IMPLEMENTAÇÃO DO ALGORITMO DE RICHARDSON-LUCY EM ARQUITETURAS RECONFIGURÁVEIS APLICADO AO PROBLEMA DE BORRAMENTO DE IMAGENS

OSCAR EDUARDO ANACONA MOSQUERA

DISSERTAÇÃO DE MESTRADO EM SISTEMAS MECATRÔNICOS
DEPARTAMENTO DE ENGENHARIA MECÂNICA

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

IMPLEMENTAÇÃO DO ALGORITMO DE RICHARDSON-LUCY EM ARQUITETURAS RECONFIGURÁVEIS APLICADO AO PROBLEMA DE BORRAMENTO DE IMAGENS

OSCAR EDUARDO ANACONA MOSQUERA

Orientador: Carlos Humberto Llanos Quintero

Coorientador: Daniel Mauricio Muñoz Arboleda
Universidade de Brasília

DISSERTAÇÃO DE MESTRADO EM SISTEMAS MECATRÔNICOS

Publicação: ENM.DM-85/15

BRASÍLIA-DF, 11 de Março de 2015

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

IMPLEMENTAÇÃO DO ALGORITMO DE RICHARDSON-LUCY EM ARQUITETURAS RECONFIGURÁVEIS APLICADO AO PROBLEMA DE BORRAMENTO DE IMAGENS

OSCAR EDUARDO ANACONA MOSQUERA

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO REQUISITO PARCIAL PARA A OBTENÇÃO DO GRAU DE MESTRE EM SISTEMAS MECATRÔNICOS.

Banca Examinadora

Prof. Carlos Humberto Llanos Quintero, Dr. (ENM-UnB)
(Orientador)

Profa. Mylène Christine Queiroz de Farias, PhD. (ENE-UnB)
(Examinadora Externa)

Prof. Flávio de Barros Vidal, Dr. (CIC-UnB)
(Examinador interno)

BRASÍLIA-DF, 11 de Março de 2015

FICHA CATALOGRÁFICA

ANACONA MOSQUERA, OSCAR EDUARDO

Implementação do algoritmo de Richardson-Lucy em arquiteturas reconfiguráveis aplicado ao problema de borramento de imagens [Distrito Federal] 2015.

xiv, 76p. 210 × 297 mm (ENM/FT/UnB, Mestre, Sistemas Mecatrônicos, 2015). Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.

Departamento de Engenharia Mecânica.

1. Processamento de Imagens	2. FPGAs
3. Restauração de Imagens	4. Algoritmo Richardson-Lucy
5. Índice de similaridade estrutural	6. Sistemas embarcados
I. ENM/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

ANACONA-MOSQUERA, OSCAR. (2015). Implementação do algoritmo de Richardson-Lucy em arquiteturas reconfiguráveis aplicado ao problema de borramento de imagens. Dissertação de Mestrado em Sistemas Mecatrônicos, Publicação ENM.DM-85/15, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 76p.

CESSÃO DE DIREITOS

AUTOR: Oscar Eduardo Anacona Mosquera.

TÍTULO: IMPLEMENTAÇÃO DO ALGORITMO DE RICHARDSON-LUCY EM ARQUITETURAS RECONFIGURÁVEIS APLICADO AO PROBLEMA DE BORRAMENTO DE IMAGENS.

GRAU: Mestre **ANO**: 2015

É concedida à Universidade de Brasília permissão para reproduzir cópias desta dissertação de mestrado e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte dessa dissertação de mestrado pode ser reproduzida sem autorização por escrito do autor.

Oscar Eduardo Anacona Mosquera

SCRN 704/5 Bloco G ent 34, Apto 200

70730-670 Brasília-DF-Brasil

Dedicatória

À Cynthia;

À minha mãe querida e meu querido pai;

Às minhas irmãs e os meus irmãos.

OSCAR EDUARDO ANACONA MOSQUERA

Agradecimentos

Ao meu orientador, o professor Carlos Humberto Llanos pela amizade, apoio, confiança, dedicação, paciência e orientação. Por ter me permitido trabalhar ao seu lado, aprendendo um pouco mais a cada dia.
Aos meus colegas e amigos: Laura, Ludmyla, Janier, Renato, Luis, Jones, Daniel, Camilo, Sergio, Milton, Herlys, Liz Carolina, Gloria Liliana, Miguel, Eduardo, Willian, William, Miltão e demais, pela amizade, ajuda e colaboração no desenvolvimento deste trabalho.
À FUNAPE (Fundação de Apoio e Pesquisa) e Petrobras pelo suporte financiero a este trabalho.
Ao Grupo de Automação e Controle (GRACO/UnB) e a todos os meus professores pelo suporte e formação acadêmica.

OSCAR EDUARDO ANACONA MOSQUERA

"...conversamos sobre meu gosto por nuvens e sobre como a maioria das pessoas perdeu a habilidade de ver o lado bom das coisas, embora a luz por trás das nuvens seja uma prova quase diária de que ele existe."

Matthew Quick

(O lado bom da vida)

RESUMO

Este trabalho apresenta a implementação em *hardware* de um algoritmo para a restauração para imagens que tenham sofrido degradação por movimento relativo entre a câmera e a cena (*motion blur*). O borramento da imagem é modelado matematicamente com o processo de *convolução* entre a função de degradação (*Point Spread Function*-PSF) e a imagem real, sendo a restauração da imagem real o processo inverso (*deconvolução*). O algoritmo de restauração implementado neste trabalho é conhecido como algoritmo de Richardson-Lucy (*RLA*). Neste caso, implementou-se o RLA em uma plataforma *hardware* FPGA (*Field Programmable Gate Array*) usando a linguagem de descrição de *hardware* VHDL (*Very Hight Description Language*), assumindo ausência de ruído aditivo no sistema de captura da imagem. A metodologia para avaliar a plataforma consistiu em simular a arquitetura projetada no ModelSim, fornecendo como dados de entradas as imagens degradadas. A degradação das imagens foi obtida usando a funções *fspecial* e *imfilter* do *Matlab*, as quais permitiram simular o borramento de imagens por movimentos da câmera (deslocamento e ângulo). Adicionalmente, a avaliação da qualidade das imagens coletadas foi realizada usando a métrica SR-SIM (*Spectral Residual Based Similarity*), assim como executando uma verificação visual das mesmas. O sistema implementado fornece um pixel processado por cada ciclo de relógio da FPGA, depois de um tempo de latência, sendo 12.425 vezes mais rápido que o mesmo algoritmo implementado no processador NIOS II. Adicionalmente foram feitas comparações rodando o algoritmo em um PC Intel Core-i3 a 3,2GHz. Neste caso, a implementação do algoritmo foi realizada usando a biblioteca OpenCV. Resultados de simulações e testes com imagens reais são apresentados para dar suporte à aplicabilidade em vídeo.

ABSTRACT

This work presents the hardware implementation of an image restoration algorithm, in which the images are blurred by relative motion between camera and the scene. The blurred image process is mathematically modeled by a convolution process between the original image and the point-spread function (PSF) of the blurring system, being the image restoration the inverse process (a deconvolution process). The restoration algorithm that was implemented in this work is known as Richardson-Lucy (RLA) algorithm. In this case the RLA was implemented in an FPGA-based platform using the hardware description language VHDL (Very Hight Description Language), and assuming the absence of additive noise in the capturing image system. The methodology for evaluating the platform consists of simulating the designed architecture in the ModelSim platform, providing as data input the blurred images. The blurring process of the images was achieved by using the Matlab functions *fspecial* e *imfilter*, which allowed the simulation of blurred images by camera movements (displacement and angle). Additionally, the quality evaluation of the collected images was achieved using the SR-SIM (Spectral Residual Based Similarity) metric as well as by a visual verification of the images. The implemented system provides a processed pixel per clock cycle of the FPGA, after a latency time, being 12.425 times faster than the same algorithm implemented in software (running in the NIOS processor at 100 MHz). Additionally, comparisons have being done by running the same algorithm in a PC Intel Core-i3 with 3,2GHz. In this case, the algorithm implementation was developed using the OpenCV library. The results of simulations and respective testing with real images are also presented in order to give support to video applications.

