FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Módulos reconfiguráveis dinamicamente para processamento de imagens vídeo

Bruno Falcão Dantas

Tese submetida no Âmbito do

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Major de Telecomunicações

Orientador: Prof. Hélio Mendonça (Doutor)

Março de 2009

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES 2007/2008

A Dissertação intitulada

"MÓDULOS RECONFIGURÁVEIS DINAMICAMENTE PARA PROCESSAMENTO DE IMAGENS VÍDEO"

foi aprovada em provas realizadas em 27/Fevereiro/2009

o júri

Presidente Professor Doutor Pedro Henrique Henriques Guedes de Oliveira
Professor Catedrático da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar Convidado da Universidade de Aveiro

Professor Doutor Hélio Mendes de Sousa Mendonça
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.

Autor - BRUNO FALCÃO DANTAS

Bruno Falcão Dantas

Faculdade de Engenharia da Universidade do Porto

Resumo

Este documento apresenta um trabalho que propõe tirar proveito do crescente aumento de densidade lógica e da capacidade de adaptação dinâmica do hardware dos dispositivos reconfiguráveis (FPGAs), para o uso em processamento de imagem em tempo real, tarefa que requer um elevado recurso computacional. Este trabalho enumera a criação de uma biblioteca de módulos que são reconfiguráveis em tempo de execução na FPGA, permitindo assim, uma adaptação da segmentação de vídeo aos vários ambientes previstos, ao qual a captura pode se submeter. Estes módulos reconfiguráveis, são filtros baseados em diversas topologias, desde a teoria da convolução, morfologia matemática e histogramas das imagens. Filtros que também são configuráveis através de valores enviados pela porta série, a partir de um interface instalado num computador. É descrito também, o desenvolvimento de um módulo que permite o uso de memórias inseridas no dispositivo onde se encontra a FPGA, para guardar ou reproduzir imagens da cadeia de processamento. Imagens essas na memória que a interface no computador consegue aceder, permitindo a recolha e inserção de imagens processadas. Este documento mostra ainda, o processo de implementação num dispositivo que permite a reconfiguração dinâmica dos módulos que fazem parte da biblioteca.

Abstract

This document presents a work that proposes to take the most of the increasing logic density and hardware's dynamic adapting ability of reconfigurable devices (FPGA's) to apply it to real-time based image processing. This task requires high computational resources. This piece of work list the creation of a module's library, which will be reconfigurable at FPGA's execution time, thus allowing the adaptation of video segmentation to various predicted environments and to which capture may be submitted. These reconfigurations modules are filters based on several topologies, like the convolution theory, mathematical morphology and image histograms. These filters are configurable as well through data sent by serial port, with an interface on a computer. It also describes the development of a module that allows the use of embedded memory on the FPGA board, to save and reproduce images of the chain of processing. Using the interface, the computer can access the images on the memory, allowing the collection and integration of image for the processing. This document also shows the process of modules implementation, which are part of the library, on a device that allows dynamic reconfiguration.

Agradecimentos

Gostaria de agradecer a todas as pessoas que contribuíram directa ou indirectamente na realização deste trabalho, e pelas quais guardo uma grande estima. Por uma contribuição mais directa, fica o meu agradecimento especial ao meu orientador, professor Hélio Mendonça, aos professores José Carlos Alves, João Canas Ferreira e aos colegas de trabalho Bruno Monteiro e Francisco Basadre.

Quero também deixar o meu enorme agradecimento ao meus pais, Joaquim Dantas e Maria Falcão, pelo apoio e paciência durante toda a realização do curso.

Por fim, à minha noiva Patrícia Andrade, pela compreensão e apoio devido à minha permanente ausência nestes últimos tempos.

Bruno Dantas

"Progress, of the best kind, is comparatively slow. Great results cannot be achieved at once; and we must be satisfied to advance in life as we walk, step by step."

Samuel Smiles

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ASIC	*Aplication Specific Integrated Circuit*
CLB	*Configurable Logic Block*
CPU	*Central Processing Unit*
DCM	*Digital Clock Manager*
DRL	*Dynamically Reconfigurable Logic*
DSP	*Digital Signal Processing*
FIFO	*First In, First Out*
FIR	*Finite Impulse Response*
FPGA	*Field-Programmable Gate Array*
IIR	*Infinite Impulse Response*
IOB	*Input/Output Block*
JTAG	*Joint Test Action Group*
LVTTL	*Low Voltage Transistor to Transistor Logic*
PR	*Partial Reconfiguration*
RGB	*Red Green Blue*
RTR	*Run Time Reconfiguration*
SDRAM	*Synchronous Dynamic Random Access Memory*
VDAC	*Video Digital to Analog Converter*
VHDL	*VHSIC Hardware description Language*
VHSIC	*Very-High-Speed Integrated Circuit*

Capítulo 1

Introdução

1.1 Âmbito do trabalho

Todo o conceito deste trabalho está inserido no âmbito da área de microelectrónica e sistemas embutidos e pretende-se com ele, colaborar num projecto de investigação que tem como tema "Reconfiguração Dinâmica de Protocolos de Comunicação" [3]. Este projecto procura abordar a reconfiguração dinâmica de dispositivos lógicos DRL para a implementação de aplicações complexas multi-tarefa com requisitos de execução em tempo real. Explora a adaptação dinâmica do *hardware*, tendo em conta as necessidades específicas de uma aplicação durante a sua execução, estas abordagens conduzem a benefícios tais como desempenho, custo, qualidade dos resultados e consumo. Este projecto tenta também explorar o problema da segmentação [4] "foreground/background" de vídeo em tempo real com aplicações em codificação de vídeo e videovigilância.

1.2 Objectivos do trabalho

Actualmente o sistema de processamento de imagens é uma tarefa que exige elevados recursos computacionais e é normalmente realizado por microcomputadores, onde com a utilização de softwares específicos as imagens podem ser processadas. O objectivo deste tema é tirar partido do crescente aumento da densidade lógica dos dispositivos reconfiguráveis FPGAs, para explorar a implementação de um sistema idêntico em *hardware* dedicado. Pretende-se então, numa primeira fase projectar uma biblioteca de módulos de *hardware* destinados ao processamento de imagens de vídeo em tempo real. Numa segunda fase, vai ser explorada a reconfiguração dinâmica desses módulos na FPGA, permitindo assim uma adaptação dinâmica do módulo a implementar perante uma alteração das condições.

1.3 Estrutura da Dissertação

Para além da introdução, esta dissertação contêm mais 6 capítulos. No capítulo 2, é descrito o estado da arte onde são apresentados estudos relacionados com o projecto. O capítulo 3, fala do projecto em si e apresenta para além dos filtros que foram criados para a biblioteca de módulos, um módulo que ajudou a mostrar resultados do trabalho. É também apresentada uma solução para testar uma cadeia de filtros. No final do capitulo são apresentadas algumas imagens que são resultado do projecto. O capítulo 4, apresenta o programa de interface que foi criado para comunicar com os módulos, e as alterações necessárias no projecto para permitir o mesmo. O capítulo 5, fala na migração do projecto para outra placa e na adaptação deste para o suporte à reconfiguração dinâmica. Como conclusão o capítulo 6 mostra as vantagens e desvantagens do funcionamento deste projecto e fala no possível futuro trabalho neste âmbito.

Capítulo 2

Estado de arte

Este capítulo apresentará um pequeno estudo sobre temas que foram fundamentais para o desenvolvimento do projecto que é proposto.

2.1 Processamento de imagem

A área de processamento digital de imagens tem atraído grande interesse nas últimas duas décadas e surgiu, principalmente, da necessidade de melhorar a qualidade da informação gráfica para interpretação humana. Agora, o objectivo do seu uso consiste em melhorar o aspecto visual de certas feições estruturais para o analista humano e fornecer outras perspectivas da sua interpretação para assim gerar produtos que possam ser posteriormente submetidos a outros processamentos. A evolução da tecnologia de computação digital, com métodos capazes de realçar as informações contidas nas imagens para a posterior interpretação e análise humana, bem como o desenvolvimento de novos algoritmos para lidar com sinais bidimensionais permite uma gama de aplicações cada vez maior. Como resultado dessa evolução, a tecnologia de processamento digital de imagens vem ampliando os seus domínios, que incluem as mais diversas áreas.

2.2 Representação de imagens Digitais

O termo imagem monocromática, ou simplesmente imagem, refere-se à função bidimensional de intensidade da luz f(x,y), onde x e y representam as coordenadas espaciais e o valor f em qualquer ponto (x, y) representa um valor proporcional ao brilho ou nível de tons de cinza. Uma imagem digital pode ser considerada como sendo uma matriz cujos índices de linhas e de colunas identificam um ponto na imagem, e o correspondente valor do elemento da matriz identifica o nível de cinza, ou brilho naquele ponto. Cada um destes elementos é designado por pixel que é uma abreviação de "*picture element*" (elemento de figura). Quanto mais píxeis uma imagem tiver melhor é a sua resolução e qualidade.

Figura 2.1: Convenção dos eixos para representação de imagens digitais.

2.3 Segmentação de Vídeo

A segmentação de imagens ou vídeos, [4, 5, 6] refere-se à identificação de regiões homogéneas na imagem. Considera uma imagem ou vídeo numa partição de regiões não sobrepostas, numa espécie de camadas, cuja união resulta na imagem inteira. A segmentação de imagem é frequentemente descrita como o processo que subdivide uma imagem nas suas partes constituintes e extrai apenas as partes de interesse, designadas por objectos. É o primeiro passo e também uma das funções mais difíceis e críticas na análise das imagens, pois influencia todos os passos subsequentes da análise de imagem.

2.4 Filtros

O processamento digital de sinal, conhecido por DSP, trata-se do processamento feito a um sinal contínuo, neste caso imagens, convertido em sequências de números e símbolos. Este processamento pretende alterar o conteúdo do sinal, neste caso imagem, de forma a retirar ou salientar conteúdos específicos. A figura 2.2 mostra num diagrama de blocos, as operações envolvidas para este tratamento de sinal.

O bloco AD trata-se de uma conversão analógica para digital, neste caso um sinal x(t) numa sequência x[n]. Esta conversão recorre à leitura do sinal em intervalos de tempo constantes designados por amostragem, dando assim origem a uma sequência de valores que se traduzem no sinal

Figura 2.2: Diagrama de blocos de um DSP

de forma digital. Na Figura 2.3 podemos ver um exemplo de conversão de um sinal analógico para digital e a reconversão para analógico.

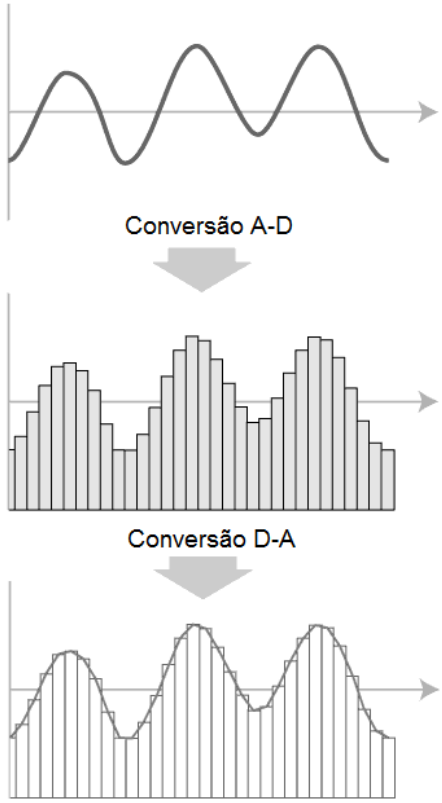

Figura 2.3: Exemplo de conversão A/D e D/A.

O filtro digital processa a sequência de números x[n] baseado em várias operações matemáticas, alterando assim as características do sinal. No que toca a processamento de imagens, a conversão A/D resulta numa sequência de valores que correspondem à intensidade dos pixeis. O filtro manipula as matrizes de pixeis da imagem de forma a obterem-se informações desejadas, ou eliminar aquelas que prejudicariam o correcto funcionamento da aplicação. A implementação destes filtros, podem ser tratados de várias maneiras, sendo elas: a nível de ponto em que o cálculo do pixel só depende do pixel original; operação local, que para o cálculo de um pixel são usados os pixeis vizinhos para alem do original; e por fim a operação global em que é usado todo o *frame*, ou seja, todos os pixeis da imagem. Dependendo da duração da resposta ao impulso, os filtros podem ser classificados em dois grupos:

1. Filtros Digitais de resposta ao impulso de duração finita (FIR), cuja operação é regida por equações lineares de diferenças com coeficientes de natureza não recursiva.

2. Filtros Digitais de resposta ao impulso de duração infinita (IIR), cuja operação é regida por equações lineares de diferenças com coeficientes constantes de natureza recursiva.

Os filtros FIR [7], são filtros que têm uma resposta ao impulso de duração finita não recursiva, ou seja, a saída depende apenas da entrada actual e entradas anteriores como ilustra a estrutura na figura 2.4.

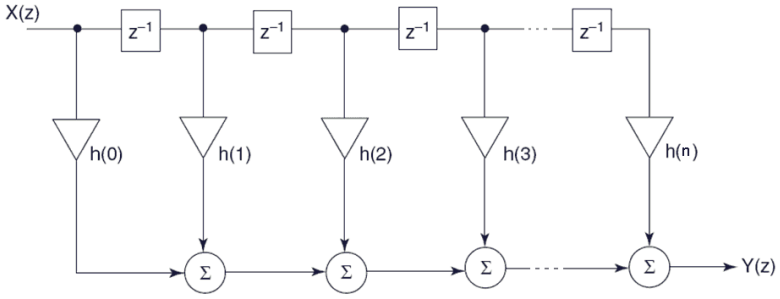

Figura 2.4: Estrutura de um filtro FIR.

A função X(z) representa o sinal de entrada, os valores de h(0) a h(n) são coeficientes que determinam o tipo de filtragem que se pretende e a função Y(z) é o sinal de saída, como resultado do sinal X(z) filtrado. O cálculo da saída é traduzido na seguinte forma:

$$Y(z) = h(0)x(z) + h(1)x(z)z^{-1} + h(2)x(z)z^{-2} + \ldots + h(n)x(z)z^{-n} \tag{2.1}$$

A grande vantagem destes tipos de filtro é a propriedade de realizar uma resposta em frequência com fase linear. Já os filtros IIR [8], são filtros que têm uma resposta ao impulso de duração infinita de natureza recursiva, ou seja, a saída não só depende da entrada actual mas também do resultado da saída anterior. Mais uma vez a figura 2.5 mostra a estrutura de um filtro digital IIR.

Mais uma vez, a função X(z) representa o sinal de entrada, os valores de a(0) a a(n) e b(1) a b(n) são coeficientes que determinam o tipo de filtragem que se pretende e a função Y(z) é o sinal de saída. O cálculo da saída é executado na seguinte forma:

$$\begin{aligned} Y(z) \quad = \quad & a(0)x(z) + a(1)x(z)z^{-1} + a(2)x(z)z^{-2} + \ldots + a(n)x(z)z^{-n} + & (2.2) \\ & + b(1)x(z)z^{-1} + b(2)x(z)z^{-2} + \ldots + b(n)x(z)z^{-n} & (2.3) \end{aligned}$$

Os filtros IIR, ao contrário dos FIR são difíceis de controlar a fase, mas os recursos computacionais são menores para se obter um mesmo filtro.

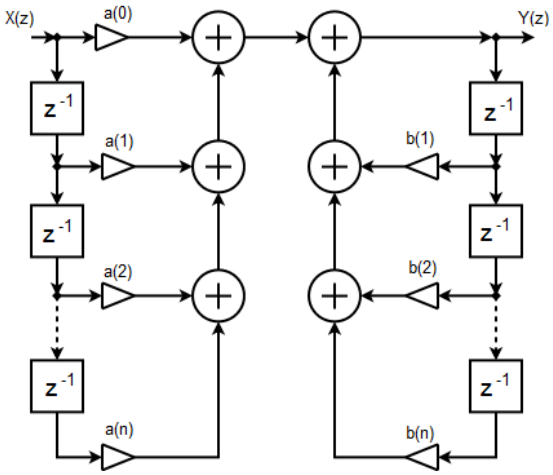

Figura 2.5: Estrutura de um filtro IIR.

2.4.1 Filtros baseados em histogramas

Um histograma representa a frequência de ocorrência do valor da intensidade dos píxeis numa imagem [9, 1]. Com ele conseguimos visualizar imediatamente se há mais pixeis de valor alto (claros) ou de valor baixo (escuro), ou seja, se uma imagem tem baixo contraste ou alto contraste, como podemos ver na Figura 2.6.

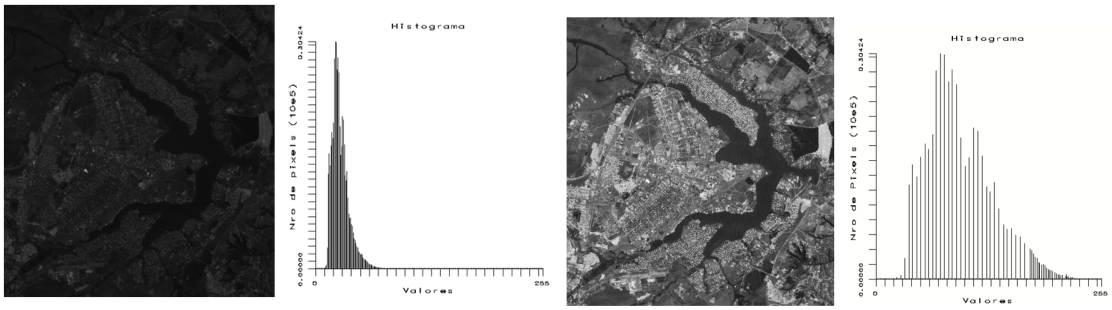

Figura 2.6: Exemplo de figura e histograma com baixo e alto contraste.

Como podemos ver no histograma da figura 2.7 com baixo contraste os níveis de cinza ocupam um pequeno intervalo de valores possíveis, já na imagem de alto contraste, os níveis de cinza ocupam quase todo o intervalo possível de valores. Um filtro baseado na manipulação de um histograma usa uma função que mapeia as variações dentro de um intervalo original de tons de cinza [0,N-1] da imagem original para um outro intervalo desejado [0,M-1].

A função referida define os pixeis com um valor x, para um valor y calculado. As aplicações mais usuais neste tipo de filtro são as de esticar, equalizar ou cortar zonas de intensidade do pixel. Já existe também algum estudo sobre remoção de ruído [10].

No caso de esticar, filtros conhecidos por "*Contrast stretching*" [11, 12] as hipóteses são várias:

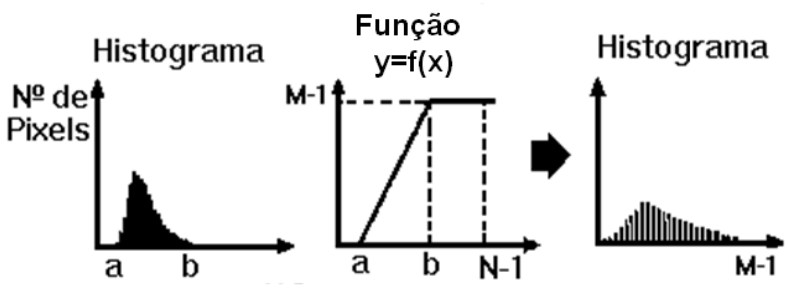

Figura 2.7: Ilustração da manipulação do histograma de uma imagem.

- Linear

$$y = Ax + B \tag{2.4}$$

- Raiz quadrada

$$y = A \cdot \sqrt{x} \tag{2.5}$$

- Quadrado

$$y = A \cdot x^2 \tag{2.6}$$

- Logaritmo

$$y = A \cdot log(x+1) \tag{2.7}$$

- Negativo

$$y = -(A \cdot x + b) \tag{2.8}$$

Na figura 2.8 podemos ver um exemplo do histograma original e o que resulta após o processamento para cada uma das funções do "*Contrast stretching*. Na figura 2.9 podemos ver as funções de como alterar o brilho e o contraste.

