FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Demonstrador Remoto de Reconfiguração Dinâmica de FPGAs

Hugo Daniel Ferreira Marques

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Orientador: João Canas Ferreira (Prof. Doutor)

20 de Julho de 2012

Resumo

A reconfiguração dinâmica consiste na adaptação de *hardware* genérico, com a intenção de acelerar algoritmos ou secções de algoritmos. A adaptação é realizada através de um ficheiro de configuração e são precisos apenas alguns décimos de segundo para realizar a reconfiguração. A reconfiguração dinâmica é realizada com a ajuda de um processador genérico que recebe ordens e atua,em concordância, enviando novas configurações para a memória da Field-programmable gate array (FPGA).

O "Demonstrador Remoto de Reconfiguração Dinâmica de FPGAs" é um projeto que visa implementar um sistema que transpareça para o utilizador o conceito de reconfiguração parcial dinâmica e simultaneamente demonstre as suas vantagens e possíveis usos.

Para este efeito foi construída uma cadeia de processamento de imagem com quatro partições reconfiguráveis. Estas partições podem ser reconfiguradas, a pedido do utilizador, para efetuar as operações pretendidas numa imagem submetida pelo próprio utilizador.

O utilizador pode submeter a imagem, escolher os filtros pretendidos e observar os resultados, através de uma interface gráfica desenvolvida para este efeito, que comunica remotamente com o sistema reconfigurável.

Abstract

Dynamic reconfiguration consists in adapting hardware in order to accelerate algorithms or portions of algorithms. The reconfiguration itself is done through a reconfiguration file called bitstream. Controlling the whole operation, a generic processor receives orders and acts according to those orders sending new configurations to the FPGA memory.

The "FPGA Dynamic Reconfiguration Remote Demonstrator" is a project that intends to implement a dynamic reconfigurable system that allows the user to understand the concepts and the advantages of it's use.

For this to be accomplished, it was built an image processing chain which has four reconfigurable slots that are used as image filters. The configuration of these slots is changed according to user requests.

The user can submit the image for processing, pick up which filters he wants and observe the results, through graphical interface that communicates with the reconfigurable system remotely.

Agradecimentos

Pela sua orientação e conhecimento, envio um agradecimento ao Prof. João Canas Ferreira.

Pelo companheirismo, ajuda, boa disposição e amizade, agradeço aos colegas e amigos que durante o semestre partilharam o local de trabalho comigo.

Por me formarem e educarem, por me darem força, carinho e amor, agradeço aos meus pais: Raul Marques e Maria Emília Marques.

Hugo Daniel Ferreira Marques

"Any sufficiently advanced technology is indistinguishable from magic."

Arthur C. Clarke

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ASIC Application-Specific Integrated Circuit

CLB Configurable Logic Block

DMA Direct Memory Access

DRE Data Realignment Module

DVI Digital Visual Interface

FIFO First In First Out

FPGA Field-programmable gate array

GPP General-Purpose Processor

HDL Hardware Description Language

ICAP Internal Configuration Access Port

IOB Input Output Block

LUT Look-up Tables

NCD Native Circuit Description

NGD Native Generic Database

PLB Processor Local Bus

SoC System on Chip

SRAM Static random-access memory

ROM Read Only Memory

RTL Register Tranfer Level

TCL Tool Command Language

XPS Xilinx Platform Studio

XST Xilinx Synthesis Technology

Capítulo 1

Introdução

O presente documento relata e descreve todo o trabalho desenvolvido, no período designado para a dissertação. Neste capítulo encontra-se contextualizado o propósito do projeto, bem como os seus objetivos e, ainda, uma breve descrição da estrutura deste relatório.

A computação reconfigurável adquiriu maior ênfase após o aparecimento do circuito tipo FPGA *Field-programmable gate array*, nos anos oitenta, mais concretamente em 1985. Atualmente, devido ao aumento da complexidade e das funcionalidades disponíveis em um só dispositivo, existe um esforço, cada vez maior, na procura de uma solução capaz englobar todas as necessidades e evitar a utilização de várias unidades *System on Chip (SoC)*. Tendo em conta esta premissa, a reconfiguração computacional é encarada, cada vez mais, como uma possibilidade viável.

A reconfiguração computacional, que tem como principal finalidade acelerar algoritmos, consiste na adaptação de hardware genérico, a pedido ou conforme as necessidades de processamento [1]. É suportado por um *General-Purpose Processor (GPP)* e lógica reconfigurável em hardware. O GPP tem como funcionalidade o controlo comportamental das tarefas que correm no hardware reconfigurável e pode, também, executar comunicações externas.

Do ponto de vista tecnológico, poderá dizer-se que existem dois extremos. Num deles, encontra-se a computação executada mediante um *Application-Specific Integrated Circuit (ASIC)*, um circuito integrado, que proporciona uma função única. No outro, está a computação executada por um GPP, igualmente conhecido como microprocessador, que é programado com o software necessário para resolver um dado problema. A computação reconfigurável está situada entre os dois, consegue um desempenho muito mais elevado que um GPP e, simultaneamente, mantém um nível elevado de flexibilidade, em comparação com um ASIC, que não é flexível.

Sendo uma tecnologia que se prevê que, com o seu desenvolvimento, estará cada vez mais presente em dispositivos do quotidiano, torna-se relevante que o conceito de reconfiguração dinâmica seja introduzido a estudantes na área de engenharia eletrónica. É neste contexto que se insere o meu projeto: construir uma ferramenta que funcione, pedagogicamente, como um ponto de partida na aprendizagem daquilo que é a reconfiguração dinâmica.

1.1 Objetivos

Em concordância com o título, "Demonstrador remoto de reconfiguração dinâmica de FPGAs", o objetivo primordial deste projeto é desenvolver e implementar uma aplicação, baseada em reconfiguração dinâmica, que demonstre as vantagens da utilização desta tecnologia [2].

Para realçar as operações efetuadas em FPGA, decidiu-se desenvolver um sistema aplicado ao processamento de imagem, desta forma é fácil para o utilizador observar o impacto da operação que acabou de realizar. Assim, consegue-se uma maior eficiência, do ponto de vista pedagógico.

Em hardware, foi implementada uma cadeia de processamento em série, onde se encontram filtros que funcionam como áreas que são reconfiguráveis a pedido do utilizador.

Para a seleção das operações a efetuar na FPGA, bem como para a visualização dos resultados, o utilizador deve possuir uma interface remota que comunica com a placa de desenvolvimento, transferindo a imagem a ser processada e as instruções para efetuar a reconfiguração.

Como resultado final, o utilizador deve visualizar a sua imagem produto e, ainda, os indicadores de desempenho. Esta interface deve ter um cariz pedagógico, ou seja, deve transparecer para o utilizador o mecanismo da reconfiguração dinâmica.

Escolheu-se, para este efeito, utilizar a placa *Virtex 5 ML505*, que possui um *soft-core* denominado *MicroBlaze*. Neste processador deve ser implementada a mesma biblioteca de filtros que será implementada em hardware. Desta forma, será possível efetuar comparações de desempenho entre a implementação em *FPGA* e em *software*. Como resultado último, espera-se que o desempenho da implementação em hardware seja substancialmente melhor do que em software.

Embora seja um sistema com interface remota, no desenvolvimento deste trabalho, os esforços foram concentrados em duas vertentes: desenvolvimento da arquitetura em hardware, que permita fazer o varrimento da imagem e encadear vários módulos para a filtrar, e uma segunda vertente, que é a reconfiguração a pedido do utilizador.

O meu trabalho pode ser utilizado segundo dois modos: utilizador principiante e utilizador avançado. O utilizador principiante utilizará apenas a interface remota, para efetuar operações de reconfiguração e observar os resultados. Assim, esta aplicação deverá ajudar o utilizador a adquirir os conceitos básicos de reconfiguração dinâmica. Já o utilizador avançado deverá utilizar o meu sistema base para expandir o seu, ou seja, seguindo um tutorial fornecido, este deverá ser capaz de desenvolver os seus próprios módulos, adicioná-los às partições reconfiguráveis, executá-los e observar os seus resultados, através da interface remota.

Com o meu projeto, espero contribuir para motivar novos aprendizes, na área de sistemas digitais, introduzindo o tema de reconfiguração dinâmica e mostrando as vantagens da sua utilização, esperando que os utilizadores do meu sistema fiquem elucidados e motivados o suficiente para eles próprios contribuírem para o desenvolvimento de novas aplicações utilizando reconfiguração dinâmica.

1.2 Estrutura do Documento

A estrutura deste documento contempla oito capítulos mais a secção de anexos. No presente capítulo 1 foram descritos os objetivos e o contexto do trabalho. No próximo, capítulo 2, chamado de "Estado da Arte", encontra-se uma descrição sobre a arquitetura de uma FPGA, o fluxo de projeto de um sistema com reconfiguração dinâmica e são também expostas outras aplicações com reconfiguração dinâmica.

O capítulo 3, chamado de "Arquitetura do Sistema Base", serve para evidenciar a abordagem geral do sistema, explicitando sob a forma de blocos ou sub-sistemas o fluxo de informação que o percorre.

Os capítulos 4 e 5 englobam a implementação em *hardware*. O capítulo quatro descreve, essencialmente, o módulo de *hardware* que foi desenvolvido por mim. Além disso, é também mencionado o funcionamento de outros dois módulos que desempenham funções relevantes no sistema. O capítulo cinco, chamado de "Implementação e Desenvolvimento de Filtros", menciona e expõe a abordagem utilizada para implementação dos vários filtros de imagem.

O capítulo 6 descreve o *software* implementado tanto no processador *MicroBlaze* como na interface remota desenvolvida usando JAVA.

O capítulo 7, chamado de "Validação e Verificação", mostra os resultados obtidos no funcionamento do sistema.

O capítulo 8 contempla as conclusões de todo o trabalho desenvolvido ao londo do período de dissertação.

Capítulo 2

Estado da Arte

No corpo deste capítulo expõe-se o grau de desenvolvimento da tecnologia usada na minha dissertação. Na primeira secção é realizada uma descrição da arquitetura de uma FPGA genérica e explicitadas as características da *Virtex ML505*. Na segunda secção é detalhado o fluxo de projeto de um sistema com reconfiguração dinâmica, bem como mencionadas as ferramentas de suporte que podem ser usadas na construção deste sistema. Na terceira e ultima secção, são expostos trabalhos que utilizam reconfiguração dinâmica e estão relacionados com a minha dissertação.

2.1 Arquitetura da FPGA

A FPGA usada no desenvolvimento da minha dissertação é fabricada pela *Xilinx*. Por essa razão, a arquitetura descrita está relacionada com os produtos fabricados por esta companhia. Contudo, os princípios gerais são os mesmos usados no fabrico de FPGAs por outras marcas.

2.1.1 Arquitetura Genérica

Uma FPGA é essencialmente formada por conjuntos de três tipos de blocos: *Configurable Logic Block (CLB)*, *Input Output Block (IOB)* e matrizes de interconexão.

Os CLBs são os componentes principais de uma FPGA, não só porque existem em maior quantidade, mas também porque são responsáveis pela implementação de funções lógicas. As funções são realizadas com o auxílio de Look-up Tables (LUT)s que armazenam, sob a forma de tabela, todos os possíveis resultados para uma dada função.

Numa FPGA, os CLBs possuem: uma ou mais LUTs de 4 bits de entrada, *flip-flops* do tipo D e lógica de conexão à sua volta [3]. Cada LUT de 4 bits contém 16 células de memória, facto que é independente da complexidade . A lógica de conexão, é formada por portas lógicas e é responsável pelo encaminhamento dos sinais de entrada e de saída da LUTs, *multiplexers* e trincos [1].

Durante o processo de configuração de uma FPGA, as LUTs são preenchidas com as funções necessárias e o hardware em seu redor é configurado de forma a construir o sistema complexo requerido.

A figura 2.1, mostra a estrutura da uma FPGA com interconexão segmentada. Este conceito será abordado de seguida.

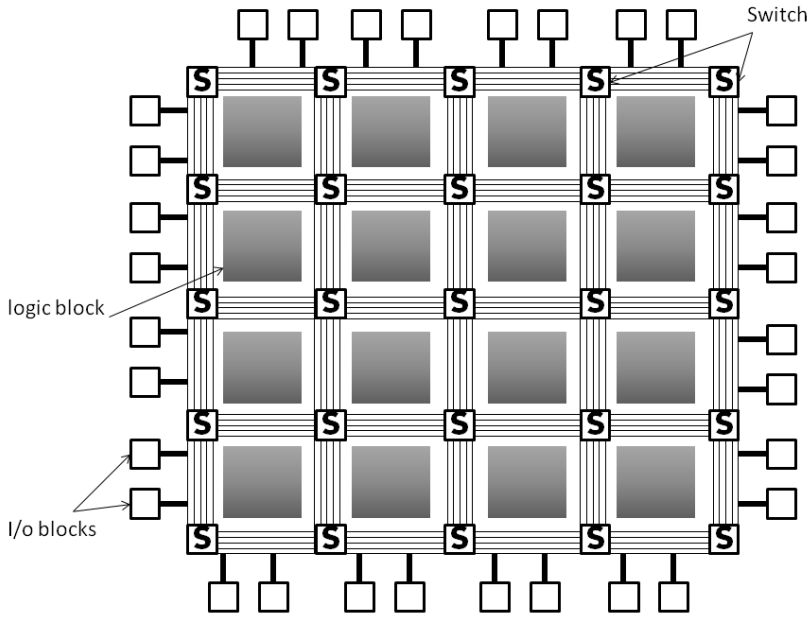

Figura 2.1: Arquitetura da FPGA com interconexão segmentada.

Os IOBs têm a função de conectar os sinais lógicos internos a um pino de saída da FPGA. A cada pino de saída está associado apenas um IOB. Estes possuem, armazenado na sua memória interna, os valores de tensão padrão a que cada pino deve obedecer e a direção de comunicação. Não só é possível estabelecer comunicações unidirecionais, em qualquer direção, mas também, é possível estabelecer comunicações bidirecionais [1].

As matrizes de interconexão permitem a ligação arbitrária de CLBs e IOBs. Fundamentalmente, existem dois tipos de interconexão: direta e segmentada [3].

A interconexão direta é feita de conjuntos de ligações, dispostas em todas as direções, ao longo do dispositivo. Os CLBs colocam a informação no canal mais adequado de acordo com o destino pretendido.

A interconexão segmentada é baseada em linhas que podem ser interconectadas usando matrizes *switch* programáveis. As matrizes estão todas ligadas por linhas que atravessam todo o dispositivo. Desta forma aumenta a velocidade de comunicação e diminui os desfasamentos na propagação dos sinais.

A vantagem da interconexão direta é que as resistências e capacidades parasitas são praticamente constantes, assim a propagação dos sinais é mais previsível. Por outro lado, a interconexão segmentada apresenta menor dissipação de energia [1].

Nas FPGAs da *Xilinx*, toda a memória de configuração é feita de *Static random-access memory (SRAM)*, ou seja, memória volátil: quando o dispositivo é desligado, toda a sua configuração é perdida. É possível que em algumas placas exista memória *Read Only Memory (ROM)* ou, por

exemplo, um módulo leitor de cartões de memória flash, onde se encontra armazenado o ficheiro de configuração da FPGA que é carregado quando esta é iniciada.

O ficheiro de configuração da memória SRAM é chamado de *bitstream* e pode ser completo ou parcial de acordo com a extensão de memória a ser configurada.

2.1.2 Arquitetura da Virtex 5 ML 505

De forma semelhante à arquitetura genérica descrita anteriormente, a FPGA da placa *Xilinx Virtex 5* é também constituída por três blocos principais: CLBs, IOBs e lógica de interconexão.

Cada unidade de CLB é constituída por dois *slices* equivalentes. Cada *slice* contém quatro LUTs, três multiplexadores dedicados, lógica aritmética dedicada e registos adicionais.

Nesta FPGA as LUTs podem ser configuráveis como tabelas de 6 bits de entrada ou 5 bits de entrada com dupla saída. Conetados às LUTs encontram-se três tipos de multiplexadores: F7AMUX, F7BMUX e F8MUX. O F7MUX e F7BMUX combinam as saídas das LUTs para construir circuitos combinacionais com 7 bits de entrada. Já o F8MUX é usado para combinar as saídas dos outros dois multiplexadores (F7AMUX e F7BMUX).

A lógica aritmética dedicada possibilita a implementação de funções aritméticas com velocidades de execução elevadas. Cada CLB suporta duas cadeias de transporte, uma por cada *slice*, que se estendem ao longo do *slice* chegando a várias LUTs. Assim torna-se possível implementar funções aritméticas e lógicas de grande complexidade.

Para além do que já foi mencionado, existem quatro registos de 1 bit que podem ser configurados como *flip-flops* do tipo D ou trincos. A entrada destes registos é selecionada por multiplexadores AMUX-DMUX. Estes multiplexadores são programados durante a configuração da FPGA e, por isso, não acessíveis pelo utilizador [4] [5].

Cada CLB utiliza a interconexão segmentada para fazer o encaminhamento dos sinais.

A *Virtex 5* possui blocos de memória RAM, 36Kbits *dual-port*, que podem ser programados com várias larguras e profundidades. Cada bloco de 36Kbits, pode ainda ser configurado para ser utlizado como dois blocos de 18Kbits independentes. Um bloco de memória tem dois modos de operação: leitura e escrita. Existe ainda a possibilidade de encadear os vários blocos para formar um bloco de memória maior [5].

Cada IOB subdivide-se em elementos de memória, multiplexadores e *buffers*. Em memória são armazenados os valores de tensão padrão a que o pino deve obedecer. Os multiplexadores auxiliam na configuração do sinal de saída.

A tabela 2.1 mostra as caraterísticas da placa *Virtex 5*:

.

XC5VLX50T	CLBs	*Array*	120 x 30
		Slices	7200
		Max Distributed RAM (Kbit)	480
	DSP48E Slices[1]		32
	Block RAM Blocks	18Kbit	120
		36Kbit	60
		Max	2160
	CMTs [2]		6
	Endpoint Blocks for PCI Express		1
	Max RocketIO Transceivers	GTP	12
	Total I/O Banks		15
	Max User I/O		450

Tabela 2.1: Caraterísticas da placa *Virtex 5 ML505*

2.2 Fluxo de Projeto

Num sistema com reconfiguração parcial de FPGAs, o fluxo de projeto pode ter várias variantes e existem enumeras ferramentas e métodos aplicados às etapas de desenvolvimento. A *Xilinx* sugere dois modos básicos de reconfiguração dinâmica: modo diferencial (*Difference-based*) e o modo baseado em módulos (*Module-based*) [6].

O modo diferencial de reconfiguração dinâmica deve ser usado quando apenas se pretende efetuar pequenas alterações no *design*. É especialmente útil para se realizar alterações nas funções armazenadas nas LUTs ou alterar os conteúdo de blocos de memória dedicada. Neste caso, o *bitstream* parcial contém apenas informação sobre as diferenças relativas ao *design* que se encontra, naquele presente momento, implementado na FPGA. Este método tem vantagem de ser muito rápido, uma vez que as diferenças entre os dois *designs* podem ser muito pequenas.

O modelo de reconfiguração baseado em módulos deve ser utilizado no caso de se pretender efetuar alterações de dimensão significativa nas secções de memória reconfiguráveis. Este modelo está, geralmente, associado à troca de módulos reconfiguráveis, com determinada função, por outros módulos com uma funcionalidade diferente.

