 END IF;
275             END LOOP;
276         END IF;
277     END IF;
278 END PROCESS;
279
280 -- Atualiza parâmetros de informação atual do jogo
281 -- type: sequential
282 param_jogo: PROCESS (clk27M, rstn)
283 BEGIN
284     IF (rstn = '0') THEN
285         q_vidas <= 3;
286         q_pontos <= 0;
287         q_rem_moedas <= 240;
288     ELSIF (clk27M'event and clk27M = '1') THEN
289         IF (pacman_dead = '1' and update_info = '1') THEN
290             q_vidas <= q_vidas - 1;
291         END IF;
292
293         IF (fan_died = '1') THEN
294             q_pontos <= q_pontos + 200;
295         ELSIF (pac_move = '1') THEN
296             IF (got_fruta = '1') THEN
297                 q_pontos <= q_pontos + 500;
298             ELSIF (got_coin = '1') THEN
299                 q_pontos <= q_pontos + 10;
300                 q_rem_moedas <= q_rem_moedas - 1;
301             ELSIF (got_spc_coin = '1') THEN -- moeda especial não conta para o
302                 q_pontos <= q_pontos + 50; -- término do jogo!
303             ELSIF (nwc = '1') THEN
304                 q_pontos <= q_pontos + 1; -- modo "especial", evita um latch!

```

```

305         END IF;
306
307         IF (got_coin = '1' or got_spc_coin = '1') THEN
308             reg_coin_we <= '1'; -- registra uma moeda comida para ser apagada
309         ELSE
310             reg_coin_we <= '0';
311         END IF;
312     END IF;
313 END IF;
314 END PROCESS param_jogo;
315
316 -- purpose: Processo para que gera todos os sinais de desenho de overlay
317 -- (ie, sobre o fundo) da vidas, do pacman e dos fantasmas de acordo
318 -- com a varredura de line e col durante PERCORRE_QUADRO
319 -- type : combinational
320 des_overlay: PROCESS (pac_pos_x, pac_pos_y, pac_cur_dir, sig_blink, vidas_arr, fruta_id,
321                      fan_pos_x, fan_pos_y, fan_state, fan_cur_dir, line, col, frutas_com,
322                      q_fruta_com)
323     VARIABLE x_offset, y_offset: t_offset;
324     VARIABLE ovl_blk_tmp: t_ovl_blk_sym;
325 BEGIN
326     ovl_blk_tmp := BLK_NULL; -- este será o bloco que vai pra VGA
327     -- A hierarquia dos desenhos tem o último desta lista como mais
328     -- importante. Isto é, se ele for desenhado, nenhum outro irá
329     -- aparecer por cima
330
331     -- Desenho da fruta no jogo
332     y_offset := line - FRUTA_Y + 2;
333     x_offset := col - FRUTA_X + 2;
334     IF (x_offset >= 0 and x_offset < 5 and
335         y_offset >= 0 and y_offset < 5) THEN -- região de desenho:
336         ovl_blk_tmp := FRUTA_BLKMAP(fruta_id)(y_offset, x_offset);
337     END IF;
338
339     FOR i in 0 to FAN_NO-1 LOOP -- Desenho dos fantasmas
340         y_offset := line - fan_pos_y(i) + 2;
341         x_offset := col - fan_pos_x(i) + 2;
342         IF (x_offset >= 0 and x_offset < 5 and
343             y_offset >= 0 and y_offset < 5) THEN
344             IF (fan_state(i) = ST_VULN_BLINK) THEN
345                 IF (sig_blink(5) = '0') THEN -- pisca no final do modo vulnerável
346                     ovl_blk_tmp := FAN_VULN_BLKMAP(y_offset, x_offset);
347                 ELSE
348                     ovl_blk_tmp := BLK_NULL;
349                 END IF;
350             ELSIF (fan_state(i) = ST_VULN) THEN
351                 ovl_blk_tmp := FAN_VULN_BLKMAP(y_offset, x_offset);
352             ELSIF (fan_state(i) = ST_DEAD) THEN
353                 ovl_blk_tmp := FAN_DEAD_BLKMAP(fan_cur_dir(i))(y_offset, x_offset);
354             ELSE
355                 ovl_blk_tmp := FAN_BLKMAPS(i)(fan_cur_dir(i))(y_offset, x_offset);
356             END IF;
357         END IF;
358     END LOOP;
359
360     -- Desenho do pacman
361     y_offset := line - pac_pos_y + 2;
362     x_offset := col - pac_pos_x + 2;
363     IF (x_offset >= 0 and x_offset < 5 and
364         y_offset >= 0 and y_offset < 5) THEN
365         IF (sig_blink(5) = '0') THEN -- pacman com boca aberta
366             ovl_blk_tmp := PAC_BLKMAPS(pac_cur_dir)(y_offset, x_offset);
367         ELSE
368             IF (pac_cur_dir = DIREI or pac_cur_dir = ESQUE) THEN
369                 ovl_blk_tmp := PAC_FECH_BLKMAP(y_offset, x_offset);
370             ELSE
371                 ovl_blk_tmp := PAC_FECV_BLKMAP(y_offset, x_offset);
372             END IF;
373         END IF;
374     END IF;
375
376     -- Desenhos do HUD (Head-Up Display) à direita do mapa:
377
378     FOR i in 0 to 2 LOOP -- Desenho dos ícones de vida
379         IF (vidas_arr(i) = '1') THEN
380             y_offset := line - VIDA_ICONS_Y(i) + 2;
381             x_offset := col - VIDA_ICONS_X(i) + 2;
382             IF (x_offset >= 0 and x_offset < 5 and
383                 y_offset >= 0 and y_offset < 5) THEN
384                 ovl_blk_tmp := PAC_BLKMAPS(DIREI)(y_offset, x_offset);
385             END IF;
386         END IF;
387     END LOOP;
388
389     -- Desenho da lista de frutas comidas
390     FOR i in 0 to MAX_FRUTA_COM-1 LOOP
391         IF (i < q_fruta_com) THEN
392             y_offset := line - (FRUTA_ICONS_Y0 - i*6) + 2;
393             x_offset := col - FRUTA_ICONS_X + 2;
394             IF (x_offset >= 0 and x_offset < 5 and
395                 y_offset >= 0 and y_offset < 5) THEN
396                 ovl_blk_tmp := FRUTA_BLKMAP(frutas_com(i))(y_offset, x_offset);

```

```

397         END IF;
398     END IF;
399 END LOOP;
400
401     ovl_blk_in <= ovl_blk_tmp;
402 END PROCESS;
403
404 -- Detecta quando uma fruta foi comida, atualizando a lista
405 -- no HUD e gerando o reset do controlador de frutas
406 -- type: sequential
407 PROCESS (clk27M, rstn)
408 BEGIN
409     IF (rstn = '0') THEN
410         q_fruta_com <= 0;
411     ELSIF (clk27M'event and clk27M = '1') THEN
412         IF (got_fruta = '1' and pac_move = '1') THEN
413             frutas_com(q_fruta_com) <= fruta_id;
414             q_fruta_com <= q_fruta_com + 1;
415             fruta_rstn <= '0';
416         ELSE
417             fruta_rstn <= '1';
418         END IF;
419     END IF;
420 END PROCESS;
421
422 -- Determina quando o pacman comeu uma fruta
423 got_fruta <= '1' WHEN (pac_pos_x = FRUTA_X and pac_pos_y = FRUTA_Y and fruta_id /= 0)
424 ELSE '0';
425
426 -- Determina quando o pacman colidiu com cada um dos fantasmas
427 -- type: combinational
428 PROCESS (pac_pos_x, pac_pos_y, fan_pos_x, fan_pos_y)
429 VARIABLE off_x, off_y: t_offset;
430 BEGIN
431     FOR i in 0 to FAN_NO-1 LOOP
432         off_x := pac_pos_x - fan_pos_x(i);
433         off_y := pac_pos_y - fan_pos_y(i);
434         -- a tolerância para colisão é uma região 5x5. Isto é,
435         -- os centros dos objetos podem estar distantes entre
436         -- si em, no máximo, 2 blocos
437         IF (off_x >= -2 and off_x <= 2 and off_y >= -2 and off_y <= 2) THEN
438             pac_fans_hit(i) <= '1';
439         ELSE
440             pac_fans_hit(i) <= '0';
441         END IF;
442     END LOOP;
443 END PROCESS;
444
445 -- Define dado que entra na ram de cenário
446 -- type: combinational
447 def_block_in: PROCESS (le_cenario, cen_addr)
448 BEGIN
449     IF (le_cenario = '1') THEN
450         block_in <= CONV_TAB.BLK(MAPA_INICIAL(cen_addr));
451     ELSE
452         block_in <= BLK_PATH; -- Caso que a moeda é comida pelo pacman
453     END IF;
454 END PROCESS;
455
456 -- Converte representação inteira para unária a fim
457 -- de mostrar a informação de vidas na tela
458 -- type: combinational
459 led_vidas: PROCESS (q_vidas)
460 BEGIN
461     IF (q_vidas = 3) THEN
462         vidas_arr <= "111";
463     ELSIF (q_vidas = 2) THEN
464         vidas_arr <= "011";
465     ELSIF (q_vidas = 1) THEN
466         vidas_arr <= "001";
467     ELSE
468         vidas_arr <= "000";
469     END IF;
470 END PROCESS led_vidas;
471
472
473 -- Processos que definem a FSM principal. Alguns sinais de controle são definidos
474 -- apenas para um estado e portanto estão localizados no process seguinte
475 -- type: combinational
476 logica_mealy: PROCESS (estado, fim_varredura, timer, long_timer, q_rem_moedas, q_vidas,
477                        col, line, pac_pos_x, pac_pos_y, pacman_dead, reg_coin_we, fan_died)
478 BEGIN
479     case estado is
480     when CARREGA_MAPA => IF (fim_varredura = '1') THEN -- Estado CARREGA_MAPA:
481         pr_estado <= ESTADO_INICIAL; -- Percorre linhas e colunas escrevendo o
482     ELSE pr_estado <= CARREGA_MAPA; -- conteúdo de MAPA_INICIAL na memória.
483     END IF; -- Usado para (re)inicializar o jogo inteiro
484     line_rstn <= '1';
485     line_enable <= '1';
486     col_rstn <= '1';
487     col_enable <= '1';
488

```