Sumário

Lista de Figuras ... **xi**

Lista de Tabelas ... **xii**

1 INTRODUÇÃO ... **1**

1.1 CONTEXTUALIZAÇÃO ... 1

1.2 DEFINIÇÃO DO PROBLEMA E MOTIVAÇÕES ... 3

1.3 Objetivos ... 4

1.3.1 Objetivo Geral ... 4

1.3.2 Objetivos específicos ... 4

1.4 ASPECTOS METODOLÓGICOS DO TRABALHO ... 4

1.5 CONTRIBUIÇÕES DO TRABALHO ... 5

1.6 ESTRUTURA DO TEXTO ... 5

2 PLATAFORMAS PARA O PROCESSAMENTO DE IMAGENS E VÍDEO **6**

2.1 ARQUITETURAS PROGRAMÁVEIS VIA *SOFTWARE* ... 7

2.1.1 GPPs-*General Purpose Processors* ... 7

2.1.1.1 Arquiteturas do tipo *Von Neumann* ... 8

2.1.1.2 Barreira de memória (*The memory wall*) ... 9

2.1.1.3 Barreira de potência (*The power wall*) ... 9

2.1.1.4 Barreira do paralelismo no nível de instrução (*Instruction-level parallelism wall*) ... 10

2.1.2 Processadores Digitais de Sinais (DSPs) ... 10

2.1.3 Unidades de Processamento Gráfico (GPUs) ... 11

2.2 CIRCUITOS INTEGRADOS DE APLICAÇÃO ESPECÍFICA (ASICs) ... 12

2.3 ARQUITETURAS RECONFIGURÁVEIS ... 13

2.3.1 FPGAs-*Field Programmable Gate Arrays* ... 13

2.3.1.1 Descrição geral da família Cyclone IV-E ... 16

2.4 CONCLUSÃO DO CAPÍTULO ... 17

3 ALGORITMOS DE PROCESSAMENTO E RESTAURAÇÃO DE IMAGENS **18**

3.1 IMAGENS DIGITAIS 18
3.2 TIPOS DE OPERAÇÕES EM UMA IMAGEM 19
3.2.1 Cadeia geral de um sistema de visão computacional 20
3.3 TRANSFORMAÇÕES ESPACIAIS 21
3.3.1 Filtragem por convolução 21
3.3.1.1 Complexidade computacional 22
3.3.1.2 Exemplos de aplicação 23
3.3.2 Função de espalhamento de ponto (PSF) 25
3.4 RESTAURAÇÃO DE IMAGENS 26
3.4.1 Algoritmo Richardson-Lucy 27
3.5 MÉTRICAS DE QUALIDADE EM IMAGENS 28
3.5.1 *Structural similarity index* (SSIM) 29
3.6 TRABALHOS CORRELATOS À IMPLEMENTAÇÃO DE RESTAURAÇÃO DE IMAGENS EM *HARDWARE* 32
3.7 CONCLUSÕES DO CAPÍTULO 36

4 IMPLEMENTAÇÃO DO ALGORITMO PARA A RESTAURAÇÃO DE IMAGENS 37
4.1 CONSIDERAÇÕES INICIAIS 37
4.1.1 Função de espalhamento de ponto (PSF) 37
4.2 IMPLEMENTAÇÃO DO ALGORITMO RICHARDSON-LUCY EM *SOFTWARE* 41
4.3 IMPLEMENTAÇÃO DO ALGORITMO RICHARDSON-LUCY EM *HARDWARE* 42
4.3.1 Unidade de redução de cores 43
4.3.2 Unidade de filtragem por convolução 44
4.3.2.1 Arquitetura de vizinhança 46
4.3.2.2 Arquitetura de disponibilização da vizinhança 47
4.3.2.3 Arquitetura de convolução 49
4.3.3 Unidades de operações aritméticas 50
4.4 VERIFICAÇÃO FUNCIONAL E AVALIAÇÃO DE QUALIDADE 50
4.5 VALIDAÇÃO DA ARQUITETURA 52
4.6 CONCLUSÃO DO CAPÍTULO 54

5 TESTES DE IMPLEMENTAÇÃO E RESULTADOS 55
5.1 ARQUITETURA DO ALGORITMO RICHARDSON-LUCY 55
5.2 SIMULAÇÃO COMPORTAMENTAL 57
5.2.1 Redução de cores 58
5.2.2 Filtragem por convolução 59
5.2.3 Algoritmo Richardson-Lucy 60

5.3 VALIDAÇÃO DA ARQUITETURA 67
5.4 CONCLUSÕES DO CAPÍTULO 68

6 Conclusões e Trabalhos Futuros 70
6.1 Comentários Finais e Conclusões 70
6.2 Propostas de Trabalhos Futuros 71

REFERÊNCIAS BIBLIOGRÁFICAS 72

Lista de Figuras

1.1 Restauração de uma imagem degradada por movimento [3] ... 2

2.1 As Tendências do processador de Kathy Yelick [16] ... 8
2.2 Diferencia de desempenho entre os processadores e a memoria ao longo do tempo [17] 9
2.3 Estrutura do Sensor [21] ... 11
2.4 Estrutura geral de um FPGA [23] ... 14
2.5 Arquitetura de interconexão de um FPGA [23] ... 14
2.6 Etapas de um projeto com FPGAs [25] ... 15

3.1 Representação de imagens digitais, adaptado de [19] ... 19
3.2 (a) Cadeia de processamento e (b) redução na quantidade de dados [19] ... 20
3.3 Deslocamento da máscara para o filtro espacial ... 22
3.4 Operação de janelamento na borda da imagem de entrada ... 23
3.5 Exemplos de filtros passa baixas:(a) vertical, (b) horizontal, (c) diagonal $+45^o$ e (d) diagonal -45° ... 24
3.6 Filtragem espacial passa-baixa ... 24
3.7 Exemplos de filtros passa-altas:(a) vertical, (b) horizontal, (c) diagonal $+45^o$ e (d) diagonal -45° ... 25
3.8 Filtragem espacial passa-alta ... 25
3.9 Diagrama esquemático da função de dispersão de ponto [33] ... 26
3.10 Comparação de métricas de qualidade para imagens distorcidas [39]. (a) Einstein original. MSE=0 e SSIM=1, (b) Einstein distorcido com mudança de luminância. MSE=309 e SSIM=0,987, (c) Einstein distorcido com ruído Gaussiano. MSE=309 e SSIM=0,576, (d) Einstein distorcido por borramento. MSE=308 e SSIM=0,641 . 29
3.11 Diagrama esquemático da medida do índice de similaridade estrutural (SSIM) [38] 30
3.12 Imagem degradada por movimento espacialmente variante [35] ... 34

4.1 Exemplo do *loop unrolling* do RLA. (a) Implementação em *software* e (b) implementação em *hardware* ... 38

4.2 Configuração das máscaras para a PSF. (a) Primeira abordagem e (b) Segunda abordagem 39
4.3 Preenchimento dos valores da PSF para uma máscara global de 9×9 pixels. (a) PSF para deslocamentos de $X = 5$ pixels, (b) PSF para deslocamentos de $X = 3$ pixels e (c) máscara global 40
4.4 Arquitetura geral do RLA 43
4.5 Unidade de conversão de RGB para escala de cinza 45
4.6 Filtragem por convolução 46
4.7 Arquitetura para a operação de janelamento para uma imagem de 800×525 pixels 46
4.8 Arquitetura para a disponibilização dos pixels entre linhas da imagem e entre *frames* do vídeo 48
4.9 Exemplo dos casos para disponibilizar 49
4.10 Arquitetura para a convolução 49
4.11 Vinculação dos valores da PSF à memória ROM 50
4.12 Tela do *MegaWizard Plug-In Manager* para configuração dos blocos da multiplicação e a divisão 51
4.13 Diagrama de *testbench* 51
4.14 Transformação de uma imagem bidimensional em um arranjo unidimensional 52
4.15 Arquitetura geral do sistema 53
4.16 Kit de desenvolvimento modelo VEEK-MT, composto principalmente pela câmera e o LCD 53

5.1 Metodologia da verificação comportamental do algoritmo, usando ModelSim 57
5.2 Resultados da verificação comportamental e avaliação da qualidade para trânsito 1, usando como referência as imagens obtidas de *Matlab*. (a) Imagem original, (b) Zoom da imagem em escala de cinza e (c) Zoom da imagem distorcida por borramento 58
5.3 Resultados da verificação comportamental e avaliação da qualidade para tartaruga, usando como referência as imagens obtidas de *Matlab*. (a) Imagem original, (b) Zoom da imagem em escala de cinza e (c) Zoom da imagem distorcida por borramento 59
5.4 Resultados da verificação comportamental e avaliação da qualidade para sapo, usando como referência as imagens obtidas de *Matlab*. (a) Imagem original, (b) Zoom da imagem em escala de cinza e (c) Zoom da imagem distorcida por borramento 59

