UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

IMPLEMENTAÇÃO DE TÉCNICAS DE PROCESSAMENTO DE IMAGENS NO DOMÍNIO ESPACIAL EM SISTEMAS RECONFIGURÁVEIS

JONES YUDI MORI ALVES DA SILVA

ORIENTADOR: CARLOS HUMBERTO LLANOS QUINTERO

DISSERTAÇÃO DE MESTRADO EM SISTEMAS MECATRÔNICOS

PUBLICAÇÃO: ENM.DM - 31/10
BRASÍLIA/DF: JANEIRO – 2010

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

IMPLEMENTAÇÃO DE TÉCNICAS DE PROCESSAMENTO DE IMAGENS NO DOMÍNIO ESPACIAL EM SISTEMAS RECONFIGURÁVEIS

JONES YUDI MORI ALVES DA SILVA

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISÍTOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM SISTEMAS MECATRÔNICOS.

APROVADA POR:

_____

Profº Carlos Humberto Llanos Quintero, PhD (ENM-UnB) (Orientador)

_____

Profº. Guilherme Caribé de Carvalho, PhD (ENM-UnB) (Examinador Interno)

_____

Profº. Pedro de Azevedo Berger, PhD (CIC-UnB) (Examinador Externo)

BRASÍLIA/DF, 27 DE JANEIRO DE 2010

FICHA CATALOGRÁFICA

DA SILVA, JONES YUDI MORI ALVES

Implementação de Técnicas de Processamento de Imagens no Domínio Espacial em Sistemas Reconfiguráveis [Distrito Federal] 2010.

xvii, 127p., 210 x 297 mm (ENM/FT/UnB, Mestre, Sistemas Mecatrônicos, 2010).

Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.

Departamento de Engenharia Mecânica.

1.Processamento de Imagens 2.Hardware Reconfigurável

3.FPGA 4.Sistemas Embarcados

I. ENM/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

MORI, J.Y. (2010). Implementação de Técnicas de Processamento de Imagens no Domínio Espacial em Sistemas Reconfiguráveis. Dissertação de Mestrado em Sistemas Mecatrônicos, Publicação ENM.DM-31/10, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 127p.

CESSÃO DE DIREITOS

AUTOR: Jones Yudi Mori Alves da Silva.

TÍTULO: Implementação de Técnicas de Processamento de Imagens no Domínio Espacial em Sistemas Reconfiguráveis.

GRAU: Mestre ANO: 2010

_____

Jones Yudi Mori Alves da Silva
SMPW Quadra 24 Conjunto 03 Lote 05 Casa A
Park Way
71745-403 Brasília – DF – Brasil.

DEDICATÓRIA

Este trabalho é dedicado à minha família, cujo incentivo ao estudo e à busca pelo conhecimento, tornou possível a realização de mais uma etapa de minha vida.

Também o dedico aos tantos como eu, muitos melhores do que eu, que não tiveram a mesma oportunidade que eu tive.

AGRADECIMENTOS

Agradeço primeiramente à minha família, pelo incentivo constante e pelos conselhos.
Ao Professor Carlos Humberto Llanos, amigo e orientador, pela dedicação, sabedoria e amizade com que sempre dirigiu este trabalho. E também à sua esposa Carminha, pela paciência das reuniões feitas até nos fins-de-semana.
Ao Professor Pedro Berger, pelas discussões que ajudaram a nortear o caos de minhas idéias.
Ao Professor Guilherme Caribé, pelo grande interesse em meu trabalho, e pelas conversas que expandiram a visão sobre a aplicabilidade deste projeto.
Aos demais professores que contribuíram com minha formação, seja em disciplinas, seja em discussões sobre os projetos: Prof. José Maurício Motta, Prof. Edson Paulo, Prof. Walter Brito, Prof. Sadek Alfaro, Prof. Ricardo Lopes de Queiroz, Prof. Maurício Ayala.

À CAPES e à Universidade de Brasília, pelo suporte financeiro com bolsa e estrutura física.

Ao Marrocos, que continua abrindo as portas para seus alunos.

Aos amigos de Colômbia, Peru e Argentina, que são tantos que corro o risco de esquecer alguém: Daniel Muñoz, Yesid, Patiño, Dife, Diego, Panita, Lili, Jorge Minda, Diana, Cláudia, Ana Maria, Edgard, Éber, Fernand, Ronald, Rodrigo, Janier, Gerardo, Marú, Jorge Cormane, entre tantos outros...

Um agradecimento especial aos amigos da SEIT - Soluções em Engenharia e Inovações Tecnológicas, na figura de seus diretores-engenheiros: Engº Ênio Henrique, Engº Ênio Prates e Engº João Marcelo.

Aos grandes amigos do Brasil que conheci durante este período: Rosi, Arthur, Jorge Maia, Marcão, Rodrigo Queiroz, Thiago Siqueira, Luciano Franco, Daniel Bebiano, Filipe Bode, Luizão Toledo, Anderson, Evandro, Léo Júnior...

A todos que de alguma forma contribuíram para que este trabalho pudesse ser realizado.

EPÍGRAFE

"O JEITO é uma condição rara que se caracteriza por uma intuição extrema sobre todas as coisas mecânicas e elétricas, com outras disfunções sociais... ele vai ser ENGENHEIRO..."

Dilbert, de Scott Adams.

RESUMO

Cada vez mais o mercado exige aplicações de processamento de imagens e vídeos com restrições de tempo real. Novos produtos são lançados quase que diariamente, levando a integração de sistemas a patamares inimagináveis até poucos anos atrás. Dispositivos móveis lidam com aplicativos que exigem um poder de processamento cada vez maior. Nas indústrias, sistemas de visão computacional necessitam extrair a maior quantidade de informações de uma imagem, no menor intervalo de tempo possível, fazendo com que a demanda por processamento seja cada vez maior. O tempo de desenvolvimento de novas arquiteturas é caro e demorado, por vezes não sendo suficiente para atender às novas demandas em um prazo razoável. Paralelamente a isso, as arquiteturas comuns de processamento por vezes não são capazes de processar todas as informações necessárias nos intervalos de tempo desejados. Por esse motivo, novos sistemas processadores vêm sendo desenvolvidos na tentativa de explorar o poder de processamento previsto nas pesquisas sobre computação paralela. Já existem dispositivos móveis com processadores de mais de dois núcleos disponíveis nas lojas, a preços razoavelmente acessíveis. Com o intuito de buscar uma alternativa de projeto que permita um rápido desenvolvimento dos sistemas, com facilidade de testes e baixo custo, este trabalho propõe o estudo dos algoritmos mais comuns de processamento de imagens e a identificação de estruturas que permitam a descrição direta em arquiteturas de hardware desses algoritmos. A metodologia seguida buscou particionar os algoritmos em suas estruturas mais simples, permitindo a identificação dos tipos de paralelismo presentes e a proposicão de arquiteturas que exploram essas diferentes formas de paralelismo em arquiteturas sistólicas simples. Como resultado foram propostas e implementadas arquiteturas diversas para algumas das operações mais comuns de processamento de imagens. Um sistema completo de captura, processamento e visualização de imagens foi implementado, oferecendo uma plataforma de hardware reconfigurável extremamente flexível, permitindo o desenvolvimento e testes de novos algoritmos e arquiteturas.

ABSTRACT

Increasingly, the market requires applications of image processing and video with real time constraints. New products are launched almost daily, leading systems integration to unimaginable levels just few years ago. Mobile devices handle applications that require more and more processing power. In industry, machine vision systems need to extract the greatest amount of information of an image in the shortest possible time, causing the grow of processing demand. The development time of new architectures is expensive and time consuming, and sometimes the time-to-market is not enough to meet the new demands in a reasonable time. Parallel, the common processing architectures are often not able to process all the necessary information within the time allowed. Therefore, new systems processors have been developed in an attempt to exploit the processing power of parallel computing, making possible mobile processors with more than two cores available at a price reasonably low. In order to seek an alternative design that allows rapid development of systems, ease of testing and low cost, this work proposes the analysis of most common image processing algorithms and the identification of structures that allow for architectures design for these algorithms. The approach partitioned the algorithms in their simpler structures, allowing the identification of the types of parallelism present and propose architectures that exploit these different forms of parallelism in simple systolic architectures. As a result it have been proposed and implemented various architectures for some of the most common image processing algorithms in space domains, such as filters based on convolution, correlation, rank order filters, segmentation and binary morphology. A complete system for capturing, processing and displaying images has been implemented, providing a reconfigurable hardware platform extremely flexible, allowing the development and testing of new algorithms and architectures.

SUMÁRIO

LISTA DE FIGURAS

Figura 3.7 – O último passo: centralizar a máscara sobre outro pixel da imagem original, repetindo os passos anteriores. 37
Figura 3.8 – Imagens (a) sem ruído, (b) com ruído impulsivo e (c) com ruído branco. 39
Figura 3.9 – Máscara de média 3x3 e o resultado de sua aplicação sobre a Figura 3.8(b) 39
Figura 3.10 – Máscara Gaussiana 5x5, média 0 e desvio-padrão 1, e a correspondente imagem filtrada. 40
Figura 3.11 – máscaras para realce por derivação em *x* (vertical) e *y* (horizontal) 40
Figura 3.7 – O último passo: centralizar a máscara sobre outro pixel da imagem original, repetindo os passos anteriores. 37
Figura 3.8 – Imagens (a) sem ruído, (b) com ruído impulsivo e (c) com ruído branco. 39
Figura 3.9 – Máscara de média 3x3 e o resultado de sua aplicação sobre a Figura 3.8(b) 39
Figura 3.10 – Máscara Gaussiana 5x5, média 0 e desvio-padrão 1, e a correspondente imagem filtrada. 40
Figura 3.11 – máscaras para realce por derivação em *x* (vertical) e *y* (horizontal) 40

LISTA DE SIGLAS E ABREVIATURAS

ADC	Analog-Digital Converter
AHDL	Altera Hardware Description Language
ASM	Auto-Sequencing Memory
CPLD	Complex Programmable Logic Device
DLP	Data-Level Parallelism
DSP	Digital Signal Processor
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
GPGPU	General Purpose Graphics Processing Unit
GPP	General Purpose Processor
GPU	General Purpose Unit
HD	High-Definition
ILP	Instruction-Level Parallelism
LCD	Liquid Crystal Display
MAC	Multiply And Accumulate
MIMD	Multiple-Input, Multiple-Data
MISD	Multiple-Input, Single-Data
NASA	National Aero-Space Agency
RTL	Register Transfer Level
SIMD	Single-Instruction, Multiple-Data
SISD	Single-Instruction, Single-Data
VHDL	VHSIC-HDL Very-High Scale of Integration Circuit - Hardware Description Language
VLSI	Very Large Scale of Integration

INTRODUÇÃO

Um dos grandes desafios tecnológicos de nossa época está no desenvolvimento de sistemas que consigam reproduzir, ou ao menos aproximar, as capacidades humanas de sentir e, principalmente, interpretar o mundo (Pedrini,2008). A visão do homem pode não ser a melhor dentre os animais (na verdade estamos em um patamar inferior com relação a muitas outras espécies), mas a nossa capacidade de interpretação de informações visuais é bastante vasta. A velocidade com que nosso cérebro é capaz de processar imagens, detectando padrões e extraindo informações é algo extraordinário e extremamente difícil de alcançar com os modelos e tecnologias atuais.

A análise de informações visuais é uma vasta área, já muito explorada, mas com muito a ser descoberto e desenvolvido. O estudo de técnicas para automação ou semi-automação do processo de análise dessas informações requer não apenas a compreensão do funcionamento do nosso aparelho visual natural, mas também a utilização dos avançados recursos tecnológicos que temos hoje à disposição.

O mercado de aplicações de imagens e vídeos, conhecidas conjuntamente com as aplicações de som como multimídia, é um dos setores que têm apresentado maior crescimento nos últimos anos, exigindo o rápido desenvolvimento de mais e melhores soluções tecnológicas para atender às demandas de versatilidade, dinamismo, velocidade e qualidade dessas aplicações (Hongtu,2007). O desenvolvimento de produtos para esse mercado requer grande atenção às melhorias quanto ao desempenho dos sistemas, medido de diversas maneiras, tais como resolução, largura de banda, robustez, consumo de energia e convergência digital (integração de diferentes utilidades em um mesmo equipamento, como os novos aparelhos de telefonia celular, que incluem câmeras, áudio, GPS, etc.). Com todas essas implicações, novas metodologias de projeto e arquiteturas de hardware são constantemente estudadas, principalmente para o desenvolvimento de aplicações de processamento em tempo real (Kehtarnavaz&Gamadia,2006).

O estudo de sistemas embarcados é um campo que requer muita atenção nesse atual momento tecnológico. Cada vez mais produtos e tecnologias vêm sendo desenvolvidos

com o intuito de agregarem-se maiores capacidades de processamento e inteligência, bem como robustez, flexibilidade e baixo consumo de energia, a sistemas de pequeno porte.

Sistemas embarcados (*Embedded Systems*) possuem requisitos de hardware e software diferenciados em relação aos sistemas comuns. Geralmente, são utilizados em aplicações com restrições de espaço, peso, energia, etc., devendo, portanto, ter muitas de suas funções otimizadas de modo a garantir seu funcionamento de forma confiável, sob as mais diversas situações.

São muito comuns hoje equipamentos com requisitos de processamento em tempo real, ou seja, aplicações com tempo de resposta extremamente pequenos, onde os *deadlines* de tarefas são os principais requisitos de desempenho da aplicação.

Processar imagens em tempo real é uma tarefa custosa do ponto de vista computacional, visto que mesmo imagens de tamanho pequeno são submetidas a diversas operações para extração de informações como: filtragens, transformações de cores, binarização, detecção de bordas, etc. Essas operações requerem uma grande velocidade de processamento, que a maioria dos sistemas comuns não consegue suprir, ocasionando falhas e atrasos nas aplicações.

A arquitetura comum do hardware de um sistema computacional prevê a ocorrência de atrasos oriundos da presença de gargalos no fluxo de dados e de instruções. Tais atrasos são agravados quando a quantidade de informações é muito elevada, como nos sistemas de processamento de imagens. As câmeras comuns (filmadoras e fotográficas) adquirem imagens estaticamente ou em vídeos, com taxa de 30 fps (*frames per second* – quadros por segundo). Porém, não é difícil encontrarem-se aplicações não-industriais com câmeras que adquirem mais de 150 fps. Quando essas imagens podem ser armazenadas para análise e tratamento posteriores, sistemas mais simples equipados com hardware adicional para captura e gravação em alta velocidade suprem as necessidades da aplicação. Porém, em sistemas embarcados de tempo real (com as restrições de memória, tamanho, peso e velocidade), a presença dos gargalos pode levar ao colapso.

Um processador de uso geral (*GPP – General Purpose Processor*) contém ALUs (*Arithmetic Logical Units*) com tamanho de palavras fixos (8, 16, 32, 64 bits) e tem padrões de controle e fluxo de dados pré-determinados. Como o próprio nome diz, um

processador de uso geral deve atender a uma ampla gama de algoritmos, tendo de ser genérico, fato esse que dificulta a otimização dos algoritmos.

As arquiteturas mais usuais (sejam de GPPs ou DSPs) baseiam-se em dois modelos básicos: a Arquitetura de Princeton, mais conhecida como Arquitetura de Von Neumann (GPPs) e a Arquitetura Harvard (DSPs e microcontroladores). As diferenças principais entre as duas arquiteturas são nas unidades aritméticas e nas formas de acesso à memória (Patterson&Hennessy,2005).

O modelo de Von Neumann prevê a existência de um único espaço de memória, acessado por um único barramento (Figura 1.1a). Para a execução de uma operação, o processador deve acessar a memória para buscar uma instrução e para buscar os dados necessários à execução dessa instrução, não sendo considerada uma boa escolha quando a aplicação necessita de uma grande quantidade de acessos à memória, como no caso do processamento digital de sinais (imagens e vídeos são casos particulares de sinais digitais) (Eyre&Bier,1998).

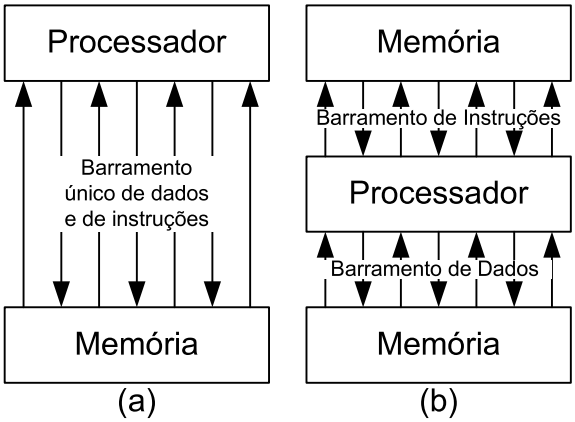


Figura 1.1 – Modelos básicos de arquiteturas: (a) Von Neumann; (b) Harvard. (Eyre&Bier,1998)

Por exemplo, para um filtro FIR (*Finite Impulse Response*), o processador deve completar uma operação MAC (*Multiply and Accumulate*) e fazer diversos acessos à memória em apenas um ciclo de instrução (Figura 1.2). Especificamente, o processador deve perfazer as seguintes tarefas:

- Buscar a instrução MAC;
- Ler o valor da amostra apropriado;

- Ler o valor do coeficiente apropriado;
- Executar o deslocamento dos valores amostrados.

O processador deve, então, executar um total de quatro acessos à memória em apenas um ciclo de instrução.

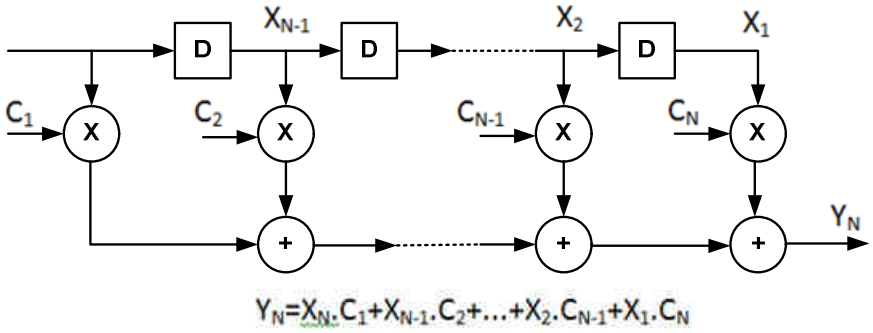


Figura 1.2 – Filtro FIR típico.

Em uma arquitetura de memória tipo Von Neumann, quatro acessos à memória consumiriam no mínimo quatro ciclos de instrução. Embora DSPs possuam *hardware* aritmético necessário para perfazer operações MAC em um ciclo de relógio apenas, eles não seriam capazes de executar uma operação completa por ciclo de relógio com uma hierarquia de memória estruturada como na arquitetura de Von Neumann. Por essa razão, no lugar da arquitetura de Von Neumann, os DSPs utilizam arquiteturas baseadas no modelo Harvard (Figura 2b) (Eyre&Bier,1998), (Patterson&Hennessy,2005).

Em uma arquitetura do tipo Harvard, existem dois espaços de memória separados: memória de dados e memória de programa. O processador conecta-se a essas memórias por dois barramentos independentes, permitindo dois acessos simultâneos, dobrando a largura de banda da memória, permitindo a busca de instruções e dados paralelamente (Eyre&Bier,1998), (Patterson&Hennessy,2005).

Diversas alternativas podem ser estudadas e aplicadas para a solução dos problemas observados nas arquiteturas de Von Neumann e de Harvard. Uma delas, foco deste trabalho, é a definição de arquiteturas de processamento de imagens e vídeos de alta velocidade e flexibilidade, utilizando para isso algoritmos adaptados especialmente para implementação em hardware reconfigurável. A utilização de FPGA's (*Field Programmable Gate Arrays*) oferece a flexibilidade necessária ao desenvolvimento do sistema proposto, sendo uma plataforma já bastante estudada para aplicações de alto

desempenho. Um ponto importante é que as FPGAs podem implementar de maneira natural o paralelismo inerente aos algoritmos utilizados para processamento de imagem e visão computacional (Gokhale et.al.,2005). Um ASIC (*Application Specific Integrated Circuit*) poderia ser utilizado, porém longo tempo de desenvolvimento e o elevado custo só viabilizariam projetos de larga escala de produção.

Em uma FPGA, o funcionamento do circuito pode ser reconfigurado de modo que arquiteturas e funcionalidades completamente diferentes podem ser implementadas no mesmo dispositivo. Existem dois tipos básicos de FPGAs: os de granularidade fina (*fine-grain*) executam operações de apenas um bit de largura, definindo circuitos no nível de portas lógicas [Bra05]. Desse modo, tem-se uma grande flexibilidade de projeto, porém o trabalho de reconfiguração do dispositivo é bastante demorado. Já dispositivos de granularidade grossa (*coarse-grain*), representados principalmente pelas rDPAs (*reconfigurable data-path arrays* - arranjos de via de dados reconfiguráveis) já operam em nível de transferência entre registradores (RTL - *Register Transfer Level*), tendo sido muito utilizados para a implementação de aceleradores de hardware para diversas aplicações (Becker&Hartenstein,2003). As arquiteturas reconfiguráveis em geral têm sido vistas como ferramentas poderosas para duas grandes aplicações: bioinformática e supercomputação. Na área de bioinformática, o principal desafio está na otimização de algoritmos para processamento de dados biológicos, e.g. comparação de seqüências de DNA (Jacobi et.al, 2005). Na área de supercomputação, o desenvolvimento de metodologias para a utilização das FPGAs como aceleradores configuráveis de acordo com a aplicação é um dos grandes desafios.

A execução direta de algoritmos de hardware em FPGA oferece aumentos de velocidade tipicamente entre 10 e 100 vezes, em comparação com o mesmo algoritmo em software. Isso tem atraído muitas pesquisas na área de Processamento Digital de Sinais. Os FPGAs oferecem ainda outras vantagens em relação aos microprocessadores genéricos e aos DSPs (*Digital Signal Processors)*, tendo em conta a sua flexibilidade, em aplicações de alto desempenho e pequeno volume, e ainda mais particularmente em aplicações que possam explorar larguras de bits específicas e com alto grau de paralelismo de instruções (Gokhale et.al.,2005).

A aceleração no desempenho pela implementação dos algoritmos em FPGAs vem do fato de que, como o hardware é programável, pode ser configurado especificamente para um

algoritmo, aproveitando o paralelismo intrínseco do mesmo. A configuração do hardware pode perfazer uma aritmética de precisão arbitrária, com os tipos de dados, número de unidades aritméticas e interconexões entre os blocos de processamento definidos unicamente pelo algoritmo. Tal flexibilidade permite a otimização da implementação, de modo a melhorar ainda mais o processamento. Adicionalmente, a implementação do algoritmo diretamente em hardware permite aliviar as limitações do modelo de Von Neumann, especificamente: (a) execução seqüencial de instruções e (b) limitação do barramento de comunicação entre memória e processador. Em um algoritmo mapeado em hardware não existe execução de instruções (e, portanto, não existe "contador de programa", que na verdade é um "contador de instruções"). Neste sentido, os dados são introduzidos no hardware, sendo processados através de estruturas do tipo *pipeline*, para gerar os resultados desejados. No mapeamento de um algoritmo em hardware as estruturas de laços de repetição, tais como (*for*, *repeat*, *while*, etc.) são mapeadas em *pipeline*, controladas por seqüência de eventos e contadores de dados (Hartenstein,2006).

Estruturas de processamento baseadas em arranjos sistólicos (Figura 1.3) permitem eliminar totalmente as limitações do paradigma de Von Neumann. Neste caso, os dados entram na estrutura computacional (em hardware), onde somente é feito o controle do acesso dos dados ao sistema. Assim, o sistema computacional possui, no lugar de um "contador de instruções" um "contador de dados". O modelo de memória utilizado é conhecido como ASM (*Auto-Sequencing Memory*), ou Memória Auto-Seqüencial, em que a memória utiliza o contador de dados da arquitetura para incrementar o endereço de leitura de dados.

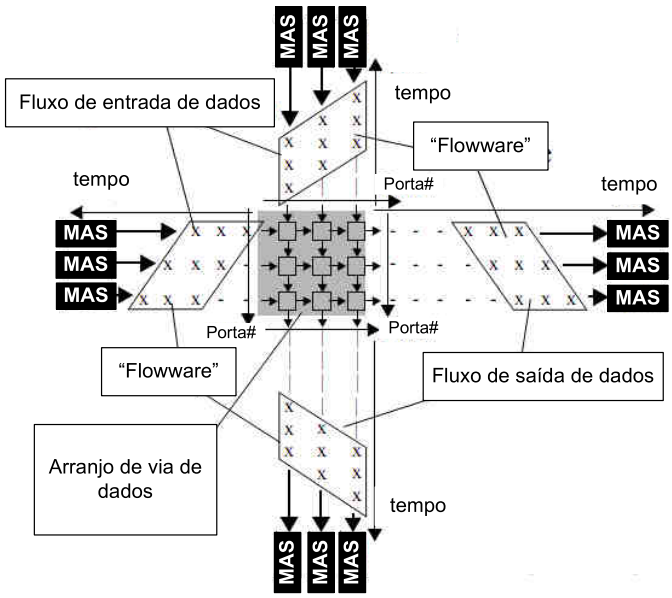


Figura 1.3 – Modelo de Arranjo Sistólico (Hartenstein,2006).

A implementação em lógica reconfigurável pode eliminar a necessidade de utilizarem-se instruções, bastando o acesso à memória de dados. A arquitetura mostrada na Figura 1.4 exemplifica uma possível arquitetura de um filtro FIR.

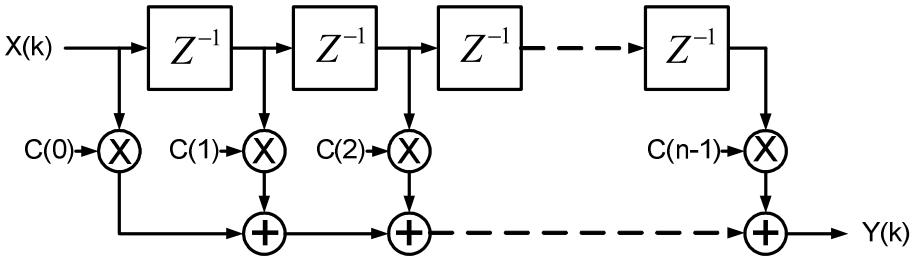


Figura 1.4 – Implementação simples de um filtro FIR.

Comparando-se as Figuras 1.2 e 1.4, percebemos uma grande semelhança entre as duas, o que nos mostra o alto grau de especialização da implementação em hardware do filtro FIR.

Poder-se-ia facilmente, em um dispositivo de lógica programável, como uma FPGA, melhorar a implementação mostrada na Figura 1.4, construindo-se uma arquitetura com estrutura de *pipeline*, como pode ser visto na Figura 1.5.

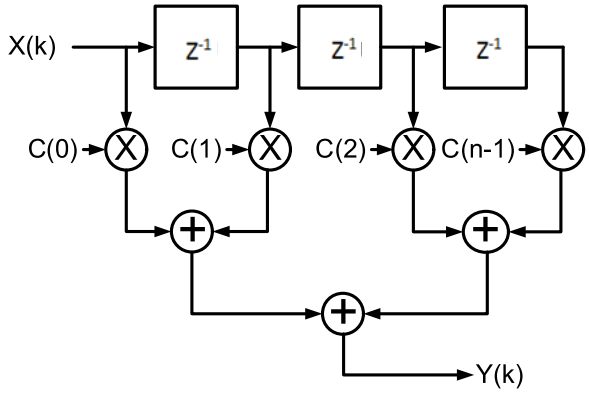


Figura 1.5 – Implementação do Filtro FIR com estrutura em *Pipeline*.

Uma arquitetura desse tipo eliminaria a busca por instruções juntamente com a busca por dados, eliminando o chamado Gargalo de Von Neumann (estricção na seqüência de execução de uma instrução, devido à necessidade de acessos à memória de dados e de programa).

A proposta deste trabalho é o estudo dos algoritmos computacionais mais comuns de processamento de imagens em software, para adaptá-los e utilizá-los como base para o desenvolvimento de outros a serem implementados em hardware, visando solucionar alguns dos problemas de desempenho de sistemas de processamento de imagens embarcados com requisitos de tempo real.

1. Metodologia do Trabalho

Para o desenvolvimento deste trabalho, inicialmente foram estudadas algumas das principais técnicas e algoritmos de processamento de imagens e vídeos, de modo a identificarem-se os pontos mais críticos da cadeia de processamento, no quesito de custo computacional.

Após essa identificação, os algoritmos escolhidos tiveram sua estrutura de funcionamento básico estudada em detalhes e, a partir daí, foram analisadas diversas arquiteturas para a aceleração do processamento, com base, principalmente, nas técnicas de paralelismo e *pipelining*. Além da aceleração do processamento, houve ainda a preocupação com a identificação e eliminação de redundâncias de cálculos, de modo a economizarem-se recursos de hardware, diminuindo a utilização de multiplicadores embarcados (*embedded multipliers*) e elementos lógicos (*logical elements*). Adicionalmente, se teve em conta o desempenho das técnicas implementadas em hardware.

As arquiteturas definidas foram separadas com base em uma mescla das divisões clássicas de arquiteturas paralelas de acordo com o tipo de paralelismo e fluxo de dados e instruções apresentados, e nos conceitos de computação reconfigurável.

2. Objetivos

2.1 Objetivos gerais

Em geral, a base deste trabalho encontra-se na utilização de uma metodologia para a transformação de técnicas e algoritmos de processamento de imagens no ***domínio espacial*** em estruturas que permitam identificar formas de melhorar o desempenho do processamento, em termos de velocidade e área.

2.2 Objetivos específicos

O objetivo primário deste trabalho é a implementação de um sistema de captura, processamento e visualização de imagens, embarcado em *hardware* reconfigurável. Um objetivo secundário é o estudo para identificação e eliminação de redundâncias no processamento, para economia de multiplicadores.

3. Resultados alcançados

Com a utilização da metodologia citada, foi possível analisar e propor melhorias nos algoritmos, pela implementação de arquiteturas distintas que exploram os conceitos de paralelismo, *pipelining* e eliminação de redundâncias. A abordagem clássica da computação paralela em conjunto com as bases da computação reconfigurável mostraram-se bastante úteis para a organização do método seguido, sendo possível dividir as técnicas de processamento de imagens em grupos, de modo a facilitar a exploração dos conceitos citados.

Uma plataforma completa de captura, processamento e visualização de imagens foi implementada, permitindo a validação das arquiteturas propostas e viabilizando a continuidade deste trabalho em projetos subseqüentes.

4. Organização do trabalho

No Capítulo 2 é feita uma revisão acerca do estado da arte das diferentes áreas envolvidas neste trabalho. Inicialmente é feita uma breve introdução ao Processamento de Imagens e Vídeos, mostrando seus objetivos, aplicações mais comuns. Após isso são mostradas as principais formas de implementação dos algoritmos de processamento de imagens e vídeos, mostrando as arquiteturas mais utilizadas de sistemas de hardware e software da área. Finalmente é feita uma abordagem sobre sistemas reconfiguráveis, identificando suas principais vantagens e desvantagens em relação às outras tecnologias, e mostrando qual a linha de raciocínio a ser seguida ao longo do restante do trabalho.

No capítulo 3 são feitas algumas análises que permitem delinear uma metodologia a ser utilizada para a identificação de gargalos, redundâncias e outros problemas no fluxo de processamento de imagens e vídeos, de modo a facilitar a busca de soluções para esses

problemas. São explicados os algoritmos de processamento de imagens selecionados para análise, tecendo considerações acerca de suas características estruturais, e indicando soluções para melhoria dos critérios de desempenho das implementações desses algoritmos. As vantagens e desvantagens de cada solução são discutidas e utilizadas para a proposição de arquiteturas que permitam melhorar o processamento com relação a uma ou mais métricas.

No capítulo 4 é explicada a plataforma de hardware utilizada na implementação do sistema proposto. Algumas características de interesse para a definição das arquiteturas são realçadas e detalhes específicos do FPGA utilizado são discutidos rapidamente.

No capítulo 5, as implementações das arquiteturas propostas em uma linguagem de descrição de hardware de alto nível são mostradas. Detalhes de configuração e explicações do código-fonte são feitas, mostrando os passos de cada implementação.

No capítulo 6 são mostrados os resultados de implementação das arquiteturas na plataforma de hardware utilizada. Exemplos comuns de seqüências de algoritmos em processamento de imagens foram implementados juntamente com os blocos de captura e visualização de imagens, mostrando a funcionalidade do sistema completo.

No capítulo 7 são feitos comentários acerca dos resultados obtidos, bem como sobre a metodologia utilizada para o desenvolvimento das soluções apresentadas. Algumas propostas de trabalhos futuros são oferecidas, indicando as possibilidades de melhorias e aplicações dos métodos dispostos neste projeto.

2 – TRATAMENTO DE IMAGENS

Iniciamos este capítulo com uma pequena introdução à área de Processamento de Imagens e Vídeos, mostrando algumas de suas aplicações. Em seguida, destacamos alguns dos principais problemas de implementação de sistemas que trabalham com imagens, dando maior ênfase para a grande massa de dados a processar. As implementações mais comuns são mostradas, juntamente com suas arquiteturas de hardware e software. Por fim, é apresentada uma reflexão sobre sistemas reconfiguráveis e suas vantagens e desvantagens em relação às demais tecnologias.

2.1 POR QUE TRATAR IMAGENS?

A utilização de técnicas de processamento de imagens na resolução de problemas, por meio da análise de informações visuais, já envolve inúmeras áreas de aplicação, principalmente devido ao grande avanço tecnológico na área de computação ao longo das últimas três décadas. Dentre as áreas de aplicação mais comuns, podemos citar: a área militar, automação industrial, medicina, sensoriamento remoto, segurança e multimídia (Pedrini,2008).

Na área militar, são encontradas aplicações como rastreamento de alvos móveis, auxílio à navegação de veículos não-tripulados e identificação de cenas e alvos em imagens aéreas.

Na área de automação industrial são muito comuns equipamentos de inspeção visual de produtos, rótulos e embalagens, com o intuito de efetuar o descarte de produtos com não-conformidades. Outra utilização é na localização de objetos para manipulação por robôs e na monitoração de processos de fabricação em tempo real, para realimentação de controladores.

Na medicina, os sistemas de auxílio a diagnósticos permitem aos médicos uma maior segurança quando da análise de imagens oriundas de tomógrafos, aparelhos de raios X, ultra-sonografia e ressonância magnética. Atualmente não só técnicas mais simples como ajustes de contraste e brilho são utilizadas, mas também equipamentos mais sofisticados

que oferecem, com base na imagem e em um banco de dados, listagens de possíveis males do paciente.

No campo de Ciência e Engenharia de Materiais, imagens de microscópios são muito utilizadas em análises metalográficas e cristalográficas, permitindo a identificação de estruturas e padrões de formação em materiais, auxiliando na melhoria de processos de fabricação e na busca de novas aplicações e materiais.

O sensoriamento remoto permite a análise e interpretação não só de imagens no espectro visual, mas também imagens oriundas de diversos tipos de sensores, como infravermelhos e magnéticos, permitindo a identificação de zonas de mineração, correntes marítimas, focos de incêndios, desmatamento e crescimento urbano, permitindo um melhor planejamento por parte do governo e empresas, no sentido de aproveitar os recursos naturais e planejar estradas, rodovias e zoneamentos territoriais.

Sistemas de controle de acesso já utilizam desde a leitura de digitais até a identificação de padrões na íris, retina ou ainda o reconhecimento de faces. O reconhecimento óptico de caracteres pode identificar automaticamente uma assinatura, permitindo a autenticação de cheques ou ainda a conversão de imagens em textos.

Já no campo do entretenimento, novas aplicações surgem a todo momento. Videogames extremamente realistas utilizam tecnologias muito avançadas de computação gráfica, bem como algoritmos otimizados, principalmente de renderização de imagens. Câmeras fotográficas já são capazes de efetuar um disparo automático pela detecção de um sorriso e as altas resoluções dos sensores comuns em aparelhos celulares exigem que existam algoritmos de compressão e descompressão de imagens e vídeos embarcados em dispositivos cada vez menores. Houve ainda, no início do século XXI, a consolidação dos sistemas de televisão digital, com recursos avançados, como interatividade e alta definição de imagens. A Internet já é utilizada como meio de transmissão de vídeos até mesmo em tempo real, dependendo de aquisição e processamento velozes, bem como de algoritmos de compressão e descompressão de dados extremamente sofisticados para permitir maiores resoluções em canais de transmissão com largura de banda limitada.

2.2 – TIPOS DE PROCESSAMENTO DE IMAGENS E VÍDEOS

O processamento de imagens e vídeos em tempo real já é uma realidade nas indústrias há vários anos, apresentando recentemente uma grande expansão no mercado de aplicações multimídia. Para um bom entendimento das ferramentas necessárias para a implementação dos algoritmos, faz-se necessária uma introdução das bases de conhecimento da área.

Inicialmente, vamos conceituar os fundamentos desse campo do conhecimento. Analisando os tipos básicos de operações mais comuns nos algoritmos de processamento de imagens e vídeos, nota-se que a exploração dos diferentes tipos de paralelismo inerente a essas operações, pode levar ao atendimento de parte das necessidades do processamento em tempo real. A seguir serão discutidos alguns sistemas de processamento de imagens e vídeos, e em seguida será apresentado um breve histórico desses sistemas, com uma descrição acerca das decisões de projeto mais comuns nas implementações dessas aplicações.

2.2.1 – Paralelismo em operações de processamento de imagens e vídeos

Essencialmente, imagens e vídeos digitais são sinais multidimensionais com grande quantidade de dados, exigindo muitos recursos de processamento e de memória (Kehtarnavaz&Gamadia,2006). Por exemplo, seja uma típica imagem digital, com MxN *pixels* e P *bits* de precisão. Tal imagem possui MxNxP *bits* de dados, sendo que cada pixel pode ser suficientemente representado por 1 *byte*, ou 8 *bits*, à exceção de aplicações médicas e científicas, nas quais a utilização de 12 ou mais *bits* por *pixel* é necessária para um aumento da precisão dos resultados. No caso de imagens coloridas temos a triplicação da massa de dados, uma vez que o padrão mais comum é o RGB, com três canais de cores de mesma precisão. Caso o interesse da aplicação seja no processamento de vídeos, a taxa de aquisição de imagens deve ser levada em consideração, pois nesse caso, a velocidade de processamento deve ser tal que todas as imagens sejam processadas, sem perdas de frames. O processamento de uma imagem única é feito sobre as duas dimensões espaciais do plano da imagem, já no caso de um vídeo, muitas vezes é necessário atuar-se sobre os dados de imagens em seqüência, levando em consideração também a dimensão temporal.

Quando trabalhamos com o processamento de imagens e vídeos em tempo real, o problema de lidar-se com a grande quantidade de dados e cálculos torna-se uma grande preocupação.

Uma simples câmera de vídeo digital, capturando um vídeo colorido de resolução padrão VGA (640x480) a 30 frames por segundo, requer um processamento a uma taxa de 27 milhões de pixels por segundo. Casos mais críticos, como câmeras de alta definição (HD-*High Definition*), onde cerca de 83 milhões de pixels por segundo devem ser processados para uma resolução de 1280x720 pixels por quadro, já são comuns. E esse é um problema que só tende a crescer, pois o mercado clama por taxas de aquisição e resoluções cada vez maiores, exigindo que uma quantidade cada vez maior de dados seja processada em intervalos de tempo cada vez menores.

O conceito de processamento paralelo, algo familiar para os projetistas da área de arquitetura de computadores, é uma das chaves para lidar-se com problemas envolvendo quantidades massivas de dados (De Rose et.al., 2008). Muito do que se pode fazer na implementação de sistemas eficientes de processamento está centrado em quão bem a implementação, de hardware e software, explora as diferentes formas de paralelismo em um algoritmo: paralelismo em nível de dados (DLP - *Data Level Parallelism*) e paralelismo em nível de instrução (ILP - *Instruction Level Parallelism*). O paralelismo em nível de dados é encontrado quando se pode aplicar a mesma operação a diferentes conjuntos de dados. Já o paralelismo em nível de instrução encontra-se na aplicação de distintas operações independentes de forma simultânea (De Rose et.al., 2008).

Para ter-se uma melhor visualização dos conceitos de paralelismo e como se aplicam aos problemas analisados, é necessário tentar enxergar mais de perto os tipos de operações envolvidas no processamento de imagens e vídeos.