```

489         we                <= '1';
490         timer_rstn        <= '0';
491         timer_enable      <= '0';
492         cen_addr          <= col + SCR.WDT*line;
493
494     when REINICIO => IF (long_timer = '1') THEN -- Estado REINICIO:
495         pr_estado <= ESTADO_INICIAL; -- Aguarda um intervalo de alguns segundos
496         antes
497         ELSE -- de continuar o jogo. Além disso, ativa o
498         sinal
499         pr_estado <= REINICIO; -- restartn para reinicializar os dados
500         END IF; -- necessários. O pacman e os fantasmas voltam
501         as
502         line_rstn        <= '1'; -- suas posições originais mas as moedas do
503         mapa
504         line_enable      <= '1'; -- e a pontuação permanecem.
505         col_rstn        <= '1';
506         col_enable      <= '1';
507         we                <= '0';
508         timer_rstn      <= '1';
509         timer_enable    <= '1';
510         cen_addr        <= 0;
511
512     when FIM_JOGO => pr_estado <= FIM_JOGO; -- Estado FIM_JOGO:
513         line_rstn        <= '1'; -- Não realiza ação e fica ocioso nesse estado.
514         line_enable      <= '1'; -- Alguma mensagem é mostrada no display.
515         col_rstn        <= '1'; -- O jogo acabou (as vidas do pacman se
516         esgotaram
517         col_enable      <= '1'; -- ou este comeu todas as moedas do mapa) e o
518         we                <= '0'; -- circuito deve ser resetado.
519         timer_rstn      <= '0';
520         timer_enable    <= '0';
521         cen_addr        <= 0;
522
523     when ESTADO_INICIAL => IF (timer = '1') THEN -- Estado ESTADO_INICIAL:
524         pr_estado <= PERCORREQUADRO; -- Primeiro estado durante operação normal do
525         jogo
526         ELSE -- Responsável pelo atraso principal da
527         animação,
528         pr_estado <= ESTADO_INICIAL; -- após o qual será feita a varredura no estado
529         do
530         END IF; -- cenário.
531         line_rstn        <= '0';
532         line_enable      <= '0';
533         col_rstn        <= '0';
534         col_enable      <= '0';
535         we                <= '0';
536         timer_rstn      <= '1';
537         timer_enable    <= '1';
538         cen_addr        <= 0;
539
540     when PERCORREQUADRO => IF (fim_varredura = '1') THEN -- Estado PERCORREQUADRO:
541         pr_estado <= ATUALIZA_LOGICA_1; -- Varre a memória de cenário, lendo as
542         regiões vizinhas
543         ELSE -- dos objetos e, ao mesmo tempo, escrevendo
544         as posições
545         pr_estado <= PERCORREQUADRO; -- atuais dos mesmos na memória de overlay.
546         END IF;
547         line_rstn        <= '1';
548         line_enable      <= '1';
549         col_rstn        <= '1';
550         col_enable      <= '1';
551         we                <= '0';
552         timer_rstn      <= '0';
553         timer_enable    <= '0';
554         cen_addr        <= col + SCR.WDT*line;
555
556     when ATUALIZA_LOGICA_1 => IF (pacman_dead = '1') THEN -- Estado ATUALIZA_LOGICA_1:
557         IF (q_vidas = 0) THEN -- Faz as checagens fundamentais de final de
558         jogo ou
559         pr_estado <= FIM_JOGO; -- reinício, além de habilitar o próximo
560         movimento
561         ELSE -- do pacman.
562         pr_estado <= REINICIO;
563         END IF;
564         ELSIF (q_rem_moedas <= 0) THEN
565         pr_estado <= FIM_JOGO;
566         ELSE
567         pr_estado <= ATUALIZA_LOGICA_2;
568         END IF;
569         line_rstn        <= '1';
570         line_enable      <= '0';
571         col_rstn        <= '1';
572         col_enable      <= '0';
573         we                <= '0';
574         timer_rstn      <= '0';
575         timer_enable    <= '0';
576         cen_addr        <= 0;
577
578     when ATUALIZA_LOGICA_2 => pr_estado <= MEMORIA_LWR; -- Estado ATUALIZA_LOGICA_2:
579         line_rstn        <= '1'; -- Habilita o próximo movimento dos fantasmas.
580         line_enable      <= '0';

```

```

569         col_rstn      <= '1';
570         col_enable    <= '0';
571         we             <= '0';
572         timer_rstn    <= '0';
573         timer_enable   <= '0';
574         cen_addr      <= 0;
575
576     when MEMORIA_WR => pr_estado <= ESTADO.INICIAL; -- Estado MEMORIA_WR:
577         line_rstn      <= '0'; -- Escreve apenas um bloco que é o novo valor da
578             célula
579         line_enable     <= '0'; -- atual do pacman na memória de cenário. Isso
580             permite
581         col_rstn        <= '0'; -- apagar a moeda que havia sob o pacman se ela
582             foi
583         col_enable      <= '0'; -- comida. É o último estado do ciclo normal,
584             nele
585         we              <= reg_coin.we; -- as informações globais são atualizadas.
586         timer_rstn      <= '0';
587         timer_enable     <= '0';
588         cen_addr        <= pac.pos-x + SCR.WDT * pac.pos-y;
589
590     when others => pr_estado <= CARREGA_MAPA;
591         line_rstn      <= '0';
592         line_enable     <= '0';
593         col_rstn        <= '0';
594         col_enable      <= '0';
595         we              <= '0';
596         timer_rstn      <= '1';
597         timer_enable     <= '0';
598         cen_addr        <= 0;
599     END case;
600 END PROCESS logica_mealy;
601
602 -- Define sinais de controle da FSM usados em apenas UM ESTADO
603 -- type: combinational
604 sinais_extras: PROCESS (estado, atua_en)
605 BEGIN
606     IF (estado = PERCORREQUADRO)
607     THEN varre_tela <= '1';
608     ELSE varre_tela <= '0';
609     END IF;
610
611     IF (estado = CARREGA_MAPA)
612     THEN le_cenario <= '1';
613     ELSE le_cenario <= '0';
614     END IF;
615
616     IF (estado = REINICIO)
617     THEN restartn <= '0';
618     ELSE restartn <= '1';
619     END IF;
620
621     IF (estado = ATUALIZA_LOGICA_2)
622     THEN fan_atua <= '1';
623     ELSE fan_atua <= '0';
624     END IF;
625
626     IF (estado = ATUALIZA_LOGICA_1)
627     THEN pac_atua <= '1';
628     ELSE pac_atua <= '0';
629     END IF;
630
631     IF (estado = FIMJOGO)
632     THEN endgame <= '1';
633     ELSE endgame <= '0';
634     END IF;
635
636     IF (estado = MEMORIA_WR)
637     THEN update_info <= '1';
638     ELSE update_info <= '0';
639     END IF;
640 END PROCESS;
641
642 -- Avança a FSM para o próximo estado
643 -- type: sequential
644 seq_fsm: PROCESS (clk27M, rstn)
645 BEGIN
646     IF (rstn = '0') THEN
647         estado <= POWER_UP;
648     elsif (clk27M'event and clk27M = '1') THEN
649         estado <= pr_estado;
650     END IF;
651 END PROCESS seq_fsm;
652
653 -- Atualiza contadores de número de atualizações
654 -- Gera enables de atualizações para cada velocidade de atualização
655 -- type: sequential
656 atual_counters: PROCESS (clk27M, rstn)
657 VARIABLE atual_cont: t_vet_velocs;
658 BEGIN
659     IF (rstn = '0') THEN

```

```

656         atual_cont := (OTHERS => 0);
657         sig_blink <= (OTHERS => '0');
658     ELSIF (clk27M'event and clk27M = '1') THEN
659         IF (estado = ATUALIZA.LOGICA.2) THEN
660             FOR i IN 0 to VEL.NO-1 LOOP
661                 IF (atual_cont(i) = VEL.DIV(i)-1) THEN
662                     atual_cont(i) := 0;
663                     atua_en(i) <= '1';
664                 ELSE
665                     atual_cont(i) := atual_cont(i) + 1;
666                     atua_en(i) <= '0';
667                 END IF;
668             END LOOP;
669             sig_blink <= sig_blink + 1; -- EN usado para piscagem
670         END IF;
671     END IF;
672 END PROCESS;
673
674 -- Easter egg! :)
675 -- Esse PROCESS está fazendo qualquer coisa
676 PROCESS (clk27M, rstn)
677     VARIABLE hit_count: INTEGER := 0;
678 BEGIN
679     IF (rstn = '0') THEN
680         hit_count := 0;
681         nwc <= '0';
682     ELSIF (clk27M'event and clk27M = '1') THEN
683         IF ((pac.pos_x = MAP.X.MIN or pac.pos_x = MAP.X.MAX) and pac.move = '1' and
684             not (pac.pos_y = MAP.Y.MIN and pac.pos_y = MAP.Y.MAX)) THEN
685             hit_count := hit_count + 1;
686         ELSIF (atua_en(4) = '1' and pac.atua = '1' and hit_count > 0) THEN
687             hit_count := hit_count - 1;
688         END IF;
689
690         IF (hit_count = 5) THEN
691             nwc <= '1'; -- WARNING: no wall collisions!!
692         END IF;
693     END IF;
694 END PROCESS;
695
696 -- Contadores utilizados para atrasar a animação (evitar
697 -- que a atualização de quadros fique muito veloz).
698 p_contador0: COMPONENT counter
699     PORT MAP (clk      => clk27M,
700              rstn      => timer_rstn,
701              en        => timer_enable,
702              max        => DIV.FACT - 1,
703              q          => contador);
704
705 p_contador1: COMPONENT counter
706     PORT MAP (clk      => clk27M,
707              rstn      => timer_rstn, --mesmo reset do contador 0, porém
708              en        => timer, --contagem a cada término do contador 0
709              max        => 127,
710              q          => long_cont);
711
712 -- O sinal "timer" indica a hora de fazer nova atualização
713 timer <= '1' WHEN (contador = DIV.FACT - 1)
714 ELSE '0';
715
716 -- Timer para mostrar um evento na tela (poucos segundos)
717 long_timer <= '1' WHEN (long_cont = 127)
718 ELSE '0';
719
720 -- Processos que sincronizam o reset assíncrono, de preferência com mais
721 -- de 1 flipflop, para evitar metaestabilidade.
722 -- type : sequential
723 build_rstn: PROCESS (clk27M)
724     VARIABLE temp : STD.LOGIC; -- flipflop intermediario
725 BEGIN
726     IF (clk27M'event and clk27M = '1') THEN
727         rstn <= temp;
728         temp := reset.button;
729     END IF;
730 END PROCESS build_rstn;
731 END comportamento;

```

5.2 Componentes

Listing 2: Definições do Jogo