5.5 Resultados da verificação comportamental e avaliação da qualidade para trânsito 2, usando como referência as imagens obtidas de *Matlab*. (a) Imagem original, (b) Zoom da imagem em escala de cinza e (c) Zoom da imagem distorcida por borramento 60
5.6 Zoom da sequência de restauração para trânsito 2 61
5.7 SR-SIM para várias imagens restauradas em *hardware* comparadas com a imagem obtida em *Matlab* 62
5.8 SR-SIM para as imagens restauradas em *Matlab* e *hardware* comparadas com a imagem sem borramento 63
5.9 Restauração das imagens em *hardware*. (a), (d), (g) e (j) Imagens originais. (b), (e), (h) e (k) Imagens borradas. (c), (f), (i) e (l) Imagens restauradas 65
5.10 Desempenho do algoritmo rodando em *software* (usando um código C++, e OpenCV em um PC a 3.2 GHz) e rodando na arquitetura proposta 66
5.11 Desempenho do algoritmo rodando em *software* (usando um código C em um PC a 3.2 GHz e em NIOS II a 100 MHz) e rodando na arquitetura proposta 66
5.12 Validação da arquitetura do RLA. 69

Lista de Tabelas

3.1 Carga computacional para a convolução de uma imagem em termos das operações aritméticas [32] ... 23

3.2 Comparação das diferentes implementações do RLA ... 35

4.1 Alguns filtros gerados usando a função *fspecial* do *toolbox* de *Matlab* ... 38

5.1 Recursos de *hardware* para a arquitetura RLA proposta na Cyclone IV-E Altera ... 56

5.2 Recursos de *hardware* para a arquitetura RLA proposta na Stratix V da Altera ... 56

5.3 Resultados da simulação para a arquitetura de convolução ... 60

5.4 SR-SIM para as imagens restauradas em *Matlab* e *hardware* comparadas com a imagem sem degradação ... 62

5.5 Resultados da simulação para a arquitetura de restauração ... 64

5.6 Desempenho do algoritmo rodando em *software* (C++) e em *hardware* para diferentes tamanhos de máscara ... 67

LISTA DE SÍMBOLOS

ASIC	Application Specific Integrated Circuits
ALU	Unidade Aritmética e Lógica
ASP	Application Specific Processors
CLB	Configurable Logic Block
CPLD	Complex Programmable Logic Devices
CPU	Central Processing Unit
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
GPU	Graphics Processing Units
GPGPU	General Purpose Processing on a Graphics Processing Unit
HW/SW	Hardware/Software
IOB	Input Output Block
LE	Logic Element
LUT	Look-Up Table
ML	Maximization Likelihood Principle
MSB	Most Significant Bit
NRE	Non Recurring Costs
PC	Personal Computer
PE	Processing Element
PLD	Programmable logic device
PSF	Point Spread Function
RLA	Richardson-Lucy Algorithm
ROM	Read Only Memory

SR-SIM	Spectral Residual Based Similarity
SSIM	Structural Similarity Index
UAV	Unmanned Aerial Vehicles
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language

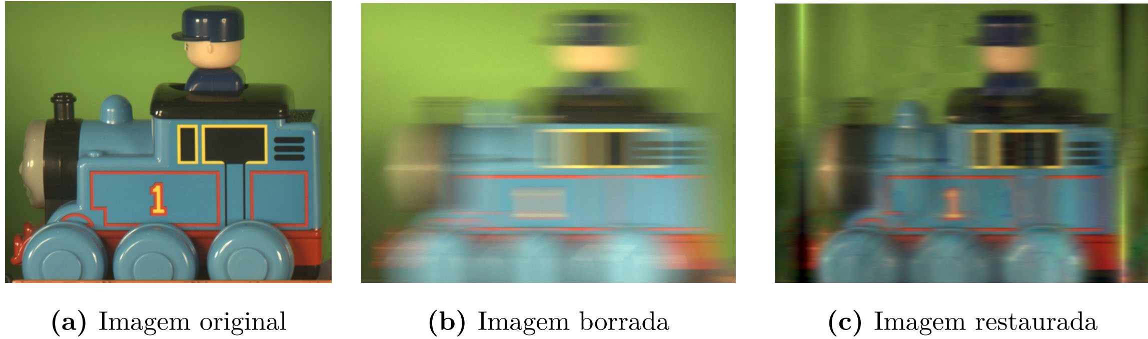
1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Algoritmos de restauração de imagens são amplamente utilizados para a recuperação de imagens que foram degradadas durante o processo de aquisição [1]. A utilização destes algoritmos gerou vários trabalhos de pesquisa na área de visão computacional, principalmente por causa do seu elevado custo computacional [1]. Um dos grandes desafios tecnológicos de nossa época encontra-se no desenvolvimento de técnicas de restauração de imagens que usem algoritmos velozes, que possam ser facilmente paralelizados e, portanto, factíveis de serem embarcados em um único *chip*. Estes algoritmos são ideais para as implementações em sistemas de visão que usem restauração de imagens em tempo real em aplicações como navegação de robôs terrestres, subaquáticos e aéreos (por exemplo em um Veículo Aéreo Não Tripulado-VANT), reconhecimento de padrões, microscopia, aprendizagem de máquina, aplicações em física nuclear e de partículas, entre outras [1, 2].

As degradações podem ocorrer por movimentos mecânicos rápidos da câmera durante a aquisição de imagens (*imaging*), em um tempo curto de exposição da cena e/ou pelo movimento de vários objetos numa cena (*motion blur*, vide Figura 1.1b); cada objeto podendo ter velocidades diferentes; efeitos de lente, abertura do diafragma da câmera(*shutter*) e/ou pelas condições ambientais [3, 4]. Do ponto de vista teórico, o borramento pode ser modelado como uma operação matemática denominada convolução em 2-D [1]. A convolução ocorre entre a imagem real e uma função que define a degradação, denominada de função de espalhamento de ponto (*Point Spread Function-PSF*) [1, 5]. O processo inverso da convolução, para obter a imagem original, é referido como deconvolução. A deconvolução é o processo matemático que reverte os efeitos da PSF, resultando na estimativa de uma imagem real (vide Figuras 1.1a e 1.1c) [5]. Geralmente, este processo faz parte de técnicas de *restauração de imagens*.

Este processo de restauração de imagens mediante o uso de técnicas de convolução apresenta um grande custo computacional, mas por sua vez, permite a obtenção de imagens com detalhes estruturais finos [4]. Por outro lado o mesmo processo de restauração pode ser desenvolvido (a) conhecendo previamente a PSF (*nonBlind deconvolution*), (b) ignorando completamente a



(a) Imagem original (b) Imagem borrada (c) Imagem restaurada

Figura 1.1. Restauração de uma imagem degradada por movimento [3]

PSF (*blind deconvolution*) ou (*c*) conhecendo parcialmente a PSF (*Myopic deconvolution*) [4, 6, 7]. O processo de restauração da imagem pode ser realizado em uma única etapa (método não iterativo) ou em várias etapas (método iterativo) [4]. Entre as abordagens possíveis para resolver o problema de restauração, os algoritmos iterativos apresentam uma melhor estimação da imagem real [6]. Dentro dos algoritmos iterativos, o algoritmo proposto por Richardson e Lucy (1974) (*Richardson-Lucy Algorithm-RLA*) é amplamente utilizado em astronomia e microscopia, destacando-se por ser simples e de fácil implementação [4, 7]. O RLA é derivado do princípio da máxima verossimilhança (*maximization likelihood principle ML*), assumindo um modelo de Poisson para a distribuição do ruído presente na imagem [5]. No entanto, o RLA converge lentamente (gerando um alto custo computacional), divergindo depois de um certo número de iterações fato este que amplifica o ruído durante cada iteração, como um efeito colateral do processo de restauração [1, 8].

Neste contexto, técnicas para a aceleração do RLA são discutidas em [1, 8, 9] as quais consistem em reduzir o número de iterações aumentando a velocidade de convergência da restauração da imagem em cada iteração. É importante salientar que o RLA não possui critério estabelecido para limitar o número de iterações antes do início da divergência da imagem restaurada [1]. No processo de restauração da imagem o ruído aditivo nas imagens pode ser reduzido com filtros Gaussianos na imagem inicial ou antes da imagem convergir para um valor mínimo, previamente estabelecido [6]. No caso do RLA (no quesito de desempenho), o mesmo usa duas convoluções em cada iteração; portanto, apresentando um alto custo computacional para um sistema embarcado [6].

