Jogo “PACASA” (Crossy Road)

Bernardo Santos e João Domingos

Engenharia Eletrotécnica e de Computadores
Eletrónica Programável

2222033@my.ipleiria.pt, 2222034@my.ipleiria.pt

Resumo

Este documento refere o relatório do desenvolvimento do jogo PACASA. Inicialmente é apresentada a introdução e objetivos do trabalho, de seguida é apresentado o jogo PACASA, bem como a explicação dos blocos principais que o constituem e diagramas de funcionamento. No final do documento constam os resultados do trabalho e a conclusão do mesmo. O código é todo desenvolvido em VHDL e é implementado numa placa Digilent Basys 3 a partir do software Vivado.

Os requisitos para este projeto são os seguintes:

- Interação com o utilizador;
- O utilizador pode ganhar e perder pontos e/ou ganhar ou perder vidas.

1. Introdução

O jogo PACASA foi concebido para oferecer uma experiência simples e divertida, baseada no conceito de atravessar obstáculos dinâmicos. Durante o desenvolvimento, foram aplicados princípios fundamentais de programação, design de jogos e lógica computacional.

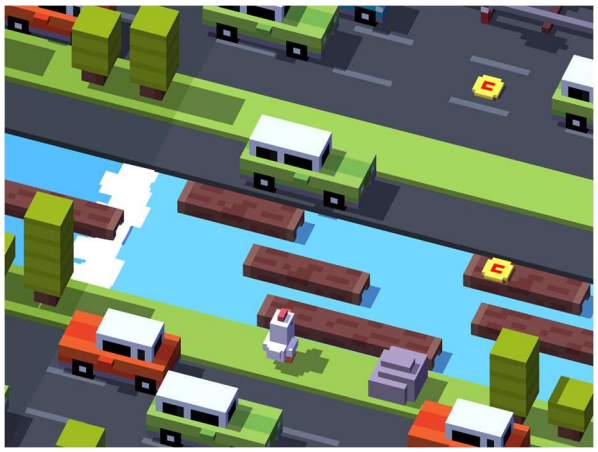

Figura 1. Crossy Road (jogo original)

No nosso jogo a galinha é a personagem principal e é controlada pelo utilizador através teclado ou botões da FPGA. A galinha poder ser movimentada para frente, para trás e para os lados. Por cada chegada da galinha ao topo do ecrã o jogador ganha um ponto até chegar ao nível 20, onde chegará a casa. No entanto, caso a galinha toque em um carro ou caia na água (não conseguindo pousar em um tronco), volta automaticamente ao início.

Os carros movimentam-se de forma horizontal em diferentes direções e velocidades, simulando o trânsito de uma estrada. Já os troncos deslocam-se em rios, servindo como plataformas móveis para

a galinha atravessar. Estes elementos dinâmicos aumentam gradualmente a dificuldade do jogo, incentivando o jogador a desenvolver estratégias rápidas e precisas para alcançar o objetivo.

3. PACASA

O código VHDL contém blocos como: vga_top, chicken, vga_sync led_driver, , clockwizard, basys3_master.

