Utilização da linguagem Handel-C na criação e implementação de um Gap Puzzle

Bruno Pimentel

***Resumo* – Este artigo descreve os desafios encontrados e as estratégias utilizadas no desenvolvimento de um programa que consiste num Gap Puzzle.**

Quando o programa inicia, a imagem utilizada no puzzle é transferida da FlashRAM para o banco de memória SSRAM externa à FPGA. O mapa do puzzle é baralhado e iniciam-se os processos concorrentes e cíclicos que suportam a funcionalidade necessária ao Gap Puzzle.

Criou-se uma hierarquia de camadas que permite identificar o elemento gráfico em foco, do qual se extrai o valor de pixel que se pretende projectar. Com a imagem de puzzle armazenada na SSRAM, recorrendo a um pipeline, é possível projectar em cada ciclo de relógio o valor de pixel extraído no ciclo anterior. Optou-se por um mapeamento referencial das peças do puzzle, para suportar os deslocamentos das mesmas. O ponteiro do rato pode tomar várias formas armazenadas na ROM com valores que correspondem a cores.

Para adaptar o programa às especificidades do hardware utilizado, procurou-se: minimizar o número de recursos da FPGA utilizados, utilizar expressões Handel-C partilhadas e evitar operações longas nas zonas críticas do programa.

***Abstract* – This article describes the challenges found and strategies used in the development of a program that consists of a Gap Puzzle. The program was created to run on the Celoxica RC100 board, using the Handel-C language in the DK1 environment.**

When the program starts, the puzzle image is transferred from the FlashRAM to the SSRAM external to the FPGA. The puzzle map is mixed up and a few cyclic and concurrent processes are launched to support the required functionality.

A layer hierarchy was created to allow the identification of the focused graphic item, from which the needed pixel value is read. With the puzzle image stored in the SSRAM, it is possible to send to the monitor screen, in each clock cycle, a pixel value read in the previous cycle (using a pipeline for such a purpose). A referential mapping of the puzzle pieces was implemented to support their displacement. The mouse pointer can be shaped into different figures, stored in the ROM, using values that correspond to colors.

To properly adapt the program to the hardware characteristics, the following strategies were enforced: minizing the number of FPGA resources, using Handel-C shared expressions and avoiding long operations in the program critical areas.

I. INTRODUÇÃO

A Computação Reconfigurável é particularmente importante no âmbito do curso de Computadores e Telemática na medida em que permite apreender os requisitos essenciais para a criação de soluções em *hardware* reconfigurável. Neste sentido, importa perguntar: como modelar a solução, tendo em conta as características do *hardware* utilizado? que cuidados ter para suportar interacção com periféricos de entrada e de saída? que estratégias utilizar para rentabilizar os recursos disponíveis na FPGA?

Para a realização deste projecto, recorreu-se, sobretudo, aos conhecimentos transmitidos pelo professor da disciplina e à pesquisa bibliográfica.

De entre as linguagens utilizadas na especificação de sistemas digitais, optou-se pelo Handel-C por ser de alto nível. O programa foi desenvolvido no ambiente DK1 e destinado a correr na placa RC100 da Celoxica. No site da Celoxica [1], encontra-se informação detalhada acerca da linguagem, do *software* e da placa utilizados.

A. Objectivos

O presente trabalho visa, fundamentalmente, os seguintes objectivos:

- aprender a criar um modelo adequado à solução pretendida e ao *hardware*;
- responder com sucesso à especificidade dos dispositivos periféricos seguintes: teclado, rato e monitor VGA;
- interiorizar técnicas de optimização e de reutilização dos recursos da FPGA;
- aprender a utilizar os *displays* de sete segmentos, a FlashRAM, a SSRAM e a ROM.

B. Descrição global do trabalho

O programa consiste num Gap Puzzle 4x4, que utiliza uma imagem de 512x480 pixels, com uma profundidade de cor de 24 bits, organizados no modelo RGB, como impõe o *driver* do ecrã. O puzzle é projectado no ecrã e o número de deslocamentos de peças já realizados é apresentado nos *displays* de sete segmentos. Estes *displays* integram a placa RC100, na qual corre o programa. A intervenção do utilizador é feita através de um rato e de um teclado, directamente ligados à placa. A

figura 1 ilustra a interligação das componentes de *hardware* utilizadas.