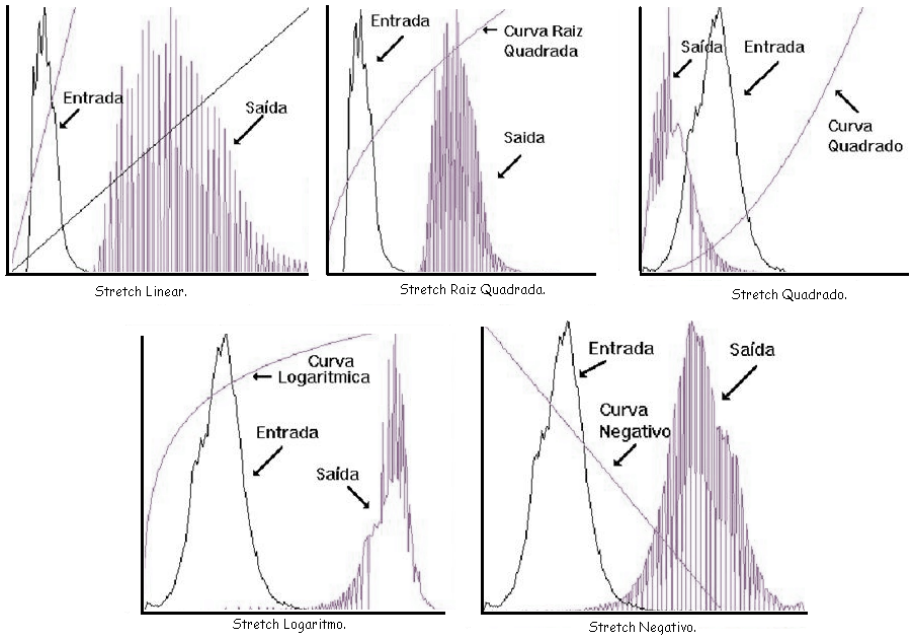

Figura 2.8: Ilustração das varias funções *Contrast stretching*.

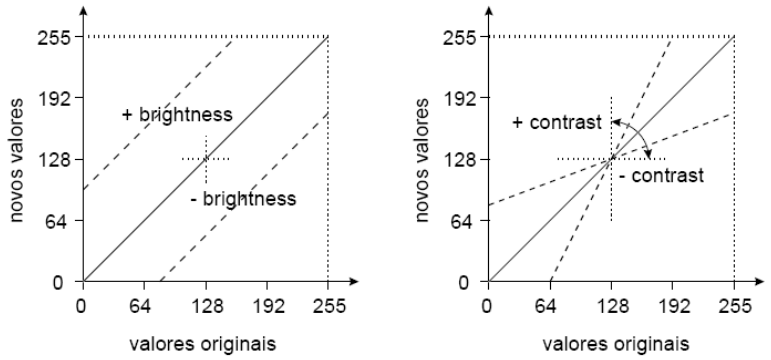

Figura 2.9: Representações das funções para mudar brilho e contraste.

2.4.2 Filtros baseados em convolução

Na matemática, o teorema da convolução [9, 13] indica que sob circunstâncias apropriadas [14] a transformada de Fourier de uma convolução é o produto das transformadas de Fourier de dois sinais. Neste caso, a convolução no domínio do tempo significa uma multiplicação no domínio das frequências. A sequência representada na figura 2.10 mostra uma operação, no domínio real e no domínio das frequências, de uma sinusóide modelada com um ruído também ele sinusoidal, mas com uma frequência mais elevada. No domínio da frequência o ruído manifesta-se através dos impulsos mais afastados do centro.

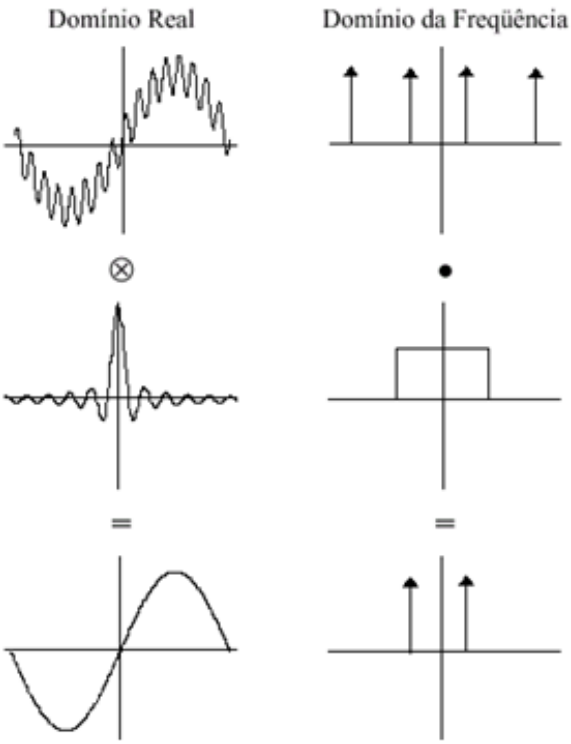

Figura 2.10: Comparação entre Filtragem no Domínio Real e no da Frequência.

Facilmente reparamos que a filtragem no domínio da frequência é muito simples, pois equivale apenas a uma multiplicação. Assim, um filtro de convolução pode ser representado na forma da Figura 2.11 e a imagem resultante será calculada da seguinte forma:

$$G(u,v) = H(u,v) \cdot F(u,v) \tag{2.9}$$

$$g(x,y) = TF^{-1}[G(u,v)] \tag{2.10}$$

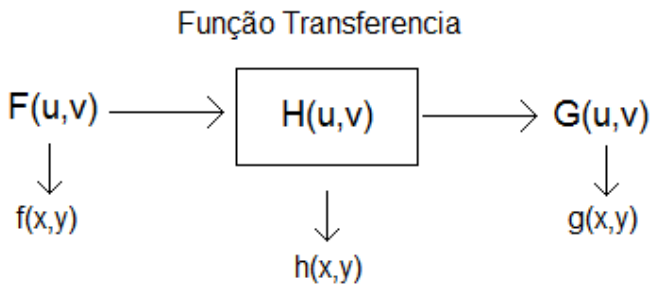

Figura 2.11: Diagrama de um filtro de Convolução

g(x,y) é a imagem que resulta da original f(x,y) após uma filtragem que neste caso é a multiplicação pela matriz h(x,y) que se trata de uma representação espacial da função transferência H(u,v). No caso discreto g(x,y) é calculado da seguinte forma:

$$g(x,y) = \frac{1}{N} \cdot \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) \cdot h(x-i, y-j) \tag{2.11}$$

A Matriz de convolução que é multiplicada pela imagem, conhecida por máscara, é um conjunto de valores que é responsável por definir, para cada elemento, o modo como este se relaciona com os seus vizinhos. Esta máscara percorre todos os pixeis da imagem e alterando os seus valores de acordo com os valores dos vizinhos e da máscara. Os valores da máscara são utilizados como pesos aplicados sobre os valores originais na imagem. A Figura 2.12 ilustra como a máscara percorre a imagem e como esta influencia os seus valores.

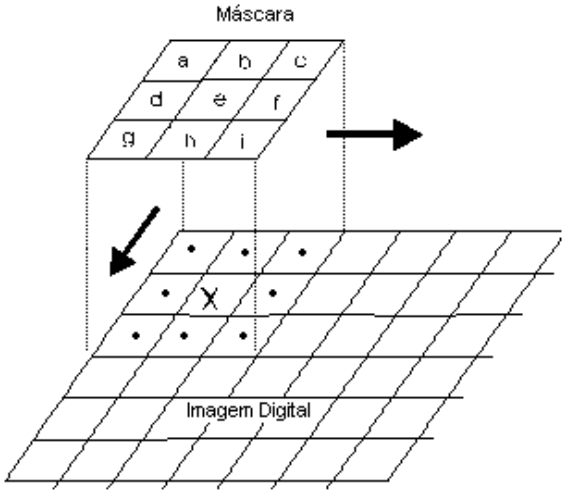

Figura 2.12: Descrição da máscara a percorrer a imagem original.

A aplicação da máscara com centro na posição (i,j), sendo i o número de uma dada linha e j o número de uma dada coluna sobre a imagem, consiste na substituição do valor do pixel na posição (i,j) por um novo valor, o qual depende dos valores dos pixeis vizinhos e dos pesos da máscara.

Quanto à matriz h(x,y), como já foi dito, trata-se de uma representação espacial da função transferência H(u,v). Assim, por exemplo, para obter um filtro passa baixo que atenua ou elimina os componentes de alta frequência do domínio de Fourier a matriz será a seguinte:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Ao valor final calculado, deve-se ainda dividir pelo somatório dos valores da matriz, valor representado por N na formula acima, para que este não ultrapasse os limites do valor de um pixel.

Este filtro, conhecido por *blur*, traduz-se numa redução dos detalhes da imagem. Para uma média mais ponderada, a matriz deve realçar mais o pixel central, como por exemplo:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Seguem-se agora algumas matrizes que se traduzem em filtros conhecidos:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Traduz-se num filtro passa alto, que atenua ou elimina as componentes de baixa frequência, resultando no aumento do contraste e nitidez da imagem. O seu efeito é normalmente conhecido por *Sharpen*.

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & - & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

Este filtro conhecido por *Edge enhance* produz um efeito de detecção de bordas ou arestas, em que toda a imagem fica escura excepto as linhas que salientam os contornos, numa certa direcção definida na matriz, dos objectos nas imagens. O seu funcionamento corresponde a deslocar a imagem e subtrair da imagem original.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Trata-se de um filtro Laplaciano, conhecido por *Edge Detect*, o seu funcionamento é idêntico ao anterior mas a detecção de bordas de contornos funciona a toda à volta dos objectos na imagem.

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & -2 \\ 1 & 1 & -1 \\ 2 & 1 & 0 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} 2 & 1 & 0 \\ 1 & 1 & -1 \\ 0 & -1 & -2 \end{bmatrix}$$

Um filtro *emboss* dá um efeito 3D de sombra à imagem, que se traduz num efeito *bumpmap* [15]. Trata-se de um efeito de superfície com rugosidade parecido com um mapa de textura. É também um efeito com direcção. Se o pixel central da matriz deste filtro for substituindo por zero, a imagem é subtraída e o resultado será apenas o relevo da imagem. Os trabalhos [16, 17, 18] mostram-nos possíveis implementações deste filtro numa FPGA.

2.4.3 Filtros baseados em matemática morfológica

Sendo uma imagem um conjunto de pixeis num plano cartesiano, estes filtros gozam das bases matemáticas da teoria de conjuntos [9, 13, 1, 19]. O seu princípio é extrair informações relativas à geometria e topologia do conjunto de pixeis de uma imagem, pela transformação através de um conjunto bem definido chamado elemento estruturante. Assim todas as operações sobre conjuntos tornam-se aplicáveis no domínio do processamento de imagem, como por exemplo a intercepção, união, complemento e diferença. A figura 2.13 ilustra algumas dessas operações.

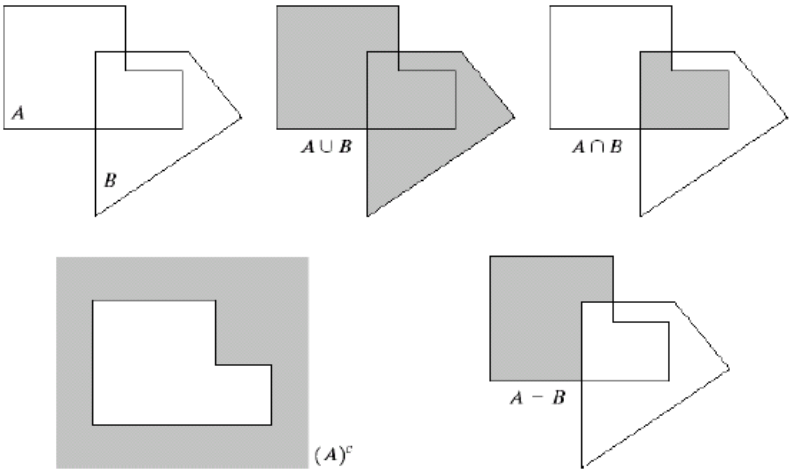

Figura 2.13: Operações de conjuntos. (Retirado de [1])

A morfologia matemática é uma abordagem ao processamento de imagens baseada na forma do objecto de interesse. Adequadamente utilizadas, as operações tendem a simplificar os dados da imagem, preservando as características de forma essencial e eliminando irrelevâncias. Ela é particularmente útil na identificação de objectos, extracção de características de objectos e identificação de defeitos relacionados à forma dos objectos. As transformações básicas da morfologia matemática são a dilatação e a erosão. Outras transformações são combinações destas, tais como " *Opening´´* e " *Closing´´*.

A dilatação é a transformação morfológica que combina dois conjuntos através da adição vectorial de elementos dos conjuntos. Sendo o conjunto A a imagem original a ser processada e um conjunto B conhecido por elemento estruturante, com parâmetros escolhidos para a transformação e que normalmente é uma matriz de tamanho 3x3, a dilatação é expressa da seguinte forma:

$$A \oplus B = c \in E_n \mid c = a + b \; para \; todo \; a \in A \; e \; b \in B \tag{2.12}$$

O efeito da dilatação sobre uma imagem é o crescimento ou expansão do objecto em relação ao fundo como podemos ver na figura 2.14.

A erosão é a operação morfológica dual da dilatação. Ela é uma transformação morfológica que combina dois conjuntos utilizando a subtracção vectorial de elementos dos conjuntos. O seu efeito sobre uma imagem é o encolhimento do objecto em relação ao fundo.

$$A \ominus B = x \in E_n \mid x + B \in A \ para\ todo\ b \in B \tag{2.13}$$

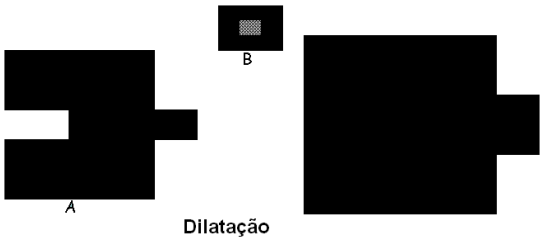

Figura 2.14: Exemplo do efeito de dilatação. (Retirado de [1])

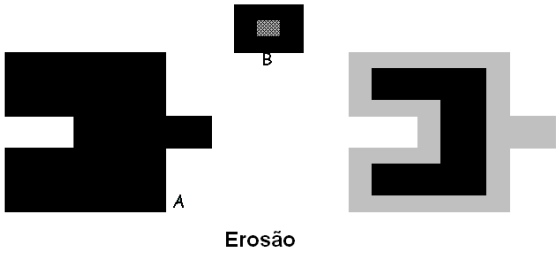

Figura 2.15: Exemplo do efeito de erosão. (Retirado de [1])

Se encadearmos filtros dentro destas topologias, conseguimos obter filtros designados por *Opening* que consiste em aplicar um filtro de erosão seguido de dilatação, utilizando o mesmo elemento estrutural e *Closing* que consiste em aplicar uma dilatação seguido de erosão. A operação *Opening* tem como objectivos principais, separar objectos muito próximos de uma imagem, ou seja, criar espaços entre objectos na imagem e eliminar ruídos do tipo pixeis negros espalhados aleatoriamente por toda a imagem. A sua função é definida da seguinte forma:

$$A^{\circ}\ B = (A \ominus B) \oplus B \tag{2.14}$$

Já a operação *Closing* tem como objectivos juntar objectos muito próximos de uma imagem, isto é, elimina espaços entre objectos na imagem e eliminar falhas dentro dos objectos, ou seja, pixeis brancos espalhados aleatoriamente por toda a imagem. A sua função é definida da seguinte forma:

$$A \cdot B = (A \oplus B) \ominus B \tag{2.15}$$

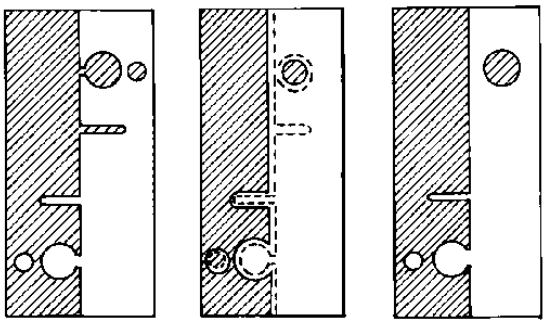

Figura 2.16: Ilustração do filtro Opening.

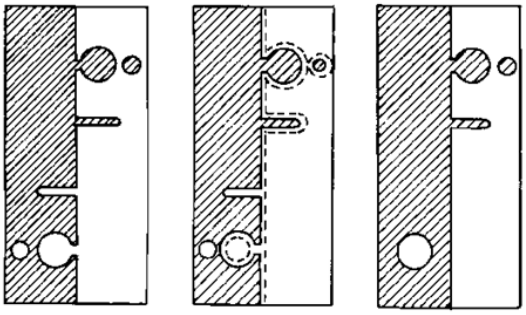

Figura 2.17: Ilustração do filtro Closing.

O processamento de imagens com matemática morfológica é, na sua maioria, aplicado em imagens binárias como foi visto até agora. Para processar as imagens em tons de cinza, os valores da matriz do elemento estruturante correspondem aos pixeis considerados no sinal de entrada.

Deste modo o resultado de um filtro de dilatação é dado por:

$$\delta_B(f) = max\{x_k,\ k \in B\} \tag{2.16}$$

O resultado consiste na escolha do valor máximo dos valores da função de entrada, mas que pertençam ao grupo definido pelo elemento estruturante. O processo para um filtro de erosão é idêntico, apenas muda a escolha do valor que passa a ser o mínimo. A sua função é a seguinte:

$$\varepsilon_B(f) = min\{x_k,\ k \in B\} \tag{2.17}$$

A figura 2.18 ilustra estes dois processos.

Quanto aos filtros *Opening* e *Closing*, também estes podem ser aplicados com filtros na escala de cinzentos. O processo é o mesmo, basta encadear os filtros de erosão seguido de dilatação e vice-versa. São então definidas da seguinte forma:

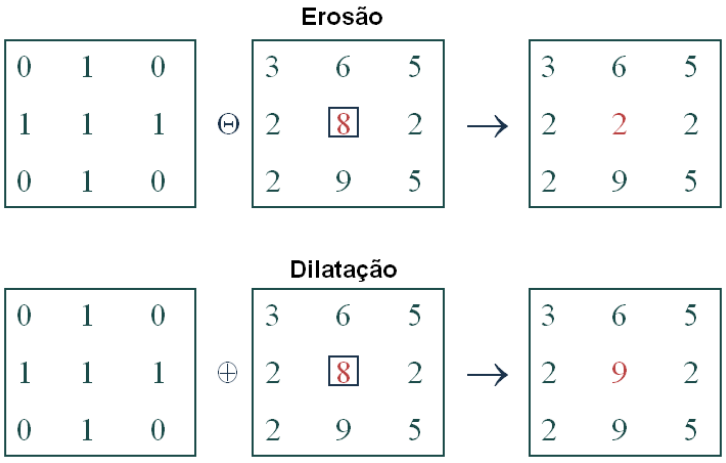

Figura 2.18: Ilustração dos processos Opening e Closing em escala de cinzentos.

- Opening

$$\gamma_B(f) = \delta_B \circ \varepsilon_B \tag{2.18}$$

- Closing

$$\varphi_B(f) = \varepsilon_B \circ \delta_B \tag{2.19}$$

Os trabalhos referenciados [20, 21, 22, 23, 24] revelam possíveis implementações deste filtro em FPGAs.

2.5 FPGAs (*Field Programable Gate Array*)

As FPGAs são dispositivos que suportam a implementação de circuitos lógicos relativamente grandes. Surgiram da evolução de dispositivos lógicos programáveis [25] e foram desenvolvidos em 1983 pela empresa Xilinx Inc, sendo apenas lançadas no ano de 1985 como dispositivos que poderiam ser programados de acordo com as aplicações do utilizador. A sua arquitectura [26] consiste num grande arranjo de células lógicas ou blocos lógicos configuráveis contidos num único circuito integrado. Cada célula contém capacidade computacional para implementar funções lógicas e realizar o encaminhamento das ligações para comunicação entre elas. Em algumas famílias de dispositivos da XIlinx, as FPGAs são compostas por três tipos de componentes: CLBs, circuitos idênticos, que um utilizador pode usar para construir elementos funcionais lógicos, que resultam da reunião de flip-flops (entre 2 e 4) e a utilização de lógica combinacional; IOBs, circuitos responsáveis pelo interface das entradas e saídas das combinações dos CLBs, são basicamente *buffers*, que funcionam como um pino bidireccional de entrada e saída da FPGA; "*Switch Matrix*", trata-se da rede de interligações programáveis entre CLBs e IOBs, a configuração é estabelecida

por programação interna das células de memória estática, que determinam funções lógicas e conexões internas implementadas no FPGA entre os CLBs e os IOBs.