É comum a estratificação das operações de processamento de imagens e vídeos em três níveis: baixo, médio e alto, sendo que a diferença entre os níveis baseia-se na relação produzida entre os dados de entrada e de saída. As operações de baixo nível recebem uma imagem em sua entrada e produzem uma imagem em sua saída. Operações de nível médio recebem uma imagem na entrada e produzem atributos de imagem na saída. Já as operações de alto nível recebem os atributos em sua entrada e os interpretam, em geral para perfazer alguma ação ou tomar alguma decisão (Gonzalez&Woods,2000). A seguir será descrito melhor cada nível, mostrando os tipos de paralelismo mais presentes em cada um.

2.2.2 – Operações de Baixo Nível

As operações de baixo nível transformam imagens em imagens, ou seja, os operadores lidam diretamente com a matriz de dados em nível de pixels. Tais operações incluem as transformações de cores, ajustes de contraste, filtragem, redução de ruídos, realce de bordas, algumas transformações no domínio da freqüência, etc. Um dos principais objetivos dessas operações é o realce de características da imagem, preparando-a para operações subseqüentes no estágio de nível intermediário, como a simples visualização ou mesmo uma extração de características.

Há uma subdivisão entre as operações deste nível em três tipos: (a) pontuais, (b) de vizinhança (c) ou globais. As operações pontuais são as mais simples, e transformam um pixel de entrada em um pixel de saída, sendo que essas operações não dependem de valores dos pixels vizinhos. Operações aritméticas, lógicas e limiarização incluem-se nesta categoria. O paralelismo de dados aqui é muito óbvio, já que as operações pontuais são executadas para cada um dos pixels da imagem de entrada de modo independente (Figura 2.1).

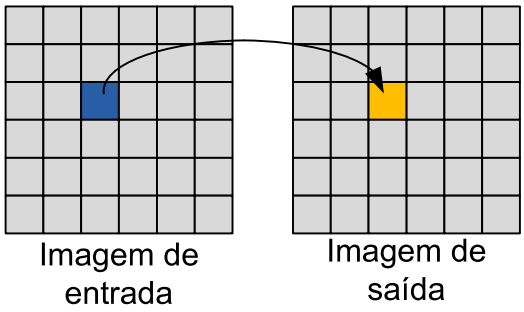


Figura 2.1 – Operação pontual: um pixel de entrada gera como resultado um pixel de saída (adaptada de (Kehtarnavaz&Gamadia,2006)).

As operações de vizinhança são mais complexas que as operações pontuais, utilizando, para cada pixel de saída, os dados dos vizinhos do correspondente pixel de entrada. A quantidade de cálculos, como pode ser visto, cresce de acordo com o tamanho da vizinhança. Algumas operações deste conjunto são: a convolução e correlação espaciais, a filtragem, a suavização, o realce de contornos, etc. O paralelismo de dados fica é evidenciado pela repetição das mesmas operações sobre todos os pixels da imagem de entrada (Figura 2.2).

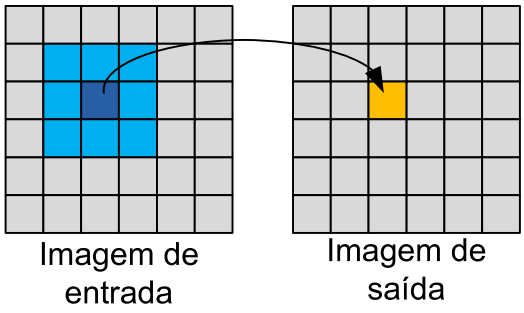


Figura 2.2 – Operação de vizinhança: um conjunto de pixels de entrada gera um pixel de saída (adaptada de (Kehtarnavaz&Gamadia,2006)).

No caso das operações globais, para a geração de um único pixel de saída, temos que trabalhar com todos os pixels de entrada. Um bom exemplo é a Transformada Rápida de Fourier. Tais operações lidam com quantidades de dados enormes de cada vez (Figura 2.3).

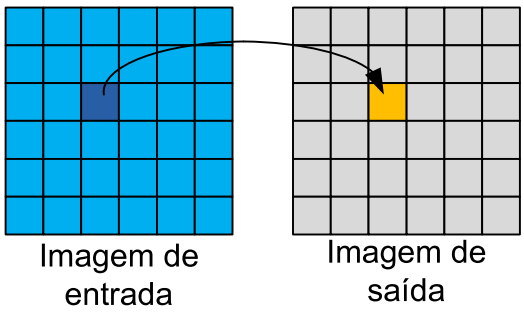


Figura 2.3 – Operação global: todos os pixels da imagem de entrada geram um pixel de saída (adaptada de (Kehtarnavaz&Gamadia,2006)).

Como descrito, as operações de baixo nível necessitam da execução repetida de um mesmo algoritmo sobre todos os pixels da imagem de entrada independentemente. Percebe-se que essas operações envolvem quantidades massivas de dados, com estruturas de processamento previamente conhecidas e acessos à memória intensos. Tais características tornam as operações de baixo nível fortes candidatas à exploração de paralelismo em nível de dados.

2.2.3 – Operações de nível intermediário

Nas operações de nível intermediário, imagens são traduzidas em formas mais abstratas de informação, retratando certos atributos ou características de interesse da imagem. As operações aqui continuam lidando com os dados no nível de pixel, mas suas saídas apresentam uma redução significativa na quantidade de dados em relação às entradas. A

segmentação da imagem em regiões, a extração de bordas, linhas e contornos, bem como outros atributos de interesse, como dados estatísticos fazem parte do rol de operações desta classe. A redução na quantidade de dados, ou a conversão de uma imagem em um conjunto de características que a representa, é o objetivo desta etapa do processamento. As estruturas de cálculo aqui também são bastante regulares, e necessitam de muitas repetições de operações independentes sobre a imagem de entrada, tornando-as fortes candidatas à exploração do paralelismo em nível de dados.

2.2.4 – Operações de alto nível

A função das operações de alto nível é a interpretação do conjunto abstrato de dados vindo dos níveis intermediários e a execução de uma análise do conteúdo de uma cena com base em conhecimento. Tais operações incluem o reconhecimento e classificação de objetos, ou ainda a tomada de decisões de controle. As estruturas de processamento nesta etapa são mais irregulares, e utilizam operações inerentemente seqüenciais, sobre uma massa menor de dados, em relação às outras etapas. Essas características aliam-se aos baixos requisitos de largura de banda de acesso à memória, tornando essas operações candidatas à exploração de paralelismo no nível de instruções.

2.2.5 – Diversidade de operações em processamento de imagens e vídeos

A grande variedade de operações de processamento de imagens e vídeos vai das operações mais regulares e com grande fluxo de dados, até as irregulares e com poucos dados no fim da cadeia. Uma cadeia típica de processamento combina os três níveis em um sistema completo, como mostrado na Figura 2.4, em que são representados a cadeia de processamento e a redução na quantidade de dados em cada etapa, para uma dada imagem de NxN *pixels* e P *bits* de precisão, exemplificando um reconhecimento facial. Na etapa de Pré-processamento, há uma filtragem para eliminação de ruídos, mantendo a mesma quantidade de dados da imagem original (N^2P bits). Na etapa de Segmentação, há uma redução da quantidade de dados, pela transformação da imagem filtrada em uma imagem binária, de apenas 1 bit por pixel (N^2 bits). Na etapa de Análise há uma redução maior na quantidade de dados, com a produção de um vetor de características representativas do contorno da imagem binarizada, resultando em K bits de dados. Por último, na etapa de Interpretação, é gerada a informação sobre o reconhecimento ou não da face, com o fornecimento na saída de apenas 1 bit.

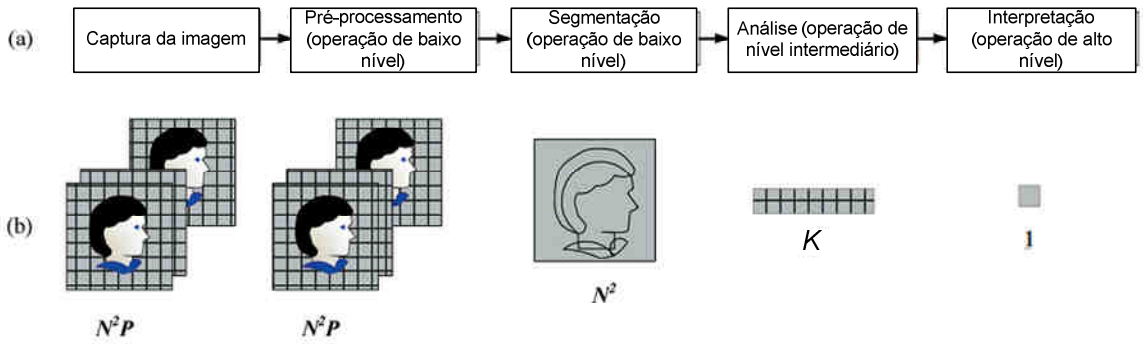


Figura 2.4 – Cadeia de processamento (a) e redução na quantidade de dados (b) (adaptada de [Keh08]).

2.3 – ARQUITETURAS DE PROCESSAMENTO DE IMAGENS E VÍDEOS

Agora será apresentado um breve histórico sobre as plataformas de hardware e software, culminando com o estado da arte em hardware de processamento de imagens e vídeos, consolidando assim a motivação para o desenvolvimento deste trabalho.

2.3.1 – Histórico das plataformas de hardware para processamento de imagens e vídeos

No fim da década de 1950 ocorreu nos Estados Unidos, mais precisamente em um laboratório do *NIST* (*National Institute of Standard and Technology*), um dos primeiros registros de processamento de imagem digital. Havia sido construído um equipamento digitalizador que guardava as imagens na memória de um computador que as processava. Nesses primeiros testes os experimentos envolviam filtros de realce de bordas.

A corrida espacial, principalmente nos anos 1960, impulsionou o desenvolvimento de sistemas mais avançados na NASA, para a obtenção de imagens mais claras para a exploração espacial.

Os campos de inspeção industrial e imageamento médico foram uns dos que mais demandaram o rápido desenvolvimento de aplicações de processamento de imagens. Já naquela época, devido ao paralelismo facilmente identificado nas operações de níveis baixo e médio, as arquiteturas específicas para processamento de imagens eram construídas com paralelismo massivo [Keh08]. Os primeiros computadores utilizados para

processamento digital de imagens eram grandes mainframes paralelos, porém, a busca pela miniaturização e os avanços nas tecnologias VLSI levaram ao desenvolvimento de soluções de processamento de alto desempenho, pequenas e com baixo consumo de energia, oferecendo a possibilidade de utilização de dispositivos de tamanho reduzido, mas com grande poder de processamento.

Era comum, quando as aplicações possuíam requisitos de processamento em tempo real, a utilização de múltiplas placas com diversos processadores trabalhando em paralelo, especialmente em aplicações médicas e militares, onde o custo geralmente não é um fator determinante. Mas quando do surgimento dos primeiros processadores programáveis de sinais digitais, esse pensamento começou a mudar. No fim da década de 1980 surgiram os primeiros DSPs comerciais, desenvolvidos especialmente para acelerar os algoritmos de processamento de sinais, o que forneceu as bases para a era dos sistemas computacionais embarcados.

Ainda nos anos 1980, houve o surgimento dos dispositivos lógicos programáveis, como os *CPLDs* e *FPGAs*, cujo objetivo era unir a flexibilidade do software com a velocidade do hardware dedicado dos *ASICs*. Nos anos 1990, o desempenho tanto dos *DSPs* quanto dos *FPGAs* cresceu bastante, de modo a atender à demanda dos dispositivos multimídia, surgindo o conceito de *SoC*, cuja premissa básica é o embutimento de todo o poder de processamento necessário em uma única pastilha [Jia07]

Um esforço mais recente da comunidade científica tem sido na adaptação do grande poder de processamento presente nas unidades de processamento gráfico (*GPUs – Graphics Processing Units*) encontradas nos *PCs* e *laptops* modernos para o processamento de imagens e vídeos. Unidades de co-processamento são mais utilizadas atualmente em sistemas não embarcados, mas já estão sendo desenvolvidas muitas pesquisas na área [Jia07].

2.3.2 – Plataformas de hardware para processamento de imagens e vídeos

Devido aos grandes avanços tecnológicos nas últimas décadas, tem-se hoje à disposição diversos tipos de plataformas de hardware distintas, com vantagens e desvantagens umas em relação às outras. Temos então, de analisar as plataformas disponíveis para implementação, de modo a efetuar a melhor escolha da arquitetura de hardware a ser utilizada. Em [So07] e (Kehtarnavaz&Gamadia,2006) pode ser encontrado um bom

resumo de alguns sistemas de processamento de imagens e vídeos. Os pontos de maior interesse para este trabalho são listados a seguir.

2.3.2.1 – ASICs – *Application Specific Integrated Circuits*

Os Circuitos Integrados de Aplicação Específica, devido ao seu alto grau de especialização, oferecem o melhor desempenho dentre os processadores para algoritmos de imagens e vídeos. Uma de suas principais desvantagens é o tempo de desenvolvimento do projeto, e a pouca flexibilidade de configurações pós-fabricação oferecida.

2.3.2.2 – GPPs – *General Purpose Processors*

Os Processadores de Propósito Geral englobam os processadores da linha Pentium, ARM e Spark, entre outros. São extremamente flexíveis na camada de software, porém sua utilização é geralmente feita em conjunto com um sistema operacional. Essa integração hardware/software já vem sendo desenvolvida há um tempo considerável, sendo considerada bastante otimizada. Os processadores encontrados nas residências e estações de trabalho mundo afora possuem um alto desempenho e arquiteturas fortemente paralelas, com características que permitem explorar o paralelismo de instruções em aplicações de processamento de imagens e vídeos (Kehtarnavaz&Gamadia,2006). Tais processadores, porém, são grandes consumidores de energia, não sendo adequados para sistemas embarcados.

Um pouco mais recentemente, surgiram as arquiteturas com mais de um núcleo de processamento (arquiteturas *multicore*), aumentando consideravelmente o desempenho desses processadores. Essas arquiteturas multicore surgiram principalmente devido às limitações de freqüência de operação enfrentadas pelos processadores dos PCs atuais. Porém, já existem sistemas embarcados com arquiteturas *multicore*, como a 3ª versão do IPhone (Apple Corporation), que possui a arquitetura mostrada na Figura 2.5.

A idéia por trás da utilização de um sistema multiprocessado em um aparelho multimídia como o IPhone, é de exploração de processamento paralelo, já que o mesmo dispositivo integra funções de navegação na Internet, telefonia móvel, câmera fotográfica e acessibilidade avançada (display LCD multi-toque e sensores internos de movimento, como acelerômetros e giroscópios).

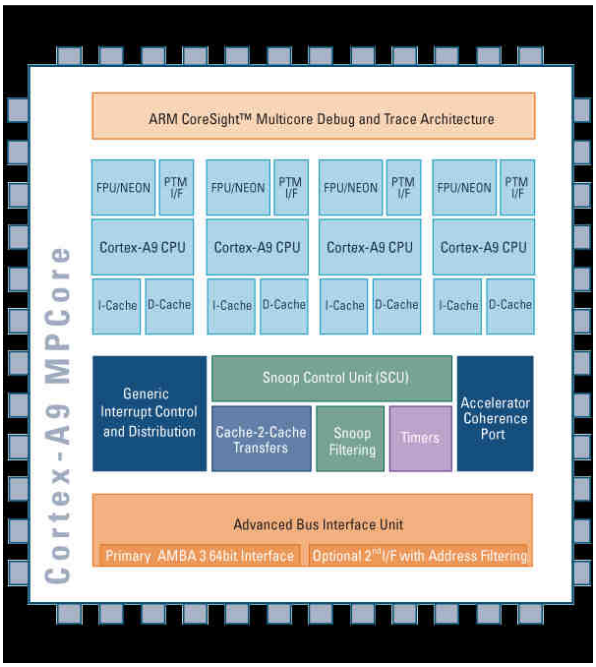


Figura 2.5 – Arquitetura multicore presente no novo IPhone da Apple Corporation.

2.3.2.3 – Processadores de Sinais Digitais (DSPs)

Os DSPs são processadores programáveis com características arquiteturais que os tornam vantajosos para o processamento de sinais. Muitos dos DSPs possuem arquitetura de acesso à memória do tipo Harvard, unidades MAC e registradores de deslocamento otimizados, além de instruções específicas de processamento de sinais em seu conjunto de instruções, permitindo uma compilação mais eficiente dos programas. Por permitirem programação através de linguagens de alto nível oferecem uma grande flexibilidade, facilitando o desenvolvimento das aplicações. Além disso, possuem tamanho reduzido e consumo relativamente baixo de energia, o que os coloca como os melhores candidatos quando o assunto é o desenvolvimento de um sistema embarcado. Os DSPs podem trabalhar tanto com aritmética de ponto fixo quanto de ponto flutuante, dependendo da precisão necessária à aplicação. Outra característica muito forte dos DSPs é a previsibilidade de seu processamento, garantindo um bom grau de segurança nas aplicações. Com todos esses pontos positivos, os DSPs são uma opção bastante viável em aplicações de processamento de imagens e vídeos. Mais recentemente têm sido incluídos como co-processadores em sistemas embarcados comuns, como telefones celulares, câmeras digitais, etc.

2.3.2.4 – Processadores de Imagens Digitais

Assim como os DSPs possuem características arquiteturais que os tornam mais apropriados para o processamento de sinais, em desenvolvimentos recentes surgiram processadores programáveis com unidades de hardware dedicadas a operações de processamento de imagens. A maioria desses blocos de processamento específico executa operações de baixo nível, como convolução e morfologia (Akita,2003).

Alguns integram conversores A/D a cada foto-sensor da matriz de aquisição de imagem e operam em todos os pixels de forma paralela. A dificuldade de fabricação desses sistemas é enorme, de modo que ainda não existem comercialmente e ainda são factíveis para resoluções baixas, como 64x64 pixels (Kagami,2002).

Outros processadores de imagens possuem buffers de linha que permitem o armazenamento temporário de partes da imagem para um processamento mais rápido, porém essa estrutura demanda um espaço grande no chip, de modo que os sistemas implementados com esses buffers apresentam restrições no tamanho das imagens (Murata,1998).

2.3.2.5 – GPUs – Graphics Processing Units

Nos anos 2000 chegou ao mercado um novo tipo de processador, dedicado a atender às demandas de renderização de gráficos tridimensionais em tempo real dos modernos vídeo-jogos. Um processador desse tipo atualmente ultrapassa a quantidade de 120 *GFLOPs* (*Giga Floating-Point Operations per Second* – Bilhões de operações de ponto-flutuante por segundo) de poder de processamento, enquanto um processador Intel Pentium IV de 3.0GHz executa apenas 12 *GFLOPs* (Kehtarnavaz&Gamadia,2006).

Grande parte do poder de processamento das GPUs está em sua arquitetura que explora paralelismo e pipelining de modo avançado. O rol de operações que esses processadores executam é restrito, de modo que pôde ser extremamente otimizado. Uma GPU tipicamente possui diversos processadores dedicados trabalhando em paralelo, cada um com até 128 bits de profundidade de cores (4 vezes mais que os usuais 32 bits) por pixel [Ke04]. Isso permite um grau de precisão na renderização muito alto, alcançando um realismo impressionante.

Essas unidades de processamento gráfico (*GPUs*) eram como *ASICs*, com poucos recursos de configuração. Porém com o tempo foi sendo incorporada maior flexibilidade e possibilidades de programação, atraindo a atenção da comunidade de computação de alto desempenho. Atualmente diversas empresas e centros de pesquisa de todo o mundo já possuem trabalhos em desenvolvimento com o intuito de utilizar as GPUs para outras aplicações que não a renderização 3D, um conceito conhecido como GPGPU (*General-Purpose processing on a Graphics Processing Unit*). As GPUs já são utilizadas em problemas de processamento de imagens e vídeos em tempo real, em aplicações complexas como a reconstrução de imagens médicas de ressonância magnética e ultra-som. Um dos problemas enfrentados pelas primeiras gerações de GPUs eram os barramentos utilizados para conexão aos PCs, já que o barramento utilizado, PCI (*Peripheral Component Interconect*), não era suficientemente rápido para atender às aplicações. O advento do barramento PCI Express está, ao menos temporariamente, suprindo às necessidades de conexão desses dispositivos. Outra questão importante é que as GPUs atuais foram desenvolvidas para apresentarem um alto desempenho, a despeito do consumo de energia, consumindo ainda mais que um GPP comum. Essa característica deixaria as GPUs de fora do mercado de aplicações embarcadas, porém já há um esforço de desenvolvimento de GPUs para sistemas embarcados.

2.3.2.6 – Sistemas de Lógica Reconfigurável

Os sistemas de lógica reconfigurável são geralmente arranjos de componentes lógicos, interligados por uma rede programável. Seu nascimento se deu com o intuito de unirem-se a flexibilidade dos processadores e o desempenho dos ASICs em um único dispositivo. Os principais dispositivos dessa família são os FPGAs e os CPLDs. Suas características construtivas e de programação as tornam extremamente flexíveis e adaptáveis às necessidades específicas de cada aplicação, permitindo a implementação dos algoritmos diretamente em hardware. Assim como os DSPs, estes dispositivos oferecem uma grande previsibilidade de execução, característica essencial em sistemas com requisitos de tempo real. Por serem tão flexíveis, os FPGAs podem ser programados para explorar de modo eficiente as diferentes formas de paralelismo presente nos algoritmos de processamento de imagens e vídeos. Assim sendo, pode-se ter a implementação de diferentes algoritmos de baixo, médio e alto níveis, permitindo que sistemas completos sejam embarcados em um único chip. Outra característica importante é a grande largura de banda de acesso à memória, e a flexibilidade de configuração desse acesso, permitindo um grau de

especialização muito grande, adequando cada tipo de operação às memórias disponíveis de modo otimizado. Sua principal desvantagem é o grande consumo de energia, se comparado aos DSPs, e o tempo de treinamento de projetistas da área. Porém, com o constante avanço tecnológico, tal desvantagem vem diminuindo cada vez mais, fazendo com que os FPGAs já despontem como fortes candidatos a quebrar a hegemonia dos DSPs em aplicações embarcadas, principalmente para processamento de imagens e vídeos (Gokhale et.al.,2005), (Porter,2001). Diversas ferramentas, como geradores de código, bibliotecas em linguagens de alto nível como C, softwares de simulação, ambientes gráficos de desenvolvimento e etc., vêm sendo desenvolvidas com o intuito de elevar-se o nível de abstração do desenvolvimento de sistemas com FPGAs, de modo a facilitar o treinamento de equipes de projetistas.

2.3.3 – Classificação de Arquiteturas Paralelas

Para se alcançarem melhores resultados no desenvolvimento e na implementação de arquiteturas específicas de processamento de imagens e vídeos, deve-se tomar proveito das características de cada algoritmo, identificando-se os principais problemas de desempenho e o que pode ser feito para melhorar esses pontos mais críticos. Sendo o foco deste trabalho a implementação de arquiteturas de hardware baseadas em *pipelining*, paralelismo e eliminação de redundâncias, deve-se encontrar uma maneira de transcrever os algoritmos de processamento de imagens em arquiteturas que possam ser implementadas e testadas. Isso implica em encontrar-se uma relação entre os diferentes tipos de paralelismo encontrados em processamento de imagens e as classificações mais comuns de arquiteturas de hardware. Primeiramente explicitar-se-á como são feitas normalmente as classificações das arquiteturas e então, no capítulo 3, formalizar a metodologia seguida para a transcrição dos algoritmos de processamento de imagens em arquiteturas de hardware.

A classificação de arquiteturas de hardware pode ser feita segundo diversos critérios, sendo a mais comum a Taxonomia de Flynn, em que os computadores são divididos em classes e o processo de computação resulta da interação entre um fluxo de instruções e um fluxo de dados (Júnior,2006), [Nav08]. Seguindo a proposta de Flynn, existem quatro classes de arquiteturas:

- SISD – *Single Instruction, Single Data* – possui um fluxo único de instruções e um fluxo único de dados. Um Elemento de Processamento (EP) executa uma seqüência de instruções sobre um conjunto de dados, sendo executada uma operação por vez.

Esse modelo corresponde à arquitetura clássica de Von Neumann e engloba os microprocessadores comuns das estações de trabalho.

- MISD – *Multiple Instruction, Single Data* – Trata-se de um modelo puramente teórico, não tendo sido registrada nenhuma implementação desta categoria. Consistiria na aplicação de várias instruções ao mesmo tempo sobre um único dado.

- SIMD - *Single Instruction Multiple Data* – Consiste na aplicação de um fluxo único de instruções seqüenciais aplicado sobre fluxos de dados distintos, de modo paralelo. Pode ser vista como várias máquinas SISD executando a mesma instrução sobre conjuntos de dados independentes, de forma paralela.

- MIMD – *Multiple Instruction, Multiple Data* – corresponde a várias instruções sendo aplicadas simultaneamente a múltiplos conjuntos de dados. A maioria dos computadores ditos paralelos se encaixa nessa categoria.

2.4 – HARDWARE PROGRAMÁVEL

A Computação Reconfigurável (RC – *Reconfigurable Computing*) é o uso de lógicas programáveis para acelerar a computação (Gokhale et.al.,2005). Esse conceito surgiu nos anos 1980, com o lançamento no mercado dos FPGAs. A principal novidade desse conceito era que os dispositivos podiam ser reprogramados quantas vezes fosse necessário e que diferentes algoritmos eram implementáveis diretamente em hardware, de modo semelhante às implementações de diferentes softwares com um processador convencional.

Como já comentado no Capítulo 1, a execução de algoritmos diretamente em hardware oferece algumas vantagens, como a aceleração na velocidade de processamento (geralmente entre 10 e 100 vezes), o alto grau de especialização que uma implementação pode atingir, e as otimizações que podem ser feitas devido à grande flexibilidade desses dispositivos.

Com vinte anos de mercado, tais dispositivos já são bem conhecidos dos projetistas de hardware, com ao menos meia-dúzia de fabricantes em escala mundial. Assim sendo, temos uma grande quantidade de dispositivos disponíveis no mercado, sendo necessário o conhecimento das principais características construtivas para uma melhor seleção de qual tecnologia melhor se aplicaria a cada projeto. O intuito desta seção é apenas de fazer um levantamento dos principais dispositivos e como é feita a implementação em hardware dos algoritmos, de modo a mostrar a diversidade de opções.

Um FPGA consiste de um arranjo bidimensional de blocos lógicos, como mostrado na Figura 2.6. Tais blocos são ligados por elementos de interconexão programável e os elementos nas bordas do arranjo possuem algumas características adicionais para utilização como entradas e saídas do dispositivo (blocos de IO) (Aragão,1998).

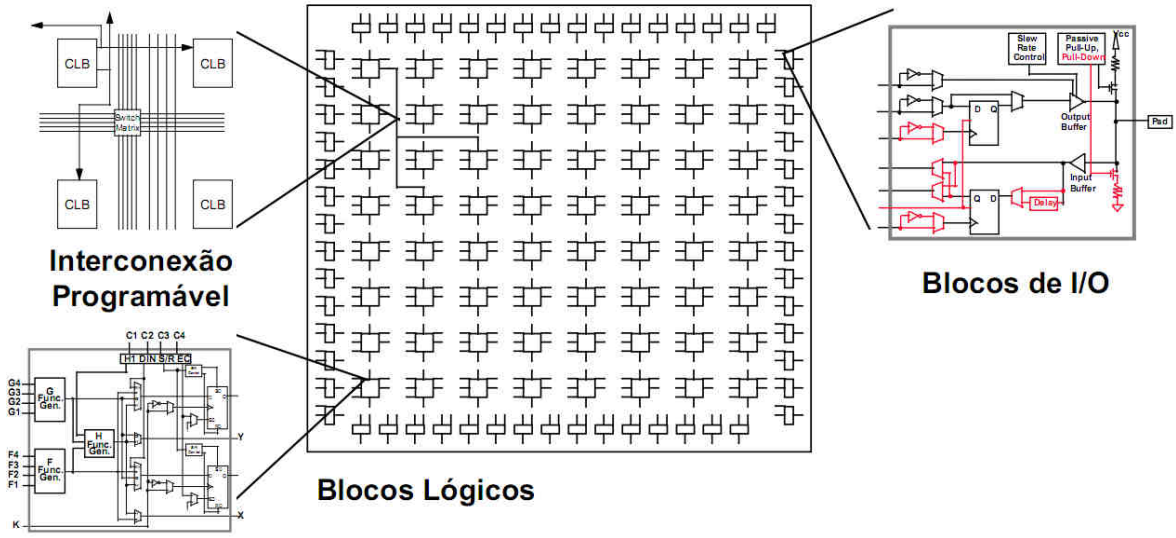


Figura 2.6 – Estrutura de um FPGA (Aragão,1998).

Quando da escolha de um determinado FPGA, três aspectos determinam o desempenho de um circuito implementado e também a densidade (grau de integração) do FPGA. Devem ser considerados: (a) a tecnologia de programação utilizada, (b) a arquitetura interna dos blocos lógicos e (c) a arquitetura de roteamento.

2.4.1 – Tecnologia de programação

A programação de um FPGA implica em ligar e desligar inúmeras chaves lógicas e as propriedades elétricas dessas chaves influenciam sobremaneira nas características de desempenho de cada dispositivo. A Tabela 2.1 mostra as três principais tecnologias de programação de um FPGA e algumas características de cada uma (Vahid,2007).

Tabela 2.1 – Resumo das tecnologias de programação de FPGAs.

Tecnologia / Característica	SRAM	Anti-Fuse	Gate Flutuante
Tamanho	Pequeno	Grande	Médio
Velocidade de Programação	Alta	Baixa	Média
Volatilidade	Sim	Não	Não
Reprogramabilidade	Sim	Não	Sim

2.4.2 – Blocos Lógicos

Os blocos lógicos dos FPGAs comerciais são capazes de implementar uma grande variedade de funções lógicas, tanto combinacionais quanto seqüenciais, podendo ser simples como um transistor ou tão complexo quanto um microprocessador. Geralmente pode-se dividi-los em dois grupos básicos já citados rapidamente no Capítulo 1: granularidade fina e granularidade grossa.

Os blocos lógicos de granularidade fina (*fine-grain*) executam operações de apenas um bit de largura, definindo circuitos no nível de portas lógicas [Bra05]. Desse modo, tem-se uma grande flexibilidade de projeto, porém o trabalho de reconfiguração do dispositivo é bastante demorado, e como possuem a largura de apenas um bit, geralmente são utilizados aos milhares, mesmo para aplicações simples, necessitando de grande quantidade de trilhas de conexão.

Já os blocos lógicos de granularidade grossa (*coarse-grain*), representados principalmente pelas rDPAs (*reconfigurable data-path arrays* - arranjos de via de dados reconfiguráveis) já operam em nível de transferência entre registradores (RTL - *Register Transfer Level*), tendo sido muito utilizados para a implementação de aceleradores de hardware para diversas aplicações (Becker&Hartenstein,2003).

Ambos os tipos de blocos lógicos possuem adicionalmente algum elemento de lógica seqüencial, geralmente um flip-flop tipo D.

2.4.3 – Arquitetura de Roteamento

A arquitetura de roteamento define a interconexão entre os blocos lógicos, sendo de extrema importância quando da síntese de um circuito, pois pode determinar a freqüência

máxima de operação e qual o espaço ocupado pela implementação. É importante saber se uma dada arquitetura permite o roteamento completo, ou seja, se um determinado circuito pode ser implementado naquele FPGA específico.

Assim como qualquer dos outros elementos da estrutura de um FPGA, os blocos de roteamento e interconexão ocupam espaço, sendo um ponto importante o balanceamento entre a flexibilidade da arquitetura de roteamento e a área a ser ocupada por ela.

2.4.4 – Arquiteturas de FPGAs comerciais

Hoje são comercializados 4 tipos de arquiteturas básicas de FPGAs, ilustradas na Figura 2.6 : arranjo simétrico, baseadas em linhas, hierárquicas e mar de portas (*sea-of-gates*).

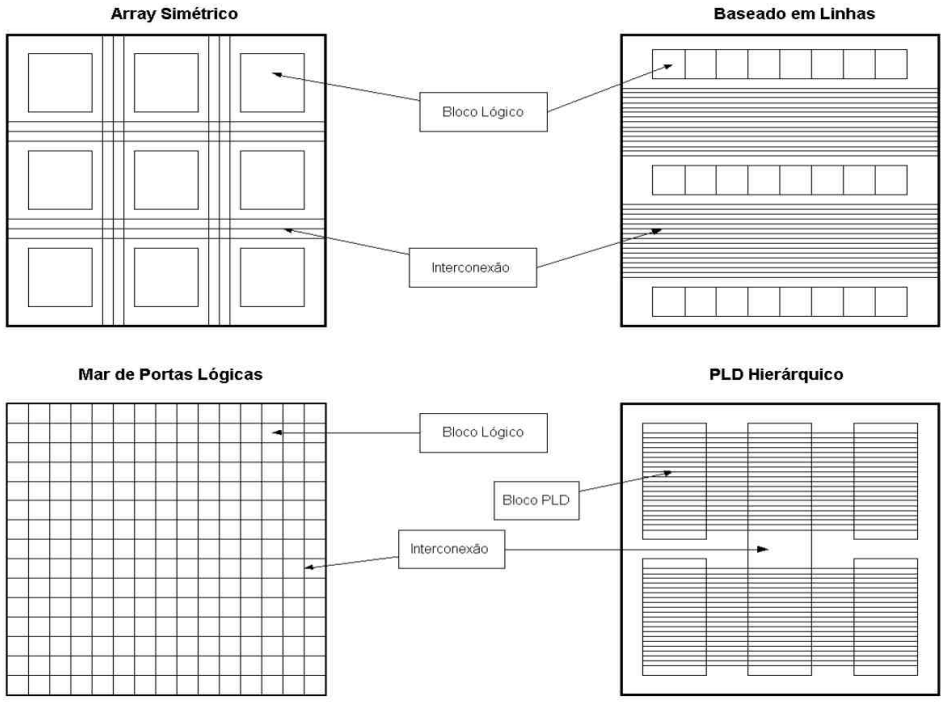


Figura 2.6 – Arquiteturas de FPGAs comerciais (Aragão,1998).

Os fabricantes de FPGAs provêm dispositivos que combinam os diferentes tipos de características citados (forma de programação, blocos lógicos, roteamento e arquitetura geral), além de mais algumas particularidades de cada um, como as interfaces de IO, multiplicadores embarcados, e até mesmo processadores inteiros previamente configurados. A seleção de um determinado dispositivo deve levar em consideração todas

essas características, pois podem influenciar significativamente no sucesso de uma determinada implementação (Aragão,1998).

2.4.5 Ciclo de desenvolvimento com FPGAs

O ciclo de desenvolvimento de um projeto com FPGAs exige o uso de ferramentas EDA (*Eletronic Design Automation*) apropriadas, devido à grande complexidade envolvida nesses dispositivos. A Figura 2.7 mostra as diferentes etapas desse processo. Na maior parte das vezes o projetista é responsável pela etapa de especificação e entrada do projeto, seja por meio de HDLs (*Hardware Description Languages*), seja por meio de desenhos esquemáticos. As outras etapas geralmente são automatizadas e otimizadas nas ferramentas de EDA, permitindo que o projetista trabalhe em um nível mais alto de abstração.

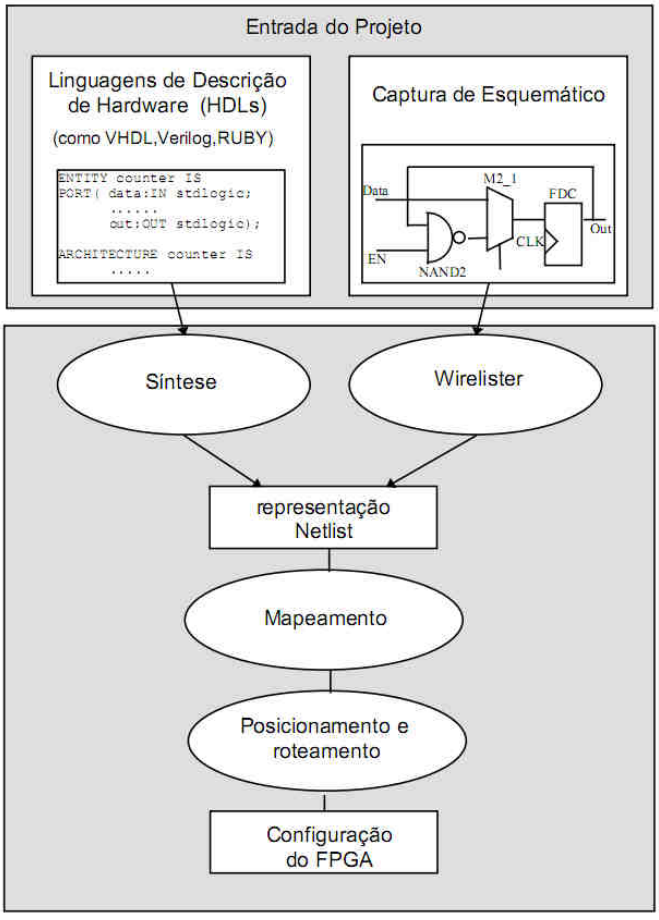


Figura 2.7 – Etapas de um projeto com FPGAs (Aragão,1998).

A etapa de entrada do projeto é geralmente feita de duas maneiras: (a) utilizando uma linguagem de descrição de hardware, ou (b) utilizando uma ferramenta de projeto esquemático. As linguagens de descrição de hardware mais comuns são a Verilog e a

VHDL, sendo esta última um padrão IEEE para descrição de hardware. Ambas são linguagens de alto nível, permitindo uma maior abstração ao projetista.

Devido ao grau de abstração do projeto de alto nível, em que os projetistas se preocupam mais com o funcionamento do sistema que com seu desempenho, essa fase de entrada do projeto não é otimizada, sendo necessária a utilização de técnicas complexas para otimizar os circuitos gerados, minimizando a utilização de recursos, por meio de redução das equações lógicas. Essa fase é chamada Mapeamento. A fase seguinte é a de Posicionamento, em que um aplicativo específico seleciona o local em que cada bloco lógico da implementação será colocado, de acordo com o modelo e arquitetura do FPGA. Após o Posicionamento, começa a fase de roteamento, em que um algoritmo utiliza a arquitetura de interconexão para ligar os blocos lógicos, sendo necessário assegurar que todas as conexões desejadas sejam feitas, e ao mesmo tempo tentar minimizar os atrasos do circuito, maximizando sua velocidade de processamento. A última etapa é a geração do arquivo de configuração do FPGA, ou seja, os dados necessários para configurar o circuito desejado dentro do dispositivo reconfigurável.

Neste trabalho foi utilizada a linguagem VHDL em conjunto com uma ferramenta de desenho esquemático integrada à ferramenta EDA Quartus II, da Altera Corporation. O Quartus II permite o projeto por meio da descrição em linguagens de hardware e por meio de desenhos esquemáticos, automatizando os processos de síntese, mapeamento e roteamento.

3 ANÁLISE DE ALGORITMOS DE PROCESSAMENTO DE IMAGENS

Agora terá início o processo de definição de uma metodologia para mapeamento dos algoritmos de processamento de imagens em hardware, ou seja, um conjunto de considerações teóricas e identificação de padrões que permita extrair características mais gerais e comuns destes algoritmos, de modo a facilitar a separação quanto aos tipos de instrução, nível de processamento, grau de paralelismo e redundâncias.

Como já citado, a quantidade de técnicas de processamento de imagens e vídeos torna inviável a contemplação de todas neste trabalho. Por esse motivo, escolheram-se operações que combinadas formam a base dos algoritmos mais complexos de processamento de imagens e permitem explorar bem os conceitos desejados para a implementação em hardware reconfigurável. Neste capítulo serão mostrados os algoritmos escolhidos, sua utilização em processamento de imagens e como foi feita a identificação de características de interesse para sua implementação.

3.1 - Limiarização

Muitas vezes quando da análise de uma imagem, não é de interesse saber a intensidade luminosa de um determinado objeto, mas simplesmente obter sua forma. Para tanto, uma imagem pode ter reduzida a quantidade de informação presente, passando por uma operação de limiarização. A limiarização de uma imagem consiste na transformação de seus valores de intensidade (“escala de cinza”) em uma representação com apenas dois valores, ou seja, uma imagem preto-e-branco no sentido literal.

A limiarização (ou binarização) na sua forma mais comum, pode ser uma operação do tipo pontual (tópico 2.2.2), sendo caracterizada pela comparação do valor de cada pixel com um valor limiar global pré-determinado. Caso o pixel apresente um valor maior que o limiar, sua saída correspondente será um pixel de valor ‘1’. Caso contrário, a saída terá valor ‘0’.