```

1 LIBRARY ieee;
2 USE ieee.STD.LOGIC.1164. all;
3 USE ieee.NUMERIC.STD. all;
4
5 PACKAGE pac_defs IS
6     -----

```

```

7  -- Definições de dados, constantes e tipos para o jogo
8
9
10 -- Constantes Básicas
11 CONSTANT SCR_HGT : INTEGER := 96; -- Resolução de blocos usada (hgt linhas por wdt colunas)
12 CONSTANT SCR_WDT : INTEGER := 128;
13 CONSTANT TABLEN: INTEGER := 93; -- Maior coordenada do tabuleiro
14 CONSTANT FAN_NO: INTEGER := 2; -- Número de fantasmas no jogo. O código foi projetado para
15 -- esse número possa ser facilmente alterado. Provavelmente,
16 -- será necessário apenas definir um mapa de sprites em
17 -- "des_overlay" para os novos fantasmas e criar novas teclas
18 -- de controle no "player_dir.vhd"
19 CONSTANT FRUTA_NO: INTEGER := 2; -- Tipos de frutas no jogo
20
21 -- Tipos básicos
22 SUBTYPE t_color_3b is std_logic_vector(2 downto 0);
23 SUBTYPE t_pos is INTEGER range 0 to TABLEN;
24 SUBTYPE t_offset is INTEGER range -TABLEN to TABLEN;
25 SUBTYPE t_fan_time is INTEGER range 0 to 1000;
26 TYPE t_direcao is (CIMA, DIREI, BAIXO, ESQUE, NADA);
27
28 -- A legenda pros elementos no cenário é dada por t_tab_sym
29 -- ' ': vazio, '.': caminho, 6 tipos de parede de acordo com a orientação,
30 -- C: moeda, P: moeda especial, D: porta
31 TYPE t_tab_sym is (' ', '.', '|', '-', 'Q', 'W', 'E', 'R', 'C', 'P', 'D');
32
33 -- Legenda de direção usada para o piloto automático dos fantasmas
34 -- C: cima, E: esquerda, B: baixo, D: direita
35 TYPE t_dir_sym is (' ', 'C', 'E', 'B', 'D');
36
37 -- Os blocos de cenário (4 bits) e overlay (9 bits)
38 SUBTYPE t_blk_id is STD_LOGIC_VECTOR(3 downto 0);
39 SUBTYPE t_ovl_blk_id is STD_LOGIC_VECTOR(8 downto 0);
40
41 -- Definição dos blocos (cenário e overlay)
42 TYPE t_blk_sym is (BLK_NULL, BLK_PATH, BLK_WALL_V, BLK_WALL_H, BLK_WALL_Q, BLK_WALL_W, BLK_WALL_E,
43 BLK_WALL_R, BLK_COIN, BLK_SPC_COIN, BLK_DOOR);
44
45 TYPE t_ovl_blk_sym is (...) --- TRECHO DE CÓDIGO SUPRIMIDO
46
47 -- "Funções" para blocos
48 TYPE t_blk_bool is array(t_blk_sym) of boolean;
49 CONSTANT WALKABLE: t_blk_bool := -- define quais blocos são percorráveis
50 (BLK_PATH => true, BLK_COIN => true, BLK_SPC_COIN => true, OTHERS => false);
51 TYPE c_tab_blk is array(t_tab_sym) of t_blk_sym;
52 CONSTANT CONV_TAB_BLK: c_tab_blk :=
53 (' ' => BLK_NULL, '.' => BLK_PATH, '|' => BLK_WALL_V, '-' => BLK_WALL_H, 'Q' => BLK_WALL_Q, 'W' =>
54 BLK_WALL_W,
55 'E' => BLK_WALL_E, 'R' => BLK_WALL_R, 'C' => BLK_COIN, 'P' => BLK_SPC_COIN, 'D' => BLK_DOOR);
56
57 -- Tabelas de blocos
58 TYPE t_tab is array(0 to SCR_HGT-1, 0 to SCR_WDT-1) of t_tab_sym;
59 TYPE t_blk_sym_3x3 is array(-1 to 1, -1 to 1) of t_blk_sym;
60
61 -- Tipos indexados de armazenamento gráfico em blocos e pixels
62 TYPE t_sprite5 is array(0 to 4, 0 to 4) of STD_LOGIC;
63 TYPE t_ovl_blk_5x5 is array(0 to 4, 0 to 4) of t_ovl_blk_sym;
64 TYPE t_ovl_blk_dir_vet is array(t_direcao) of t_ovl_blk_5x5;
65 TYPE t_fans_ovl_blk_dir_vet is array(0 to FAN_NO-1) of t_ovl_blk_dir_vet;
66 TYPE t_fruit_ovl_blk_vet is array(0 to FRUTA_NO) of t_ovl_blk_5x5;
67 TYPE t_sprite5_vet is array(t_sprite5) of t_sprite5;
68 TYPE t_ovl_sprite5_vet is array(t_ovl_sprite5) of t_sprite5;
69
70 TYPE t_fan_state is (ST_VIVO, ST_PRE_VULN, ST_VULN, ST_VULN_BLINK,
71 ST_DEAD, ST_PRE_DEAD, ST_FIND_EXIT, ST_FUGA);
72
73 -- Tipos em array para os fantasmas
74 TYPE t_fans_pos is array(0 to FAN_NO-1) of t_pos;
75 TYPE t_fans_dirs is array(0 to FAN_NO-1) of t_direcao;
76 TYPE t_fans_blk_sym is array(0 to FAN_NO-1) of t_blk_sym;
77 TYPE t_fans_blk_sym_3x3 is array(0 to FAN_NO-1) of t_blk_sym_3x3;
78 TYPE t_fans_states is array(0 to FAN_NO-1) of t_fan_state;
79 TYPE t_fans_times is array(0 to FAN_NO-1) of t_fan_time;
80 SUBTYPE t_fans_bits is STD_LOGIC_VECTOR(0 to FAN_NO-1);
81
82 TYPE t_vidas_pos is array(0 to 2) of t_pos;
83 SUBTYPE t_fruta_id is INTEGER range 0 to FRUTA_NO; -- 0 significa sem fruta
84
85 -- Constantes do jogo
86 CONSTANT CELL_IN_X : INTEGER := 42; -- posição da célula principal dentro da cela
87 CONSTANT CELL_IN_Y : INTEGER := 44;
88 CONSTANT CELL_OUT_Y : INTEGER := 35; -- posição Y da célula principal fora da cela
89 CONSTANT MAP_X_MAX : INTEGER := 82; -- coordenadas limites do mapa
90 CONSTANT MAP_X_MIN : INTEGER := 2;
91 CONSTANT MAP_Y_MAX : INTEGER := 93;
92 CONSTANT MAP_Y_MIN : INTEGER := 1;
93 CONSTANT VIDA_ICONS_X: t_vidas_pos := (90, 90, 90);
94 CONSTANT VIDA_ICONS_Y: t_vidas_pos := (89, 83, 77);
95 CONSTANT FRUTA_X: t_pos := 42;
96 CONSTANT FRUTA_Y: t_pos := 53;
97 CONSTANT FRUTA_ICONS_X: INTEGER := 90;

```



```

97 CONSTANT FRUTA_CONS_Y0: INTEGER := 65;
98 CONSTANT MAX_FRUTA_COM: INTEGER := 5;
99
100 CONSTANT VEL_NO: INTEGER := 5;
101 SUBTYPE t_velocs is INTEGER range 0 to 100;
102 TYPE t_vet_velocs is array(0 to VEL_NO-1) of t_velocs;
103 --divisores de atualização para: 0=pacman, 1= fant vuln, 2=fant, 3=fant morto
104 CONSTANT VEL_DIV: t_vet_velocs := (6, 8, 5, 4, 60);
105
106 --Fatores de divisão do clock de 27MHz, usados para atualização do
107 --estado do jogo ("velocidade de execução") e do display
108 CONSTANT DIV_FACT: INTEGER := 202500;
109 CONSTANT DISP_DIV_FACT: INTEGER := 27000000/4;
110
111 -- lista de frutas comidas recentemente
112 TYPE t_fruta_vet is array(0 to MAX_FRUTA_COM - 1) of t_fruta_id;
113
114 subtype sentido is INTEGER range -1 to 1;
115 TYPE t_direc is array(0 to 1) of sentido;
116 TYPE t_direc_vet is array(t_direcao) of t_direc;
117
118 CONSTANT DIRS: t_direc_vet := (CIMA => (-1, 0), DIREI => ( 0, 1),
119                               BAIXO => ( 1, 0), ESQUE => ( 0,-1),
120                               NADA => ( 0, 0));
121
122 TYPE t_tab_array is array(0 to SCR_WDT*SCR_HGT-1) of t_tab_sym;
123 TYPE t_dir_mapa is array(0 to SCR_HGT-1, 0 to SCR_WDT-1) of t_dir_sym;
124
125 -- Mapa de inicialização da RAM inferior (cenário), a legenda está mais acima.
126 --
127 -- Esse mapa pode ser customizado livremente, sendo necessário apenas definir
128 -- algumas constantes de posições especiais (por exemplo, as da "jaula" e das
129 -- posições iniciais) e, opcionalmente, criar um novo mapa de piloto
130 -- automático para fantasmas (veja abaixo).
131 CONSTANT MAPA_INICIAL: t_tab_array; --- TRECHO DE CÓDIGO SUPRIMIDO
132
133 -- Mapa de piloto automático para os fantasmas. Legenda do tipo t_dir_sym.
134 --
135 -- Neste mapa, estão armazenados apenas a próxima direção do percurso de um fantasma
136 -- quando este está morto. Não é preciso definir uma direção para todas as casas:
137 -- nas indefinidas, o fantasma será teletransportado diretamente para a jaula,
138 -- mas o seu tempo de inatividade não deverá sofrer alteração.
139 CONSTANT FAN_PERCURSO: t_dir_mapa; --- TRECHO DE CÓDIGO SUPRIMIDO
140
141 END pac_defs;

```

Listing 3: Controlador do pacman