Na implementação de um algoritmo de visão computacional em um processador convencional devem ser escolhidos alguns aspectos técnicos pelo programador para alcançar alguma vantagem em desempenho (velocidade). Entre eles pode-se mencionar a técnica de deconvolução escolhida, a linguagem de programação, o compilador, o sistema operacional, o tipo de processador, assim como o sistema/dispositivo de entrada/saída [10]. De fato, o tipo de algoritmo usado em uma plataforma computacional determina o número de operações de entrada/saída executadas [10].

Os algoritmos de restauração de imagens podem ser implementados em três tipos de arquiteturas: (*a*) programáveis, (*b*) reconfiguráveis e (*c*) de aplicações específicas [11].

As arquiteturas programáveis incluem Processadores de Propósito Geral (*General Purpose Processors-GPPs*) e os Processadores de Aplicações Específicas (*Application-Specific Processors-ASPs*) [11]. As arquiteturas reconfiguráveis permitem programar *hardware* (*hardware reconfigurável*), tal como no caso dos FPGAs (*Field Programmable Gate Arrays-FPGAs*) [11]. Uma abordagem possível para o projeto de uma simulação para um problema é mapear diretamente o algoritmo em *hardware*. Para tanto é possível usar FPGAs para esta tarefa, assim como circuitos dedicados do tipo ASICs (*Application-Specific Integrated Circuits-ASICs*) [11]. A escolha da plataforma computacional vai depender das vantagens e as desvantagens que possam ter para uma aplicação em particular, além dos seguintes critérios: precisão, espaço ocupado, consumo de energia, confiabilidade, complexidade de desenvolvimento e custo de fabricação (por exemplo NREs-Non Recurring Costs) [11, 12].

1.2 DEFINIÇÃO DO PROBLEMA E MOTIVAÇÕES

A degradação de imagens tem provocado uma grande motivação para desenvolver técnicas de restauração usando algoritmos que sejam eficientes do ponto de vista de desempenho, facilmente paralelizáveis e implementáveis em plataformas de sistemas embarcados.

Os algoritmos de restauração são muito usados por exemplo para a restauração de imagens que tem sofrido degradação ocasionada por defeitos ópticos, movimentos da câmera (*blur motion*) ou pelas condições do ambiente [4]. A restauração de imagens é muito usada nas áreas da astronomia e microscopia, mas sua aplicação pode ser estendida para outras áreas como a da indústria do petróleo. Exploradores subaquáticos equipados com câmeras são frequentemente usados para inspeção das condições das plataformas marinhas ou para operações que precisem de sistemas de visão.

Dentro das arquiteturas usadas para o processamento de imagens, destaca-se as arquiteturas reconfiguráveis por causa de seu desempenho e eficiência. A força das arquiteturas reconfiguráveis encontra-se no seu alto nível de paralelismo e no desempenho que pode ser conseguido para uma aplicação específica. Neste contexto, o algoritmo de Richardson-Lucy é muito usado por ser simples e de fácil implementação, além de possuir duas convoluções que são facilmente paralelizáveis. Neste sentido, é importante o desenvolvimento de plataformas para recuperar imagens que foram borradas por movimento, para se obter assim um sistema embarcado de visão computacional, acrescentado-se uma metodologia de verificação para validar os resultados obtidos.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo principal do presente trabalho é desenvolver uma arquitetura de *hardware* para a restauração de imagens em tempo real, utilizando o algoritmo Richardson-Lucy.

1.3.2 Objetivos específicos

(a) Desenvolver uma arquitetura de filtragem de imagens por convolução.

(b) Desenvolver uma metodologia para a simulação e verificação das arquiteturas propostas.

(c) Implementar um sistema de captura e visualização de imagens processadas.

(d) Implementar o RLA em um sistema embarcado junto com o sistema de captura e visualização para validar os resultados.

1.4 ASPECTOS METODOLÓGICOS DO TRABALHO

A metodologia utilizada no desenvolvimento do presente trabalho é composta de seis etapas, a saber:

(a) Na primeira etapa foi realizada uma revisão do estado da arte das principais técnicas e algoritmos de restauração de imagens, levando em conta aspectos de implementação em *hardware* e *software* e sua aplicação em sistemas embarcados.

(b) Na segunda etapa o algoritmo escolhido teve sua estrutura de funcionamento básico estudada em detalhes e foi analisada a arquitetura para a aceleração da restauração, com base, principalmente, nas técnicas de paralelismo e *pipelining*.

(c) Na terceira etapa, foi projetada a arquitetura de *hardware* em VHDL da filtragem por convolução para vários tamanhos de máscaras e saída de igual tamanho.

(d) Na quarta etapa, identificou-se as redundâncias de cálculos, de modo a economizarem-se recursos de *hardware*.

(e) Na quinta etapa, foi implementada uma metodologia de simulação de imagens degradadas por movimento e restauradas em *Matlab*. Para obter os resultados da arquitetura, usou-se as imagens degradadas geradas em *Matlab* e sua verificação em *ModelSim*.

(f) Na sexta etapa, foi realizada uma avaliação entre as imagens restauradas pela arquitetura e as imagens restauradas em *Matlab* (imagens de referêrencia), usando-se a métrica SR-SIM (*Spectral Residual Based Similarity*).

(g) Na sétima etapa e última etapa, a arquitetura foi implementado em um sistema de visão em tempo real usando uma placa DE2-115 da Terasic que utiliza um FPGA (*Cyclone VI-E*), uma câmera digital CMOS (*complementary metal-oxide semiconductor*) da Terasic, um *display* LCD e uma interface VGA (*Video Graphics Array*).

1.5 CONTRIBUIÇÕES DO TRABALHO

As principais contribuições são as seguintes:

Implementação em hardware do algoritmo Richardson-Lucy: foi implementada em FPGA uma abordagem paralela do algoritmo Richardson-Lucy. Considerando o estado de arte de processamento de imagens em sistemas embarcados, a implementação do algoritmo em sua forma básica (sem aceleração de sua convergência) em um *pipeline* (usando um *loop unrolling*), além da metodologia de verificação usada neste trabalho, é um bom ponto de partida para implementações que permitam a aceleração do algoritmo.

1.6 ESTRUTURA DO TEXTO

Esta dissertação está organizada da seguinte forma: no Capítulo 2 é feita uma revisão das plataformas utilizadas para processamento de imagens; o Capítulo 3 apresenta-se uma revisão do processamento de imagens, a abordagem matemática do RLA e uma revisão bibliográfica dos trabalhos correlatos; no Capítulo 4 são apresentadas as arquiteturas da convolução e do RLA propostas junto com a metodologia para a simulação e verificação das arquiteturas; no Capítulo 5 são apresentados os resultados dos testes e a análise; finalmente, o Capítulo 6 apresenta as conclusões e propostas de trabalhos futuros.

2 PLATAFORMAS PARA O PROCESSAMENTO DE IMAGENS E VÍDEO

Este capítulo apresenta os tópicos referentes a plataformas que podem ser usadas para o projeto de sistemas embarcados, os quais podem ser direcionados para diferentes tarefas, tais como o processamento de imagens.

Para o caso da restauração de imagens várias plataformas computacionais podem ser usadas. A proliferação de plataformas para este tipo de tarefa é devido à grande quantidade de informação e cálculos matemáticos que devem ser executados. Para este caso, as principais operações matemáticas são: (a) transformada rápida de Fourier (*Fast Fourier Transform*), (b) filtragem em tempo-real e (c) operações matriciais (por exemplo a inversão de matrizes usando métodos simples como redução de Gauss-Jordan) [13]. No entanto, a maioria das plataformas não estão otimizadas para fazer este tipo de operações. Em aplicações em tempo-real a informação deve ser salva, ordenada, comparada, deslocada, etc [13]. Isto faz com que o tempo de execução das instruções seja um fator determinante na aplicação. Entretanto, as plataformas devem oferecer um desempenho adequado levando em conta os aspectos de *software* e *hardware* embarcados nas mesmas [10].

No intuito de cumprir esta exigência, têm sido desenvolvidas e propostas diferentes plataformas, com a intenção de aproveitar o paralelismo dos algoritmos a serem implementados [13]. Embora cada uma dessas arquiteturas tenha suas vantagens e desvantagens inerentes todas elas dependem fortemente da capacidade de possuir uma grande quantidade de recursos que garantam o paralelismo em nível de execução. Exemplos destas plataformas são: (a) arquiteturas programáveis, (b) arquiteturas reconfiguráveis e (c) arquiteturas de aplicações específicas [11]. A escolha da plataforma vai depender dos seguintes critérios: (a) espaço ocupado, (b) consumo de energia, (c) confiabilidade, (d) complexidade de desenvolvimento e (e) custo de fabricação [12]. A seguir serão descritas os tipos de plataformas mais utilizados na atualidade para a tarefa de processamento de imagens.