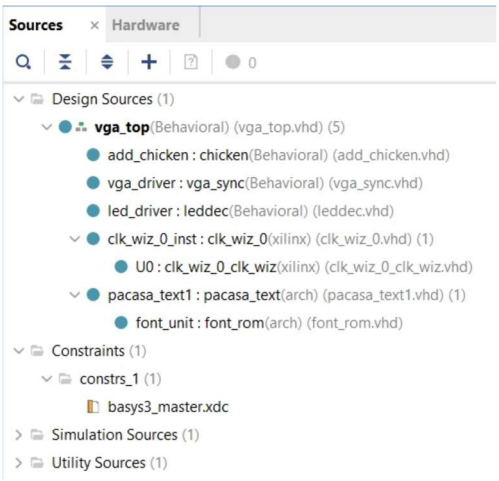

Figura 2.1. Sources

2.1- Vga_top

O bloco vga_top tem como função juntar todos os blocos que o compõe (chicken, vga_sync, led_driver). É neste bloco onde é feita a configuração dos botões de movimento da galinha (b_up, b_down, b_left, b_right), também como o botão de reset que coloca a galinha a meio do ecrã em baixo do ecrã (ponto inicial do jogo) e finalmente o score (pontuação).

```
vga_driver : entity work.vga_sync
PORT MAP(
    pixel_clk => ck_25,  -- Clock de pixel
    red_in => S_red,  -- Canal de cor vermelha
    green_in => S_green,  -- Canal de cor verde
    blue_in => S_blue,  -- Canal de cor azul
    red_out => vga_red(3),  -- Saída do canal vermelho
    green_out => vga_green(1),  -- Saída do canal verde
    blue_out => vga_blue(1),  -- Saída do canal azul
    pixel_row => S_pixel_row,  -- Linha do pixel
    pixel_col => S_pixel_col,  -- Coluna do pixel
    hsync => vga_hsync,  -- Sinal de sincronização horizontal
    vsync => S_vsync  -- Sinal de sincronização vertical
);

vga_vsync <= S_vsync; -- Conecta a saída de sincronização vertical

-- Instancia o componente "leddec" que controla os displays de 7 segmentos
led_driver : entity work.leddec
PORT MAP(
    dig => led_mpx,  -- Clock para multiplexação de displays
    f_data => S_data,  -- Dados a serem exibidos
    anode => anode,  -- Sinal para os anodos do display
    seg => seg  -- Sinais para os segmentos do display
);
```

Figura 2.2. Excerto código Vga_top

2.2- Chicken

O bloco chicken é responsável por todo o design e movimentação do jogo. Neste caso é composto pelo design da galinha, dos carros, dos troncos e das barras que simulam a estrada e o rio, e também das suas movimentações.

Para o design da galinha foi adicionado a CHICKEN_ROM é composto por uma estrutura de 12 por 16 pixéis. A galinha pode ser movimentada para direita, esquerda, cima e baixo através dos botões da FPGA.

```
    type rom_type_chicken is array (0 to 11,0 to 15) of
        std_logic_vector (2 downto 0);
    constant CHICKEN_ROM: rom_type_chicken :=
(
  ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
  ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "110", "110", "000", "000", "000", "000"),
  ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "110", "000", "110", "000", "000", "000"),
  ("000", "000", "110", "110", "000", "000", "000", "000", "000", "110", "000", "110", "000", "110", "000", "000"),
  ("000", "000", "110", "000", "000", "000", "000", "000", "000", "000", "000", "000", "110", "000", "000", "000"),
  ("000", "000", "110", "000", "000", "000", "000", "000", "000", "000", "110", "000", "110", "000", "000", "000"),
  ("000", "000", "000", "110", "000", "110", "000", "000", "000", "000", "000", "110", "000", "000", "000", "000"),
  ("000", "000", "000", "000", "110", "000", "000", "000", "000", "000", "110", "000", "000", "000", "000", "000"),
  ("000", "000", "000", "000", "000", "110", "110", "110", "110", "110", "000", "000", "000", "000", "000", "000"),
  ("000", "000", "000", "000", "000", "000", "000", "110", "000", "110", "000", "000", "000", "000", "000", "000"),
  ("000", "000", "000", "000", "000", "000", "000", "110", "110", "110", "000", "000", "000", "000", "000", "000"),
  ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000")
);

    signal chicken_square_on: std_logic;
    signal chicken_rgb: std_logic_vector(2 downto 0);
    signal rom_addr_chicken: unsigned(3 downto 0);
    signal rom_col_chicken: signed(5 downto 0);
    signal rom_bit_chicken: std_logic_vector(2 downto 0);
```

Figura 2.3. - Rom da galinha

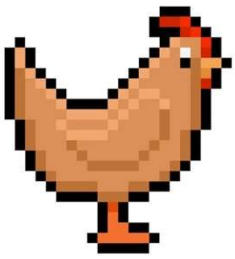

Figura 2.4. Galinha "utilizada"