```

1 LIBRARY ieee;
2 USE ieee.STD_LOGIC_1164.all;
3 USE ieee.NUMERIC_STD.all;
4 USE work.PAC_DEFS.all;
5
6 ENTITY ctrl_pacman IS
7 PORT (
8   clk, rstn      :IN STD_LOGIC;
9   atualiza        :IN STD_LOGIC;
10  key_dir         :IN t_direcao; --tecla de ação lida pelo teclado
11  pac_area        :IN t_blk_sym_3x3; --mapa 3x3 em torno da posição atual
12  pac_pos_x, pac_pos_y :BUFFER t_pos;
13  pac_cur_dir     :BUFFER t_direcao;
14  got_coin, got_spec_coin :OUT STD_LOGIC
15 );
16 END ctrl_pacman;
17
18 ARCHITECTURE behav OF ctrl_pacman IS
19   SIGNAL pac_nxt_cel, pac_dir_cel, pac_esq_cel, pac_cim_cel, pac_bai_cel: t_blk_sym;
20
21   CONSTANT PAC_START_X : INTEGER := 42;
22   CONSTANT PAC_START_Y : INTEGER := 71;
23 BEGIN
24   -- Calcula possíveis parâmetros envolvidos no próximo movimento
25   -- do pacman
26   -- type: combinational
27   PROCESS (pac_cur_dir, pac_area)
28   BEGIN
29     --calcula qual seriam as proximas celulas visitadas pelo pacman
30     pac_nxt_cel <= pac_area(DIRS(pac_cur_dir)(0), DIRS(pac_cur_dir)(1));
31
32     IF (WALKABLE(pac_area(DIRS(pac_cur_dir)(0), DIRS(pac_cur_dir)(1)))) THEN
33       -- aqui o pacman conseguirá andar para a próxima casa
34       -- então as células candidatas para fazer curva estão nas diagonais
35       pac_dir_cel <= pac_area(DIRS(pac_cur_dir)(0), DIRS(pac_cur_dir)(1) + 1);
36       pac_esq_cel <= pac_area(DIRS(pac_cur_dir)(0), DIRS(pac_cur_dir)(1) - 1);
37       pac_cim_cel <= pac_area(DIRS(pac_cur_dir)(0)-1, DIRS(pac_cur_dir)(1));
38       pac_bai_cel <= pac_area(DIRS(pac_cur_dir)(0)+1, DIRS(pac_cur_dir)(1));
39     ELSE
40       -- caso o pacman esteja travado (parado) em alguma parede

```

```

41     pac_dir_cel <= pac_area( 0, 1);
42     pac_esq_cel <= pac_area( 0, -1);
43     pac_cim_cel <= pac_area(-1, 0);
44     pac_bai_cel <= pac_area( 1, 0);
45 END IF;
46 END PROCESS;
47
48 -- purpose: Este processo irá atualizar a posição do pacman e definir
49 --           suas ações no jogo.
50 -- type      : sequential
51 p_atualiza_pacman: PROCESS (clk, rstn)
52 VARIABLE nxt_move, key_dir_old: t_direcao;
53 BEGIN
54     IF (rstn = '0') THEN
55         pac_pos_x <= PAC_START_X;
56         pac_pos_y <= PAC_START_Y;
57         pac_cur_dir <= NADA;
58         nxt_move := NADA;
59     ELSIF (clk'event and clk = '1') THEN
60         IF (atualiza = '1') THEN
61             -- Checa teclado para "agendar" um movimento
62             IF (key_dir /= NADA and key_dir_old = NADA) THEN
63                 nxt_move := key_dir;
64             END IF;
65
66             -- atualiza direção
67             IF (nxt_move = CIMA and WALKABLE(pac_cim_cel)) THEN
68                 pac_cur_dir <= CIMA;
69                 nxt_move := NADA;
70             ELSIF (nxt_move = DIREI and WALKABLE(pac_dir_cel)) THEN
71                 pac_cur_dir <= DIREI;
72                 nxt_move := NADA;
73             ELSIF (nxt_move = BAIXO and WALKABLE(pac_bai_cel)) THEN
74                 pac_cur_dir <= BAIXO;
75                 nxt_move := NADA;
76             ELSIF (nxt_move = ESQUE and WALKABLE(pac_esq_cel)) THEN
77                 pac_cur_dir <= ESQUE;
78                 nxt_move := NADA;
79             END IF;
80
81             IF (WALKABLE(pac_nxt_cel)) THEN -- atualiza posição
82                 IF (pac_pos_x = MAP_X_MAX) then -- teletransporte
83                     pac_pos_x <= MAP_X_MIN + 1;
84                 ELSIF (pac_pos_x = MAP_X_MIN) then
85                     pac_pos_x <= MAP_X_MAX - 1;
86                 ELSIF (pac_pos_y = MAP_Y_MIN) then
87                     pac_pos_y <= MAP_Y_MAX - 1;
88                 ELSIF (pac_pos_y = MAP_Y_MAX) then
89                     pac_pos_y <= MAP_Y_MIN + 1;
90                 ELSE
91                     pac_pos_x <= pac_pos_x + DIRS(pac_cur_dir)(1);
92                     pac_pos_y <= pac_pos_y + DIRS(pac_cur_dir)(0);
93                 END IF;
94             END IF;
95             key_dir_old := key_dir;
96         END IF;
97     END IF;
98 END PROCESS;
99
100 -- Sinais de controle externos ficam ativos durante um ciclo de "atualiza"
101 got_coin <= '1' WHEN (pac_nxt_cel = BLK_COIN and atualiza = '1')
102 ELSE '0';
103
104 got_spc_coin <= '1' WHEN (pac_nxt_cel = BLK_SPC_COIN and atualiza = '1')
105 ELSE '0';
106 END behav;

```

Listing 4: Controlador dos fantasmas

```

1 LIBRARY ieee;
2 USE ieee.STD_LOGIC_1164.all;
3 USE ieee.NUMERIC_STD.all;
4 USE work.PAC_DEFS.all;
5
6 ENTITY ctrl_fans IS
7     PORT (
8         clk, rstn      :IN STD_LOGIC;
9         atualiza        :IN STD_LOGIC;
10        -- diferentes velocidades para atualizar (lenta -> rápida)
11        atua_en         :IN STD_LOGIC_VECTOR(0 to 2);
12        keys_dir        :IN t_fans_dirs; -- teclas de ação do teclado
13        -- mapas 3x3 em torno da posições atuais
14        fan_area        :IN t_fans_blk_sym_3x3;
15        spc_coin        :IN STD_LOGIC;
16        pac_fans_hit     :IN UNSIGNED(0 to FAN_NO-1);
17        fan_pos_x, fan_pos_y :BUFFER t_fans_pos;
18        fan_state       :BUFFER t_fans_states;
19        fan_cur_dir     :BUFFER t_fans_dirs;

```

```

20     pacman_dead, fan_died :OUT STD_LOGIC
21 );
22 END ctrl_fans;
23
24 ARCHITECTURE behav OF ctrl_fans IS
25     SIGNAL fan_nxt_cel, fan_dir_cel, fan_esq_cel, fan_cim_cel, fan_bai_cel: t_fans_blk_sym;
26     SIGNAL fan_tempo: t_fans_times;
27     SIGNAL fan_rstn_tempo: t_fans_bits;
28     SIGNAL pr_fan_state: t_fans_states;
29
30     -- Tempos (em n. de atualizações) para os estados temporários dos fantasmas
31     CONSTANT FAN_TIME_VULN_START_BLINK : INTEGER := 650;
32     CONSTANT FAN_TIME_VULN_END : INTEGER := 850;
33     CONSTANT FAN_TIME_DEAD : INTEGER := 600;
34
35     CONSTANT FANS_START_X : t_fans_pos := (40, 48);
36     CONSTANT FANS_START_Y : t_fans_pos := (44, 44);
37 BEGIN
38     -- Calcula possíveis parâmetros envolvidos no próximo movimento
39     -- de todos os fantasmas. Análogo ao do pacman (veja ctrl_pacman).
40     -- type: combinational
41     PROCESS (fan_area, fan_cur_dir)
42     BEGIN
43         FOR i in 0 to FAN_NO-1 LOOP
44             fan_nxt_cel(i) <= fan_area(i)(DIRS(fan_cur_dir(i))(0), DIRS(fan_cur_dir(i))(1));
45
46             IF (WALKABLE(fan_area(i)(DIRS(fan_cur_dir(i))(0), DIRS(fan_cur_dir(i))(1)))) THEN
47                 fan_dir_cel(i) <= fan_area(i)(DIRS(fan_cur_dir(i))(0), DIRS(fan_cur_dir(i))(1) + 1);
48                 fan_esq_cel(i) <= fan_area(i)(DIRS(fan_cur_dir(i))(0), DIRS(fan_cur_dir(i))(1) - 1);
49                 fan_cim_cel(i) <= fan_area(i)(DIRS(fan_cur_dir(i))(0) - 1, DIRS(fan_cur_dir(i))(1));
50                 fan_bai_cel(i) <= fan_area(i)(DIRS(fan_cur_dir(i))(0) + 1, DIRS(fan_cur_dir(i))(1));
51             ELSE
52                 fan_dir_cel(i) <= fan_area(i)(0,1);
53                 fan_esq_cel(i) <= fan_area(i)(0,-1);
54                 fan_cim_cel(i) <= fan_area(i)(-1,0);
55                 fan_bai_cel(i) <= fan_area(i)(1,0);
56             END IF;
57         END LOOP;
58     END PROCESS;
59
60     -- Atualiza as posições dos fantasmas e define
61     -- suas ações no jogo de acordo com seus estados.
62     -- type: sequential
63     p_atualiza_fan: PROCESS (clk, rstn)
64     VARIABLE keys_dir_old, nxt_move: t_fans_dirs;
65     BEGIN
66         IF (rstn = '0') THEN
67             fan_pos_x <= FANS_START_X;
68             fan_pos_y <= FANS_START_Y;
69             fan_cur_dir <= (others => NADA);
70             nxt_move := (others => NADA);
71             ELSIF (clk'event and clk = '1') THEN
72                 IF (atualiza = '1') THEN
73                     FOR i in 0 to FAN_NO-1 LOOP
74                         CASE fan_state(i) IS
75                         -- Estados de movimento livre do jogador
76                         WHEN ST_VIVO | ST_PRE_VULN | ST_VULN | ST_VULN_BLINK =>
77                             -- fantasma é mais rápido quando vivo
78                             IF ((fan_state(i) = ST_VIVO and atua_en(1) = '1') or
79                                 ((fan_state(i) /= ST_VIVO and atua_en(0) = '1')) THEN
80                                 -- Checa teclado para "agendar" um movimento
81                                 IF (keys_dir(i) /= NADA and keys_dir_old(i) = NADA) THEN
82                                     nxt_move(i) := keys_dir(i);
83                                 END IF;
84
85                                 IF (nxt_move(i) = CIMA and WALKABLE(fan_cim_cel(i))) THEN
86                                     fan_cur_dir(i) <= CIMA;
87                                     nxt_move(i) := NADA;
88                                 ELSIF (nxt_move(i) = DIREI and WALKABLE(fan_dir_cel(i))) THEN
89                                     fan_cur_dir(i) <= DIREI;
90                                     nxt_move(i) := NADA;
91                                 ELSIF (nxt_move(i) = BAIXO and WALKABLE(fan_bai_cel(i))) THEN
92                                     fan_cur_dir(i) <= BAIXO;
93                                     nxt_move(i) := NADA;
94                                 ELSIF (nxt_move(i) = ESQUE and WALKABLE(fan_esq_cel(i))) THEN
95                                     fan_cur_dir(i) <= ESQUE;
96                                     nxt_move(i) := NADA;
97                                 END IF;
98
99                                 IF (WALKABLE(fan_nxt_cel(i))) THEN -- atualiza posicao
100                                     IF (fan_pos_x(i) = MAP_X_MAX) then -- teletransporte
101                                         fan_pos_x(i) <= MAP_X_MIN + 1;
102                                     ELSIF (fan_pos_x(i) = MAP_X_MIN) then
103                                         fan_pos_x(i) <= MAP_X_MAX - 1;
104                                     ELSIF (fan_pos_y(i) = MAP_Y_MAX) then
105                                         fan_pos_y(i) <= MAP_Y_MIN + 1;
106                                     ELSIF (fan_pos_y(i) = MAP_Y_MIN) then
107                                         fan_pos_y(i) <= MAP_Y_MAX - 1;
108                                     ELSE
109                                         fan_pos_x(i) <= fan_pos_x(i) + DIRS(fan_cur_dir(i))(1);
110                                         fan_pos_y(i) <= fan_pos_y(i) + DIRS(fan_cur_dir(i))(0);
111                                     END IF;

```