No âmbito deste projeto, é mais adequada a utilização de um modelo de reconfiguração modular. Por essa razão, na presente secção, o fluxo de projeto e as ferramentas usadas que serão descritas, encontram-se direcionadas para o modelo escolhido.

A reconfiguração da FPGA é realizada descarregando um ficheiro de configuração para a sua memória. No caso da reconfiguração parcial, este ficheiro, chamado de *bitstream* parcial, apenas reconfigura zonas específicas de memória, deixando intactas as zonas estáticas do sistema.

[1] Cada *slice* DSP48E contém um multiplicador 25 x 18, um somador e um acumulador.

[2] Cada *Clock Management Tile* (CMT) contém dois *Digital Clock Manager* (DCMs) e uma *Phased Locked Loop* (PLL)

O processo de gerar as *bitstreams* parciais é apenas a etapa final de toda a sequência que culmina com os vários ficheiros .bit que serão utilizados.

Este processo começa na modelação em *Hardware Description Language (HDL)* de um circuito *top-level*. Dentro deste circuito está toda a lógica estática e, encontram-se instanciados, como caixa negra, os módulos reconfiguráveis. Uma caixa negra é um módulo para qual estão instanciadas as entradas e saídas, mas não possui lógica interna e, por essa razão, não tem uma *netlist* definida. [7]

Após a validação da funcionalidade do módulo de topo, este deve ser traduzido numa *netlist*. Para este processo, chamado de síntese *Register Tranfer Level (RTL)*, é utilizada a ferramenta *Xilinx Synthesis Technology (XST)* [8]. Esta ferramenta sintetiza ficheiros em HDL e produz, como resultado, um ficheiro que descreve a interligação de blocos primitivos e contém informação sobre as restrições do *design*. Estes blocos podem ser portas lógicas, *flip-flops*, *buffers*, RAMs, entre outros.

O ambiente gráfico que a *Xilinx* disponibiliza, onde podem ser desenvolvidos e sintetizados os módulos RTL chama-se *Projnav*. Contudo, para realizar a síntese, também se podem usar *scripts Tool Command Language (TCL)*.

As *netlists* dos módulos reconfiguráveis devem ser geradas de forma independente da lógica estática. Para este efeito, os módulos devem ser sintetizados como se tratassem de um módulo de topo. Desta forma gera-se a *netlist* que será posteriormente encaixada nas partições instanciadas como caixa negra.

A próxima fase é chamada de *floorplanning* [9] [10]. Nesta fase são definidas as restrições de área onde se colocará a lógica reconfigurável. Estas restrições delimitam a área atribuída a cada partição. Assim é também possível estimar os recursos disponíveis dentro da área delimitada. A *Xilinx* disponibiliza um ambiente gráfico, chamado *Planahead* [9], onde se realiza o *floorplanning* e a implementação.

No *Planahead* é adicionada a *netlists* do *top-level*, onde se encontram instanciadas as *black-box*. Estas, serão definidas como partições reconfiguráveis às quais serão atribuídas, para cada, uma área reconfigurável na memória da FPGA.

A figura 2.2 mostra a planta da FPGA no *Planahead*, é aqui que são selecionadas e instanciadas as áreas reconfiguráveis.

No *Planahead*, para cada módulo diferente reconfigurável, deve-se criar uma nova configuração. Cada configuração realizará, conforme os módulos escolhidos, implementações diferentes.

A fase final é chamada de implementação. Esta fase engloba várias etapas: tradução(*Translate*), mapeamento(*Map*), colocação e encaminhamento(*Place and Route*) e, finalmente, geração de *bitstreams*.

O processo de tradução, conjuga todas as *netlists* e *constraints* [11] do projeto e gera, como saída, um ficheiro *Native Generic Database (NGD)*. Este ficheiro descreve toda a lógica do *design* utilizando as primitivas da *Xilinx*. É também durante este processo que são adicionados, automaticamente, os pinos de partição para todos os sinais de interface com as partições, que não são lógica global. Os pinos de partição fornecem caminhos conhecidos para a área reconfigurável. Ou seja,

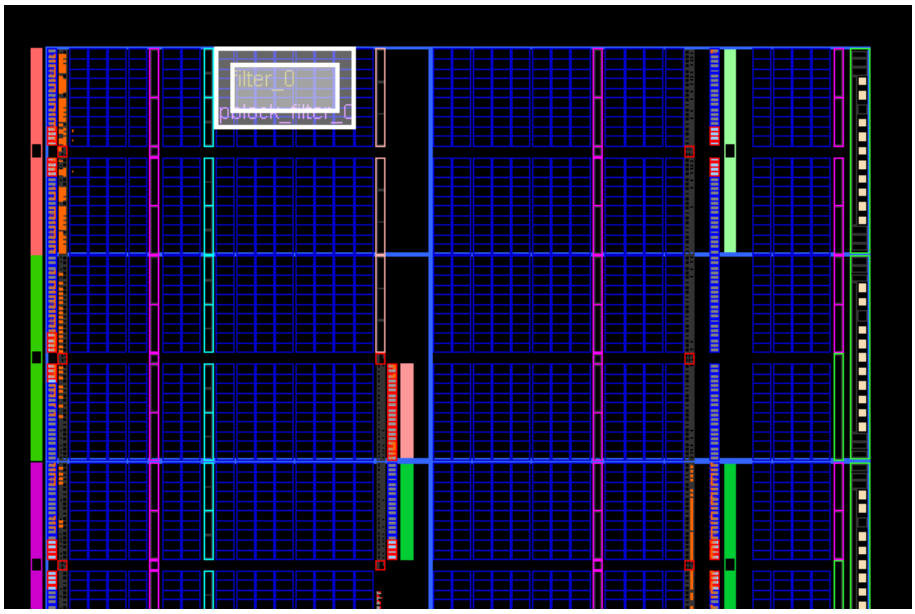

Figura 2.2: *Floorplanning* no *Planahead*.

fazem a conexão física e lógica da área reconfigurável à área estática. No que diz respeito à comunicação com as partições reconfiguráveis, estes pinos são uma tecnologia mais recente em relação à utilização de *bus macros* [12], que não são mais do que fios dedicados à comunicação entre as partições reconfiguráveis e estáticas. Todo este processo é também conhecido como *NGDbuild*.

Na etapa de mapeamento, a lógica descrita no ficheiro NGD é mapeada em elementos da FPGA, como por exemplo, CLBs e IOBs. No final desta etapa é criado um ficheiro *Native Circuit Description (NCD)* que representa fisicamente o circuito mapeado em componentes *Xilinx* da FPGA.

A colocação e encaminhamento (*Place and Route*) é, como o nome refere, composta por duas fases. A colocação é a fase em que é decidido como serão dispostos todos os componentes lógicos no espaço da FPGA. Já a fase de encaminhamento faz a ligação entre os componentes que foram previamente colocados.

Por fim, ainda no *Planahead* e depois de verificadas as configurações, são gerados os *bitstreams* parciais e totais para cada configuração.

Na figura seguinte encontra-se um diagrama representativo do fluxo de projeto descrito:

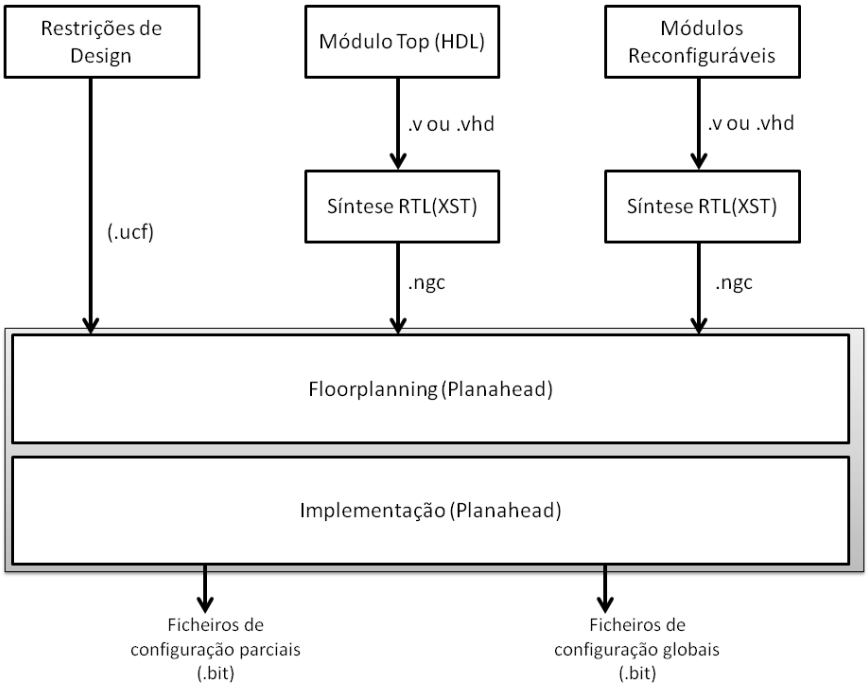

Figura 2.3: Fluxo de projeto de reconfiguração dinâmica.

2.3 Trabalhos Relacionados

O interesse na utilização desta tecnologia, está diretamente relacionado com sua capacidade de aumentar a velocidade computacional na resolução de tarefas. Foi já provado que a computação reconfigurável em FPGA é capaz de produzir este efeito em enumeras aplicações, como por exemplo:

- Sistemas Embarcados: Cada vez mais dispositivos do dia a dia fornecem várias funcionalidades. Um carro tem o Global Positioning System (GPS), rádio, sistemas multimédia, Anti-lock Braking System (ABS) e Electronic Stability Control (ESP). Para que tudo isto seja possível, é necessário que vários sistemas embarcados coexistam. A utilização de computação reconfigurável permite reduzir o número sistemas embarcados dedicados a cada tarefa, uma vez que não precisam de existir todos ao mesmo tempo. Isto permite a redução do consumo de energia e do peso do carro.

- Aplicações de Segurança de Rede: Atualmente existem vários algoritmos de criptografia implementados em redes. Este facto faz com que os sistemas de segurança tenham que adaptar os seus algoritmos de encriptação e desencriptação dinamicamente. A reconfiguração dinâmica é uma solução para este problema, uma vez que não é preciso ter todos os algoritmos conectados ao barramento de rede em simultâneo.

- Aplicações Multimédia: Este tipo de aplicações são caraterizados pela necessidade de processamento de grandes quantidades de informação em tempo real. Existem várias normas de codificação de imagem/vídeo e diferentes necessidades na velocidade de processamento. A computação reconfigurável possibilita soluções para a adaptação às diferentes necessidades com custos reduzidos e elevadas performances.

Nas próximas duas secção estão descritos dois trabalhos que utilizam reconfiguração dinâmica. O primeiro é orientado na área de sistemas embarcados. O segundo está focado numa perspetiva pedagógica.

2.3.1 Sistema para Reconfiguração Dinâmica *On-Demand*

Atualmente, um automóvel já dispõe de imensas unidades de controlo de funcionalidades vocacionadas para conforto dos passageiros. Na verdade, as funcionalidades têm vindo a aumentar com o desenrolar dos tempos e prevê-se que esta tendência se mantenha. Se para cada função existisse uma unidade de controlo dedicada, isto traduzir-se-ia num desperdício de energia, espaço e custos.

Em 2004, foi desenvolvido um trabalho [13] onde foram incorporadas várias funcionalidades de um automóvel, como por exemplo, o controlo elétrico dos estofos e dos vidros das janelas. Neste sistema, implementado na *Xilinx Virtex II*, desenvolveram-se algoritmos que multiplexavam os módulos de *hardware* tendo como objetivo a poupança de energia.

Nesta implementação, de acordo com a tecnologia utilizada na placa, a separação física das áreas reconfiguráveis é realizada através de *bus macros* que conetam as linhas de sinais entre os módulos e o processador que dita a reconfiguração.

Todos os ficheiros *bitstream* parciais, com a informação relativa a cada módulo, encontram-se dentro de um cartão de memória *flash*. Estes ficheiros são carregados pelo *Internal Configuration Access Port (ICAP)* que por sua vez controla a reconfiguração da memória da FPGA.

O sistema de tempo real é implementado no *soft-core MicroBlaze*. Este processador faz a gestão da entrada e saída de mensagens de todo o sistema. As mensagens vindas de periféricos externos são descomprimidas, processadas e enviadas para os módulos reconfiguráveis. Posteriormente, o resultado da função é enviado de volta para o periférico .

A estrutura básica de todo o sistema é dividida em quatro tarefas principais:

- Processo de reconfiguração: O processo de reconfiguração é feito em quatro passos. Primeiramente o sistema procura um *slot* vazio de um total de quatro disponíveis para o posicionamento dos módulos. Se todos os *slots* estiverem ocupados, o sistema procura um módulo que esteja inativo. A reconfiguração com um novo módulo é apenas efetuada caso exista espaço livre ou módulos inativos. Caso contrário o pedido de reconfiguração é guardado em buffer até que seja possível a sua utilização. Depois de selecionado o *slot* onde se realizará a reconfiguração, é guardada a informação de estado do módulo atual. Por fim, o novo módulo é reconfigurado e é carregado com a sua informação de estado guardada previamente. Este

sistema é *event-driven*, ou seja, o processo de reconfiguração é despertado pela chegada de novas mensagens vindas do exterior.

- Gestão da entrada de mensagens: Recebe mensagens, descodifica a informação e converte-as em mensagens internas. Posteriormente, encaminha as mensagens para o módulo a que se destinam.

- Gestão do envio de mensagens: Recebe mensagens dos módulos, codifica-as de forma adequada e envia para os periféricos externos.

- Unidade de gestão do *buffer* de mensagens: Guarda e encaminha as mensagens que não puderam ser entregues no seu módulo de destino, ou porque estes foram desativados, ou porque foram removidos.

Para além das unidades mencionadas acima, existe ainda a rotina de atendimento às interrupções que recebe e guarda as mensagens para mais tarde serem tratadas. Esta rotina é responsável pelo evoluir do sistema (*event-driven*).

Os resultados da implementação deste sistema são ainda preliminares. Contudo, foi provado que esta é uma solução viável. No total existiam quatro *slots* reconfiguráveis na *FPGA* e um total de oito módulos reconfiguráveis. Isto significa que de oito funcionalidades possíveis, quatro delas podem correr paralelamente.

A aplicação deste sistema é mais viável para funcionalidades que não sejam consideradas como criticas do automóvel. Por exemplo, é impensável colocar o controlo do *airbag* como um candidato para se efetuar multiplexagem. Ainda assim, fica provado que é possível agregar várias funcionalidades, não criticas do automóvel, numa FPGA. Com isto, resulta um reaproveitamento do espaço, poupança de energia e diminuição dos custos de construção.

2.3.2 Demonstrador de Reconfiguração Parcial de FPGAs

Recentemente realizou-se um projeto que pretendia implementar um sistema de processamento de vídeo com reconfiguração dinâmica em FPGA [2] também ele com uma orientação pedagógica. Neste sistema, a unidade de processamento de vídeo é reconfigurada em tempo real e tinha como principal objetivo ilustrar de forma educativa, para os utilizadores, o que era a reconfiguração dinâmica.

Esta implementação foi realizada na *Xilinx Virtex ML506*. Para além da placa é também necessária a utilização de um computador e um monitor para visualização do vídeo.

O computador comunica com a placa através de uma comunicação série e desta forma é enviada uma *stream* de vídeo e ordens de reconfiguração. Para além disto, no arranque do sistema, são enviados os *bitreams* para a placa, onde são guardados numa posição de memória definida.

A arquitetura do sistema é separada em duas partes: processamento do vídeo e gestão da reconfiguração.

A unidade de processamento de vídeo recebe informação do descodificador da placa e guarda-a na memória interna no formato de 128 por 128 pixeis. De seguida a informação passa pela módulo de processamento reconfigurável e o resultado é guardado, novamente, na memória interna. Por fim o módulo de saída lê e envia a informação da *frame* para o monitor que se encontra ligado a um controlador *Digital Visual Interface (DVI)*. O grande objetivo passa por realizar a reconfiguração dinâmica da unidade de processamento de vídeo, sem interromper a stream, mantendo 60 *frames* por segundo. Posto isto, são propostos dois tipos de transcodificação: aumento da qualidade da *stream* para 256 por 256 pixeis e quadruplicação da *stream*.

Para se proceder à reconfiguração dinâmica, utilizou-se o processador *MicroBlaze* e o módulo ICAP.

Os *bitstreams* foram gerados seguindo um fluxo de projeto semelhante ao descrito na secção 2.2. Neste projeto a experiência dos alunos tem duas fases: o desenvolvimento e execução de uma aplicação C para transferir os *bitstreams* para a placa e envio de ordens de reconfiguração, sob a forma de TAGS, para o *MicroBlaze* interpretar e executar.

Neste projeto o utilizador constrói uma aplicação em *software* para carregar na placa e executar as reconfigurações que pretende. No projeto que desenvolvi, o utilizador possui uma interface gráfica remota, e é aí que ordena as operações a efetuar e pode observar os seus resultados.

Capítulo 3

Arquitetura do Sistema Base

3.1 Descrição de Alto Nível

Como já foi referido anteriormente, no capítulo 1, a reconfiguração dinâmica é efetuada com a ajuda de um GPP que recebe e processa ordens de reconfiguração. Posto isto, o sistema implementado por mim é composto por três partes distintas: o *software* desenvolvido para a interface de utilizador, o *software* desenvolvido para o *MicroBlaze* e o módulo hardware ao qual irei designar de "Processador de Imagem".

A aplicação de interface remota tem como objetivo o envio das imagens a processar, envio das opções de reconfiguração e demonstração de resultados. Esta aplicação foi desenvolvida utilizando a linguagem JAVA.

A comunicação entre a aplicação remota e a placa *Virtex 5 ML505* é realizada, utilizando *sockets*, através de um protocolo TCP/IP.

No entanto, esta secção encontra-se focada na descrição geral do sistema e do seu fluxo de informação.

O processador *MicroBlaze* funciona como cérebro do sistema base. Recebe informação via *ethernet* e atua consoante essa mesma informação.

Antes de processada, a imagem recebida é armazenada em memória RAM DDR2 e posteriormente transferida para o módulo de processamento de imagem. A transferência da imagem é efetuada de modo progressivo e fraccionado. Ou seja, o módulo Direct Memory Access (DMA) realiza transferências da memória para o módulo e, alternadamente, em sentido contrário. É através deste processo que é efetuado o varrimento da imagem pela cadeia de filtros em série implementados no módulo "Processador de Imagem".

Dentro deste módulo, existem quatro *slots* ligados em série, estes podem ser reconfigurados com o filtro de imagem desejado. A reconfiguração é realizada pelo *core* HWICAP que, ordenado pelo *MicroBlaze*, lê a *bistream* parcial armazenada no cartão de memória e transfere-a para a zona de memória correspondente. Na figura 3.1 as partições onde serão configurados os filtros estão representadas dentro do módulo "Processador de Imagem"pela designação de P1 a P4.