A Figura 3.1 mostra três operações de limiarização para a mesma imagem (a) de oito bits (valores de 0 a 255), com valores de limiar iguais a (b) 128, (c) 64 e (d) 32.

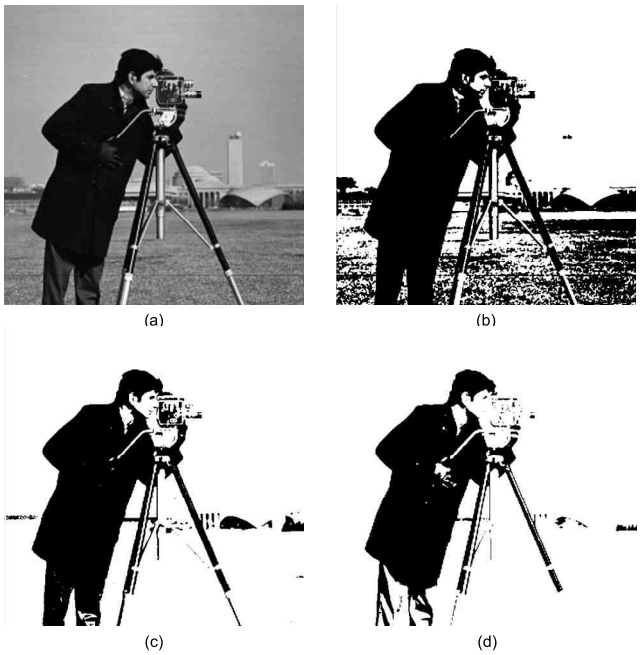


Figura 3.1 – Exemplo de binarização com limiares diversos:
(a)imagem original (b) 128 (c) 64 e (d) 32.

Nota-se que houve uma perda gradativa da quantidade de informações, principalmente do fundo da imagem, de acordo com o limiar.

3.1.1 – Arquiteturas para o Algoritmo

A limiarização é realizada de modo independente para cada pixel, possuindo uma característica de paralelismo de dados. Pela Taxonomia de Flynn, o modelo sugerido seria de uma máquina SIMD, ou seja, uma única instrução (binarização) aplicada a diversos dados ao mesmo tempo. Tal arquitetura também possui as características de utilização apenas de Memórias Privadas, já que, como as operações são pontuais, cada processador (elemento de processamento) acessa apenas os dados de uma posição espacial da imagem (um único pixel).

Uma implementação ideal consistiria basicamente em uma matriz de elementos de processamento (EPs), com duas entradas (o pixel de interesse e o limiar) e uma saída (Figura 3.2). O número de EPs seria igual ao número de pixels da imagem.

A arquitetura de processamento para limiarização, apresentada na Figura 3.2, possui características bastante simples, com cada elemento de processamento sendo composto basicamente de duas entradas, um comparador e uma saída. Tal simplicidade implica em

baixa utilização de recursos para um único EP. Neste caso seria possível binarizar uma imagem completa em apenas um ciclo de relógio, atingindo uma condição ótima.

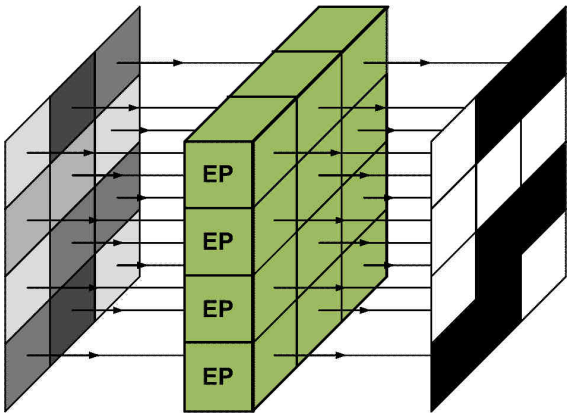


Figura 3.2 – Matriz de elementos de processamento em uma operação de limiarização.

Porém, essa arquitetura só é viável atualmente para imagens pequenas, pois há a necessidade de que todos os pixels da imagem estejam disponíveis simultaneamente no início do processamento. Como a maioria das memórias utilizadas apresenta poucas portas de dados, outras arquiteturas devem ser pensadas. Neste caso o "Gargalo de von Neumann" torna-se bastante evidente, já que restrições de acesso à memória limitam a capacidade de processamento total do sistema.

Uma alternativa mais factível é aproveitar a característica serial das memórias mais comuns e dos sistemas de aquisição de imagens. Geralmente, os pixels chegarão ao processador um a um, em um fluxo de dados serial (também chamado de *stream* de dados), sincronizados por um sinal de relógio. No caso da limiarização, poder-se-ia trabalhar com apenas um elemento de processamento que receberia e processaria os pixels um a um, como mostra a arquitetura da Figura 3.3.

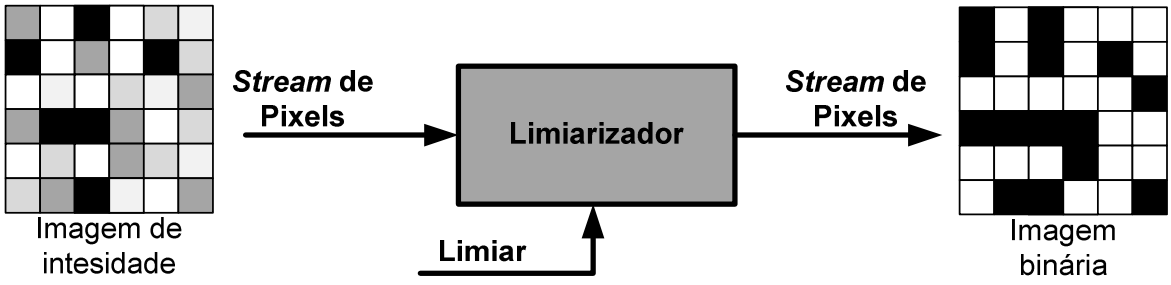


Figura 3.3 – Arquitetura para Limiarização Global.

3.2 – Filtragem Espacial por Convolução/Correlação

Qualquer sistema de aquisição e transmissão de dados (uma câmera adquire e transmite dados visuais) está sujeito ao ambiente em que se encontram os dados e os equipamentos utilizados. Fatores como temperatura, pressão, luminosidade, campos elétricos e magnéticos, etc., podem influenciar sobremaneira na qualidade de uma determinada imagem. Com a influência desses fatores externos, uma imagem pode sofrer degradação ou perda de qualidade em virtude da introdução de ruídos, perda de contraste, borramento e distorções (Pedrini,2008).

A eliminação de ruídos em sinais é um campo de estudo já há muito consolidado, sendo essencial para isso a utilização de filtros. Um filtro consiste em um operador de transformação da imagem original em uma imagem com características de interesse realçadas (Pedrini,2008). Existem duas categorias básicas de divisão das técnicas para filtragem de imagens: (a) filtragem no domínio da freqüência e (b) filtragem no domínio espacial. A primeira categoria envolve a atuação sobre o espectro de freqüências da imagem, utilizando principalmente a Transformada de Fourier. Já a segunda categoria consiste na manipulação direta dos pixels da imagem [Gon06].

Neste trabalho são analisadas apenas as técnicas baseadas no domínio espacial, porém, muito do que é discutido pode ser aproveitado para uma extrapolação ao domínio da freqüência, principalmente as características de partição dos algoritmos em suas operações mais simples e a identificação dos tipos de paralelismo presentes.

Existem basicamente três tipos de filtros: (a) passa-baixa, (b) passa-faixa e (c) passa-alta, cada um responsável por realçar/atenuar determinadas características da imagem. Um filtro passa-baixa atenua detalhes da imagem, causando um efeito de borramento, suavizando mudanças abruptas de intensidade na imagem (mudanças abruptas representam altas freqüências espaciais). Já um filtro passa-alta realiza um aguçamento dos detalhes mais finos da imagem, realçando as altas freqüências espaciais.

A maioria das aplicações de filtros espaciais é baseada em uma operação chamada janelamento (*windowing*), descrita a seguir, nas Figuras 3.4 a 3.7. Tal operação classifica-se como uma *operação de vizinhança*, pois, como será mostrado, um conjunto de pixels de entrada gera um pixel de saída (tópico 2.2, capítulo 2).

O primeiro passo consiste em sobrepor a máscara do filtro a uma região da imagem,

centrada em um pixel (Figura 3.4). A escolha da máscara definirá o tipo de filtragem efetuado.

Figura 3.4 – Sobreposição da máscara sobre a imagem.

O segundo passo consiste em multiplicar cada valor da máscara pelo valor do pixel superposto (Figura 3.5).

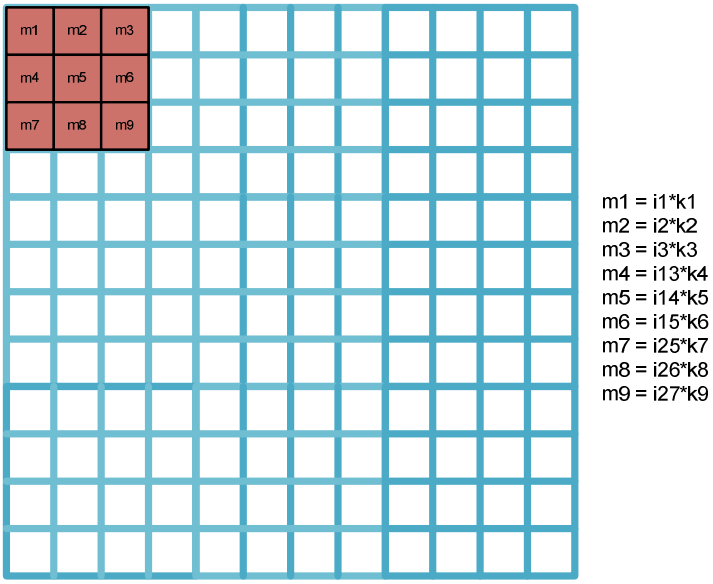


Figura 3.5 – Multiplicação da mascara pelos pixels superpostos.

O terceiro passo corresponde à soma dos produtos do passo anterior e a atribuição do valor

da soma ao pixel correspondente ao pixel central na imagem de saída (Figura 3.6).

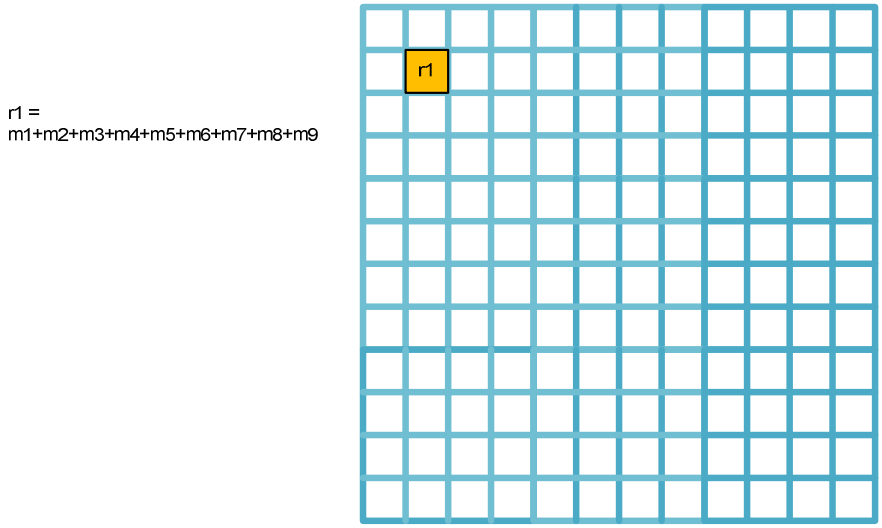


Figura 3.6 – Soma das multiplicações e atribuição do resultado ao pixel correspondente na imagem de saída.

O último passo consiste em deslocar máscara centralizando-a em um pixel vizinho ao pixel anterior na imagem original, e repetirem-se os passos anteriores (Figura 3.7). Essa seqüência deve ser repetida até todos os pixels da imagem original terem sido contemplados.

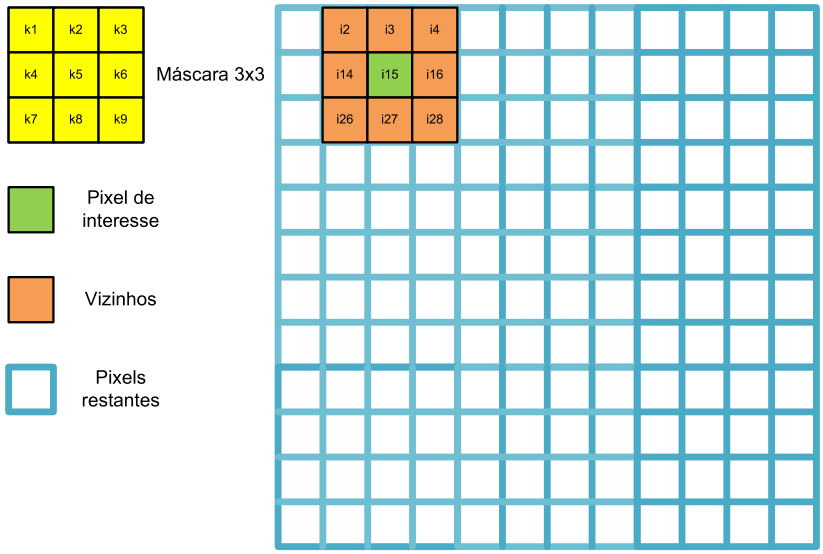


Figura 3.7 – O último passo: centralizar a máscara sobre outro pixel da imagem original, repetindo os passos anteriores.

Uma imagem digital representa um espaço finito, e as operações de janelamento

apresentam uma distorção quando de sua aplicação sobre as bordas da imagem, já que os pixels de borda não possuem todos os seus vizinhos. Assim sendo, um método bastante comum de evitar-se esse efeito de borda consiste em aplicar-se o filtro apenas aos pixels que possuem todos os vizinhos, eliminando os outros. Desse modo, ocorre uma redução no tamanho da imagem final em relação à original.

A operação de janelamento é muito semelhante às operações de convolução e correlação discretas bidimensionais, cujas definições matemáticas são dadas pela Equações 3.1 e 3.2, respectivamente (Gonzalez&Woods,2006).

$$k(x,y) * i(x.y) = \sum_{l=\left[-\frac{m}{2}\right]}^{\left[\frac{m}{2}\right]} \sum_{j=\left[-\frac{n}{2}\right]}^{\left[\frac{n}{2}\right]} k(l,j).i(x-l,y-j) \qquad 3.1$$

$$k(x,y) \circ i(x.y) = \sum_{l=\left[-\frac{m}{2}\right]}^{\left[\frac{m}{2}\right]} \sum_{j=\left[-\frac{n}{2}\right]}^{\left[\frac{n}{2}\right]} k(l,j).i(x+l,y+j) \qquad 3.2$$

Os filtros, máscaras ou *kernels*, $k(x,y)$, podem ter tamanhos e formas variados, possuindo valores determinados arbitrariamente de acordo com a função. No tópico a seguir, serão mostrados alguns exemplos de aplicação da convolução e da correlação.

3.2.1 – Exemplos de aplicação

Existem dois tipos básicos de ruído que são adicionados às imagens durante o processo de aquisição por câmeras:

- Ruídos impulsivos – caracterizados por pontos claros e escuros, principalmente, sendo também conhecidos como *speckle* (salpico), ou *salt-and-pepper* (sal-e-pimenta).
- Ruído branco – caracterizado por sua distribuição aleatória, tendo componentes em toda a faixa de valores possíveis para os pixels.

A figura 3.8 mostra três imagens, sendo (a) sem ruídos, (b) com ruído impulsivo e (c) com ruído branco.

Figura 3.8 – Imagens (a) sem ruído, (b) com ruído impulsivo e (c) com ruído branco.

As características de cada tipo de ruído fornecem aos projetistas de filtros, as informações necessárias à especificação dos valores utilizados nas máscaras de filtragem.

O ruído tipo *sal-e-pimenta* apresenta características de variações bruscas de intensidade de cor, o que o classifica como um ruído de alta freqüência, e sugere a utilização de um filtro passa-baixa. Um dos filtros que poderia ser utilizado é o filtro de média. A figura 3.9 mostra a máscara do filtro de média 3x3, e o resultado de sua aplicação à figura 3.8(b).

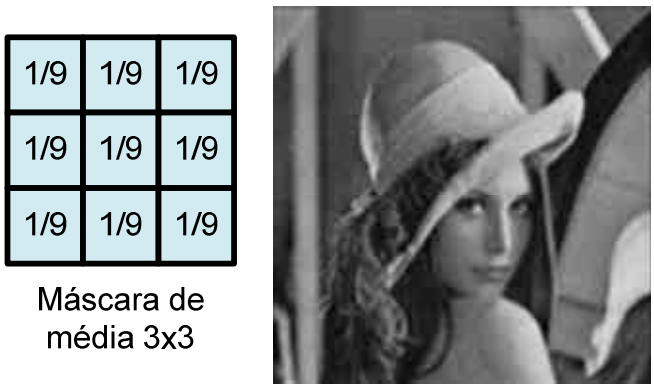


Figura 3.9 – Máscara de média 3x3 e o resultado de sua aplicação sobre a Figura 3.8(b).

Observa-se na Figura 3.9 que o filtro de média eliminou o ruído sal-e-pimenta, porém tornou a imagem um tanto borrada, pela atenuação das características de alta freqüência. Um dos grandes problemas relacionados à utilização de filtros passa-baixa é a supressão de detalhes finos e bordas das imagens [Ped80].

Para a imagem da Figura 3.8(b), com a presença de ruído branco, um filtro apropriado seria o Gaussiano, um filtro estatístico que também pode ser aplicado utilizando a convolução. A máscara do filtro Gaussiano é obtida pela discretização de uma função Gaussiana

bidimensional. Os valores da máscara são valores assumidos pela Gaussiana, com média e desvio-padrão de acordo com o projeto do filtro. A Figura 3.10 mostra uma máscara de filtro Gaussiano de tamanho 5x5, com média 0 e desvio padrão 1.

1	4	6	4	1
4	16	24	16	1/9
6	24	36	24	1/9
4	16	24	16	1/9
1	4	6	4	1

Máscara Gaussiana

Figura 3.10 – Máscara Gaussiana 5x5, média 0 e desvio-padrão 1, e a correspondente imagem filtrada com essa máscara.

Na maioria das vezes, quando da análise de uma imagem, queremos extrair características dos objetos que a compõem. Algumas vezes queremos identificar detalhes finos das imagens, e realçá-los, de modo a facilitar sua identificação. Os detalhes finos apresentam características de alta freqüência, ou seja, mudanças mais bruscas nos níveis de intensidade das imagens, sendo sugerida a utilização de filtros passa-alta para seu realce.

O cálculo da média de uma imagem é análogo à operação de integração, e apresenta uma característica de filtragem passa-baixa. Com base nisso, pode-se supor que a diferenciação (operação oposta à integração) é análoga à filtragem passa-alta. Assim sendo, ao calcularmos a taxa de variação das intensidades da imagem, teremos uma filtragem passa-alta.

Algumas máscaras muito utilizadas para o realce por derivação são mostradas na Figura 3.11.

-1	-1	-1
0	0	0
1	1	1

Prewitt - horizontal

-1	0	1
-1	0	1
-1	0	1

Prewitt - vertical

-1	-2	-1
0	0	0
1	2	1

Sobel - horizontal

-1	0	1
-2	0	2
-1	0	1

Sobel - vertical

Figura 3.11 – máscaras para realce por derivação em *x* (vertical) e *y* (horizontal).

As máscaras verticais da Figura 3.11 realçam características finas de maior variação na direção x, e as máscaras horizontais na direção y. A Figura 3.12 mostra os resultados de aplicação das máscaras da Figura 3.11.

Observando os resultados da Figura 3.12, pode-se confirmar a utilização dos filtros de realce por derivação na identificação de altas freqüências espaciais. As figuras 3.12(a) e (c) mostram um maior destaque nas linhas horizontais (mais notável nos olhos e na boca), enquanto que as figuras 3.12(b) e (d) destacam mais as linhas verticais (observar o nariz e o cabelo).

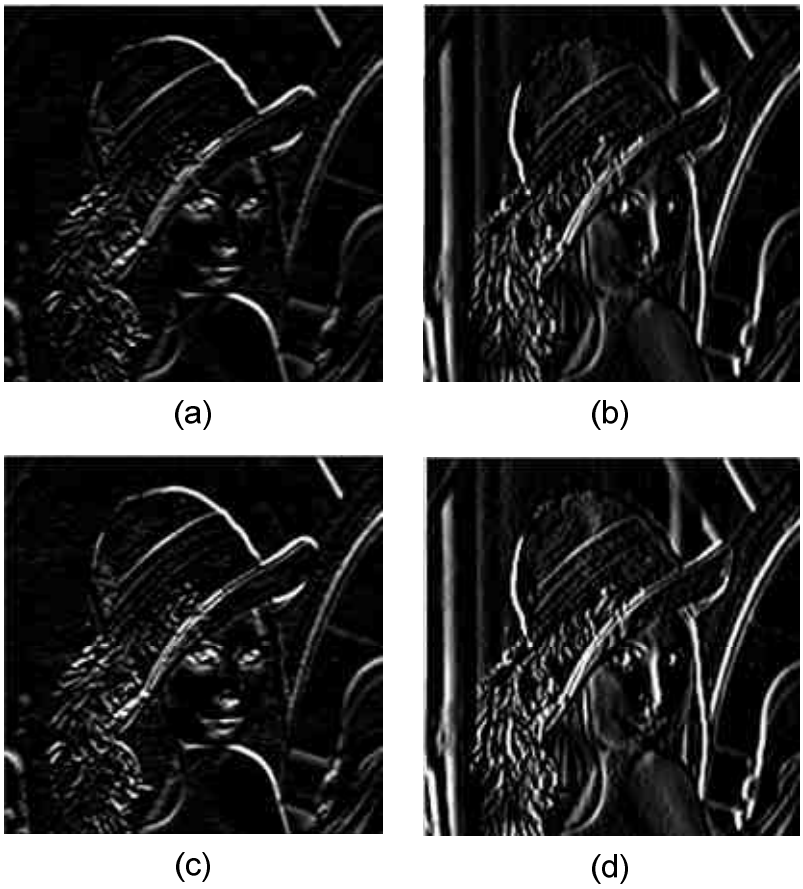


Figura 3.12 – Resultados de aplicação dos filtros da Figura 3.8. (a)Prewitt horizontal, (b)Prewitt vertical, (c) Sobel horizontal, (d)Sobel vertical.

A Correlação (Equação 3.2) pode ser aplicada para filtragem do mesmo modo que a Convolução, por meio de uma operação de *Windowing*, bastando para isso calcular-se a transposta da máscara usada na Convolução. Por exemplo, caso desejássemos filtrar uma imagem por Correlação utilizando a máscara do filtro Prewitt Vertical, Figura 3.11,

bastaria executar a operação de Convolução com a máscara do filtro Prewitt Horizontal.

Porém, há outra aplicação muito simples da Correlação na identificação de padrões de objetos [Gon06]. A Figura 3.13 ilustra esse conceito. Nessa figura, a primeira imagem é a original, composta de alguns símbolos; a segunda imagem é o padrão a ser buscado, e a última imagem é o resultado da operação de Correlação entre a imagem original e o padrão. O ponto com nível de cinza mais intenso corresponde à posição de melhor correspondência (casamento) entre o padrão e a imagem.

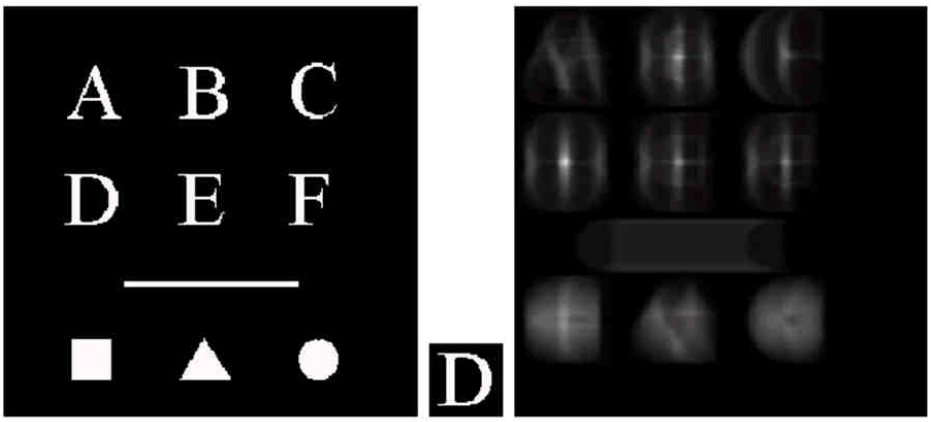


Figura 3.13 – Casamento por Correlação. (a) Imagem original, (b) padrão a ser identificado, (c) imagem resultante da operação de Correlação da imagem original com o padrão.

3.2.2 – Arquiteturas dos Algoritmos

As equações 3.1 e 3.2 (Convolução e Correlação, respectivamente) mostram que os cálculos para cada pixel, ou seja, o cálculo de vizinhança para um pixel de interesse não interfere nos cálculos dos pixels restantes, sendo ou não vizinhos. Isso é uma característica de paralelismo de dados (no caso, paralelismo de pixels): a mesma função é aplicada independentemente em todos os pixels da imagem.

Realizando as mesmas análises feitas para o algoritmo de Limiarização, a arquitetura proposta também seria do tipo SIMD, com uma única operação (Convolução/Correlação) aplicada a vários dados, de forma independente. Porém, neste caso, ter-se-ia a utilização de Memória Compartilhada, visto que, por ser uma Operação de Vizinhança, tem-se de utilizar mais de cada pixel simultaneamente. A Figura 3.14 mostra o compartilhamento de pixels nas operações para cada saída.

A Figura 3.14(a) mostra a imagem de entrada, e a 3.14(b) mostra a imagem de saída. As diferentes hachuras (45°, 135°, horizontal e vertical) ilustram os dados utilizados para o cálculo de cada pixel de saída. No exemplo foi utilizada uma máscara de tamanho 3x3, o que equivale a dizer que cada EP utiliza nove valores da imagem de entrada para o cálculo de um único pixel de saída. O compartilhamento chega a ser de até nove EPs utilizando o mesmo dado de entrada. Máscaras maiores exigiriam compartilhamentos maiores.

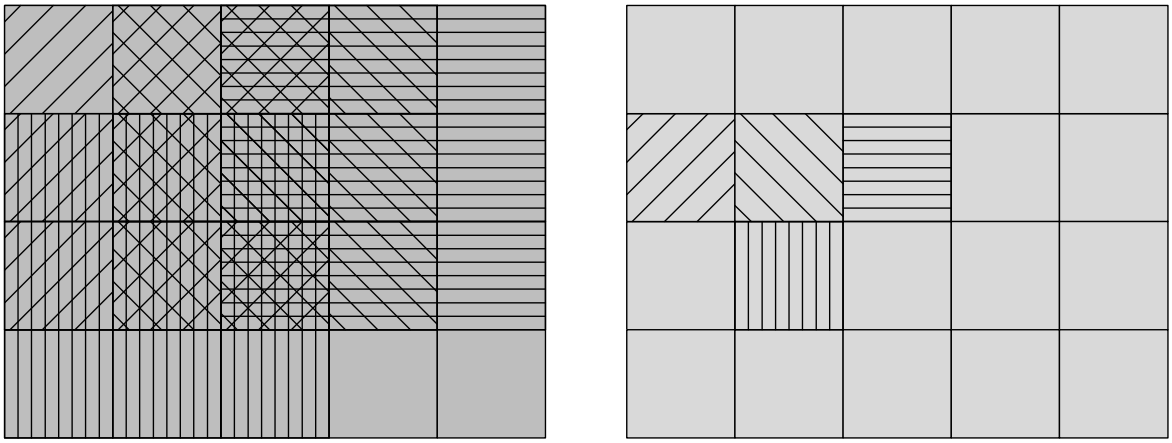


Figura 3.14 – Compartilhamento de memória no cálculo da Convolução/Correlação.

A arquitetura ótima para o processamento neste caso é semelhante à da Limiarização, com cada elemento de processamento trabalhando de forma independente e paralela. Os mesmos problemas de acesso à memória são encontrados nesta arquitetura.

Os elementos de processamento para esta operação são mais complexos que os da Limiarização, sendo conveniente a análise de sua estrutura interna, para permitir melhorias. Seja $f(x,y)$ a imagem de entrada, k(x,y) a máscara de filtragem, e (x_i,y_i) as coordenadas do pixel de interesse. O pixel de saída correspondente, r(x,y) é calculado pela equação 3.3.

$$r_{(x)(y)} = k_{(x-1)(y-1)} \cdot f_{(x-1)(y-1)} + k_{(x)(y-1)} \cdot f_{(x)(y-1)} + k_{(x+1)(y-1)} \cdot f_{(x+1)(y-1)} +$$
$$+ k_{(x-1)(y)} \cdot f_{(x-1)(y)} + k_{(x)(y)} \cdot f_{(x)(y)} + k_{(x+1)(y)} \cdot f_{(x+1)(y)} +$$
$$+ k_{(x-1)(y+1)} \cdot f_{(x-1)(y+1)} + k_{(x)(y+1)} \cdot f_{(x)(y+1)} + k_{(x+1)(y+1)} \cdot f_{(x+1)(y+1)} \quad 3.3$$

Na aritmética, a operação de multiplicação tem precedência em relação à operação de soma, e nas arquiteturas também ocorre da mesma maneira, ou seja, primeiro devem ser executadas todas as operações de multiplicação, para então proceder-se à soma. Pode-se

observar na Equação 3.3 que as operações de multiplicação são todas independentes entre si, ou seja, o resultado de uma não depende do resultado de outra, de modo que todas podem ser executadas simultaneamente sem problemas.

Essa análise interna do EP de Convolução/Correlação nos induz à arquitetura da Figura 3.15, na qual fica explícita a separação entre as operações e também o conceito de Operação de Vizinhança.

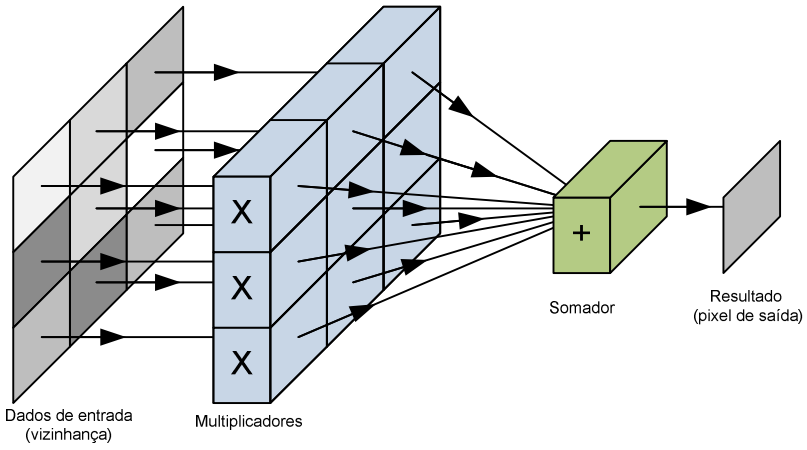


Figura 3.15 – Elemento de Processamento para Convolução/Correlação.

Um processador comum, com arquitetura SISD e com capacidade de executar uma instrução por ciclo de relógio levaria, a menos do tempo de acesso à memória, 10 ciclos de relógio para executar o cálculo de um pixel de saída (9 multiplicações e uma soma), considerando que possua um somador com 9 entradas disponíveis (geralmente as somas são executadas com pares de entrada, o que elevaria a contagem para 17 ciclos de relógio). A arquitetura da Figura 3.15 calcularia um pixel de saída em dois ciclos de relógio apenas (um para as multiplicações e um para a soma).

Se o processamento a ser feito fosse de um vídeo, o ganho de desempenho seria ainda maior, pois se poderia configurar essa arquitetura para trabalhar como um *Pipeline.* A Figura 3.16 mostra um diagrama de tempos em que se pode verificar a diferença de desempenho da arquitetura da Figura 3.15 trabalhando (a) sem *Pipeline* e (b) com *Pipeline*.

Com essa estrutura configurada como um *Pipeline,* após um período inicial de latência (tempo para a ocupação de todos os estágios do *Pipeline*) de três ciclos de relógio, o desempenho de saída seria de apenas um ciclo de relógio por pixel, ou seja, a cada ciclo de relógio seria disponibilizado um pixel na saída do EP.

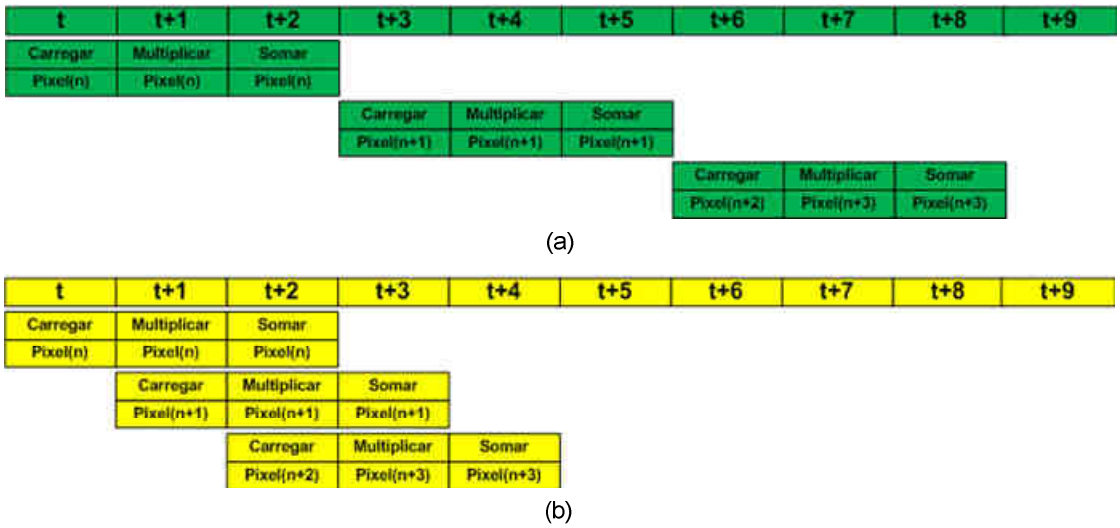


Figura 3.16 – Diagrama temporal para a arquitetura da Figura 3.15 configurada (a) sem *Pipeline* e (b) com *Pipeline*.

Tal como visto para a limiarização, o "Gargalo de von Neumann" mais uma vez se faz presente, e pode-se utilizar as características seriais das memórias e das câmeras, para montar uma máquina de fluxo de dados que permita realizar o processamento sem a necessidade de se ter todos os pixels da imagem disponíveis ao mesmo tempo.

Uma primeira idéia é a utilização do elemento de processamento da Figura 3.15 com uma memória local para armazenar a vizinhança, ou seja, carrega-se serialmente todos os pixels da vizinhança e após isso se procede aos cálculos de multiplicação e soma. Dependendo de o carregamento ser feito a partir de uma memória global contendo a imagem, ou a partir de uma câmera, pode-se montar duas arquiteturas distintas.

Caso a imagem já esteja armazenada em uma memória global, pode-se simplesmente acrescentar à arquitetura da Figura 3.15 um bloco de carregamento de dados, como mostra a Figura 3.17.

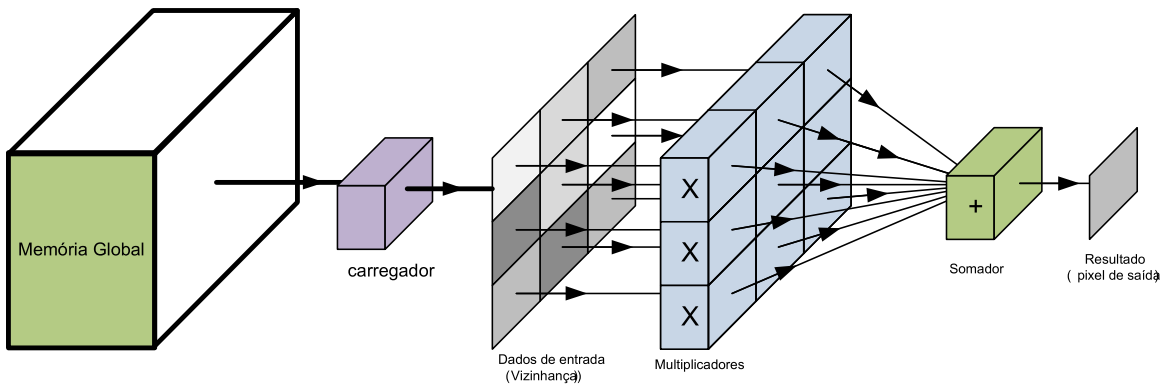


Figura 3.17 – Arquitetura para Convolução/Correlação com apenas um EP.

Essa arquitetura utilizaria apenas um elemento de processamento do tipo da Figura 3.15. Seu diagrama temporal (já com um *Pipeline*) está mostrado na Figura 3.18, e nos informa que o tempo de carregamento dos pixels na memória de vizinhança faria com que o fluxo de dados calculados seja de apenas um pixel a cada oito ciclos de relógio, com uma latência inicial de dez pixels.

t	t+1	t+2	t+3	t+4	t+5	t+6	t+7	t+8	t+9	t+10
Carregar Pixel(n)	Carregar Pixel(n+1)	Carregar Pixel(n+2)	Carregar Pixel(n+3)	Carregar Pixel(n+4)	Carregar Pixel(n+5)	Carregar Pixel(n+6)	Carregar Pixel(n+7)	Carregar Pixel(n+8)	Multiplicar vizinhança	Somar
									Carregar Pixel(n+1)	Carregar Pixel(n+2)

Figura 3.18 – Diagrama temporal para arquitetura da Figura 3.17, já com um *Pipeline.*

Já no caso de se processarem imagens oriundas de uma câmera, talvez não haja a necessidade de armazenamento antes do processamento. Desse modo, uma arquitetura interessante seria a da Figura 3.19.

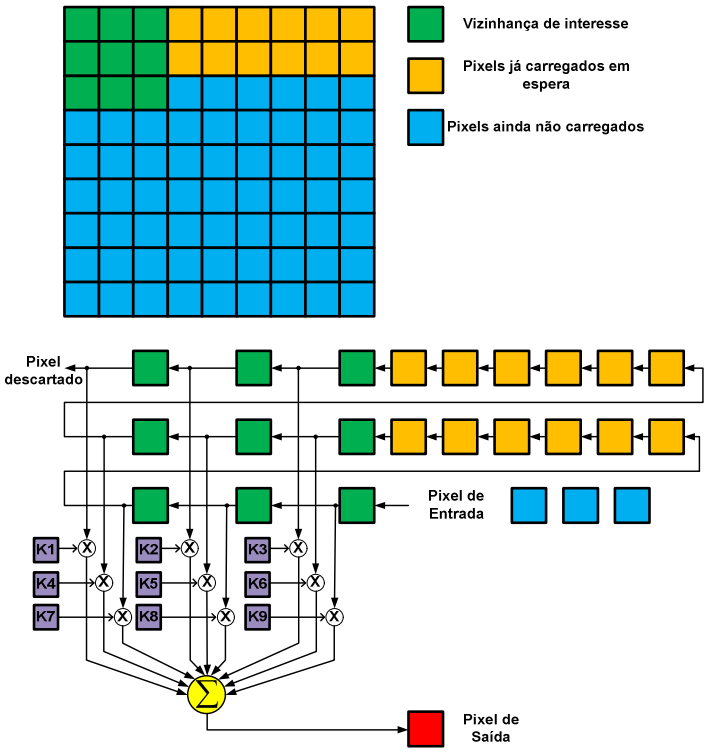


Figura 3.19 – Arranjo Sistólico para Convolução Bidimensional (Wong,2005).

O processo que ocorre na Figura 3.19 é semelhante à seqüência das Figuras 3.4 a 3.7, porém, ao invés de deslocarmos a máscara sobre a imagem, fazemos com que os pixels

passem um a um pela arquitetura. A estrutura composta pelos elementos de cores verde e laranja é um registrador de deslocamento, em que a cada ciclo de aquisição de pixel, todos os pixels são deslocados no mesmo sentido, descartando-se o mais antigo, e carregando-se o mais novo. Cada linha desse registrador de deslocamento equivale exatamente a uma linha da imagem, e os elementos de cor laranja são memórias locais de armazenamento temporário.