```

112         END IF;
113
114         keys_dir_old(i) := keys_dir(i);
115     END IF;
116     -- Estados de piloto automático
117     WHEN ST_DEAD | ST_PRE_DEAD | ST_FIND_EXIT =>
118         IF (atua.en(2) = '1') THEN
119             -- Movimento automático do fantasma para a cela com velocidade maior
120             CASE FAN.PERCURSO(fan_pos_y(i), fan_pos_x(i)) IS
121                 WHEN 'C' =>
122                     fan_pos_y(i) <= fan_pos_y(i) - 1;
123                     fan_cur_dir(i) <= CIMA;
124                 WHEN 'B' =>
125                     fan_pos_y(i) <= fan_pos_y(i) + 1;
126                     fan_cur_dir(i) <= BAIXO;
127                 WHEN 'E' =>
128                     fan_pos_x(i) <= fan_pos_x(i) - 1;
129                     fan_cur_dir(i) <= ESQUE;
130                 WHEN 'D' =>
131                     fan_pos_x(i) <= fan_pos_x(i) + 1;
132                     fan_cur_dir(i) <= DIREI;
133                 WHEN OTHERS =>
134                     fan_pos_x(i) <= CELL.IN_X; -- restaura posição
135                     fan_pos_y(i) <= CELL.IN_Y;
136             END CASE;
137         END IF;
138         -- Apenas movimento vertical até sair da jaula
139         -- Supõe que fan_pos_x já vale CELL.IN_X, apenas anda pra cima
140         WHEN ST_FUGA =>
141             IF (atua.en(1) = '1') THEN
142                 fan_pos_y(i) <= fan_pos_y(i) - 1;
143                 fan_cur_dir(i) <= CIMA;
144             END IF;
145         END CASE;
146     END LOOP;
147     END IF;
148 END PROCESS p_atualiza_fan;
149
150
151 -- Gera o próximo estado de cada fantasma para atualização
152 -- e os sinais de controle do estado. Há uma FSM por fantasma.
153 -- type: combinational
154 p_fan_next_state: PROCESS (fan_state, spc_coin, pac_fans_hit, fan_tempo,
155                             fan_pos_x, fan_pos_y, atua_en)
156     VARIABLE pacman_dead_var, fan_died_var: STD_LOGIC;
157 BEGIN
158     pacman_dead_var := '0'; -- um OR de todos os fantasmas
159     fan_died_var := '0'; -- um OR de todos os fantasmas
160
161     FOR i in 0 to FAN.NO-1 LOOP
162         CASE fan_state(i) IS
163             WHEN ST_VIVO => -- estado normal, capaz de matar o pacman
164                 IF (pac_fans_hit(i) = '1') THEN
165                     pr_fan_state(i) <= ST_VIVO;
166                     pacman_dead_var := '1';
167                 ELSIF (spc_coin = '1') THEN
168                     pr_fan_state(i) <= ST_PRE_VULN;
169                 ELSE
170                     pr_fan_state(i) <= ST_VIVO;
171                 END IF;
172                 fan_rstn_tempo(i) <= '0';
173             WHEN ST_PRE_VULN => -- apenas zera o contador de tempo antes de VULN
174                 pr_fan_state(i) <= ST_VULN;
175                 fan_rstn_tempo(i) <= '0';
176                 pacman_dead <= '0';
177             WHEN ST_VULN => -- estado (temporário) sensível ao pacman
178                 IF (pac_fans_hit(i) = '1') THEN
179                     pr_fan_state(i) <= ST_PRE_DEAD;
180                     fan_died_var := '1';
181                 ELSIF (fan_tempo(i) > FAN.TIME_VULN_START.BLINK) THEN
182                     pr_fan_state(i) <= ST_VULN.BLINK;
183                 ELSIF (spc_coin = '1') THEN -- comer outra spc_coin deve zerar
184                     pr_fan_state(i) <= ST_PRE_VULN; -- o tempo de VULN
185                 ELSE
186                     pr_fan_state(i) <= ST_VULN;
187                 END IF;
188                 fan_rstn_tempo(i) <= '1';
189             WHEN ST_VULN.BLINK => -- parte final do estado vulnerável (piscante)
190                 IF (pac_fans_hit(i) = '1') THEN
191                     pr_fan_state(i) <= ST_PRE_DEAD;
192                     fan_died_var := '1';
193                 ELSIF (fan_tempo(i) > FAN.TIME_VULN_END) THEN
194                     pr_fan_state(i) <= ST_VIVO;
195                 ELSIF (spc_coin = '1') THEN
196                     pr_fan_state(i) <= ST_PRE_VULN;
197                 ELSE
198                     pr_fan_state(i) <= ST_VULN.BLINK;
199                 END IF;
200                 fan_rstn_tempo(i) <= '1';
201             END IF;
202         END CASE;
203     END LOOP;

```

```

204
205     WHEN ST.PREDEAD => --apenas zera contador de tempo antes de DEAD
206         pr_fan_state(i) <= ST.DEAD;
207         fan_rstn_tempo(i) <= '0';
208         pacman_dead <= '0';
209
210     WHEN ST.DEAD => --estado (temporário) sem controle do fantasma,
211         --este apenas foge para a cela
212         IF (fan_tempo(i) > FAN.TIME.DEAD) THEN
213             pr_fan_state(i) <= ST.FIND.EXIT;
214         ELSE
215             pr_fan_state(i) <= ST.DEAD;
216         END IF;
217         fan_rstn_tempo(i) <= '1';
218
219     WHEN ST.FIND.EXIT => --estado de busca da entrada da cela
220         IF (fan_pos_x(i) = CELL.IN.X and fan_pos_y(i) = CELL.IN.Y) THEN
221             pr_fan_state(i) <= ST.FUGA;
222         ELSE
223             pr_fan_state(i) <= ST.FIND.EXIT;
224         END IF;
225         fan_rstn_tempo(i) <= '0';
226
227     WHEN ST.FUGA => --fuga da cela
228         IF (fan_pos_y(i) = CELL.OUT.Y) THEN
229             pr_fan_state(i) <= ST.VIVO;
230         ELSE
231             pr_fan_state(i) <= ST.FUGA;
232         END IF;
233         fan_rstn_tempo(i) <= '0';
234     END CASE;
235 END LOOP;
236
237     pacman_dead <= pacman_dead_var;
238     fan_died <= fan_died_var;
239 END PROCESS p_fan_next_state;
240
241 -- Avança as FSMs para o próximos estados
242 -- type: sequential
243 seq_fsm_fan: PROCESS (clk, rstn)
244 BEGIN
245     IF (rstn = '0') THEN -- asynchronous reset (active low)
246         fan_state <= (OTHERS => ST.FIND.EXIT);
247     ELSIF (clk'event and clk = '1') THEN
248         fan_state <= pr_fan_state;
249     END IF;
250 END PROCESS seq_fsm_fan;
251
252 -- Contadores de tempo para os estados dos fantasmas
253 -- type: sequential
254 fan_counters: PROCESS (clk, fan_rstn_tempo)
255 BEGIN
256     FOR i in 0 to FAN.NO-1 LOOP
257         IF (fan_rstn_tempo(i) = '0') THEN
258             fan_tempo(i) <= 0;
259         ELSIF (clk'event and clk = '1') THEN
260             IF (atualiza = '1') THEN
261                 fan_tempo(i) <= fan_tempo(i) + 1;
262             END IF;
263         END IF;
264     END LOOP;
265 END PROCESS fan_counters;
266 END behav;

```

Listing 5: Controlador das frutas

```

1  LIBRARY ieee;
2  USE ieee.STD.LOGIC.1164.all;
3  USE ieee.NUMERIC.STD.all;
4  USE work.PAC.DEFS.all;
5
6  ENTITY ctrl_frutas IS
7      PORT (
8          clk, rstn          :IN STD.LOGIC;
9          enable             :IN STD.LOGIC;
10         fruta              :OUT t_fruta_id
11     );
12 END ctrl_frutas;
13
14 ARCHITECTURE behav OF ctrl_frutas IS
15     CONSTANT MIN_ESPERA: INTEGER := 2000;
16     CONSTANT DURACAO: INTEGER := 1000;
17
18     SIGNAL time_wait, rnd_cont: INTEGER range 0 to 50000;
19     SIGNAL time_dur: INTEGER range 0 to DURACAO;
20     SIGNAL frut_cont: INTEGER range 0 to FRUTA.NO;
21     SIGNAL fruta_reg: t_fruta_id := 1;
22 BEGIN

```