Na figura seguinte estão apenas representados os *cores* suficientes para que seja percetível a sequência de operações do sistema:

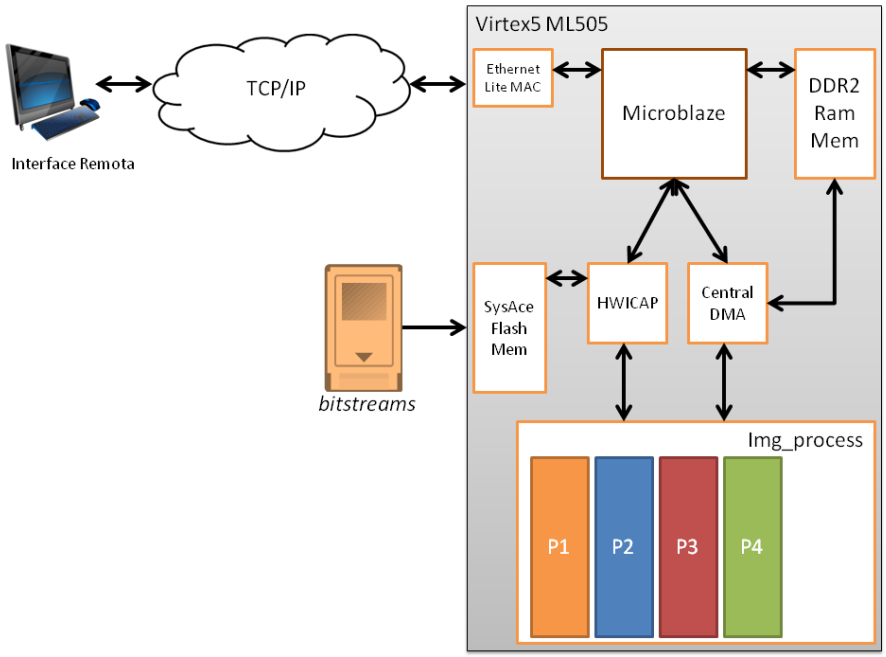

Figura 3.1: Esquema de alto nível do sistema base.

A interface entre todos os módulos é efetuada utilizando o barramento *PLB* [14]. Neste barramento o processador *MicroBlaze* tem o estatuto de *master* enquanto todos os outros *cores*, à exceção do DMA, são *slaves*. O módulo do DMA comporta-se tanto como *slave* ou *master* conforme o estado de operação. Detalhes sobre o seu comportamento são apresentados na secção 4.4.

Na figura 3.2, encontram-se dispostos todos os *IPCores* utilizados na implementação do sistema.

A *Xilinx* disponibiliza uma ferramenta chamada de *Xilinx Platform Studio (XPS)*, onde é possível escolher, através de um catálogo, os *IPCores* que queremos utilizar no sistema base. Por forma a comunicarem entre si, é necessário que todos os *cores* estejam conectados a um barramento. Neste caso utilizou-se o barramento PLB, escolhido e configurado no XPS.

O sistema base construído, que serve de suporte para a implementação de toda a aplicação, contém os seguintes *IPCores*:

- LogiCORE IP XPS UART Lite (v1.02a): Este módulo é usado para imprimir mensagens de estado pela porta série, servindo assim de análise e *debbug*. A velocidade de transmissão

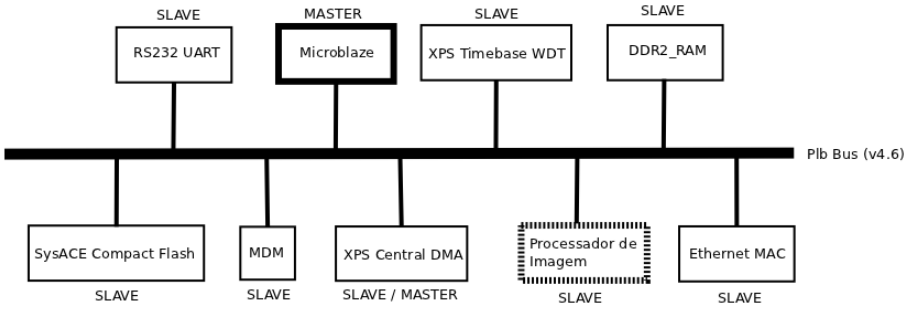

Figura 3.2: Vista do barramento PLB.

usada é de 115200 bits/s;

- LogiCORE IP XPS Watchdog Timer (v1.02a): É um contador que é utilizado para medir o tempo de processamento da imagem no *MicroBlaze*;
- XPS SYSACE (System Ace) Interface Controller (v1.01a): Tem como função a leitura de cartões de memória *flash*. No caso concreto deste projeto, no cartão de memória, encontram-se armazenados os ficheiros de configuração parcial da FPGA;
- LogiCORE IP XPS HWICAP (v5.01a): O ICAP é responsável pela reconfiguração parcial da memória da FPGA. Transfere o *bitstream* parcial para o espaço de memória reservado para reconfiguração;
- LogiCORE IP XPS Ethernet Lite Media Access Controller: Responsável pela comunicação *ethernet*(TCP/IP). A comunicação entre a placa e a interface remota é realizada utilizando este módulo e as bibliotecas de lwip *light weight internet protocol*;
- LogiCORE IP XPS Central DMA Controller (v2.03a): Executa, a pedido, a transferência de informação entre dois espaços de memória endereçáveis. É utilizado para transferir pedaços da imagem armazenados na memória para o módulo "Processador de Imagem"e em sentido contrário também;
- LogiCORE IP XPS Interrupt Controller (v2.01a): Este módulo tem como entrada todos os sinais de interrupção do sistema e como saída apenas uma interrupção. Ou seja, faz a gestão das interrupções de todos os módulos de sistema, escalonando-as por prioridades;
- MicroBlaze Debug Module (MDM) (v2.00b): Permite modo de *debug* por JTAG. Foi utilizado durante o desenvolvimento do sistema.

O *core* "Processador de Imagem"possui uma cadeia de filtros de imagem reconfiguráveis. Este módulo foi desenvolvido por mim e será descrito no capítulo (4).

Capítulo 4

Processador de Imagem

O "Processador de Imagem" é um módulo nuclear no sistema implementado. É aqui que se encontra a funcionalidade base de todo o sistema, ou seja, a cadeia de filtros reconfiguráveis que realiza o processamento da imagem.

Neste módulo, para além de toda a arquitetura lógica de suporte ao processamento de imagem, existem também contadores que permitem medir o desempenho do sistema.

Esta secção do relatório serve para descrever toda a implementação do módulo "Processador de Imagem". Na primeira secção é descrita a funcionalidade e a abordagem que permitiu, a cada filtro, efetuar o varrimento da imagem. Na segunda secção é descrita de forma detalhada a arquitetura em sub-módulos do "Processador de Imagem". Na terceira secção está explicitada toda a lógica envolvida nos contadores de desempenho. Por último, na quarta secção é mostrada a interface das memórias FIFO com o barramento PLB em *burst mode*.

4.1 Propósito e Funcionamento

4.1.1 Alimentação do Filtro

Nas operações locais, em processamento digital de imagem, cada ponto resultado é obtido através de uma função que usa como argumentos o ponto homólogo da imagem original e um conjunto de vizinhos desse mesmo ponto. Ao conjunto total de pontos chama-se de "janela".

Tendo em conta este tipo de operações, decidiu-se desenvolver uma arquitetura que permitisse a utilização de janelas com tamanho 3x3. Assim sendo, cada instância do módulo reconfigurável tem a capacidade de receber 3 bytes em paralelo, em que cada um pertence a uma linha diferente.

Para realizar o varrimento da imagem, cada filtro deve ser precedido de duas memórias FIFO (filas). Desta forma são criadas duas correntes sincronizadas de *bytes*. Sendo que cada corrente pertence a uma linha distinta. Com o intuito de tornar todo o processo síncrono, ordenou-se que a fila superior fará a corrente da linha mais antiga e a fila inferior, fará a corrente da linha intermédia. Sempre que as filas estão em andamento são introduzidos *bytes* no filtro às quais estão ligadas. Para além disso, como a fila intermédia está ligada à fila superior, esta é atualizada com a informação proveniente da fila intermédia. Após a entrada de uma nova linha no processo, a linha

da fila superior deixa de ser considerada e a linha que estava na fila intermédia passa a estar na fila superior.

Na figura 4.1 é apenas considerado, para efeitos demonstrativos, o processo de varrimento da imagem. Não está contemplado no esquema, a origem dos dados nem o destino dos dados processados.

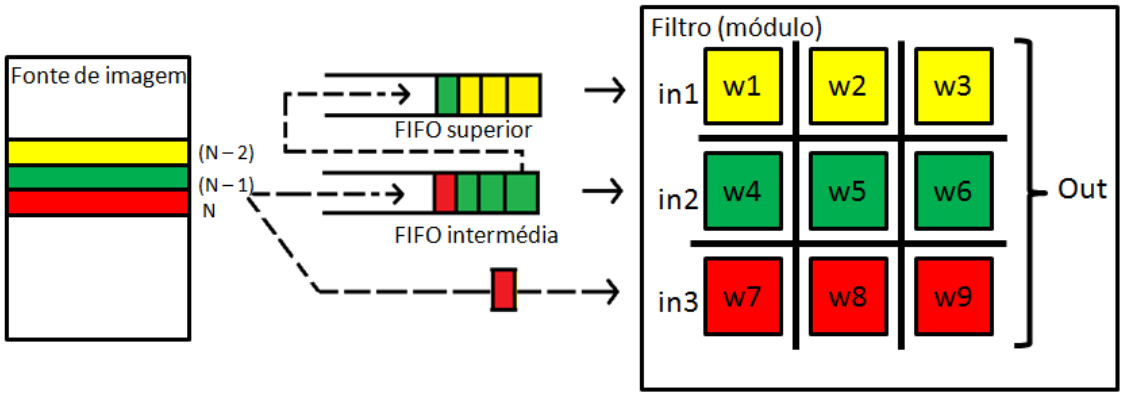

Figura 4.1: Processo de varrimento da imagem.

Desprezando, para já, a estrutura que alimenta as filas, a figura 4.1 representa uma fotografia ao processo quando este está a ler a linha N. As duas filas funcionam como uma memória que guarda e introduz no filtro as duas linhas anteriores à atual.

A linha mais atual a entrar no processo tem entrada direta na porta "in3" e simultaneamente atualiza a fila intermédia. Por sua vez, os *bytes* provenientes desta, atualizam a fila superior. É desta forma que é mantida a hierarquia dos dados da imagem. Ou seja, a fila superior faz sempre a corrente de *bytes* da linha (N-2), a fila intermédia faz a corrente da linha (N-1) e a linha N é passada diretamente da fonte de dados.

4.1.2 Cadeia de Processamento

No sistema implementado, foram encadeados em série quatro módulos reconfiguráveis responsáveis por realizarem as operações na imagem. Como é demonstrado no capitulo 7, através dos resultados da síntese, a sequência de quatro módulos reconfiguráveis é razoável e permite margem de manobra para o desenvolvimento de novos filtros tendo em conta os recursos e o espaço da FPGA.

Inicialmente os *bytes* da imagem são recebidos através do barramento PLB e são armazenados numa memória de entrada do tipo FIFO. Esta é a estrutura que alimenta as memórias FIFO que precedem o primeiro filtro da cadeia, sendo que a entrada dos *bytes* é sempre realizada alimentando a fila intermédia.

Nos andares seguintes (segundo, terceiro e quarto filtro), as filas intermédias são alimentadas pela saída do filtro precedente. No caso do ultimo filtro, a sua saída alimenta a fila que armazena

os dados da imagem já processados e prontos para serem recolhidos, chamada FIFO de saída.

Por ultimo, a estrutura de saída é lida através do barramento PLB.

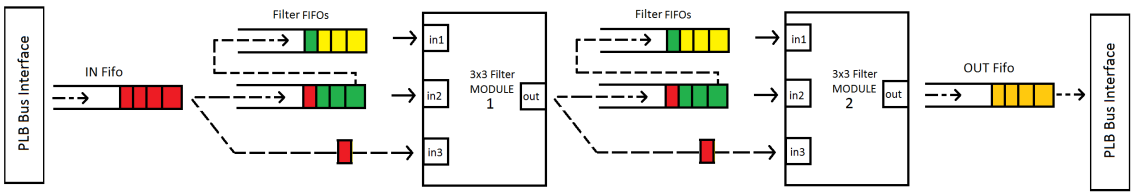

Figura 4.2: Cadeia de processamento

Na figura 4.2 encontra-se apenas representada uma sequência de dois filtros. Contudo, a arquitetura de suporte para a cadeia de quatro filtros é homóloga à demonstrada na figura, sendo apenas necessário acrescentar dois módulos e quatro memórias FIFO, duas precedentes a cada filtro. Estas filas têm o comprimento de 2048 *bytes* e uma largura de 8 bits.

O controlo da cadeia de processamento é realizado por várias máquinas de estado, cada uma dedicada a cada andar de processamento. Este tema é abordado na secção 4.3.

4.2 Arquitetura em Módulos

O "Processador de Imagem" consiste em um módulo superior chamado de *"img_process.vhd"*. Aqui encontram-se instanciados e ligados entre si mais quatro módulos: o *"user_logic.v"*, o *"soft_reset.vhd"*, o *"PLB46_Slave_ Burst.vhd"* e o "*Interrupt_Control.vhd*".

Todos os sub-módulos mencionados são gerados e parametrizados através do XPS. O processador pode enviar uma ordem de *reset* e o módulo "*soft_ reset.vhd*"encarrega-se de gerar o estimulo correspondente deixando o "*user_logic.v*"no seu estado inicial. A recepção e envio de informação entre o módulo e o barramento é efetuada através do adaptador "*PLB46_Slave_Burst.vhd*". No XPS escolheu-se que a comunicação teria suporte para transferências em modo rajada. Este modo é necessário para suportar o tipo de transferências executadas pelo DMA. O módulo "*Interrupt_Control.vhd*"não é utilizado nesta implementação, mas foi deixado propositadamente para uma possível expansão do sistema. Contudo, a sua função é a de gerar uma interrupção em *hardware* para quebrar o funcionamento do processador com uma chamada para atender a sua interrupção.

O módulo *user_logic.v* é onde se encontra implementada toda a lógica para efetuar o processamento de imagem. Ou seja, é aqui que se encontra a cadeia de filtros e todo o suporte para o seu funcionamento.

Como será explicado na secção 6, este módulo recebe conjuntos de *bytes* da imagem a uma cadência que depende do desempenho do DMA e do processador. A transferência é efetuada utilizando os 32bits disponíveis no barramento PLB, ou seja, são recepcionados 4 *bytes* a cada ciclo

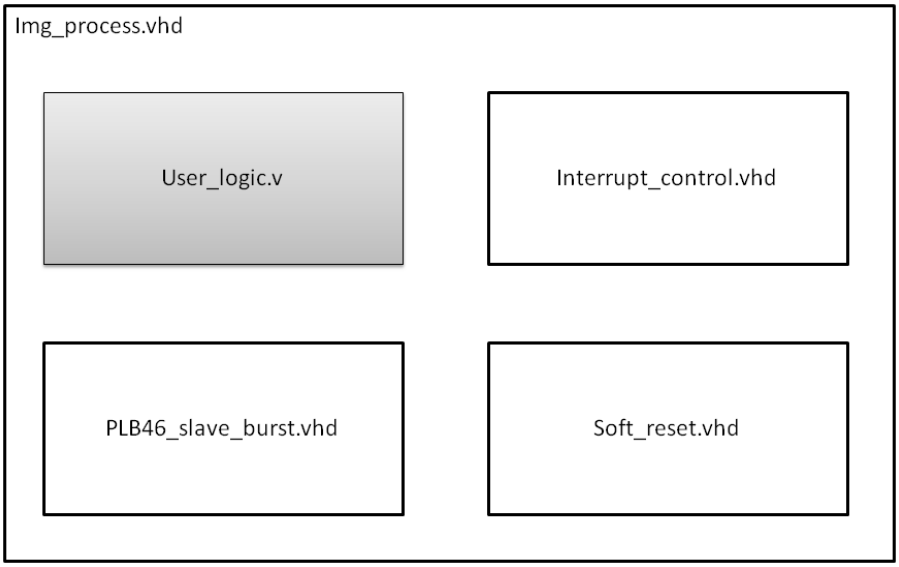

Figura 4.3: Módulo top "Processador de Imagem"

de relógio de leitura. Isto significa que os conjuntos de *bytes* recebidos em rajada serão sempre um total múltiplo de quatro.

A cadeia de filtros implementada recebe informação *byte* a *byte* e, uma vez que a informação se encontra num paralelo de quatro *bytes*, colocou-se, antes da cadeia de filtros, um módulo para converter em série a informação recebida. A FIFO de entrada, que está colocada antes do módulo de conversão série, serve para acumular a informação que chega enquanto se está a processar a conversão.

Depois de processados, os *bytes* são agrupados em conjuntos de quatro e armazenados na memória FIFO de saída, posteriormente o DMA efetua a transferência dos dados aqui armazenados.

Na figura 4.4, estão representados os sinais de entrada e saída do módulo que comunicam com o barramento e os blocos constituintes do módulo *user_logic.v*.

4.2.1 Sinais do PLB

- **Bus2IP_Clk**: Sinal de relógio que alimenta o circuito. Definido como 125Mhz.
- **Bus2IP_Reset**: Sinal *reset*, reinicia o todo o circuito.
- **Bus2IP_Addr**: Com tamanho de 32bits, este sinal pode ser usado para aceder a posições de memória especificas de blocos instanciados dentro do módulo *user_logic.v*. Uma vez que as estruturas de memória utilizadas são do tipo FIFO, este sinal não é utilizado.
- **Bus2IP_CS**: Dentro do módulo existem duas estruturas de memória que podem ser acedidas pelo processador. Este sinal, *chip select* de 2bits, é utilizado para escolher qual das memórias FIFO se quer aceder.