2.1 ARQUITETURAS PROGRAMÁVEIS VIA *SOFTWARE*

Os processadores programáveis via *software* oferecem flexibilidade e versatilidade, sendo fáceis de programar, testar e verificar pois o algoritmo é implementado em *software* [14]. No entanto, a flexibilidade faz com que o processador sacrifique o desempenho em aplicações de processamento intensivo ou operações com restrições em tempo real, devido principalmente ao gargalo de von Neumann [13], o qual envolve limitações ligadas ao processamento sequencial de instruções e a limitações físicas do barramento de dados [14]. Neste caso, pode-se obter muita flexibilidade com este tipo de sistema, sobretudo porque o único que deve ser feito é trocar o programa implementado no processador para rodar uma nova solução.

Neste contexto, o GPP é a arquitetura programável que faz uso do modelo de Von Neumann [13], sendo projetado para executar várias tarefas perdendo no quesito de desempenho para o caso de aplicações específicas [10]. Isto tem impulsionado a criação de processadores dedicados para uma aplicação em particular ou processadores específicos de aplicativos (*Application Specific Processors-ASPs*) [10]. Dentro dos ASPs encontram-se os DSPs, processadores de sinais digitais (*Digital Signal Processors-DSPs*) e as unidades de processamento gráfico (*Graphics Processing Units-GPUs* [14].

2.1.1 GPPs-*General Purpose Processors*

Plataformas baseadas em GPPs, são capazes de suportar qualquer algoritmo que possa ser convertido em um programa de computador [11]. O desempenho de um programa depende da eficácia dos algoritmos usados no programa, do compilador usado para criar e traduzir o programa ao nível de instrução de máquina, e da eficiência do processador na execução dessas instruções [11]. Como desvantagem pode-se citar a execução das instruções é sequencial quando usa-se um único processador [11] . Uma forma de melhorar o desempenho é aumentar o número de instruções executadas por segundo. Ao dividir a execução da instrução em um conjunto de etapas, várias instruções podem executar concorrentemente um *pipeline* de instruções. Esta técnica é a mais usada para melhorar o desempenho de um processador.

Dentre dos GPPs mais populares encontram-se, por exemplo, os microprocessadores usados em PCs (*Personal Computers*) ou *workstations*, processadores embarcados do tipo *soft-processor* (*NIOS* e *Microblaze*) e processadores embarcados do tipo *hard-processors* (*ARM*, *PowerPC* e, em geral, os encontrados em *microcontroladores*) [11, 15].

2.1.1.1 Arquiteturas do tipo *Von Neumann*

A maioria da arquitetura de um PC moderno está baseada na arquitetura *Von Neumann.* As cinco unidades clássicas de um PC são: as entradas (*Input*), saídas (*Output*), armazenamento (*Memory*), via de dados (*Datapath*), assim como a via de controle, ou controlador (*Control*), sendo que a via de dados e o controlador formam a CPU (*Central Processing Unit*) [10]. A CPU é o responsável por executar as instruções transferidas a partir da memória, executar as instruções e armazenar os resultados de volta na memória. O grande problema desse modelo de arquitetura é que para aplicações de tempo-real o item de desempenho fica comprometido, sobretudo pelo processo de escrita/leitura de/para a memória [12].

Recentemente, foram introduzidos vários núcleos (*multicore*) para resolver o gargalo do processamento sequencial com o intuito de melhorar o desempenho, tendo em conta algumas limitações, sobre tudo no quesito de consumo de potência (vide Figura 2.1). Aplicações independentes, ou partes do mesmo aplicativo, são paralelizadas sobre várias unidades de processamento. Isto significa que a velocidade de processamento poderia potencialmente aumentar com um fator igual com o número de núcleos de processamento, limitadas deste caso à Lei de Amdahl [11]. Este tipo de arquitetura consegue aliviar os três principais problemas de um processador: (*a*) barreira de memoria (*memory wall*) , (*b*) barreira de potencia (*power wall*) e (*c*) barreira de nível de instruções em paralelo (*ILP wall*) [10], temas que serão discutidos a seguir.

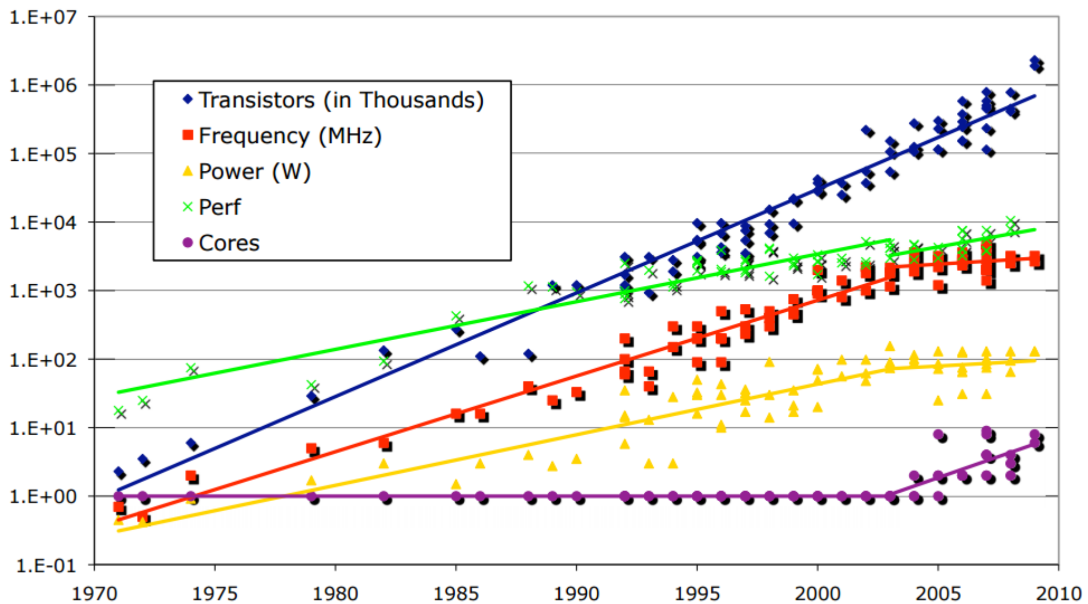


Figura 2.1. As Tendências do processador de Kathy Yelick [16]

2.1.1.2 Barreira de memória (*The memory wall*)

Ao contrário dos processadores, a velocidade das memórias não acompanha o desenvolvimento das arquiteturas de computadores, especificamente o incremento da vazão (*througput*) da execução de instruções (vide Figura 2.2). Neste caso, o desempenho dos programas está limitado pela velocidade com que as operações de escrita/leitura de/para a memória são executadas, seguindo os passos de decodificação/execução relacionados com as instruções [10]. Em consequência, gera-se um tempo de latência de acesso á memoria, trazendo assim uma perda de desempenho devido ao fato de que isto limita o beneficio de se ter um processador com alto desempenho (*througput*).

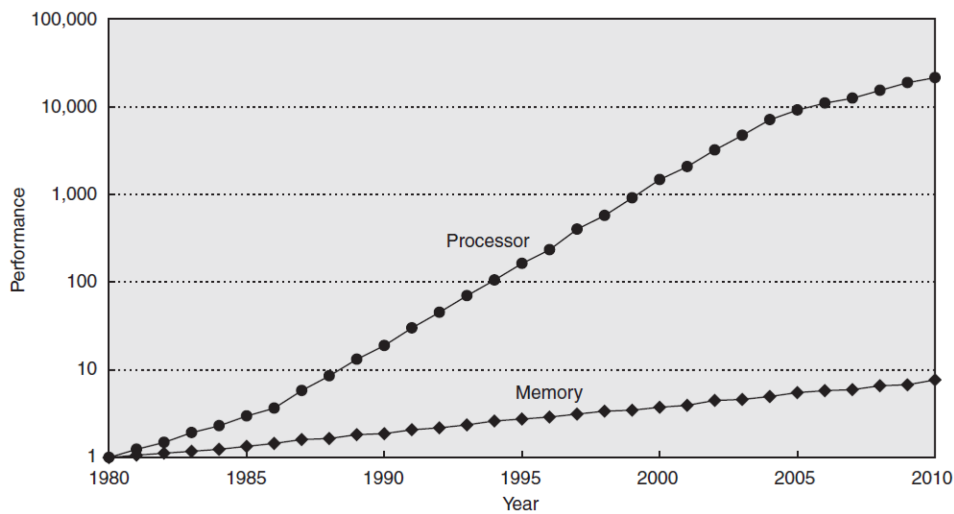


Figura 2.2. Diferencia de desempenho entre os processadores e a memoria ao longo do tempo [17]

A Figura 2.2 apresenta a razão de melhoria da memória (1,07 por ano desde 1980) no quesito de latência. A linha de processadores assume uma melhoria de 1,25 até o ano 1986, 1,52 até 2004, e uma melhoria de 1,20 após essa data.