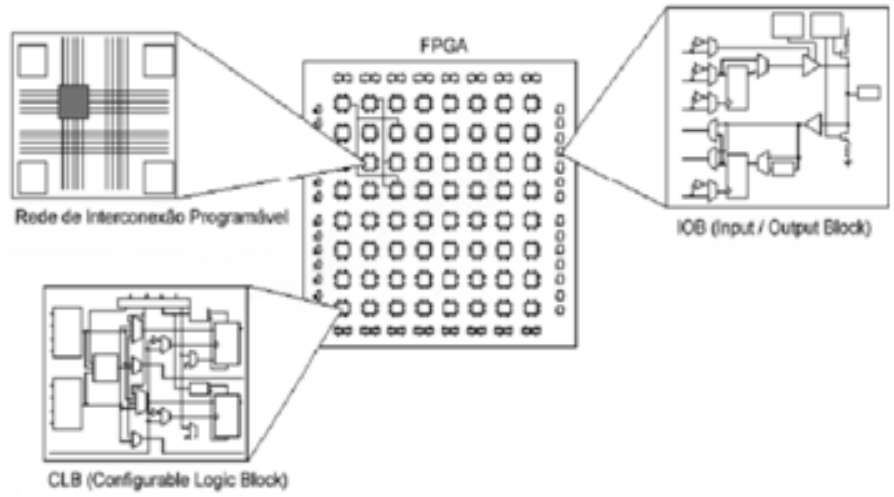

Figura 2.19: Representação das diferentes partes de uma FPGA.

De origem estes circuitos FPGAs não possuem qualquer tipo de funcionalidade, exigindo assim, uma programação inicial para que determinadas funções possam ser executados pelo circuito. Essa programação é carregada através de um arquivo binário designado por *bit stream*, cujo conteúdo (0 ou 1) define o estado do elemento reconfigurável da FPGA, ou parte dele.

2.5.1 As Vantagens da FPGA

Esta secção faz uma apreciação a nível de flexibilidade e desempenho dos sistemas em FPGAs face a outras tecnologias tais como sistemas baseados em CPUs (microprocessadores, microcontroladores e DSPs) e sistemas em ASICs que são dispositivos fabricados para uma aplicação específica.

Qualquer função pode ser programada para correr num CPU e de uma forma relativamente simples pois basta programar instruções que ficam armazenadas numa memória. No caso das FPGAS e ASICs é mais difícil, uma vez que isso implica o desenvolvimento de um circuito lógico capaz de exercer tal função. Em caso de necessidade de reprogramação, tanto o CPU como a FPGA permitem-no e mais uma vez, de uma forma simples no caso de utilização de um CPU. Para um ASIC é impossível pois, trata-se de *hardware* estático e uma reprogramação exige o fabrico de um novo produto. Assim, do ponto de vista da flexibilidade, os sistemas baseados em CPUs são sem duvidada a melhor opção. Já do ponto de vista da performance, os papeis invertem-se totalmente. A não necessidade de "*fetch*", "*decode*" ou "*memory access overhead*" usado nos CPUs, num circuito lógico dedicado, torna este tipo de dispositivos mais rápidos e com um menor consumo. A implementação de algoritmos em *hardware* é particularmente eficiente para aplicações que trabalham com manipulação directa de bits, como por exemplo: compressão de dados, reconhecimento de padrões, criptografia, tratamento de imagens, processamento de sinais, etc. Estas

vantagens são ainda maiores quando são usadas arquitecturas baseadas em paralelismo e *pipelining* que é o caso da maioria dos algoritmos aplicados no processamento de imagens ou vídeos. A implementação de um circuito num ASIC exige uma densidade lógica menor em relação à implementação numa FPGA, o que faz dele um dispositivo mais rápido e com um consumo menor, sendo a melhor opção a nível de performance. O crescimento da densidade lógica e a integração de *PowerPCs* nas FPGAs têm vindo a melhor estes dispositivos e em condições ideais, estes dispositivos tiram partido das melhores características tanto do *software* como do *hardware*. A Figura 2.20 mostra o enquadramento ao nível da flexibilidade e performance das FPGAs face aos CPUs e ASICs.

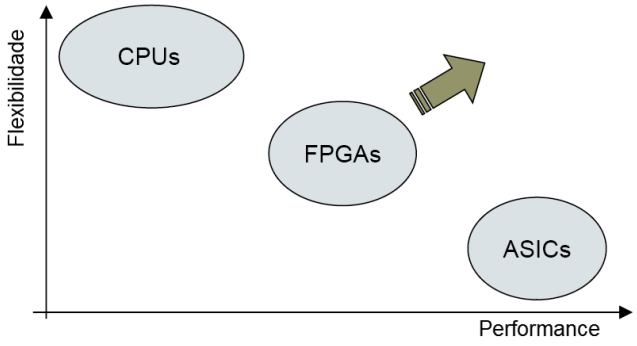

Figura 2.20: Avaliação das FPGAs vs. CPUs vs. ASICs.

2.6 Reconfiguração dinâmica

Uma das principais qualidades dos dispositivos FPGAs é sua capacidade para serem reprogramados dinamicamente. Actualmente, modelos tais como Virtex Series da Xilinx, AT4000 e AT6000 da Atmel já permitem reconfiguração dinâmica de alta velocidade. Uma FPGA é denominada como "dinamicamente reconfigurável" se permitir a reconfiguração de alguns blocos lógicos e segmentos de encaminhamento de ligações, enquanto outras partes asseguram um funcionamento contínuo do sistema. Esta reconfiguração de alta velocidade permite então que o *hardware* se adapte a mudanças no sinal de entrada ou sinais externos, em tempo de execução. A este procedimento de reconfiguração em tempo de execução é chamado de *Run Time Reconfiguration* (RTR). Ao contrário dos computadores que trabalham as funções sequencialmente através de unidades de tempo, as arquitecturas reconfiguráveis funcionam com unidades configuradas no espaço. Perante estas características, a reconfiguração dinâmica revela vantagens importantes [27], pois além de garantir o funcionamento do serviço sem interrupções, permite a adaptabilidade do sistema, que pode ser a nível do seu funcionamento, adaptando a sua lógica perante os sinais capturados, ou diminuindo o desempenho para o básico necessário melhorando assim o consumo. Com a reconfiguração dinâmica, os sistemas ficam ainda mais robustos e versáteis pois em determinados ciclos de tempo o sistema pode realizar uma auto-avaliação e fazer um restauro

em caso de faltas detectadas, caso este seja necessário. Permite também a partilha dos recursos da FPGA. A Reconfiguração dinâmica oferece então importantes benefícios para obter alta performance, enquanto minimizamos os recursos de *hardware* requeridos nas implementações de muitas aplicações. A implementação de sistemas que exigem flexibilidade, alto desempenho, alta taxa de transferência de dados e eficiência no consumo de energia são agora possíveis com esta tecnologia. Isto inclui aplicações de televisão digital, processamento de imagem em tempo real e sinais adaptáveis, comunicações sem fios reconfiguráveis, sistemas de computação de alto desempenho, produtos para consumo actualizáveis remotamente, etc. A reconfiguração de uma FPGA pode ser total onde o dispositivo reconfigurável é inteiramente alterado, ou apenas parcial em que a configuração permite apenas que uma porção da área seja reconfigurada [total ou parcial]. Uma reconfiguração parcial pode ser não disruptiva, quando as porções do sistema que não são reconfiguradas permanecem completamente funcionais durante o ciclo de reconfiguração permitindo assim o funcionamento contínuo do sistema [28, 29, 30]. Quando a reconfiguração parcial afecta outras partes do sistema tipicamente implica uma paragem no sistema inteiro. Reconfiguração parcial não disruptiva é frequentemente abreviada para reconfiguração parcial. Em resumo, a reconfiguração dinâmica, também chamada de *run-time reconfiguration*, *on-the-fly reconfiguration* ou *in-circuit reconfiguration*, termos estes, normalmente associados à reconfiguração do dispositivo sem que este interrompa a sua execução. Uma forma de expressar a reconfiguração parcial não disruptiva que implica não haver necessidade de reiniciar o circuito ou remover elementos durante a reconfiguração.

2.6.1 Virtex II

A Virtex-II Pro [2] trata-se de uma plataforma de FPGAs com uma vasta gama de unidades que, no que diz respeito às estruturas de lógia interna, vão desde baixa densidade, 3168 células lógicas no caso do modelo XC2VP2, até alta densidade, 99216 células lógias no caso do modelo XC2VP100. O estudo em particular desta família de dispositivos deve-se ao facto de ser um modelo que permite a reconfiguração dinâmica e que a FPGA disponível para a elaboração deste trabalho faz parte desta família de placas. A placa em questão é o modelo XC2VP30 que possui dois PowerPCs e uma densidade de 30816 células lógicas, o que equivale a 13696 *slices*.

Os blocos lógicos configuráveis (CLB) desta placa estão organizados num conjunto e são usados para implementar circuitos lógicos e síncronos. Cada elemento CLB está ligado a uma *switch matrix* para ter acesso à rede de interconexões programável como mostra a figura 2.21. Cada elemento CLB possui quatro *slices* iguais com ligação rápida ao CLB. Os quatro *slices* estão divididos em duas colunas de dois elementos com a cadeia lógica de *carry* independente e uma cadeia de *shift* comum. Cada *slice* contem 2 geradores de funções de quatro entradas cada, *carry* lógico, portas lógicas aritméticas, multiplexadores e dois elementos de armazenamento. Como mostra a figura 2.22 cada função geradora de funções pode ser programada para ser uma LUT de quatro entradas, memória RAM de 16bit ou *shift register*.

O processo de configuração da virtex necessita de carregar os dados de configuração numa memória SDRAM *Static Random Access Memory* interna do dispositivo [31]. Os únicos modos

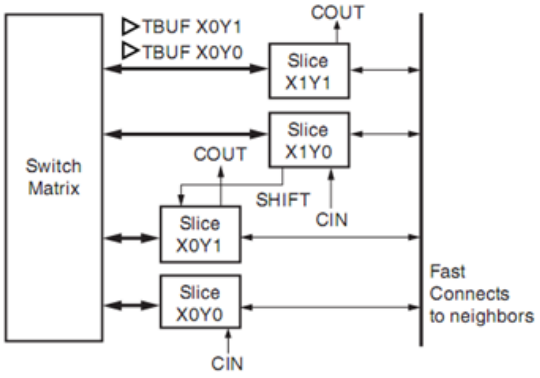

Figura 2.21: Diagrama geral do CLB. Retirada de [2]

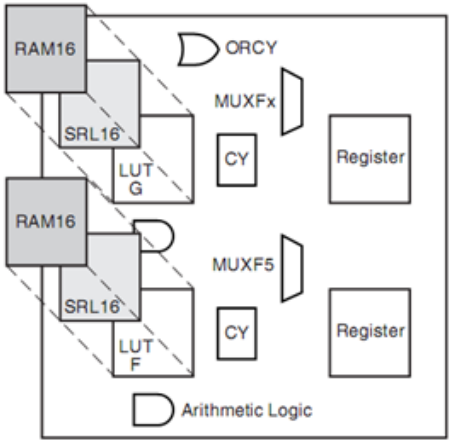

Figura 2.22: Vista do Slice em detalhe. Retirada de [2]

que permitem a reconfiguração dinâmica parcial é o *selectMAP* que funciona em modo paralelo usando um barramento de dados de 8bit bidireccional e o modo *Boundary-Scan* [32] cuja configuração é feita em série usando unicamente o TAP. Neste modo, através do registo de configuração, os dados são carregados pelo TDI e convertidos em pacotes de dados para o canal de configuração interno.

Capítulo 3

Projecto

Este capítulo enumera, descreve e explica o funcionamento dos módulos que fazem parte do projecto que será implementado na FPGA. Neles estão incluídos os filtros que fazem parte da biblioteca de módulos reconfiguráveis, um módulo que permite guardar imagens numa memória SDRAM e que permite usa-las como entrada para os filtros ou guardar como resultados destes. Descreve também os passos que foram necessários para testar uma cadeia de filtros, o qual envolveu uma divisão do projecto em dois para implementação em duas placas Spartan3.

3.1 O Projecto

Nesta fase desenvolveram-se os módulos, que fazem parte da biblioteca, individualmente e sem preocupações relativamente ao suporte para a reconfiguração dinâmica. Este desenvolvimento foi elaborado para a placa Spartan3 da digilent, não só por motivos de logística, mas também pela disponibilidade de um projecto já funcional nesta placa que permitiu uma boa base de trabalho.

Um outro módulo foi desenvolvido para usar a memória SRAM da placa e guardar imagens que podem ser usadas como entrada para os filtros, ou guardar imagens que resultam destes. De forma a permitir o envio de dados para configurar os filtros foi elaborado um programa de comunicação com a FPGA através da porta serie.

Numa fase seguinte, com o objectivo de testar o encadeamento de filtros de forma a podermos usar vários níveis de filtragem, dividiu-se o projecto em dois, pois uma FPGA não possui recursos suficientes para a implementação de uma cadeia. Uma das placas fará a recepção de vídeo através da câmara e terá parte da cadeia de filtros, a outra, para além da outra parte da cadeia de filtros terá também a saída de vídeo para o monitor.

As ferramentas que foram necessárias para esta fase do projecto foram as seguintes:

- **Xilinx ISE Design Suite 10.1** - trata-se de um conjunto de ferramentas de *software*, fornecidas pela Xilinx que suporta as várias fases de um projecto, desde a sua criação, desenvolvimento, algum teste, sintetização e implementação. Deste conjunto os programas mais

usados foram o Project Navigator, usado para a criação, desenvolvimento e sintetização do projecto; o CORE Generator que permitiu a criação de alguns módulos, nomeadamente os DCM's necessários; por fim o iMPACT que permite a implementação do *bit stream* na FPGA usando um cabo JTAG. Este software encontra-se disponível em [33].

- **ModelSim XE III 6.1e** - Esta ferramenta de *software*, fornecida pela Mentor Graphics, permite a simulação e teste dos módulos criados para o projecto. Disponível em [34].
- **Spartan-3 Starter Board da digilent** - [35] é uma placa, apresentada na figura 3.1 com o chip XC3S200-FT256 onde o projecto será inicialmente implementado.
- **Câmara digital** - Câmara [36], que faz a aquisição de vídeo para a FPGA.
- **Módulo de vídeo** - este modulo é o interface entre a FPGA e o monitor, ele faz o armazenamento e controlo de imagem e para tal usa 3 FIFO's, uma para cada canal do RGB e um VDAC que converte o sinal digital para analógico.
- **Osciloscópio digital** - Usado para análise dos sinais entre a placa com a FPGA e os periféricos câmara e módulo de vídeo.

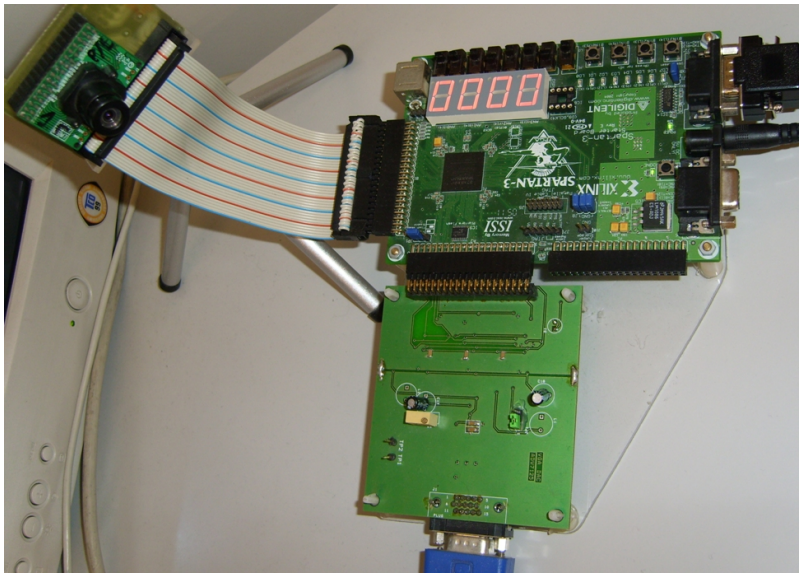

Figura 3.1: Fotografia do dispositivo Spartan-3 com os periféricos ligados

O projecto base, que permitiu o arranque deste trabalho foi disponibilizado na disciplina de PSDI (Projecto de Sistemas Digitais) pelos Professores José Carlos Alves e António José Araújo cuja cadeia de vídeo tem a constituição ilustrada na figura 3.2

Trata-se de um projecto funcional para a placa Spartan3, que adquire o vídeo através de uma câmara e reproduz num monitor, deixando condições para a criação de módulos que efectuem processamento do vídeo, nomeadamente, filtros. O projecto trabalha o vídeo com uma resolução

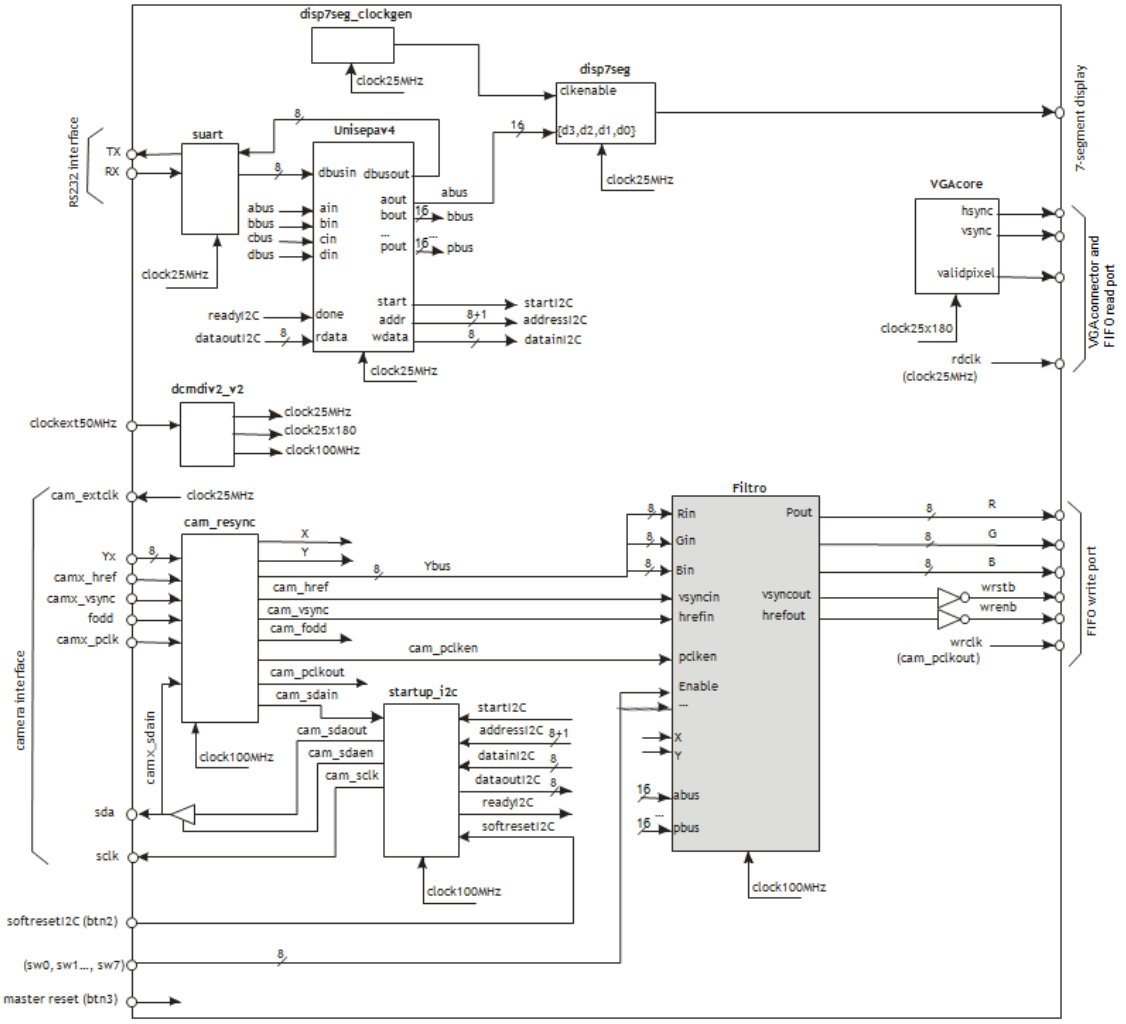

Figura 3.2: Representação da cadeia de vídeo.

de 640x480 píxeis e uma escala de 256 níveis de cinza por pixel. A chegada de pixéis é sequencial e percorre a *frame* linha a linha da esquerda para a direita e de cima para baixo. A frequência de chegada destes ao módulo é de 12,5MHz e a frequência base do projecto é de 100MHz, deixando assim uma margem de oito *clocks* de 10ns para efectuar o cálculo do pixel de saída. O cumprimento deste tempo para o cálculo na projecção dos módulos é vital pois de outro modo o filtro iria acumular um atraso que levaria a perda de informação. O projecto inclui também um módulo de interface com um computador, para que este possa enviar dados para os módulos dos filtros, possibilitando assim, configurações externas.