O diagrama temporal mostrado na Figura 3.18 assume que todos os nove pixels da vizinhança devem ser carregados sempre que uma saída tiver de ser calculada. No caso da arquitetura da Figura 3.19, após um período de latência de (MxN+3) ciclos de relógio (considerando uma imagem de MxN pixels, e a máscara 3x3) o fluxo de saída seria de um pixel por ciclo de relógio.

3.2.3 – Arquiteturas para eliminação de redundâncias

As melhorias arquiteturais discutidas até agora levaram em consideração principalmente o critério de tempo de processamento por pixel. Porém, a utilização dos recursos disponíveis (a área utilizada em um *chip*, multiplicadores dedicados, etc.) também deve ser considerada pelos projetistas.

Serão agora feitas algumas considerações, tomando como base a arquitetura da Figura 3.15, para diminuição dos recursos utilizados. Em (Kehtarnavaz&Gamadia,2006) são discutidas diversas formas de para melhoria de desempenho de softwares para processamento em tempo real de imagens e vídeos. Algumas das idéias presentes nessa discussão também podem ser relevantes para projetos de hardware. A principal delas é a eliminação de redundâncias no processamento, em outras palavras, evitar cálculos desnecessários, que consomem recursos de tempo, espaço e energia.

A Convolução é uma operação de vizinhança, sendo muito utilizada para eliminação de ruídos ou realce de características (de alta, média e baixa freqüência espacial). Por ser uma operação de vizinhança, existem espaços de memória sendo compartilhados para o cálculo de mais de um pixel de saída. Até agora se considerou o compartilhamento dos dados de entrada apenas (as vizinhanças na imagem original). A partir daqui será considerado o

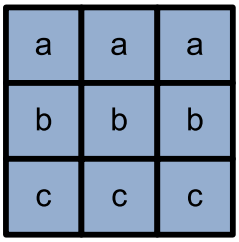
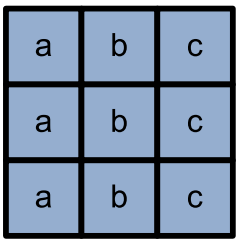
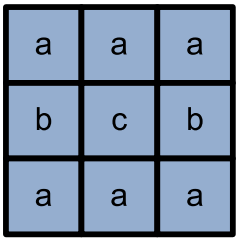
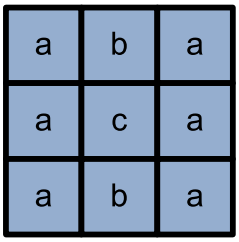
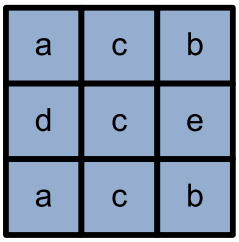
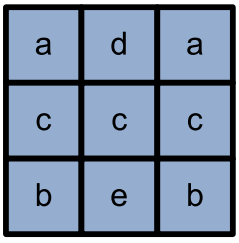
compartilhamento de dados já processados.

A base para a eliminação das redundâncias de processamento na operação de convolução está na análise da máscara a ser utilizada, identificando possíveis simetrias que possam ser utilizadas para minimizar a utilização de multiplicadores nas operações de convolução.

A Tabela 3.1 mostra as máscaras 3x3 de aplicações mais comuns. Foram colocadas letras nas máscaras no lugar de números, como variáveis. Letras iguais equivalem a valores iguais.

Tabela 3.1: Máscaras 3x3 de aplicações comuns.

Tipo de máscara	Máscara	Aplicações comuns
A	a a a a a a a a a	Filtro de Média Filtro-Caixa Ganho
B	a a a a b a a a a	Filtro Passa-Alta Detecção de Pontos
C	a b a b c b a b a	Filtro Gaussiano Laplaciano Laplaciano do Gaussiano
D	a a a b b b a a a	Detecção de Retas Horizontais
E	a b a a b a a b a	Detecção de Retas Verticais

F		Gradiente de Prewitt Direção Vertical
G		Gradiente de Prewitt Direção Horizontal
H		Laplaciano (componente vertical)
I		Laplaciano (componente horizontal)
J		Gradiente de Sobel Horizontal
K		Gradiente de Sobel Vertical

Fazendo uma análise das máscaras apresentadas, pode-se verificar algumas relações entre linhas e colunas. Tais relações permitem derivar arquiteturas para o aproveitamento de operações redundantes, no caso multiplicações, de modo a permitir o processamento em paralelo de dois pixels por ciclo, sem a necessidade de dobrarem-se os recursos utilizados.

1ª. Arquitetura: Máscaras A, D e F.

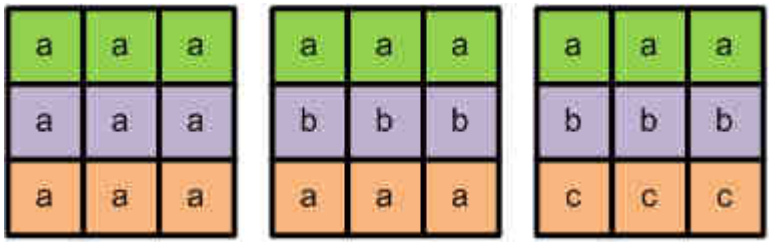


Figura 3.20 – Máscaras com simetrias horizontais identificadas.

Analisando essas máscaras, pode-se perceber que os valores das colunas repetem-se, ou seja, o elemento da 1ª linha da 1ª coluna é o mesmo elemento da 1ª linha da 2ª e da 3ª colunas. Observando a arquitetura da Figura 3.20, percebe-se que, para o cálculo simultâneo de dois pixels, ao invés de apenas um, podemos fazer uma sobreposição das duas máscaras de convolução, conforme a Figura 3.21, em que se suprimiram as linhas de dados, para facilitar a compreensão. Na imagem à esquerda podemos ver como seria a sobreposição da máscara com a imagem, e os elementos inseridos no pipeline. Na imagem à direita, temos a imagem resultante, com a perda das bordas. Os pixels em azul são os dois pixels calculados simultaneamente. Nesta arquitetura a cada ciclo do algoritmo deve-se carregar (ou esperar o carregamento) de dois pixels, e não apenas um. Com tal arquitetura pode-se economizar seis multiplicações redundantes.

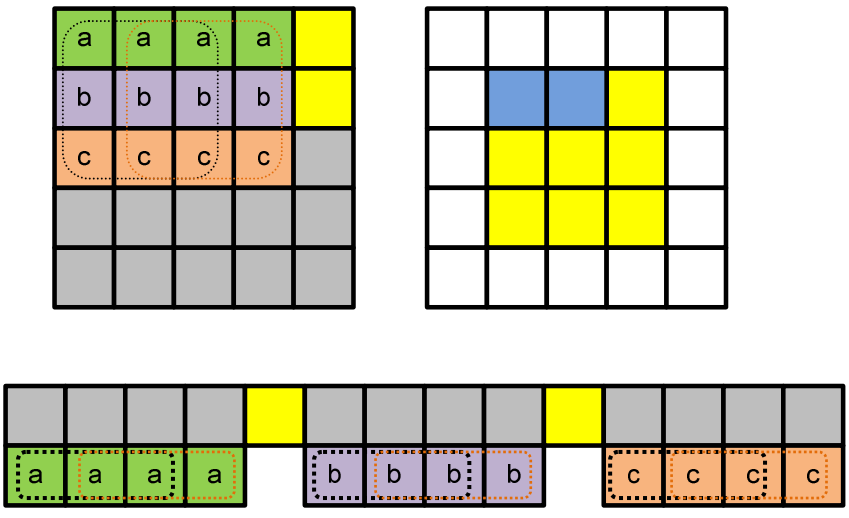


Figura 3.21 – 1ª. Arquitetura para compartilhamento de multiplicações.

2ª Arquitetura: máscaras B, C, E, H, I, K

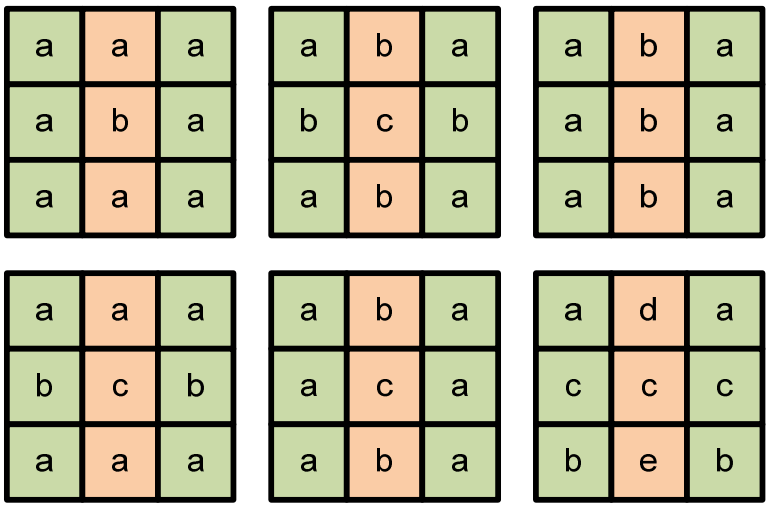


Figura 3.22 – Máscaras com simetrias verticais.

Analisando-se as máscaras da Figura 3.22, pode-se perceber que os valores da 1ª e 3ª colunas repetem-se, ou seja, o elemento da 1ª linha da 1ª coluna é o mesmo elemento da 1ª linha da 3ª coluna. Podemos então fazer uma sobreposição como indica a Figura 3.23, para o cálculo simultâneo de dois pixels. Com essa arquitetura tem-se o aproveitamento de três multiplicações.

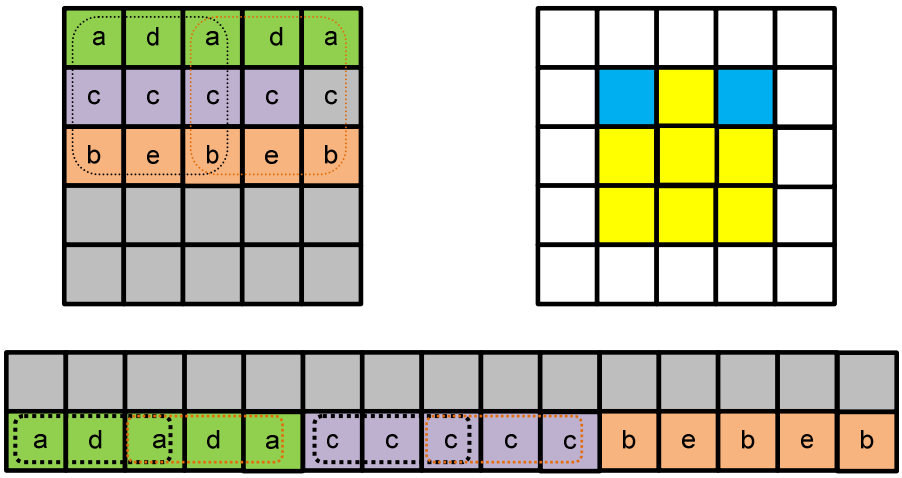


Figura 3.23 – 2ª. arquitetura para compartilhamento de multiplicações.

3ª Arquitetura: Máscaras A, E, G

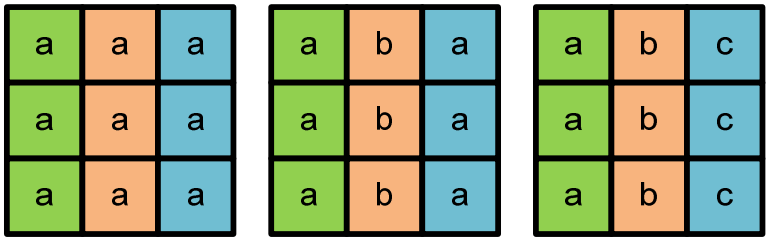


Figura 3.24 – Máscaras com simetrias verticais.

Observando-se as máscaras da Figura 3.24, pode-se notar características semelhantes às máscaras da 1ª arquitetura, com uma rotação de 90 graus, o que nos leva a uma arquitetura semelhante para o aproveitamento dos cálculos redundantes. Na Figura 3.25 mostra-se a arquitetura idealizada para esse grupo de máscaras de convolução. Vale ressaltar aqui que a memória de armazenamento temporário é maior que o das arquiteturas anteriores, ou seja, deve-se esperar um tempo de latência maior para o carregamento dos pixels necessários. A economia nessa arquitetura é de seis multiplicações.

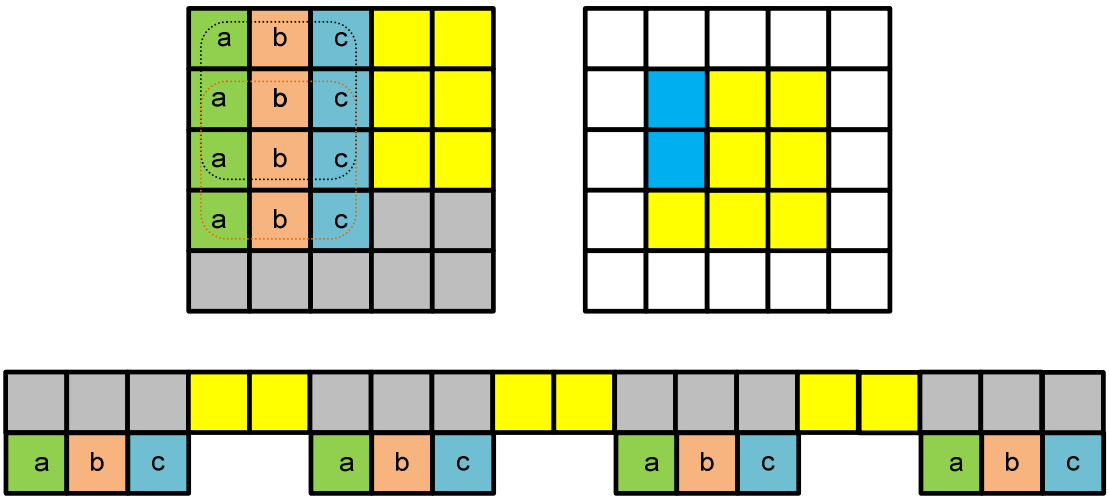


Figura 3.25 – 3ª. Arquitetura para compartilhamento de multiplicações.

4ª Arquitetura: Máscaras A, B, D, E, G, H, I, J

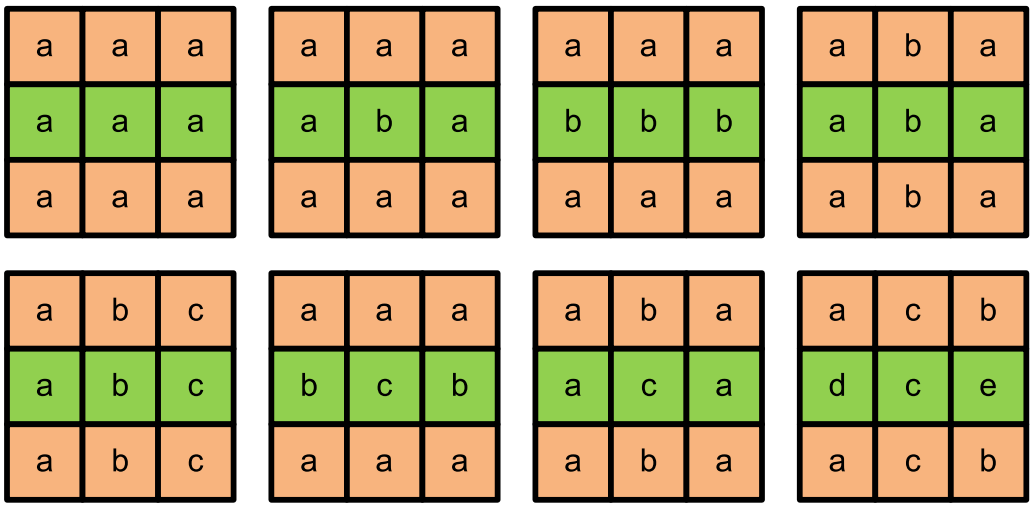


Figura 3.26 – Máscaras com simetria horizontal.

Analisando-se as máscaras da Figura 3.26, há uma semelhança com as máscaras da 2ª arquitetura, com uma rotação de 90 graus. Então, assim como feito para a 3ª arquitetura em relação com a 1ª, pode-se pensar em arquitetura semelhante para a 4ª com relação à 2ª. Há uma economia de três multiplicações.

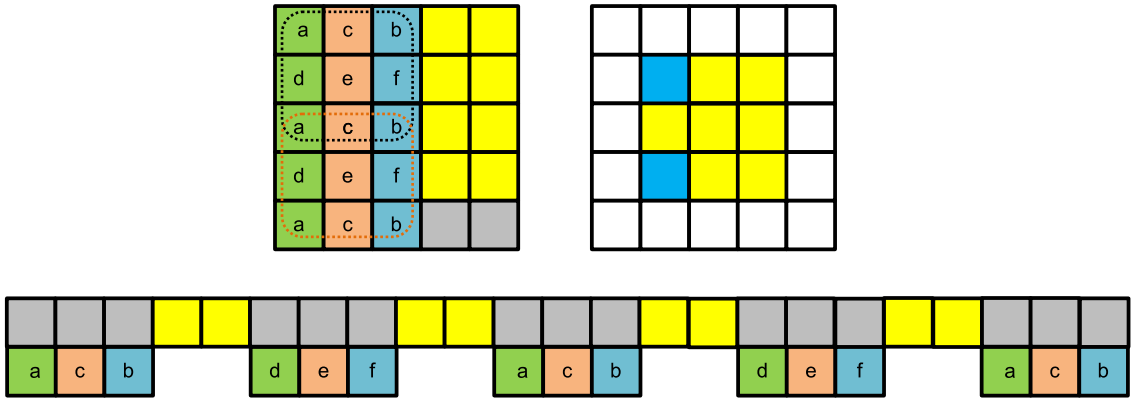


Figura 3.26 – 4ª. Arquitetura para compartilhamento de multiplicações.

Toda a análise para eliminação de redundâncias nas multiplicações foi feita para a utilização de máscaras com dimensões 3x3, porém, pode-se extrapolar facilmente as arquiteturas para máscaras maiores.

3.3 – Filtros de Ordem

Filtros de Ordem (*Rank Order Filters*), são bastante utilizados para eliminação de ruídos do tipo sal-e-pimenta, além de também serem utilizados para realce e atenuação de algumas características das imagens.

Esses filtros são classificados como Operações de Vizinhança, consistindo na utilização de dados de uma vizinhança do pixel de interesse para a geração de um pixel de saída. Seu funcionamento baseia-se em ordenar os elementos da vizinhança e assumir como saída um dos valores, geralmente o maior, o menor ou a mediana.

3.3.1 – Exemplos de aplicação

A Figura 3.27 mostra (a) uma imagem sem ruído, (b) imagem com ruído sal-e-pimenta e (c) a imagem filtrada com um filtro de Mediana de vizinhança 3x3.

Figura 3.27 – (a) Imagem original, (b) imagem com ruído, (c) imagem filtrada.

Observando com atenção as imagens da Figura 3.27, notamos que o filtro de Mediana eliminou o ruído totalmente, porém, causou um efeito de suavização de bordas e detalhes finos da imagem (observar os detalhes da pluma do chapéu).

Os filtros de ordem também podem ser utilizados para realce de certas características da imagem, como mostra a Figura 3.28. Na imagem 3.28(b) a saída é assumida como o menor valor da vizinhança, e na imagem 3.28(c) assume-se o maior valor da vizinhança. Os resultados mostram que a utilização do mínimo local atenua detalhes mais claros da imagem, enquanto que a utilização do máximo local atenua detalhes mais escuros.

Figura 3.28 – (a) imagem original, (b) resultado do filtro de mínimo local, (c) resultado do filtro de máximo local.

Um resultado interessante é a combinação dos filtros de mínimo e máximo para realce de bordas da imagem (Figura 3.29), conseguida subtraindo-se o resultado do filtro de mínimo do resultado do filtro de máximo, por exemplo, subtraindo a imagem 3.28(b) da imagem 3.28(c).

Figura 3.29 – Realce de bordas pela subtração da imagem 3.28(a) da imagem 3.28(b).

3.3.2 – Arquitetura dos algoritmos

Assim como nos outros algoritmos já comentados, os filtros de ordem operam de modo independente para cada pixel de saída, sendo classificada a arquitetura como do tipo SIMD. Por ser uma Operação de Vizinhança, há a utilização de um mesmo valor de entrada por mais de um elemento de processamento, caracterizando a arquitetura como de Memória Compartilhada.

Diversas arquiteturas já foram propostas para ordenação de números [Akl01],[Veg06],[Brä00]. Supondo uma vizinhança de 9 pixels a processar, a arquitetura mais encontrada na literatura é um arranjo sistólico, mostrado na Figura 3.30, em que cada

bloco comparador executa um procedimento conhecido como *Compare-and-Route* (Compare-e-Roteie): compara as duas entradas e atribui o maior valor a uma saída e o menor valor à outra.

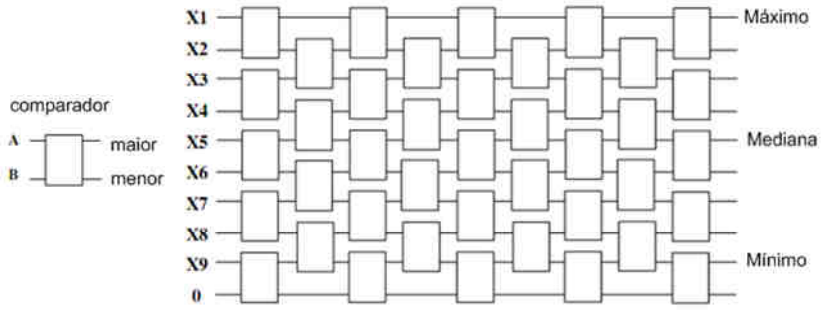


Figura 3.30 – Arranjo sistólico clássico, para ordenação de nove pixels [Veg06].

O Arranjo da Figura 3.30 efetua a ordenação de todos os nove pixels da vizinhança. Nem sempre é necessária a ordenação de todos os elementos, por vezes nos interessa saber apenas o valor da Mediana, do Máximo ou do Mínimo. As Figuras 3.31 a 3.32 mostram algumas possibilidades de redução de consumo de elementos de processamento para os casos desejados.

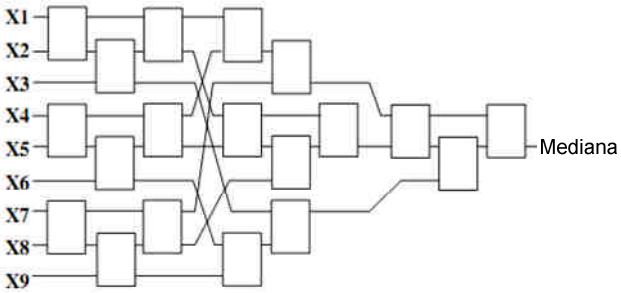


Figura 3.31 – Arranjo sistólico otimizado para cálculo da Mediana [Veg06].

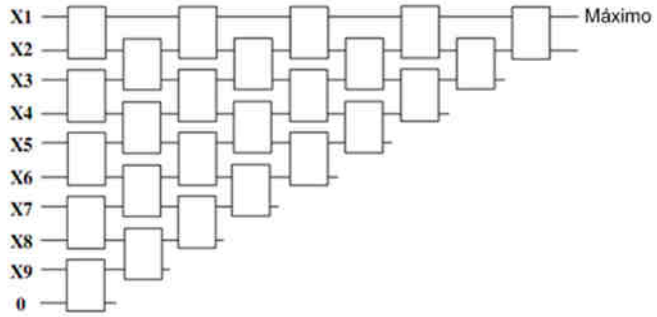


Figura 3.32 – Arranjo Sistólico com eliminação de comparadores desnecessários para cálculo do Máximo da vizinhança.

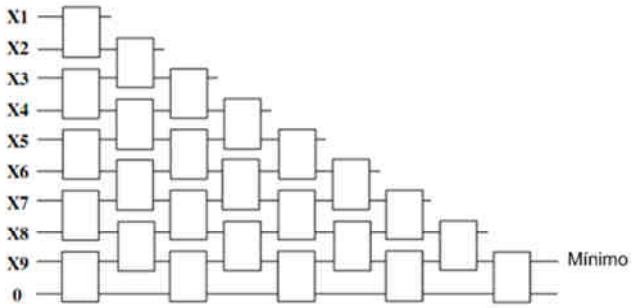


Figura 3.33 – Arranjo sistólico com eliminação de comparadores desnecessários para cálculo do Mínimo da vizinhança.

É interessante notar que os blocos comparadores podem ser vistos como *elementos de processamento* das arquiteturas sistólicas, as quais, por sua vez, são os EPs dos Filtros de Ordem. Os arranjos sistólicos apresentados nas Figuras 3.30 a 3.33 são divididas em camadas, por exemplo, o arranjo da Figura 3.33 possui nove camadas de processamento. Em cada camada podemos identificar paralelismo de dados (o resultado de um EP não influencia no resultado de outro EP) e uma sub-arquitetura do tipo SIMD. A união de todas as camadas forma um *pipeline* com latência de nove ciclos, ou seja, após nove ciclos de relógio ter-se-ia um desempenho de um ordenamento por ciclo.

3.4 – Morfologia Binária

A morfologia é o estudo da forma, da aparência dos objetos em uma imagem [Brä00]. Existem dois operadores básicos de morfologia, chamados Erosão e Dilatação. A Erosão atua no sentido de remover pixels das bordas externas dos objetos, enquanto que a Dilatação acrescenta pixels a essas bordas.

O processo de cálculo de erosão ou dilatação possui estrutura semelhante à Convolução/Correlação. Ambas são Operações de Vizinhança, e utilizam máscaras para o processo. No caso da Morfologia, o processo baseia-se em sobrepor uma máscara à vizinhança, e efetuar operações lógicas entre os elementos sobrepostos. A máscara neste caso é chamada Elemento Estruturante, pois está relacionado à estrutura, à forma.

O elemento estruturante é binário, assim como a imagem, possuindo apenas dois valores: branco e preto, verdadeiro e falso, zero e um. Considerando o elemento estruturante e a vizinhança da Figura 3.34, os valores do pixel de saída resultante da Erosão e da Dilatação são calculados como mostrado na Figura 3.35.

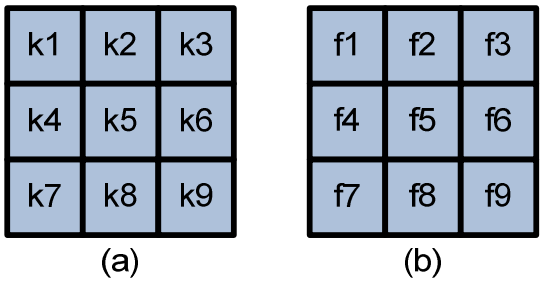


Figura 3.34 – (a) Elemento Estruturante, (b) Vizinhança.

Um exemplo das operações de Dilatação e Erosão é dado na Figura 3.35. Nesse exemplo, um pixel branco é considerado como ‘0’ e um pixel preto como ‘1’, para facilitar a visualização.

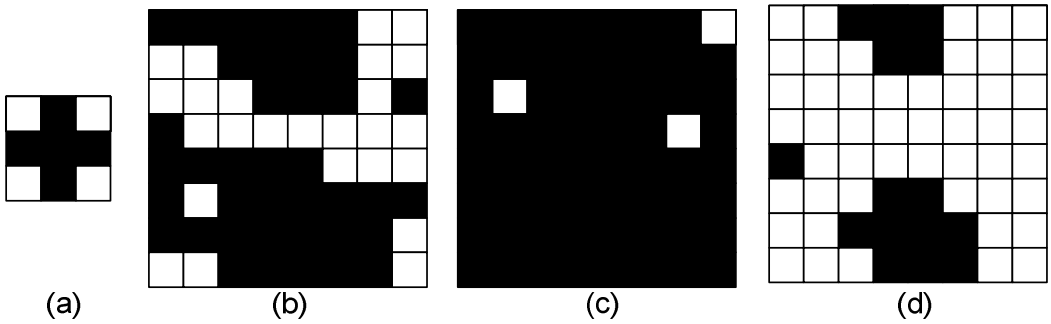


Figura 3.35 – (a) Elemento Estruturante, (b) imagem original, (c) Dilatação, (d) Erosão.

Em uma Erosão, comparam-se os valores do elemento estruturante e os valores da vizinhança de sobreposição. Caso todos os pixels ‘1’ sejam sobrepostos a pixels ‘1’, o resultado também será ‘1’. Se ao menos um dos pixels ‘1’ do elemento estruturante sobrepor-se a um pixel ‘0’, o resultado será ‘0’.

Já em uma dilatação, caso ao menos um pixel ‘1’ do elemento estruturante sobrepor-se a um pixel ‘1’, o resultado será ‘1’. Se não houver nenhuma sobreposição de um pixel ‘1’ por outro pixel ‘1’, o resultado será ‘0’.

Em aplicações práticas, combinações de dilatação e erosão são muito freqüentes, e bastante poderosas. Geralmente essas combinações são simplesmente a aplicação em seqüência desses operadores. Por exemplo, seja a imagem com um texto digitalizado e já binarizada da Figura 3.36. Nota-se a presença de diversos pontos pretos espalhados pela imagem, caracterizando algum tipo de ruído. Realizando uma erosão da imagem, resulta-se na imagem da Figura 3.37, onde o ruído não está mais presente. Entretanto, pode-se perceber que as letras do texto ficaram um pouco degradadas. Efetuando uma operação de dilatação, obtêm-se a imagem da Figura 3.38, na qual o ruído não está presente, e o texto teve seu

aspecto melhorado. Essa seqüência Erosão-Dilatação é chamada Abertura. Sua operação análoga é conhecida como Fechamento (seqüência Dilatação-Erosão).

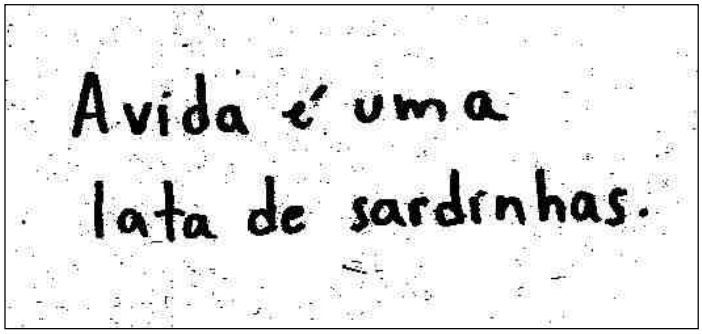


Figura 3.36 – Texto digitalizado e binarizado, com a presença de ruídos.

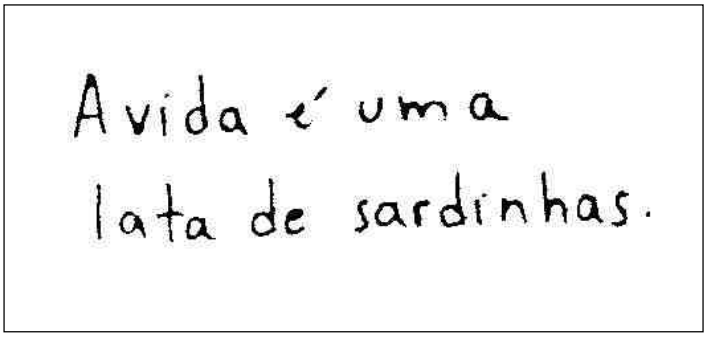


Figura 3.37 –Erosão do texto. O ruído foi eliminado, mas o texto está degradado.

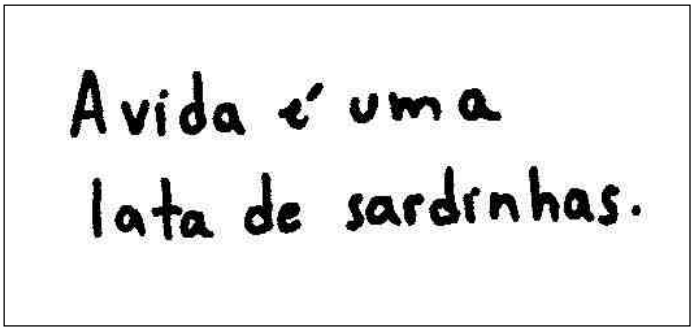


Figura 3.38 – Resultado da Dilatação do texto da Figura 3.37. O ruído foi eliminado e o texto teve seu aspecto melhorado.

As operações morfológicas binárias podem ser estendidas para imagens de intensidade. Os filtros de ordem de Máximo e de Mínimo (Tópico 3.2) representam operações de Dilatação e Erosão em imagens de intensidade, respectivamente.

3.4.1 – Arquitetura dos algoritmos

As operações de Dilatação e Erosão são classificadas como Operações de Vizinhança, pois necessitam de dados de uma vizinhança para seus cálculos. As arquiteturas propostas para essas operações são semelhantes às arquiteturas das operações de Convolução/Correlação,

e as mesmas considerações quanto aos acessos à memória podem ser feitas aqui também.

Os elementos de processamento para Dilatação e Erosão são baseados em operações lógicas simples, e seu projeto pode ser feito com base em tabelas-verdade. Considerando o elemento estruturante e a vizinhança de interesse da Figura 3.35, o cálculo da erosão e da dilatação pode ser dado como na Figura 3.39.

Erosão

k_i	f_i	e_i
0	0	1
0	1	1
1	0	0
1	1	1

Dilatação

k_i	f_i	d_i
0	0	0
0	1	0
1	0	0
1	1	1

$$e_i = \overline{k_i}.\overline{f_i} + \overline{k_i}.f_i + k_i.f_i$$

$$d_i = k_i.f_i$$

$$Erosão = e_1.e_2.e_3.e_4.e_5.e_6.e_7.e_8.e_9$$

$$Dilatação = d_1 + d_2 + d_3 + d_4 + d_5 + d_6 + d_7 + d_8 + d_9$$

Figura 3.39 – Operações de Erosão e Dilatação binárias.

Os elementos de processamento para Erosão e Dilatação utilizam dois passos para o cálculo do resultado para cada vizinhança, o primeiro é o cálculo de e_i/d_i e o segundo o cálculo final. Esses passos são seqüenciais, como os passos de multiplicação e soma das operações de Convolução/Correlação, e pode-se utilizar uma arquitetura de *Pipeline* para realizar essas operações com um fluxo de saída de 1 pixel calculado por ciclo de relógio, após o período de latência inicial.

Para as operações combinadas de Abertura e Fechamento, também se pode utilizar um *Pipeline* com as operações de Erosão e Dilatação em seqüência, sem a necessidade de armazenamento temporário de imagens intermediárias.

3.5 – Segmentação

O processo de segmentação de imagens digitais faz uma partição do conjunto de dados de entrada em estruturas relevantes para uma determinada aplicação (Pedrini,2008). As abordagens mais gerais de segmentação são baseadas na detecção de descontinuidades ou similaridades na imagem. Os métodos baseados em descontinuidades visam dividir a imagem com base nas mudanças abruptas de intensidade, enquanto que os baseados em similaridades buscam agrupar os pontos da imagem em conjuntos com valores similares de intensidade.

Neste trabalho são analisados apenas os métodos baseados na detecção de descontinuidades nas imagens.

Pode-se encontrar basicamente quatro tipos de descontinuidade em uma imagem: pontos, segmentos de retas, junções e bordas [Gon06]. O método mais comum de identificação é baseado no processo de Convolução, já descrito no tópico 3.2.

3.5.1 – Detecção de Pontos e Retas

Pontos isolados em uma imagem podem ser detectados pela aplicação da máscara da Figura 3.40(a). O pixel de interesse é considerado um ponto se a resposta da máscara for maior que um determinado limiar.

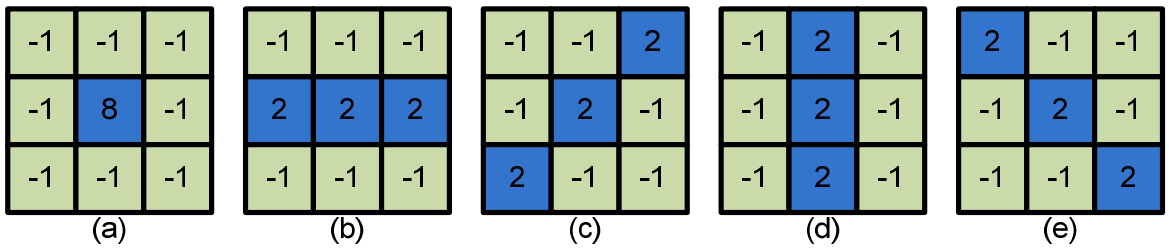


Figura 3.40 – Máscaras para detecção de (a)pontos, (b)retas horizontais, (c)retas a 45 graus, (d)retas verticais, (e)retas a 135 graus.

Segmentos de retas em ângulos determinados podem ser detectados pela aplicação das máscaras (b)horizontal, (c)45 graus, (d)vertical e (e)135 graus. A resposta da máscara é máxima quando a reta em questão passar pela linha central da máscara.

3.5.2 – Detecção de Bordas

Uma borda é o limite entre duas regiões com propriedades distintas de intensidade (Pedrini,2008). Sabe-se do Cálculo Diferencial e Integral que a derivada de uma função determina a sua taxa de variação. Uma borda significa, geralmente, uma mudança abrupta na intensidade dos pixels, ou seja, uma alta taxa de variação. Pode-se inferir, então, que o cálculo da derivada da imagem forneceria pontos de máximo, onde provavelmente encontraremos as bordas dos objetos nas imagens.

As imagens possuem duas dimensões no espaço, de modo que se deve utilizar o conceito de derivadas parciais, calculando-se as derivadas em X e em Y de modo independente, e combinando-as depois. O operador de Gradiente nos fornece as ferramentas necessárias para isso, sendo dado pelas Equações 3.4 e 3.5 (Magnitude e Direção do Gradiente,

respectivamente).

$$mag(\nabla f) = \sqrt[2]{G_x^2 + G_y^2} = \sqrt[2]{\frac{\partial f^2}{\partial x} + \frac{\partial f^2}{\partial y}} \quad 3.4$$

$$\theta(x, y) = \arctan\left(\frac{G_y}{G_x}\right) \quad 3.5$$

Algumas aproximações comumente feitas para o cálculo da magnitude do gradiente são dadas nas equações 3.6 e 3.7.

$$\nabla f \approx |G_x| + |G_y| \quad 3.6$$

$$\nabla f \approx \max(|G_x|, |G_y|) \quad 3.7$$

A Figura 3.41 mostra o resultado da utilização da Equação 3.6, com a soma das imagens das Figuras 3.12(c) e 3.12(d).

Figura 3.41 – Soma dos módulos das imagens 3.9(c) e 3.9(d).

3.5.3 – Arquiteturas dos Algoritmos

Todas as operações citadas nesta seção compõem-se de seqüências de operações já discutidas anteriormente, em conjunto com algumas operações mais simples. A análise será feita sobre o conjunto das operações.

A detecção de pontos e retas pode ser implementada por uma convolução seguida uma limiarização, sendo os pixels brancos os pontos ou retas detectados. Essa arquitetura permite a utilização de um *Pipeline* com dois passos (convolução e limiarização).

A detecção de bordas, com a utilização das Equações 3.6 e 3.7 permitiria explorar uma arquitetura do tipo MISD, já que temos duas convoluções distintas sendo aplicadas aos

mesmos dados de entrada, como mostra a Figura 3.42.

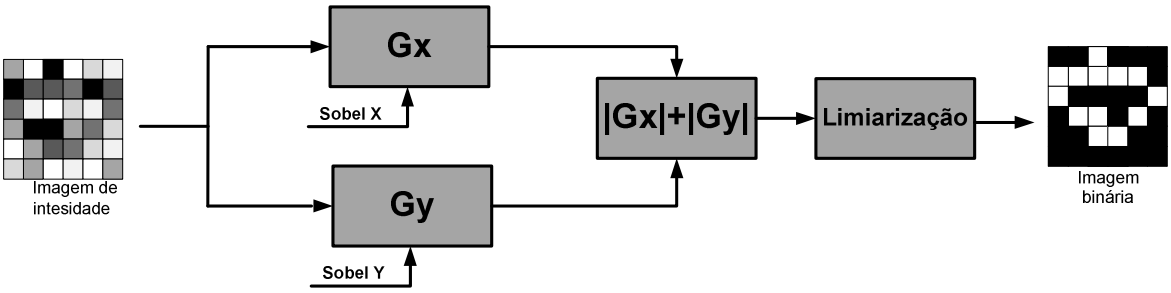


Figura 3.42 – Arquitetura para detecção de bordas.