```

23 -- Conta os tempos alternadamente de espera e duração
24 -- da fruta_reg, amostrando contadores aleatórios
25 -- type: sequential
26 PROCESS (clk, rstn)
27 BEGIN
28     IF (rstn = '0') THEN
29         time_wait <= 0;
30         time_dur <= 0;
31         fruta_reg <= 1;
32     ELSIF (clk'event and clk = '1') THEN
33         IF (enable = '1') THEN
34             IF (fruta_reg = 0) THEN
35                 IF (time_wait = 0) THEN
36                     fruta_reg <= frut_cont + 1;
37                     time_dur <= DURACAO;
38                 ELSE
39                     time_wait <= time_wait - 1;
40                 END IF;
41             ELSE
42                 IF (time_dur = 0) THEN
43                     fruta_reg <= 0;
44                     time_wait <= MINESPERA + rnd_cont;
45                 ELSE
46                     time_dur <= time_dur - 1;
47                 END IF;
48             END IF;
49         END IF;
50     END IF;
51 END PROCESS;
52
53 fruta <= fruta_reg;
54
55 random: ENTITY work.counter PORT MAP
56 (clk => clk, rstn => '1', en => '1', max => 2*MINESPERA, q => rnd_cont);
57 fruta_random: ENTITY work.counter PORT MAP
58 (clk => clk, rstn => '1', en => '1', max => FRUTA_NO-1, q => frut_cont);
59 END behav;

```

Listing 6: Leitor de teclas

```

1 --
2 -- decodifica tecla pressionada
3 -- em direcao
4 --
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.numeric_std.all;
9 use work.pac_defs.all;
10
11 -- entrada: codigo proveniente do teclado
12 -- saida: mudancas de direcao do player
13 -- code entrada codificada das teclas
14 -- (maximo tres teclas pressionadas 16 bits cada)
15 entity player_dir is
16 port(
17     code: IN STD_LOGIC_VECTOR(47 downto 0);
18     p1_dir, p2_dir, p3_dir: OUT t_direcao
19 );
20 end entity player_dir;
21
22 architecture rtl of player_dir is
23     signal key_1, key_2, key_3 : std_logic_vector(15 downto 0);
24
25 begin
26     -- Modelo:
27     --> 2-players -> cada um aperta uma tecla
28     -- Codigo referente a cada tecla deve estar em key_1, key_2 ou key_3
29     -- Problemas referentes a entrada de teclas:
30     --> se um player aperta 3 teclas, o outro nao tera a sua tecla lida
31     key_1 <= code(47 downto 32); -- terceira tecla pressionada
32     key_2 <= code(31 downto 16); -- segunda tecla pressionada
33     key_3 <= code(15 downto 0); -- primeira tecla pressionada
34
35     -- P 1 Teclas
36     -- Movimentacao | Tecla | codigo
37     -- Cima | (Numpad 8) | x"0075"
38     -- Baixo | (Numpad 5) | x"0073"
39     -- Esquerda | (Numpad 4) | x"006B"
40     -- Direita | (Numpad 6) | x"0074"
41
42     -- P 2 Teclas
43     -- Movimentacao | Tecla | codigo
44     -- Cima | (W) | x"001d"
45     -- Baixo | (S) | x"001b"
46     -- Esquerda | (A) | x"001c"
47     -- Direita | (D) | x"0023"
48

```

```

49 -- P 3 Teclas
50 -- Movimentacao | Tecla | codigo
51 -- Cima          | (I)  | x"0043"
52 -- Baixo         | (K) | x"0042"
53 -- Esquerda      | (J) | x"003B"
54 -- Direita       | (L) | x"004B"
55
56 --Implementação mais simples para 2 players com 3 teclas
57 p1_dir <= CIMA WHEN (key_1 = x"0075" or key_2 = x"0075" or key_3 = x"0075")
58 ELSE DIREI WHEN (key_1 = x"0074" or key_2 = x"0074" or key_3 = x"0074")
59 ELSE BAIXO WHEN (key_1 = x"0073" or key_2 = x"0073" or key_3 = x"0073")
60 ELSE ESQUE WHEN (key_1 = x"006B" or key_2 = x"006B" or key_3 = x"006B")
61 ELSE NADA;
62
63 p2_dir <= CIMA WHEN (key_1 = x"001d" or key_2 = x"001d" or key_3 = x"001d")
64 ELSE DIREI WHEN (key_1 = x"0023" or key_2 = x"0023" or key_3 = x"0023")
65 ELSE BAIXO WHEN (key_1 = x"001b" or key_2 = x"001b" or key_3 = x"001b")
66 ELSE ESQUE WHEN (key_1 = x"001c" or key_2 = x"001c" or key_3 = x"001c")
67 ELSE NADA;
68
69 p3_dir <= CIMA WHEN (key_1 = x"0043" or key_2 = x"0043" or key_3 = x"0043")
70 ELSE DIREI WHEN (key_1 = x"004B" or key_2 = x"004B" or key_3 = x"004B")
71 ELSE BAIXO WHEN (key_1 = x"0042" or key_2 = x"0042" or key_3 = x"0042")
72 ELSE ESQUE WHEN (key_1 = x"003B" or key_2 = x"003B" or key_3 = x"003B")
73 ELSE NADA;
74 END rtl;

```

Listing 7: Display de 7-segmentos

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4
5 ENTITY disp IS
6     port(
7         CLK: IN STD_LOGIC;
8         EN: IN STD_LOGIC;
9         VIDAS: IN INTEGER range 0 to 5:=1;
10        PNT: IN INTEGER range 0 to 9999:=0;
11        PEDRAS: IN INTEGER range -10 to 255:=0;
12        seg0, seg1, seg2, seg3: OUT STD_LOGIC_VECTOR (6 downto 0) -- Display sete segmentos
13    );
14 END;
15
16 architecture struct of disp is
17     SIGNAL P0, P1, P2, P3: std_logic_vector(3 downto 0):="0000"; -- recebe pontuacao passada pelo top level
18     SIGNAL HEX0, HEX1, HEX2, HEX3: STD_LOGIC_VECTOR(6 downto 0):="0000000"; --intermediario pontuacao
19     SIGNAL aux_seg0, aux_seg1, aux_seg2, aux_seg3: STD_LOGIC_VECTOR(6 downto 0):="1111111"; --usado para
        rolagem do display
20
21     COMPONENT conv_7seg IS
22     PORT (x: IN STD_LOGIC_VECTOR(3 downto 0);
23          y: OUT STD_LOGIC_VECTOR(6 downto 0));
24     END COMPONENT conv_7seg;
25
26     SIGNAL cont1, cont2: INTEGER range 0 to 100;
27     SIGNAL show_pts, frase_rstn: STD_LOGIC;
28 BEGIN
29     PROCESS(PNT)
30     BEGIN
31         P0 <= std_logic_vector(to_unsigned(PNT mod 10,4)); --unidade
32         P1 <= std_logic_vector(to_unsigned((PNT/10 mod 10),4)); --dezena
33         P2 <= std_logic_vector(to_unsigned((PNT/100 mod 10),4)); --centena
34         P3 <= std_logic_vector(to_unsigned((PNT/1000 mod 10),4)); --milhar
35     END PROCESS;
36
37     PROCESS (clk)
38     VARIABLE alfa_code: STD_LOGIC_VECTOR(6 downto 0):="0000000"; -- intermediario letras
39     BEGIN
40         IF (clk'event and clk = '1') THEN
41             IF (en = '1') THEN
42                 IF (PEDRAS <= 0) THEN
43                     case (cont1) IS
44                         WHEN 0 =>
45                             alfa_code := "0010001"; --Y
46                         WHEN 1 =>
47                             alfa_code := "1000000"; --O
48                         WHEN 2 =>
49                             alfa_code := "1000001"; --U
50                         WHEN 3 =>
51                             alfa_code := "1111111"; --
52                         WHEN 4 =>
53                             alfa_code := "0001000"; --A
54                         WHEN 5 =>
55                             alfa_code := "0101111"; --R
56                         WHEN 6 =>
57                             alfa_code := "0000110"; --E
58                         WHEN 7 =>

```

```

59         alfa_code := "1111111"; --
60     WHEN 8 =>
61         alfa_code := "0000111"; --T
62     WHEN 9 =>
63         alfa_code := "0001011"; --H
64     WHEN 10 =>
65         alfa_code := "0000110"; --E
66     WHEN 11 =>
67         alfa_code := "1111111"; --
68     WHEN 12 =>
69         alfa_code := "0000011"; --B
70     WHEN 13 =>
71         alfa_code := "0000110"; --E
72     WHEN 14 =>
73         alfa_code := "0010010"; --S
74     WHEN 15 =>
75         alfa_code := "0000111"; --T
76     WHEN others =>
77         alfa_code := "1111111";
78     END case;
79
80     ELSIF (VIDAS = 0) THEN
81         case (cont2) IS
82             WHEN 0 =>
83                 alfa_code := "0010001"; --Y
84             WHEN 1 =>
85                 alfa_code := "1000000"; --O
86             WHEN 2 =>
87                 alfa_code := "1000001"; --U
88             WHEN 3 =>
89                 alfa_code := "1111111"; --
90             WHEN 4 =>
91                 alfa_code := "1000111"; --L
92             WHEN 5 =>
93                 alfa_code := "1000000"; --O
94             WHEN 6 =>
95                 alfa_code := "0010010"; --S
96             WHEN 7 =>
97                 alfa_code := "0000110"; --E
98             WHEN others =>
99                 alfa_code := "1111111"; --
100         END case;
101     END IF;
102
103     IF (PEDRAS<=0 or VIDAS=0) THEN
104         aux_seg3 <= aux_seg2; -- display
105         aux_seg2 <= aux_seg1; -- rolante
106         aux_seg1 <= aux_seg0;
107         aux_seg0 <= alfa_code;
108     ELSE
109         aux_seg3 <= "1111111"; -- apaga palavras
110         aux_seg2 <= "1111111"; -- quANDO jogo e
111         aux_seg1 <= "1111111"; -- reiniciado ou
112         aux_seg0 <= "1111111"; -- iniciado
113     END IF;
114 END IF;
115 END IF;
116 END PROCESS;
117
118 frase_rstn <= '1' WHEN (PEDRAS <=0 or VIDAS = 0)
119 ELSE '0';
120
121 p_contador0: ENTITY WORK.counter
122 PORT MAP (clk => CLK,
123           rstn => frase_rstn,
124           en => en,
125           max => 24,
126           q => cont1);
127
128 p_contador1: ENTITY WORK.counter
129 PORT MAP (clk => CLK,
130           rstn => frase_rstn,
131           en => en,
132           max => 16,
133           q => cont2);
134
135 hexseg0: conv_7seg port map(P0, HEX0); --casa da unidade da pontuacao
136 hexseg1: conv_7seg port map(P1, HEX1); --casa da dezena da pontuacao
137 hexseg2: conv_7seg port map(P2, HEX2); --casa centena da pontuacao
138 hexseg3: conv_7seg port map(P3, HEX3); --casa de mil pontos
139
140 --se nem perdeu todas as vidas ou terminou todas as pecas mostra pontuacao,
141 --caso contrario mostra as frases.
142 show_pts <= '1' WHEN ((PEDRAS <= 0 and cont1 >= 20) or (VIDAS = 0 and cont2 >= 12)
143                     or (VIDAS /=0 and PEDRAS > 0))
144 ELSE '0';
145
146 seg0 <= HEX0 WHEN (show_pts = '1') ELSE aux_seg0;
147 seg1 <= HEX1 WHEN (show_pts = '1') ELSE aux_seg1;
148 seg2 <= HEX2 WHEN (show_pts = '1') ELSE aux_seg2;
149 seg3 <= HEX3 WHEN (show_pts = '1') ELSE aux_seg3;
150 END struct;