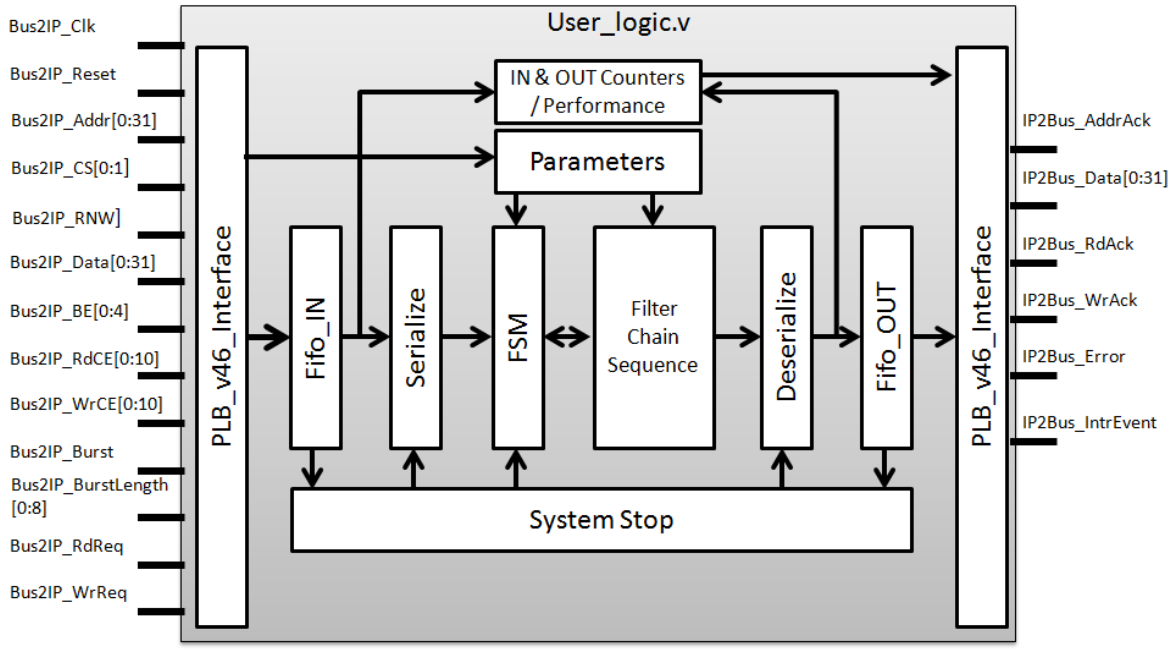

Figura 4.4: Módulo *user_ logic.v*

- **Bus2IP_RNW**: Este é o sinal *Read-Not-Write* e é utilizado pelo barramento para sinalizar que está a efetuar uma leitura. Desta forma faz-se exclusão mutua entre leituras e escritas.
- **Bus2IP_Data**: Com tamanho de 32bits, este é o sinal de onde são lidos os dados para o interior do módulo. No caso do sistema em questão, estes dados podem ser provenientes de transferências do DMA ou do processador.
- **Bus2IP_BE**: Este sinal é chamado de *Byte-Enable* e é utilizado para indicar quais os *bytes* a ler do sinal *Bus2IP_ Data*.
- **Bus2IP_RdCE**: Este sinal permite multiplexar a leitura dos registos escravos (*slv_regs*). Existem 11 registos, logo o sinal tem dimensão de 11 *bits*.
- **Bus2IP_WrCE**: Mesma tarefa e dimensão que o Bus2IP_RdCE, mas direcionado para funcionalidade de escrita.
- **Bus2IP_Burst**: Quando ativado indica que a operação de escrita ou leitura está a utilizar o protocolo em modo *burst* (rajada). É ativado no inicio de uma transferência em rajada e desativado no final do penúltimo compasso da transferência.
- **Bus2IP_BurstLength**: Indica o número de *bytes* associados à transferência em rajada.
- **Bus2IP_RdReq**: Indica que o presente ciclo, é de leitura. No caso de transferências em rajada, o sinal fica ativo durante todo ciclo dessa transferência.
- **Bus2IP_WrReq**: Mesmo comportamento do sinal Bus2IP_RdReq, mas neste caso, aplicado a operações de escrita.

- **IP2Bus_AddrAck**: Quando ativo, incrementa, por cada ciclo de relógio, o contador de endereço da transferência em rajada.
- **IP2Bus_Data**: Este é o sinal associado à saída de informação do módulo. A informação lá contida deve ser válida quando o sinal IP2Bus RdAck está ativo.
- **IP2Bus_RdAck**: Este sinal é utilizado para dar conhecimento de que os dados em Bus2IP_Data foram adquiridos.
- **IP2Bus_WrAck**: Deve ser ativado quando é colocada informação válida no sinal IP2Bus_ Data.

Todos os sinais foram instanciados definidos usando o XPS.

4.3 Caraterização e Descrição dos Módulos

4.3.1 Leitura da FIFO de entrada e conversão Paralelo - Série

Como mencionado anteriormente esta FIFO serve essencialmente para evitar a perda de informação enquanto se dá a conversão série para paralelo. Encontra-se diretamente conectada aos sinais de interface com o barramento PLB, ou seja, recebe diretamente do sinal Bus2IP_Data os dados a armazenar. Na figura 4.5, encontra-se representada a lógica para recepção e armazenamento dos *bytes* da imagem:

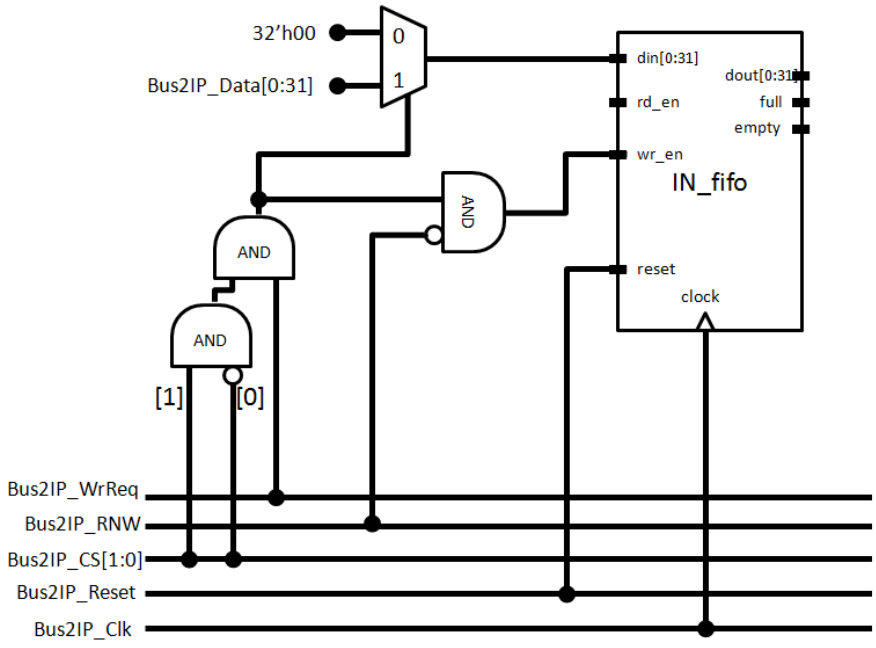

Figura 4.5: Esquema RTL da aquisição de dados do barramento PLB.

A imagem anterior pode ser traduzida explicitando que quando o sinal de entrada "Bus2IP_CS"contém o valor "1"e "Bus2IP_WrReq"está ativo, a informação que se encontra em "Bus2IP_Data"é encaminhada para a entrada "din". Este valor só é guardado na memória FIFO quando a condição anterior se verifica e o sinal "Bus2IP_RNW"se encontra desativo.

Na tabela 4.1 estão representadas as caraterísticas desta estrutura:

Largura da Entrada	32 bit
Largura da Saída	32 bit
Profundidade	4096
Block RAMs Usadas	4 (de 36Kbits)

Tabela 4.1: Caraterísticas da FIFO de entrada.

A leitura e conversão da informação, proveniente da FIFO de entrada, é realizada utilizando três módulos complementares. Dois módulos são geradores de impulsos "*enable*"e o terceiro efetua a respetiva conversão.

O "fifo_read_en_gen"inicia a sua função sempre que a memória FIFO deixa de estar vazia, gerando um pedido de leitura de quatro em quatro ciclos de relógio. Uma que vez que a reposta da memória FIFO é válida um ciclo depois de efetuado o pedido, o impulso que ordena a conversão no módulo de conversão, é gerado através do impulso de pedido de leitura atrasado de um ciclo de relógio, ou seja, introduzindo um registo intermédio. Esta técnica é chamada de *pipelining* e será abordada no capítulo 5.

Na figura seguinte estão representados os módulos para leitura e conversão, bem como as suas ligações entre si:

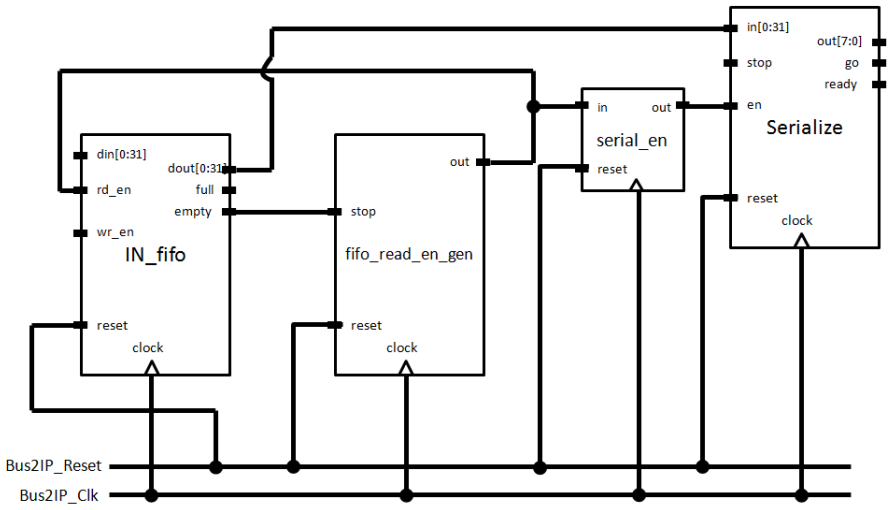

Figura 4.6: Esquemático da conversão Paralelo - Série.

O módulo "*serialize*", sempre que recebe um impulso de "*enable*", inicia uma sequência de quatro operações, onde coloca na saída, a cada ciclo de relógio, um *byte* que é parte do sinal de

entrada. Começa pelo *byte* mais significativo e acaba no menos significativo.

Para além da saída em série *byte* a *byte*, existe um sinal de *ready* e um sinal *go*. O sinal *ready* está ativo sempre que o módulo não se encontra no processo de conversão. O sinal *go* não possui relevância para a funcionalidade aqui descrita, serve apenas para avaliação de condições de congelamento do sistema, que serão abordadas à frente na presente secção.

A figura 4.7, mostra o diagrama temporal do processo de conversão Paralelo - Série:

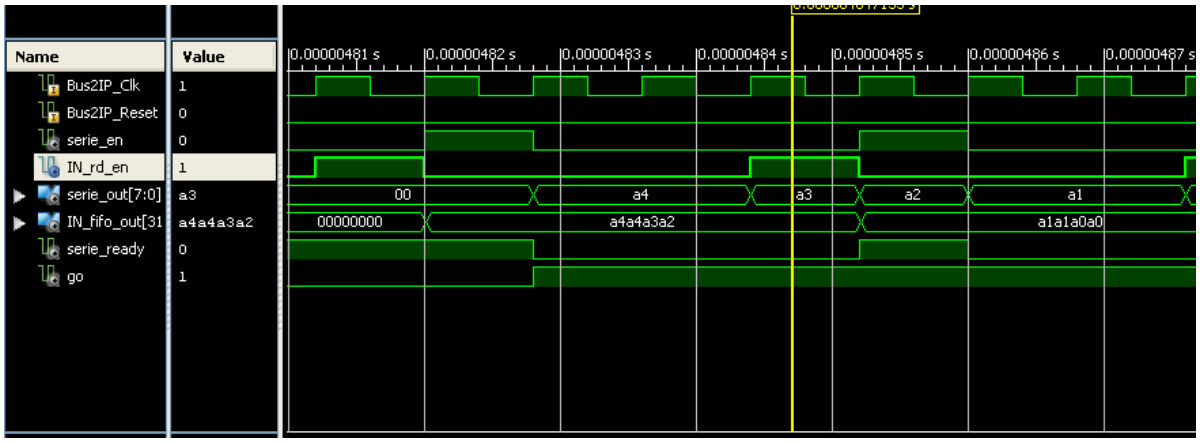

Figura 4.7: Simulação temporal da conversão Paralelo - Série.

O estado e a saída em série, são avaliados pelo módulo FSM, que encaminha os bytes em série para a entrada da FIFO intermédia que antecede o primeiro filtro.

4.3.2 Máquina de Estados e Cadeia de Filtros

A cadeia de filtros está colocada em série e incorpora quatro unidades. Como já foi mencionado, para auxiliar no sincronismo e no varrimento da imagem, precedente a cada filtro estão colocadas duas memórias FIFO, tal como mostra a figura 4.2.

As características de cada memória estão descriminadas na tabela 4.2:

Largura da Entrada	8 bits
Largura da Saída	8 bits
Profundidade	2048
Block RAMs Usadas	1 (de 18Kbits)

Tabela 4.2: Caraterísticas das FIFOs auxiliares de cada filtro.

O processamento dos filtros e o controlo das memórias FIFO é realizado através de uma máquina de estados. Na verdade, o bloco representado na figura 4.4 com o nome "FSM"incorpora quatro máquinas de estado que controlam cada filtro e as suas respetivas filas de forma independente. Embora o controlo do processamento e do preenchimento das filas seja feito da mesma forma para todos os andares de processamento, a utilização de controlo independente, com máquinas de estado individuais, deve-se ao facto de cada andar de processamento estar, a cada instante,

a processar partes da imagem diferentes. Por exemplo, o primeiro filtro pode já ter acabado de processar a imagem toda, mas o ultimo filtro ainda estar no início da antepenúltima linha.

Na figura 4.8, encontra-se ilustrado sob a forma de um diagrama de blocos, o controlo de um dos filtros:

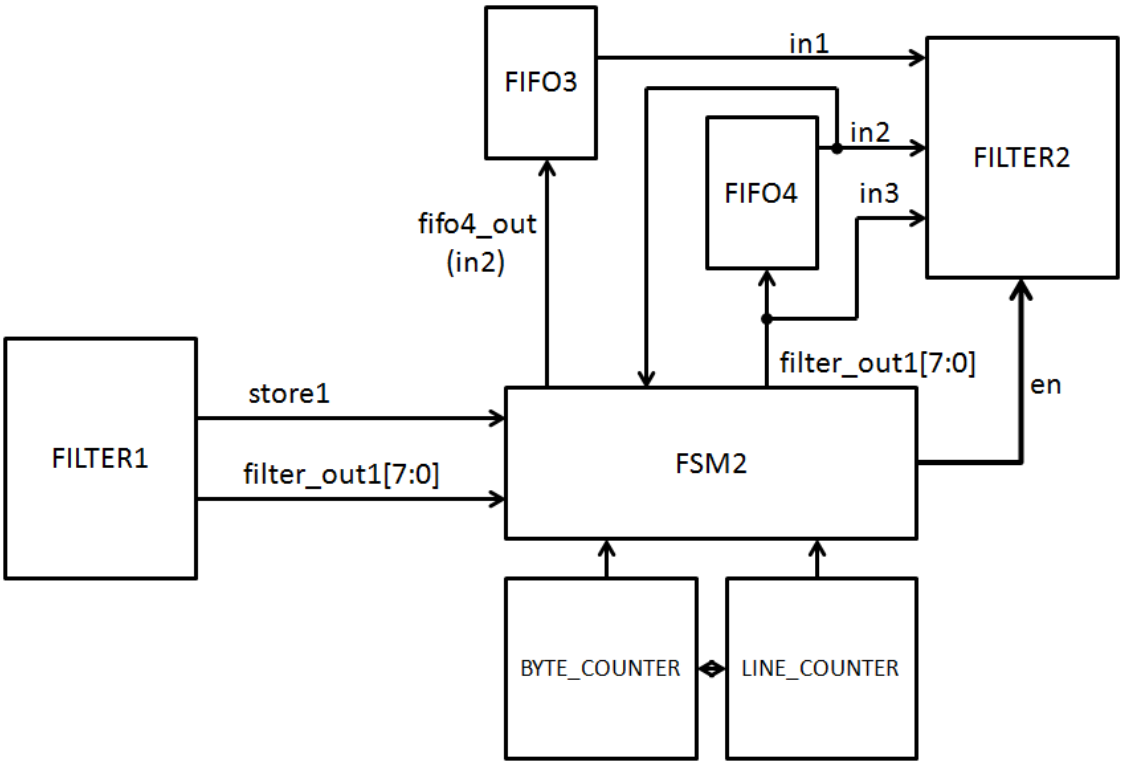

Figura 4.8: Diagrama de blocos do controlo do processamento de um filtro.

O bloco "FSM2"funciona como um multiplexador. Recebe os *bytes* do filtro anterior ou da fila de entrada e coloca-os na fila intermédia que precede o filtro. Para além disso, recebe também os *bytes* provenientes da fila intermédia e encaminha-os para a fila superior. Desta forma, à medida que o processamento é realizado a fila superior é atualizada com a linha que será a mais antiga após finalizado o processamento da linha atual.

Como mostra a figura 4.9, existem quatro estados de processamento: "*WAIT*", "*FILL_FIFO4*", "*PROCESS1*", "*PROCESS2*"e "*FINAL*". A evolução entre os estados acontece mediante o varrimento da imagem, que é medido através dos contadores adjacentes ao módulo "FSM2". Estes contadores indicam a posição na linha e coluna que estão a ser processados naquele momento.

O estado "*WAIT*"espera que o sinal "*store1*"fique ativo. Quando isto acontece, o *byte* que se encontra na saída do filtro é enviado para a fila intermédia e a partir daí, já no estado "*FILL_FIFO4*", carrega-se a FIFO4 com a primeira linha da imagem.

Acabada a etapa anterior, inicia-se o processamento da imagem. Aqui, no estado "*PROCESS1*", o módulo da máquina de estados ativa o sinal de "*enable*"do filtro e começa a descarregar

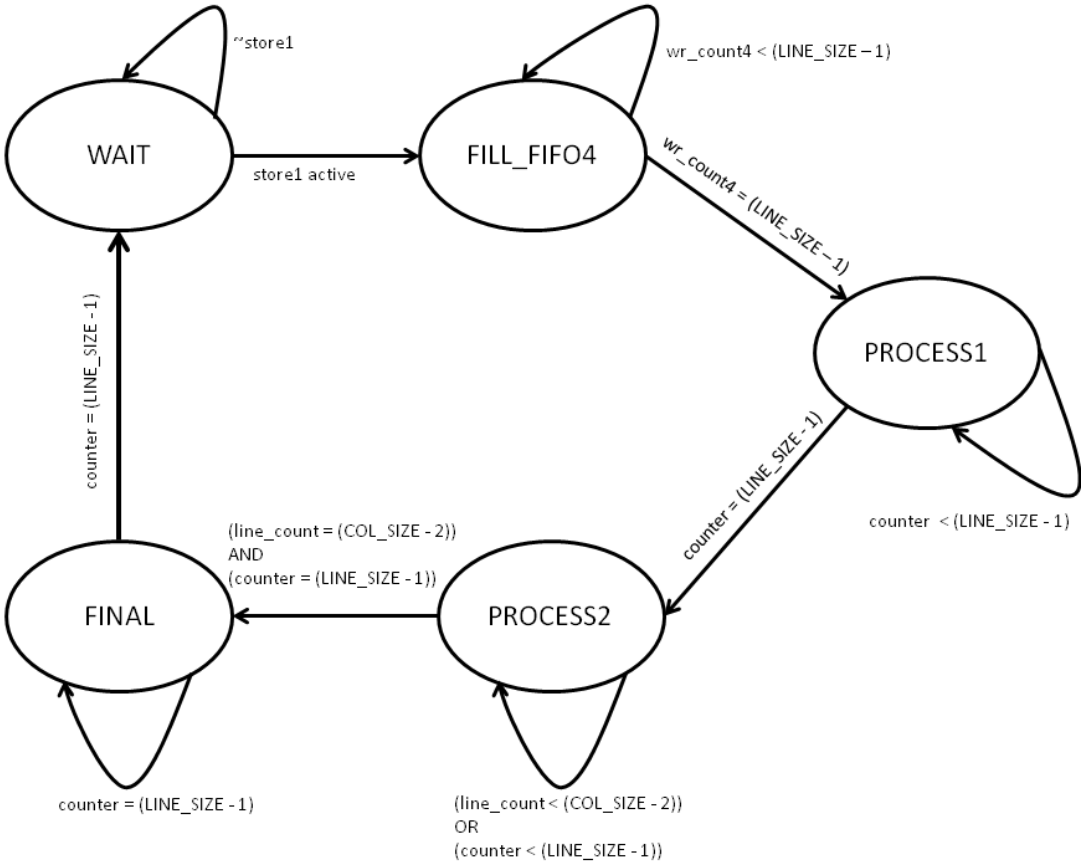

Figura 4.9: Diagrama da máquina de estados.

a fila intermédia. Neste estado, uma vez que a fila superior ainda se encontra vazia, são passadas para o filtro as duas primeiras linhas da imagem. A primeira linha encontra-se na fila intermédia e a segunda linha é passada diretamente do filtro anterior, como aliás esta representado na figura 4.8. Ao mesmo tempo que isto acontece, a fila intermédia é atualizada com a linha dois e a fila superior fica com a linha um. Concluído esta etapa, o próximo estado é o "*PROCESS2*".