Neste exemplo fica evidenciada a vantagem de utilização de um FPGA para a implementação dos algoritmos. Um processador comum primeiro calcularia o gradiente em *X*, armazenando-o em uma memória, calcularia em seguida o gradiente em *Y* e só então a soma e a limiarização. Na arquitetura proposta na Figura 3.42, *Gx* e *Gy* podem ser calculados paralelamente, diminuindo significativamente o tempo total de processamento.

3.6 – Conclusões do capítulo

Neste capítulo foi explicado o funcionamento dos algoritmos selecionados para implementação neste trabalho e suas aplicações. Mostrou-se como pode ser feita a partição desses algoritmos e sua transcrição em arquiteturas passíveis de implementação em hardware reconfigurável.

Em cada algoritmo tentou-se extrair as informações sobre paralelismo de dados e de instruções, com vistas à correta proposição da arquitetura mais indicada para cada tipo de processamento.

CAPÍTULO 4

4. AMBIENTE DE DESENVOLVIMENTO

Para os testes das arquiteturas propostas no Capítulo 3, foi implementada uma plataforma para captura, processamento e visualização de imagens digitais. Neste capítulo serão mostrados os equipamentos utilizados e descritos os softwares de apoio necessários ao desenvolvimento do sistema.

4.1 – Kit de desenvolvimento DE2

Todas as implementações utilizaram como base um kit de desenvolvimento para FPGAs da Terasic Inc. Esse kit utiliza um FPGA CycloneII da Altera Corp., e provê uma grande variedade de interfaces de entrada e saída para diversos periféricos. As especificações do kit são as que seguem (Figura 4.1):

- FPGA Altera CycloneII 2C35
- Dispositivo de configuração Altera EPCS16
- USB Blaster para programação e controle de dispositivos
- 4 *pushbuttons*
- 18 *switches*
- 18 *leds* vermelhos
- 9 *leds* verdes
- Osciladores de 50MHz e 27MHz
- Interface de entrada e saída de áudio
- Porta de saída VGA com conversor DA de 10 bits
- Decodificador NTSC/PAL, para entrada de vídeo

- Controlador Ethernet 10/100 Mbits/s
- Controlador USB Host/Slave
- Interface serial RS232
- Interface PS2
- Interface infravermelho
- Duas portas de 40 pinos de entrada/saída de uso geral
- Memória SRAM de 512KBytes
- Memória SDRAM de 8MBytes
- Memória Flash de 4MBytes

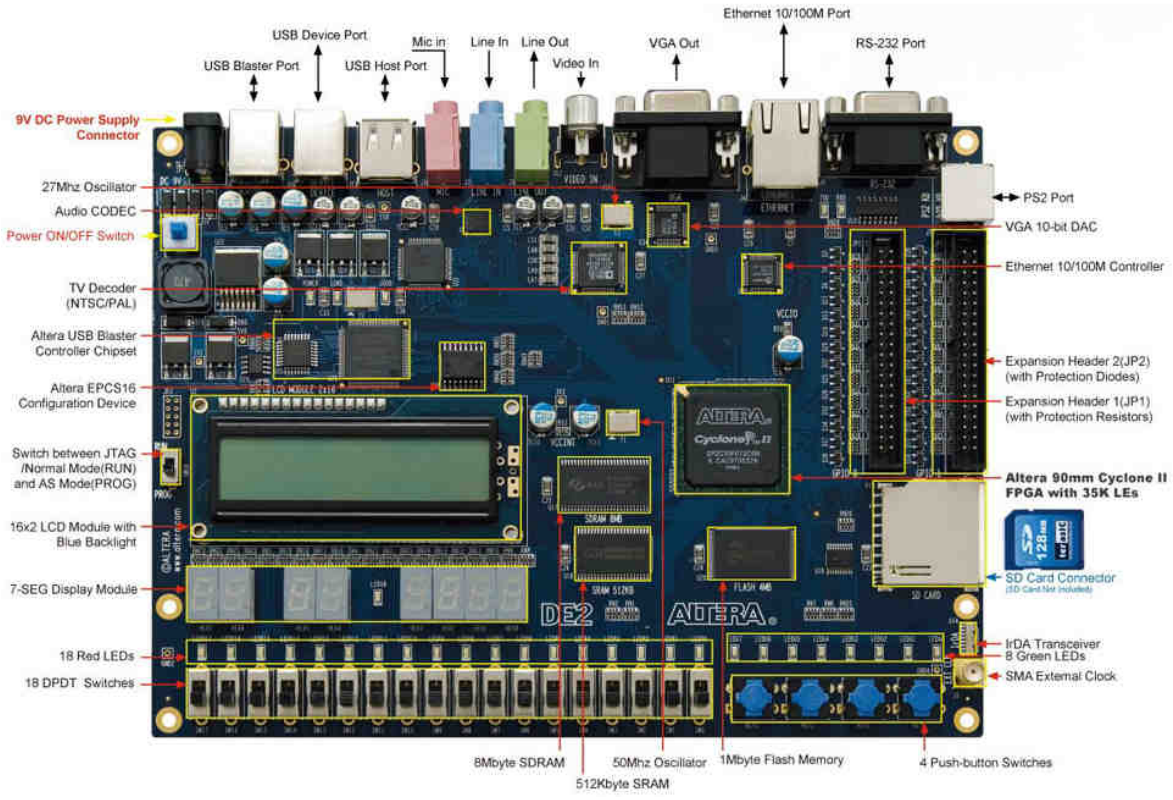


Figura 4.1 – Kit de desenvolvimento DE2, Terasic Inc., contendo diversos periféricos.

4.2 – FPGA Cyclone II

Os FPGAS da família Cyclone II da Altera possuem algumas características internas de interesse às implementações feitas neste projeto. A Tabela 4.1 mostra algumas dessas

características, sendo o modelo utilizado no kit DE2 é o EP2C35, mostrado na 5ª. coluna.

Tabela 4.1 – Algumas características de interesse na família Cyclone II.

Feature	EP2C5 *(2)*	EP2C8 *(2)*	EP2C15 *(1)*	EP2C20 *(2)*	EP2C35	EP2C50	EP2C70
LEs	4,608	8,256	14,448	18,752	33,216	50,528	68,416
M4K RAM blocks (4 Kbits plus 512 parity bits	26	36	52	52	105	129	250
Total RAM bits	119,808	165,888	239,616	239,616	483,840	594,432	1,152,000
Embedded multipliers *(3)*	13	18	26	26	35	86	150
PLLs	2	2	4	4	4	4	4

Os FPGAS dessa família possuem arquiteturas baseadas em linhas e colunas para implementação das lógicas. Essas linhas e colunas são compostas de *logic array blocks* (LABs), *embedded memory blocks* e *embedded multipliers*. A Figura 4.2 mostra a organização interna de um FPGA da família CycloneII.

Nos tópicos que seguem são apresentadas algumas características mais relevantes dos blocos internos da CycloneII.

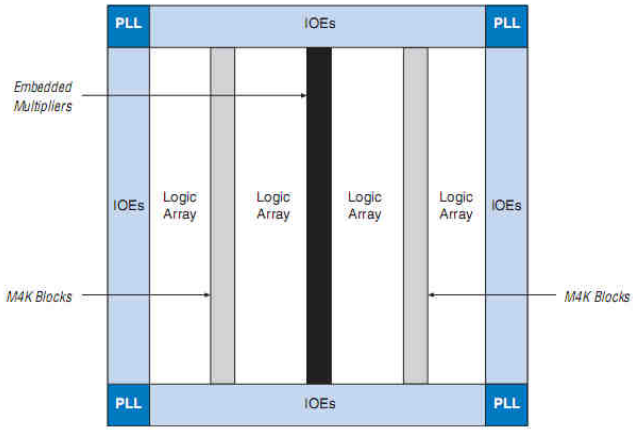


Figura 4.2 – Diagrama de organização interna de um FPGA da família Cyclone II.

4.2.1 – *Logic Array Blocks/Logic Elements*

Cada LAB possui 16 elementos lógicos, que são pequenas unidades lógicas para implementação de funções lógicas de forma eficiente. Essas unidades possuem as seguintes características:

- *Look-up table*: é uma unidade de quatro entradas, capaz de implementar qualquer

função lógica de 4 variáveis.

- Um registrador programável: cada registrado possui sinais de *clear*, *clock* e dados, podendo ser configurado para funcionar como um *flip-flop* tipo D, T, JK ou SR, permitindo uma grande gama de funcionalidades.

Com esses elementos lógicos são implementadas funções como contadores, somadores, subtratores, funções aritméticas, acumuladores e comparadores, além de ser possível também a implementação de pequenas memórias.

4.2.2 – *Embedded Memory*

Os blocos de memória embarcada na CycloneII consistem de colunas de blocos de memória do tipo M4K. Esse tipo de memória inclui registradores de entrada que sincronizam a escrita, e registradores de saída que permitem trabalhar como em um *pipeline*, melhorando o desempenho do sistema.

Cada bloco M4K pode implementar memórias de diversos tipos, com ou sem bits de paridade, possuindo as características da Tabela 4.2.

Tabela 4.2 – Características dos blocos M4K de memória.

Característica	Descrição
Desempenho máximo	250 MHz
Total de bits de RAM por bloco	4608
Configurações possíveis	4K x 1, 2K x 2, 1K x 4, 512 x 8, 512 x 9, 256 x 16, 256 x 18, 128 x 32, 128 x 36
Bits de paridade	Um bit de paridade para cada byte. O bit de paridade, junto com alguma lógica configurada, pode implementar a checagem de paridade para detecção de erros e garantia da integridade dos dados.

Tais blocos ainda podem ser configurados para operarem de forma otimizada em diversos modos, descritos na Tabela 4.3.

Tabela 4.3 – Modos de memória M4K.

Modo de memória	Descrição
Single-port memory	Não permite leitura e escrita simultâneos.
Simple dual-port memory	Permite leitura e escrita simultâneos.
Simple dual-port with mixed width	Permite diferentes comprimentos de dados de leitura e escrita.

True dual-port memory	Permite qualquer combinação de operações *dual-port*: duas leituras, duas escritas, uma escrita e uma leitura, com freqüências distintas.
True dual-port with mixed width	É o modo *True dual-port*, com a possibilidade de diferentes comprimentos de dados de leitura e escrita.
Embedded shift register	Implementação de registradores de deslocamento. Os dados são escritos em cada endereço na borda de descida do *clock* e lidos de cada endereço na borda de subida.
ROM	A memória é pré-carregada utilizando-se um arquivo tipo .mif, durante a configuração do dispositivo.
FIFO *buffers*	Implementação de FIFO com *clock* único ou duplo (leitura e escrita).

Cada multiplicador embarcado pode implementar duas multiplicações de 9x9 bits ou uma multiplicação de 18x18 bits, a freqüências de até 250 MHz. São arranjados em colunas no FPGA.

4.2.3 – *Embedded Multipliers*

As FPGAs da família CycloneII possuem multiplicadores otimizados para funções de processamento digital de sinais que necessitem de muitas multiplicações, como filtros FIR, a FFT, DCT, entre outras. Cada bloco multiplicador pode ser utilizado de dois modos diferentes, dependendo da necessidade da aplicação, descritos na Tabela 4.4.

Tabela 4.4 – Modos de operação dos multiplicadores embarcados.

Modo de operação	Descrição
18-bit *Multiplier*	Uma multiplicação de dois operandos de 18 bits.
9-bit *Multiplier*	Duas multiplicações de dois operandos de 9 bits cada.

4.3 – A Câmera Digital

Para captura das imagens foi utilizada uma câmera digital de 5 Mega Pixel de resolução real, com conexão própria para o Kit DE2, modelo TRDB_D5M (Terasic Inc.), Figura 4.3. Algumas características dessa câmera são:

- Controle de tempo de exposição do sensor, configurável via registradores

- Sensor CMOS de 5 Mega Pixel, com resolução configurável (máximo de 2592 x 1944 pixels ativos) e conversor AD de 12 bits por canal de cor.
- Barramento de dados de saída paralelo, com sinais separados R, G e B.
- Taxa de aquisição variável, de acordo com parâmetros de resolução e tempo de exposição do sensor, de acordo com as Tabelas 4.5 e 4.6.

Tabela 4.5 – Resoluções-Padrão

Resolution	Frame Rate	Sub-sampling Mode	Column_Size (R0x04)	Row_Size (R0x03)	Shutter_Width_Lower (R0x09)	Row_Bin (R0x22 [5:4])	Row_Skip (R0x22 [2:0])	Column_Bin (R0x23 [5:4])	Column_Skip (R0x23 [2:0])
2592 x 1944 (Full Resolution)	15.15	N/A	2591	1943	<1943	0	0	0	0
2,048 x 1,536 QXGA	23	N/A	2047	1535	<1535	0	0	0	0
1,600 x 1,200 UXGA	35.2	N/A	1599	1199	<1199	0	0	0	0
1,280 x 1,024 SXGA	48	N/A	1279	1023	<1023	0	0	0	0
	48	skipping	2559	2047		0	1	0	1
	40.1	binning	2559	2047		1	1	1	1
1,024 x 768 XGA	73.4	N/A	1023	767	<767	0	0	0	0
	73.4	skipping	2047	1535		0	1	0	1
	59.7	binning	2047	1535		1	1	1	1
800 x 600 SVGA	107.7	N/A	799	599	<599	0	0	0	0
	107.7	skipping	1599	1199		0	1	0	1
	85.2	binning	1599	1199		1	1	1	1
640 x 480 VGA	150	N/A	639	479	<479	0	0	0	0
	150	skipping	2559	1919		0	3	0	3
	77.4	binning	2559	1919		3	3	3	3

Tabela 4.6 – Resoluções *Wide-Screen*

Resolution	Frame Rate	Sub-sampling Mode	Column_Size (R0x04)	Row_Size (R0x03)	Shutter_Width_Lower (R0x09)	Row_Bin (R0x22 [5:4])	Row_Skip (R0x22 [2:0])	Column_Bin (R0x23 [5:4])	Column_Skip (R0x23 [2:0])
1,920 x 1,080 HDTV	34.1	N/A	1919	1079	<1079	0	0	0	0
1,280 x 720 HDTV	67.6	N/A	1279	719	<719	0	0	0	0
	67.6	skipping	2559	1439	<719	0	1	0	1
	56.4	binning	2559	1439	<719	1	1	1	1

Figura 4.3 – Câmera modelo TRDB_D5M (Terasic Inc.).

4.4 – O *Display* LCD

A visualização das imagens, processadas ou não, foi feita utilizando um *Display* LCD, modelo TRDB_LTM, da Terasic Inc. (Figura 4.4). Esse *display* possui uma interface para conexão no Kit DE2, e as seguintes características:

- Interface de dados paralela RGB de 24 bits
- Correção de brilho e contraste configuráveis
- Resolução de 800 x 480 x RGB pixels

Figura 4.4 – Display LCD, modelo TRDB_LTM (Terasic Inc.).

4.5 – O software Altera Quartus II

Foi utilizada a versão 9.1 da ferramenta EDA Quartus II, da Altera Corp., a qual permite a criação e simulação de projetos, além da síntese e programação dos dispositivos FPGAs. As Figuras 4.5 e 4.6 mostram capturas de telas de codificação e simulação, respectivamente.

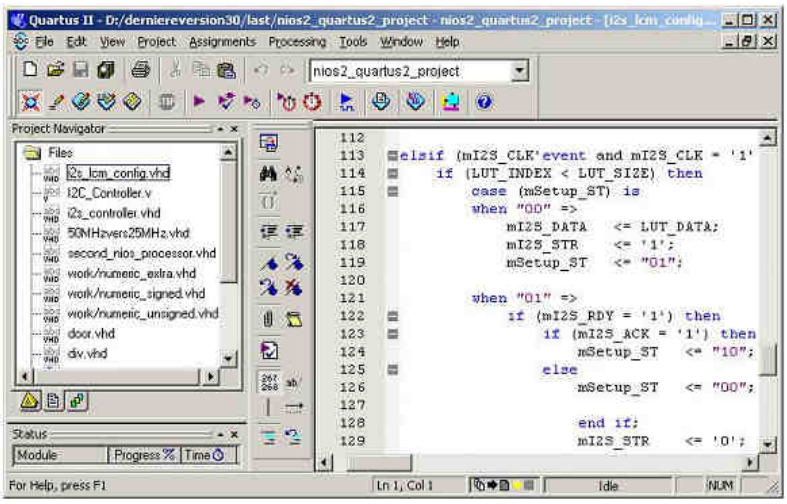


Figura 4.5 – Tela do Quartus II, mostrando um exemplo de código VHDL.

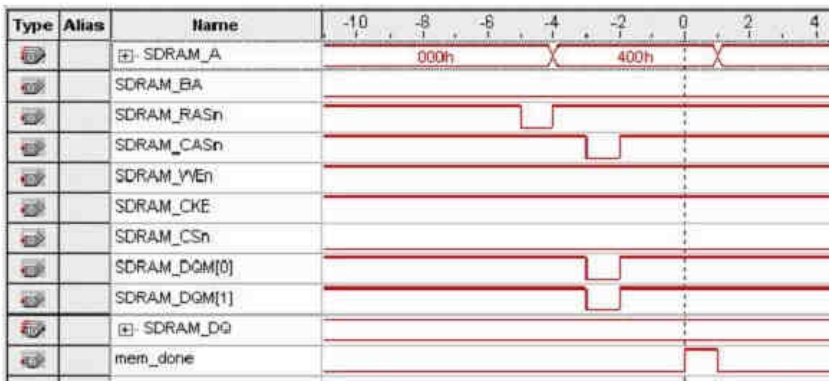


Figura 4.6 – Tela do Quartus II, mostrando um exemplo de simulação.

O Quartus II suporta o desenvolvimento em três linguagens diferentes de descrição de hardware:

- VHDL – *Very High Speed Integrated Circuits Hardware Description Language*. É uma linguagem padrão ISO para a descrição de hardware.
- Verilog *Hardware Description Language*. Linguagem desenvoldia pela *Cadence Systems,* empresa especializada em ferramentas CAD para projetos de circuitos.
- AHDL – Altera *Hardware Description Language*. Linguagem desenvolvida pela própria Altera.

Existe ainda a possibilidade de criarem-se projetos utilizando diagramas de blocos representativos dos códigos descritos nas *HDLs*. A Figura 4.7 mostra um exemplo no Quartus II.

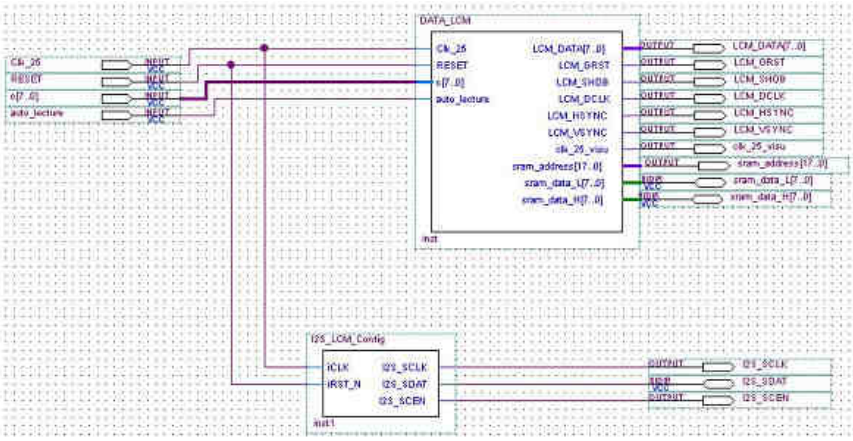


Figura 4.7 – Tela do Quartus II, exemplificando um projeto utilizando Diagramas de Blocos.

4.6 – O sistema Completo

A Figura 4.8 mostra o sistema completo implementado, composto do Kit DE2, da Câmera Digital e do Display.

Figura 4.8 – Sistema completo utilizado: Kit DE2, Câmera Digital e *Display* LCD.

4.7 – Conclusões do capítulo

O sistema de hardware utilizado apresenta algumas limitações que influenciaram na escolha das arquiteturas implementadas. Com o intuito de utilizar todo o potencial de

desenvolvimento disponível, houve uma preocupação em desenvolver as arquiteturas de processamento de modo que pudessem ser utilizadas de forma direta nessa plataforma. A câmera utilizada fornece as imagens pixel a pixel, com os canais R, G e B disponíveis de forma paralela, utilizando um processo de varredura da matriz fotossensível. O display LCD, por sua vez, recebe os dados também pixel a pixel, em um processo de varredura do display e com os canais de cores separados.

A câmera fornece ainda um sinal cuja borda de subida indica que um novo pixel foi disponibilizado em suas saídas. Esse sinal foi utilizado para sincronização de todos os módulos do sistema, tanto os de processamento de imagem quanto os necessários para visualização das imagens no display.

As memórias disponíveis no Kit DE2, como já previsto, não possuem portas de dados em número suficiente para a disponibilização simultânea de todos os pixels. Como a câmera fornece os dados um a um, de modo serial, evitou-se a utilização de memórias para armazenamento das imagens capturadas, trabalhando-se sobre o *stream* de pixels proveniente da câmera de modo direto.

Assim sendo, as arquiteturas implementadas seguiram a idéia discutida no tópico 3.1.1, Figura 3.3, com um fluxo serial de dados percorrendo a arquitetura.

CAPÍTULO 5

5 - IMPLEMENTAÇÃO DAS ARQUITETURAS

Neste capítulo será descrita a implementação das diferentes arquiteturas de processamento de imagens sugeridas no Capítulo 3. Detalhes da transcrição dos algoritmos em linguagem VHDL serão fornecidos, de modo a mostrar as diferentes formas de implementação, com ênfase ao desenvolvimento utilizando o aparato citado no Capítulo 4.

5.1 – Redução de cores

O foco deste trabalho foi no processamento de imagens de intensidade (nível de cinza) apenas. A câmera fornece três canais de cores de 12 bits, porém, o sistema de controle de captura implementado pelo fabricante prevê a utilização de 8 bits apenas. Desse modo, todo o processamento foi implementado com base na premissa de ter-se 8 bits por canal de cor. Foi implementado um código redutor de cores, transformando os três canais de cores em apenas um canal de 8 bits (escala de cinza).

Existem diversos métodos para a determinação do valor apropriado de nível de cinza, quando da conversão de imagens coloridas. Neste trabalho, o bloco redutor calcula a média simples dos canais de cor (R,G,B), entregando aos blocos subseqüentes um pixel apenas (Figura 5.1(a)). O circuito é síncrono, sendo o sinal de *clock* oriundo da câmera.

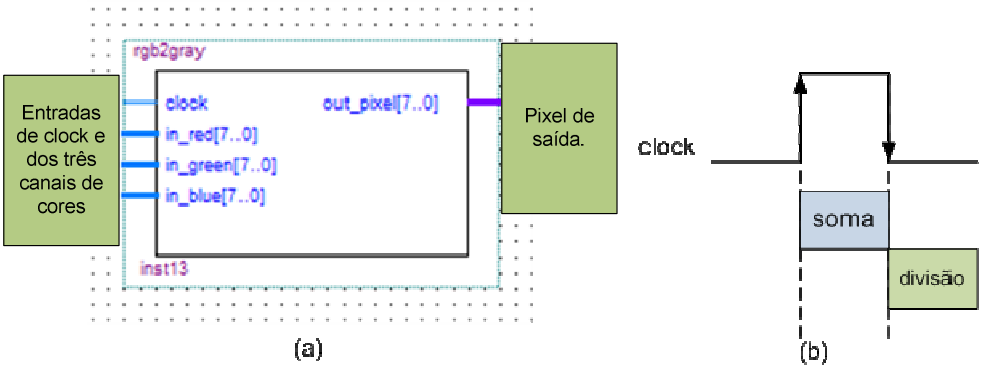


Figura 5.1 – Bloco redutor de cores, RGB para escala de cinza.

Os cálculos para a redução de cores são feitos em um único ciclo de relógio. Para isso o algoritmo de conversão foi dividido em dois processos, sendo o primeiro sensível à borda

de subida do clock e responsável pela soma dos três canais, e o segundo sensível à borda de descida do clock, efetuando a divisão, Figura 5.1(b). Esse bloco de conversão possui uma estrutura bastante simples. O Quartus II fornece dados de síntese desse bloco, mostrados na Tabela 5.1. Tais dados representam a utilização de recursos de cada bloco de processamento implementado.

Tabela 5.1 – Dados de síntese do bloco de conversão de cores.

Elementos Lógicos	Memory Bits	M4Ks	DSP 9x9	DSP 18x18
95	0	0	0	0

5.2 – Binarização

O bloco de binarização de imagens possui duas entradas e uma saída. As entradas são o *clock* de sincronismo e o pixel a ser binarizado, e a saída é o pixel já binarizado, Figura 5.2(a). Esse circuito opera em apenas um ciclo de clock, efetuando a comparação e a atribuição da saída na borda de subida do clock, Figura 5.2(b). A Tabela 5.2 mostra os dados de síntese desse bloco.

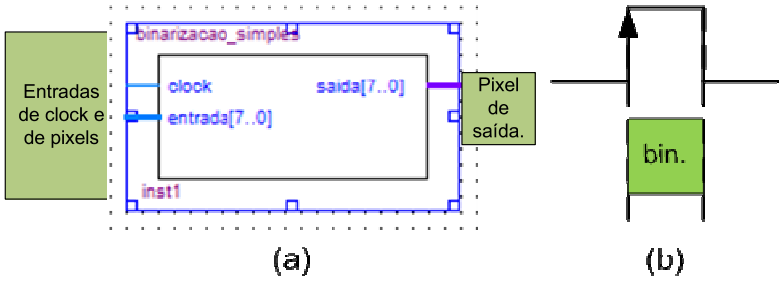


Figura 5.2 – Bloco de binarização.

Tabela 5.2 – Dados de síntese do bloco de binarização.

Elementos Lógicos	Memory Bits	M4Ks	DSP9x9	DSP18x18
3	0	0	0	0

5.3 – Operações de *Windowing*

As arquiteturas propostas para os algoritmos que envolvem o processamento de vizinhanças utilizam primeiramente a operação de *windowing*, seguida dos blocos específicos operações de cada algoritmo. Antes de mostrar os blocos específicos, será

mostrada a arquitetura utilizada para a operação de *windowing*, mostrando suas características próprias e suas interfaces com os outros blocos.

Conforme já explicado no Capítulo 3, para o processamento de uma vizinhança necessitamos ter todos os dados da vizinhança disponíveis. Neste trabalho, o tamanho de vizinhança considerado foi de 3x3 pixels, mas a extrapolação é simples para outros tamanhos de vizinhança. Como a câmera envia os dados da imagem de acordo com sua captura, os pixels formam um fluxo que se origina no canto superior esquerdo da imagem e vai até o canto inferior direito da imagem. Os pixels são enviados linha a linha, de cima para baixo, e coluna a coluna dentro de cada linha.

Desse modo, a arquitetura básica para a operação de *windowing* consiste em parte da arquitetura de convolução da Figura 3.19, caracterizada por um arranjo sistólico (Figura 5.3).

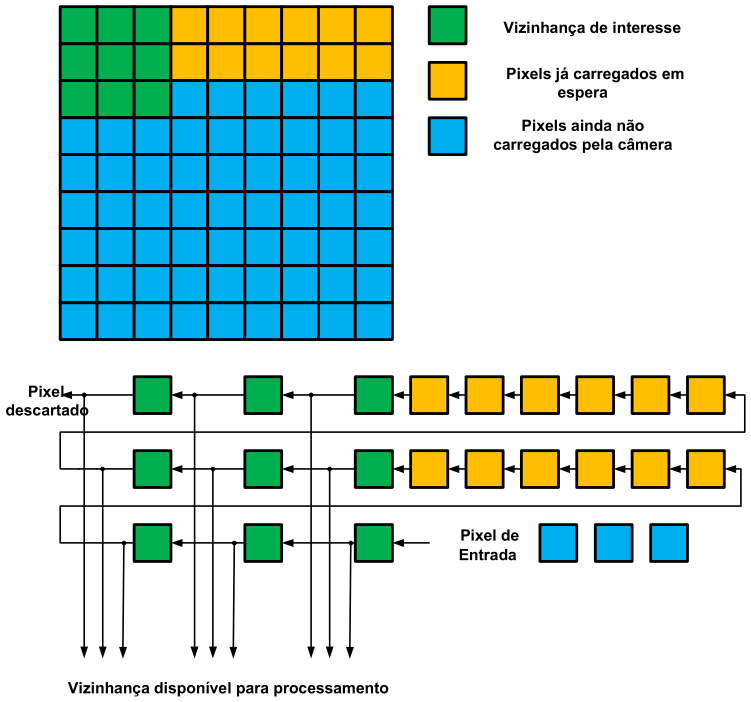


Figura 5.3 – Arquitetura para operação de *Windowing*.

Essa arquitetura permite que, ao carregar-se um novo pixel, toda a *vizinhança* necessária ao processamento fique disponível simultaneamente. Há uma latência inicial, pois é necessário que todo o registrador de deslocamento esteja preenchido por pixels, antes de processar-se a primeira vizinhança. No caso da Figura 5.3, a latência é de 21 ciclos de carregamento de pixels. Para uma imagem de dimensões $M \times N$, e uma vizinhança desejada

$L \times P$, a latência inicial pode ser calculada pela Equação 5.1.

$$latência = (P - 1).M + L \qquad 5.1$$

Após esse período inicial de latência, a arquitetura fornece uma nova vizinhança a cada novo carregamento.A implementação dessa arquitetura é baseada em um registrador de deslocamento de comprimento igual à latência. Um registrador de deslocamento é um circuito que opera simplesmente em um esquema FIFO (*First In, First Out*), ou seja, o primeiro pixel a entrar é o primeiro pixel a sair, no caso, a ser eliminado.

Quando da codificação e testes dessa arquitetura, foi verificado que para imagens a partir de 100 pixels de largura, aproximadamente, a simples implementação utilizando os elementos lógicos do FPGA não seria possível, pois o consumo desses recursos seria muito elevado. Duas alternativas poderiam ser exploradas: (a) memórias externas e (b) memória interna do FPGA. Como visto no Capítulo 4, as FPGAs da família CycloneII oferecem a capacidade de configuração de sua memória interna como registradores de deslocamento. Nos testes realizados, essa configuração funcionou adequadamente, não tendo sido testada a utilização das memórias externas nas operações de *windowing*.

Para configurar um registrador de deslocamento na FPGA, o Quartus II oferece uma ferramenta chamada *MegaWizard Plugin Manager* que permite escolher qual o tipo de modo de operação desejado (*shift-register*, FIFO, *dual-mode*, etc) e outros parâmetros como o comprimento de palavra utilizado. A Figura 5.4 mostra a primeira tela de configuração do *MegaWizard*, com a opção escolhida.

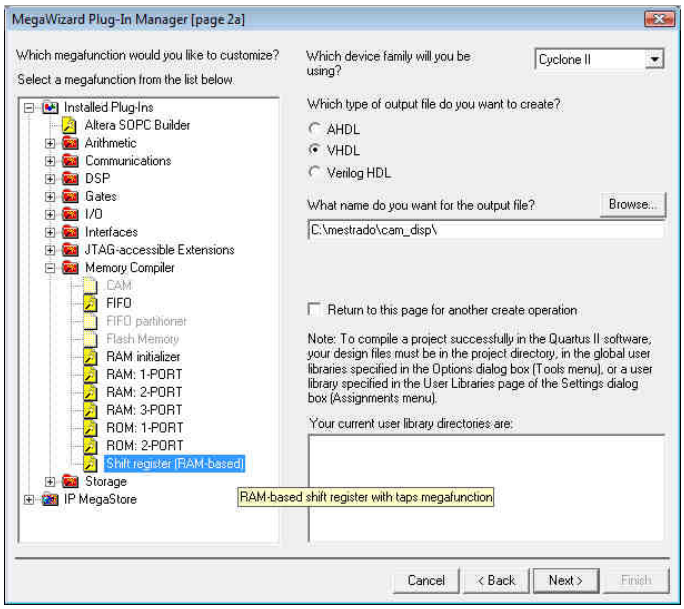


Figura 5.4 – Primeira tela de configuração do *MegaWizard Plugin Manager*.

O *display* utilizado neste projeto possui largura visível de 800 pixels, então, conforme a equação 5.1, o comprimento do registrador de deslocamento, para uma máscara 3×3, seria de 1603 elementos. Porém, tal registrador não permitiria o acesso aos pixels desejados da vizinhança, então a arquitetura foi dividida em três partes, conforme a Figura 5.5.

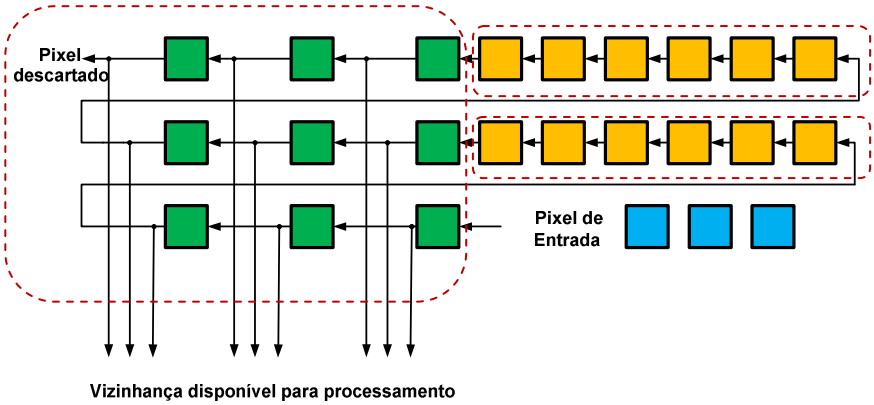


Figura 5.5 – Divisão da arquitetura em três partes.

Desse modo, a implementação da arquitetura ficou composta de dois registradores de deslocamento (na cor laranja) e um bloco com a função de controlar o carregamento e disponibilização dos pixels (na cor verde).

Outro fator importante está na configuração dos parâmetros dos registradores de deslocamento. A princípio, seu comprimento seria de 797 pixels (800 do total subtraindo 3 da máscara), porém não é possível implementar esse registrador com os exatos 797 pixels, sendo feita uma aproximação para 768 pixels (valor mais próximo disponível na ferramenta). Desse modo, para contabilizar os 800 pixels de uma linha, o bloco de cor verde passa a ter 32 pixels, e não mais três (verificar que: 797 – 768 + 3 = 32). A Figura 5.6 mostra como ficou a divisão dessa arquitetura.

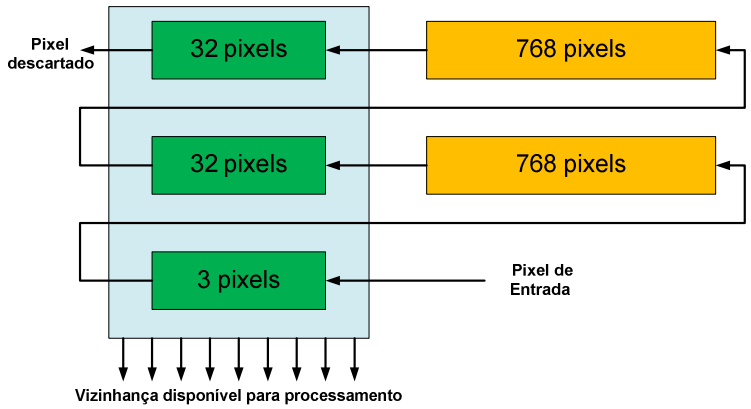


Figura 5.6 – Divisão da arquitetura em três blocos.

Após a escolha do tipo de arranjo de memória desejada (Figura 5.4), devem-se escolher os parâmetros corretos para configuração do registrador de deslocamento. A Figura 5.7 mostra as configurações selecionadas para cada um dos dois registradores necessários.

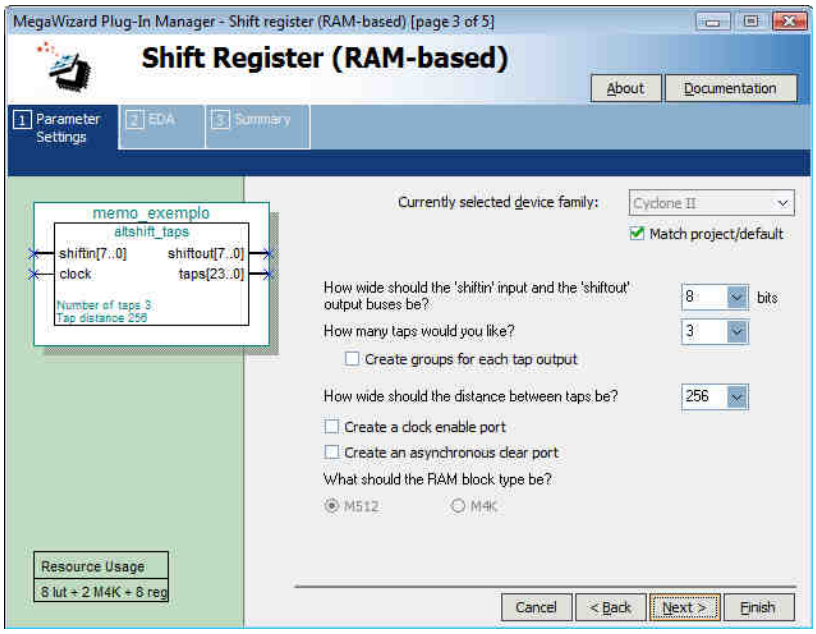


Figura 5.7 – Configurações selecionadas para um registrador de deslocamento.

Um *tap* é uma saída intermediária no registrador, de modo a permitir a leitura de dados que estejam no meio do registrador. A posição dos *taps*, porém, não é arbitrária, sendo eqüidistantes um do outro. A entrada de dados é o primeiro *tap* e a saída não é um *tap*. Observando as configurações da Figura 5.7, vemos a escolha de três *taps*, com distância de 256 pixels entre cada *tap*. Desse modo temos exatamente 768 pixels no registrador de deslocamento. A Figura 5.8 mostra o bloco de registrador de deslocamento implementado (verificar que: $3 \times 256 = 768$).

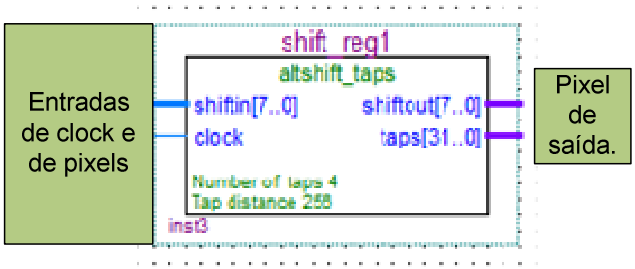


Figura 5.8 – Bloco registrador de deslocamento implementado.

Será mostrada agora como foi feita a implementação do terceiro bloco, com registradores de 32 pixels. Nesse bloco, conforme mostra a Figura 5.6, existem três registradores de deslocamento, sendo dois de 32 pixels e um de apenas três pixels. Esse bloco possui ainda quatro entradas e 10 saídas. A Figura 5.9 mostra o bloco implementado.

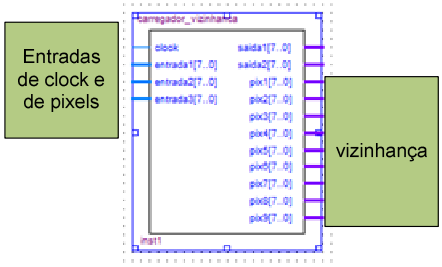
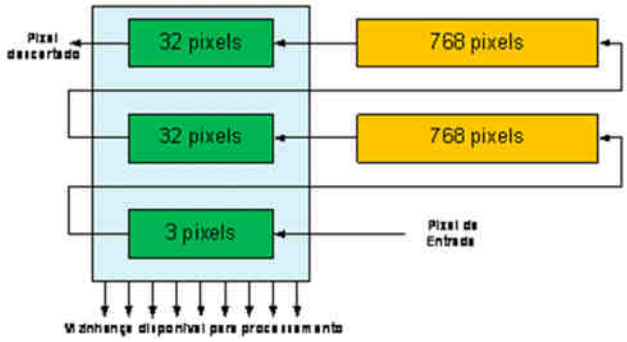
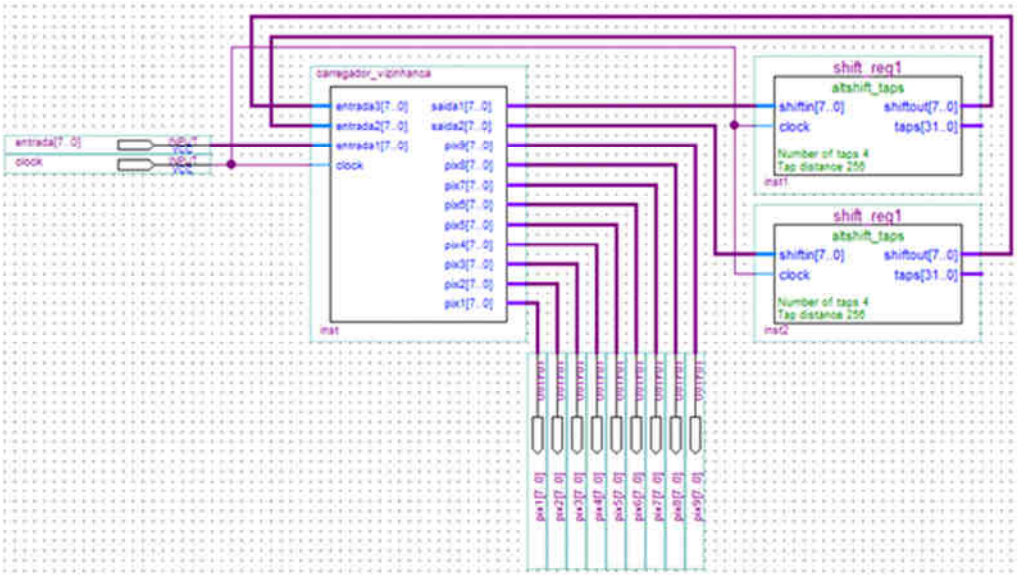


Figura 5.9 – Bloco carregador de vizinhança.