```


Listing 8: Controlador VGA

```

1  -- VGA controller component
2  -- Based on original design by Rafael Auler
3  --
4  -- Modified to generate up to 640x480 different
5  -- pixels through mapping logical blocks of size
6  -- 5x5 pixels into user-defined RGB sprites
7  -- Also, it has two logical screen maps, overlaid
8  -- one on top of another.
9
10 library ieee;
11 use ieee.std_logic_1164.all;
12 use ieee.numeric_std.all;
13 USE work.PAC_DEFS.all;
14 USE work.PAC_SPRITES.all;
15
16 entity vgacon is
17   generic (
18     -- When changing this, remember to keep 4:3 aspect ratio
19     -- Must also keep in mind that our native resolution is 640x480, and
20     -- you can't cross these bounds (although you will seldom have enough
21     -- on-chip memory to instantiate this module with higher res).
22     NUMHORZBLOCKS : natural := 128; -- Number of horizontal pixels
23     NUMVERTBLOCKS : natural := 96;  -- Number of vertical pixels
24   )
25   port (
26     clk27M, rstn      : in  std_logic;
27     write_clk, write_enable : in  std_logic;
28     write_addr        : in  integer range 0 to
29       NUMHORZBLOCKS * NUMVERTBLOCKS - 1;
30     data_in           : in  t_blk_sym;
31     vga_clk           : buffer std_logic; -- Ideally 25.175 MHz
32     vga_pixel         : out t_color_3b;  -- at 25.2 MHz
33     data_block        : out t_blk_sym;   -- at 27 MHz
34     hsync, vsync      : out std_logic;
35     ovl_in            : in  t_ovl_blk_sym;
36     ovl_we            : in  std_logic;
37   end vgacon;
38
39 architecture behav of vgacon is
40   -- Two signals: one is delayed by one clock cycle. The monitor control uses
41   -- the delayed one. We need a counter 1 clock cycle earlier, relative
42   -- to the monitor signal, in order to index the memory contents
43   -- for the next cycle, when the pixel is in fact sent to the monitor.
44   signal h_count, h_count_d : integer range 0 to 799; -- horizontal counter
45   signal v_count, v_count_d : integer range 0 to 524; -- vertical counter
46
47   signal read_addr : integer range 0 to NUMHORZBLOCKS * NUMVERTBLOCKS - 1;
48   signal h_drawarea, v_drawarea : std_logic;
49   signal vga_data_out : t_blk_sym;
50   signal vga_ovl_data_out : t_ovl_blk_sym;
51   signal id_vga_data_out, id_data_block, id_data_in : t_blk_id;
52   signal id_ovl_in, id_vga_ovl_data_out : t_ovl_blk_id;
53   signal vga_pixel_0 : t_color_3b;
54 begin
55
56   -- This is our PLL (Phase Locked Loop) to divide the DE1 27 MHz
57   -- clock and produce a 25.2MHz clock adequate to our VGA controller
58   divider : work.vga_pll port map (clk27M, vga_clk);
59
60   -- This is our main dual clock RAM. We use our VGA clock to read contents from
61   -- memory (block type value). The user of this module may use any clock
62   -- to write contents to this memory, modifying blocks individually.
63   vgamem0 : work.dual_clock_ram -- RAM used for the background elements
64   generic map (
65     MEMSIZE => NUMHORZBLOCKS * NUMVERTBLOCKS,
66     MEMWDT => 4)
67   port map (
68     a_clk      => vga_clk,
69     b_clk      => write_clk,
70     a_address  => read_addr,
71     b_address  => write_addr,
72     b_data_in  => id_data_in,
73     a_data_out => id_vga_data_out,
74     b_data_out => id_data_block,
75     b_we       => write_enable);
76   vgamem1 : work.dual_clock_ram -- RAM used for overlaid characters
77   generic map (
78     MEMSIZE => NUMHORZBLOCKS * NUMVERTBLOCKS,
79     MEMWDT => 9)
80   port map (
81     a_clk      => vga_clk,
82     b_clk      => write_clk,
83     a_address  => read_addr,
84     b_address  => write_addr,
85     b_data_in  => id_ovl_in,

```

```

86     a_data_out      => id_vga_ovl_data_out ,
87     b_we           => ovl_we);
88
89 -- Block enumeration to/from logic_vector conversions
90 id_data_in         <= std_logic_vector(to_unsigned(t_blk_sym'pos(data_in), 4));
91 id_ovl_in          <= std_logic_vector(to_unsigned(t_ovl_blk_sym'pos(ovl_in), 9));
92 vga_data_out       <= t_blk_sym'val(to_integer(unsigned(id_vga_data_out)));
93 data_block         <= t_blk_sym'val(to_integer(unsigned(id_data_block)));
94 vga_ovl_data_out   <= t_ovl_blk_sym'val(to_integer(unsigned(id_vga_ovl_data_out)));
95
96 -- purpose: Increments the current horizontal position counter
97 -- type : sequential
98 -- inputs : vga_clk , rstn
99 -- outputs: h_count , h_count_d
100 horz_counter: process (vga_clk , rstn)
101 begin -- process horz_counter
102     if rstn = '0' then -- asynchronous reset (active low)
103         h_count <= 0;
104         h_count_d <= 0;
105     elsif vga_clk'event and vga_clk = '1' then -- rising clock edge
106         h_count_d <= h_count; -- 1 clock cycle delayed counter
107         if h_count = 799 then
108             h_count <= 0;
109         else
110             h_count <= h_count + 1;
111         end if;
112     end if;
113 end process horz_counter;
114
115 -- purpose: Determines if we are in the horizontal "drawable" area
116 -- type : combinational
117 -- inputs : h_count_d
118 -- outputs: h_drawarea
119 horz_sync: process (h_count_d)
120 begin -- process horz_sync
121     if h_count_d < 640 then
122         h_drawarea <= '1';
123     else
124         h_drawarea <= '0';
125     end if;
126 end process horz_sync;
127
128 -- purpose: Increments the current vertical counter position
129 -- type : sequential
130 -- inputs : vga_clk , rstn
131 -- outputs: v_count , v_count_d
132 vert_counter: process (vga_clk , rstn)
133 begin -- process vert_counter
134     if rstn = '0' then -- asynchronous reset (active low)
135         v_count <= 0;
136         v_count_d <= 0;
137     elsif vga_clk'event and vga_clk = '1' then -- rising clock edge
138         v_count_d <= v_count; -- 1 clock cycle delayed counter
139         if h_count = 699 then
140             if v_count = 524 then
141                 v_count <= 0;
142             else
143                 v_count <= v_count + 1;
144             end if;
145         end if;
146     end if;
147 end process vert_counter;
148
149 -- purpose: Updates information based on vertical position
150 -- type : combinational
151 -- inputs : v_count_d
152 -- outputs: v_drawarea
153 vert_sync: process (v_count_d)
154 begin -- process vert_sync
155     if v_count_d < 480 then
156         v_drawarea <= '1';
157     else
158         v_drawarea <= '0';
159     end if;
160 end process vert_sync;
161
162 -- purpose: Generates synchronization signals
163 -- type : combinational
164 -- inputs : v_count_d , h_count_d
165 -- outputs: hsync , vsync
166 sync: process (v_count_d , h_count_d)
167 begin -- process sync
168     if (h_count_d >= 659) and (h_count_d <= 755) then
169         hsync <= '0';
170     else
171         hsync <= '1';
172     end if;
173     if (v_count_d >= 493) and (v_count_d <= 494) then
174         vsync <= '0';
175     else
176         vsync <= '1';
177     end if;