Embora tenha sido introduzida a ROM da galinha no nosso código, não conseguimos fazer com que apareca no ecrã, dessa maneira decidimos alterar para algo mais simples de modo a ser implementável. Decidimos substituir a galinha por um quadrado cor-de-rosa (quase cor-de-pele). O quadrado tem de lado 8 píxeis, semelhante ao carro que de seguida será apresentado.

```
mchicken : PROCESS
BEGIN
   WAIT UNTIL rising_edge(v_sync);
   IF up = '1' THEN
        direction <= 1;
   ELSIF down = '1' THEN
        direction <= 2;
   ELSIF left = '1' THEN
        direction <= 3;
   ELSIF right = '1' THEN
        direction <= 4;
   ELSIF reset = '1' THEN
        direction <= 5;
   ELSE
        direction <= 0;
   END IF;

   IF direction = 1 THEN
        chicken_y <= chicken_y - chicken_hop;
   ELSIF direction = 2 THEN
        chicken_y <= chicken_y + chicken_hop;
   ELSIF direction = 3 THEN
        chicken_x <= chicken_x - chicken_hop;
   ELSIF direction = 4 THEN
        chicken_x <= chicken_x + chicken_hop;
   ELSIF direction = 5 THEN
        chicken_x <= CONV_STD_LOGIC_VECTOR(465, 11);
        chicken_y <= CONV_STD_LOGIC_VECTOR(580, 11);
        chicken_dead <= '0';
        win <= '0';
   ELSIF direction = 0 THEN
        chicken_x <= chicken_x;
   END IF;
   --- collision detection for car1, 2 and 3
   IF ((chicken_x >= car1_x - 28 AND chicken_x <= car1_x + 28)
   AND (chicken_y >= car1_y - size_car_x AND chicken_y <= car1_y + size_car_x)) OR
   ((chicken_x >= car2_x - 28 AND chicken_x <= car2_x + 28)
   AND (chicken_y >= car2_y - size_car_x AND chicken_y <= car2_y + size_car_x)) OR
   ((chicken_x >= car3_x - 28 AND chicken_x <= car3_x + 28)
   AND (chicken_y >= car3_y - size_car_x AND chicken_y <= car3_y + size_car_x)) OR
   ((chicken_x >= car4_x - 28 AND chicken_x <= car4_x + 28)
   AND (chicken_y >= car4_y - size_car_x AND chicken_y <= car4_y + size_car_x)) OR
   ((chicken_x >= car5_x - 28 AND chicken_x <= car5_x + 28)
```

Figura 2.5. Excerto código movimentação da galinha

Figura 2.6. Botões movimento

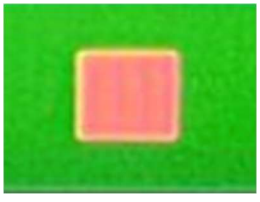

Figura 2.7. Foto demonstrativa galinha

O design dos carros é composto por um retângulo de 24 bits de comprimento e 16 de largura. Visto que não conseguimos implementar as ROM's dos carros tivemos de optar por desenhar um retângulo.

No que diz respeito às suas movimentações, o jogo é composto por quatro carros, que se movimentam em sentidos opostos (sempre na horizontal) e também com velocidades diferentes.

```
c1draw : PROCESS (car1_x, car1_y, pixel_row, pixel_col) IS
BEGIN
    IF (pixel_col >= car1_x - size_car_x) AND
     (pixel_col <= car1_x + size_car_x) AND
         (pixel_row >= car1_y - size_car_y) AND
         (pixel_row <= car1_y + size_car_y) THEN
            car1_on <= '1';
    ELSE
        car1_on <= '0';
    END IF;
    END PROCESS;
    -- process to move car1 once every frame (i.e. once every vsync pulse)
    mcar1 : PROCESS
    BEGIN
        WAIT UNTIL rising_edge(v_sync);
        -- allow for bounce off top or bottom of screen
        IF car1_x + size_car_x >= 900 THEN
            car1_x_motion <= "11111111100"; -- -4 pixels
        ELSIF car1_x <= size_car_x THEN
            car1_x_motion <= "00000000100"; -- +4 pixels
        END IF;
        car1_x <= car1_x + car1_x_motion; -- compute next car1 position
    END PROCESS;
```