Esse bloco foi implementado para, juntamente com os registradores de deslocamento, disponibilizar uma nova vizinhança a cada ciclo de relógio.



(a)



(b)

Figura 5.10 – Bloco completo para disponibilização de uma vizinhança 3x3. (a) arquitetura

proposta. (b) implementação da arquitetura no Quartus II.

A Figura 5.10 mostra o conjunto completo implementado, juntamente com uma cópia da Figura 5.6, para efeitos de comparação. Nota-se a grande similaridade entre a arquitetura proposta e a efetivamente implementada. O bloco carregador de vizinhança, apesar de bastante simples, é mais bem entendido pela análise de seu código VHDL, o que será feito agora, trecho a trecho. A Figura 5.11 mostra a declaração de portas desse bloco. As linhas de 9 a 11 contêm as chamadas às bibliotecas de referência para o projeto. Nas linhas de 16 a 30 são declaradas as portas de IO, seu tipo e largura de dados.

```
 9   library IEEE;
10   use ieee.std_logic_1164.all;
11   use ieee.numeric_std.all;
12
13
14   entity carregador_vizinhanca is
15   port    (
16               entrada3: in unsigned(7 downto 0);
17               entrada2: in unsigned(7 downto 0);
18               entrada1: in unsigned(7 downto 0);
19               saida1: out unsigned(7 downto 0);
20               saida2: out unsigned(7 downto 0);
21               clock : in std_logic;
22               pix9: out unsigned(7 downto 0);
23               pix8: out unsigned(7 downto 0);
24               pix7: out unsigned(7 downto 0);
25               pix6: out unsigned(7 downto 0);
26               pix5: out unsigned(7 downto 0);
27               pix4: out unsigned(7 downto 0);
28               pix3: out unsigned(7 downto 0);
29               pix2: out unsigned(7 downto 0);
30               pix1: out unsigned(7 downto 0)
31           );
32   end carregador_vizinhanca;
```

Figura 5.11 – Declaração de portas do bloco carregador de vizinhança.

O trecho de código da Figura 5.12 é responsável pela declaração dos sinais internos ao bloco, assim como das constantes utilizadas e dos tipos de dados definidos.

```
33
34   architecture Behavioral of carregador_vizinhanca is
35
36       constant comp_reg : integer:=32;
37
38       type arranjo_memoria is array (natural range<>) of unsigned(7 downto 0);
39
40       signal linha1   :   arranjo_memoria(1 to comp_reg):=((others=> to_unsigned(0,8)));
41       signal temp1    :   arranjo_memoria(1 to comp_reg):=((others=> to_unsigned(0,8)));
42
43       signal linha2   :   arranjo_memoria(1 to comp_reg):=((others=> to_unsigned(0,8)));
44       signal temp2    :   arranjo_memoria(1 to comp_reg):=((others=> to_unsigned(0,8)));
45
46       signal linha3   :   arranjo_memoria(1 to 3):=((others=> to_unsigned(0,8)));
47       signal temp3    :   arranjo_memoria(1 to 3):=((others=> to_unsigned(0,8)));
48
```

Figura 5.12 – Declaração dos sinais, constantes e tipos de dados utilizados.

Na linha 36 é declarada a constante *comp_reg* (comprimento dos registradores de deslocamento internos), com valor 32. Na linha 38 é declarado um tipo de dado chamado

arranjo_memoria, como um *array* de vetores de 8 bits. As linhas seguintes declaram seis sinais de tipo *arranjo_memoria*, com comprimento igual a *comp_reg*. Esses sinais serão utilizados como registradores de deslocamento, recebendo os sugestivos nomes de *linha1, linha2* e *linha3*. Os registradores temporários *temp1, temp2* e *temp3* terão sua utilização explicada mais adiante.

A Figura 5.13 mostra o processo responsável pelo deslocamento dos registros e o carregamento de um novo pixel.

```
49  begin
50  process (clock)
51      begin
52          if (clock'event and clock = '1') then
53              temp1(2 to (comp_reg))<=linha1(1 to (comp_reg-1));
54              temp1(1)<=(entrada1);
55
56              temp2(2 to (comp_reg))<=linha2(1 to (comp_reg-1));
57              temp2(1)<=(entrada2);
58
59              temp3(2 to (3))<=linha3(1 to (3-1));
60              temp3(1)<=(entrada3);
61          elsif(clock'event and clock = '0')then
62              linha1<=temp1;
63              linha2<=temp2;
64              linha3<=temp3;
65          end if;
66  end process;
67
```

Figura 5.13 – Processo de deslocamento e carregamento de novo pixel.

A linha 50 indica que o processo é sensível ao sinal de entrada *clock.* A linha 52 verifica a borda de subida do sinal *clock*, e a linha 61 verifica a borda de descida desse sinal. Os trechos de código da linhas 53-54, 56-57 e 59-60 funcionam de forma equivalente, então será explicado o funcionamento apenas das linhas 53-54.

A linha 53 codifica o procedimento mostrado na Figura 5.14(a), em que os dados do registrador *linha1* são transferidos parcialmente para o registrador *temp1*. Simultaneamente, na linha 54, um novo pixel é carregado da entrada.

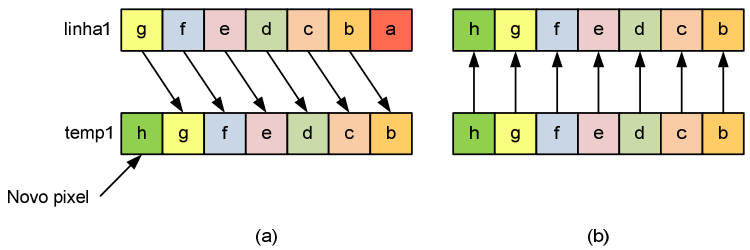


Figura 5.14 – Processo de deslocamento e carregamento de novo pixel.

Os trechos das linhas 62, 63 e 64 também são equivalentes. A linha 62 codifica o

procedimento da Figura 5.14(b), em que é copiado o conteúdo completo do registrador *temp1* para o registrador *linha1*. Esse processo permite que, em um único ciclo de subida e descida do sinal *clock,* os pixels sejam deslocados e um novo pixel seja carregado.

A Figura 5.15 mostra o trecho de código do processo para atribuição dos valores às saídas intermediárias do bloco, que são ligadas aos registradores de deslocamento (Figura 5.8).

```
67
68  process (clock)
69      begin
70          if(clock'event and clock='0')then
71              saida1<=(linha1(comp_reg));
72              saida2<=(linha2(comp_reg));
73          end if;
74  end process;
75
```

Figura 5.15 – Processo de atribuição dos valores às saídas intermediárias.

Esse processo, como codificado na linha 70, é sensível à borda de descida de *clock*. Desse modo, simultaneamente ao carregamento de valores nas posições indicadas dos registradores *linha1* e *linha2,* os valores são atribuídos às portas *saida1* e *saida2,* que são ligadas aos registradores de deslocamento implementados com os blocos M4K da FPGA (Figura 5.10).

A Figura 5.16 mostra a atribuição de valores dos registradores *linha1*, *linha2* e *linha3* às saídas do bloco. As saídas *pix* representam a vizinhança disponibilizada para processamento em blocos subseqüentes.

```
75
76  process (clock)
77      begin
78          if(clock'event and clock='1')then
79              pix1<=linha3(3);
80              pix2<=linha3(2);
81              pix3<=linha3(1);
82
83              pix4<=linha2(3);
84              pix5<=linha2(2);
85              pix6<=linha2(1);
86
87              pix7<=linha1(3);
88              pix8<=linha1(2);
89              pix9<=linha1(1);
90
91          elsif(clock'event and clock='0')then
92
93          end if;
94  end process;
95
96
97  end Behavioral;
```

Figura 5.16 – Atribuição de valores às saídas do bloco.

O circuito completo de disponibilização de uma vizinhança pode ser agrupado em um

conjunto único, encapsulado dentro de apenas bloco. A Figura 5.17 mostra o aspecto desse bloco. A Tabela 5.3 mostra os dados de síntese do Quartus II para o bloco completo de disponibilização de vizinhança 3x3.

Tabela 5.3 – Dados de síntese do circuito de disponibilização da vizinhança.

Bloco	Elementos Lógicos	Memory Bits	M4Ks	DSP 9x9	DSP 18x18
Disponibilizador de Vizinhança (Fig. 5.10/5.17)	1113	16256	4	0	0
Registrador de deslocamento (cada, Fig.5.8)	12	8128	2	0	0
Carregador de Vizinhança (Fig.5.9)	1089	0	0	0	0

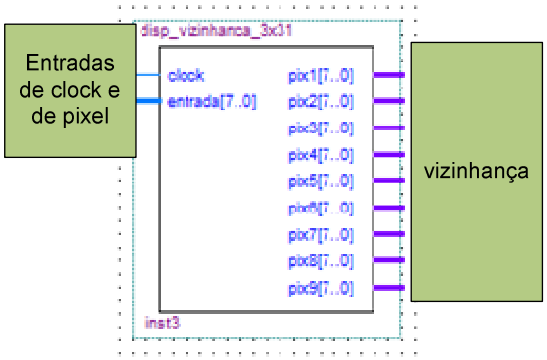


Figura 5.17 – Bloco de disponibilização de vizinhança.

A arquitetura do bloco da Figura 5.17 disponibiliza em suas saídas a vizinhança 3×3 de interesse, sendo a base para a implementação de diversas outras arquiteturas citadas no Capítulo 3 (convolução, correlação, filtros de ordem e morfologia). Os tópicos seguintes mostram a implementação dessas outras arquiteturas, todas utilizando o bloco da Figura 5.17.

5.4 – Convolução/Correlação

A implementação da arquitetura para convolução/correlação consiste na descrição em VHDL do elemento de processamento da Figura 3.17, adaptada na Figura 5.18. A disponibilização de vizinhança, descrita no tópico 5.3 está representada pela parte (a) da Figura 5.18, restando, então, a implementação das partes (b) multiplicadores e (c) somador.

A saída do bloco disponibilizador de vizinhança é composta de nove pixels, que são entradas do bloco de convolução, juntamente com o sinal de *clock* e os nove valores da máscara a ser utilizada.

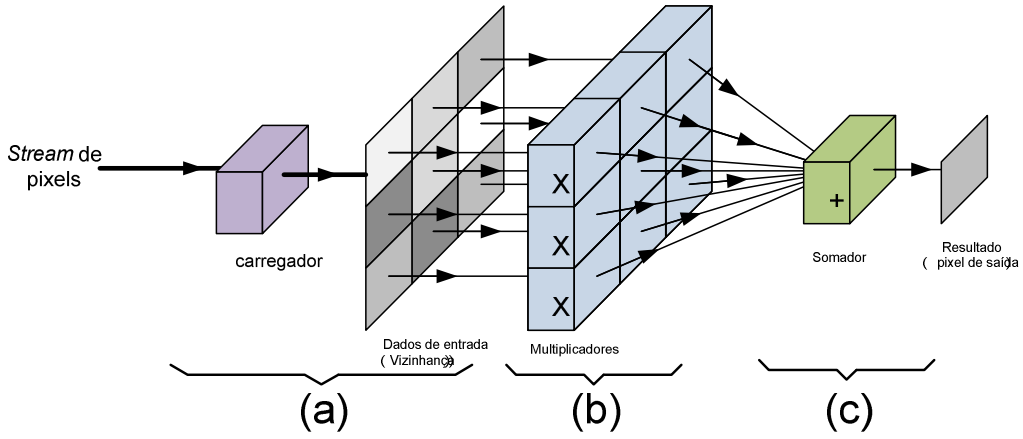


Figura 5.18 – Arquitetura de convolução/correlação.

O bloco de convolução é composto de três etapas, mostradas na Figura 5.18 (a), (b) e (c) tendo sido implementado em dois processos, sendo um responsável pelas multiplicações e outro responsável pela soma. O carregamento de vizinhança foi excluído do bloco de convolução, por ser uma operação comum a outros algoritmos. As nove multiplicações são sensíveis à borda de subida do *clock* e a soma é feita na borda de descida. Um bloco contendo apenas constantes foi implementado para servir de entrada da máscara de convolução. A Figura 5.19 mostra o conjunto montado para convolução, contendo o bloco de disponibilização da vizinhança, o bloco da máscara e o bloco de convolução propriamente dito, e a Tabela 5.4 mostra os dados de síntese do Quartus II para o bloco de convolução.

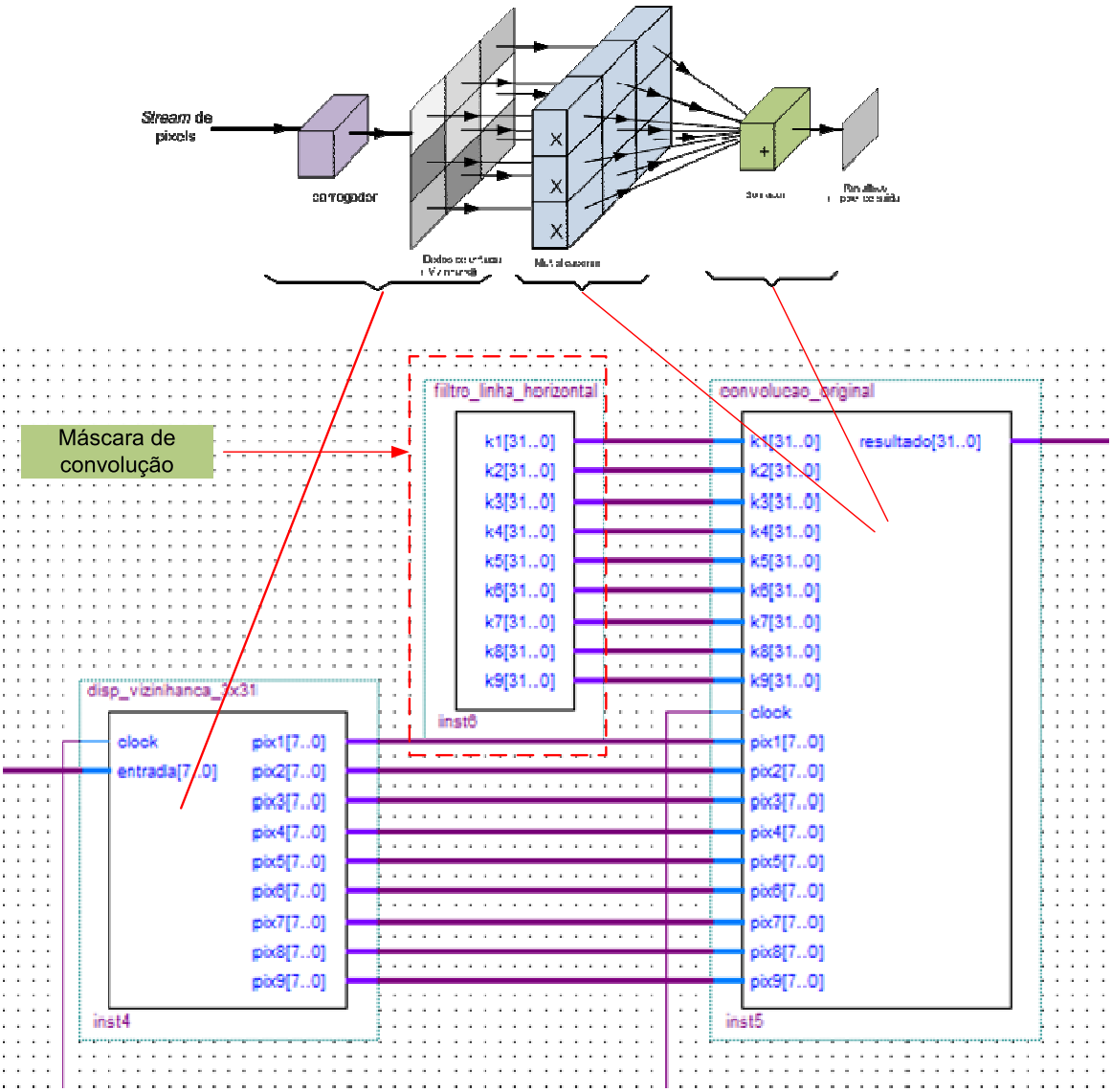


Figura 5.19 – Arquitetura para convolução/correlação.

Tabela 5.4 – Dados de síntese do bloco de convolução/correlação.

Bloco	Elementos Lógicos	Memory Bits	M4Ks	DSP 9x9	DSP 18x18
Convolução	78	0	0	0	6

5.5 – Filtros de Ordenamento

A base dos filtros de ordenamento são os comparadores, como visto no tópico 3.3.2. Um comparador simplesmente recebe como entradas o clock e dois pixels a serem comparados, atribuindo a duas saídas (maior, menor) os valores adequados. A Figura 5.20 mostra (a) o código VHDL completo do comparador e (b) o bloco comparador implementado.

```
14  library IEEE;
15  use ieee.std_logic_1164.all;
16  use ieee.numeric_std.all;
17  entity comparador is
18  port    (
19              clock : in std_logic;
20              pixel1: in integer;
21              pixel2: in integer;
22              maior: out integer;
23              menor: out integer
24          );
25  end comparador;
26  architecture Behavioral of comparador is
27  begin
28      process(clock)
29          begin
30              if(clock'event and clock='1')then
31                  if(pixel1 > pixel2)then
32                      maior <= pixel1;
33                      menor <= pixel2;
34                  else
35                      maior <= pixel2;
36                      menor <= pixel1;
37                  end if;
38              end if;
39      end process;
40  end Behavioral;
```

(a)

(b)

Figura 5.20 – (a) Código VHDL e (b) bloco do comparador.

Para o cálculo do máximo, do mínimo e da mediana de uma vizinhança, diversos comparadores devem ser organizados em arquiteturas como as mostradas nas Figuras 3.30. 3.31, 3.32 e 3.33. A Figura 5.21 repete o arranjo sistólico da Figura 3.30 e a Figura 5.22 mostra a montagem feita com comparadores como o da Figura 5.20.

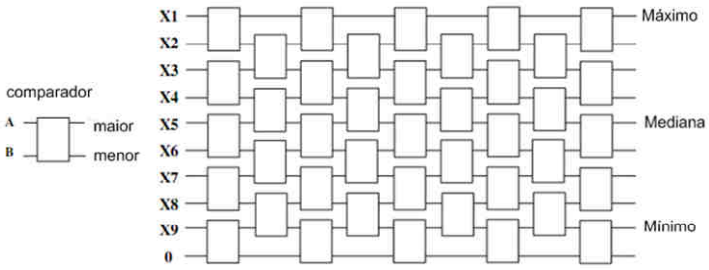


Figura 5.21 – Arranjo sistólico para ordenação de nove pixels.

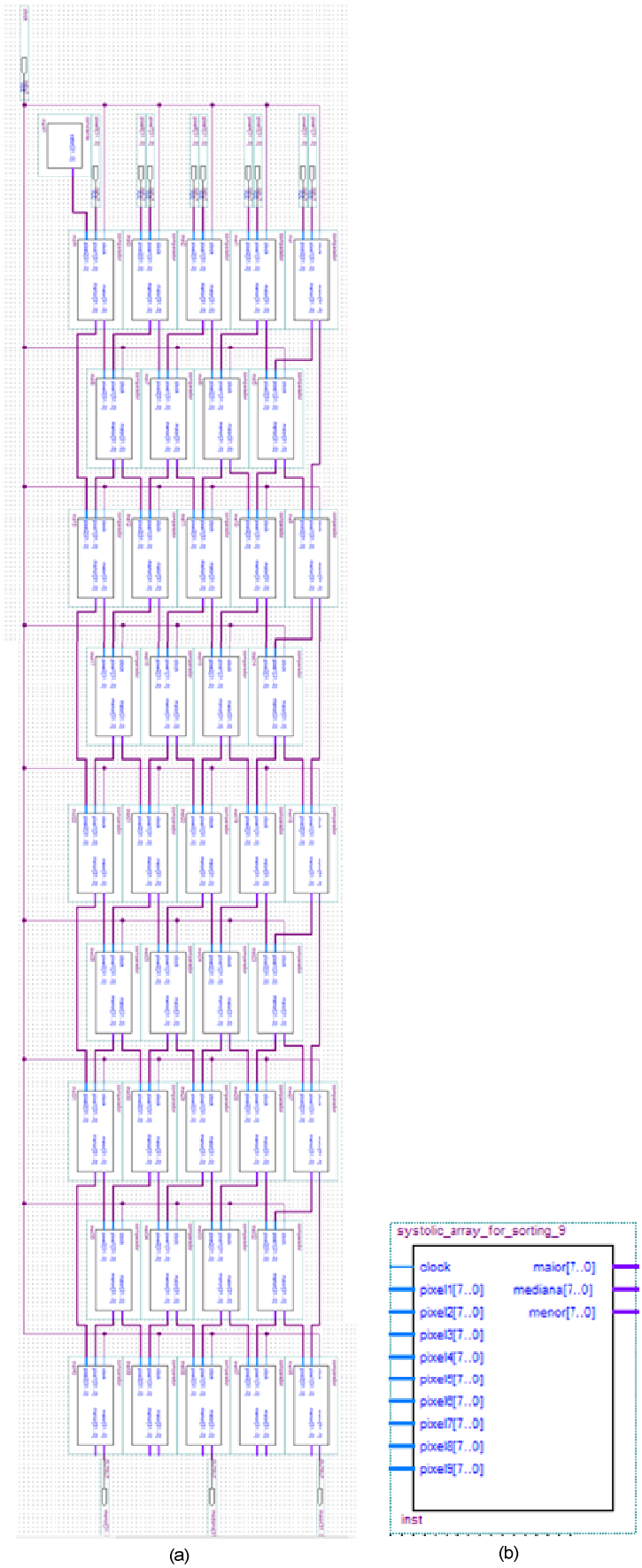


Figura 5.22 – Arranjo sistólico para ordenamento de nove pixels e seu bloco equivalente.

A utilização desse bloco de ordenamento é feita em conjunto com o bloco de disponibilização de vizinhança, assim como a convolução. A Figura 5.23 mostra as conexões feitas para o ordenamento de uma vizinhança 3x3.

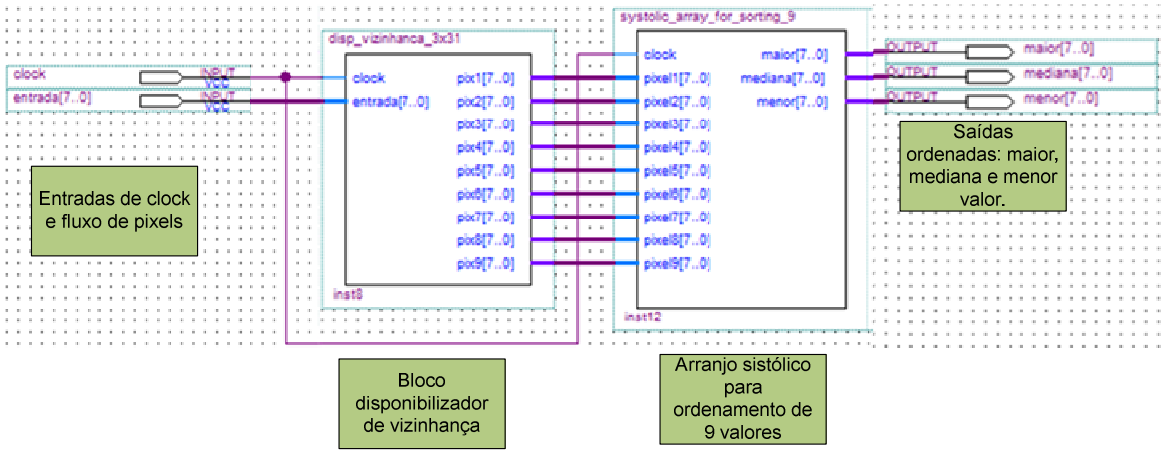


Figura 5.23 – Arquitetura completa para ordenamento de vizinhança 3x3.

A Tabela 5.5 mostra os dados de síntese do QuartusII.

Tabela 5.5 – Dados de síntese do comparador e da arquitetura sistólica de ordenamento.

Bloco	Elementos Lógicos	Memory Bits	M4Ks	DSP 9x9	DSP 18x18
Comparador	11	0	0	0	0
Arq. Sistólica	480	0	0	0	0

5.6 – Morfologia Binária

As operações morfológicas básicas, erosão e dilatação, também são implementadas utilizando como base o bloco de disponibilização de vizinhança. A Figura 5.24 (cópia da Figura 3.39) mostra como são calculados os valores de erosão e dilatação.

Erosão

k_i	f_i	e_i
0	0	1
0	1	1
1	0	0
1	1	1

Dilatação

k_i	f_i	d_i
0	0	0
0	1	0
1	0	0
1	1	1

$$e_i = \overline{k_i}.\overline{f_i} + \overline{k_i}.f_i + k_i.f_i$$

$$d_i = k_i.f_i$$

$$Erosão = e_1.e_2.e_3.e_4.e_5.e_6.e_7.e_8.e_9$$

$$Dilatação = d_1 + d_2 + d_3 + d_4 + d_5 + d_6 + d_7 + d_8 + d_9$$

Figura 5.24 – Cálculos para erosão e dilatação.

Os blocos de erosão/dilatação recebem os nove pixels da vizinhança, f_i na figura, e o elemento estruturante, k_i. Os valores e_i e d_i são calculados em um primeiro passo, sendo utilizados no segundo passo, que calcula finalmente o resultado da erosão e da dilatação, segundo as fórmulas da Figura 5.24. Ambas as operações são implementadas em apenas um ciclo de *clock*, sendo o primeiro passo calculado na borda de subida e o segundo na borda de descida. A Figura 5.25 mostra a arquitetura completa para uma operação de dilatação. Cabe ressaltar que os blocos de dilatação e erosão são intercambiáveis, possuindo as mesmas entradas e saídas, mudando apenas a lógica interna.

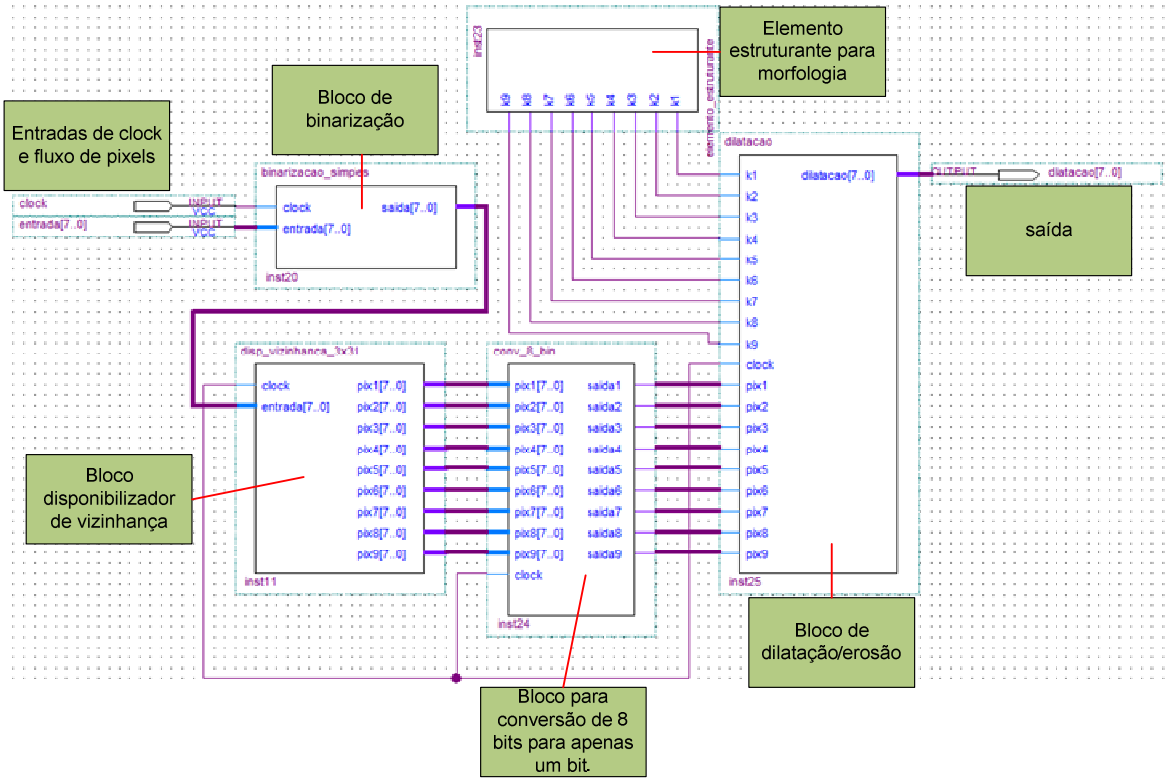


Figura 5.25 – Arquitetura para erosão/dilatação.

Na Figura 5.25 pode-se observar alguns blocos extras, além dos blocos de disponibilização de vizinhança, dilatação e elemento estruturante O bloco de binarização aparece porque a arquitetura é para trabalhar com morfologia binária, sendo necessária a binarização da imagem. O bloco *conv_8_bin* é dedicado à redução dos pixels da vizinhança, de oito bits para apenas um. A Tabela 5.6 mostra os dados de síntese para erosão/dilatação.

Tabela 5.6 – Dados de síntese para arquitetura de erosão/dilatação.

Bloco	Elementos Lógicos	Memory Bits	M4Ks	DSP 9x9	DSP 18x18
conv_8_bin	21	0	0	0	0
elemento	0	0	0	0	0

estruturante					
dilatação	11	0	0	0	0
erosão	11	0	0	0	0

5.7 – Operações aritméticas

As operações de segmentação analisadas no Capítulo 3 utilizavam a convolução e os filtros de ordem como base, seguidos de outras funções matemáticas. Tais funções são: *valor absoluto*, *soma*, *subtração* e *raiz quadrada*. Tais funções poderiam ser implementadas diretamente em VHDL, porém, o Quartus II disponibiliza a ferramenta já citada *MegaWizard Plugin Manager*, que contém diversas funções parametrizáveis e otimizadas para os FPGAs da Altera. Assim sendo, serão mostrados a seguir os procedimentos de configuração de algumas das funções matemáticas utilizadas.

5.7.1 – Somador/Subtrator

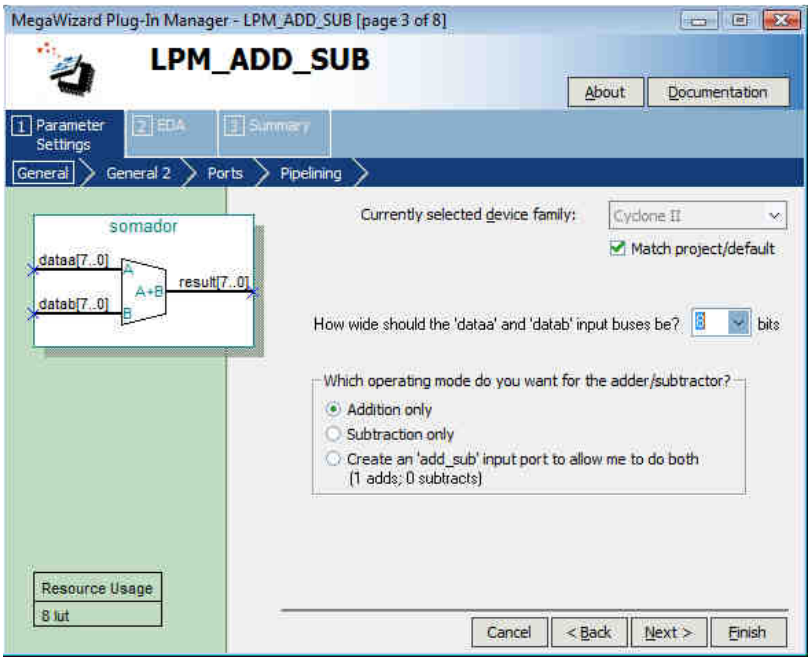


Figura 5.26 – Primeira tela de configuração de um *somador/subtrator*.

Na primeira tela de configuração, o bloco *somador/subtrator* permite a utilização de palavras com até 256 bits. Existem três configurações possíveis: apenas somador, apenas subtrator ou ambas as operações com uma entrada seletora.

Na Figura 5.27, há a opção de escolher-se uma das entradas como constante, bem como selecionar se os números são com ou sem sinal. No caso de escolher-se números com sinal,

a representação será em complemento-dois.

A terceira tela de configuração, Figura 5.28, permite configurar uma entrada de *carry*, uma saída de *carry* e uma saída indicativa de *overflow*.

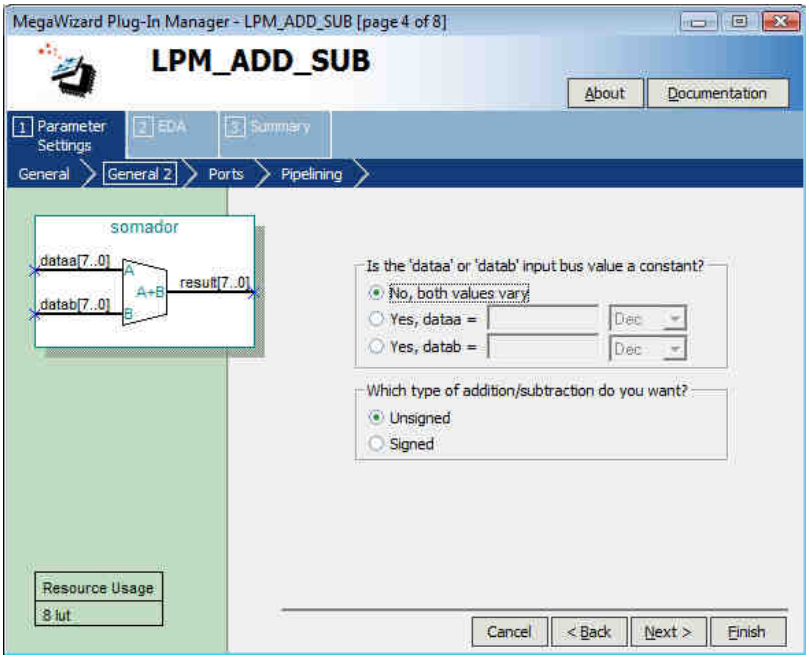


Figura 5.27 – Segunda tela de configuração de somador/subtrator.

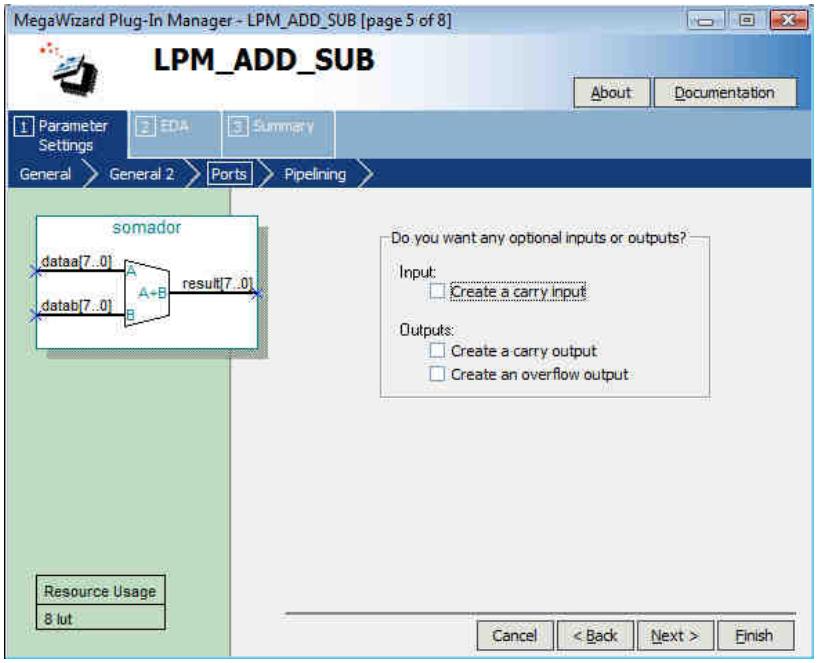


Figura 5.28 – Terceira tela de configuração de somador/subtrator.

A quarta e última tela de configuração do somador/subtrator (Figura 5.29), permite a

inserção de um *clock* para *delay* do bloco, muito útil em sistemas que necessitam de boa sincronização entre seus blocos, como em um *pipeline*. Para este trabalho, optou-se por incluir essa entrada de *clock*. Também é possível acrescentar um sinal de ativação (*enable*) e de limpeza das entradas (*clear*).

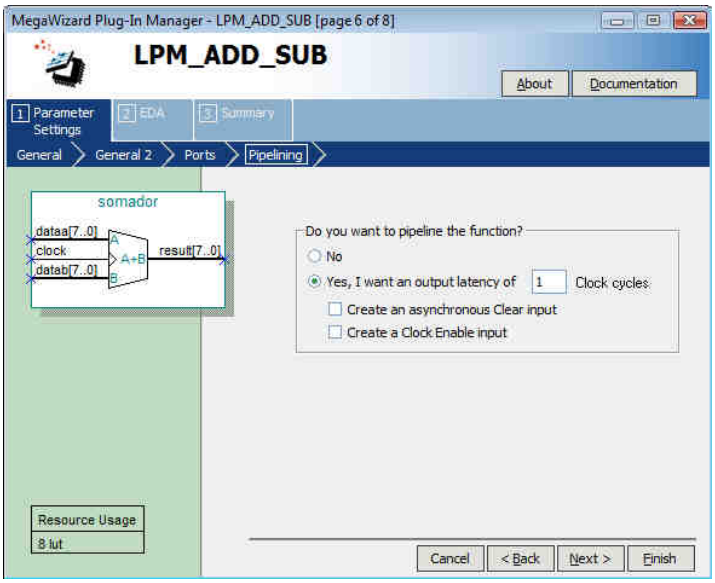


Figura 5.29 – Quarta tela de configuração do bloco somador/subtrator.

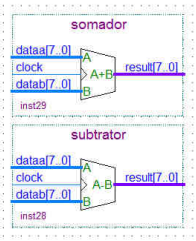


Figura 5.30 – Blocos somador e subtrator implementados.

5.7.2 – Valor Absoluto

A única tela de configuração do bloco de valor absoluto é mostrada na Figura 5.31.

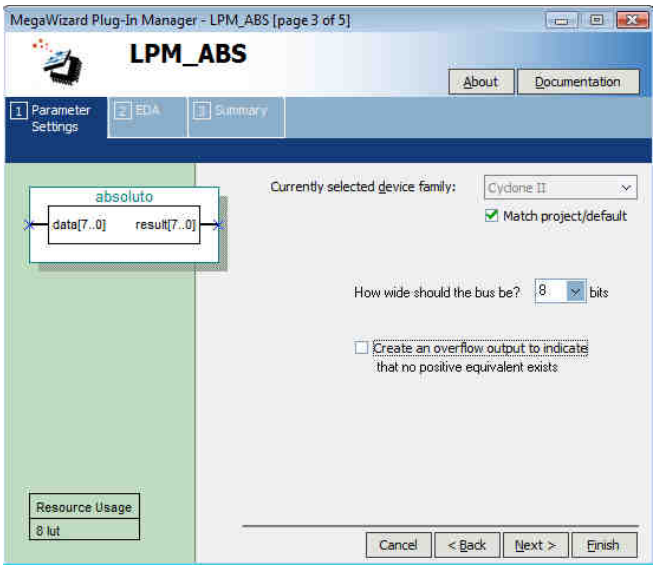


Figura 5.31 – Tela de configuração do bloco de valor absoluto.