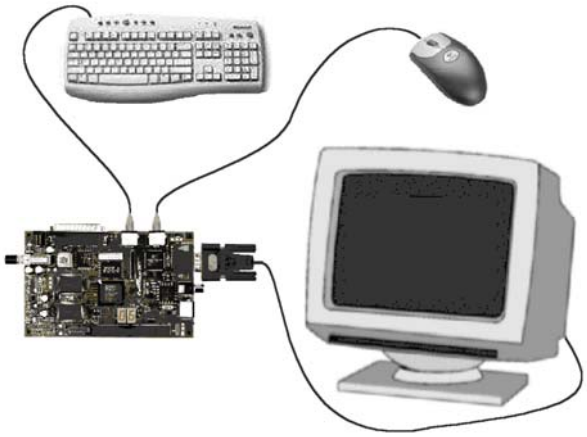

Fig. 1 - Interligação da placa RC100 e periféricos

É possível deslocar as peças, tanto clicando com o botão esquerdo do rato nas peças adjacentes à posição sem peça, como pressionando as setas do *numpad* do teclado. Em qualquer altura se pode reiniciar o puzzle, clicando simultaneamente nos dois botões do rato ou pressionando a tecla *Home* do *numpad*. A tecla *End* do *numpad* permite organizar o puzzle instantaneamente.

II. Organização Básica do Programa

Após o *bitstream* ter sido carregado para a placa, a inicialização de algumas variáveis é efectuada e a imagem é transferida da FlashRAM para a SSRAM externa:

```
par
{  puzzle_complete = 0;
   …
   LoadImage();
}
```

Seguidamente, os *displays* de 7 segmentos são activados e o mapa do puzzle baralhado:

```
randomize_puzzle();
RC100Set7SegEnable(0b11);
```

A partir desta fase, o programa consiste em vários processos cíclicos que permitem:

- controlar o que se projecta no ecrã, em cada instante;
- receber controlos provenientes do rato e do teclado, accionando os procedimentos correspondentes;
- controlar a utilização dos *displays* de 7 segmentos;
- detectar se o puzzle foi terminado;
- controlar, a nível físico, os dispositivos da placa e os periféricos, no caso dos *drivers*.

Estas funcionalidades correm em paralelo, uma vez que têm que ser executadas em simultâneo:

```
par
{  …
   Display(&Mouse, &Video);
   Mouse_clicking(&Mouse);
   Keyboard_pressing(*Keyboard.ReadChannel);
   SevenSegmentDisplaying(sy@sx);
   puzzle_completion();
}
```

A contínua repetição destes processos é feita de uma forma independente para permitir que os processos mais lentos não atrasem o recomeço dos mais rápidos.

III. Modelação e Estratégias Adoptadas

A. Suporte para a disposição gráfica

Este programa utiliza diversos elementos gráficos, como a imagem, a grelha e o buraco do puzzle, o ponteiro do rato e outros, os quais aparecem no ecrã de forma simultânea. Sob o ponto de vista do controlador do ecrã, apenas um pixel é enviado em cada ciclo de relógio e é necessário fornecer-lhe o valor correcto para as coordenadas do ecrã indicadas pelo *driver* de vídeo.

Assim, importa saber a que elemento gráfico essas coordenadas correspondem e, a partir daí, determinar o valor do pixel a projectar nesse ciclo de relógio.

A solução para este problema foi modelada com uma hierarquia em camadas de elementos gráficos (fig. 2).

Estas camadas podem abranger a totalidade da área do ecrã ou apenas parte dela (como o ponteiro do rato ou a grelha de puzzle). Por outro lado, as camadas podem estar activas ou não, dependendo da fase do jogo. Finalmente, cada um dos seus pixels pode encontrar-se opaco ou transparente.

Assim, para cada pixel do ecrã, a camada a ser projectada é a mais elevada das que, simultaneamente, se encontrarem activas, contemplarem as coordenadas respectivas e apresentarem um valor de pixel opaco para as mesmas.