2.1.1.3 Barreira de potência (*The power wall*)

Devido ao grande incremento na densidade de transistores por chip, conforme à Lei de Moore [10], o consumo de energia em um chip poderia-se incrementar de maneira substancial (vide Figura 2.1). Além dos componentes eletrônicos, os incrementos e/ou decrementos da frequência de operação e o número de transistores conectados a uma saída (*fanout*) traz consigo uma maior e/ou menor produção de calor (vide Figura 2.1). A Figura 2.1 apresenta um limite da frequência conforme são desenvolvidas novas arquiteturas de processadores no intuito de reduzir o consumo de energia.

A carga capacitiva relacionada com os transistores, a frequência de operação assim como a tensão elétrica de operação estão relacionados com a equação a seguir:

$$Potência \propto \frac{1}{2} \times Capacitância \times Tensão^2 \times Freq_operação, \tag{2.1}$$

no qual, consegue-se limitar a frequência de operação, o número de transistores e a tensão por *chip*. Uma forma de controlar o consumo de potência é pelo uso de sistemas de arrefecimento, os quais têm sido usados durante décadas com este intuito. Entretanto, são ineficientes quando os mesmos têm que ser adaptados a plataformas com restrições de custo, área do sistema e volume.

2.1.1.4 Barreira do paralelismo no nível de instrução (*Instruction-level parallelism wall*)

As dificuldades associadas ao paralelismo no nível de instrução tem os seguintes gargalos: (*a*) instruções com diferentes tempos de execução, (*b*) instruções que dependem da instrução anterior (dependência de dados) e (*c*) os desvios condicionais (dependência de controle). Na procura de soluções para este problema, tem-se chegado a um limite denominado de barreira de nível de instruções em paralelo (*Instruction-Level Parallelism wall- ILP wall*) [10]. Além disso, o aumento da complexidade do *hardware* traz consigo um aumento de consumo de potência, principalmente se a arquitetura possui múltiplos estágios no *pipeline*. Para garantir o equilíbrio entre o consumo de potência e o desempenho, o *pipeline* não deve ser profundo, ou seja, o *pipeline* deve ter poucos estágios (*non-deep pipeline*) [18, 11].

2.1.2 Processadores Digitais de Sinais (DSPs)

Os DSPs fazem parte das arquiteturas programáveis otimizados em termos de velocidade para processar dados em tempo real, executando assim algoritmos matemáticos complexos para diversas aplicações, tais como filtragem, correlações, transformadas rápidas de Fourier (FFT) e outras operações matemáticas. Os mesmos também podem ser usados para aplicações de telecomunicações, geração e reconhecimento de voz, processamento de imagens e vídeo [15]. Devido a sua flexibilidade, é fácil encontrar recursos para DSPs (bibliotecas em C++) que executam tarefas específicas. Os DSPs possuem unidades dedicadas MAC (*multiply-accumulate*), registradores de deslocamento otimizados, além de instruções específicas de processamento de sinais em seu conjunto de instruções, permitindo assim uma compilação mais eficiente dos programas [19].

Estas operações possuem um grande nível de paralelismo. No caso de processamento de imagens os mesmos incluem as operações pixel a pixel ou em uma região grande das imagens [20].

No entanto, já existem DSPs avançados que paralelizam as instruções sequenciais [20]. As plataformas DSPs têm grande demanda em processamento em aplicações de uso intensivo de dados, tais como vídeo e internet em celulares ou aparelhos portáteis, mantendo baixo custo e baixo consumo de potência.

Neste sentido, na referência [21] é proposto um sistema de visão em um chip para uma imagem com resolução de 64 × 64 pixels. O sistema de detecção de imagem consiste basicamente de um foto-sensor e um DSP que integra um conversor analógico-para-digital (ADC) (como visto na Figura 2.3). O foto-sensor fundamentalmente contém um fotodetector e um pré-amplificador [21].

No quesito de desvantagens dos DSPs pode ser observado que os mesmos representam arquiteturas de von Neumann com *data-paths* sofisticados, incluindo várias Unidades Lógicas Aritméticas (ALUs) e barramentos específicos. Neste sentido, os mesmos representam arquiteturas de fluxo de instruções, tal como os GPPs.

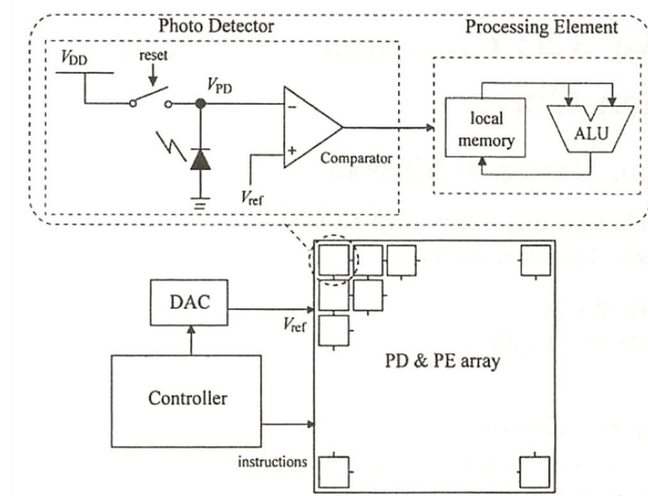


Figura 2.3. Estrutura do Sensor [21]

2.1.3 Unidades de Processamento Gráfico (GPUs)

GPUs são arquiteturas dedicadas inicialmente para tarefas de renderização de imagens, as quais foram adotadas pela industria principalmente para a área de vídeo-jogos. Este tipo de processador consiste de um *pipeline* (programável), com estágios especializados onde cada tipo de operação está definida em *hardware*. Este *pipeline* possui tipos de unidades programáveis de processamento ou *shaders*, por exemplo: *shader vertex*, encarregado de transformar as coordenadas dos vértices na cena para um sistema comum; *shader pixel*, encarregado da texturização das

cenas; e o renderização/rasterização, encarregado de transformar informações vetoriais em uma imagem de pixels. Os GPUs permitem a escabilidade das soluções na área de processamento de imagens, possuindo adicionalmente altos recursos de paralelismo e desempenho no quesito de aritmética em ponto flutuante [14].

Não obstante, nos últimos anos tem surgido a aplicação de GPUs para a execução de tarefas diferentes ao processamento de imagens e de vídeo, devido ao *pipeline* com grande capacidade de programabilidade. Esta técnica de programação ficou conhecida como GPGPU (*General Purpose Processing on a Graphics Processing Unit*). Algoritmos mapeados para GPUs têm bom desempenho se comparado com sua execução em GPPs, específicamente quando o algoritmo depende principalmente do alto volume de dados e não da complexidade das operações envolvidas. Os GPUs são utilizados em diversas aplicações tais como: supercomputadores, computadores híbridos de CPUs e GPUs, bioinformática, simulação de dinâmica molecular, criptografia e segurança de sistemas. A tendência é que as GPUs se desenvolvam cada vez mais, melhorando o desempenho e alcançando um menor consumo de energia [14].

Com respeito às deficiências dos GPUs temos que as mesmas representam processadores de Von Neumann com melhoras no quesito de paralelismo, onde as instruções devem ser executadas em cada elemento de processamento na forma sequencial. Neste caso, este tipo de dispositivos podem ser vistos como arquiteturas de von Neumann com vias de dados (*datapaths*) sofisticados, com alta capacidade de comunicação entre seus elementos; representando arquiteturas mistas no quesito de fluxo de dados/ instruções.

2.2 CIRCUITOS INTEGRADOS DE APLICAÇÃO ESPECÍFICA (ASICs)

Um ASIC é um circuito customizado para uma tarefa dedicada [12]. Este tipo de abordagem, para o projeto de sistemas em chip, permite reduzir a quantidade de lógica digital padrão, melhorando tanto o desempenho como o consumo de potência. Os ASICs melhoram o desempenho em comparação com outro tipo de metodologias de projeto de sistemas, tais como os dispositivos lógicos programáveis (PLDs), vide seção 2.3. Além do anterior, os ASICs melhoram quesitos de segurança na área de propriedade intelectual de módulos do projeto, tendo em conta a dificuldade de se aplicar uma engenharia reversa [22]. Os ASICs, uma vez fabricados, não possuem flexibilidade e a validação de sua solução pode levar de meses a anos. Por outro lado, ASICs têm a desvantagem dos altos custos de desenvolvimento no NRE (*Non-Recurring Engineering-NRE*) [12, 22].