Figura 2.8. Design e movimentação dos carros

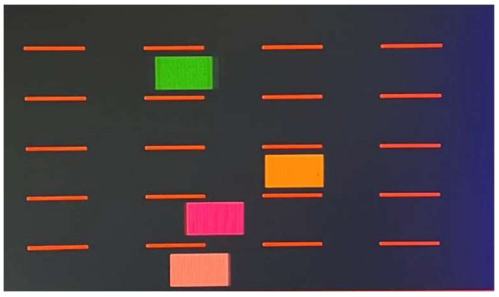

Figura 2.9. Foto demonstrativa carros

Para implementar os troncos é utilizado um retângulo de 16 pixéis de largura e 30 pixéis de comprimento. Tal como os carros não conseguimos implementar as ROM´s e tivemos de optar por este design. O jogo é composto por três troncos que se deslocam tal como os carros, ou seja, movimentam-se na horizontal em sentidos opostos, e também com velocidades diferentes.

```
wldraw : PROCESS (wood1_x, wood1_y, pixel_row, pixel_col) IS
BEGIN
    IF (pixel_col >= wood1_x - sizewood_x) AND
     (pixel_col <= wood1_x + sizewood_x) AND
         (pixel_row >= wood1_y - sizewood_y) AND
         (pixel_row <= wood1_y + sizewood_y) THEN
            wood1_on <= '1';
    ELSE
        wood1_on <= '0';
    END IF;
    END PROCESS;
    -- process to move car1 once every frame (i.e. once every vsync pulse)
    mwood1 : PROCESS
    BEGIN
        WAIT UNTIL rising_edge(v_sync);
        -- allow for bounce off top or bottom of screen
        IF wood1_x + sizewood_x >= 900 THEN
            wood1_x_motion <= "11111111100"; -- -4 pixels
        ELSIF wood1_x <= sizewood_x THEN
            wood1_x_motion <= "00000000100"; -- +4 pixels
        END IF;
        wood1_x <= wood1_x + wood1_x_motion; -- compute next wood1 position
    END PROCESS;

    w2draw : PROCESS (wood2_x, wood2_y, pixel_row, pixel_col) IS
BEGIN
    IF (pixel_col >= wood2_x - sizewood_x) AND
     (pixel_col <= wood2_x + sizewood_x) AND
         (pixel_row >= wood2_y - sizewood_y) AND
         (pixel_row <= wood2_y + sizewood_y) THEN
            wood2_on <= '1';
    ELSE
        wood2_on <= '0';
    END IF;
    END PROCESS;
    -- process to move car1 once every frame (i.e. once every vsync pulse)
```

Figura 2.10. Design e movimentação dos troncos

Figura 2.11. Foto demonstrativa troncos

De modo a tornar o jogo mais realista foram adicionadas barras horizontais, que permitem simula a estrada, um rio, relva e limites superiores e inferiores. A implementação das barras é bastante simples, sendo preciso apenas indicar o tamanho, a cor, e movê-las para o plano posterior, de modo que não tapem a galinha.

```
draw_barras: process(pixel_row, pixel_col)
begin
    -- Barra superior
    if (pixel_row >= BARRA_SUPERIOR_TOP and pixel_row <= BARRA_SUPERIOR_BOTTOM) then
        barra_superior_on <= '1';
    else
        barra_superior_on <= '0';
    end if;

    -- Barra score
    if (pixel_row >= BARRA_SUPERIOR_SCORE_TOP1 and pixel_row <= BARRA_SUPERIOR_SCORE_BOTTOM1) then
        barra_superior_score_on1 <= '1';
    else
        barra_superior_score_on1 <= '0';
    end if;

    if (pixel_row >= BARRA_SUPERIOR_SCORE_TOP2 and pixel_row <= BARRA_SUPERIOR_SCORE_BOTTOM2) then
        barra_superior_score_on2 <= '1';
    else
        barra_superior_score_on2 <= '0';
    end if;

    -- Barra inferior
    if (pixel_row >= BARRA_INFERIOR_TOP and pixel_row <= BARRA_INFERIOR_BOTTOM) then
        barra_inferior_on <= '1';
    else
        barra_inferior_on <= '0';
    end if;
```