No estado "*"PROCESS2*"são passadas para o filtro três linhas simultaneamente. A linha mais recente, a linha intermédia e a linha mais antiga. A linha mais recente é passada diretamente do filtro anterior para a filtro seguinte e simultaneamente atualiza a fila intermédia. A linha intermédia sai da fila do meio para o filtro e atualiza a fila superior. A linha mais antiga é passada pela fila superior. Desta forma é efetuado o varrimento, por linhas, de cima para baixo, do corpo da imagem. Este processo é concluído quando se acaba de processar a penúltima linha.

No último estado, de nome "*FINAL*", são descarregadas as filas. A fila superior contém a penúltima linha e a fila intermédia contem a última. Após conclusão desta etapa, o processamento volta ao estado inicial.

4.3.3 Conversão Série - Paralelo e Armazenamento

Depois de processados, é necessário agrupar os *bytes* sob a forma de quatro em paralelo para serem armazenados na fila de saída. Esta fila está diretamente ligada à interface com o barramento PLB que tem uma largura de 32 bits para leitura de dados.

Para realizar o processo descrito, a saída do ultimo filtro da cadeia de processamento, está ligada a um módulo chamado "*deserialize*". Este módulo recebe uma série de quatro *bytes* e produz o seu equivalente em paralelo. O primeiro *byte* ocupa a posição mais significativa e assim sucessivamente até o último que ocupa a posição menos significativa.

Na figura seguinte está representado um diagrama de blocos que demonstra a conexão e os módulos usados para a conversão:

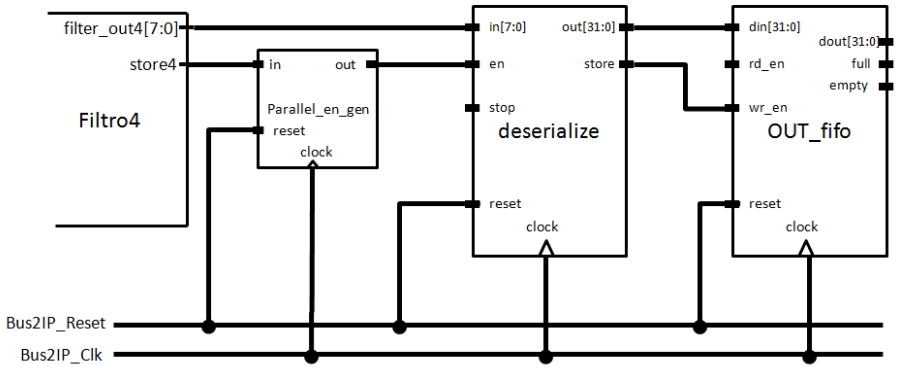

Figura 4.10: Esquema RTL da conversão Série - Paralelo.

A partir do do momento que o sinal "*store4*"se encontra ativo, o módulo "*parallel_en_gen*"gera um impulso de "*enable*"com a duração de um ciclo de relógio e período de quatro. Desta forma, a saída do módulo "*deserialize*"é atualizada de quatro em quatro ciclos de relógio com uma saída de 32 bits que contém os quatro últimos bytes que entraram.

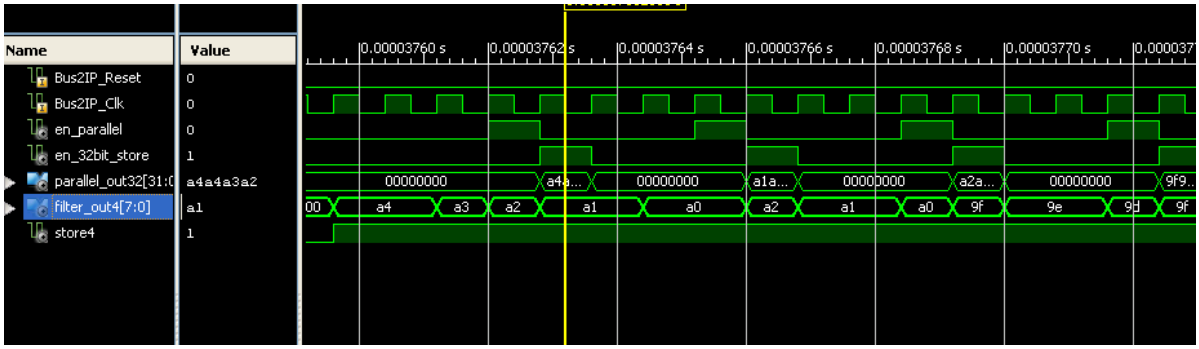

Figura 4.11: Simulação temporal da conversão Série - Paralelo.

O pedido para leitura da fila de saída é efetuado pelo DMA através do barramento PLB. A lógica associada a este pedido é a seguinte:

```
assign rd_en = ((Bus2IP_CS == 2) && (Bus2IP_RNW == 1) && Bus2IP_RdReq) ? 1 : 0;
```

Desenvolvendo o código acima, o impulso do pedido de leitura é efetuado quando o sinal do barramento "*Bus2IP_CS*"seleciona a segunda memória e define um pedido de leitura, ativando o sinal "*Bus2IP_RNW*"e "*Bus2IP_RdReq*".

Como já foi mencionado anteriormente, o módulo possui duas estruturas de memória instanciadas e um banco de registos. Uma vez que, do ponto de vista externo, interessa apenas ler a fila de saída e o banco de registos a escolha entre o que é encaminhado para o barramento é feita utilizando o sinal "*Bus2IP_CS*" à semelhança do que acontece com o pedido de leitura.

```
assign IP2Bus_Data = (Bus2IP_CS == 2) ? OUT_fifo_out : slv_ip2bus_data;
```

4.3.4 Paragem do Sistema

O processamento da imagem é realizado com transferências de tranches para a fila de entrada e consecutiva leitura da tranche processada e armazenada na fila de saída. Para evitar dessincronização e perda de dados quando a fila de entrada estiver vazia ou a fila de saída estiver cheia, o módulo "*system_ stop*"está constantemente a avaliar o estado das filas e lança um sinal de *stop* quando uma das condições se verifica. Este sinal está conetado aos módulos de conversão, tanto Série - Paralelo como Paralelo - Série e ao módulo que possui as máquinas de estado que controlam o varrimento da imagem em cada filtro. Sempre que este sinal está ativo, o processamento é imediatamente congelado no instante em que se encontra. Quando o sinal é desativado o processamento global continua o seu curso normal sem perder sincronismo.

Em resumo as duas condições para paragem do sistema são:

- Memória FIFO de entrada vazia. Esta condição possuí uma exceção: Se a FIFO estiver vazia mas o contador do número de *bytes*, que já saíram da fila de entrada, for igual ao número total de *bytes* da imagem, então significa que não é necessário parar o sistema, uma vez que os *bytes* que faltam processar já se encontram inseridos na cadeia de filtros.

- Memória FIFO de saída cheia. Se esta fila estiver cheia, ao continuar o processamento os novos *bytes processados* não poderão ser armazenados e serão perdidos.

4.3.5 Banco de Registos e Contadores

Quando o módulo "*user_logic.v*"é gerado pelo XPS escolheu-se que este teria um banco de registos com interface com o barramento. Toda a interface de leitura e escrita nos registos foi então

gerada automaticamente.

Mediante a implementação em questão, escolheu-se que seriam necessários um total de onze registos de 32 bits. Estes registos são utilizados para passar valores de controlo para lógica interna do módulo e são chamados de "*slv_reg*".

Parâmetros de registos:

- "*IMAGE_SIZE*": É o tamanho da imagem e é armazenado no registo "*slv_reg0*". Este registo está ligado à entrada da máquina de estados e a todos os filtros.
- "*Level1*"a "*Level4*": Cada um destes registos está ligado ao seu filtro correspondente. Armazenam um valor para efetuar operações pontuais na imagem. Por exemplo: Somar a cada *byte* da imagem o valor de "*Level*". "*Level1*"e "*Level2*"estão armazenados em "*slv_reg3*"e "*slv_reg4*"respetivamente. Já "*Level3*"e "*Level4*"estão armazenados em "*slv_reg7*"e "*slv_reg8*".
- "*LINE_SIZE*": É o parâmetro de largura da imagem e encontra-se no "*slv_reg5*". Este registo está conetado à máquina de estados e a todos os filtros.
- "*COL_SIZE*": É o parâmetro de altura da imagem e encontra-se no "*slv_reg6*". Este registo está conetado à máquina de estados e em todos os filtros.

Contadores:

No total existem quatro contadores com interface com o PLB. Dois dos contadores medem o tempo que demora a processar uma imagem e os outros dois contam o número de *bytes* à saída da fila de entrada e à entrada da fila de saída.

Os contadores associados às memórias FIFO estão na posição do "*slv_reg1*"e "*slv_reg2*".

Para medir o tempo de processamento, decidiu-se utilizar dois contadores: um contador efetivo e outro chamado de contador total.

O contador efetivo mede o tempo em que efetivamente existe processamento, ou seja, no caso do sistema parar, o contador também pára. Este contador dá a noção do tempo de processamento que se obteria caso não houvesse atrasos por parte do *software* na leitura e escrita das tranches da imagem.

O contador total conta o tempo total que se demora a processar imagem contabilizando os tempos parados. Ou seja, o tempo total despendido desde que entra o primeiro *byte* até que que sai o último.

4.4 Funcionamento do ICAP e DMA

4.4.1 XPS Central DMA

A FPGA da placa *Virtex 5* contém apenas 2160Kbits de memória RAM. Isto torna impossível o armazenamento de imagens com mais de 270000 pixeis. Por essa razão decidiu-se que a imagem

seria guardada na memória DDR2 e seria utilizado o módulo DMA para transferir-la.

O DMA fornece um serviço de acesso direto à memória para periféricos conetados no PLB. A funcionalidade deste controlador é a transferência de dados entre dois endereços, a origem e o destino, sem intervenção do processador.

A arquitetura do DMA é composta por quatro módulos principais: interface *slave*, interface *master*, *FIFO* e *Data Realignment Module (DRE)*.

A interface *slave* é responsável por interagir com o barramento PLB na escrita e na leitura dos registos do DMA. Desta forma, durante o decorrer de uma transferência são alterados: os endereços de origem e destino, o comprimento da transferência e os registos de *status*. A interface *slave* é ainda responsável por gerar interrupções baseadas na conclusão da operação ou caso ocorra um erro na transferência.

A interface *Master* é responsável por efetuar as transações de informação especificadas pelos dois endereços e pelo comprimento. As transferências são realizadas usando, quando necessário, rajadas. É, por isso, imperativo que os módulos de destino e origem suportem transferências em *burst mode*. Para além da funcionalidade de transferência, esta interface também deteta erros e atualiza o endereço de destino, o endereço origem, o registo de comprimento e o registo de *status*, no decorrer da operação. À medida que são transferidos *bytes* de informação, o registo de comprimento é decrementado.

Antes de colocada na memória FIFO, a informação proveniente da origem é ordenada pelo módulo interno DRE.

A memória FIFO interna do controlador armazena a informação para ser transferida localmente. Pode ser configurada com profundidade de 1, 16, 32, ou 48. Desta forma a memória FIFO proporciona que possam ser feitas leituras e escritas em simultâneo conforme o seu estado de preenchimento.

A operação de transferência é iniciada quando são escritos os registos de endereço de origem, endereço de destino, registo de controlo e o registo que indica o comprimento da transferência. Primeiramente a informação é lida do endereço de origem para a FIFO interna e, de seguida, escrita da FIFO para o endereço de destino. Com o objetivo de aumentar a eficiência, o módulo DMA tenta, sempre que possível, realizar transferências em rajadas (*burst*). Se ocorrer algum erro durante a transferência, esta é abortada no ponto em que se sucedeu o erro e é reportado através do registo de *status* do DMA. Se não ocorrer nenhum erro, a transferência procede com sucesso até ao final e o no registo de comprimento estará com o valor zero. [15]

4.4.2 HWICAP

O módulo de *hardware* XPS HWICAP permite a um *soft-core*, neste caso o *MicroBlaze*, efetuar leituras e escritas na memória de configuração da FPGA. Este acesso é feito através da porta de acesso de configuração interna, denominada de ICAP, que permite ao utilizador escrever um programa em *software* ordenando que o processador modifique o circuito implementado na FPGA.

A arquitetura do ICAP é composta essencialmente por dois módulos: o módulo de interface com o barramento PLB e módulo HWICAP.

A interface com o barramento é bidireccional e suporta transferências em rajadas.

O módulo HWICAP possui memórias FIFO de escrita e de leitura onde é armazenada a configuração localmente. Para realizar a reconfiguração, o processador escreve para a memória FIFO de escrita os dados relativos à configuração e esta é transferida simultaneamente para a memória da FPGA. É também possível o processador ler a configuração presente em memória, a pedido, o ICAP coloca a informação na memória FIFO de leitura que fica assim disponível para o processador ler.

A frequência máxima de operação do ICAP, varia conforme a FPGA de utilizada. Na *Virtex 5*, a frequência máxima é de 100Mhz. Se a frequência a que opera o barramento for inferior a 100Mhz, então a frequência máxima do ICAP baixa para a mesma a que se encontra o barramento [16].

A figura 4.12, realça a circunstância em que é usado o módulo ICAP no sistema:

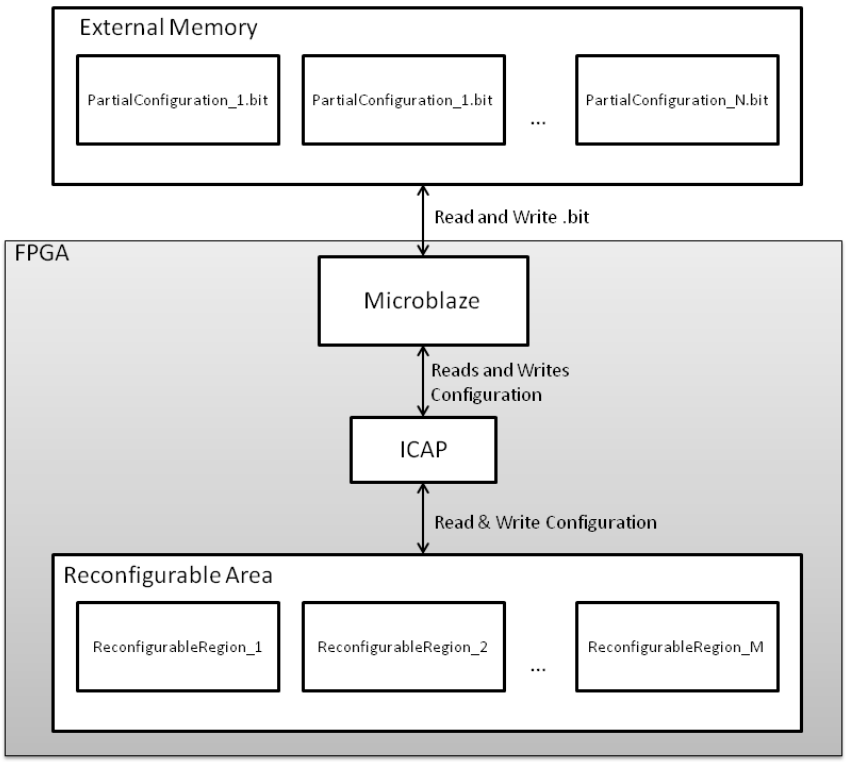

Figura 4.12: Diagrama de fluxo do processo de reconfiguração.

Capítulo 5

Implementação e Desenvolvimento de Filtros

No capitulo anterior falou-se sobre a implementação do módulo "Processador de Imagem". Dentro deste módulo existem quatro partições, cada uma correspondente a um filtro, instanciadas como caixas negras. Neste capítulo, discutem-se os conceitos adjacentes ao desenvolvimento de filtros que serão aplicáveis às partições reconfiguráveis.

Na secção 5.1 é descrito o modelo que é usado para o desenvolvimento de novos filtros. Na secção 5.2 é discutida a arquitetura que a lógica de cada filtro deve seguir. Na secção 5.3 é referido o processo de geração de *bitstreams* parciais.

5.1 Modelo do Filtro

A lógica para implementação de um filtro é sempre construída tendo por base um modelo já definido. Este modelo consiste numa caixa negra que se encontra instanciada quatro vezes no módulo *user_logic*. Como mostra a secção de código seguinte, a caixa negra consiste na definição dos sinais de entrada e de saída do módulo, deixando-o vazio de lógica.

```
1 module filter(clock,reset,IMAGE_SIZE, COL_SIZE, LINE_SIZE,en,in1,in2,in3,level,
      store,out);

3 input clock, reset, en;
  input [7:0] in1, in2, in3,level;
5 input [31:0] IMAGE_SIZE, COL_SIZE, LINE_SIZE;
  output [7:0] out;
7 output store;

9 endmodule
```

No *Planahead* são definidas as partições reconfiguráveis adicionando as *netlists* construídas a partir da descrição dos filtros a implementar.

Os sinais *IMAGE_SIZE*, *COL_SIZE* e *LINE_SIZE* são parâmetros de informação relativa à imagem que se encontra a ser processada e encontram-se no banco de registos que é preenchido pelo processador. Estes sinais contêm os valores do tamanho total da imagem, largura e altura respetivamente.

Os *bytes* da imagem são colocados, com auxílio das filas de espera nas entradas *in1*, *in2* e *in3*. Sendo que a entrada *in1* está conectada com a fila superior e possui sempre a linha mais antiga, a entrada *in2* está conectada com a fila intermédia estando associada à penúltima linha e a entrada *in3* está conectada com a saída do módulo anterior e por isso recebe os *bytes* da linha mais recente.

O sinal *level* é alimentado por um dos registos do banco que se encontra no módulo *user_logic.v* e serve como parâmetro que o utilizador pode usar de diversas formas. Por exemplo, pode ser uma constante para realizar operações pontuais, como clarear a imagem somando a cada *byte* o valor *level*.

O sinal *out* é definido consoante a lógica implementada e é o resultado do processamento do filtro. O sinal *store* deve ser ativo a partir do momento em que o filtro produz resultados. Este sinal serve para informar a estrutura que se segue que o resultado contido em *out* é válido para ser armazenado. A partir deste momento, a cada ciclo de relógio, o filtro deve produzir uma nova saída válida e o *store* deve ficar sempre ativo. Isto apenas é interrompido caso o sinal *en* fique desativo, uma vez que este sinal é responsável por acionar e congelar o filtro.

A arquitetura implementada foi projetada tendo em vista dois tipos de operações em imagem: pontuais e locais. Contudo existe liberdade para o desenvolvimento de futuro de novos filtros que possam diferir no tipo de operações. A instância deve possuir sempre o cabeçalho referido em 5.1, mas a lógica descrita dentro do módulo pode tomar várias formas. Contudo, para que os resultados sejam os esperados, devem-se cumprir com as regras de funcionamento da arquitetura.

Embora cada filtro implementado tenha caraterísticas diferentes, a lógica que suporta a sua operação pode ser semelhante. Principalmente quando são filtros que realizam o mesmo tipo de operação.