É possível trabalhar com palavras de até 256 bits. No caso, a entrada é considerada um inteiro com sinal, na representação complemento-dois. Pode-se habilitar uma saída que representa um *overflow*, no caso a impossibilidade de representar o valor absoluto com a quantidade de bits especificada.

Figura 5.32 – Bloco de extração de valor absoluto.

5.7.3 – Raiz Quadrada

O bloco de raiz quadrada também possui apenas uma tela de configuração, que permite a escolha do tamanho de palavra (até 256 bits) e configurá-lo com um *clock* de sincronização.

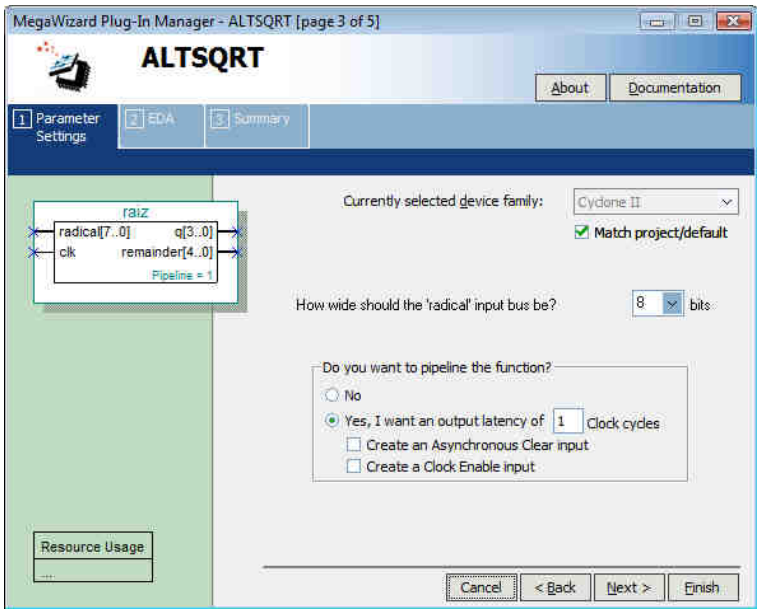


Figura 5.33 – Tela de configuração de raiz quadrada.

5.7.4 – Multiplicador

A primeira tela de configuração do multiplicador (Figura 5.34) permite a escolha de multiplicarem-se dois números ou elevar um número ao quadrado. Neste trabalho foi utilizada a segunda opção. Pode-se também trabalhar com números com e sem sinal, com até 256 bits e representação em complemento-dois (Figura 5.35).

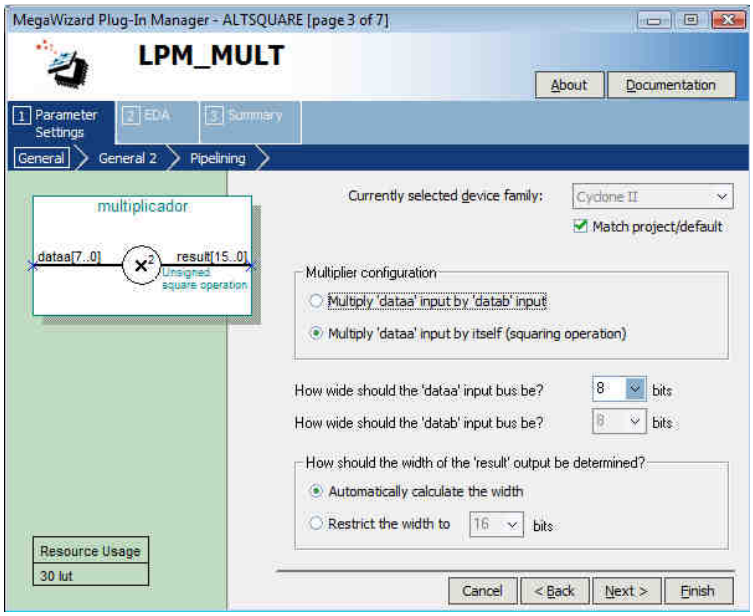


Figura 5.34 – Escolha entre multiplicação simples e elevação ao quadrado.

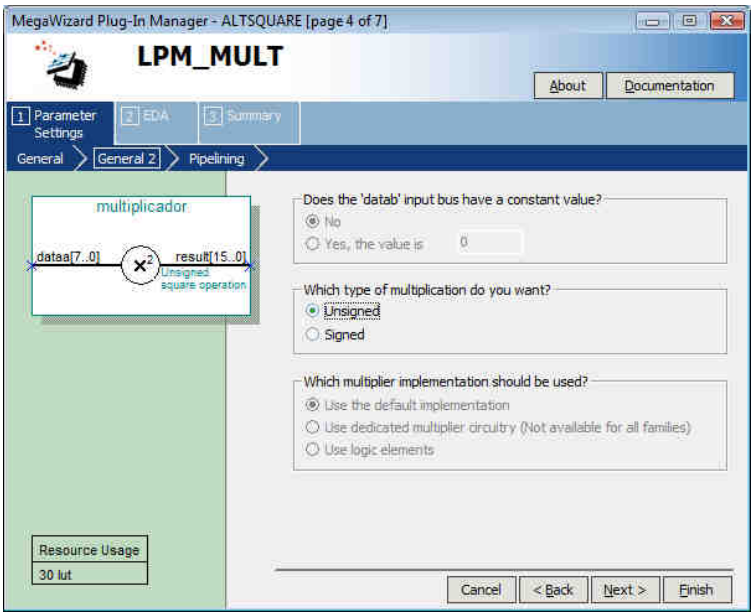


Figura 5.35 – Pode-se escolher entre números com ou sem sinal.

Finalmente, pode-se acrescentar um *clock* de sincronização (Figura 5.36), além de um *clear* e um sinal de *enable*.

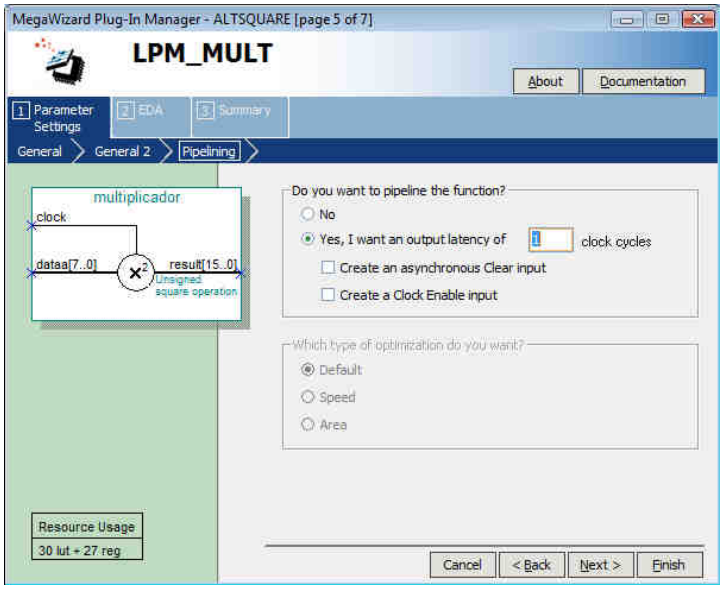


Figura 5.36 – Inclusão de sinais de *clock*, *clear* e *enable*.

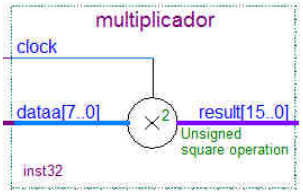


Figura 5.37 – Bloco multiplicador (elevação ao quadrado).

5.8 – Métodos para testes e validações das arquiteturas

Quando da captura de imagens por uma câmera, não é possível prever qual serão os valores de todos os pixels de uma imagem, apenas após a captura pode-se verificar pixel a pixel. A implementação dos algoritmos deve ser testada com entradas conhecidas, de modo que possa ser feita uma verificação e validação dos cálculos feitos.

Com esse intuito, foram implementados dois sistemas de testes, cuja diferença básica é a forma de entrada de dados. Essa entrada, como já visto, deve ser serial, de modo a simular o funcionamento da câmera. Serão descritos a seguir os dois sistemas utilizados para os testes.

O primeiro sistema consiste na implementação de uma memória internamente na FPGA, com valores pré-carregados, e lê-la sincronamente, de acordo com um clock.

O segundo sistema utiliza uma comunicação serial RS232 entre o KIT DE2 e um computador comum o qual envia os pixels por sua porta serial, com uma freqüência arbitrária.

5.8.1 – Utilização de memória pré-carregada

Utilizando o *MegaWizard Plugin Manager,* foi implementada uma memória ROM de uma saída, sincronizada com um clock. As Figuras 5.38 a 5.41 mostram as etapas de configuração dessa memória no *MegaWizard*.

Foi definida uma imagem de testes quadrada, com tamanho 32x32, totalizando 256 words de 8 bits (Figura 5.38). Configurou-se a saída para ser síncrona com o mesmo clock da entrada de endereços (Figura 5.39). Desse modo, a saída é atualizada de forma instantânea. Para o pré-carregamento da imagem, utilizou-se um arquivo de inicialização (*teste1.mif*), com extensão **.mif* (*Memory Initialization File*), Figura 5.40.

Um script (Figura 5.41) foi criado para gerar um arquivo de inicialização a partir de uma imagem qualquer. O arquivo gerado armazena os pixels com endereços seqüenciais, de 0 a 255. Como entrada da memória ROM, é necessário passar um endereço, o que foi feito criando-se um contador binário, de 8 bits. Assim sendo, quando do início do processo, o contador é zerado e incrementado de forma síncrona com o clock.

A Figura 5.42 mostra o circuito com a memória ROM, o contador e a arquitetura de

disponibilização de vizinhança, já explicada neste capítulo. Cabe ressaltar que, por ser a imagem de teste (32x32) bem menor que as capturadas pela câmera e sem a necessidade de pixels de sincronização (1056x525), o comprimento dos registradores de deslocamento para armazenamento de linhas teve de ser reduzido.

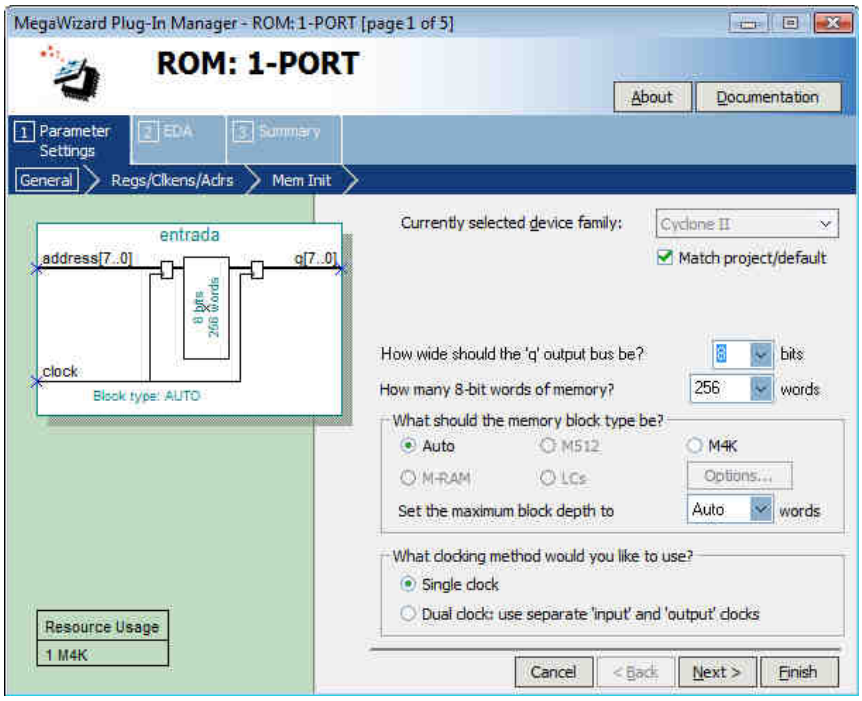


Figura 5.38 – Determinação do tamanho da memória e das palavras.

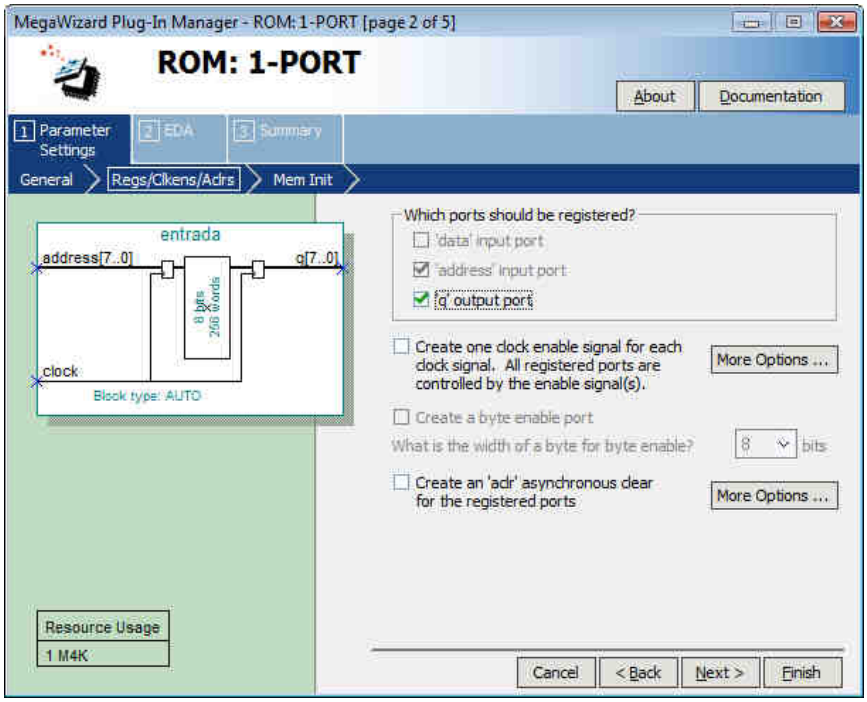


Figura 5.39 – Tornando a saída síncrona.

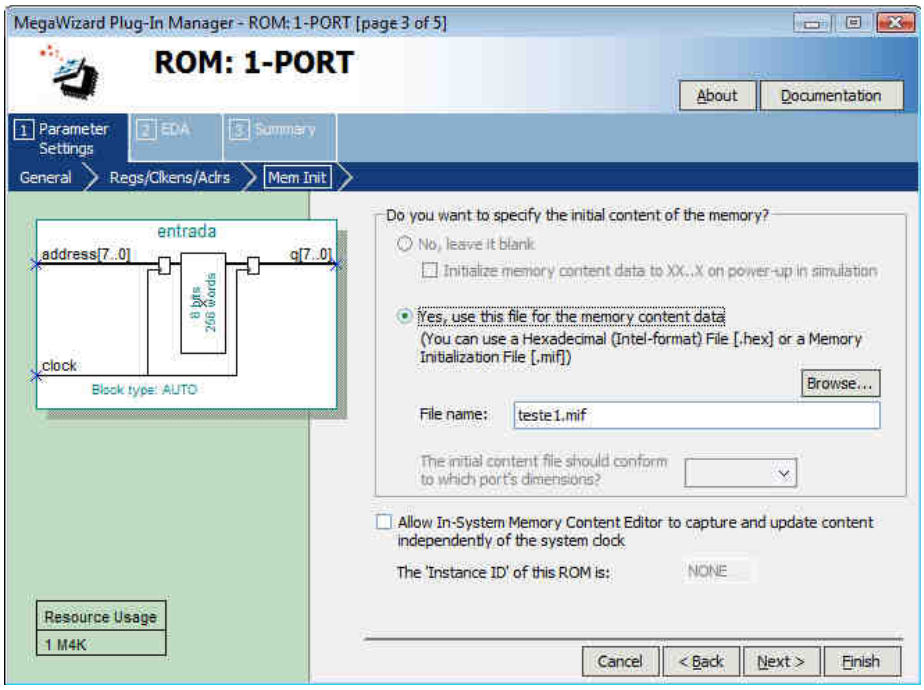


Figura 5.40 – Seleção de arquivo de inicialização.

```
%%
% Script para gerar um arquivo *.mif a partir de uma imagem
% em escala de cinza

% abertura do arquivo de saida
fid = fopen('teste1.mif', 'w');

%cabeçalho do arquivo
fprintf(fid,'WIDTH=8;\n');
fprintf(fid,'DEPTH=256;\n');
fprintf(fid,'ADDRESS_RADIX=UNS;\n');
fprintf(fid,'DATA_RADIX=UNS;\n');
fprintf(fid,'CONTENT BEGIN\n');

% leitura da imagem a ser convertida
I = imread('imagem.bmp');
I = rgb2gray(I);
[M N C] = size(I);

% percorrer imagem pixel a pixel, escrevendo no arquivo mif
contador = 0;
for i=1:M
    for j=1:N
        fprintf(fid,'   %d : %d;\n',contador,I(i,j));
    end;
end;

% fim do arquivo mif
fprintf(fid,'END;');

% fechar arquivo gerado
fclose(fid);
close all;
clear all;
```

Figura 5.41 – Script em MATLAB para geração de arquivo **.mif* a partir de uma imagem.

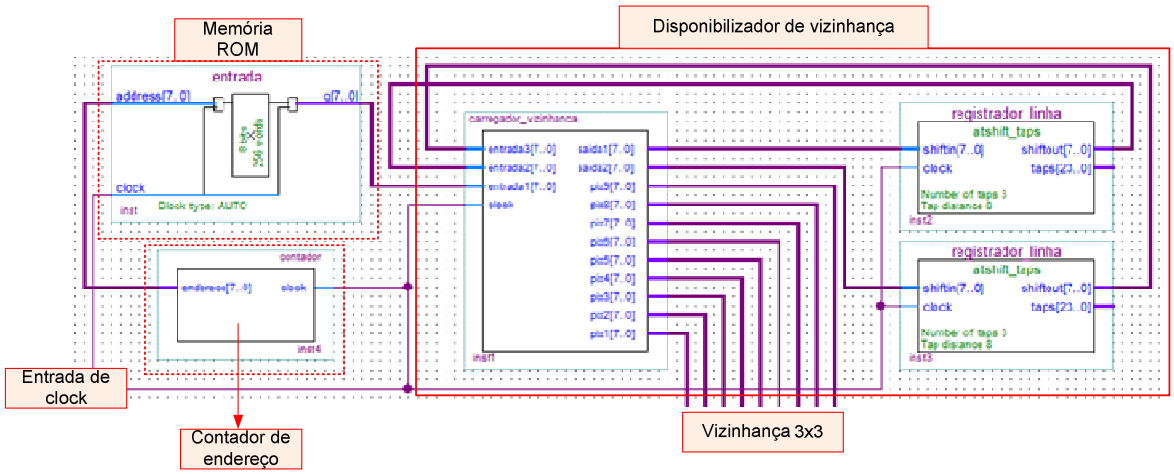


Figura 5.42 – Circuito montado com a utilização da memória ROM.

A verificação dos dados calculados pelas arquiteturas descritas neste capítulo foi feita via simulação, no próprio Quartus II. O tópico 5.9 mostra o desenvolvimento e simulação das arquiteturas de convolução mostradas no capítulo 3.

5.8.2 – Comunicação RS232

Outro sistema de testes utilizado baseou-se em enviar uma imagem, pixel a pixel, de forma serial para o Kit DE2, simulando a entrada de uma câmera. O circuito responsável pela comunicação serial possui uma saída que tem valor lógico '1' quando um dado novo é recebido pela porta serial. Esse sinal foi utilizado como *clock* de sincronização do resto do sistema.

A Figura 5.43 mostra o circuito montado com o bloco de comunicação serial e os módulos para disponibilização de vizinhança.

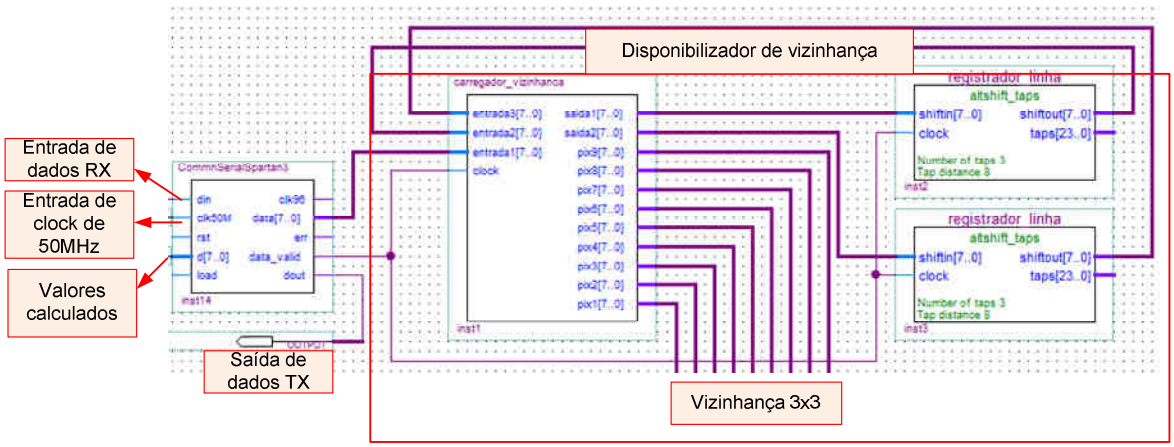


Figura 5.43 – Circuito para testes com a comunicação serial RS232.

A vantagem da utilização da comunicação serial no lugar da memória ROM é a de que o circuito pode enviar de volta ao computador os valores dos pixels calculados, facilitando a validação dos cálculos.

5.9 – Outras arquiteturas de Convolução

No capítulo 3 (tópico 3.2.3 - Arquiteturas para eliminação de redundâncias), foram propostas algumas arquiteturas alternativas para o cálculo da convolução, eliminando cálculos redundantes de modo a economizarem-se os multiplicadores internos dos FPGAs.

Para a utilização dessas arquiteturas, deve-se modificar o bloco de disponibilização de vizinhança, para que forneça em suas saídas os pixels desejados. As etapas de multiplicação e soma continuam praticamente as mesmas. Para essas arquiteturas, a memória de entrada foi codificada como um *array* em VHDL diretamente, sem a utilização do MegaWizard. Isso foi feito para facilitar o controle da entrada de pixels, que se torna mais complexo, pois em algumas dessas arquiteturas dois novos pixels tem de ser carregados ao mesmo tempo.

Os tópicos a seguir mostram resultados de síntese e simulações, para as arquiteturas com eliminação de redundâncias. Estas implementações contemplam a operação de convolução completa, incluindo as operações de multiplicação e soma, além da disponibilização de vizinhança. A implementação e testes dessas arquiteturas foram feitos utilizando a ferramenta de simulação do Quartus II, não tendo sido testadas com a câmera.

5.9.1 – Arquitetura Comum

Para efeitos comparativos, foi testada neste primeiro tópico a arquitetura comum de convolução, para máscaras de tamanhos 3x3, 5x5 e 7x7. Foi testada ainda a duplicação dessa arquitetura, de modo que metade da imagem fosse processada por uma arquitetura e a outra metade, pela outra. A Tabela 5.7 mostra os resultados de síntese para uma única arquitetura, e a Tabela 5.8 os resultados para a duplicação.

Tabela 5.7 – Resultados de síntese e simulação para convolução simples.

tamanho da máscara	elementos lógicos	memory bits	freqüência máxima	pixels/ciclos de clock	troughput (Mpixels/s)	multiplicações
3x3	2010	11392	127,88 MHz	1 pixel/10 ciclos	12,788	9
5x5	3582	16466	119,56 MHz		11,956	25

7x7	5153	19328	105,31 MHz		10,531	49

Tabela 5.8 – Resultados de síntese e simulação para arquitetura dupla para convolução simples.

tamanho da máscara	elementos lógicos	memory bits	freqüência máxima	pixels/ciclos de clock	troughput (Mpixels/s)	multiplicações
3x3	3047	14145	112,92 MHz	2 pixels/ 10 ciclos	22,584	18
5x5	5034	17746	108,44 MHz		21,688	50
7x7	6945	21446	104,30 MHz		20,86	98

Comparando os resultados de síntese das Tabelas 5.7 e 5.8, podemos notar que a freqüência de operação diminui para máscaras maiores e também diminui quando da duplicação de máscaras de mesmo tamanho. Porém, como o a quantidade de pixels calculados por ciclo de clock é o dobro quando se duplicam os processadores, para a maior faixa de freqüências de operação, é vantajosa a utilização de dois blocos de processamento, quando há espaço disponível para essa implementação.

5.9.2 – Primeira arquitetura

Esta arquitetura calcula sobrepõe duas máscaras de convolução como na Figura 5.44. Os dados de síntese e simulação são mostrados na Tabela 5.9.

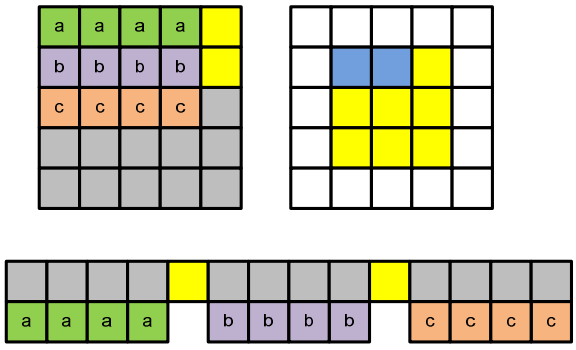


Figura 5.44 – Primeira arquitetura.

Tabela 5.9 – Resultados de síntese e simulação para convolução simples.

tamanho da máscara	elementos lógicos	memory bits	freqüência máxima	pixels/ciclos de clock	troughput (Mpixels/s)	multiplicações
3x3	4135	17383	117,31 MHz	2 pixels/10	23,462	12

				ciclos		

5.9.3 – Segunda arquitetura

Nesta arquitetura a economia de multiplicadores é feita com base na sobreposição mostrada na Figura 5.45. A Tabela 5.10 mostra os resultados de desempenho em síntese e simulação.

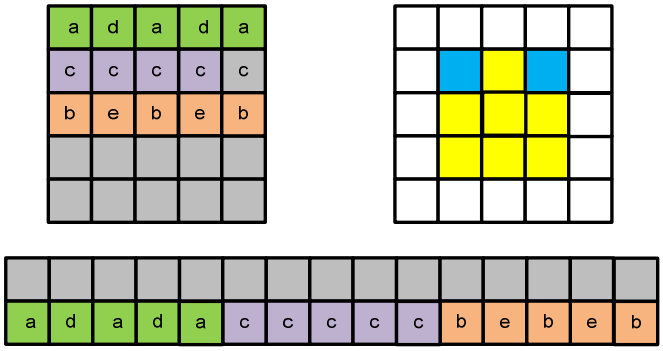


Figura 5.45 – Segunda arquitetura.

Tabela 5.10 – Resultados de síntese e simulação para arquitetura da Figura 5.45.

tamanho da máscara	elementos lógicos	memory bits	freqüência máxima	pixels/ciclos de clock	troughput (Mpixels/s)	multiplicações
3x3	4920	16406	98,05 MHz	2 pixels/14 ciclos	14,007	15

5.9.4 – Comparações entre as arquiteturas de convolução

Os gráficos das Figuras 5.46, 5.47 e 5.48 mostram dados comparativos entre as arquiteturas de convolução mostradas anteriormente.

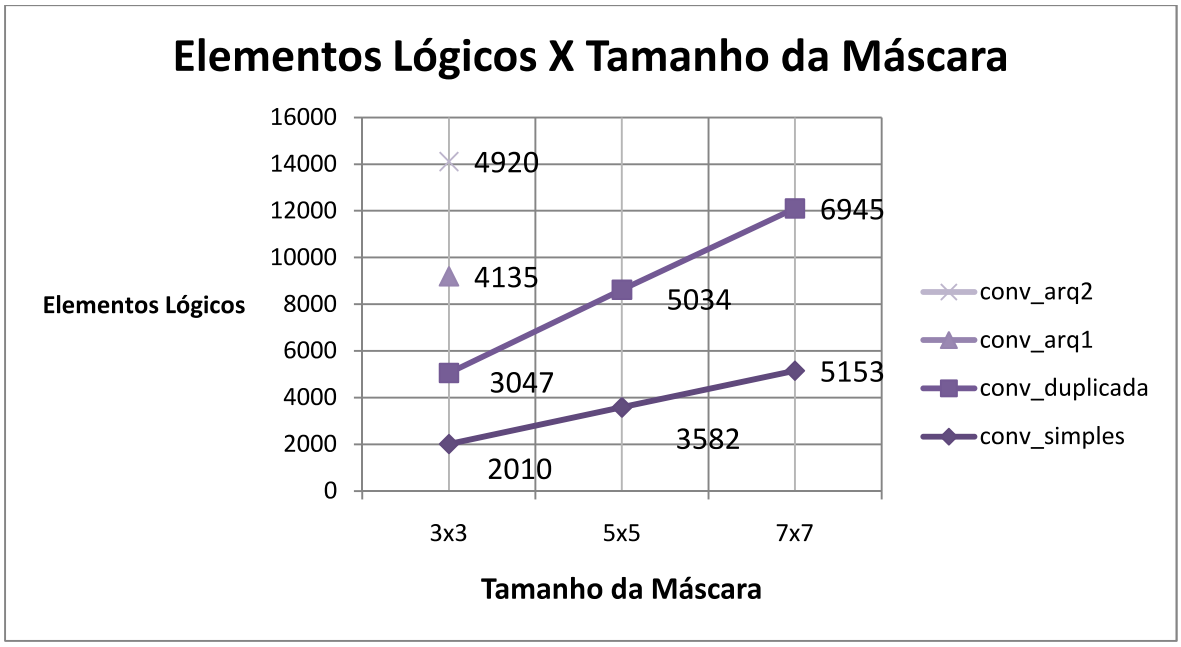


Figura 5.46 – Gráfico elementos lógicos X tamanho da máscara.

No gráfico da Figura 5.46, observa-se uma tendência de crescimento na utilização de elementos lógicos quando do aumento da máscara, para qualquer das arquiteturas. Essa tendência já era esperada, visto que máscaras maiores representam maior consumo de recursos. Apesar de a arquitetura conv_duplicada ser apenas a duplicação da arquitetura comum, observa-se que a quantidade de elementos lógicos utilizados não é o dobro, e que esse a tendência de crescimento é maior. Isso deve ser efeito do aumento do roteamento interno e da estrutura de controle para essa arquitetura.

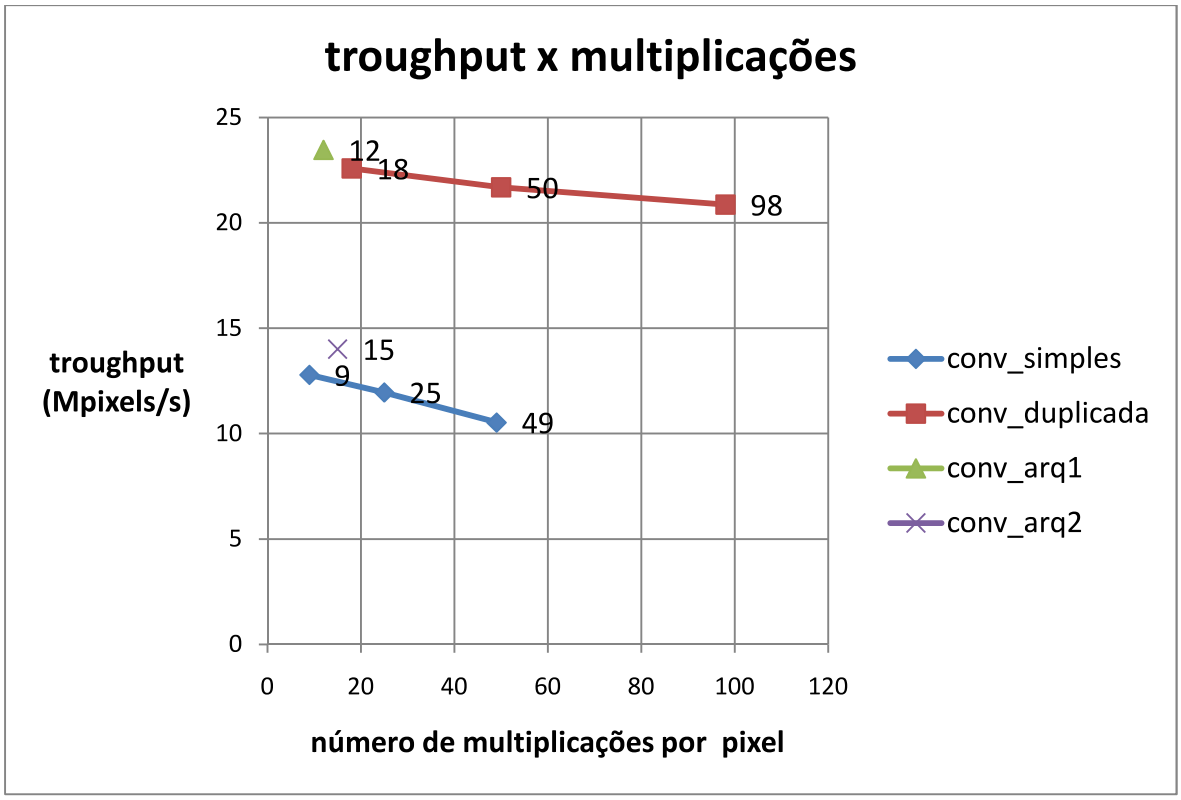


Figura 5.47 – gráfico *troughput* X multiplicações.

No gráfico da Figura 5.47, pode-se observar que o aumento da máscara gerou uma queda no fluxo máximo de dados nas arquiteturas conv_simples e conv_duplicada. Observa-se ainda que, para um mesmo tamanho de máscara (3x3), a arquitetura conv_arq1 apresenta o maior troughput, com a maior economia de operações de multiplicação.

5.10 – Conclusões do capítulo

Neste capítulo foi mostrada a transcrição em hardware das arquiteturas propostas no Capítulo 3. A utilização da ferramenta de projeto esquemático do Quartus II permitiu a visualização das arquiteturas implementadas, mostrando a efetividade do método utilizado. Foi explicitada a hierarquização possível quando do trabalho com blocos construtivos.

A facilidade de criação de sistemas completos de processamento de imagens utilizando os blocos construídos neste capítulo será mostrada no Capítulo 6, por meio da implementação de alguns exemplos, mostrando a versatilidade da plataforma utilizada.

Algumas arquiteturas foram propostas para eliminarem-se redundâncias no cálculo da convolução, por meio da análise de semelhanças das máscaras mais comuns. Esse estudo

de reaproveitamento de cálculos redundantes poderia ser estendido, não sem alguma dificuldade, às operações dos filtros de ordem e da morfologia binária.

Os projetos de circuitos devem levar em consideração o consumo dos recursos disponíveis, elaborando um *tradeoff* entre as diversas possibilidades.

6. TESTES DE IMPLEMENTAÇÃO DAS ARQUITETURAS

Neste capítulo são apresentadas algumas implementações de exemplos de processamento de imagens utilizando os blocos de circuito definidos no capítulo 5. Todos os testes foram feitos com base na montagem da Figura 6.1, que mostra a plataforma de hardware completa composta de câmera, placa de desenvolvimento e display. Para testes, utilizou-se um monitor de computador em frente à câmera, de modo a diminuir um pouco a sensibilidade à iluminação. É uma alternativa muito útil, pois se pode capturar imagens estáticas, vídeos e efetuar diversos ajustes de contraste e brilho nas imagens de origem, antes da captura pela câmera.

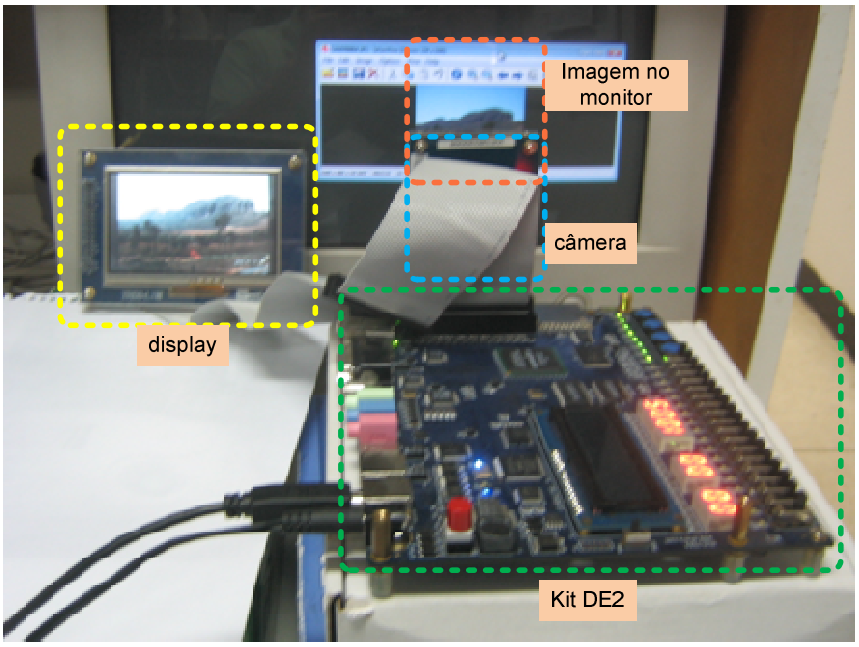


Figura 6.1 – Montagem para testes do sistema.

6.1 – Captura e visualização de imagens

O sistema implementado neste projeto utilizou alguns blocos funcionais já prontos, e outros desenvolvidos totalmente durante o trabalho. Todos os blocos de processamento de imagens foram desenvolvidos utilizando apenas as referências citadas na bibliografia.

A Terasic Inc. disponibiliza em seu site para *download* os blocos com as funcionalidades de controle da câmera e do display utilizados. O código-fonte também é fornecido, mas integralmente na linguagem Verilog. Devido à versatilidade do software Quartus II, é possível mesclar as três principais linguagens de descrição de hardware disponíveis hoje

(Verilog, VHDL e AHDL). As implementações mostradas no capítulo 5 poderiam ter sido totalmente desenvolvidas em Verilog, porém, fatores não-técnicos, optou-se pela utilização da linguagem VHDL.

A Figura 6.3 mostra o projeto RTL do sistema completo utilizado, com destaque para o bloco de processamento de imagens, denominado IPU (*Image Processing Unit*). Fora a IPU, todos os outros blocos foram disponibilizados pelo fabricante.

Os blocos de controle realizam operações como a configuração dos registradores da câmera e do display responsáveis por fixar os parâmetros de tamanho da imagem, taxa de aquisição e tempo de exposição. Há também um bloco responsável por transformar os dados do sensor da câmera em valores RGB.

A IPU foi inserida imediatamente antes do bloco que envia os dados para o display, recebendo como entrada o sinal de clock e os três canais de cores de oito bits, fornecendo em sua saída os três canais de cores já processados, conforme pode ser visto na Figura 6.2.

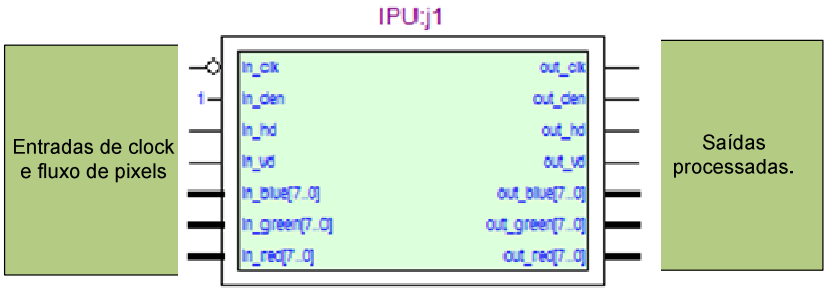


Figura 6.2 – *Image Processing Unit*.

Os sinais de entrada *in_hd*, *in_vd* e *in_den* foram colocados com o intuito apenas de organização, pois não são processados pela IPU, apenas passam por ela sem intermediários. Tais sinais são utilizados para identificação pelo display de início e fim de linha e de janela, bem com para ativação e desativação do display.