Figura 2.12. Excerto de código das barras

Figura 2.13. Foto demonstrativa barras

2.3- Vga_sync

```
BEGIN
    sync_pr : PROCESS
        VARIABLE video_on : STD_LOGIC;
    BEGIN
        WAIT UNTIL rising_edge(pixel_clk);
        -- Generate Horizontal Timing Signals for Video Signal
        -- total horizontal line width = H + H_FP + H_SYNC + H_BP
        -- Reset h_cnt when at end of line
        IF (h_cnt >= H + H_FP + H_SYNC + H_BP - 1) THEN
            h_cnt <= (others => '0');
        ELSE
            h_cnt <= h_cnt + 1;
        END IF;
        -- Pull down hsync after front porch
        IF (h_cnt >= H + H_FP) AND (h_cnt <= H + H_FP + H_SYNC) THEN
            hsync <= '0';
        ELSE
            hsync <= '1';
        END IF;

        IF (v_cnt >= V + V_FP + V_SYNC + V_BP - 1) AND (h_cnt = H + FREQ - 1) THEN
            v_cnt <= (others => '0');
        ELSIF (h_cnt = H + FREQ - 1) THEN
            v_cnt <= v_cnt + 1;
        END IF;
        IF (v_cnt >= V + V_FP) AND (v_cnt <= V + V_FP + V_SYNC) THEN
            vsync <= '0';
        ELSE
            vsync <= '1';
        END IF;
        -- Generate Video Signals and Pixel Address
        IF (h_cnt < H) AND (v_cnt < V) THEN
            video_on := '1';
        ELSE
            video_on := '0';
        END IF;
        pixel_col <= h_cnt;
        pixel_row <= v_cnt;
        -- Register video to clock edge and suppress video during blanking and sync periods
        red_out   <= red_in AND video_on;
        green_out <= green_in AND video_on;
        blue_out  <= blue_in AND video_on;
    END PROCESS;
END Behavioral;
```

Figura 2.14. Processo de sincronização

- Este bloco permite fazer sincronização vertical e horizontal, para exibir o jogo no monitor.
- É composto por dois contadores, h_cnt e v_cnt, que controlam as posições horizontais e verticais respetivamente.
- Os sinais h_sync e v_sync geram os pulsos de sincronização necessários para o monitor.

2.4- Led_driver

O bloco led_driver é o que permite ao utilizador visualizar nos display's de sete segmentos o nível em que se encontra. Sempre que o utilizador passa um nível o display incrementa e sempre que perde, o display volta a zero. Como estamos a usar 4 bits para guardar o resultado, o display apresenta 8, 9, A, B, C, D, E, F, 10, etc.

```
ARCHITECTURE Behavioral OF leddec IS
SIGNAL data : std_logic_vector(3 downto 0);
BEGIN
    data <= f_data(3 DOWNTO 0) WHEN dig = "00" ELSE

                f_data(7 DOWNTO 4) WHEN dig = "01" ELSE

                f_data(11 DOWNTO 8) WHEN dig = "10" ELSE

                f_data(15 DOWNTO 12);
    -- Turn on segments corresponding to 4-bit data word
    seg <= "1000000" WHEN data = "0000" ELSE --0
           "1111001" WHEN data = "0001" ELSE --1
           "0100100" WHEN data = "0010" ELSE --2
           "0110000" WHEN data = "0011" ELSE --3
           "0011001" WHEN data = "0100" ELSE --4
           "0010010" WHEN data = "0101" ELSE --5
           "0000010" WHEN data = "0110" ELSE --6
           "1111000" WHEN data = "0111" ELSE --7
           "0000000" WHEN data = "1000" ELSE --8
           "0010000" WHEN data = "1001" ELSE --9
           "0001000" WHEN data = "1010" ELSE --A
           "0000011" WHEN data = "1011" ELSE --B
           "1000110" WHEN data = "1100" ELSE --C
           "0100001" WHEN data = "1101" ELSE --D
           "0000110" WHEN data = "1110" ELSE --E
           "0001110" WHEN data = "1111" ELSE --F
           "1111111";
--         -- por resolver

    -- Turn on anode of 7-segment display addressed by 2-bit digit selector dig
    anode <= "1110" WHEN dig = "00" ELSE --0
             "1101" WHEN dig = "01" ELSE --1
             "1011" WHEN dig = "10" ELSE --2
             "0111" WHEN dig = "11" ELSE --3
             "1111";

END Behavioral;
```