```

```

178 end process sync;
179
180 -- determines whether we are in drawable area on screen a.t.m.
181 drawarea <= v_drawarea and h_drawarea;
182
183 -- purpose: calculates the controller memory address to read block data
184 -- type : combinational
185 -- inputs : h_count, v_count
186 -- outputs: read_addr
187 gen_r_addr: process (h_count, v_count)
188 begin -- process gen_r_addr
189     read_addr <= h_count / (640 / NUM_HORIZ_BLOCKS)
190         + ((v_count / (480 / NUM_VERT_BLOCKS))
191             * NUM_HORIZ_BLOCKS);
192 end process gen_r_addr;
193
194 -- Build color signals based on memory output and "drawarea" signal
195 -- (if we are not in the drawable area of 640x480, must deassert all
196 -- color signals).
197 PROCESS (vga_data_out, vga_ovl_data_out, drawarea, h_count, v_count)
198 VARIABLE pixel_normal, pixel_ovl: t_color_3b;
199 BEGIN
200     pixel_ovl(0) := OVL_SPRITES_RED(vga_ovl_data_out)(v_count mod 5, (h_count+4) mod 5);
201     pixel_ovl(1) := OVL_SPRITES_GRN(vga_ovl_data_out)(v_count mod 5, (h_count+4) mod 5);
202     pixel_ovl(2) := OVL_SPRITES_BLU(vga_ovl_data_out)(v_count mod 5, (h_count+4) mod 5);
203
204     pixel_normal(0) := SPRITES_RED(vga_data_out)(v_count mod 5, (h_count+4) mod 5);
205     pixel_normal(1) := SPRITES_GRN(vga_data_out)(v_count mod 5, (h_count+4) mod 5);
206     pixel_normal(2) := SPRITES_BLU(vga_data_out)(v_count mod 5, (h_count+4) mod 5);
207
208     IF (drawarea = '1') THEN
209     IF (pixel_ovl /= "000") THEN
210         vga_pixel_0 <= pixel_ovl;
211     ELSE
212         vga_pixel_0 <= pixel_normal;
213     END IF;
214     ELSE
215         vga_pixel_0 <= "000";
216     END IF;
217 END PROCESS;
218
219 -- Here, we register the pixel before sending to VGA pin
220 -- in order to help compiler recognize the critical path
221 -- to the pin and set the timing constraints accordingly.
222 -- It prevents glitches displayed in the screen.
223 PROCESS (vga_clk)
224 BEGIN
225     IF (vga_clk'event and vga_clk = '1') THEN
226         vga_pixel <= vga_pixel_0;
227     END IF;
228 END PROCESS;
229 end behav;
230
231
232 -----
233 -- The following entity is a dual clock RAM (read operates at different
234 -- clock from write). This is used to isolate two clock domains. The first
235 -- is the 25.2 MHz clock domain in which our VGA controller needs to operate.
236 -- This is the read clock, because we read from this memory to determine
237 -- the color of a pixel. The second is the clock domain of the user of this
238 -- module, writing in the memory the contents it wants to display in the VGA.
239 -----
240
241 library ieee;
242 use ieee.std_logic_1164.all;
243 USE work.PAC_DEFS.all;
244
245 entity dual_clock_ram is
246
247     generic (
248         MEMSIZE : natural;
249         MEMWDT : natural);
250     port (
251         a_clk, b_clk : in std_logic; -- support different clocks
252         b_data_in : in std_logic_vector(MEMWDT-1 downto 0); -- only b writes
253         a_address, b_address : in integer range 0 to MEMSIZE - 1;
254         b_we : in std_logic; -- write enable
255         a_data_out, b_data_out : out std_logic_vector(MEMWDT-1 downto 0));
256 end dual_clock_ram;
257
258 architecture behav of dual_clock_ram is
259     -- we only want to address (store) MEMSIZE elements
260     subtype addr is integer range 0 to MEMSIZE - 1;
261     type mem is array (addr) of std_logic_vector(MEMWDT-1 downto 0);
262     signal ram_block : mem;
263     -- we don't care with read after write behavior (whether ram reads
264     -- old or new data in the same cycle).
265     attribute ramstyle : string;
266     attribute ramstyle of dual_clock_ram : entity is "no_rw_check";
267 begin -- behav
268
269     -- purpose: Reads data from RAM

```

```

270 -- type : sequential
271 a: process (a_clk)
272 begin -- process read
273 if (a_clk'event and a_clk = '1') then -- rising clock edge
274 a_data_out <= ram_block(a_address);
275 end if;
276 end process a;
277
278 -- purpose: Reads/Writes data to RAM
279 -- type : sequential
280 b: process (b_clk)
281 begin -- process write
282 if (b_clk'event and b_clk = '1') then -- rising clock edge
283 if (b_we = '1') then
284 ram_block(b_address) <= b_data_in;
285 end if;
286 b_data_out <= ram_block(b_address);
287 end if;
288 end process b;
289
290 end behav;
291
292
293 -- The following entity is automatically generated by Quartus (a megafunction).
294 -- As Altera DE1 board does not have a 25.175 MHz, but a 27 Mhz, we
295 -- instantiate a PLL (Phase Locked Loop) to divide out 27 MHz clock
296 -- and reach a satisfiable 25.2MHz clock for our VGA controller (14/15 ratio)
297
298
299 LIBRARY ieee;
300 USE ieee.std_logic_1164.all;
301
302 LIBRARY altera_mf;
303 USE altera_mf.all;
304
305 ENTITY vga_pll IS
306 PORT
307 (
308   inclk0 : IN STDLOGIC := '0';
309   c0 : OUT STDLOGIC
310 );
311 END vga_pll;
312
313
314 ARCHITECTURE SYN OF vga_pll IS
315
316   SIGNAL sub_wire0 : STD_LOGIC_VECTOR (5 DOWNTO 0);
317   SIGNAL sub_wire1 : STD_LOGIC;
318   SIGNAL sub_wire2 : STD_LOGIC;
319   SIGNAL sub_wire3 : STD_LOGIC_VECTOR (1 DOWNTO 0);
320   SIGNAL sub_wire4_bv : BIT_VECTOR (0 DOWNTO 0);
321   SIGNAL sub_wire4 : STD_LOGIC_VECTOR (0 DOWNTO 0);
322
323
324
325   COMPONENT altpll
326   GENERIC (
327     clk0_divide_by : NATURAL;
328     clk0_duty_cycle : NATURAL;
329     clk0_multiply_by : NATURAL;
330     clk0_phase_shift : STRING;
331     compensate_clock : STRING;
332     inclk0_input_frequency : NATURAL;
333     intended_device_family : STRING;
334     lpm_hint : STRING;
335     lpm_type : STRING;
336     operation_mode : STRING;
337     port_activeclock : STRING;
338     port_areset : STRING;
339     port_clkbad0 : STRING;
340     port_clkbad1 : STRING;
341     port_clkloss : STRING;
342     port_clkswitch : STRING;
343     port_configupdate : STRING;
344     port_fbin : STRING;
345     port_inclk0 : STRING;
346     port_inclk1 : STRING;
347     port_locked : STRING;
348     port_pfdena : STRING;
349     port_phasecounterselect : STRING;
350     port_phasedone : STRING;
351     port_phasestep : STRING;
352     port_phaseupdown : STRING;
353     port_pllena : STRING;
354     port_scanaclr : STRING;
355     port_scanclk : STRING;
356     port_scanclkena : STRING;
357     port_scandata : STRING;
358     port_scandataout : STRING;
359     port_scandone : STRING;
360     port_scanread : STRING;
361     port_scanwrite : STRING;

```

```

362     port_clk0      : STRING;
363     port_clk1      : STRING;
364     port_clk2      : STRING;
365     port_clk3      : STRING;
366     port_clk4      : STRING;
367     port_clk5      : STRING;
368     port_clkena0    : STRING;
369     port_clkena1    : STRING;
370     port_clkena2    : STRING;
371     port_clkena3    : STRING;
372     port_clkena4    : STRING;
373     port_clkena5    : STRING;
374     port_extclk0    : STRING;
375     port_extclk1    : STRING;
376     port_extclk2    : STRING;
377     port_extclk3    : STRING
378 );
379 PORT (
380     inclk   : IN STD.LOGIC.VECTOR (1 DOWNT0 0);
381     clk     : OUT STD.LOGIC.VECTOR (5 DOWNT0 0)
382 );
383 END COMPONENT;
384
385 BEGIN
386     sub_wire4_bv(0 DOWNT0 0) <= "0";
387     sub_wire4      <= To_stdlogicvector(sub_wire4_bv);
388     sub_wire1      <= sub_wire0(0);
389     c0             <= sub_wire1;
390     sub_wire2      <= inclk0;
391     sub_wire3      <= sub_wire4(0 DOWNT0 0) & sub_wire2;
392
393     altpll_component : altpll
394     GENERIC MAP (
395         clk0_divide_by => 15,
396         clk0_duty_cycle => 50,
397         clk0_multiply_by => 14,
398         clk0_phase_shift => "0",
399         compensate_clock => "CLK0",
400         inclk0_input_frequency => 37037,
401         intended_device_family => "Cyclone II",
402         lpm_hint => "CBX_MODULE_PREFIX=vga_pll",
403         lpm_type => "altpll",
404         operation_mode => "NORMAL",
405         port_activeclock => "PORT_UNUSED",
406         port_areset => "PORT_UNUSED",
407         port_clkbad0 => "PORT_UNUSED",
408         port_clkbad1 => "PORT_UNUSED",
409         port_clkloss => "PORT_UNUSED",
410         port_clkswitch => "PORT_UNUSED",
411         port_configupdate => "PORT_UNUSED",
412         port_fbin => "PORT_UNUSED",
413         port_inclk0 => "PORT_UNUSED",
414         port_inclk1 => "PORT_UNUSED",
415         port_locked => "PORT_UNUSED",
416         port_pfdena => "PORT_UNUSED",
417         port_phasecounterselct => "PORT_UNUSED",
418         port_phasedone => "PORT_UNUSED",
419         port_phasestep => "PORT_UNUSED",
420         port_phaseupdown => "PORT_UNUSED",
421         port_pllena => "PORT_UNUSED",
422         port_scanaclr => "PORT_UNUSED",
423         port_scanclk => "PORT_UNUSED",
424         port_scanclkena => "PORT_UNUSED",
425         port_scandata => "PORT_UNUSED",
426         port_scandataout => "PORT_UNUSED",
427         port_scandone => "PORT_UNUSED",
428         port_scanread => "PORT_UNUSED",
429         port_scanwrite => "PORT_UNUSED",
430         port_clk0 => "PORT_USED",
431         port_clk1 => "PORT_UNUSED",
432         port_clk2 => "PORT_UNUSED",
433         port_clk3 => "PORT_UNUSED",
434         port_clk4 => "PORT_UNUSED",
435         port_clk5 => "PORT_UNUSED",
436         port_clkena0 => "PORT_UNUSED",
437         port_clkena1 => "PORT_UNUSED",
438         port_clkena2 => "PORT_UNUSED",
439         port_clkena3 => "PORT_UNUSED",
440         port_clkena4 => "PORT_UNUSED",
441         port_clkena5 => "PORT_UNUSED",
442         port_extclk0 => "PORT_UNUSED",
443         port_extclk1 => "PORT_UNUSED",
444         port_extclk2 => "PORT_UNUSED",
445         port_extclk3 => "PORT_UNUSED"
446     )
447     PORT MAP (
448         inclk => sub_wire3,
449         clk => sub_wire0
450     );
451
452 END SYN;

```

6 Conclusão