Para operações pontuais, em que não é considerada nenhuma vizinhança, utilizou-se a entrada *in2* para ler desde o primeiro ao último *byte* da imagem. A secção de código seguinte mostra a definição da saída de um filtro de *threshold*:

```
...
if(en)
begin
  out <= (in2 >= lvl) ? 8'hFF : 0;
  store <= 1;
end
else
  store <= 0;
...
```

Trata-se de um filtro simples que consiste em avaliar o *byte* da imagem e decidir se naquela posição se irá colocar preto ou branco no pixel resultado. Outras operações pontuais podem ser

realizadas de forma análoga à descrita.

Para a realização de operações locais com janelas de 3x3, é necessário armazenar para cada linha e sob a forma de registos, os três *bytes* anteriores ao atual. Isto pode ser feito com a ajuda de *shift-registers*. Desta forma, as operações com a janela são sempre realizadas sem a utilização direta das entradas. Isto evita violações temporais por parte dos módulos reconfiguráveis.

Considere-se a janela, representada na tabela 5.1, em que os coeficientes estão descriminados de w1 até w9:

w1	w2	w3
w4	w5	w6
w7	w8	w9

Tabela 5.1: Janela de coeficientes genéricos.

Para se realizar operações com uma janela do género da anterior, é necessário que a cada ciclo de relógio sejam lidos um total de 9 *bytes* da imagem. Ou seja, o *byte* central mais os que o rodeiam. Para este efeito, desenvolveu-se a arquitetura descrita na figura 5.1:

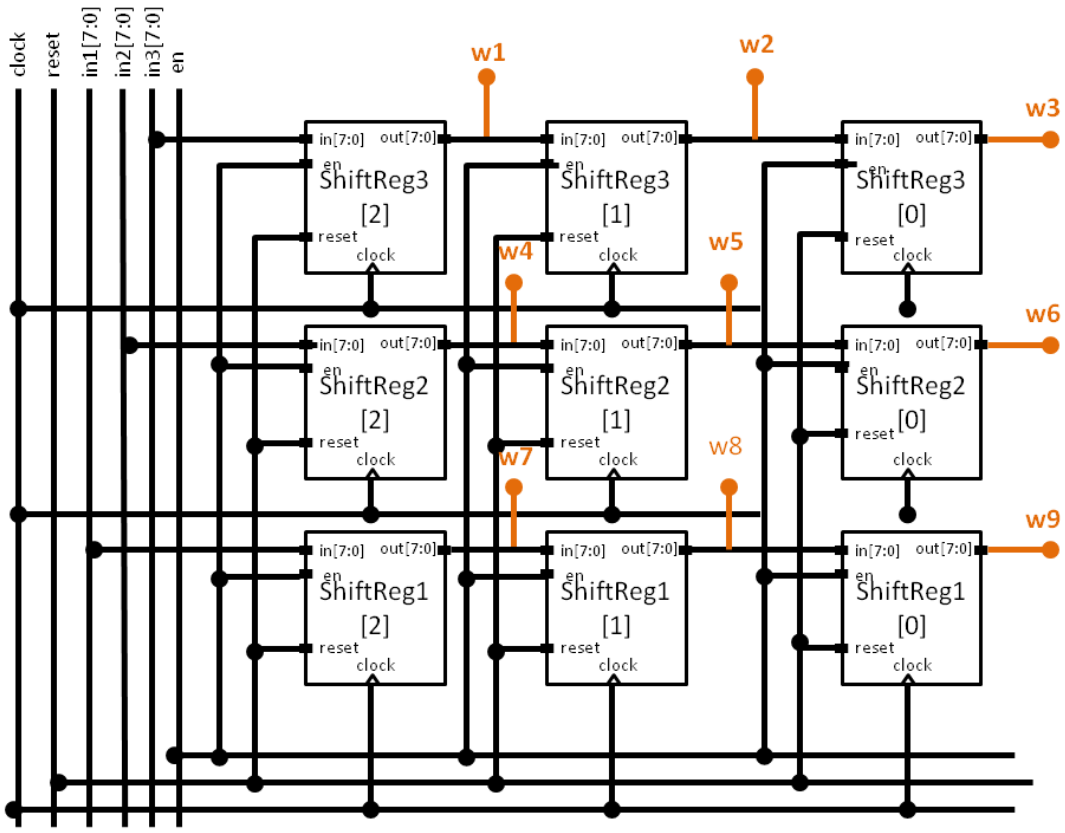

Figura 5.1: Esquema RTL para aquisição e leitura dos pixel da janela.

Para cada entrada, associada a uma linha, são guardados os três *bytes* que precedem o que se encontra naquele momento na entrada. Desta forma, por linha e a cada ciclo de relógio, existem

três*bytes*, preenchendo assim as posições da janela mencionada.

A cada ciclo de relógio com o sinal *en* ativo, realiza-se o deslocamento dos *bytes*. O registo da esquerda é atualizado com o valor que estava na entrada no ciclo de relógio anterior, o registo intermédio é atualizado com o valor que estava anteriormente no registo da esquerda e o registo da direita é atualizado com o valor que se encontrava, no ciclo de relógio anterior, no registo intermédio. Entretanto na entrada já surgiu um novo *byte*. Fica assim criado todo o suporte necessário para se realizar operações locais com janelas de vizinhança 3x3.

5.2 Arquitetura em *Pipeline*

Quando a lógica que define a saída do filtro é extensa, pode provocar atrasos na excursão do sinal. No desenvolvimento dos filtros foi necessário ter isto em consideração para que a sua implementação não baixasse o desempenho do sistema para uma frequência abaixo dos 125Mhz, que é a frequência à qual é alimentado o módulo. Nesse sentido, verificou-se que a utilização de uma arquitetura em *pipeline* ajudaria a diminuir os atrasos dos caminhos críticos, o que consequentemente levou ao aumento da frequência.

Tomemos, como exemplo, o desenvolvimento de um filtro de janela *"Laplace"*, com os seguintes coeficientes:

1	1	1
1	-8	1
1	1	1

Tabela 5.2: Coeficientes do filtro *Laplace*.

Numa arquitetura não *pipeline*, utilizando diretamente as entradas e dois registos por linha, a definição da saída do filtro seria dada por:

```
...
  out <= shiftreg1[0] + shiftreg1[1] + in1 +
       shiftreg2[0] - (shiftreg1[1] << 3) + in2 +
       shiftreg3[0] + shiftreg3[1] + in3;
...
```

Como se pode constatar, para definir a saída são necessárias sete somas, uma subtração e um *shift* de 3 bits. Tudo executado em um ciclo de relógio pode provocar atrasos que violem as restrições temporais, nomeadamente o tempo de *setup* e *hold*.

Para este caso, a frequência máxima obtida após a síntese RTL, foi de 130,005Mhz, ou seja, um período minimo de cerca de 7,692ns. Este resultado é perigosamente próximo da frequência de funcionamento do PLB (125Mhz), o que significa que este *design* é susceptível ao aparecimento de violações temporais na fase de colocação e encaminhamento.

A técnica *pipeline* consiste em partir a instrução que define a saída em vários degraus. Desta forma a lógica associada à instrução fica distribuída por várias etapas. Em cada etapa os resultados são guardados em registos intermédios e a cada ciclo de relógio esse registo é atualizado com um novo valor calculado com os valores do degrau anterior.

A figura seguinte é um diagrama que mostra as operações pelos vários degraus e os registos intermédios até à saída do filtro:

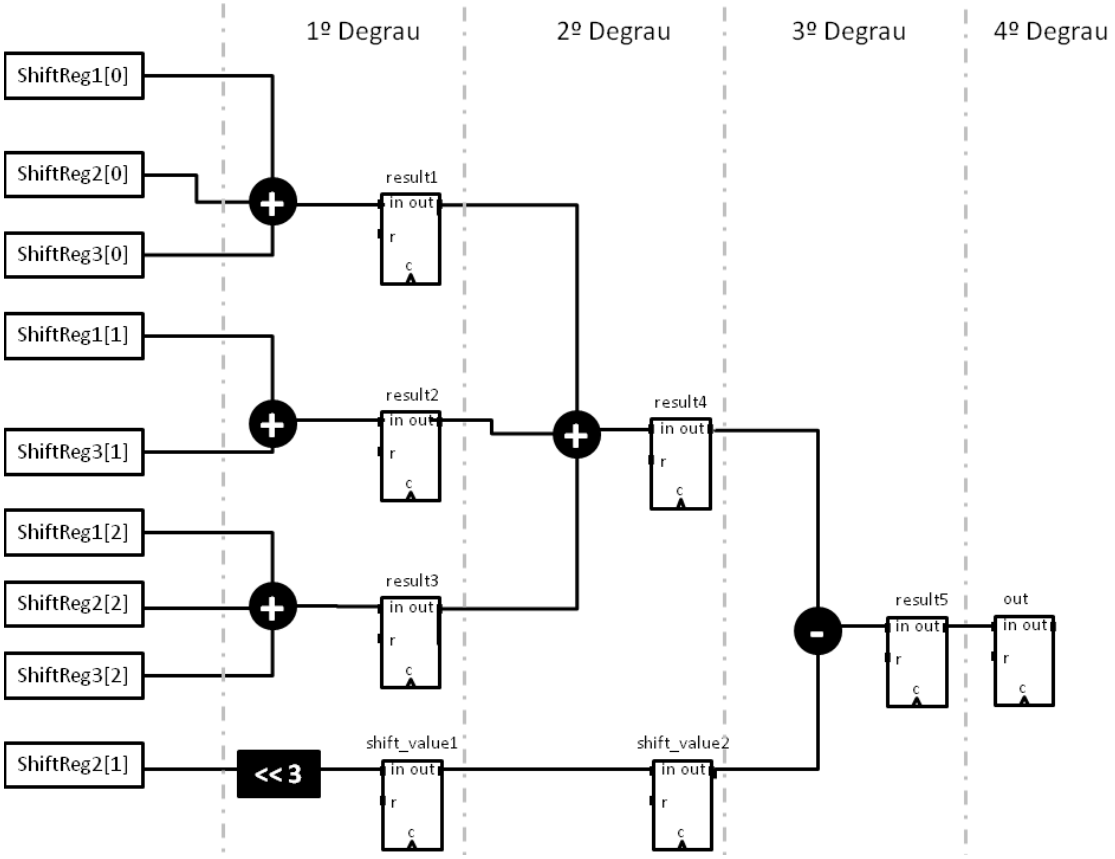

Figura 5.2: Cálculo da saída em arquitetura *pipeline*.

Este tipo de arquitetura introduz um atraso inicial, mas após a primeira saída, produz um novo resultado a cada ciclo de relógio. Neste caso, de acordo com o que foi implementado, o primeiro *byte* processado demora quatro ciclos de relógio até estar na saída do filtro.

Para além dos registos intermédios para efetuar o *pipeline*, os parâmetros de entrada estão também armazenados em registos dentro do módulo do filtro. Como na fase de colocação e encaminhamento os sinais podem ser mapeados em locais distantes das partições dos filtros, a colocação de registos às entradas dos filtros evita que aconteçam violações temporais.

Após a síntese RTL, verificou-se que com este tipo de arquitetura, a frequência máxima subiu de 130,005Mhz para 150,875MHz. Tendo em conta os 125Mhz da frequência de funcionamento do módulo, conseguiu-se atingir uma frequência máxima que inspira confiança em superar todas das restrições temporais.

5.3 *Floorplanning*

A etapa de *floorplanning* consiste na definição das partições reconfiguráveis e das suas restrições de área. Na execução desta etapa utilizou-se a ferramenta *Xilinx Planahead*.

Ao adicionar as *netlists* do sistema base foram detetadas, tal como previsto, quatro caixas negras. Isto deve-se ao facto de os módulos dos filtros estarem instanciados apenas com a declaração das entradas e saídas do bloco.

Definiram-se as caixas negras como partições reconfiguráveis e adicionaram-se as *netlists* de cada filtro a cada uma das partições.

De seguida procedeu-se á definição do posicionamento e da área das partições. Como mostra a figura 5.3, os filtros estão afastados uns dos outros. Com isto aumenta-se a robustez da reconfiguração das áreas reconfiguráveis. Por outro lado, não existe perigo de violações temporais por caminhos extensos, porque todas as entradas encontram-se *pipelined*.

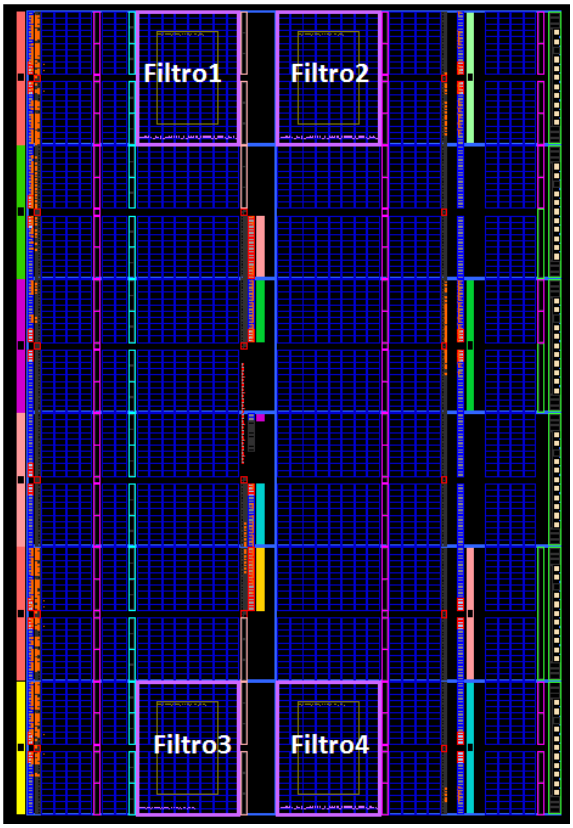

Figura 5.3: Vista da área e dos recursos da FPGA.

Todas as áreas definidas têm a mesma dimensão. Na tabela 5.3 encontram-se os recursos disponíveis em cada área e uma estimativa dos recursos necessários para a implementação do filtro *Laplace*.

Na prática, a definição das áreas reconfiguráveis é o mesmo que adicionar ao ficheiro de restrições (*.ucf*), as instâncias reconfiguráveis bem como as coordenadas que delimitam as suas áreas.

Recurso Físico	Disponível	Requerido	Utilização (%)
LUT	1280	206	17
FD_ LD	1280	215	17
SLICEL	240	50	21
SLICEM	80	17	22

Tabela 5.3: Estatísticas da área reconfigurável definida.

Na secção seguinte, é mostrada, a declaração das áreas reconfiguráveis no ficheiro "*.ucf*".

```
...
INST "img_process_0/img_process_0/USER_LOGIC_I/filter_four_1" AREA_GROUP = "
    pblock_img_process_0_USER_LOGIC_I_filter_four_1";
AREA_GROUP "pblock_img_process_0_USER_LOGIC_I_filter_four_1" RANGE=SLICE_X28Y0:
    SLICE_X43Y19;
INST "img_process_0/img_process_0/USER_LOGIC_I/filter_one_1" AREA_GROUP = "
    pblock_img_process_0_USER_LOGIC_I_filter_one_1";
AREA_GROUP "pblock_img_process_0_USER_LOGIC_I_filter_one_1" RANGE=SLICE_X12Y100
    :SLICE_X27Y119;
INST "img_process_0/img_process_0/USER_LOGIC_I/filter_three_1" AREA_GROUP = "
    pblock_img_process_0_USER_LOGIC_I_filter_three_1";
AREA_GROUP "pblock_img_process_0_USER_LOGIC_I_filter_three_1" RANGE=SLICE_X12Y0
    :SLICE_X27Y19;
INST "img_process_0/img_process_0/USER_LOGIC_I/filter_two_1" AREA_GROUP = "
    pblock_img_process_0_USER_LOGIC_I_filter_two_1";
AREA_GROUP "pblock_img_process_0_USER_LOGIC_I_filter_two_1" RANGE=SLICE_X28Y100
    :SLICE_X43Y119;
...
```

No caso da implementação do filtro *Laplace*, a área total utilizada na FPGA, pelo sistema completo, é de cerca de 39%.

Na figura 5.4 é mostrada a FPGA após a implementação. Aqui é possível observar que as áreas definidas para reconfiguração parcial não foram utilizadas para implementação de lógica estática. Toda a lógica que se encontra dentro de cada partição pertence ao filtro "*Laplace*".

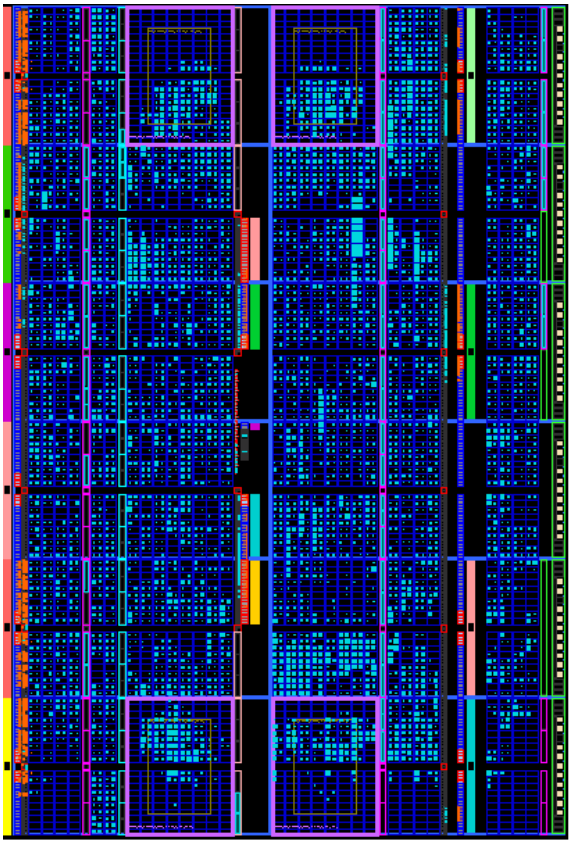

Figura 5.4: Vista da FPGA após implementação.

Capítulo 6

Implementação do Software de Comunicação e Controlo

Neste capítulo é descrito o *software* implementado no processador *microblaze* e na interface remota realizada em JAVA.

Na secção 6.1 explicitado o protocolo de comunicação entre a interface e o processador. Na secção 6.2 está descrita a forma como o protocolo é cumprido pelo processador. Na secção 6.3 é discutido a implementação da cadeia de filtros em *software*. E, por último, na secção 6.4 é mostrada a interface remota.

6.1 Protocolo de Comunicação

Na concepção do sistema, idealizou-se que a interface remota ofereceria ao utilizador a possibilidade de escolher os filtros e de submeter imagens para serem processadas. Tendo isso em conta, desenvolveu-se um protocolo de comunicação entre a interface e placa de desenvolvimento que fosse capaz de enviar a escolha de filtros, os seus parâmetros, enviar a imagem a ser processada e receber a imagem resultado, bem como as respetivas estatísticas de processamento.

Convencionou-se que a interface remota teria o papel de cliente e o servidor seria implementado na placa. O processador está continuamente à espera de receber a trama de configuração para atualizar as variáveis de controlo. A correta receção desta trama desperta o funcionamento de todo o processamento. De seguida o processador fica à espera de receber a imagem que irá ser processada.