A tabela 3.1 mostra as entradas disponíveis e as saídas essenciais que um módulo de processamento de imagem, designado por filtro, possui.

Tabela 3.1: Pinos do Filtro

Pino	Tipo	Descrição
clock	Entrada	sinal de relógio para o funcionamento síncrono com o projecto
reset	Entrada	sinal que ordena um reset.
Pin	Entrada	barramento de 8bit com a informação do nível da intensidade do pixel. O valor mais baixo representa o preto e o mais alto representa o branco
hrefin	Entrada	sinal de sincronismo que permite a sincronização horizontal da imagem, indicando o inicio e fim das linhas.
vsyncin	Entrada	sinal de sincronismo que permite a sincronização vertical da imagem, indicando o inicio e fim das imagens
pclken	Entrada	flag que indica a disponibilidade de um novo pixel no Pin.
abus, bbus, ..., pbus	Entrada	barramentos que ligam o filtro ao módulo de interface da porta serie permitindo assim a entrada de valores provenientes de um computador para configurar os filtros.
sw0,..., sw7	Entrada	interruptores
X e Y	Entrada	barramentos de 10 e 9 bits que indicam, numa matriz de 640x480, a posição do pixel que está no barramento Pin.
Pout	Saída	barramento de 8bit com a informação do nível da intensidade do pixel filtrado.
hrefout	Saída	sinal para sincronização horizontal da imagem, indicando o inicio e fim das linhas de píxeis de saída do filtro
vsyncout	Saída	sinal para sincronização vertical da imagem, indicando o inicio e fim da imagem com os píxeis de saída do filtro

3.2 Filtros

Uma vez que o interesse neste projecto é processar a segmentação de vídeo em tempo real, os filtros que foram escolhidos para serem implementados, são os que para o cálculo do pixel de saída é usado apenas o pixel original ou os seus vizinhos. Deste modo o atraso do resultado em relação à entrada do filtro será no máximo o tempo de processamento de algumas linhas de uma frame e os recursos exigidos de *hardware* não são elevados. Assim, dos filtros estudados, os que melhor encaixam neste perfil e por tanto os escolhidos para este projecto, foram um filtro de *threshold* que pertence à classe dos filtros baseados em histograma, um filtro de detecção de bordas, um de convolução, e um morfológico. Uma vez que o funcionamento teórico de alguns destes filtros foi explicado no estado de arte deste documento, nesta secção será descrita apenas a sua implementação e funcionamento.

3.2.1 Filtro de threshold

Este filtro enquadra-se nos filtros baseados em histograma da imagem, mas ao contrário da maioria dos filtros deste género, não necessita de analisar o histograma da imagem inteira. É um filtro que trabalha localmente, ou seja, pixel a pixel, não sendo necessária a análise dos restantes píxeis. A função que define este filtro é a seguinte:

$$Pout = \begin{cases} 0 & \text{if } Pin \leq k \\ 255 & \text{if } Pin > k \end{cases}$$

O Valor de k representa o nível de *threshold* que é o nível de decisão entre os 256 níveis de cinzento do pixel à entrada. Se os píxeis tiverem valores mais altos que este nível de decisão, o pixel calculado assume valor máximo de brilho, que é 255 e representa o branco. Se estiverem a baixo do valor de k, os píxeis calculados assumem o valor 0, ou seja, preto. Como resultado a imagem à saída será binária como mostra a figura 3.3 que nos mostra uma imagem original e o resultado deste filtro para vários níveis diferentes de *threshold*. Este valor é configurado através da interface com o computador. O utilizador para o alterar terá somente de o enviar pela porta série, usando para o efeito, o programa "Hsender" que está em descrito em 4.1.

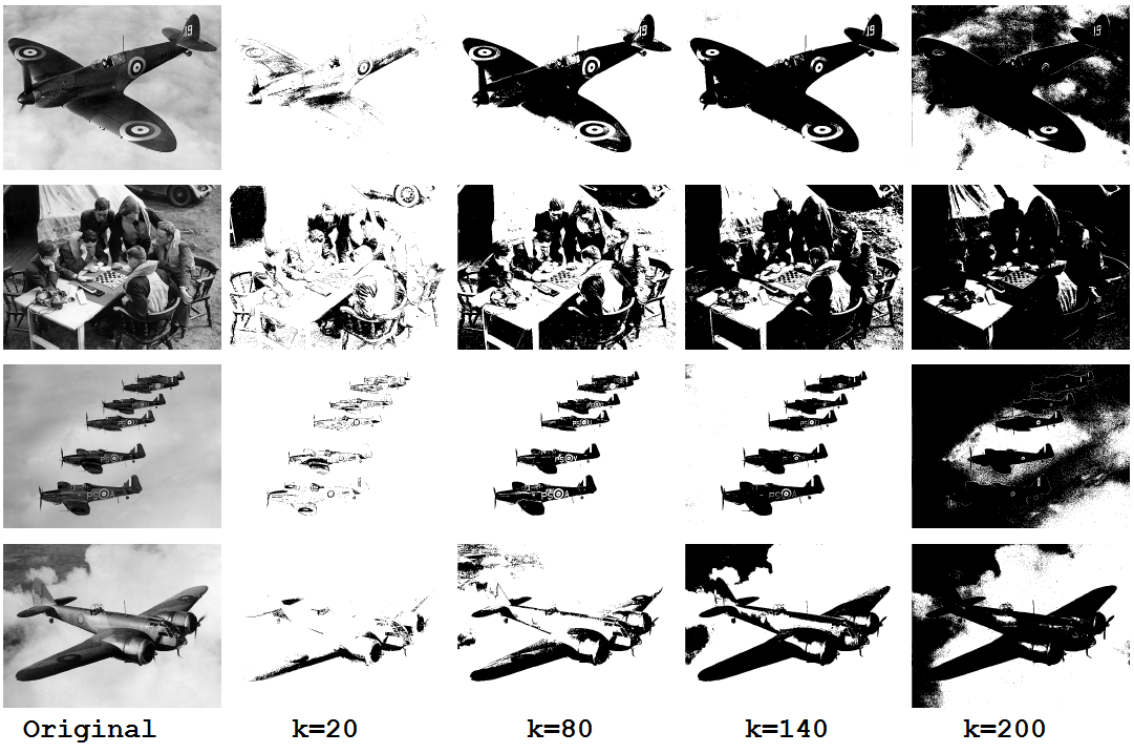

Figura 3.3: Imagens filtradas com vários níveis de *threshold*

3.2.2 Detecção de bordas

Este filtro é baseado no operador gradiente de Roberts, método não-linear, que realiza um procedimento simples e rápido de calcular. Em teoria, o operador consiste numa pequena operação de convolução de uma matriz de 2x2, em que uma máscara é simplesmente a outra, mas com uma rotação de 90°.

$$\begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

Para determinar se o pixel avaliado é ou não um pixel de borda, é usado o seguinte cálculo:

$$|G| = \sqrt{Gx^2 \cdot Gy^2} \tag{3.1}$$

Embora normalmente, uma magnitude aproximada é calculada usando:

$$|G| = |Gx| + |Gy| \tag{3.2}$$

Que é muito mais simples de calcular.

Para a implementação é necessário um módulo de memória, cuja estrutura é explicada na secção 3.2.5. Embora nessa secção descreva uma memória de tamanho 3x3, neste filtro apenas é necessário disponibilizar uma matriz 2x2, cujas coordenadas são as seguintes:

$$\begin{bmatrix} x0y0 & x1y0 \\ x0y1 & x1y1 \end{bmatrix}$$

O cálculo que substitui o valor do pixel é:

$$x0y0' = 2 \cdot (x0y0 - x1y1) + 2 \cdot (x0y1 - x1y0) \tag{3.3}$$

A arquitectura deste filtro, ilustrada na figura 3.4, para efectuar este cálculo necessita apenas de quatro dos oito ciclos de relógio disponíveis. O diagrama representado na figura 3.5 mostra máquina de estados que efectua o cálculo. A imagem que resulta deste filtro apresenta a desvantagem de depender da direcção e certas bordas, ou seja, certas bordas são mais realçadas que outras.

O módulo de sincronismo atrasa os sinais href e vsync de forma a compensar o tempo necessário para o armazenamento de pixéis na memória e cálculo do pixel. No caso deste filtro o atraso será relativo ao tempo de chegada de uma linha inteira de pixéis, mais um pixel.

Como resultado da aplicação deste filtro, obtém-se uma imagem com valores de nível altos, em regiões de limites bem definidos e valores baixos em regiões de limites suaves. Para regiões de nível constante, o valor do pixel é zero, ou seja ele destaca regiões de alta frequência espacial,

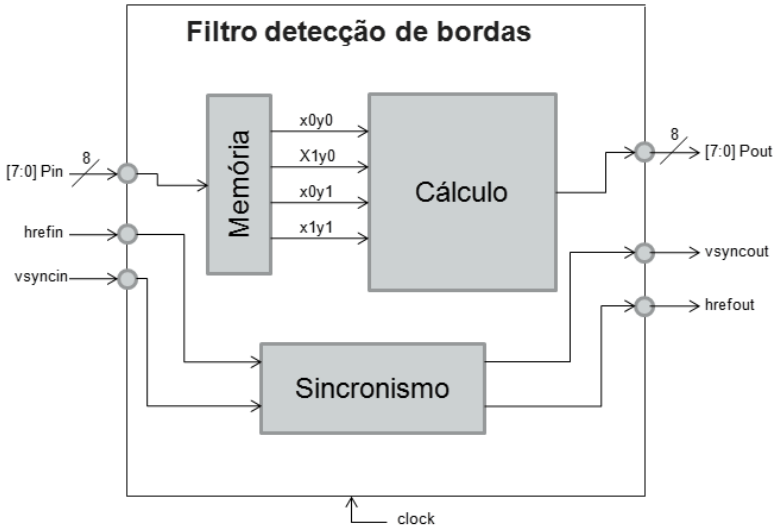

Figura 3.4: Representação da arquitectura do filtro detecção de bordas.

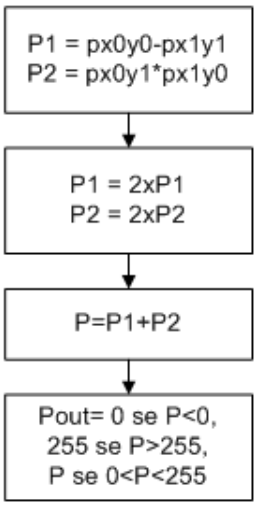

Figura 3.5: Máquina de estados do cálculo.

que de certa forma correspondem às bordas. O efeito deste filtro pode ser visualizado na fig 3.6 que são imagens retiradas da implementação deste filtro.

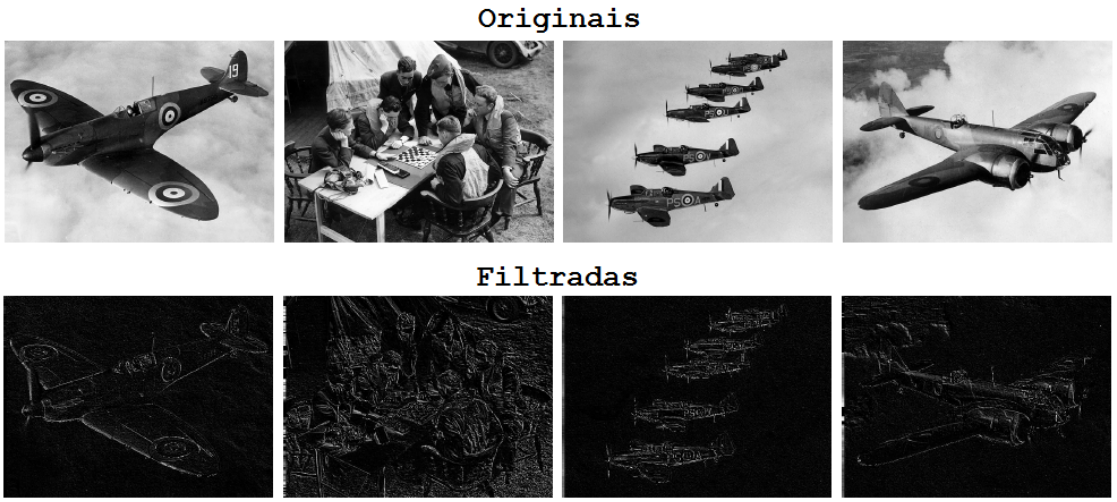

Figura 3.6: Amostra de imagens processadas pelo filtro de detecção de bordas.

3.2.3 Filtro de convolução

O módulo a desenvolver para este filtro é composto por três blocos: um de memória que armazena e disponibiliza o pixel a calcular e os seus vizinhos; um para o cálculo e outro que efectua o sincronismo da imagem, uma vez que haverá um atraso criado pela memória e pelo cálculo, tal como no filtro explicado anteriormente. A figura 3.7 mostra como estes blocos estão relacionados.

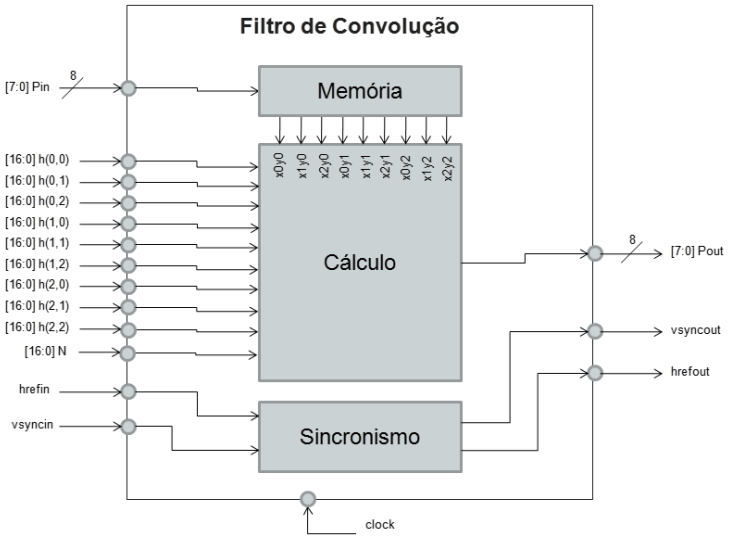

Figura 3.7: Representação da arquitectura do filtro de convolução

O módulo funciona com uma mascara de tamanho 3x3. Assumindo x e y, a posição do pixel a calcular, segundo a convenção dos eixos, como foi descrito no capítulo sobre representação de imagens digitais, a sua relação com a matriz convolução, neste caso representada por matriz h, será segundo a seguinte formula:

$$Pout(x,y) = \frac{1}{2^N} \cdot \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(x+i, y+j) \cdot h(i,j) \tag{3.4}$$

O valor a dividir na formula, deverá ser o valor mais próximo do somatório dos valores da matriz h, como foi explicado na secção 2.4.2. A divisão feita desta forma será muito mais simples em termos de implementação de *hardware* pois traduz-se num *shift-register*.

O cálculo funciona como uma máquina de estados, de forma a explorar e aproveitar o facto de haver disponibilidade de oito ciclos de relógio para que o resultado esteja disponível. Poderiam ser usados menos ciclos e o cálculo ficar mais rápido, mas desta forma e, uma vez que, o próximo cálculo só inicia após oito ciclos na mesma, são usados menos recursos relativos ao espaço e consumo na FPGA. Como se pode ver no esquema representado na figura 3.8 as nove multiplicações entre os pixéis da máscara e os da imagem estão distribuídas pelos oito estados disponíveis para o cálculo. O resultado vai sendo acumulado num registo, no penúltimo ciclo faz-se a divisão usando

um simples *shift-register* e por fim, no último ciclo, é verificado se o valor calculado sofreu um *over-flow* e corrige limitando os valores superiores e inferiores válidos para um pixel.

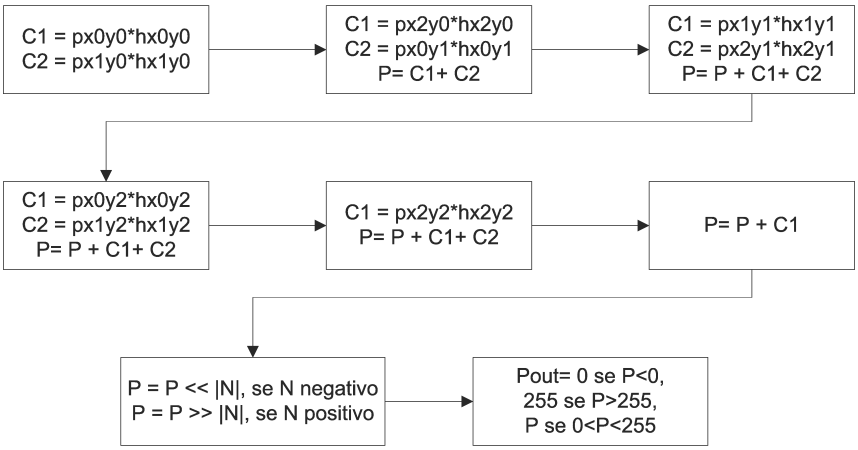

Figura 3.8: Representação da máquina de estados no módulo cálculo.

Os efeitos deste filtro podem ser os mais variados. As amostras deste filtro apresentados na figura 3.9 são apenas algumas das mais usuais configurações neste tipo de filtros, que podem ser consultadas na secção 2.4.2.

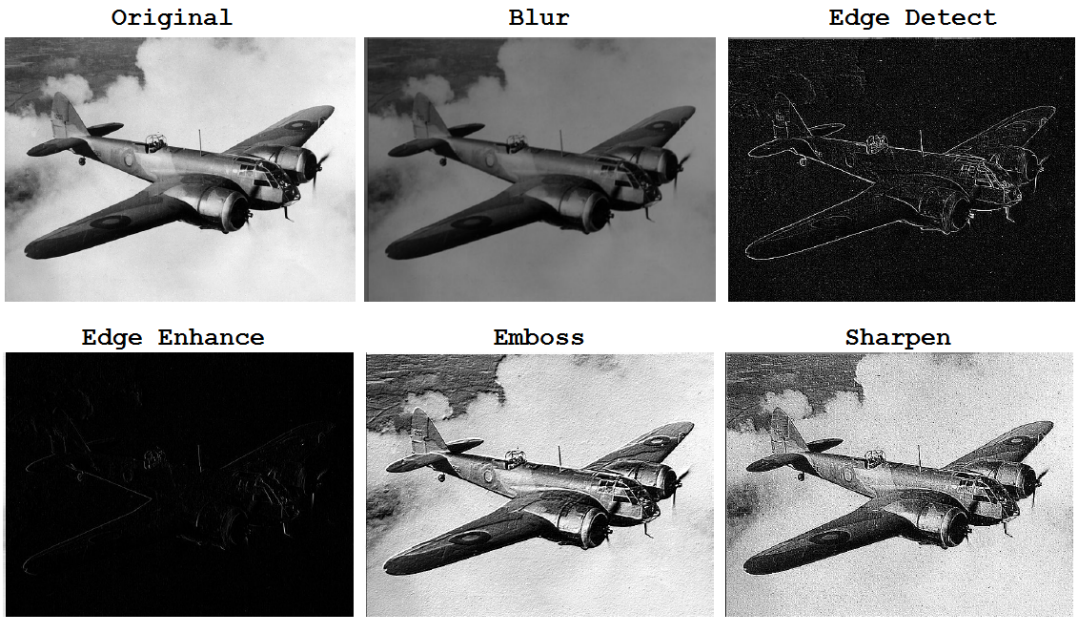

Figura 3.9: Amostra de imagens processadas pelo filtro de convolução.

