Licenciatura em Engenharia Eletrónica e Telecomunicações e Computadores

instituto superior de engenharia de lisboa

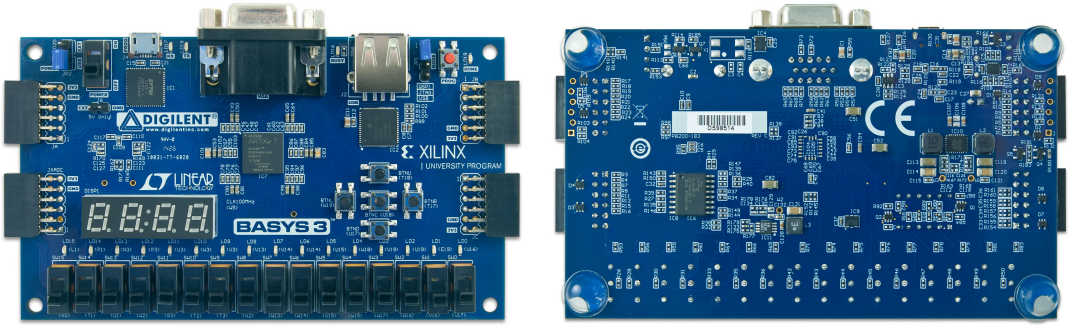


Figura 1

RELATÓRIO DO 4º TRABALHO
Interface VGA controlada por FPGA

Trabalho realizado por:

Aderlaine Lima - 51017 | Afonso Ferreira - 49340 | Miguel Rocha – 51404

Docente: José F. da Rocha

SISTEMAS ELETRÓNICOS ANALÓGICOS E DIGITAIS PROGRAMÁVEIS

2025 / 2026 inverno

ISEL

Conteúdo

Interface VGA controlada por FPGA

1 Trabalho

No âmbito de FPGA-VGA-Jogos, o circuito a desenvolver poderá ser: (i) um jogo de "pong" com dois jogadores ou (ii) um jogo de "pong" com um jogador que destrói tijolos de um muro; (iii) ou um jogo do "pong" com bolas a cair na vertical e apanhar com uma/duas raquete no fundo do ecrã; ou (iv) ou um jogo "do galo" também conhecido como "três em linha" (ou jogo "quatro em linha"). Deverá usar como interface de comando os botões/interruptores disponíveis no kit de desenvolvimento ou o teclado PS2. Usar para interface de visualização um monitor CRT externo ligado à interface VGA do kit. Sugere-se a implementação de níveis de dificuldade e de pontuação.

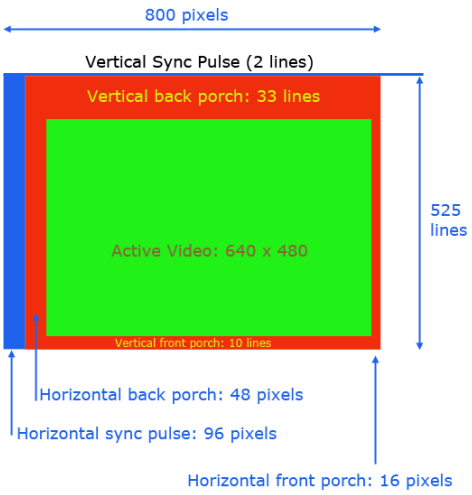
2 Introdução

Durante o percurso deste trabalho nós fizemos várias escolhas que serão mais desenvolvidas em frente.
Através da implementação de um pequeno circuito de aplicação, baseado num jogo clássico — como Pong ou Jogo do Galo — será possível consolidar os conceitos fundamentais de temporização, controlo de sinais VGA, sincronização horizontal e vertical, e gestão de entradas digitais provenientes de botões, interruptores. Nós optamos ainda por fazer um jogo mais complexo com um layout para aceder aos jogos ou às definições.

Interface VGA controlada por FPGA

3 Video Graphics Array

Para implementar a componente gráfica deste trabalho nós fizemos um pouco de investigação para melhor entender o próximo passo e descobrimos que VGA consiste em pins de cor (RGB) e pins de sincronização.



Com VGA para demonstrar, por exemplo uma resolução de 640 x 480 na verdade precisamos de uma resolução de 800 x 525, pois existem diferentes zonas de imagem, como a zona de video ativa onde ficam os 640 x 480 e outra zona para sync-up que é onde vamos sincronizar a imagem e passar informação em cada linha para minimizar distorções.

Figura 2

Para este trabalho optamos por uma resolução igual de 640 x 480, não escolhemos maior pois não sabemos a capacidade total da nossa FPGA para este funcionamento, e os jogos são mais simples então não haveria necessidade.

Na fotografia ao lado podemos ver uma timing window da informação a passar onde se nota a utilidade dos front e back porch, que são zonas onde preparamos a sincronização vertical, estes ajudam a manter o sinal, limpo e sem erros de sincronização.Sendo que nós vamos mexer um valor total de 25 Milhões de pixels por segundo (800 * 525 * 60FPS), não é necessário atualizar com um clock maior que 25MHz.