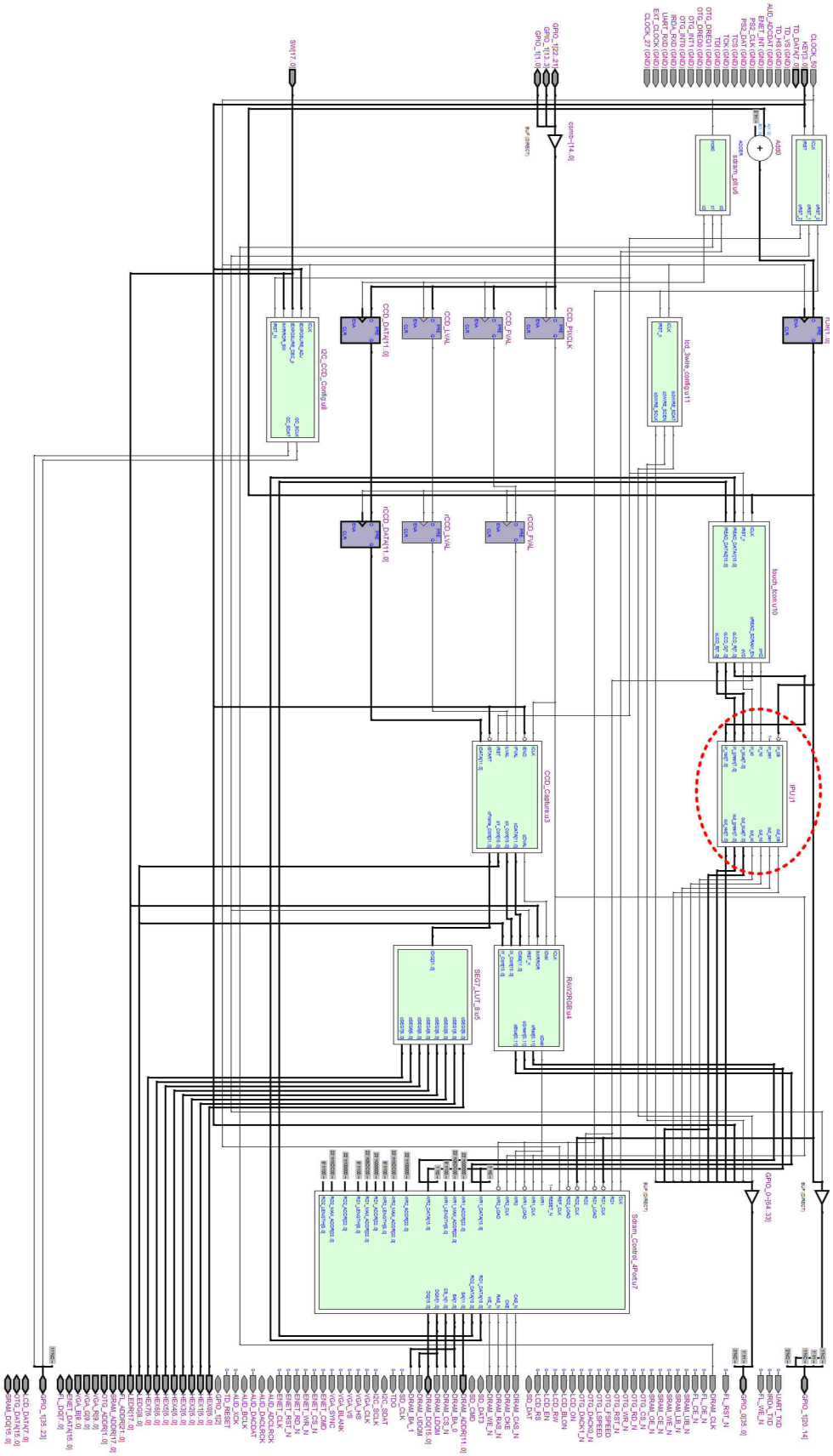
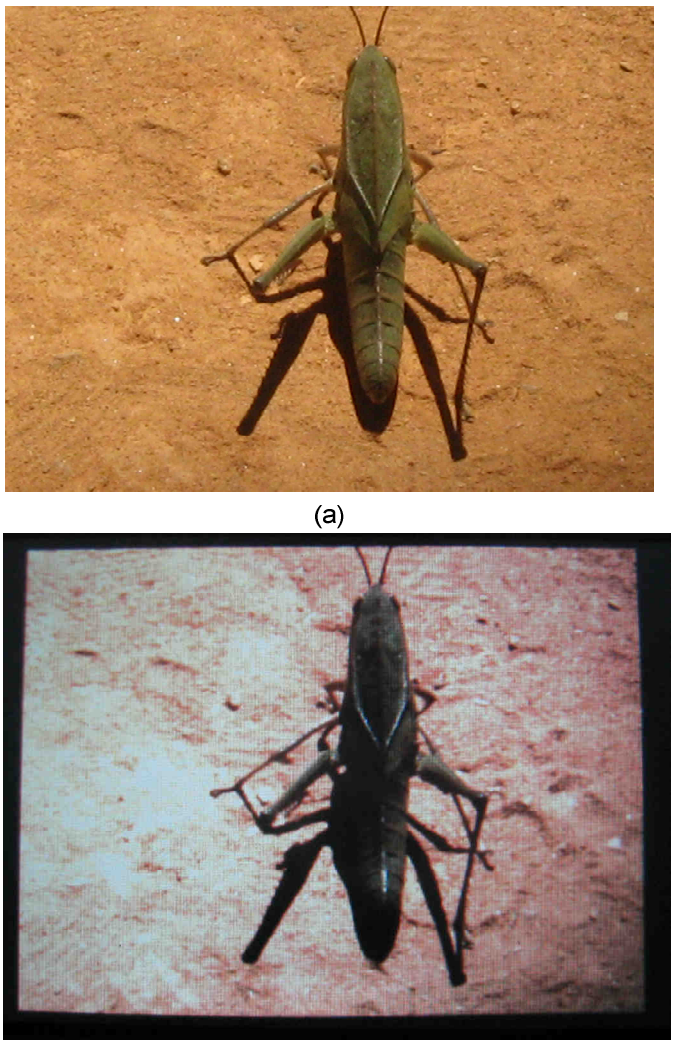


Figura 6.3 – Diagrama RTL de todo o sistema implementado.

Para captura e visualização de imagens sem qualquer processamento, basta implementar a IPU sem nenhum elemento de processamento em sua arquitetura, conforme mostrado na Figura 6.4.

Figura 6.4 – Arquitetura da IPU sem qualquer tipo de processamento.

Na figura 6.5(a) é mostrada a imagem de teste (disposta no monitor, em frente à câmera) e (b) uma fotografia do display mostrando a imagem capturada pelo sistema.



(a)

(b)

Figura 6.5 – (a) Imagem de teste, (b) imagem capturada, mostrada no *display*.

6.2 – Conversão para escala de cinza

Para conversão de imagem colorida para escala de cinza, basta acrescentar o bloco de conversão de cores, visto no capítulo 5 (Figura 6.6).

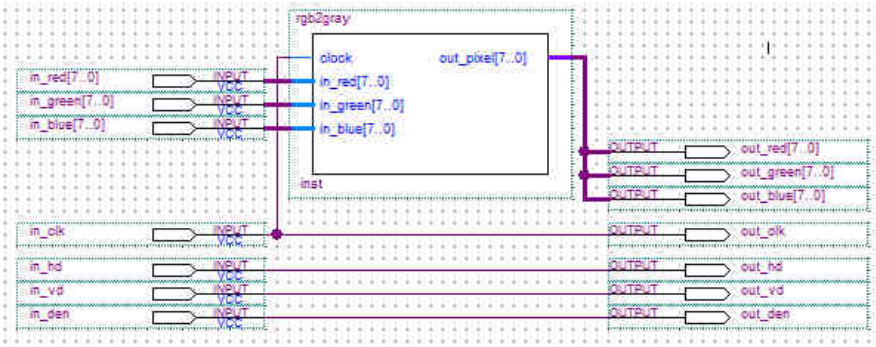


Figura 6.6 – Arquitetura da IPU para conversão de cores.

A Figura 6.7 mostra a imagem convertida para escala de cinza pela arquitetura da figura 6.6. Dois detalhes que podem ser observados são um “colar” branco no pescoço do inseto e um linha também branca em suas costas.

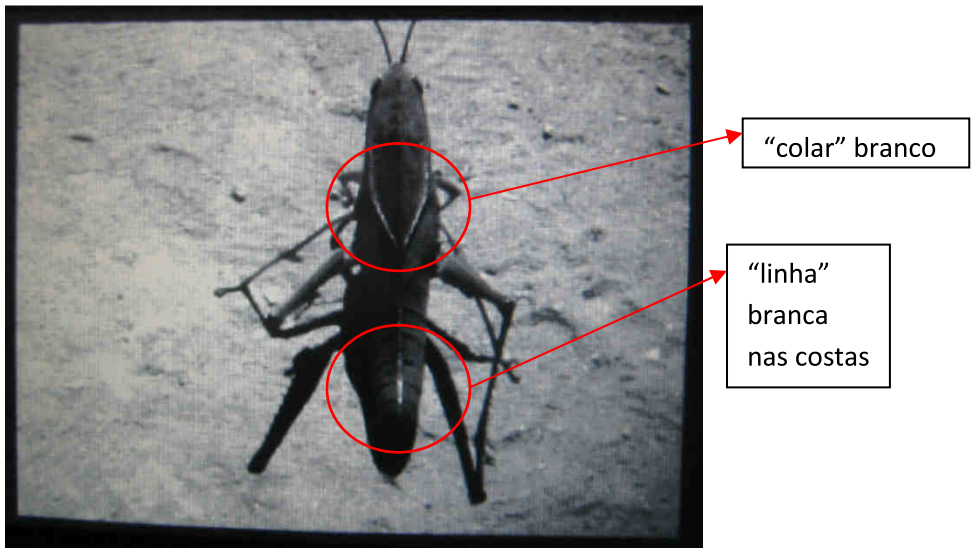


Figura 6.7 – Conversão de imagem colorida para escala de cinza.

6.3 – Filtros de Ordem

A Figura 6.8 mostra a arquitetura para filtragem por ordenamento. Para essa implementação, acrescentou-se o bloco de filtro de ordenamento logo após a conversão para escala de cinza. A Figura 6.9 mostra a imagem filtrada por (a) mediana, (b) maior valor (dilatação em escala de cinza) e (c) menor valor (erosão em escala de cinza).

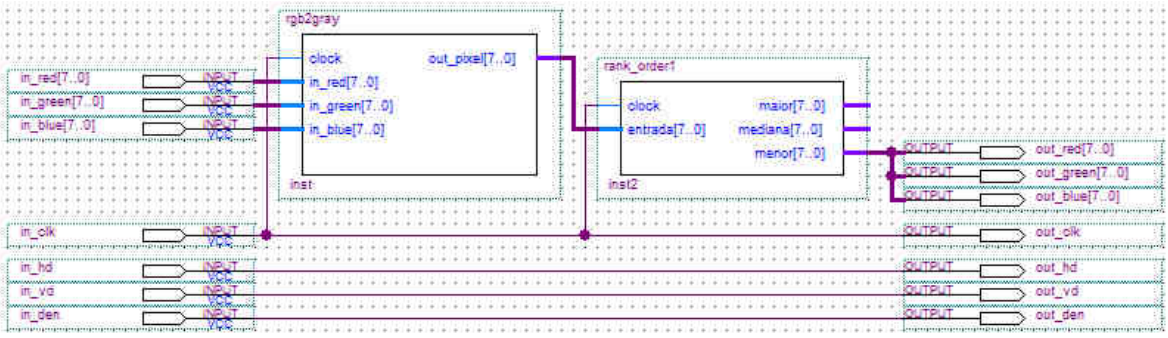
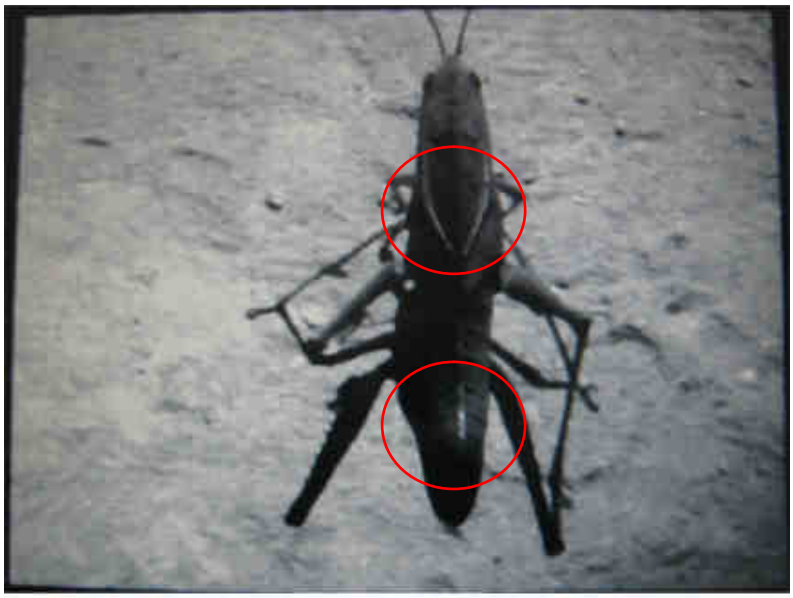


Figura 6.8 – Filtragem por ordenamento.



(a)

(b)

(c)

Figura 6.9 – Filtragem por (a) mediana, (b) por máximo, (c) por mínimo.

Na Figura 6.9 (b) e (c), pode-se notar o alargamento e o afinamento dos detalhes mais claros, respectivamente, por exemplo, o colar no pescoço do inseto e a linha branca de suas costas. Na Figura 6.9(b) esses detalhes ficam um pouco mais realçados, já na Figura 6.9(c) já são pouco notados, como efeito da dilatação em (a) e da erosão em (b).

Acrescentando um bloco subtrator à arquitetura da Figura 6.8, podemos efetuar a subtração das Figuras 6.9 (b) e (c), de modo a realçar as bordas da imagem original. A Figura 6.10 mostra a nova arquitetura com o subtrator, e a Figura 6.11 o resultado, com as bordas nitidamente destacadas.

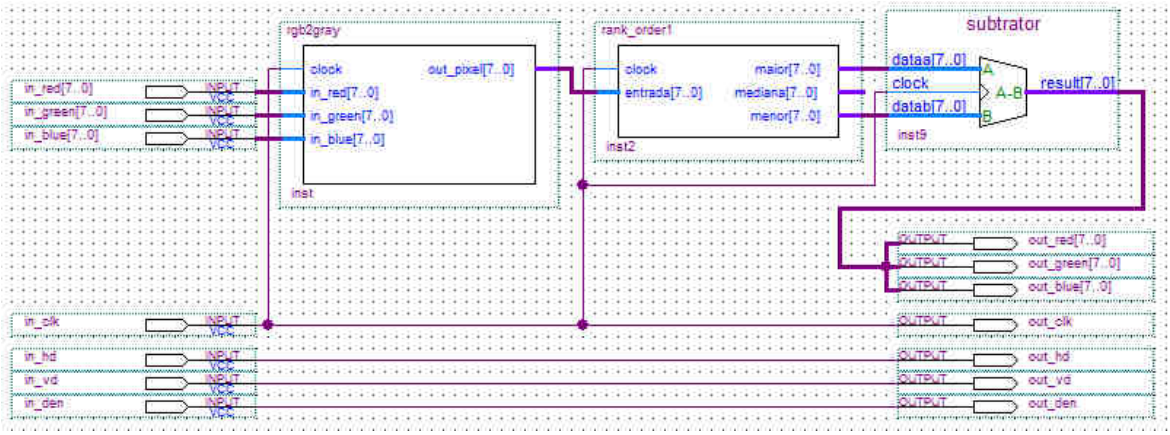


Figura 6.10 – Inclusão de operação de subtração entre as imagens das Figuras 6.9 (b) e(c).

Pode-se destacar na arquitetura da Figura 6.10 que os resultados dos filtros de dilatação e

erosão em escala de cinza (filtros de máximo e mínimo, respectivamente) são gerados simultaneamente, de forma paralela.

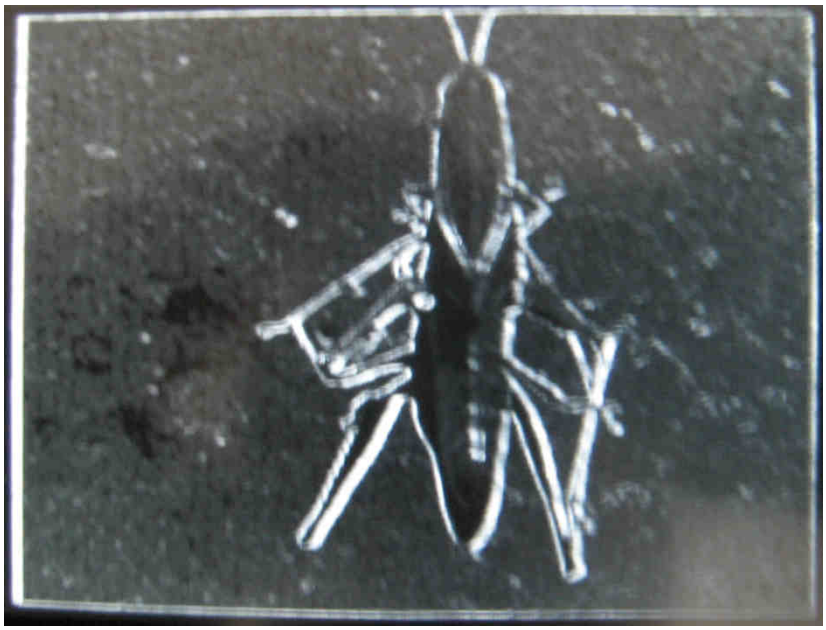


Figura 6.11 – Realce de contorno resultante da arquitetura da Figura 6.10.

6.4 – Convolução

Implementando a arquitetura da Figura 6.13, com filtro de realce de linha horizontal e uma binarização, obteve-se a imagem da Figura 6.12.

Figura 6.12 – Realce e binarização de imagem.

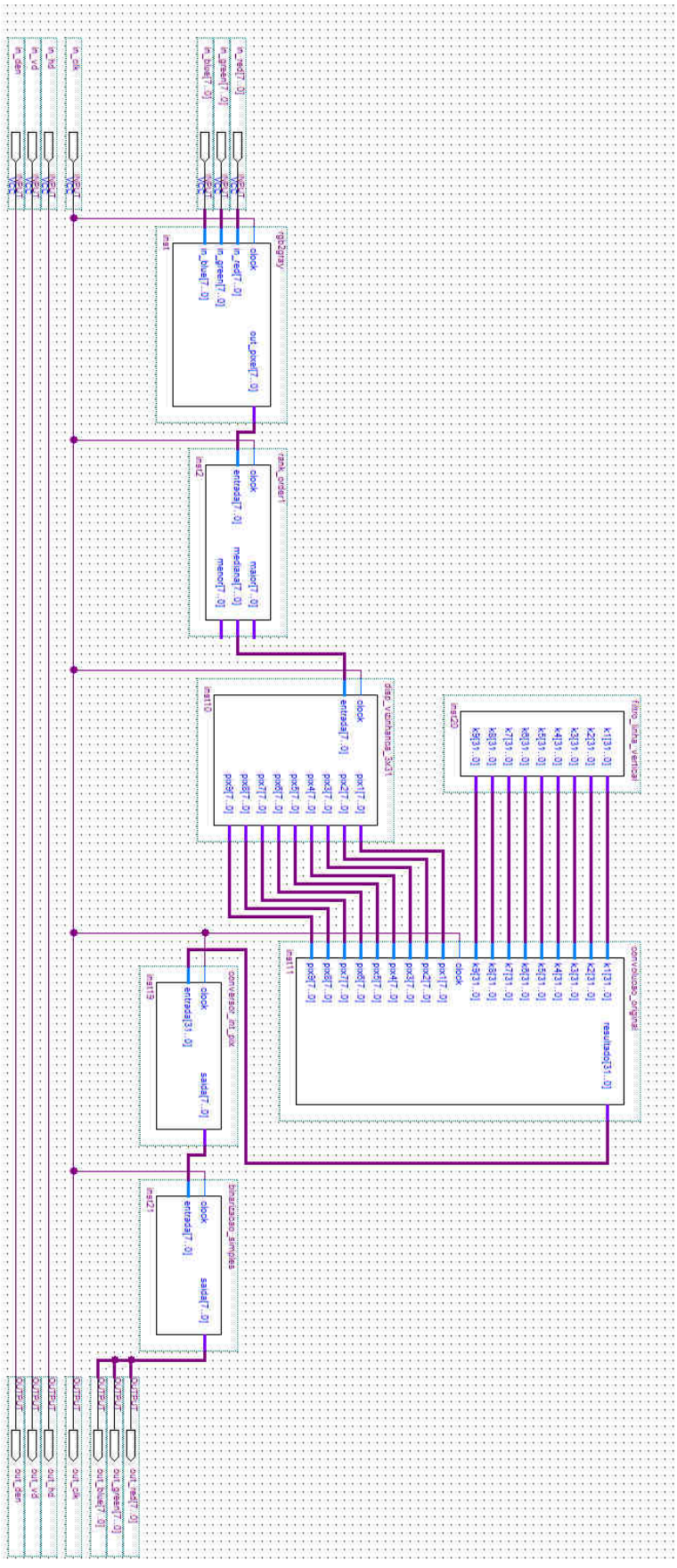


Figura 6.13 – Arquitetura para obtenção da imagem da Figura 6.12.

6.5 – Outros filtros para realce de bordas

Os filtros mais comuns para realce de bordas (Sobel, Prewitt, Roberts e Laplaciano) baseiam-se em derivadas parciais nas direções X e Y. Alguns outros filtros, considerados melhores na literatura de processamento de imagens, não são geralmente implementados por limitações de processamento. Como já discutido, em processadores de arquitetura comum, deve-se primeiramente calcular o gradiente em uma direção, armazenar o resultado em alguma memória, calcular o gradiente na outra direção e efetuar alguma operação para fusão dos dois gradientes (soma de módulos, por exemplo). Como mostrado no capítulo 3, a implementação desses algoritmos em hardware permite que os gradientes em X e em Y sejam calculados em paralelo, simultaneamente, reduzindo bastante o tempo de processamento.

Dois desses filtros são conhecidos como Operador de Kirsch (Figura 6.14) e Operador de Robinson (Figura 6.15). Ambos esses operadores baseiam-se na convolução da imagem com um conjunto de oito máscaras separadamente (como no caso do gradiente). O pixel resultante é tomado como a maior resposta do conjunto de oito. A implementação em software tomaria um tempo considerável de processamento, além da necessidade de armazenamento temporário dos resultados dessas oito máscaras. Com a arquitetura proposta na Figura 6.16, é possível efetuar o cálculo da convolução em paralelo para as oito máscaras, aumentando o consumo de recursos, porém diminuindo consideravelmente o tempo de processamento.

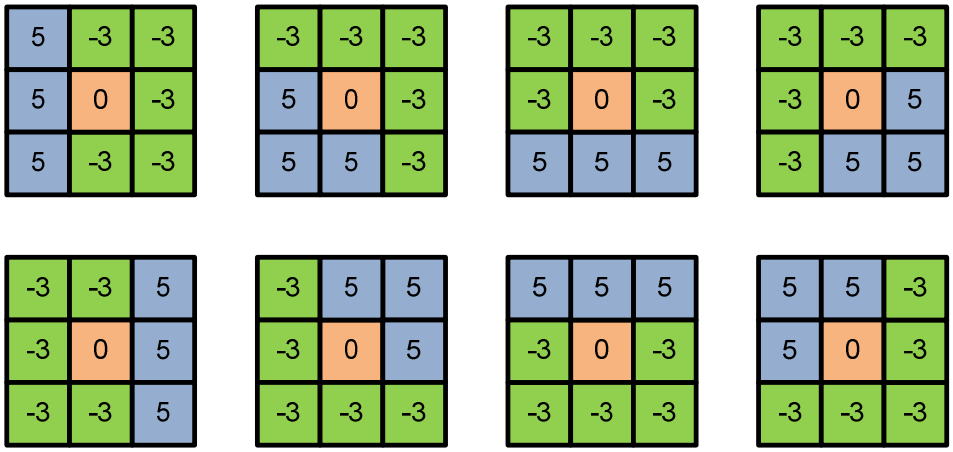


Figura 6.14 – Máscaras para o Operador de Kirsch.

1	0	-1
2	0	-2
1	0	-1

0	-1	-2
1	0	-1
2	1	0

-1	-2	-1
0	0	0
1	2	1

-2	-1	0
-1	0	1
0	1	2

-1	0	1
-2	0	2
-1	0	1

0	1	2
1	0	-1
-2	-1	0

1	2	1
0	0	0
-1	-2	-1

2	1	0
1	0	-1
0	-1	-2

Figura 6.15 – Máscaras para o Operador de Robinson.

As os operadores de Kirsch e Robinson também são baseados em derivadas parciais, porém em uma quantidade maior de direções, de modo a melhorar a detecção das bordas.

A Figura 6.16 mostra uma imagem de teste colorida. A Figura 6.17 fornece o resultado da aplicação dos filtros de Sobel. Na Figura 6.18 é mostrado o resultado da aplicação do operador de Kirsch, onde se podem observar detalhes finos da parte posterior do inseto. A Figura 6.19 é o resultado de uma operação de binarização e a Figura 6.20 é a inversão da figura 6.19, seguida de operações de dilatação e erosão.

Figura 6.16 – Imagem de teste.

Figura 6.17 – Resultado da aplicação do filtro de Sobel.

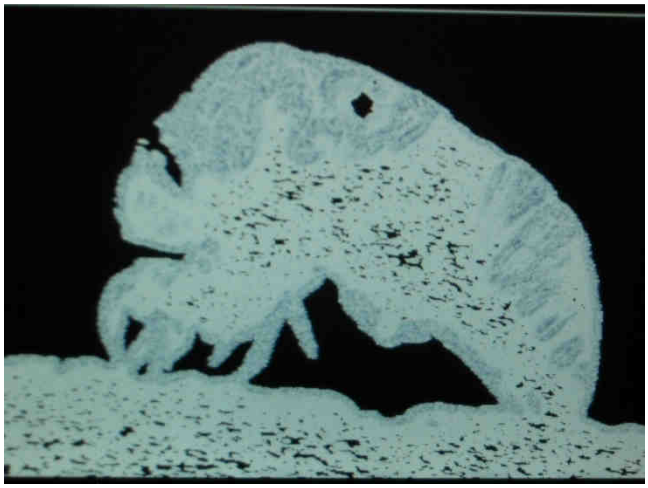


Figura 6.18 – Resultado da aplicação do operador de Kirsch.

Figura 6.19 – Binarização da imagem da Figura 6.15.

Figura 6.20 – Inversão da imagem da Figura 6.16.

A Figura 6.21 mostra a arquitetura implementada com o operador de Kirsch, onde é destacado o paralelismo entre as operações com as oito máscaras.

6.6 – Conclusões do capítulo

Neste capítulo foram mostradas alguns exemplos práticos de utilização das arquiteturas propostas ao longo do trabalho. A facilidade de implementação de sistemas distintos simplesmente pela especificação de um diagrama de blocos é uma vantagem que se destaca no processo de testes.

Os exemplos mostraram a viabilidade das implementações feitas, bem como a versatilidade e escalabilidade das arquiteturas. A flexibilidade do sistema permite a exploração de técnicas de processamento que em software geralmente não seriam utilizadas.

A plataforma de hardware utilizada mostra ser bastante útil para testes de aquisição, processamento e visualização de imagens em tempo real.

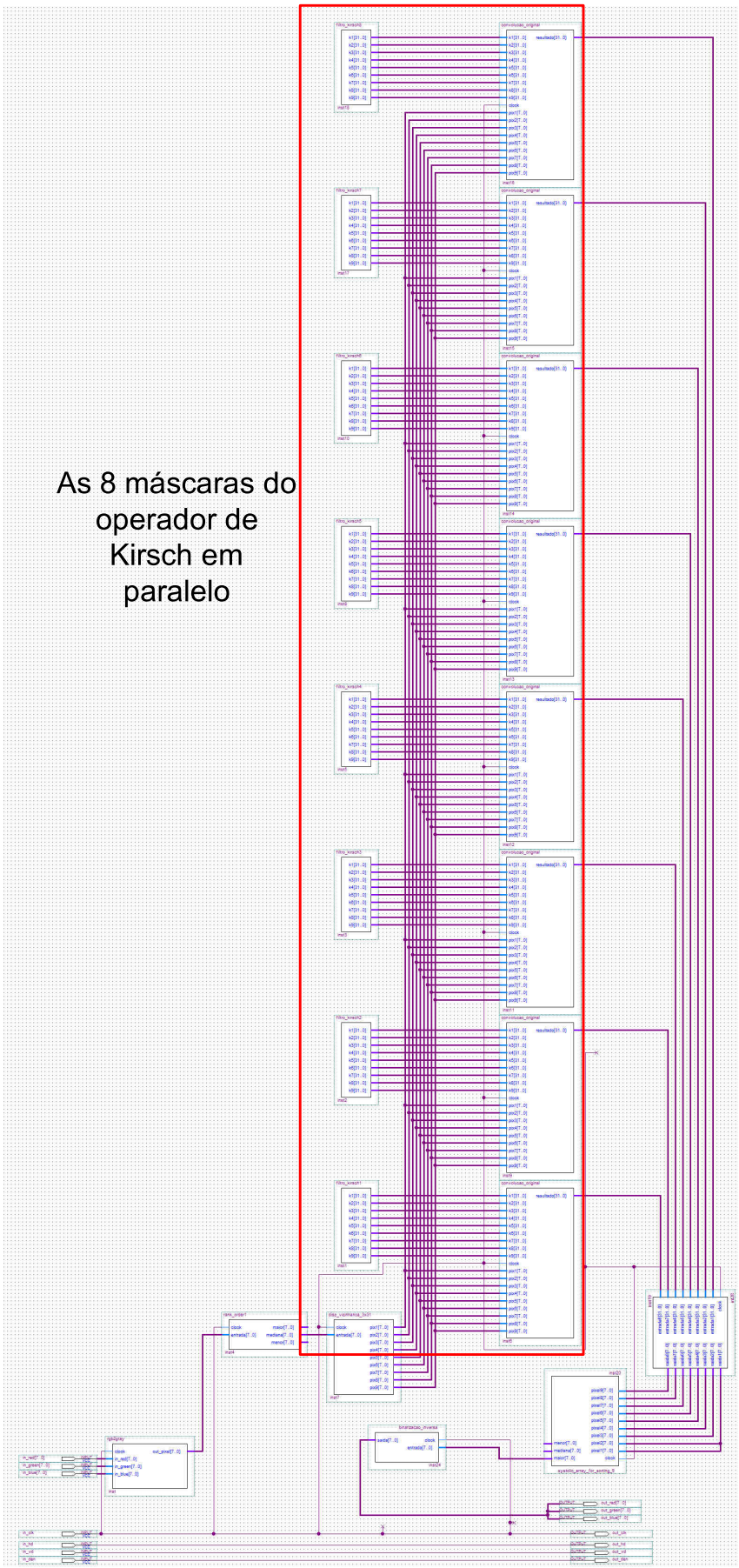


Figura 6.21 – Arquitetura para o operador de Kirsch, com as oito máscaras em paralelo.

7 – CONCLUSÕES E PROPOSTAS DE TRABALHOS FUTUROS

7.1 – Comentários Finais e Conclusões

Neste trabalho foi implementada uma plataforma completa para captura, processamento e visualização de imagens digitais. Todo o sistema foi embarcado em um dispositivo FPGA, de modo a permitir a exploração de diversas arquiteturas de hardware para processamento de imagens.

Alguns dos principais algoritmos de processamento de imagens foram analisados (no domínio espacial) para identificação de paralelismo de dados e de instruções, como forma de extraírem-se características que permitissem a proposição de arquiteturas diferentes e específicas para cada algoritmo.

Identificou-se uma característica comum a algoritmos que operam em vizinhanças, que é a etapa de disponibilização dessa vizinhança. Uma arquitetura específica para o carregamento de vizinhanças em forma de *pipeline* foi proposta, discutida e implementada com sucesso, permitindo, após um período de latência inicial, a disponibilização de toda uma nova vizinhança a cada ciclo de *clock*.

Houve uma preocupação com o consumo de recursos de hardware das arquiteturas, e uma análise de simetrias e redundâncias no processo de convolução foi realizada. As máscaras mais comuns em processamento de imagens foram analisadas e agrupadas segundo características de simetria, de modo a identificarem-se arquiteturas próprias para cada grupo, com o intuito de diminuir-se a utilização de multiplicadores internos do FPGA. O uso do paralelismo, aliado à técnica de superposição das máscaras, foi testado em simulação e teve resultados positivos de síntese, os quais mostram que o método é uma boa alternativa para a otimização de uso de recursos, e também para o aumento do troughput do sistema. Porém, a implementação dessas novas arquiteturas é mais complexa, pois exige um controle mais sofisticado de sincronização entre os processos.

A análise detalhada das seqüências de operações de cada algoritmo selecionado levou à separação desses algoritmos em etapas, permitindo a transcrição desses algoritmos em arquiteturas que exploram eficazmente suas características de paralelismo em um pipeline de operações.

Complexos algoritmos de segmentação de imagens, geralmente não implementáveis em software por questão de tempo de processamento, foram implementados de forma simples e de fácil visualização, mostrando a eficácia da utilização de hardware reconfigurável para esses casos.

A interface gráfica do Quartus II permite a fácil e rápida montagem das seqüências de algoritmos utilizando blocos simples, em uma estrutura semelhante a um fluxograma de processos. Essa simplicidade de implementação e visualização permite que testes sejam feitos de forma rápida quando do desenvolvimento de novos blocos de processamento.

Analisando os resultados das imagens pode-se constatar a aplicabilidade de FPGAs para processamento de imagens. Da mesma maneira, pode ser observado que FPGAs podem ser usadas para a implementação de sensores de visão inteligentes os quais entreguem imagens processadas para sistemas de análises de informação, sistemas especialistas, sistemas de inspeção (supervisórios) SCADA (*Supervisory Control And Data Acquisition*), entre outros.

Adicionalmente, foi feito um estudo sobre o desempenho de máscaras com diferentes configurações (em tamanho e forma de paralelismo), procurando maximizar o desempenho e minimizar o uso de multiplicadores (vide capítulo 5). Neste contexto, teve-se a preocupação de estudar de maneira cuidadosa aspectos da implementação de técnicas de processamento de imagens no domínio espacial em FPGAs.

7.2 – Propostas de trabalhos futuros

7.2.1 – Biblioteca parametrizada de processamento de imagens

A primeira proposta de continuação deste trabalho é a implementação de mais módulos de processamento de imagens, para a exploração de diversos outros algoritmos. A parametrização desses módulos é importante para facilitar a criação de outros mais complexos. A grande regularidade dos algoritmos de processamento de imagens permite a fácil identificação dos parâmetros (tamanhos de imagens, profundidade de cores, tamanhos de máscaras, etc.).

7.2.2 – Extensão do sistema ao processamento no domínio da freqüência

Este trabalho focou no desenvolvimento das arquiteturas para processamento no domínio espacial. As técnicas no domínio da freqüência exploram ferramentas matemáticas bastante

poderosas, permitindo o projeto facilitado de filtros e outros processos. A análise de características de paralelismo de dados e de instruções deve ser feita de modo a identificarem-se estruturas mais comuns entre os algoritmos, facilitando a criação das arquiteturas.

7.2.3 – Processamento de Vídeos

Uma demanda muito grande pelo processamento de vídeos surgiu com o advento da Televisão Digital no Brasil. A utilização de memórias externas para o armazenamento de seqüências de imagens, permitiria a exploração de técnicas de processamento de vídeos, tais como detecção de movimentos, compressão de dados, entre outras.

7.2.4 – Estudo formal de corretude das técnicas utilizadas

Um estudo mais formal da corretude das técnicas deste trabalho poderia ser feito utilizando-se uma ferramenta de prova chamada PVS (*Proof Verification System*). Tal ferramenta baseia-se na descrição formal dos algoritmos e na prova de seus resultados. Durante o desenvolvimento deste trabalho foi feito um teste quanto à corretude da arquitetura de convolução comum, mostrando que a implementação feita nos trabalhos de (Wong,2005) calcula de modo correto a convolução entre a imagem e a máscara utilizada.

Essas técnicas de especificação e prova das arquiteturas pode levar ao desenvolvimento de sistemas CAD que permitam a otimização de descrições de hardware com especificações de consumo de energia, de recursos, etc.

REFERÊNCIAS BIBLIOGRÁFICAS

(Akita,2003) – “An image sensor with fast objects position extraction function” – Akita, J.; IEEE Transactions on electron Devices, Nova Iorque, 2003.

(Aragão,1998) – “Uma arquitetura sistólica para solução de sistemas lineares implementada com circuitos FPGAs” – Aragão, A.C.O.S.; Dissertação de Mestrado em Ciências; ICMC - Instituto de Ciências Matemáticas e de Computação, USP, 1998.

(Becker&Hartenstein,2003) - “Configware e morphware going mainstream” - Becker, J.; Hartenstein, R.W.; Journal of Systems Architectures, 2003.

(Bräunl,2001) – “Parallel Image Processing” – Bräunl, Thomas; Editora Springer Verlag, 2001

(Eyre&Bier,1998) – “DSP Processors hit the mainstream” – Eyre, J.; Bier, J.; Berkley Design Technology Inc.; CA, USA, 1998.

(Franco,2007) – “Sincronização, captura e análise de imagens da poça de soldagem no processo GMAW convencional, no modo de transferência metálica por curto-circuito” – Franco, Luciano D.N.; Dissertação de Mestrado em Sistemas Mecatrônicos; Universidade de Brasília, 2007.

(Gokhale et.al.,2005) - “Reconfigurable Computing – Accelerating Computation with Field-Programmable Gate Arrays” - Gokhale, M.B.; Graham, P.S.; Editora Springer, 2005

(Gonzalez&Woods,2000) - “Processamento de Imagens Digitais” - Gonzalez, R.C.; Woods, R.E.; Editora Edgard Blücher, 2000

(Hartenstein,2006) – “Why we need Reconfigurable Computing Education” – Hartenstein, R.; abertura do 1st International Workshop on Reconfigurable Computing Education – WRCE2006, Alemanha, 2006.

(Jacobi et.al, 2005) – “Reconfigurable systems for sequence alignment and for general dynamic programming” – Jacobi, R.P.; Ayala-Rincón, M.; Carvalho, L.G.; Quintero, C.H.L.; Hartenstein, R.W. – Genetics and Molecular Research, São Paulo, Brasil, v.4, n.3, p. 543-552, 2005.

(Hongtu,2007) – “Design Issues in VLSI Implementation of Image Processing Hardware Accelerators – Methodology and Implementation” – Hongtu, J; Electroscience Department – Lund University – Sweden; Doctoral Thesis, 2007

(Júnior,2006) – “Seleção de variáveis e características como aplicação paralela para cluster MPI” – Júnior, F.P.; Dissertação de Mestrado em Ciência da Computação, Universidade Estadual de Maringá, 2006.

(Kagami,2002) – “A real-time visual processing system using a general-purpose vision chip” – Kagami, S.; ICRA - IEEE International Conference on Robotics and Automation, 2002.

[Ke04] – “Fast image interpolation for motion estimation using graphics hardware” – Kelly, F.; Kokaram, A.; Proceedings of SPIE The International Society for Optical Engineering; 1999.

(Kehtarnavaz&Gamadia,2006) – “Real-Time Image and Video Processing: From Research to Reality” – Kehtarnavaz, N; Gamadia, M; Editora Morgan & Claypool, 2006

(Murata,1998) – “Image processing LSI ‘ISP-IV’ based on local parallel architecture and its applications”; ICIP – International Conference on Image Processing, 1998.

(Patterson&Hennessy,2005) – “Organização e Projeto de Computadores – a interface hardware/software ” – Patterson, D.; Hennessy, J.; Editora Campus-Elsevier, 1ª. Edição, 2005.

(Pedrini,2008) – “Análise de Imagens Digitais – Princípios, Algoritmos e Aplicações” – Pedrini, Hélio ; Schwarz, W. R.; Editora Thomson, 2008

(Porter,2001) – “Evolution on FPGAs for Feature Extraction”; Porter, R.; PhD thesis on Information Technology; Queensland University of Technology, 2001.

(De Rose et.al., 2008) – “Arquiteturas Paralelas” – De Rose, C.A.F.; Navaux, P.O.A.; Editora Bookman, 2008.

(Vahid,2007) – “Sistemas Digitais – Projeto, Otimização e HDLs” – Vahid, F.; 1ª. Edição, Editora Bookman, 2007.

(Wong,2005) – “Fast 2D Convolution using Reconfigurable Computing”, - Wong, S.; Jasiunas, M.; Kearney, D.; The Eight International Symposium on Signal Processing and its Applications (ISSPA), 2005.