B. Manipulação da imagem do puzzle

Na camada mais baixa da hierarquia de elementos gráficos, encontra-se o elemento principal do programa: a imagem que é previamente gravada na FlashRAM.

Como o acesso à FlashRAM é demasiado lento para ser feito enquanto se utiliza o ecrã, a imagem é passada para a SSRAM, no início. Mesmo assim, não é possível enviar um valor de pixel lido desta memória para ser utilizado pelo *driver* de vídeo, no mesmo ciclo de relógio. Deste modo, há que implementar um *pipeline* que, em cada ciclo, envie para o ecrã o valor de pixel lido no ciclo anterior e leia o valor seguinte para uma variável temporária. Esta solução foi implementada em vários

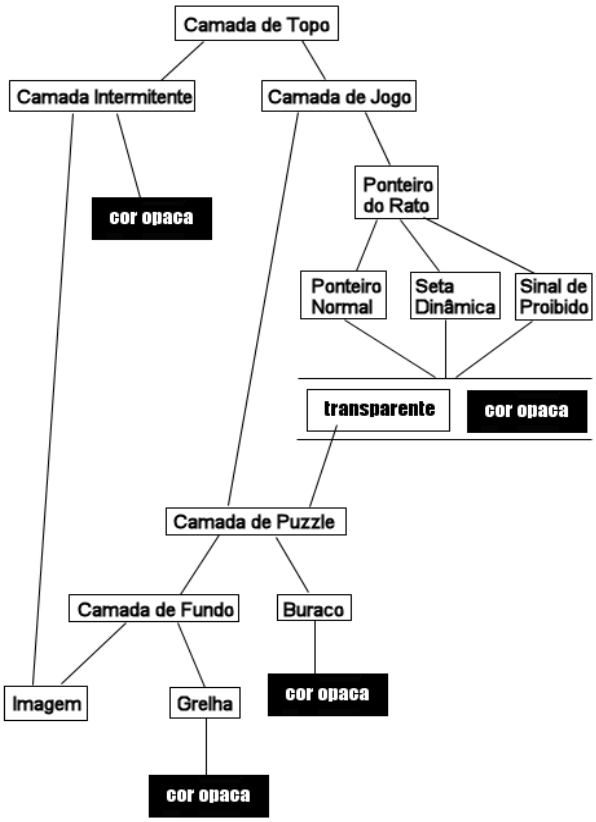

Fig. 2 - Hierarquia em camadas dos elementos gráficos a projectar

exemplos que podem ser encontrados no site da disciplina de Computação Reconfigurável [2].

Adicionalmente, há que tomar em consideração que as diversas parcelas da imagem são projectadas com uma disposição que vai variando com a intervenção do utilizador.

Para suportar esta função, não seria viável comutar, na imagem gravada na SSRAM, cada um dos 120x128 valores de pixel das duas posições em causa, em cada deslocamento. Como solução, optou-se por um mapeamento referencial das parcelas da imagem original, numa estrutura de posições do puzzle.

Em alternativa à utilização de apenas um *array* unidimensional de 16 posições, num endereçamento simples (figura 3), foram criados dois *arrays* bidimensionais 4x4, endereçando, separadamente, a linha e a coluna de cada posição (fig. 4). Isto permitiu tornar a manipulação desta informação mais intuitiva e legível a nível topológico.

No caso de a camada a projectar ser a da imagem, o endereço da SSRAM no qual sita o valor do pixel desejado