3.2.4 Filtro morfológico

Este filtro baseia-se apenas na troca do pixel a calcular pelo próprio ou por um dos seus vizinhos. Os nove píxeis de uma matriz de tamanho 3x3 são ordenados de forma crescente e troca-se o pixel central por um dos vizinhos, escolhendo o pixel de valor mais baixo, mais alto ou mediano. A troca do pixel central pelo pixel de valor mais baixo resulta numa imagem com efeito de erosão, gerando assim uma imagem mais escura. Já a escolha do pixel de valor mais alto a imagem resulta num efeito de dilatação, gerando uma imagem mais clara. A escolha do valor mediano de entre os valores da matriz resulta numa eliminação de possível ruído que a imagem possa ter. A arquitectura deste filtro, a nível de blocos será idêntico ao anterior, pois usa um bloco de memória igual para ter o acesso aos píxeis vizinhos e consequentemente um bloco de sincronismo também igual pois o atraso é o mesmo. Terá então o aspecto representado no diagrama de blocos da figura 3.10.

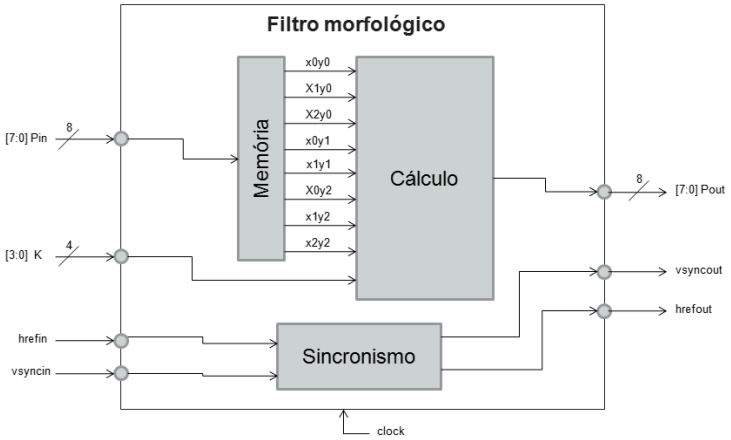

Figura 3.10: Representação da arquitectura do filtro baseado em morfologia

Na implementação deste projecto, é dada a possibilidade ao utilizador de escolher, para além da mediana, o nível de erosão ou dilatação. Assim, para a troca do pixel central, basta escolher o valor que se pretende na sequência ordenada.

Tabela 3.2: Sequência ordenada

Valor mais baixo	...	...	...	Mediana	...	...	...	Valor mais alto
k=0	...	...	...	k=4	...	...	...	k=8

Para a escolha do algoritmo de ordenação é preciso ter em atenção dois aspectos importantes, o número de iterações necessárias para obter os valores do vector ordenado, que têm de ser menores ao iguais aos oito ciclos disponíveis e a complexidade que se pode traduzir numa exigência de recursos elevada. Deste modo, o algoritmo que melhor encaixou neste perfil foi o *Odd-Even Sort* [37] que se trata de uma versão do *Bublle Sort* que funciona em modo *pipeline*. Na figura 3.11 podemos ver uma demonstração do seu funcionamento.

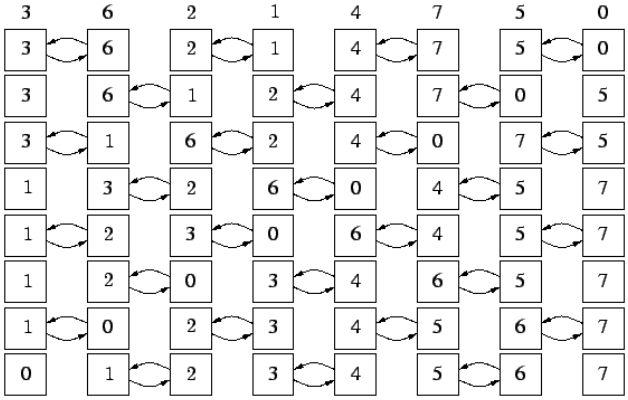

Figura 3.11: Representação do funcionamento do algoritmo *Odd-Even Sort*

Para a implementação deste algoritmo no módulo, foi apenas necessário criar nove registos, um para cada pixel, e quatro comparadores, o que significa uma lógica combinacional muita pequena e por sua vez uma baixa necessidade de recursos. Para que o vector fique completamente ordenado, o número de iterações necessárias é igual ao tamanho do vector, que neste caso é nove. Uma vez que o processamento é em tempo real e só são disponibilizados oito ciclos de relógio para cálculos entre píxeis, neste projecto o resultado com que se trabalha neste algoritmo é o obtido após a oitava iteração, que corre o risco de não ser o mais correcto, mas a probabilidade de tal acontecer é baixa e mesmo nesse caso o resultado será o valor imediatamente anterior ou posterior da ordenação final.A figura 3.12 ilustra a arquitectura da máquina de estados do módulo cálculo mostrando as distribuição das comparações pelos oito estados.

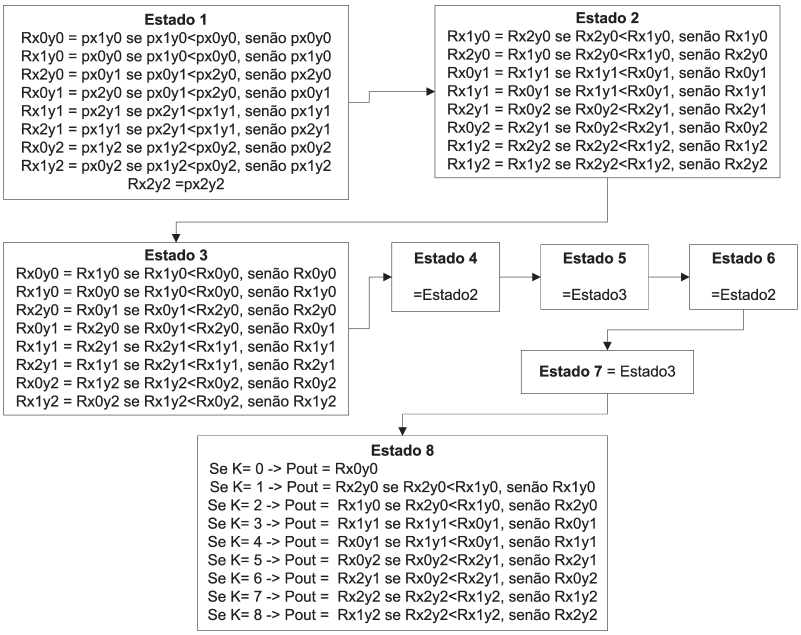

Figura 3.12: Arquitectura da máquina de estados do módulo calculo, no filtro morfológico

Como podemos ver na imagem, a selecção do tipo de filtro para mediana, erosão ou dilatação, é definida através do valor k que selecciona um dos valores dos píxeis ordenados. Esse valor é introduzido pela porta série com o interface gráfico descrito em 4.1. Os efeitos deste filtro podem ser vistos na figura 3.13 que é uma amostra de imagens processadas pelo mesmo.

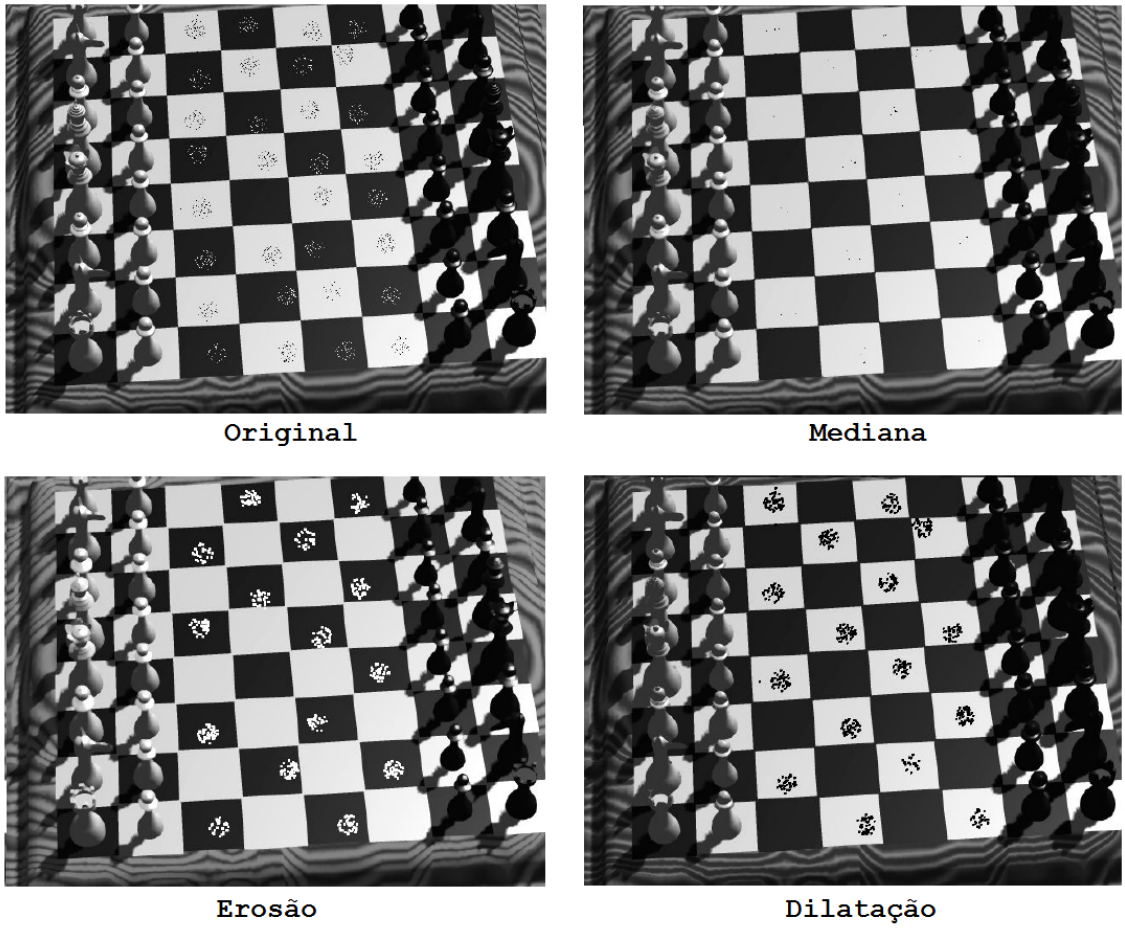

Figura 3.13: Amostra de imagens processadas pelo filtro Morfológico.

3.2.5 Memória

Alguns dos filtros que são implementados neste projecto necessitam do acesso, ao mesmo tempo, a vários píxeis de uma imagem, nomeadamente o pixel central a ser substituído após a filtragem e os píxeis vizinhos. No caso dos filtros que usam a matriz 3x3 optou-se por implementar um módulo que armazena os píxeis necessários de forma a dar acesso continuamente aos nove píxeis necessários para o cálculo. A figura 3.14 ajuda a perceber qual a informação que é necessário guardar assim como a que deve disponibilizar na saída. O pixel representado a vermelho é o que será calculado, a azul claro estão os píxeis vizinhos envolvidos no cálculo, a azul-escuro e a verde estão os píxeis que são precisos serem guardados para que a memória consiga disponibilizar os nove píxeis num processo contínuo.

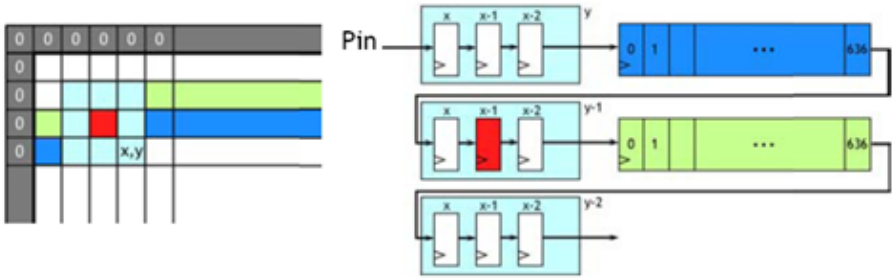

Figura 3.14: Representação do bloco memória

A melhor forma de guardar os píxeis representados a azul-escuro e verde, uma vez que o valor destes não precisa estar sempre disponível, é através de *shift-registers* [38]. Desta forma o espaço requerido para a implementação deste bloco torna-se muito menor do que por exemplo usando registos. De notar que a matriz só terá disponível à saída os nove píxeis após ter armazenado duas linhas e dois píxeis, pelo que implica um atraso equivalente para efectuar o calculo. No caso dos cantos da imagem, os valores vizinhos, ou seja, fora da imagem, vão assumir o valor zero, que eventualmente poderá provocar um contorno à volta da imagem à saída dos filtros indesejada.

3.3 Memória externa

Trata-se de um módulo que basicamente permite guardar imagens nas duas memórias estáticas [39] que estão implementadas na Saprtan3 e reproduzi-la como se de uma segmentação de vídeo se tratasse. Normalmente uma das imagens serve de entrada para os filtros, dando assim a possibilidade de os testar com imagens fixas e/ou provenientes de outra fonte que não a da câmara. A outra imagem, é uma *frame* guardada à saída dos filtros permitindo assim reter uma amostra dos seus resultados. Estas memórias possuem uma estrutura com 18 linhas de endereço de 16bit cada, perfazendo assim um total de 512Kbytes de memória em cada uma delas. Por sua vez as imagens a guardar necessitam de 300Kbytes uma vez que é necessário guardar 307200 (640x480) píxeis de 8bits. Facilmente se percebe que há memória disponível para 3 imagens, mas por motivos de simplificação na implementação, este projecto guarda apenas uma imagem em cada memória, sendo uma à entrada dos filtros e outra à saída. A figura 3.15 mostra todas as entradas e saídas deste módulo.

O funcionamento tanto para ler como escrever na memória baseia-se num contador cujo valor guardado no seu registo está ligado directamente ao barramento de endereços das duas memórias. Assim a posição é incrementada a cada chegada de dois píxeis e volta a zero quando é identificado o último pixel da imagem, através dos barramentos X e Y que indicam a posição do pixel. O intervalo de dois ciclos existe porque cada endereço armazena 16bit, neste caso, 2 píxeis de 8bit. Este modo de funcionamento, permitiu a separação das leituras e escritas em ciclos diferentes. Assim, haverá sempre leituras em ambas as memórias nos ciclos ímpares e escritas nos ciclos pares, caso estas sejam activadas. A activação de escrita funciona através de uma flag para cada

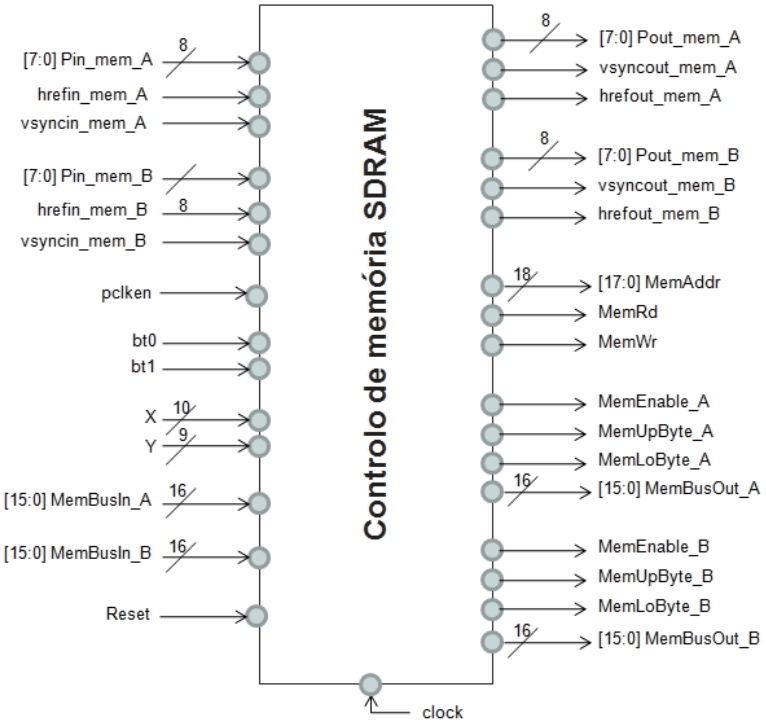

Figura 3.15: Representação do módulo usemem

imagem, que é activada no primeiro pixel da imagem quando é premido um botão correspondente e desactivada no último pixel da imagem. Mediante a flag activada só a memória correspondente é activada através do *chip enable*, não desfazendo assim os dados da outra memória.

Quanto aos tempos de acesso, segundo a *data sheet* da memória [39], tanto a leitura como a escrita só pode acontecer 10ns após a sua ordem ou actualização de endereço, tempo que é exactamente o mesmo de um ciclo de relógio neste projecto.

3.4 Encadeamento de filtros

Após ter um projecto funcional com um filtro o próximo passo foi refazer o projecto com a combinação de uma cadeia de filtros diferentes. O resultado são novos filtros, como já foi referido na secção 2.4.3 os filtro *Opening* e *Closing*, alargando assim o leque de filtros disponíveis e por sua vez a qualidade da informação que se pretende extrair das imagens. A figura 3.16 ilustra o exemplo de uma combinação possível. Como podemos ver, a saída de píxeis e sinais de sincronismo de um filtro está ligada aos da entrada do filtro posterior. Todos os restantes sinais são ligados directamente a cada filtro.

A única limitação na implementação de uma cadeia de filtros é o espaço exigido na FPGA. Foi então feito um estudo dos recursos de cada filtro e do projecto sem filtros que exigem em quantidade de *slices*. A tabela 3.3 dá uma ideia de espaço na FPGA que cada filtro exige, assim como o projecto sem filtros. A capacidade da FPGA também é apresentada de forma a poder-se

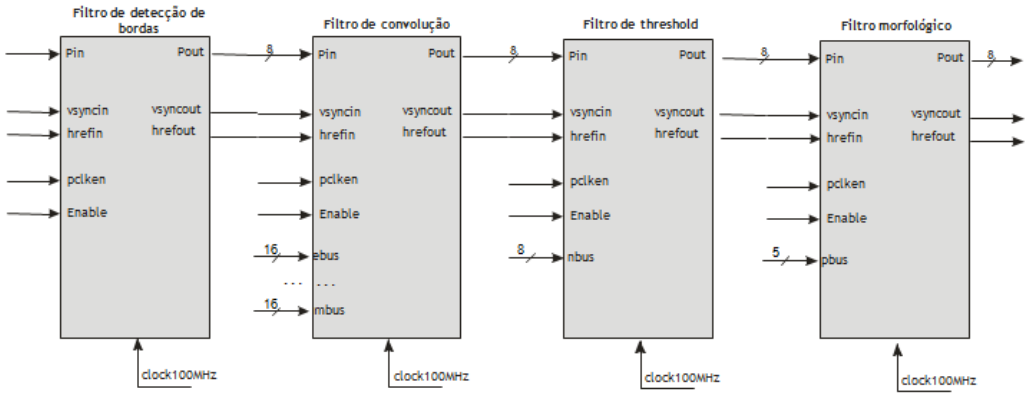

Figura 3.16: Exemplo das ligações de uma cadeia de filtros

definir o que pode ser inserido no projecto. Após a avaliação da tabela, facilmente percebe-se que a limitação nas combinações devido aos recursos exigidos é grande, pois apenas restam 1139 *slices* na FPGA para uma cadeia de filtros.

Tabela 3.3: Espaço exigido pelos módulos em slices na FPGA

Módulo ou projecto:	Slices	Percentagem de utilização
Capacidade da spartan3	1920	100 %
Filtro de convolução	664	34,6 %
Filtro morfológico	778	40,5%
Filtro de bordas	325	16,9%
Filtro de Threshold	10	0,5%
Projecto sem filtro	781	40,7%

Os filtros de convolução e morfológico são relativamente grandes e nunca poderia ser testada uma cadeia que envolvesse os dois ou a repetição de um deles. Desta forma dividiu-se o projecto em dois e foi implementado em duas placas. Assim a captura de imagem passou a ser feita numa placa e a reprodução na outra. Além do tamanho do projecto, que sem filtros é menor em cada placa, há possibilidade de colocar filtros em ambas as placas. Da tabela 3.4, podemos concluir que a placa que faz a captura deixou um espaço de 714 *slices* para filtros e a que faz a reprodução um espaço de 538 *slices*, também para filtros. A ligação entre placas é feita através de um barramento que liga os píxeis e sinais de sincronismo da saída de um filtro numa placa à entrada de outro filtro na outra placa. Para além destes sinais é necessário enviar também dois sinais de relógio do pixel (cam_pclken e cam_pclkout) para haver sincronismo de píxeis nos filtros das duas placas e nas FIFOs do módulo de vídeo. Para que os sinais de relógio também estejam sincronizados foi necessário alterar os módulos DCM de ambos os projectos. O sinal de relógio passa a ser gerado apenas numa das placas, neste caso é a que faz a aquisição de imagem, e é enviado pelo barramento para a outra placa. Como mostra a figura 3.17, o DCM desta gera os sinais de relógio

necessários através deste sinal recebido e envia novamente para a placa que gera o sinal original, um sinal de *feedback* de forma a que ambas as placas fiquem com o sinal de relógio sincronizado, uma com a outra.