A figura 6.1 mostra a troca de mensagens necessária, entre a interface e a placa, que realizam todo o processo do envio de uma nova imagem e recepção do resultado:

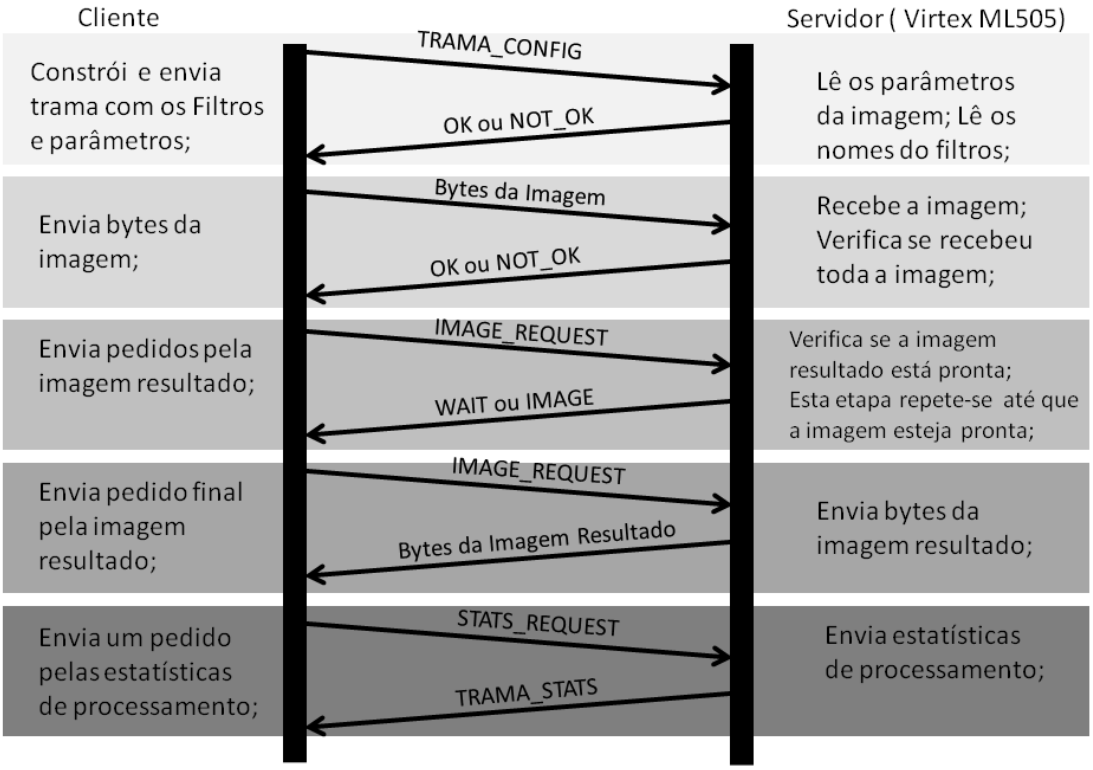

Figura 6.1: Protocolo de comunicação entre a interface e o processador.

A trama de configuração consiste em várias partes: a primeira palavra define se o processamento da imagem deverá ser realizado em *hardware* ou *software*, segue a largura e a altura da imagem, por último encontram-se os nomes dos ficheiros de configuração dos quatro filtros intercalados pelo parâmetro *level* que tem entrada em cada filtro. Este parâmetro só surtirá efeito caso o implementação do filtro lhe dê uso.

A próxima figura mostra um exemplo de uma trama de configuração:

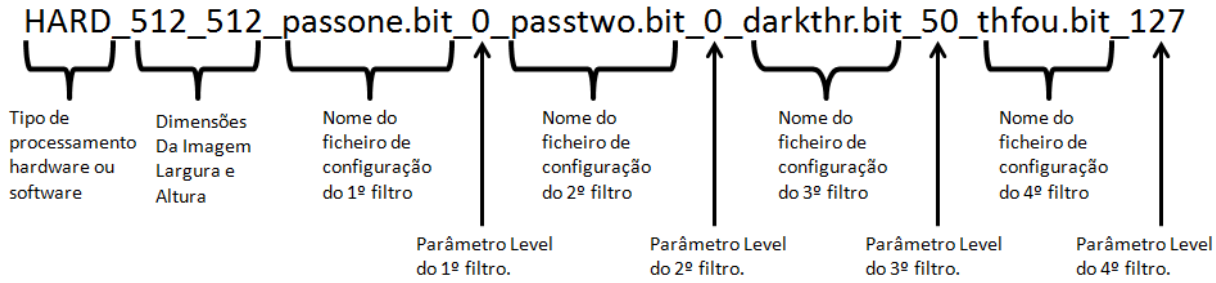

Figura 6.2: Trama de configuração do processamento.

Após a recepção da trama, o servidor fica à espera de receber a imagem a ser processada. Depois de receber e armazenar a imagem, é dada ordem para se efetuar o processamento. Nesta altura o cliente envia pedidos constantes ao servidor. Este, sempre que recebe um pedido, verifica se imagem já foi totalmente processada e envia uma de duas respostas: “WAIT” se a imagem ainda não tiver sido processada e “IMAGE” caso a imagem resultado esteja pronta. Isto significa que o processador está pronto para enviar a imagem após a receção do próximo pedido. Depois de enviada a imagem, o cliente, se a receber corretamente, faz um novo pedido. Desta vez pede as estatísticas de processamento, nomeadamente o tempo que demorou a processar a imagem.

Se ocorrer um erro em alguma das etapas de comunicação, o servidor reinicia todo este processo ficando novamente à espera da trama de configuração. Isto significa que o utilizador teria de enviar novamente uma imagem.

6.2 Processamento no Servidor

Para controlar o processamento da imagem e a reconfiguração dinâmica foram criadas duas estruturas: a estrutura “frameInfo” e a estrutura “filterParam”.

A estrutura "frameInfo"possuí variáveis que armazenam a largura e a altura da imagem, bem como o seu estado. A variável "optype"serve para indicar se o processamento requerido é em *software* ou em *hardware*. As variáveis "total_in"e "total_out"servem para medir a quantidade de imagem transferida para o módulo e quantidade de imagem processada já transferida de volta pelo DMA. A variável *received* funciona para indicar que foi recebida uma nova imagem e a variável "done"serve para indicar se a imagem recebida já foi processada ou não.

A estrutura "filterParam"serve para guardar os nomes dos ficheiros, que serão lidos do cartão de memória, para reconfigurar cada partição correspondente. O parâmetros *"level"* de cada filtro também são armazenados nesta estrutura.

A declaração das estruturas:

```
typedef struct frame_info frameInfo;
struct frame_info{
  int optype;
  int width;
  int height;
  int received;
  int total_in;
  int total_out;
  int done;
};
typedef struct filters_param filterParam;
struct filters_param{
  char filter1[6];
  int level1;
  char filter2[6];
  int level2;
  char filter3[6];
  int level3;
  char filter4[6];
  int level4;
};
```

Do lado do servidor, a comunicação é feita através de uma máquina de estados. Sempre que o cliente envia uma mensagem, o processador verifica o estado da comunicação e do processo e atua em concordância. Na figura 6.3 encontra-se representado o diagrama da máquina de estados responsável pela comunicação do lado do servidor.

No estado “DECIDE” é esperada a trama de configuração mencionada na secção 6.1. Desta trama são extraídas todas as informações necessárias para preencher as duas estruturas de controlo aqui já referidas. Para além disso, sempre que um dos nomes dos filtros é diferente daquele que já se encontrava na estrutura, é assinalado, num vector de *flags* de dimensão 4, que aquela partição, correspondente ao endereço da *flag* no vector necessita de ser reconfigurada. Este vector tem o nome de *"reconfig_need"*.

Se a trama recebida estiver correta, o próximo estado chama-se "RECV_IMAGE". Este estado espera receber os *bytes* da imagem que se deseja processar. Se a quantidade de *bytes* recebida for igual ao tamanho da imagem, então avança-se para o próximo estado.

No estado “WAIT_READY”, a imagem encontra-se em processamento. Contudo, o cliente faz pedidos constantes para verificar se a imagem processada já se encontra pronta para ser enviada. Quando isto acontece, o servidor responde afirmativamente ao cliente e este efetua um novo pedido, sendo que desta vez já "sabe"que vai receber a imagem como resposta.

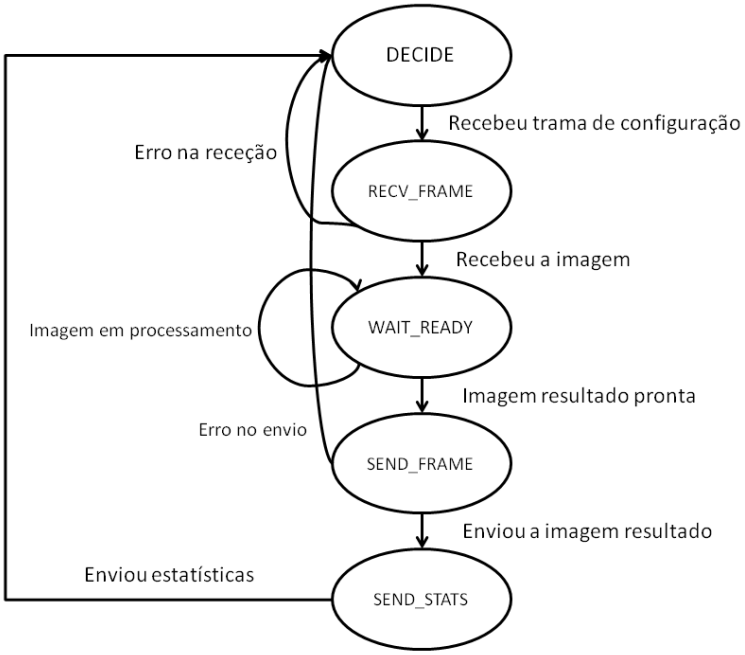

Figura 6.3: Diagrama da máquina de estados no servidor responsável da comunicação.

No estado “SEND_FRAME” a imagem processada é enviada para o cliente. Este processo ocorre de forma diferente da receção da imagem. Quando o cliente envia a imagem para o servidor, faz-lo numa só instrução uma vez que o seu *buffer* de saída é suficientemente grande para armazenar toda a imagem. Quando é o servidor a enviar a imagem de volta para o cliente, tem que o fazer fraccionando a imagem. Isto deve-se ao facto de o *buffer* de saída apenas conseguir armazenar 1460 *bytes*. Assim sendo, o cliente faz pedidos constantes e recebe como resposta porções da imagem. O tamanho destas porções pode variar conforme a o espaço disponível no *buffer* de saída do servidor. O controlo da transferência é efetuado em ambos os lados. Desta forma a transferência acaba quando se transferiu o número total de *bytes* que corresponde ao tamanho da imagem.

Depois de concluída a transferência da imagem resultado, o servidor encontra-se no último estado: "SEND_STATS". Neste estado o servidor espera pelo pedido do cliente, em que pede os tempos de processamento, e responde-lhe enviando as medidas do tempo que demorou o processamento da imagem.

6.3 Processamento da Imagem

A imagem pode ser processada em *software* ou em *hardware*. Se a opção do utilizador recair no *software*, a imagem é processada pelo próprio *MicroBlaze*. Se, pelo contrário, a opção for *hardware*, o processamento é executado pelo módulo "Processador de Imagem".

6.3.1 Filtros em *Software*

No processador criou-se uma biblioteca com os mesmos filtros que estão implementados em *hardware*. Desta forma é possível fazer uma comparação de desempenho justa.

Para efeitos comparativos e porque, em *hardware*, está implementada uma cadeia de quatro filtros, quando o processamento é efetuado em *software* são igualmente chamados quatro filtros, mesmo que sejam filtros "passa tudo", onde é apenas realizada uma cópia da imagem.

A imagem seguinte mostra a sequência de processamento da imagem por quatro filtros:

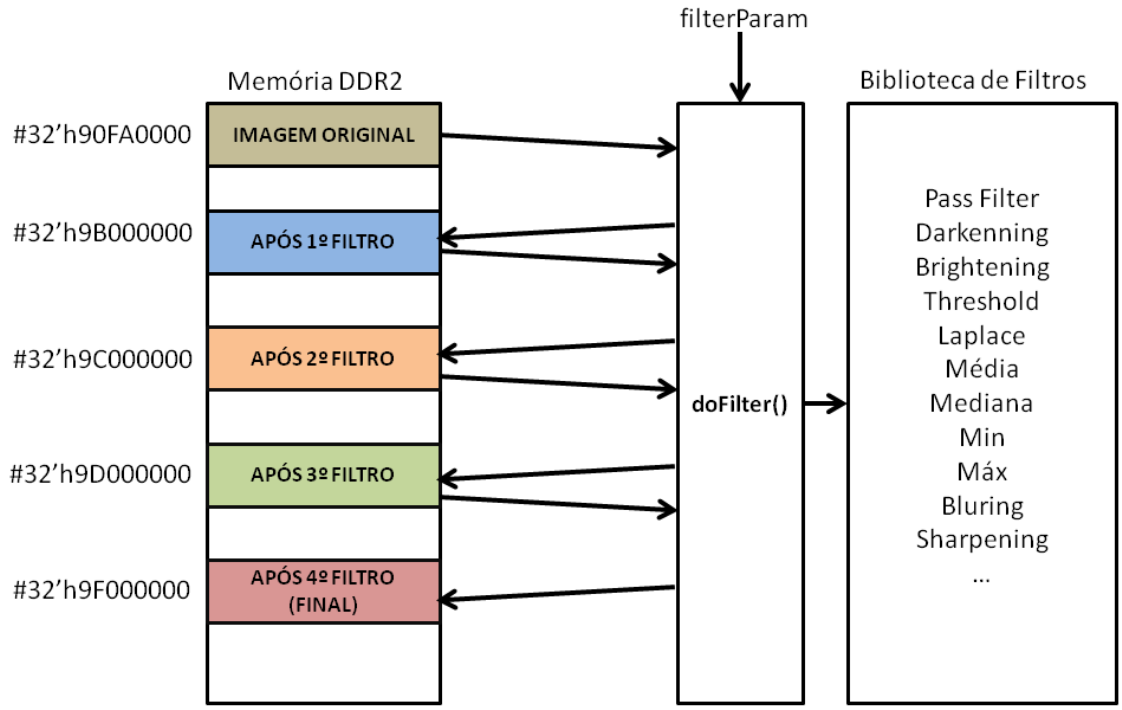

Figura 6.4: Sequência de Filtros em *software*.

A função "doFilter()"é chamada quatro vezes e decide qual dos filtros deve executar conforme os argumentos que lhe são passados. Esta recebe, para o filtro desejado, o nome do ficheiro de configuração, o parâmetro *level*, o endereço de origem e destino da imagem e as suas dimensões.

Por cada vez que a imagem é processada por um filtro diferente o resultado vai sendo colocado em endereços de memória superiores, até ser processado o último filtro.

O tempo que demora a realizar os quatro filtros é medido através de um contador. Este é disponibilizado pelo *core XPS TimeBase WDT* que se encontra conetado ao barramento PLB. O contador é incrementado à frequência que é usada pelo barramento. Antes de chamado o primeiro filtro, o contador é reiniciado e de seguida é guardada uma referência. Depois de processados todos os filtros é efetuada uma nova leitura do contador. A esta leitura é subtraída a referência e

obtém-se o tempo total despendido a processar a imagem em quatro filtros de *software*.

6.3.2 Controlo do Processamento em *Hardware*

O processamento da imagem, pelo módulo "Processador de Imagem"que foi desenvolvido, é realizado com a ajuda do DMA. Este é responsável por realizar a transferência das porções da imagem para serem processadas, e também responsável por recolher a informação resultante do processamento da cadeia de filtros.

O DMA pode ser configurado com vários modos de funcionamento, mas como neste caso são apenas realizadas transferências da memória para a FIFO de entrada e da FIFO de saída para a memória, são utilizados apenas dois tipos de funcionamento. Na transferência da memória para a FIFO do módulo, o endereço de origem é incrementado e o endereço de destino mantém-se fixo. Na transferência em sentido contrário, o endereço de origem mantém-se fixo e o endereço de destino é incrementado.

O controlo deste processo é efetuado por uma máquina de quatro estados. Na imagem seguinte é possível observar um diagrama de como é efetuada a transição entre os estados:

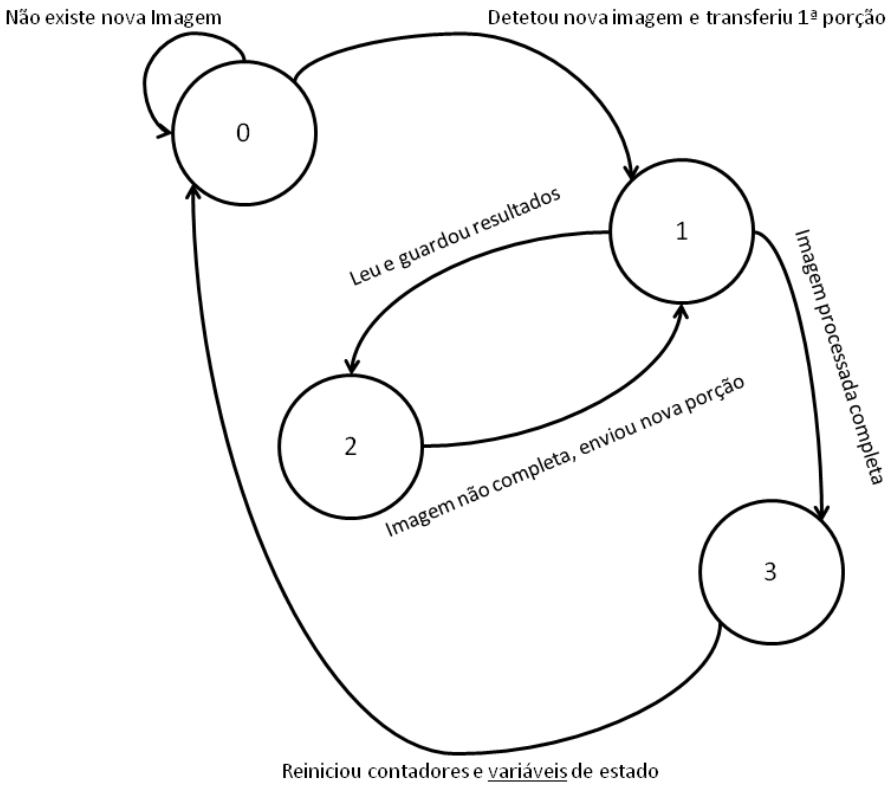

Figura 6.5: Controlo da transferência da imagem para o "Processador de Imagem".

O estado zero espera que seja recebida uma nova imagem. Quando isto acontece, percorrendo o vector *"reconfig_need"*, é verificado se existe alguma partição que necessite de ser reconfigurada. De seguida, são passados para o módulo todos os parâmetros de controle e processamento, como por exemplo, as dimensões da imagem. Por fim, ainda no estado zero, o DMA é configurado e é dada uma ordem de transferência da memória para a FIFO de entrada, de uma porção da imagem com 16384 *bytes*, que é o tamanho total da FIFO.

O estado um verifica se a transferência, efetuada no estado que lhe precede, já terminou. É então que é calculada a quantidade de informação que existe na FIFO de saída. Para isso, é efetuada uma leitura do registo que conta o número total de *bytes* que já foram colocados na fila e, utilizando a variável que conta os *bytes* já extraídos, encontra-se o restante na fila. É então que é configurado o DMA e dada uma ordem de transferência da quantidade calculada. Por fim, é verificado se a imagem já foi totalmente recebida. Se sim, avança para o estado três, caso contrário segue para o dois.