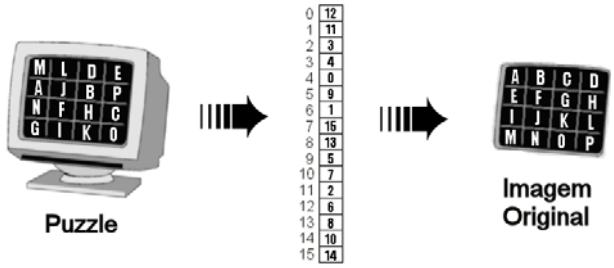

Fig. 3 - Alternativa 1: Endereçamento simples

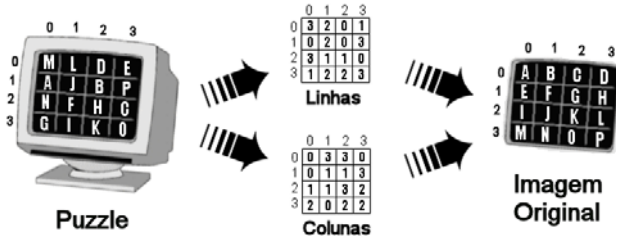

Fig. 4 - Alternativa 2: Endereçamento paralelo de linhas e colunas

depende da disposição das peças do puzzle, nesse momento, e dos deslocamentos vertical e horizontal na peça em causa. Por outro lado, o armazenamento na SSRAM é unidimensional, pelo que o acesso é feito através de uma única coordenada.

Deste modo, é necessário:

- determinar a que parcela da imagem original a peça em causa corresponde;
- adicionar às coordenadas base dessa parcela os mesmos deslocamentos que se verificavam na peça, nos eixos vertical e horizontal, para obter as coordenadas na imagem original;
- converter, por fim, estas coordenadas numa única que servirá para endereçar a SSRAM.

A figura 5 ilustra esta conversão com maior detalhe.

Quando o utilizador consegue completar o puzzle, isto é, quando a disposição das peças corresponde à da imagem original, a camada intermitente é activada. Esta camada consiste, inicialmente, na projecção alternada da imagem original e da cor preta, recorrendo a uma variável

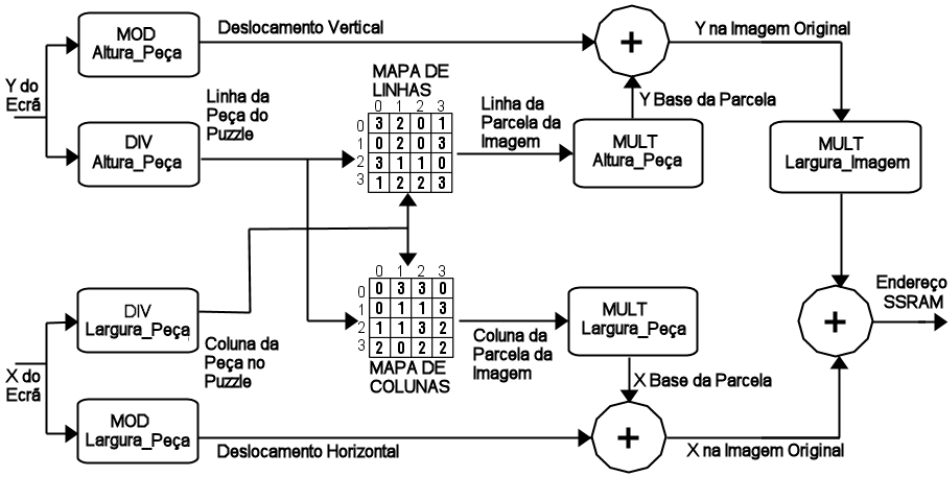

Fig. 5 - Obtenção do endereço SSRAM do valor de pixel a partir das coordenadas do ecrã

contadora. Após esta variável ter sido incrementada algumas vezes, é projectada apenas a imagem original até que o utilizador reinicie o puzzle.

C. Implementação do ponteiro do rato

Relativamente aos ponteiros do rato, como já é habitual, uma vez que utilizam uma gama de cores muito reduzida, os valores dos pixels dos ponteiros são armazenados simbolicamente numa estrutura de dados com um número reduzido de bits. Assim, associando cada número a um valor de pixel, poupa-se espaço e facilita-se a leitura gráfica a partir do código. Uma descrição mais pormenorizada deste tipo de implementação pode ser encontrada em [3, p. 767]. Concretizando, no presente trabalho, foi utilizada uma estrutura de dados de 2 bits para contemplar 4 cores que se fizeram corresponder às cores preta, vermelha, amarela e transparente. A última indica que o pixel não é opaco, passando a camada projectada a ser a subsequente, na hierarquia referida anteriormente.