Tabela 3.4: Espaço exigido pelos módulos em slices nas duas FPGAs

Modulo ou projecto:	Slices	Percentagem de utilização
Projecto sem filtros na placa com aquisição	714	37,2%
Projecto sem filtros na placa com reprodução	538	28%

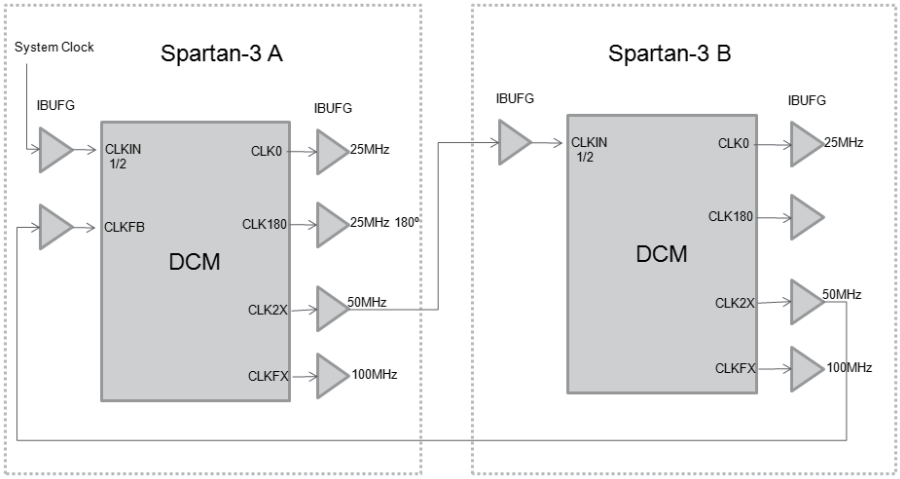

Figura 3.17: Ligação dos módulos DCM entre duas placas Spartan-3.

Os dados de posição do pixel, designados no projecto por X e Y, deveriam ser também enviados pelo barramento para que o módulo que grava as imagens nas memórias funcionem. Contudo só para estes dois sinais seriam necessárias 19 linhas do barramento e como não há disponibilidade para tal, foi criado um módulo que a partir dos sinais X e Y, são gerados apenas dois sinais que são *flags* que sinalizam o inicio e o fim de uma imagem. Com isto, o módulo (usemem) que grava as imagens nas memórias teve que ser alterado de forma a funcionar com estas *flags* e não com os sinais de posição.

Para evitar problemas devido à possibilidade do tempo de propagação no barramento ser diferente de umas ligações para outras, os sinais de sincronismo e píxeis são novamente sincronizados através de um novo módulo (sinc.v) que possui registos síncronos com o relógio de 100MHz. Este procedimento provoca o atraso de um ciclo de relógio mas que é comum a todos os sinais.

A tabela 3.5 mostra os dados que são enviados pelo barramento que une as placas e os pinos que são usados para tal.

Com uma cadeia de filtros a capacidade de processamento de imagem tornou-se maior, ainda para mais, porque uma ordem diferente dos mesmos filtros numa cadeia dá origem a resultados diferentes. Seguem então apenas algumas amostras de imagens de cadeias de filtros. A figura 3.18

Tabela 3.5: Barramento de ligação entre placas

Pino IDC-40p	Dados
4	relógio de pixel disponível (pclken)
5	sinal de sincronismo horizontal (href)
6	sinal de sincronismo vertical (vsync)
7,8,...,14	pixel (P)
16	relógio (clk)
17	sinal *feedback* do relógio (clkfb)
18	flag inicio de imagem
19	flag fim de imagem
21	relógio de pixel disponível para módulo de vídeo (cam_pclkout)

mostra uma sequência de imagens que resulta de uma cadeia de filtros em que o primeiro filtro é a detecção de bordas, de seguida a imagem é filtrada com o filtro de *threshold* salientando assim o efeito das bordas numa imagem binária e por fim passa num filtro morfológico configurado para funcionar como mediana para eliminar o ruído.

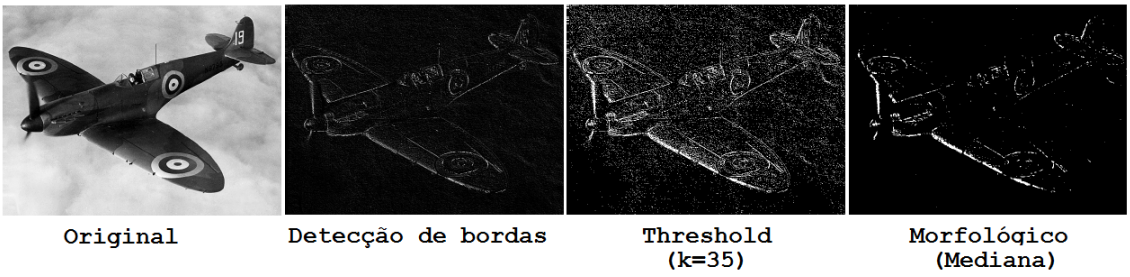

Figura 3.18: Amostra de imagem filtrada pela cadeia de filtros de detecção de bordas, *threshold* e morfológico.

A cadeia ilustrada na figura 3.19, é constituída apenas por dois filtros, um de convolução configurado para funcionar como *Edge Detect* e o seguinte, um de *threshold* para salientar as bordas.

A cadeia ilustrada na figura 3.20 mostra o resultado da combinação de dois filtros morfológicos. Na primeira linha, o filtro configurado para dilatação seguido de um outro configurado para erosão resulta num efeito *Opening*. Na segunda linha, um filtro configurado para erosão seguido de um outro configurado para dilatação, resulta num efeito *Closing*.

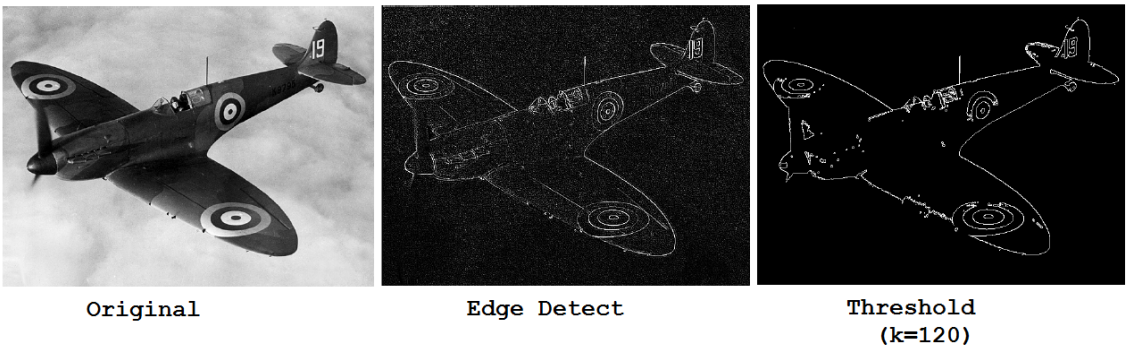

Figura 3.19: Amostra de imagem filtrada pela cadeia de filtros de convolução e *threshold*.

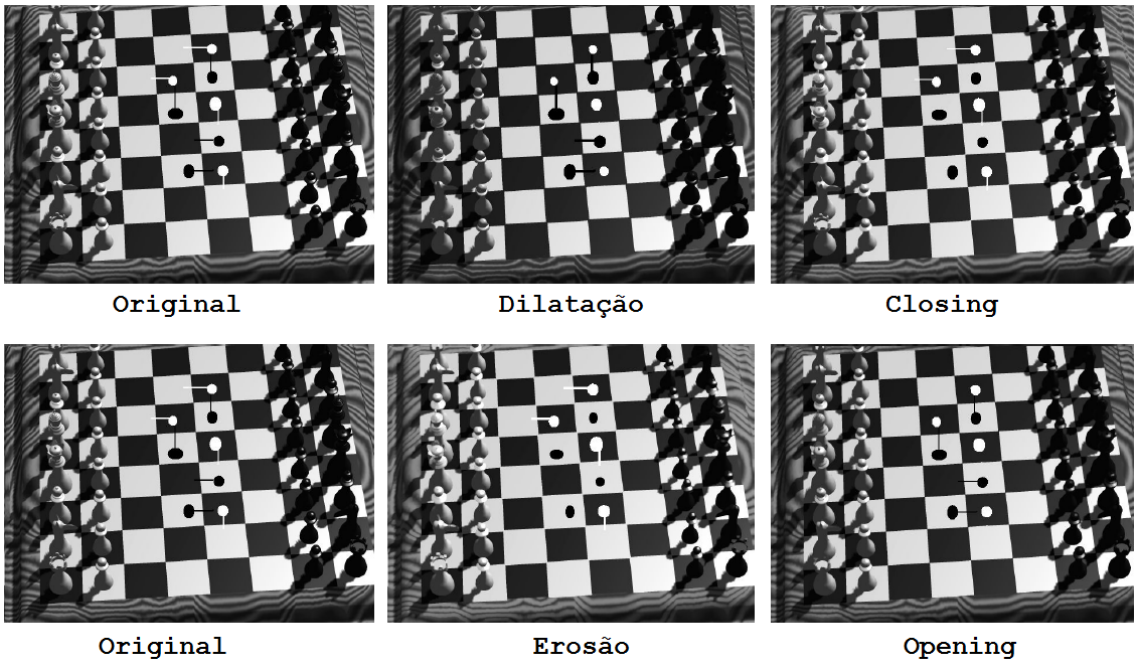

Figura 3.20: Amostra de imagem filtrada pela cadeia de filtros morfológicos.

Capítulo 4

Interface com os módulos

Neste capítulo é apresentado um programa de interface desenvolvido para comunicação com a FPGA através da porta série. O seu objectivo principal é o envio de valores para os filtros na FPGA de forma a configurá-los. São também descritas as mudanças necessárias no projecto da FPGA para permitir este interface.

Também descreve o processo de enviar/guardar e ler imagens que se encontram nas memórias SRAM que estão inseridas na placa Spartan-3. Este processo surgiu da necessidade de uso de imagens fixas e com características definidas para testar os filtros de modo a haver termos de comparação. Esta implementação, permitiu também a possibilidade de guardar amostras de imagens processadas pelos filtros.

As ferramentas que foram necessárias para esta fase do projecto foram as seguintes:

- **HDD Free Serial Port Monitor** - Este programa permitiu visualizar as transmissões na porta série durante a fase de implementação e teste da interface entre o computador e a FPGA. Disponível em [40].

- **Microsoft Visual Basic 2005 Express Edition** - Com este software foi criado o programa que irá funcionar num computador e que permite fazer a interface que comunica com a FPGA através da porta série. Este software está disponível na página oficial da Microsoft.

4.1 Programa de Interface

Trata-se de um programa desenvolvido em ambiente Microsoft Visual Basic 2005 Express Edition para computadores com o Windows como sistema operativo. A sua função é comunicar através da porta serie com a FPGA usando para tal um protocolo pré estabelecido na FPGA assim como no programa.

Nos pontos a seguir é descrito em pormenor o funcionamento de cada uma das funcionalidades deste programa.

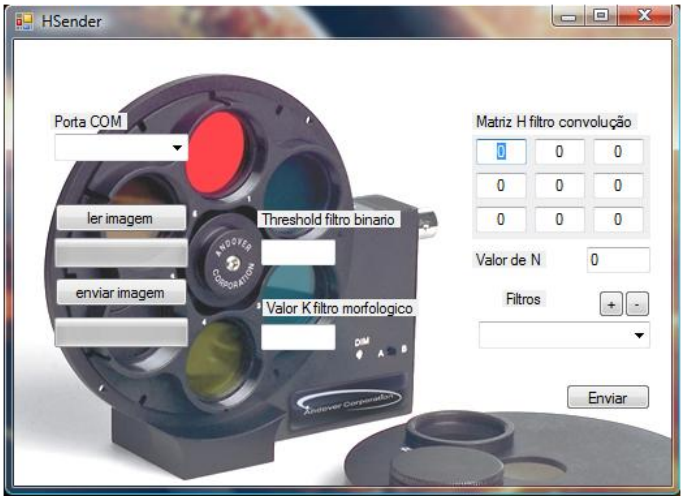

Figura 4.1: Programa de interface com o projecto

4.1.1 Envio de valores para os filtros

A principal funcionalidade deste programa é permitir ao utilizador alterar valores que influenciam o comportamento dos filtros na FPGA, sem que esta pare o seu funcionamento. Deste modo o utilizador pode alterar o tipo de filtro que pretende mudando apenas a configuração deste. No caso do filtro de convolução o programa envia os valores da matriz H e de N, que é calculado automaticamente pelo programa, mas com a possibilidade do utilizador o poder alterar. Para o filtro binário o programa envia o valor de *threshold* entre 0 e 255 e para o filtro morfológico o programa envia o valor da posição, entre 0 e 8, do vector dos valores ordenados, da qual é escolhido o pixel calculado. Em todos estes casos o protocolo de transmissão é o mesmo, onde são enviados 24 bits por cada valor a enviar, o formato da trama está ilustrado na tabela 4.1.

Tabela 4.1: Protocolo de envio de dados para filtros

Tamanho:	4bits	4bits	16bits
Significado:	Comando	Identificação	Valor

4.1.2 Guardar configuração de matrizes H

Para que o utilizador não tenha que escrever todos os valores da matriz H sempre que a quiser alterar, o programa permite guardar num ficheiro, várias combinações da matriz e atribuir-lhes um nome. Desta forma, o utilizador caso tenha já gravada uma lista de configurações, sempre que quiser mudar a que está na FPGA, basta escolher uma das configurações e clicar enviar. Para gravar basta introduzir os valores na matriz, atribuir um nome e carregar no botão ”+”. Para remover basta escolher a configuração previamente guardada e carregar no botão ”-”.

4.1.3 Importação/Exportação de imagens para a memória

Uma vez criado o módulo memória descrito na secção 3.3, que permite reproduzir imagens que estejam guardadas na memória SDRAM da placa, o programa hsender permite enviar pela porta série uma imagem de modo a trocar a que está gravada na memória que faz a reprodução para os filtros, designada por memória A. E para que haja a possibilidade de extrair da FPGA uma imagem que resulta dos filtros, o programa, também pela porta serie, consegue ler o conteúdo da imagem, previamente guardada pelo módulo, na memória à saída dos filtros, designada por memória B. Estas imagens podem ser guardadas no computador com dois formatos possíveis, sendo eles BMP e PNG. As tramas do protocolo que foi implementado, tanto no programa de interface como no módulo unisepav4, que recebe as tramas na FPGA e comunica com os filtros, funcionam com o seguinte formato representado na tabela 4.2.

Tabela 4.2: Protocolo de envio/recepção de imagens

Tamanho:	4bits	4bits	20bits	16bits	16bits	...
Significado:	Comando	xxxx	Tamanho	Dados	...	...

O módulo unisepav4, representado na figura 4.2, é quem recebe os dados da porta série e interpreta os comandos. Para adicionar as funcionalidades de importação e exportação de imagens foi necessário acrescentar novas entradas e saídas no módulo, nomeadamente para fazer as ligações dos sinais de dados e controlo às memórias. Uma vez que as ligações às memórias não podem ser feitas a vários módulos ao mesmo tempo, neste caso o usemem referido na secção 3.3 e o unisepav4, foi acrescentada uma nova saída ao módulo unisepav4, que se trata de uma flag e indica a necessidade de acesso deste módulo à memória. Assim quando esta flag está activa as ligações de acesso à memória pertencem ao módulo unisepav4, permitindo assim a importação ou exportação de imagens, todo o resto do tempo as ligações são feitas ao módulo usemem. Infelizmente este processo não permite o funcionamento dos filtros, com uso da memória, enquanto ocorre uma importação ou exportação de imagens. O processo de leitura e escrita é facilmente interpretado pelos diagramas das figuras 4.3 e 4.4.

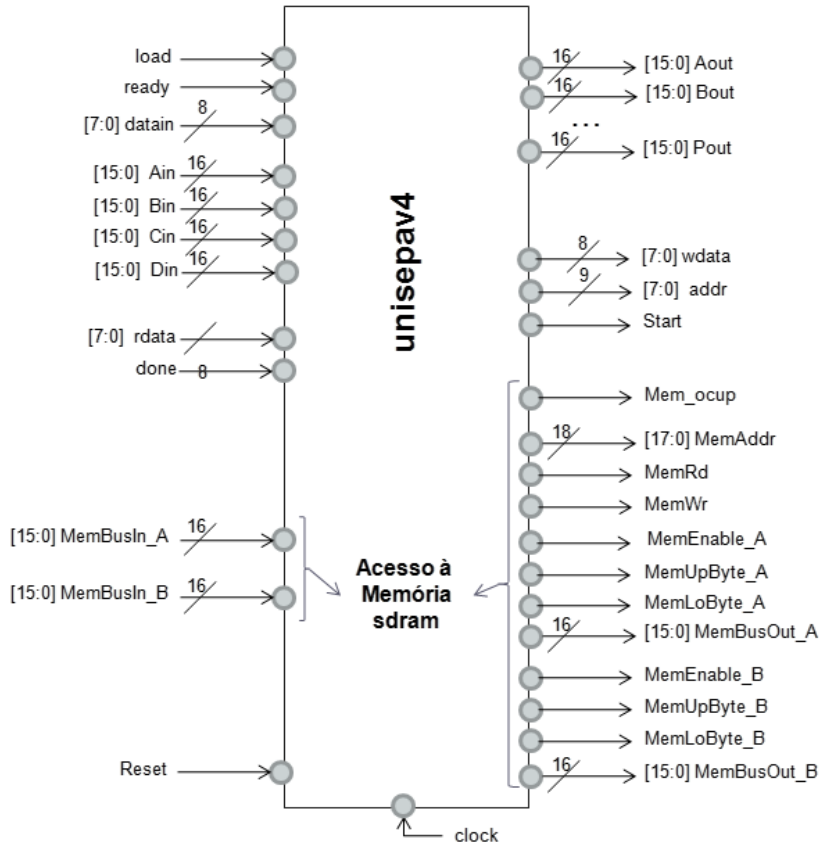

Figura 4.2: Módulo unisepav4

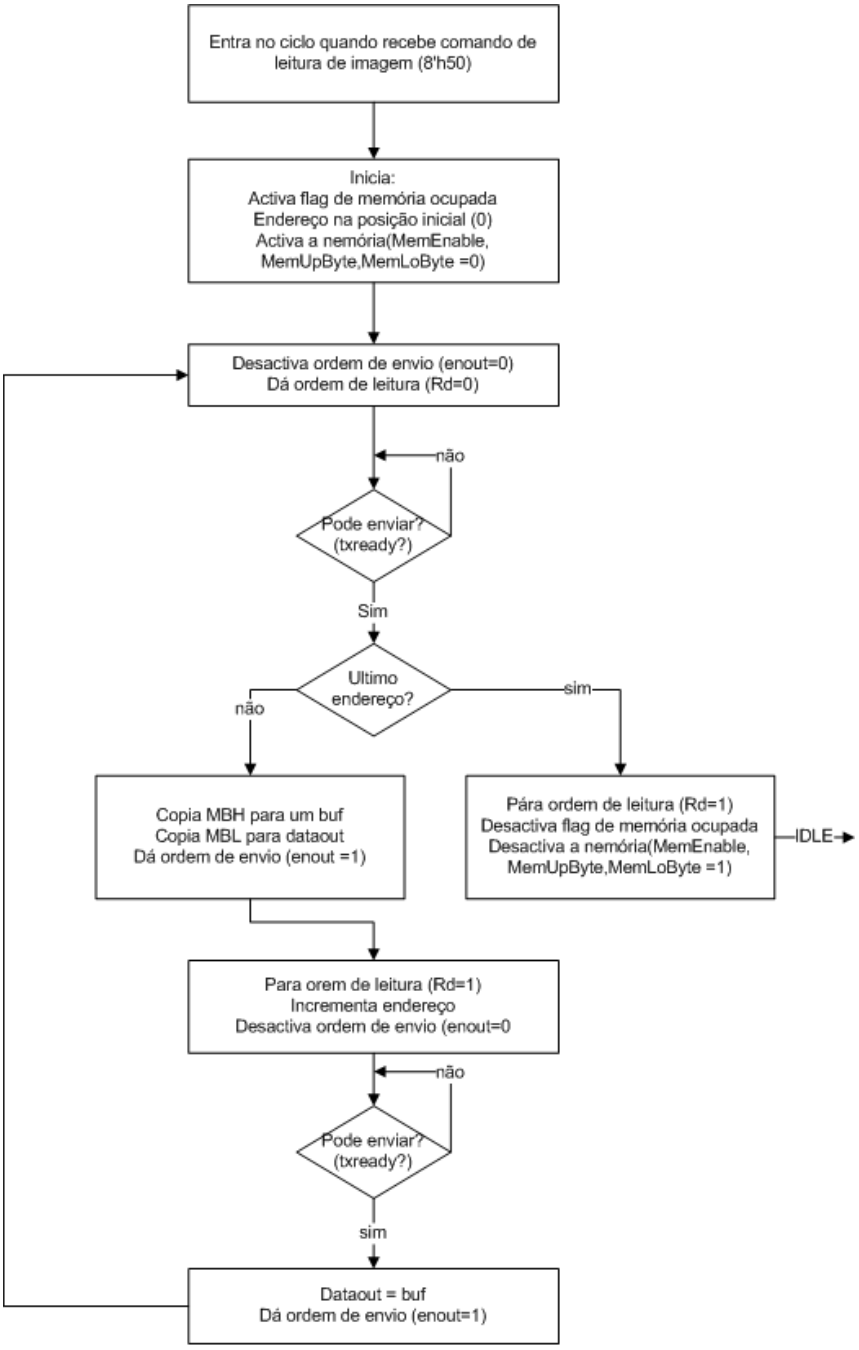

Figura 4.3: Diagrama de funcionamento da leitura de uma imagem na memória e envio pela porta