Figura 2.15. Código display 7 segmentos

Figura 2.16. Display nível 2

2.5- Basys3_master

O ficheiro de constraint basys3_master permite configurar os pinos de entrada e saída da FPGA, de acordo com os módulos implementados. Neste caso utilizamos:

- Sinal de Relógio
- Display de 7 segmentos
- Botões
- Configuração dos Pinos VGA

```
## Clock signal
set_property -dict { PACKAGE_PIN W5 IOSTANDARD LVCMOS33 } [get_ports {clk_in}];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk_in}];

##7 Segment Display
set_property -dict {PACKAGE_PIN W7 IOSTANDARD LVCMOS33} [get_ports {seg[0]}]
set_property -dict {PACKAGE_PIN W6 IOSTANDARD LVCMOS33} [get_ports {seg[1]}]
set_property -dict {PACKAGE_PIN U8 IOSTANDARD LVCMOS33} [get_ports {seg[2]}]
set_property -dict {PACKAGE_PIN V8 IOSTANDARD LVCMOS33} [get_ports {seg[3]}]
set_property -dict {PACKAGE_PIN U5 IOSTANDARD LVCMOS33} [get_ports {seg[4]}]
set_property -dict {PACKAGE_PIN V5 IOSTANDARD LVCMOS33} [get_ports {seg[5]}]
set_property -dict {PACKAGE_PIN U7 IOSTANDARD LVCMOS33} [get_ports {seg[6]}]

set_property -dict {PACKAGE_PIN U2 IOSTANDARD LVCMOS33} [get_ports {anode[0]}]
set_property -dict {PACKAGE_PIN U4 IOSTANDARD LVCMOS33} [get_ports {anode[1]}]
set_property -dict {PACKAGE_PIN V4 IOSTANDARD LVCMOS33} [get_ports {anode[2]}]
set_property -dict {PACKAGE_PIN W4 IOSTANDARD LVCMOS33} [get_ports {anode[3]}]

##Buttons
set_property -dict { PACKAGE_PIN W19 IOSTANDARD LVCMOS33 } [get_ports { b_left }];
set_property -dict { PACKAGE_PIN T17 IOSTANDARD LVCMOS33 } [get_ports { b_right }];
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { b_up }];
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { b_down }];
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { b_reset }];

##VGA Connector
set_property -dict { PACKAGE_PIN G19    IOSTANDARD LVCMOS33 } [get_ports { vga_red[0] }];
set_property -dict { PACKAGE_PIN H19    IOSTANDARD LVCMOS33 } [get_ports { vga_red[1] }];
set_property -dict { PACKAGE_PIN J19    IOSTANDARD LVCMOS33 } [get_ports { vga_red[2] }];
set_property -dict { PACKAGE_PIN N19    IOSTANDARD LVCMOS33 } [get_ports { vga_red[3] }];

set_property -dict { PACKAGE_PIN J17    IOSTANDARD LVCMOS33 } [get_ports { vga_green[0] }];
set_property -dict { PACKAGE_PIN H17    IOSTANDARD LVCMOS33 } [get_ports { vga_green[1] }];
set_property -dict { PACKAGE_PIN G17    IOSTANDARD LVCMOS33 } [get_ports { vga_green[2] }];
set_property -dict { PACKAGE_PIN D17    IOSTANDARD LVCMOS33 } [get_ports { vga_green[3] }];

set_property -dict { PACKAGE_PIN N18    IOSTANDARD LVCMOS33 } [get_ports { vga_blue[0] }];
set_property -dict { PACKAGE_PIN L18    IOSTANDARD LVCMOS33 } [get_ports { vga_blue[1] }];
set_property -dict { PACKAGE_PIN K18    IOSTANDARD LVCMOS33 } [get_ports { vga_blue[2] }];
set_property -dict { PACKAGE_PIN J18    IOSTANDARD LVCMOS33 } [get_ports { vga_blue[3] }];

set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS33 } [get_ports { vga_hsync }];
set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports { vga_vsync }];
```