O programa utiliza 2 ponteiros que são gravados na ROM: um símbolo de proibição e uma seta dinâmica (fig. 6). O primeiro é utilizado quando o ponteiro se encontra sobre uma peça que não pode ser movida; o segundo é utilizado quando esta peça é adjacente à posição sem peça, pelo que pode ser deslocada. Neste último caso, se a peça adjacente não for a que estiver por baixo da posição sem peça, as coordenadas do ponteiro são transformadas, para obter uma seta orientada para baixo, para a esquerda ou para a direita, em vez de uma orientada para cima.

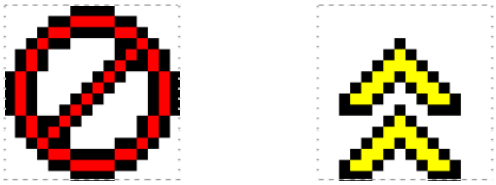

Fig. 6 - Ponteiros do rato: símbolo de proibição e seta dinâmica

Por outro lado, para que a seta não seja estática, à coordenada vertical deste ponteiro somam-se os 2 bits mais significativos de uma variável de 4 bits, cujo valor é incrementado em cada varrimento do ecrã. A adição destes 2 bits faz com que a cor de cada pixel seja, na realidade, extraída de entre 0 a 3 linhas mais acima, gerando, no ecrã, uma oscilação da seta na direcção em que esta aponta. A seta foi desenhada sem recorrer às 3 linhas superiores, para que a sua ponta não aparecesse no lado oposto.

D. Estratégias gerais

Outras estratégias foram utilizadas para optimizar vários aspectos do programa, nomeadamente, minimizar o número de recursos da FPGA utilizados por cada processo concorrente e evitar a criação de sequências demasiado morosas.

Por um lado, foram utilizadas, na medida do possível, expressões Handel-C partilhadas em alternativa às expressões macro, para que o número de recursos utilizados não se aproximasse do que a FPGA incorpora. Em diversas partes do programa, foi necessário o uso das funções auxiliares de divisão inteira e de resto da divisão inteira. Para que o compilador do programa DK1 não expandisse demasiado estas funções, determinou-se o número máximo de iterações recursivas que iriam ser utilizadas e criaram-se funções próprias que contemplavam, exaustivamente, as várias hipóteses possíveis.

Finalmente, procurou-se incluir apenas operações breves nas zonas críticas do programa, para não comprometer o desempenho dos diferentes processos, nomeadamente, os que dizem respeito ao envio de imagem para o ecrã.

IV. Conclusões

Descreveram-se os desafios encontrados e as estratégias utilizadas no desenvolvimento de um Gap puzzle.

As ferramentas utilizadas foram a linguagem Handel-C, o *software* DK1 e a placa RC100 da Celoxica.

O programa consiste em algumas operações de inicialização, seguidas de um ciclo com as funcionalidades fundamentais.

A projecção dos elementos gráficos do programa assenta numa hierarquia em camadas, para determinar que elemento é projectado em cada situação.

A imagem é transferida para a SSRAM para que, com o auxílio de um *pipeline*, seja possível obter e enviar para o ecrã um valor de pixel em cada ciclo de relógio.

Utilizou-se um mapeamento referencial de colunas e linhas, separadamente, para suportar os deslocamentos das peças de puzzle.

Procurou-se minimizar a utilização de recursos da FPGA, utilizar expressões Handel-C partilhadas e evitar operações longas, para optimizar o funcionamento do programa.

Agradecimentos

Agradeço ao Prof. Valery Sklyarov e Iouliia Skliarova pelas orientações e estímulo que muito contribuíram para a realização deste estudo e marcarão o meu futuro.

Referências

[1] URL: *http://www.celoxica.com*.

[2] URL: *http://webct.ua.pt*, 2º Semestre, disciplina Computação Reconfigurável.

[3] Pedro Almeida, Manuel Almeida. Desenvolvimento de um circuito aritmético a partir da sua especificação em Handel-C. Electrónica e Telecomunicações, Vol. 3, nº8, Janeiro 2003, pp. 765-769.