O estado número dois funciona de maneira semelhante ao um. Aqui é efetuado o calculo do espaço livre da FIFO de entrada e realizada uma transferência da quantidade calculada.

No estado número três é dada a indicação de que o processamento da imagem terminou e esta encontra-se pronta para ser enviada para o cliente. De seguida retorna para o estado inicial.

6.4 Interface Remota

A interface remota foi concebida com o objetivo de permitir ao utilizador uma fácil interação com o sistema reconfigurável implementado na placa de desenvolvimento. Assim, o utilizador tem a possibilidade de:

- Conetar-se ao servidor;
- Carregar uma imagem para ser processada;
- Escolher os filtros que quer utilizar e a sua ordem na cadeia de processamento;
- Visualizar a imagem resultado;
- Analisar o tempo necessário para o processamento da imagem.s

A figura 6.6 mostra a interface desenvolvida:

A sua utilização pode ser descrita em duas fases: preparação e execução. Ou seja, o utilizador coneta-se ao servidor e de seguida, não necessariamente por esta ordem, seleciona a imagem e os filtros que quererá utilizar. Esta é a fase de preparação. Quando estiver satisfeito com a sua seleção, carrega no botão *"run"* e inicia a execução. Esta ação desperta uma sequência de mensagens que realizam o protocolo já especificado na figura 6.1.

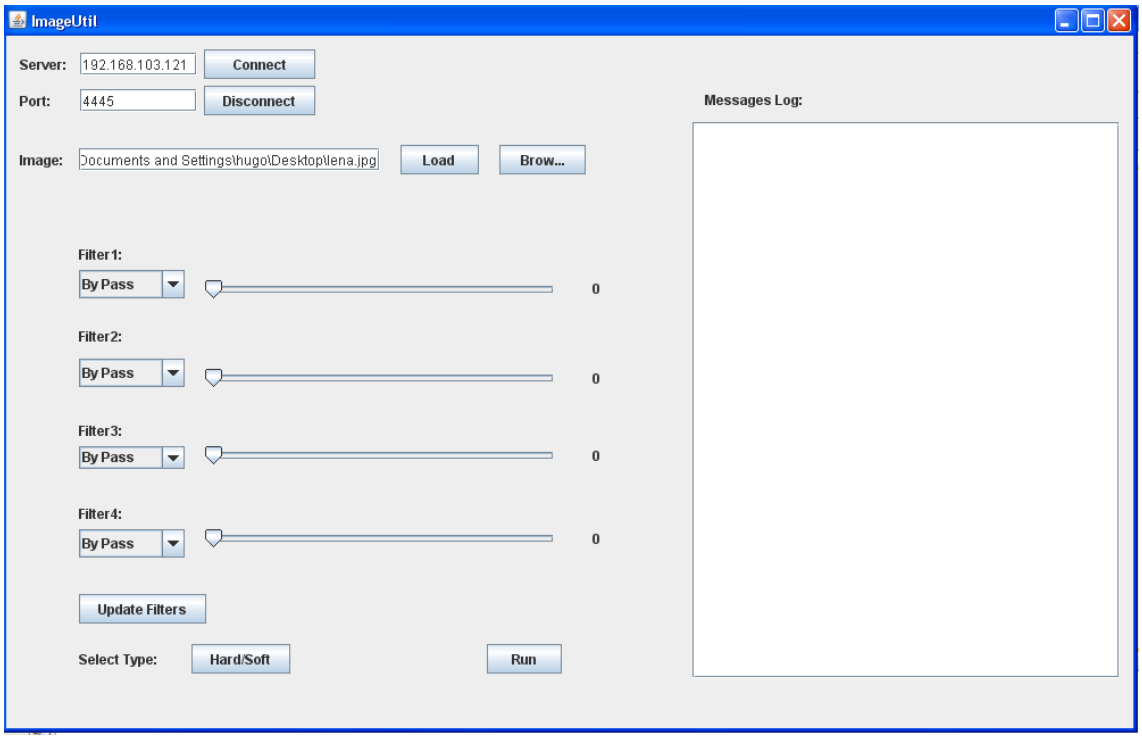

Figura 6.6: Interface gráfica remota.

A visualização das imagens é realizada através de *frames* individuais. Ou seja, quando o utilizador carrega no botão *load*, aparece uma nova frame com a imagem carregada. Quanto a imagem resultado é recebida, aparece uma outra *frame* onde se visualiza o resultado.

Uma vez que, através da adição ou remoção de ficheiros de configuração, podem ser adicionados ou removidos filtros ao sistema, os filtros existentes para configuração estão especificados num ficheiro de texto que é lido sempre que a interface é arrancada ou quando se carrega no botão *"Update Filters"*.

```
1 ...
  1#By Pass#passone.bit
3 1#Darkening#darkone.bit
  2#By Pass#passtwo.bit
5 2#Threshold#thretwo.bit
  ...
```

Listing 6.1: Excerto do ficheiro com a lista dos filtros.

A sintaxe consiste em três partes: a posição do filtro (1 a 4), o nome do filtro que aparece para escolha e o nome do ficheiro de configuração que se encontra no cartão de memória. Cada parte encontra-se separada pelo símbolo de cardinal.

A caixa de texto presente na interface, mostra mensagens ao longo de todo o processo. nomeadamente a conexão, o envio e recepção da imagem, a seleção dos filtros e as estatísticas de processamento.

Capítulo 7

Validação e Verificação

O sistema pode ser decomposto em três camadas principais: interface remota, *software* implementado no processador embutido na placa e o módulo em *hardware* desenvolvido para processar a imagem. Após a fase de testes e ajustes, verificou-se que as três camadas encontram-se funcionais como um todo. Ou seja, está criado o suporte para que o utilizador possa realizar experiências com reconfiguração dinâmica.

Todas as funcionalidades implementadas encontram-se a funcionar sem que tenha sido detetada qualquer falha. Isto significa que:

- A imagem submetida pelo utilizador é enviada e processada com sucesso;
- Da lista de filtros que se encontra implementada, os módulos são reconfigurados com sucesso mediante a escolha do utilizador;
- Todos os filtros implementados em *software* encontram-se igualmente funcionais. Desta forma o utilizador pode optar, para efeitos de comparação, por realizar a mesma operação em *hardware* ou *software*.
- No final do processamento o utilizador recebe uma mensagem com o tempo de despendido em processamento, podendo tirar as suas ilações sobre a vantagem da utilização do *hardware*.

7.1 Tempos de Processamento

Foram realizados testes em *hardware*, *MicroBlaze* e Matlab com duas imagens de tamanhos diferentes passando na mesma cadeia de processamento. Os tempos de processamento obtidos estão descritos nas tabelas 7.1 e 7.3. O processamento em Matlab foi realizado num computador com 2.2Ghz *dual-core*.

Hardware		***MicroBlaze***	**MATLAB**
Tempo Total	Tempo Efetivo	Tempo Total	Tempo Total
32 734 424 ns(0.032 73 s)	2 129 976 ns(0.002 13 s)	1.6746×10^{10} ns (16.7 s)	0.0143 s

Tabela 7.1: Imagem com tamanho 512 x 512

Hardware		***MicroBlaze***	**MATLAB**
Tempo Total	Tempo Efetivo	Tempo Total	Tempo Total
78 285 064 ns(0.078 28 s)	4 956 056 ns(0.004 96 s)	2.8838×10^{10} ns (28.8 s)	0.0629 s

Tabela 7.2: Imagem com tamanho 875 x 700

Hardware		***MicroBlaze***	**MATLAB**
Tempo Total	Tempo Efetivo	Tempo Total	Tempo Total
127 964 880 ns(0.1279 s)	8 454 200 ns(0.008 45 s)	4.32328×10^{10} ns (43.3 s)	0.1986 s

Tabela 7.3: Imagem com tamanho 875 x 700

No processamento em *hardware* o tempo total refere-se ao tempo despendido utilizando o DMA e considerando as paragens do módulo enquanto este espera por mais informação para processar. O tempo efetivo, não considera as paragens do sistema, ou seja, traduz o tempo que foi realmente despendido, pelo *hardware*, a processar a imagem.

Os tempos de processamento obtidos provam que mesmo utilizando o DMA, o desempenho do processamento em *hardware* encontra-se na mesma ordem de grandeza que o processamento em Matlab, que corre num computador com frequência mais elevada. Considerando o tempo efetivo de processamento em *hardware*, verifica-se que este é, aproximadamente, 5 vezes mais rápido do que o processamento em Matlab. Este facto que explicita a vantagem da utilização de *hardware* para acelerar o processamento.

Todos os filtros implementados gastam o mesmo tempo. As únicas diferenças que se podem notar será conforme o número de andares da arquitetura *pipepline* usada nos cálculos de operações locais, que podem acrescentar ou retirar alguns ciclos de relógio ao processamento.

Outra vantagem em relação ao processamento em *software* no *MicroBlaze* é que aumentando o número de filtros da cadeia, o tempo de processamento é aproximadamente o mesmo. Isto deve-se ao facto de toda a arquitetura de suporte para o varrimento da imagem se encontrar *pipelined* também. Já em Matlab ou no *MicroBlaze*, acrescentar filtros à cadeia de processamento aumenta significativamente o tempo de processamento, uma vez que a imagem tem que ser processada desde o início mais uma vez.

O tempo de reconfiguração de uma partição, com a área descrita na secção 5.3, ronda os 0,7

segundos, mesmo contando com o tempo de reconfiguração, a utilização do *hardware* compensava, comparativamente com o processamento no *MicroBlaze*.

O valor de 0,7 segundos obtido para o tempo dispendido na reconfiguração de uma partição, foi medido pelo processador e está diretamente associado à frequência de 50Mhz a que funciona o módulo ICAP.

A cadeia de filtros utilizada para as duas imagens foi: filtro "laplace", filtro de passagem, filtro de *threshold* e escurecimento. Na figura 7.1 é possível observar o resultado deste processamento.

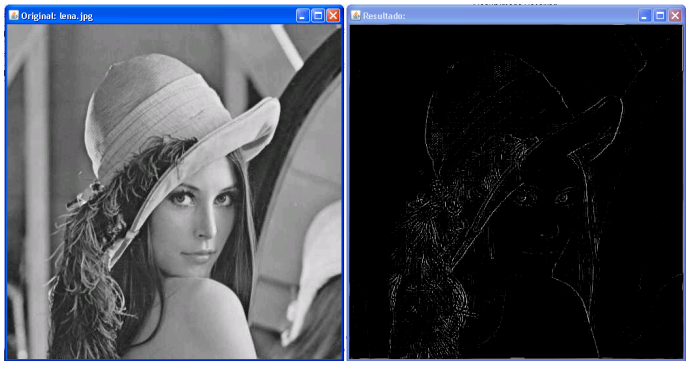

Figura 7.1: Imagem original e imagem resultado após o processamento.

7.2 Sumário de Implementação

Após a implementação do sistema desenvolvido, com filtros de passagem em todas as partições da cadeia, estas são as estatísticas de utilização da FPGA:

Recurso:	Utilização:	Disponível:	(%)
Registos (*Flip-Flops* e Trincos)	9590	28800	33
LUTs	9387	28800	32
Slices	4575	7200	63
IOBs	232	480	48
BlockRAMs/FIFOs	40	60	66
Memória Total Usada	1368KB	2160KB	63
BUFG/BUFGCTRLs	9	32	28
IDELAYCTRLs	3	16	18
BSCANs	1	4	25
BUFIOs	8	56	14
DCM_ADVs	1	12	8
DSP48Es	3	48	6
ICAPs	1	2	50
PLL_ADVs	1	6	16

Tabela 7.4: Sumário da implementação do Sistema em FPGA.

Estes resultados mostram que existia ainda bastante espaço na FPGA para expandir o sistema a mais funcionalidades. Por exemplo, era ainda possível acrescentar algumas memórias FIFOs e novas partições para filtros reconfiguráveis na cadeia de processamento.

Capítulo 8

Conclusão e Possíveis Modificações

Depois de implementado e testado o sistema, é possível concluir que este encontra-se funcional e cumpre com os objetivos inicialmente propostos. Contudo, durante o desenvolvimento do projeto, foram também pensadas abordagens diferentes e melhorias que poderiam ter sido implementadas. Neste capítulo são discutidas e descritas possíveis alterações e melhorias do sistema.

8.1 Resumo do Trabalho Realizado

Durante o período dedicado à realização do meu projeto de dissertação, foi seguida uma sequência de etapas que culminam com a implementação do "Demonstrador Remoto de Reconfiguração Dinâmica de FPGAs".

Inicialmente, começou-se por montar um sistema base escolhendo e conetando vários *IPcores* através do XPS. Depois desta fase, iniciou-se o desenvolvimento do meu próprio *core* chamado de "Processador de Imagem". Inicialmente projetou-se a cadeia de processamento com apenas duas partições para filtros reconfiguráveis e mais tarde expandiu-se para quatro. Depois de testado, adicionou-se este módulo ao sistema base, ficando este completo.

Prosseguiu-se para o desenvolvimento dos vários filtros e geração dos respetivos *bitstreams* parciais para utilizar na permuta de configurações de processamento consoante a vontade do utilizador.

A etapa seguinte foi o desenvolvimento de *software* necessário para o *microblaze*. Aqui desenvolveu-se a comunicação entre a interface remota e a placa de desenvolvimento, desenvolveu-se o controlo da transferência da imagem para o módulo de processamento e desenvolveu-se também uma biblioteca de filtros para utilizar como comparação de performance.

Por fim, desenvolveu-se a interface gráfica e realizou-se uma integração vertical de todo o sistema.

8.1.1 Operações Locais de Janela Maior

O aumento da janela para processar as imagens traduz-se na inserção de novas memórias FIFO à entrada de cada filtro, aumentando assim o número de linhas que entram em cada filtro.

Tendo em conta que no sistema atual, cada memória FIFO que precede um filtro contém 2048 *bytes* de comprimento, a inserção de uma estrutura semelhante antes de cada andar de processamento gastaria 8192 (2048×4) *bytes* em *brams*.

Da tabela 7.4 é possível verificar que, após a implementação do sistema, ficam disponíveis 99 *KBytes* em *brams*. Assim é possível concluir que a janela de processamento poderia ser expandida até um tamanho de 8x8 ($99K/8K$).

A inserção de novas memórias FIFO apenas iria aumentar a altura da janela. Para aumentar a largura bastaria aumentar, no interior de cada filtro, os *shift-registers* que armazenam os *bytes* de cada linha.

Por cada memória FIFO que se acrescentasse, teria de ser introduzida nova lógica de suporte e controlo adicional. Isto não seria problema, uma vez que os resultados de síntese, na tabela 7.4, mostram que ainda existe margem para esse acréscimo.

8.1.2 Aumento da Cadeia de Processamento

A inserção de um novo filtro na cadeia de processamento, traduzir-se-ia em acrescentar uma nova partição reconfigurável. Na verdade, não existem evidências que indiciem que o acréscimo de novas partições reconfiguráveis não seja viável. No entanto é necessário ter em conta que isto terá várias implicações que podem prejudicar a implementação e o funcionamento do sistema. Nomeadamente:

- As interligações/recursos de encaminhamento da FPGA, ficarão mais congestionada e portanto mais suscetível a que sejam utilizado caminhos longos, na fase de colocação e encaminhamento da implementação, fazendo com que se verifiquem violações temporais. Para solucionar as hipotéticas violações seria necessário baixar a frequência do sistema e, portanto, o desempenho seria também afetado.

- No caso de se querer adicionar vários andares à cadeia de processamento no seguimento do ponto anterior, poderá ser necessário que se diminua a área de cada uma das partições. Isto fará com que os filtros tenham que ser menos complexos.

- Ao aumentar o número de andares de processamento, seria necessário introduzir nova lógica para controlar cada novo andar adicionado.

8.1.3 Expansão do Sistema a Processamento de Vídeo

A ideia de utilizar vídeo, para efeitos demonstrativos da reconfiguração dinâmica, é muito apelativa e foi considerada durante o desenvolvimento do projeto. No entanto, visto o projeto ser um demonstrador remoto, o *bottleneck* de toda a operação encontra-se na receção e envio da imagem pela rede.

Considerando a utilização do DMA, o tempo total de processamento para uma imagem de 512 x 512 é de aproximadamente 0,03 segundos o que daria, sem considerar o tempo de envio e

recepção, para realizar o processamento com uma cadência de perto de 33 imagens por segundo.

O que realmente inviabiliza a realização de processamento de vídeo, neste projeto, é o tempo dispendido no envio e recepção da imagem. No envio são necessário cerca de 0,7 segundos e para receção cerca de 1,5 segundos. Estes tempos foram medidos pela aplicação remota e são referentes a imagens de tamanho 512 x 512. A disparidade entre o tempo de envio e recepção é justificada pela diferença de *buffers* de envio e recepção que existem do lado do cliente e do servidor. Neste caso, os *buffers* do cliente são maiores e, portanto, consegue-se enviar a imagem mais rapidamente.

Como se pode concluir, só para enviar e receber a imagem são necessário 2 segundos, ou seja, mesmo diminuindo o tamanho da imagem para um quarto da referida, não seria possível atingir sequer a cadência de processamento de 5 imagens por segundo.

Referências

[1] Pao-Ann Hsiung, Marco D. Santambrogio, e Chun-Hsian Huang. *Reconfigurable System Design and Verification*. CRC Press, 1 edição, Fevereiro 2009.

[2] Pierre Leray, Amor Nafkha, e Christophe Moy. Implementation Scenario for Teaching Partial Reconfiguration of FPGA. Em *Proceedings of 6th International Workshop on Reconfigurable Communication Centric Systems-on-Chip (ReCoSoC)*, Montpellier, France, Junho 2011. 6 pages.

[3] Katherine Compton e Scott Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, Junho 2002.

[4] Inc. Xilinx. *Virtex-5 Family Overview*. Inc. Xilinx,, Outubro 2011.

[5] 1-CORE Technologies. FPGA logic cells comparison. http://www.1-core.com/library/digital/fpga-logic-cells/.

[6] Wu Feng-yan Wang Lie. Dynamic partial reconfiguration in FPGAs. 2009.

[7] Inc. Xilinx. *Partial Reconfiguration Tutorial*. Inc. Xilinx,, Maio 2012.

[8] Inc. Xilinx. *XST User Guidel*. Inc. Xilinx,, Maio 2008.

[9] Inc. Xilinx. *PLanahead Software Tutorial, Design Analysis And Floorplanning for Performance*. Inc. Xilinx,, Setembro 2010.

[10] Arijit Bishnu Pritha Banerjee, Susmita Sur-Kolay. Floorplanning in modern FPGAs. 2007.

[11] Inc. Xilinx. *Constraints Guide*. Inc. Xilinx,, Dezembro 2009.

[12] Jürgen Becker Michael Hübner. Tutorial on macro design for dynamic and partially reconfigurable systems. 2009.

[13] Björn Grimm Jürgen Becker Michael Ullmann, Michael Hübner. An FPGA run-time system for dynamical on-demand reconfiguration. 2004.

[14] Inc. Xilinx. *LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a)*. Inc. Xilinx,, Setembro 2010.

[15] Inc. Xilinx. *LogiCORE IP XPS Central DMA Controller (v2.03a)*. Inc. Xilinx,, Dezembro 2010.

[16] Inc. Xilinx. *LogiCORE IP XPS HWICAP (v5.01a)*. Xilinx, Inc., Julho 2011.