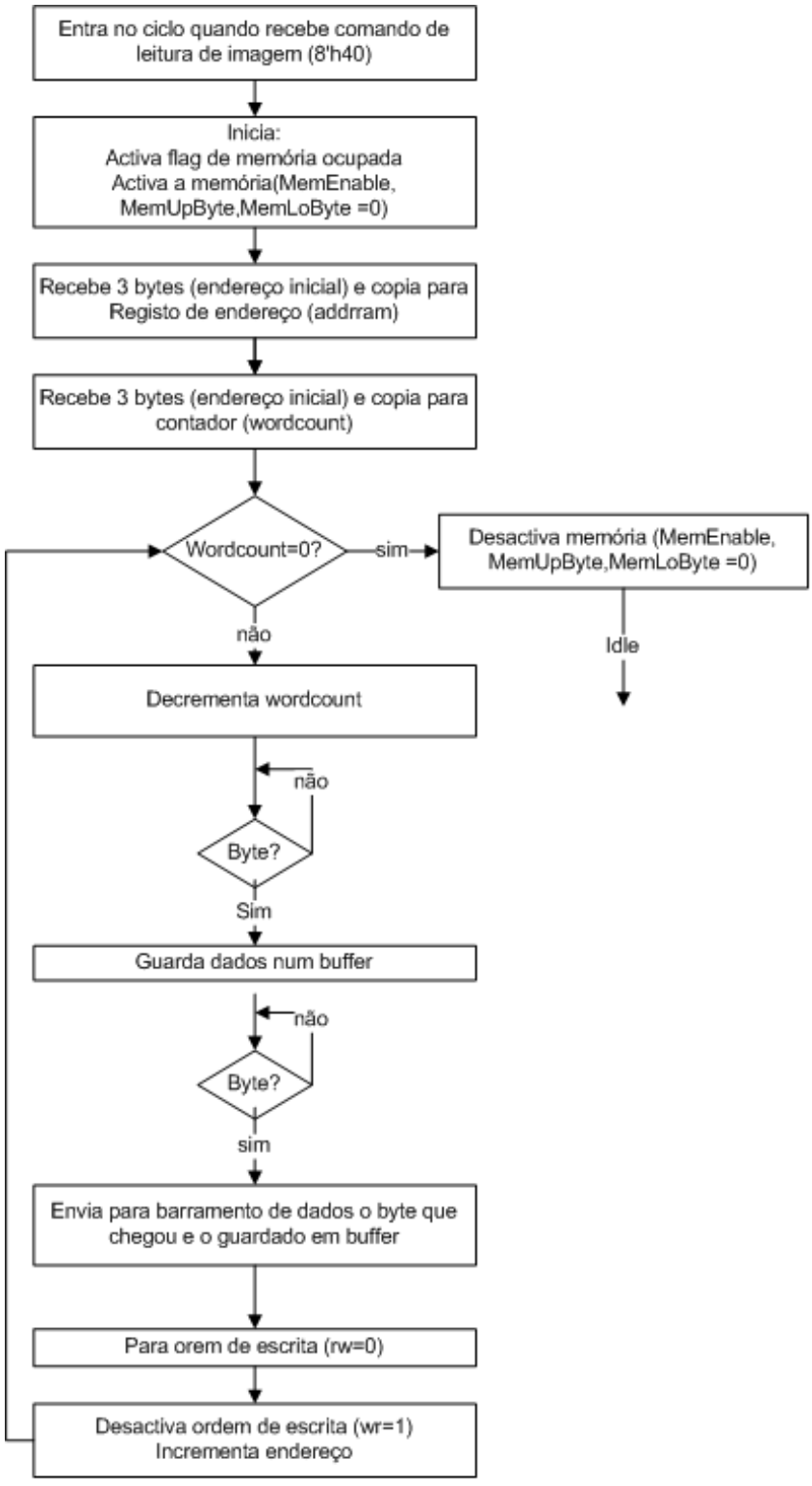

Figura 4.4: Diagrama de funcionamento da escrita de uma imagem na memória recebida pela porta

Capítulo 5

Reconfiguração Dinâmica

Neste capítulo será explicado o processo de migração do projecto que foi elaborado na placa Spartan3 para a placa Virtex II Pro, a alteração aos módulos de forma a que estes sejam suportados pela reconfiguração dinâmica e por fim explicar o processo de implementação desta.

Esta mudança é necessária para implementar a reconfiguração dinâmica, pois o dispositivo Spartan3 não a suporta. À partida, este processo parece simples mas ele envolve diversas alterações, tais como a do ficheiro ".ucf", que trata da atribuição dos pinos da FPGA; a adaptação das frequências do relógio através do DCM, uma vez que estas são baseadas no relógio de sistema da placa que neste caso é diferente; adaptação dos níveis de tensão nas ligações aos periféricos, sendo eles a câmara e o módulo de saída de vídeo para o monitor.

O passo seguinte foi o da reestruturação dos módulos, para que estes permitam a reconfiguração dinâmica. O projecto é então remodelado de forma a deixar uma área estática e apenas uma área dinâmica para implementar um dos filtros da biblioteca, podendo esta ser reconfigurada dinamicamente para qualquer outro filtro.

As ferramentas que foram necessárias para esta fase do projecto foram as seguintes:

- **Xilinx ISE Design Suite 9.2** - Idêntico ao primeiro software referido em 3.1, mas para esta fase do projecto, em que se implementa a reconfiguração dinâmica, foi necessário usar esta versão anterior.
- **Xilinx PlanAhead 10.1** - Usado na fase de implementação com suporte de reconfiguração dinâmica, este programa gera o .bit file final e selecciona as áreas dinâmicas. também ele é fornecido pela Xilinx.
- **Virtex-II Pro Development System** - [41] trata-se de outra placa, que podemos ver na figura 5.1, cujo modelo é mais avançado que a Spartan-3 e suporta reconfiguração dinâmica. O seu chip é o XC2VP30.
- **Osciloscópio digital** - Usado para análise dos sinais entre a placa com a FPGA e os periféricos câmara e módulo de vídeo.

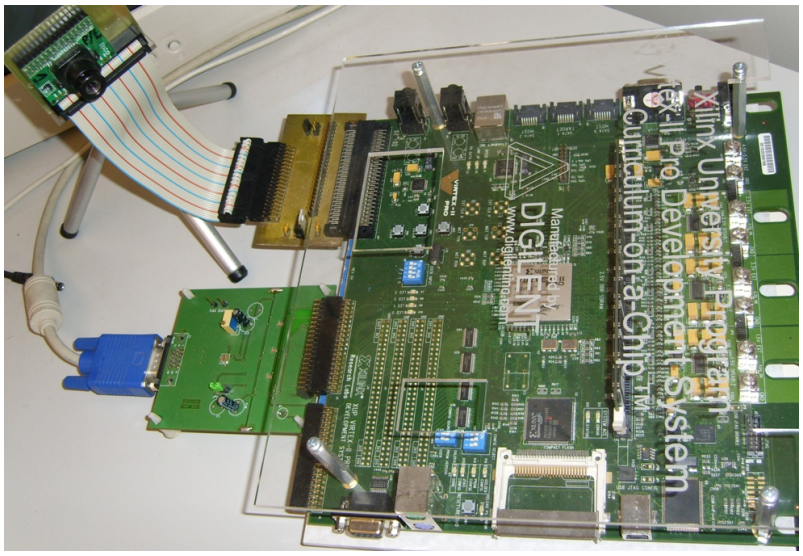

Figura 5.1: Fotografia do dispositivo Virtex-II Pro com os periféricos ligados

5.1 Migração de projecto

O primeiro passo para migrar o projecto para a nova placa Virtex II Pro, é a alteração do ficheiro ".ucf" que especifica as ligações entre os sinais do projecto lógico e os pinos da FPGA fisicamente. Uma vez que se trata de *hardware* diferente além da atribuição dos pinos aos sinais do projecto foi preciso retirar e acrescentar entradas e saídas do projecto. Consultado os esquemáticos da placa e os ficheiros ".ucf's" originais disponíveis na página oficial do fabricante, foi possível de acordo com o novo *hardware*, redefinir, acrescentar e remover entradas e saídas do projecto. Para além das redefinições dos pinos, esta alteração envolveu resumidamente os seguintes operações:

- Diminuição de 8 interruptores para 4;
- Aumento de 4 botões para 5;
- Diminuição de 8 leds para 4;
- Remoção de *displays*;
- Remoção de memória interna;
- Mudança de conectores de expansão;
- Inserção de conectores de expansão de alta velocidade.

Depois deste processo foi preciso fazer a inversão a alguns dos sinais no projecto que estão relacionados com o *hardware*, uma vez que na Spartan3 os sinais dos botões, leds e switchs são activos com o nível lógico alto e na Virtex II Pro, estes são activos com o nível lógico baixo. Nos

conectores de expansão, os níveis de tensão usados são iguais em ambas as placas, LVTTL, mas no caso da Virtex-II pro, os pinos possuem um circuito integrado de protecção que provoca uma descida destes valores. No caso do módulo de vídeo, esta diferença aparentemente não produz efeitos secundários, mas o mesmo não se pode dizer para a câmara, pois estes níveis estão muito perto dos níveis de decisão não permitindo o correcto funcionamento da mesma. A solução passou por usar um adaptador na porta de expansão de alta velocidade (J37), resultando assim num novo conector de 40 pinos idêntico aos J5 e J6, mas com os níveis de tensão LVTTL desejados. A tabela de tradução de pinos pode ser consultada no anexo A.

Outra diferença entre as duas placas é o relógio de sistema, do qual é gerado o *clock* de funcionamento para o projecto. Assim sendo, foi necessária a criação de um novo módulo DCM para adaptar o *clock* de sistema ao *clock* pretendido para o projecto. No caso da Spartan3 cujo *clock* de sistema funciona a 50MHz e é a partir deste sinal que o DCM gera 3 sinais com as frequências de 25MHz, 25Mhz com um desfasamento de 180 graus e 100MHz. Já na placa Virtex II Pro estes mesmos sinais são gerados mas a partir do *clock* de sistema que é de 100Mhz.

5.2 Projecto com suporte de reconfiguração dinâmica

Numa primeira fase o projecto apenas irá incluir uma área dinâmica onde poderá ser implementado qualquer filtro e uma estática onde será implementado todo o projecto sem filtros. A metodologia para a sua elaboração com suporte para reconfiguração dinâmica dos filtros foi retirada dos manuais da Xilinx [42, 43] e o exemplo de um caso pratico [44]. Para o desenvolvimento deste novo projecto optou-se por usar o VHDL no *top level* porque para a ligação entre as duas áreas são usadas *bus macros* e estas só podem ser implementadas nesta linguagem. O projecto antigo é instanciado como um módulo, para ser implementado como área estática, por isso são retiradas as instanciações dos filtros e acrescentadas as entradas e saídas para o acesso a eles. Para o acesso aos filtros é instanciado um módulo que não existe no projecto mas é implementado como área dinâmica para conter os filtros. Este módulo deve possuir todas as entradas e saídas exigidas por todos os filtros. As *bus macros* impõem às ferramentas de síntese uma localização fixa e definida de forma a permitir a comunicação entre ambas as áreas, isso implica que os módulos dos filtros tenham que ser alterados de forma a ficarem todos iguais do ponto de vista de entradas e saídas, ou seja, também devem possuir todas as entradas e saídas de todos os filtros. Para a síntese do projecto deverá ser usada a ferramenta ISE 9.2 com os pacotes do PR instalados e uma vez que falta o módulo referente à área dinâmica, é necessário indicar no ficheiro de configuração .ucf a indicação de que o módulo é reconfigurável, passando assim por cima do erro de síntese. Opcionalmente, pode-se manter a hierarquia " *soft"* na síntese para mais tarde haver a hipótese de escolher os lugares espaciais dos módulos que pertencem ao modulo estático, permitindo assim, a possibilidade de aproximar estes blocos dos extremos da FPGA e diminuindo a distancia aos pinos externos. Após ter o projecto sintetizado, o próximo passo foi sintetizar os filtros individualmente como se de projectos se trata-se para obter as suas *net lists*.

Como se pode verificar até aqui nenhum *bit stream* foi gerado, o projecto e os filtros apenas foram sintetizados com o objectivo de extrair as *net lists* do projecto e dos filtros. Usando agora o programa Planhead, é criado um projecto com o ficheiro .ucf e a *net list* que deve ser identificado como um projecto para reconfiguração parcial. Resumidamente no programa é preciso definir o módulo como reconfigurável e marcar a área na FPGA onde será implementado, assim como as *bus macros*, como ilustra a figura 5.2. Nesta fase é preciso ter o cuidado de definir uma área suficientemente grande e com multiplicadores de forma a poder envergar qualquer filtro. As *bus macros* devem colocadas do lado certo do módulo, uma vez estas devem corresponder ao sentido dos dados que foram definidos no projecto. Após este passo adiciona-se, no módulo reconfigurável, as *net lists* dos filtros e corre-se a preparação do *bit stream* no *ExploreAhead Runs*. Por fim é só gerar os *bit streams*, são eles o estático e um dinâmico para cada filtro. Uma vez que a informação da posição onde é implementado o circuito correspondente está no próprio *bit stream*, ou seja, para trocar o filtro que está em funcionamento, basta enviar o *bit stream* normalmente.

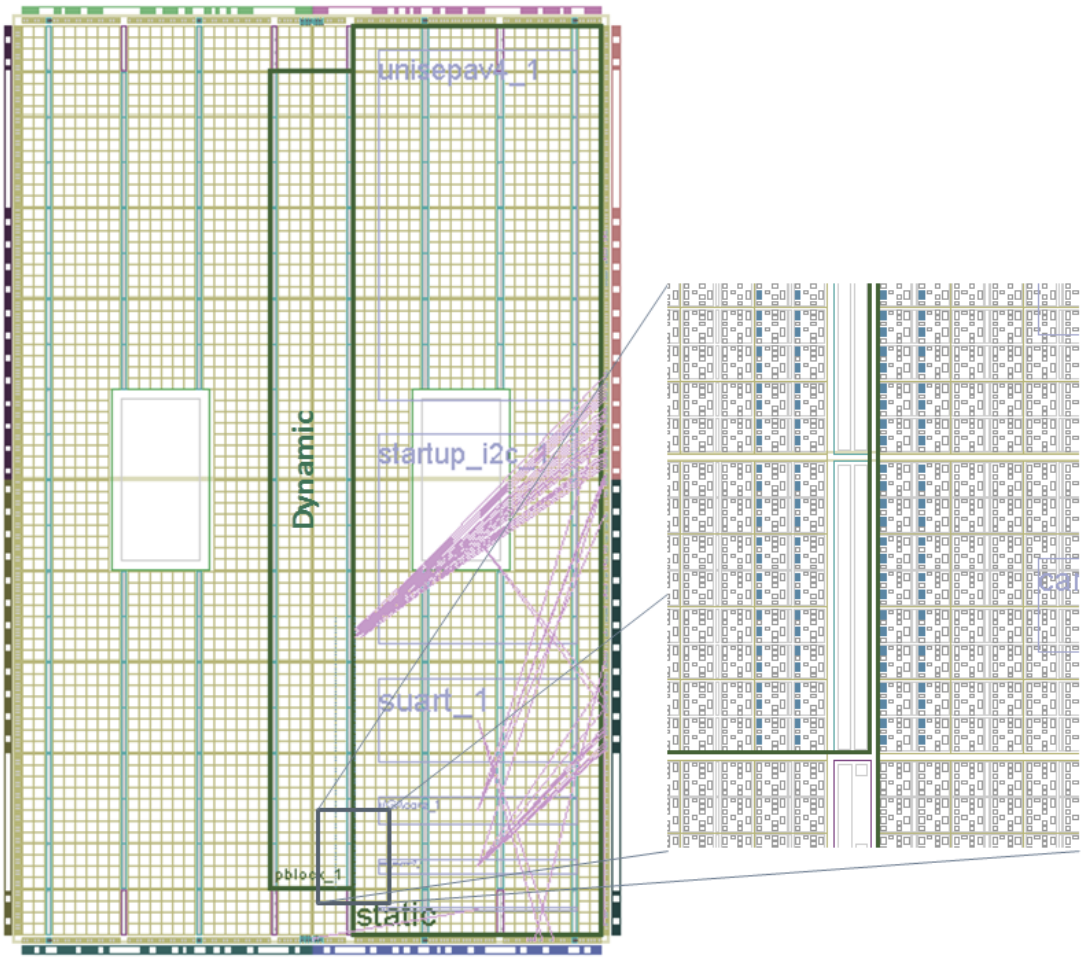

Figura 5.2: Ilustração da área dinâmica definida na FPGA. À direita, zoom que visualiza as *bus macros*

Capítulo 6

Conclusão e trabalho futuro

Conclusão

De certa forma os objectivos foram cumpridos, na medida em que foi criada uma biblioteca de filtros reconfiguráveis, que foram testados em três sistemas diferentes. Um sistema, foi apenas com o uso de uma placa Spartan3, na qual os filtros foram desenvolvidos e testados com resultados muito positivos, contudo com falta de espaço para uma possível cadeia de filtros. Foi então que se criou um segundo sistema, que consiste na ligação de duas placas spartan3 em série, de forma a distribuir o projecto e assim ganhar espaço para a cadeia de filtros. Contudo, além de mostrar ser um processo complicado devido à sincronização do relógio do sistema e dados, a transmissão de sinais em alta frequência (na ordem dos MHz) no cabo entre placas, provoca certas instabilidades no sistema. Por vezes um simples toque no cabo provoca a dessincronização da imagem. O terceiro sistema funciona com um projecto criado para a Virtex-II Pro de forma a possibilitar a reconfiguração dinâmica. Este projecto funciona, mas também mostrou ser muito instável porque uma queda de tensão provocada por uma protecção de pinos que ligam a FPGA aos periféricos, especialmente à câmara de vídeo, provoca por vezes uma paragem no funcionamento. A mudança de carga mínima, provocada pela ponta de prova de um osciloscópio no pino que leva o sinal de relógio à câmara, provoca uma paragem na geração de dados da câmara para a FPGA. Deste modo, para o bom funcionamento deste projecto na Virtex-II Pro é necessário uma mudança de periféricos para outros mais apropriados à placa.