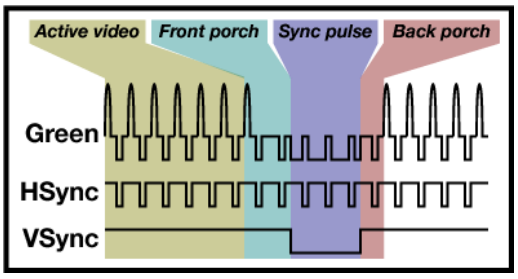


Figura 3

3.1 Pixel Management - VHDL

Para converter a teoria do VGA para o Vivado nós fizemos uma source VGA_Signal que vai gerir a informação, vai dizer aos programas quando é que estes podem passar video e exatamente onde se encontra o pixel a mudar.

```
process(clk_100Meg, reset, H_Count_D, ctick_25MHz, V_Count_D)
begin
    if reset = '1' then
        V_Count_D <= (others => '0');
    elsif rising_edge(clk_100Meg) then
        if ctick_25MHz = '1' then
            if H_Count_D >= (HT - 1) then
                if V_Count_D >= (VT - 1) then
                    V_Count_D <= (others => '0');
                else
                    V_Count_D <= V_Count_D + 1;
                end if;
            end if;
        end if;
    end if;
end process;
```

Listing 1: VGA Signal Variables

Para passar as imagens precisamos de saber em que pixel estamos para escolhermos o certo, e criar uma imagem. Então geramos contadores tanto para saber as coordenadas dos pixeis, tendo o cuidado de no Y ver se já percorremos uma linha Horizontal.Isto porque nós preenchemos os pixels da esquerda para a direita e depois descemos um.

Nesta secção nós encontramos as secção de Sync com as delimitações conhecidas. Para depois mais em frente sabermos em que secção passar o Desenho.

```
process(H_Count_Q) -- Enable H_Sync--
--Se estivermos dentro dos limites de sincronização
begin
    if (H_Count_Q >= (DH + HBP)) and (H_Count_Q <= (DH + HBP
    ↪  + HSP - 1)) then --maior que DH + HBP e menor que DH +
    ↪  HBP + HSP - 1
        H_Sync_D <= '0';
    else
        H_Sync_D <= '1';
    end if;
end process;

process(V_Count_Q) -- Enable V_Sync--
--Se estivermos dentro dos limites de sincronização
begin
    if (V_Count_Q >= (DV + VBP)) and (V_Count_Q <= (DV + VBP
    ↪  + VSP - 1)) then
        V_Sync_D <= '0';
    else
        V_Sync_D <= '1';
    end if;
end process;
```

Listing 2: VGA Enable Sync

```
process(V_Count_Q, H_Count_Q) -- Enable_out--
--Se estivermos dentro dos limites do visivel
begin
    if (V_Count_Q < (DV)) and (H_Count_Q < (DH)) then
        enable_out <= '1';
    else
        enable_out <= '0';
    end if;
end process;
```

Listing 3: VGA Enable Video

Para finalizar este programa nós delimitamos onde podemos passar video. Desta maneira não transmitimos informação desnecessária para fora e vice-versa.

4 Video Generation

Esta secção do relatório serve para demonstrar ao leitor, os passos necessários para projetar a imagem no monitor. Depois de termos as coordenadas do pixel, é uma questão de atribuir a cada pixel uma cor. Aqui em baixo tem um exemplo mais complexo, para a demonstração de um padrão.
Para pintar, nós utilizamos o estilo de blocos 16x16, como nos jogos antigos, em baixo vê-se o design do tijolo que nós fizemose e logo á direita o equivalente em VHDL.

Figura 4

```
constant BRICK : brick_column := (

    -- Row 0: 5 light, 1 dark red, 1 light, 9 brick
    (  BRICK_ORANGE, BRICK_ORANGE, BRICK_ORANGE, BRICK_ORANGE,
    ↪  BRICK_ORANGE,
       BRICK_BROWN,
       BRICK_ORANGE,
       BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED,
      ↪  BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED ),

    -- Row 1: 4 brick, 1 shade, 1 dark, 2 light, 8 brick
    (  BRICK_BROWN, BRICK_RED, BRICK_RED, BRICK_RED,
       BRICK_ORANGE_DARKER,
       BRICK_BROWN,
       BRICK_ORANGE, BRICK_ORANGE,
       BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED,
      ↪  BRICK_RED, BRICK_RED, BRICK_RED ),

    -- Row 2
    (  BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED,
       BRICK_ORANGE_DARKER,
       BRICK_BROWN,
       BRICK_ORANGE,
       BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED,
      ↪  BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED ),

    -- Row 3 (same as row 2)
    (  BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED,
       BRICK_ORANGE_DARKER,
       BRICK_BROWN,
       BRICK_ORANGE,
       BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED,
      ↪  BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED ),

    -- Row 4 (same as row 2)
    (  BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED,
       BRICK_ORANGE_DARKER,
       BRICK_BROWN,
       BRICK_ORANGE,
       BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED,
      ↪  BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED ),

    -- Row 5: 3 brick, 2 shade, 1 dark, 1 light, 9 brick
    (  BRICK_RED, BRICK_RED, BRICK_RED,
       BRICK_ORANGE_DARKER, BRICK_ORANGE_DARKER,
       BRICK_BROWN,
       BRICK_ORANGE,
       BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED,
      ↪  BRICK_RED, BRICK_RED, BRICK_RED, BRICK_RED ),

    -- E por aí a diante....
);
```