Basys 3

Para além das configurações habituais como o sinal de relógio e o SPI, é de realçar a integração dos 7 pinos do display de 7 segmentos, também como a definição dos botões que permitem subir, descer e deslocar a galinha para a direita e esquerda.

Em relação aos pinos da VGA, foram atribuídos 4 pinos para cada cor (red, green e blue).

Figura 2.17. Configuração pinos

4. RESULTADOS

Os resultados obtidos foram próximos do que pretendíamos com algumas exceções.

Inicialmente pretendíamos colocar as ROM's da galinha, do carro e dos troncos. Ao tentar-mos implementar a ROM da galinha, deparamo-nos que não iríamos fazê-lo sem comprometer o funcionamento do jogo. Desta maneira, achámos por bem focarmo-nos no funcionamento do jogo, neste caso no movimento dos objetos e nas suas colisões. Tal como referido em capítulos anteriores as ROM'S foram substituídas por quadrados ou retângulos, não comprometendo assim, o funcionamento do jogo.

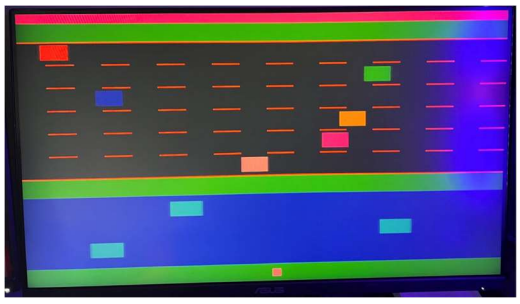

Figura 3.1 Ecrã de jogo

Figura 4.2 Nível de jogo

5. CONCLUSIONS

Este projeto ajudou-nos a consolidar os conhecimentos relativos ao funcionamento da interface VGA bem como a revisão de todos os conhecimentos adquiridos ao longo do semestre.
Tendo mais tempo para o desenvolvimento, a principal ideia que gostaríamos de ter aplicado seria conseguir implementar as ROM's.
No geral consideramo-nos satisfeitos com o resultado obtido, embora graficamente não fosse o idealizado.
No tópico abaixo encontra-se o link do nosso repositório deste projeto.

Agradecimentos: A principal contribuição que podemos destacar foi a da professora Mónica Figueiredo, que nos ajudou no desenvolvimento do projeto, quer em aulas, quer em documentação essencial, como o jogo PONG e Frogger.

REFERÊNCIAS

Bibliografia

Figueiredo, M. (29 de 11 de 2023). *Moodle.* Obtido de PONG game Sources: https://ead.ipleiria.pt/2024-25/mod/resource/view.php?id=68645

Figueiredo, M. (28 de 11 de 2023). *Moodle.* Obtido de Aula TP9 - Jogo do PONG: https://ead.ipleiria.pt/2024-25/mod/resource/view.php?id=68317

Figueiredo, M. (24 de 11 de 2024). *Moodle.* Obtido de Previous Projects: https://ead.ipleiria.pt/2024-25/mod/resource/view.php?id=165102

PONG, P. C. (s.d.). *FPGA Prototyping by CHDL examples.* WILEY.

Project, F. (s.d.). *Github.* Obtido de Github: https://github.com/alex-waldron/CPE487/tree/main/FinalProject

Santos, J. D. (s.d.). *Pacasa.* Obtido de Github: https://github.com/JDomingos1/pacasa