2.3 ARQUITETURAS RECONFIGURÁVEIS

Arquiteturas reconfiguráveis possuem *hardware* programável, ou seja, qualquer tipo de sistema integrado que pode ser configurado pelo usuário final via *software* para uma aplicação particular [23]. Este tipo de arquiteturas é chamado, de maneira genérica, de dispositivo lógico programável (*Programmable Logic Devices-PLDs*) [12]. Os mesmos possuem um arranjo de elementos lógicos e uma estrutura de interconexões e/ou estrutura lógica que pode ser programada com base nas especificações do usuário [12]. Por exemplo, qualquer algoritmo pode ser implementado em este tipo de circuito integrado, mediante técnicas de mapeamento (*mapping*) do algoritmo para estruturas de *hardware*. As duas principais classes de dispositivos do tipo PLDs são os Dispositivos Lógicos Programáveis Complexos (*Complex Programmable Logic Devices-CPLDs*) e as matrizes de portas programáveis em campo (*Field Programmable Gate Arrays-FPGAs*) [12, 24]. Comparado com os ASICs, os PLDs oferecem uma solução de baixo custo (pelo menos no item de prototipagem de sistemas), com baixa complexidade de desenvolvimento [23]. A seguir será discutida a plataforma usada neste trabalho.

2.3.1 FPGAs-*Field Programmable Gate Arrays*

Os FPGAs consistem em três recursos básicos (vide Figura 2.4): (a) blocos lógicos configuráveis (*Configurable Logic Block-CLB*) relativamente pequenos e independentes que podem ser interconectados para criar funções maiores, (b) interconexões programáveis e as (c) entradas/saídas que são configuráveis [23, 24]. Adicionalmente, os blocos lógicos são capazes de implementar funções combinacionais ou sequenciais. Os blocos lógicos contêm basicamente: (a) tabelas de consulta (*Look-Up Tables-LUT*), (b) multiplexadores e/ou portas lógicas básicas com duas ou mas entradas e (c) flip-flops. No entanto, FPGAs modernos podem ter recursos adicionais como: (a) circuitos de gerenciamento de relógio, (b) unidades de memória, (c) blocos de memoria RAM, (d) microprocessadores embarcados e (e) recursos DSP dedicados [15]. A maioria dos blocos lógicos possuem *flips-flops* e *latches* para a implementação de lógica sequencial [23, 24].

Atualmente, os FPGAs têm por base a tecnologia de programação (configuração) das memórias estáticas (*Static Random Access Memory-SRAM*), a tecnologia anti-fusível, EEPROMs (*Electrically Erasable Programmable Read-Only Memory*) e EPROMs(*Electrically Programmable Read-Only Memory*) [23].

A arquitetura de roteamento de um FPGA corresponde à maneira com que as chaves ou comutadores programáveis (*switch block*) e segmentos de trilha são posicionados para permitir a interconexão das células lógicas (vide Figura 2.5) [23, 24]. A Figura 2.5 ilustra, um exemplo de como de poderia conectar os blocos lógicos $BL0$ e $BL4$ utilizando as estruturas dessa arquitetura

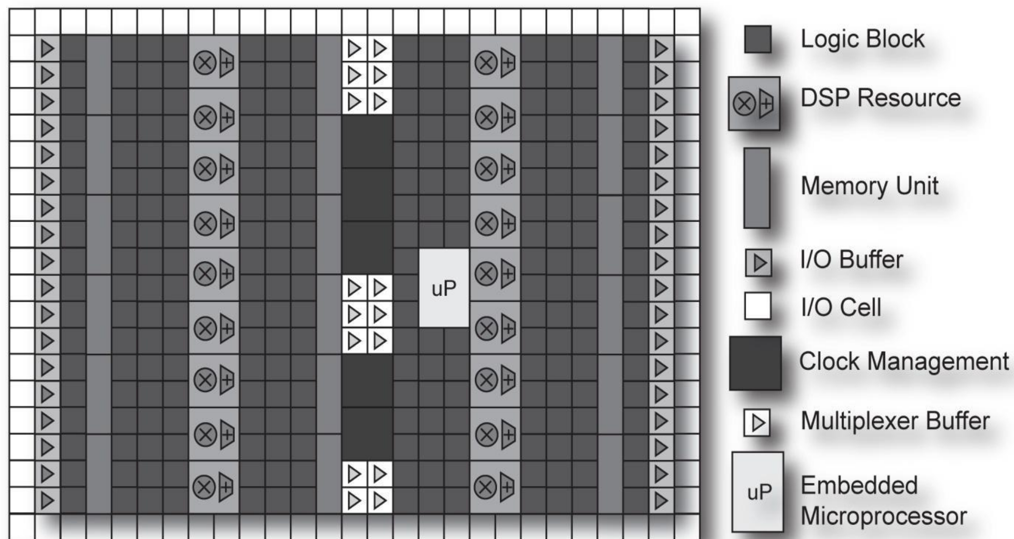


Figura 2.4. Estrutura geral de um FPGA [23]

de roteamento.

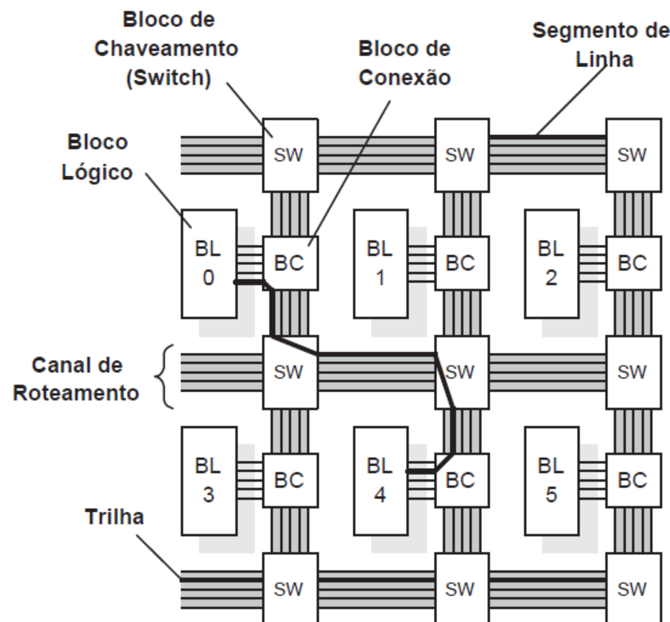


Figura 2.5. Arquitetura de interconexão de um FPGA [23]

O processo de projeto com FPGAs (*FPGA Design Flow*) está segmentado em níveis hierárquicos envolvendo as seguintes etapas (vide Figura 2.6): (*a*) desenho de esquemático, (*b*) síntese lógica, (*c*) posicionamento e roteamento, (*d*) verificação e testes e (*e*) configuração/programação do FPGA [25, 26]. A síntese lógica identifica estruturas que foram descritas no código fonte (HDL/Verilog) fornecendo o esquemático RTL e gerando o seu equivalente em componentes lógi-

cos (na forma de uma rede Booleana) [19, 25]. Neste processo é gerado um *netlist* completo para o projeto que é utilizado como entrada para o próximo passo no fluxo de desenvolvimento [25, 26]. Na etapa de mapeamento, o *netlist* é transcrito para os blocos lógicos disponíveis no dispositivo utilizado, otimizando ou minimizando o número de blocos lógicos requeridos [25]. Na etapa de roteamento é feita a conexão entre os blocos lógicos que foram gerados na etapa anterior. A lógica a ser configurada na FPGA é escrita em um arquivo chamado *bitstream*, que contêm informações eléctricas do dispositivo, tais como padrões de entrada/saída [25].