Listing 4: Brick.vhd

```
brick_x_n <= to_integer(unsigned(X)) mod 16;
brick_y_n <= to_integer(unsigned(Y)) mod 16;
...
RGB_game <= BRICK(brick_y_n)(brick_x_n);
```

Listing 5: Brick Counter

Para ser possivél passar a variável brick para a saída VGA, com umas simples contas matemáticas chega-se lá, com o módulo das nossas coordenadas com o valor 16, conseguimos retirar coordenadas da matriz do tijolo.

5 Game

O nosso Front Panel consegue ser resumido á maquina de estados demonstrada no diagrama em cima. Nós tivémos a ideia de criar o estado Release, que pode parecer simples, mas é o elemento mais importante. O Release parece simplesmente um estado que verifica se o utilizador retirou o dedo do botão.Mas é preferivél compará-lo a um terminal de autocarros, o estado Release vai estar sempre atento a uma variável "Menu_sel", variável esta que é usada por Release para redirecionar a FPGA para o estado seguinte.

5.1 Front Panel ASM

Para o estado Main, damos ao utilizador duas escolhas, Game Select e Créditos. E depois repetimos a estrutura do estado main para dois outros estados (Game Select, Pong Difficulty). O estado Créditos serve apenas para demonstrar os nossos nomes,Os estados 'ocultos', pong e TicTacToe, ocultos pois neste estado, o Front Panel não passa video.

5.2 Pong

Para o jogo Pong, a implementação foi baseada em máquinas de estado finitas, respeitando a lógica de funcionamento em hardware. O jogo foi dividido essencialmente em duas FSM independentes: uma para o controlo da raquete e outra para a lógica da bola.

A FSM da raquete possui apenas dois estados (Main e Release). No estado Main é feita a leitura dos botões de movimento (esquerda e direita), atualizando a posição horizontal da raquete dentro dos limites do ecrã. O estado Release garante que o utilizador largou o botão antes de aceitar um novo comando, evitando múltiplos movimentos causados pelo bounce dos botões.

A FSM da bola é composta por quatro estados (Spawn, Falling, Check e GameOver). No estado Spawn a bola é posicionada no topo do ecrã com uma coordenada horizontal variável, obtida a partir de um contador pseudo-aleatório já existente no projeto. No estado Falling a bola desloca-se verticalmente em direção à base do ecrã. Ao atingir a zona da raquete, o estado Check verifica se existe colisão horizontal; caso exista, uma nova bola é gerada, caso contrário o jogo entra no estado GameOver. Este último estado mantém o resultado até ocorrer um reset ou a troca de jogo no menu.

5.3 Tic-Tac-Toe

Para o TicTacToe o jogo até foi relativamente simples, com apenas 3 estados(Main, Release e Game_end), é possivel ter o jogo funcional, e até com dois seria possível. Entrando nos estados temos Main que não é nada mais que um gestor de movimentos,que recorda os movimentos do utilizador e com assistência de dois array (x,y) temos um guia para saber onde o cursor do utilizador se encontra, temos mais uma vez o Release que nos vai ajudar a ver se o utilizador largou o botão, sendo que tambem é neste estado que é verificado se o jogo já acabou,o ultimo estado Game_end serve para ajudar na demonstração em video, sendo que pode ser retirado.

Texto na ROM

6 ROM

A nossa FPGA tem a funcionalidade de guardar memória em ROM, o que é extremamente útil para nós, para dár display de Letras, números e símbolos.No nosso trabalho temos multiplas frases que temos de escrever.E para não termos o trabalho mal ótimizado nós usufruir-mos da memória ROM, temos uma função na memória que retorna '1' ou '0'.

6.1 A Função

Nós simplesmente retornamos '1' ou '0', pois não é preciso de muito mais, com '0' a indicar que não tem nada e 1 que contêm o formato da letra.
Este formato é muito semelhante ao do Brick em que temos uma matriz de '0' ou '1' sendo que no '1' põe-se uma única côr que nos irá retornar o formato da letra.

7 Sources

- Figura 1 — fonte: https://digilent.com/reference/programmable-logic/basys-3/reference-manual
- Figura 2 — fonte: https://electronics.stackexchange.com/questions/295130/vga-timing-sync-porch-positions-fpga
- Figura 3 — fonte: https://web.mit.edu/6.111/www/s2004/NEWKIT/vga.shtml
- Figura 4 — fonte: Feito por nós.