No que diz respeito aos filtros, a qualidade dos resultados foram correspondidos às expectativas, pois e efeito produzido nas imagens e vídeos foi o esperado, conseguindo assim igualar a nível de funcionamento os sistemas baseados em CPU, mas com as vantagens de uma implementação em hardware.

Trabalho Futuro

Na sequência do trabalho desenvolvido, muitas soluções podem ser ainda melhoradas ou acrescentadas. Como por exemplo, no caso da utilização do projecto na Virtex-II pro, implementar uma

cadeia de áreas dinâmicas, permitindo assim a configuração de uma cadeia de filtros reconfigurável. Outra solução interessante, seria o estudo de uma implementação cuja cadeia de filtros é implementada como sendo estática, mas com o conhecimento exacto da localização dos registos que guardam o valor das configurações dos mesmos. Nestas condições, é criada a possibilidade de reconfigurar apenas estes registos e assim as configurações dos filtros. Com este processo é criada de certa forma, uma independência em relação ao computador à qual a FPGA está ligada via porta série, para configurar os seus filtros. Um projecto que também seria interessante desenvolver a partir deste, seria o desenvolvimento baseado num processador, aproveitando um dos CPUs integrados na Virtex-II pro, que controlaria a reconfiguração dinâmica dos filtros. A biblioteca de filtros, neste caso já em *bit streams* seria guardada numa memória DDR, cujo controlo é feito também pelo processador. Esta solução seria completamente autónoma na media que não necessitaria de um computador para realizar a configuração dos filtros ou a reconfiguração das áreas dinâmicas.

Anexo A

Tabela de tradução de pinos

A.1

Esta tabela indica como os pinos estão ligados na placa que adapta as fichas FX2-100p para IDC-40p

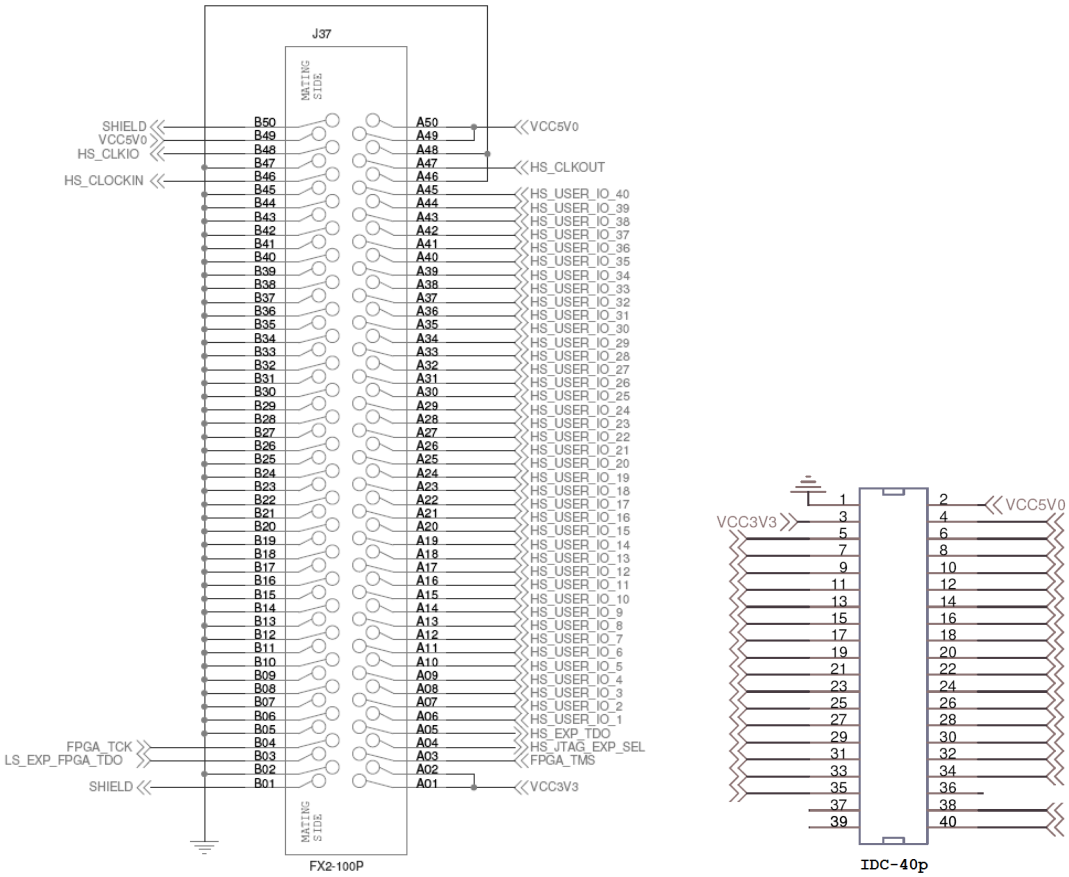

Figura A.1: Esquema das fichas FX2-100p e IDC-40p

Tabela A.1: Ligação de pinos

FPGA	FX2-100p	IDC-40p
GND	A46 e A48	1
VCC5V0	A49 e A40	2
VCC3V3	A01 e A02	3
HS_USER_IO_1	A06	4
HS_USER_IO_2	A07	5
HS_USER_IO_3	A08	6
HS_USER_IO_4	A09	7
HS_USER_IO_5	A10	8
HS_USER_IO_6	A11	9
HS_USER_IO_7	A12	10
HS_USER_IO_8	A13	11
HS_USER_IO_9	A14	12
HS_USER_IO_10	A15	13
HS_USER_IO_11	A16	14
HS_USER_IO_12	A17	15
HS_USER_IO_13	A18	16
HS_USER_IO_14	A19	17
HS_USER_IO_15	A20	18
HS_USER_IO_16	A21	19
HS_USER_IO_17	A22	20
HS_USER_IO_18	A23	21
HS_USER_IO_19	A24	22
HS_USER_IO_20	A25	23
HS_USER_IO_21	A26	24
HS_USER_IO_22	A27	25
HS_USER_IO_23	A28	26
HS_USER_IO_24	A29	27
HS_USER_IO_25	A30	28
HS_USER_IO_26	A31	29
HS_USER_IO_27	A32	30
HS_USER_IO_28	A33	31
HS_USER_IO_29	A34	32
HS_USER_IO_30	A35	33
HS_USER_IO_31	A36	34
HS_USER_IO_32	A37	35
HS_USER_IO_33	A38	38
HS_USER_IO_34	A39	40

Anexo B

Ficheiros dos projectos

Esta secção resume os projectos que foram criados ao longo desta dissertação e indica o link que disponibiliza os ficheiros que foram usados e criados. Descreve ainda, quais os ficheiros que foram apenas utilizados do projecto existente, modificados ou criados.

B.1 Projecto 1

O primeiro projecto é feito para a implementação no dispositivo Xilinx Spartan-3 FPGA. Ele permite a aquisição de vídeo através de uma câmara digital ligada ao dispositivo e a sua reprodução num monitor atreves de um módulo que possui um VDAC, deixando condições para a criação de uma cadeia de processamento das imagens/vídeo, nomeadamente filtros.

Neste projecto estão incluídos os filtros que fazem parte da biblioteca de módulos reconfiguráveis proposta, um módulo que permite guardar imagens numa memória SDRAM integrada na board e que permite usa-las como entrada para os filtros ou guardar como resultados destes.

Este projecto está disponível em `www.fe.up.pt/~ee99079/files/spartan.rar`.

A tabela B.1 mostra a lista de módulos que fazem parte deste projecto, indicando quais os originais, os alterados e os criados.

B.2 Projecto 2

Uma vez que a Spartan-3 não possuí área suficiente para a implementação de uma combinação de filtros numa cadeia. Para fazer o teste de uma cadeia de filtros, um segundo projecto foi desenvolvido para implementar em duas placas Spartan-3 ligadas entre si com um barramento de dados. Neste projecto, uma placa identificada como sendo a A, faz a captura de vídeo e parte da cadeia de filtros. A outra identificada como sendo B, faz o resto da cadeia e a reprodução de vídeo no monitor.

A tabela B.2 mostra que módulos do projecto 1 ficaram em cada placa, à excepção de todos os módulos que pertencem aos filtros, porque estes podem ficar em qualquer placa. As propriedades são em relação ao projecto 1.

Tabela B.1: Lista de módulos para projecto na Spartan-3

Nome	Descrição	Propriedade
binario.v	Filtro de threshold	Criado
bordas.v	Filtro detecção de bordas	Criado
c_bordas.v	Cálculo do filtro detecção de bordas	Criado
c_conv.v	Cálculo do filtro de convolução	Criado
c_morf.v	Cálculo do filtro morfológico	Criado
cam_resyncv2.v	Gera sinais de sincronismo e posição	Original
conv.v	Filtro de convolução	Criado
dcam_toplevel.v	módulo *top level*	Modificado
dcam_toplevel.ucf	Definição dos pinos físicos	Original
dcmdiv2_v2.v	DCM	Original
hrefvsync.v	Sincronismo de sinais para alguns filtros	Criado
mem_2x2.v	Memória para acesso a matriz 2x2 de píxeis	Criado
mem_3x3.v	Memória para acesso a matriz 3x3 de píxeis	Criado
morf.v	Filtro de morfologia matemática	Criado
startup_i2c	Comunicação I2C	Original
suart.v	Controlador da porta série	Original
unisepav4.v	interprete de comandos e acesso à memória	Modificado
usemem.v	guardar e reproduzir imagens em memória	Criado
VGAcore.v	Controlador de saída de vídeo para monitor	Original

Este projecto está disponível em `www.fe.up.pt/~ee99079/files/2spartan.rar`.

B.3 Projecto 3

Porque a Spartan-3 não suporta a reconfiguração dinâmica, surgiu a necessidade de migração de projecto para uma que servisse para o efeito. O dispositivo disponível para o efeito é a Virtex-II Pro. Foi então criado um novo projecto para este novo dispositivo. Excluindo os filtros que não foram alterados, a tabela B.3 mostra os ficheiros que fazem parte deste projecto com a sua propriedade face aos projectos anteriores.

Este projecto está disponível em `www.fe.up.pt/~ee99079/files/virtex.rar`.

B.4 Projecto 4

O último projecto disponível em `www.fe.up.pt/~ee99079/files/virtex_pr.rar` para o dispositivo Virtex-II Pro, possui uma implementação com suporte para a reconfiguração dinâmica. Estão incluídos os ficheiros para implementação em área estática nomeados na tabela B.4, os ficheiros dos projectos individuais para a área dinâmica, tabela B.5 e o projecto do programa PlanAhead onde as áreas estão definidas assim como as *bus macros*. Mais uma vez as propriedades são em relação aos projectos anteriores.

Tabela B.2: Lista de módulos para projecto em duas Spartan-3

Nome	Placa	Propriedade
cam_resyncv2.v	Placa A	Original
dcam_toplevel.v (projectoA)	Placa A	Modificado
dcam_toplevel.v (projectoB)	Placa B	Modificado
DCM_V2p	Placa A	Criado
DCM_V2p	Placa B	Criado
img_pos.v	Placa A	Criado
sinc.v	Placa B	Criado
startup_i2c	Nas duas	Original
suart.v	Nas duas	Original
unisepav4.v	Nas duas	Original
usemem.v	Placa A	Original
usemem.v (projectoB)	Placa B	Modificado
VGAcore.v	Placa B	Original

B.5 Interface com os módulos

Trata-se de um programa desenvolvido para computadores com o Windows como sistema operativo e cuja função é comunicar através da porta serie com a FPGA usando para tal um protocolo pré estabelecido na FPGA assim como no programa.

Este programas está disponível em `www.fe.up.pt/~ee99079/files/hsender.rar`, juntamente com o seu código fonte.

Tabela B.3: Lista de módulos para projecto na Virtex-II Pro

Nome	Descrição	Propriedade
cam_resyncv2.v	Gera sinais de sincronismo e posição	Original
dcam_toplevel.v	módulo *top level*	Modificado
dcam_toplevel.ucf	Definição dos pinos físicos	criado
dcm_v5.v	DCM	Criado
startup_i2c	Comunicação I2C	Original
suart.v	Controlador da porta série	Original
unisepav4.v	interprete de comandos	Original
VGAcore.v	Controlador de saída de vídeo para monitor	Original

Tabela B.4: Lista de módulos para projecto reconfigurável na Virtex-II Pro

Nome	Descrição	Propriedade
dcam_toplevel.ucf	Definição dos pinos físicos	Modificado
dcam_toplevel.v	módulo para área estática	Modificado
cam_resyncv2.v	Gera sinais de sincronismo e posição	Original
dcm_v5.v	DCM	Original
startup_i2c	Comunicação I2C	Original
suart.v	Controlador da porta série	Original
top.vhd	módulo *top level*	Criado
unisepav4.v	interprete de comandos	Original
VGAcore.v	Controlador de saída de vídeo para monitor	Original

Tabela B.5: Lista de módulos para projecto para as áreas dinâmicas na Virtex-II Pro

Nome	Descrição	Propriedade
binario.v	Filtro de threshold	Modificado
bordas.v	Filtro detecção de bordas	Modificado
c_bordas.v	Cálculo do filtro detecção de bordas	Original
c_conv.v	Cálculo do filtro de convolução	Original
c_morf.v	Cálculo do filtro morfológico	Original
conv.v	Filtro de convolução	Modificado
mem_2x2.v	Memória para acesso a matriz 2x2 de píxeis	Original
mem_3x3.v	Memória para acesso a matriz 3x3 de píxeis	Original
morf.v	Filtro de morfologia matemática	Modificado

Anexo C

Referências das imagens

A lista de figuras seguinte, trata-se de figuras que não constam dos documentos referenciados neste documento mas que também não pertencem ao autor. Deste modo, a lista apresenta os endereços de Internet da qual as figuras foram retiradas.

Figura 2.1 http://paginas.ucpel.tche.br/~vbastos/pi.htm

Figura 2.3 http://commons.wikimedia.org/wiki/File:Conversion_AD_DA.gif

Figura 2.4 http://commons.wikimedia.org/wiki/File:Fir_filter_df1.png

Figura 2.5 http://commons.wikimedia.org/wiki/File:IIR-filter.png

Figura 2.7 e 2.8 http://www.dpi.inpe.br/spring/portugues/tutorial/introducao_pro.html

Figura 2.10, 2.16 e 2.17 http://www.inf.ufsc.br/~visao/1999/textura/c_filtro.htm

Figura 2.12 http://www.dpi.inpe.br/~carlos/Academicos/Cursos/Pdi/pdi_filtros.htm

Figura 2.19 http://users.erols.com/aaps/x84lab/FPGA.html

Figura 2.20 http://www.et.byu.edu/groups/ece224web/lectures/19%20FPGA.pdf

Referências

[1] Rafael C. Gonzales and Paul Wintz. *Digital image processing (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.

[2] Virtex-ii pro and virtex-ii pro x platform fpgas complete data sheet. Technical report, Junho 2004.

[3] Resumo do projecto: Reconfiguração dinâmica de protocolos de comunicação, 2006.

[4] Yu-Jin Zhang. An overview of image and video segmentation in the last 40 years. 2006.

[5] Jaime Cardoso. *Metadata Assisted Image Segmentation*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2006.

[6] Luís Corte-Real Luís F. Teixeira, Jaime S. Cardoso. Object segmentation using background modelling and cascaded change detection. 2007.

[7] J.B. Evans. An efficient fir filter architecture. *Circuits and Systems, 1993., ISCAS '93, 1993 IEEE International Symposium on*, pages 627–630, May 1993.

[8] Zhongnong Jiang and Jr. Willson, A.N. A pipelined/interleaved iir digital filter architecture. *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, 3:2217–2220 vol.3, Apr 1997.

[9] Kenneth R. Castleman. *Digital image processing*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1996.

[10] A. Wrangsjö and H. Knutsson. Histogram filters for noise reduction. March 2003.

[11] Mark Grundland and Neil A. Dodgson. Interactive contrast enhancement by histogram warping.

[12] S. Srinivasan and N. Balram. Adaptive contrast enhancement using local region stretching. *Proc.of ASID 06, 8-12 Oct, New Delhi*.

[13] Bernd Jähne. *Digital Image Processing: Concepts, Algorithms, and Scientific Applications*. 2005.

[14] Josh Wills. Proof of the convolution theorem. 2002.

[15] Anders Hast. Bump mapping.

[16] Richard G. Shoup. Parameterized convolution filtering in an fpga. pages 274–280, 1994.

[17] Y J Ren, J G Zhu, X Y Yang, and S H Ye. The application of virtex-ii pro fpga in high-speed image processing technology of robot vision sensor. *Journal of Physics: Conference Series*, 48:373–378, 2006.

[18] Robert Turney. Two-dimensional linear filtering. Technical report, Xilinx, 2007.

[19] J. R. Parker. *Algorithms for Image Processing and Computer Vision*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[20] J. Velten and A. Kummert. Fpga-based implementation of variable sized structuring elements for 2d binary morphological operations. *Electronic Design, Test and Applications, 2002. Proceedings. The First IEEE International Workshop on*, pages 309–312, 2002.

[21] Damien Baumann e Jacques Tinembart. *Designing Mathematical Morphology Algorithms on FPGAs: An Application to Image Processing*. Springer Berlin / Heidelberg, 2005.

[22] Kazimierz Wiatr. Median and morphological specialized processors for a real-time image data processing. *EURASIP J. Appl. Signal Process.*, 2002(1):115–121, 2002.

[23] Manglem Singh and Prabin K. Bora. Two-dimensional linear prediction based median filtering.

[24] J S Smith A J Tickle and Q H Wu. Development of morphological operators for field programmable gate arrays. *Journal of Physics: Conference Series 76 (2007) 012028*, page 17, 2007.

[25] Stephen Brown and Jonathan Rose. Architecture of fpgas and cplds: A tutorial. *IEEE Design and Test of Computers*, 13:42–57, 1996.

[26] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli. Architecture of field-programmable gate arrays. *Proceedings of the IEEE*, 81(7):1013–1029, Jul 1993.

[27] Michael Schulte Emily Blem Philip Garcia, Katherine Compton and Wenyin Fu. An overview of reconfigurable hardware in embedded systems. *EURASIP Journal on Embedded Systems*, vol. 2006:19 paginas, 2006.

[28] Katherine Compton. Programming architectures for run-time reconfigurable systems. Master's thesis, Northwestern University, 1999.

[29] Miguel M. Silva e João Canas Ferreira. Development of applications with run-time support for dynamic reconfiguration. *Jornadas sobre Sistemas Reconfiguráveis*, 2005.

[30] Miguel M. Silva and João Canas Ferreira. Generation of hardware modules for run-time reconfigurable hybrid cpu/fpga systems. 2005.

[31] Miguel Lino Magalhães da Silva. Ferramentas de apoio ao teste concorrente para fpgas com reconfiguração parcial dinâmica. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2003.

[32] Supplement to ieee std 1149.1-1990, ieee standard test access port and boundary-scan architecture, Mar 1995.

[33] *Xilinx ISE Design Suite 10.1*. http://www.xilinx.com/ise/logic_design_prod/webpack.htm.

[34] *ModelSim XE III 6.1e*. http://www.xilinx.com/ise/mxe3/download.htm.

[35] *Spartan-3 Starter Kit Board User Guide*, 2005.

[36] electronics123. *C3188A 1/3" Color Camera Module With Digital Output*.

[37] Jingke Li. Sorting algorithms.

[38] Xilinx. Using look-up tables as shift registers (srl16) in spartan-3 generation fpgas. Technical report, 2005. Documento: XAPP465.

[39] Integrated Silicon Solution, Inc. *IS61LV25616L*, 2002.

[40] *HDD Free Serial Port Monitor*. http://www.serial-port-monitor.com/.

[41] *Xilinx University Program Virtex-II Pro Development System Hardware Reference Manual*, 2005.

[42] Brian Jackson. *Partial Reconfiguration Design with PlanAhead 9.2*. Xilinx, 2007.

[43] Xilinx. *Early Access Partial Reconfiguration User Guide For ISE 8.1.01i*, 2006.

[44] Xilinx. *Partial Reconfiguration Lab Implementing a Simply Color Bar design on a ML505 Demo Board*.