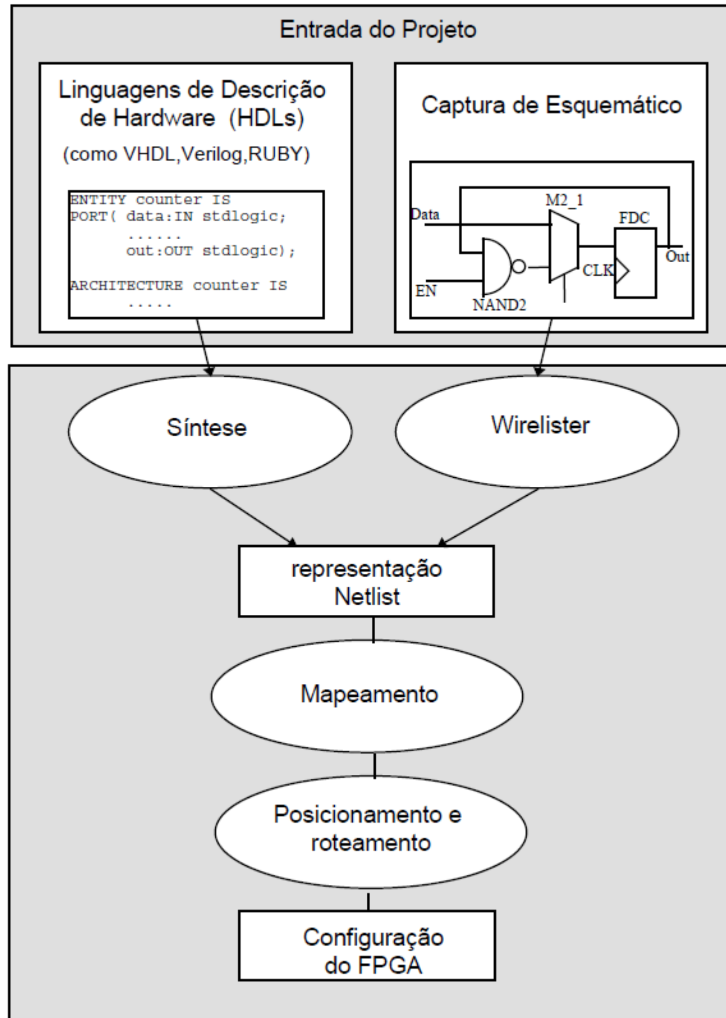


Figura 2.6. Etapas de um projeto com FPGAs [25]

No fluxo de projeto a tarefa de simulação é o tipo mais comum de verificação utilizado. Este processo imita a funcionalidade ou o comportamento de um projeto digital em um computador. Nos FPGAs, a verificação pode ser realizada em nível comportamental ou em nível de portas lógicas, ou no nível temporal (*timing*). No caso de simulação temporal, várias abordagens podem ser tomadas. Neste caso a temporização pode ser verificada com os atrasos nominais da tecnologia empregada [25].

Atualmente, os FPGAs possuem dois tipos de granularidade: fina e grossa. A granularidade fina apresenta menor quantidade de lógica nas unidades de reconfiguração como LUTs, flip-flops e multiplexadores. A vantagem desses tipos de blocos é que podem ser utilizados mas eficientemente e são mas eficientes para o projeto de circuitos de baixa complexidade, sendo que sua programação é feito no nível de bits. Em contrapartida, é necessário um grande número de segmentos de linha e chaves programáveis para o roteamento. No entanto, as FPGAs com granularidade grossa possuem maior quantidade de lógica, na forma de ULAs, multiplicadores, deslocadores, entre outros. Este tipo de arquitetura é usada quando a aplicação requer grande nível de paralelismo, como é em geral o caso em aplicações orientadas para o fluxo de dados [25, 27].

2.3.1.1 Descrição geral da família Cyclone IV-E

Neste trabalho foi usada a família Cyclone da Altera. Os dispositivos Cyclone IV-E são arranjos de portas programáveis pelo usuário com vários elementos configuráveis para projetos de sistemas de alto desempenho e alta densidade. Os dispositivos Cyclone IV implementam as seguintes funcionalidades:

(a) **Blocos I/O:** proveem a interface entre os pinos do circuito e a lógica de configuração interna. Os padrões I/O mais populares e de vanguarda são suportados por blocos I/O programáveis (IOBs).

(b) **Blocos de lógica configurável (*CLBs*):** os elementos lógicos básicos para os FPGAs da Altera proveem lógica síncrona e combinacional. Os CLBs das FPGAs Cyclone IV E são baseados em LUTs de 4 entradas e fornecem recursos e desempenhos superiores em comparação com gerações anteriores de lógica programável.

(c) **Blocos RAM:** proveem uma RAM flexível de 9Kbit tipo $truedual - port$ que são organizadas em cascata para formar blocos de memória maiores. Adicionalmente, os blocos de memória RAM da FPGA Cyclone IV-E podem ser configurados como blocos independentes de RAM de 8192×1, 4096×2, 2048×4, 1024×9, 1024×8, 1024×9, 512×16, 256×32 e 256×36.

(d) **Multiplicadores embarcados:** podem ser configurados como um multiplicador de $18 \times 18 - bits$ ou dois de $9 \times 9 - bits$.

2.4 CONCLUSÃO DO CAPÍTULO

Neste capítulo foram apresentados os conceitos básicos sobre plataformas de processamento de imagens e vídeo. Foi identificado que as características dos algoritmos limitam o desempenho dependendo da plataforma usada, fatores tais como: a potencia consumida, o tamanho das palavras e a velocidade de memória. O *tradeoff* entre as arquiteturas programáveis, reconfiguráveis e de aplicações específicas é de difícil analise, envolvendo muitos fatores, tais como: (a) tempo requerido para completar uma tarefa, (b) uso de memória e (c) potência usada associada com essa tarefa.

As arquiteturas programáveis possuem recursos de *hardware* dedicados para aplicações particulares que são fáceis de programar, testar e verificar. Isto oferece uma flexibilidade que permite ter uma fácil portabilidade de aplicações e algoritmos. Este tipo de arquiteturas podem ser usadas para executar operações específicas sendo usadas como co-processadores para acelerar cálculos que exigem uma grande quantidade de operações.

As arquiteturas de *hardware* reconfigurável junto com os recursos de *hardware* dedicado possuem um balance entre eficiência e desempenho. A maioria dos algoritmos podem ser implementados em FPGAs mediante a aritmética distribuída. As FPGAs possuem uma vantagem sobre as plataformas baseadas em instruções no quesito de aceso a memoria, consumo de potência e o nível de paralelismo. Neste contexto, o FPGA é a melhor plataforma para implementar novas arquiteturas.

3 ALGORITMOS DE PROCESSAMENTO E RESTAURAÇÃO DE IMAGENS

A restauração de imagens é parte fundamental nos sistemas de visão computacional para resolver o problema de degradação de imagens. A solução para este problema é o uso de algoritmos de restauração que permitam obter a imagem original, ou seja, uma imagem que representa a verdadeira cena. Com o propósito de mostrar os diferentes métodos de restauração de imagens e ter uma clareza sobre os mesmos, assim como entender a nomenclatura usada, este capítulo inicia com a apresentação de algumas definições conceituais sobre o processamento de imagens, incluindo suas principais características, assim como alguns exemplos. Finalmente, são apresentados alguns trabalhos que implementam restauração de imagens em plataformas computacionais, principalmente em FPGA. É muito importante observar que existem muitos caminhos para se implementar este tipo de algoritmos, o que resultam em diferentes arquiteturas, em função dos vários tipos de acessos à memória e níveis de paralelismo, entre outros.

3.1 IMAGENS DIGITAIS

Uma imagem pode ser definida como uma manifestação de um fenômeno que pode ser expresso de forma quantitativa [28]. O termo "imagem" refere-se a uma função bidimensional de intensidade de luz $f(x, y)$, onde x e y representam as coordenadas espaciais e o valor de f (em ponto qualquer) é proporcional ao brilho (ou nível de cinza) da imagem neste ponto [19]. O nível de cinza em um ponto da imagem vai depender da quantidade de luz que incide sobre a cena (vide Equação 3.1) [19]. Os componentes $i(x, y)$ e $r(x, y)$ são chamados de *iluminação* e *reflexão*. O produto das duas funções vai produzir a função $f(x, y)$ [19].

$$f(x, y) = i(x, y) \times r(x, y) \tag{3.1}$$

Os sistemas de visão não têm a capacidade de processar imagens contínuas, porém, as imagens contínuas são transformadas em imagens discretas a partir da medição de um processo físico da radiação eletromagnética [28]. A energia da radiação é digitalizada usando dois processos, a

amostragem e a quantização [28]. A amostragem discretiza a imagem nas coordenadas espaciais e a quantização discretiza a imagem no nível de cinza [19]. Cada valor digitalizado é codificado para uma unidade denominada pixel (contração de *picture element*) [28]. Assim, a forma digital do dado é que possibilita aos sistemas de visão a capacidade de processar imagens. Os pixels são organizados em uma disposição de linhas e colunas (arranjo bidimensional ou matriz), representados por um sistema de coordenadas "M" e "N" [19] (vide Figura 3.1).

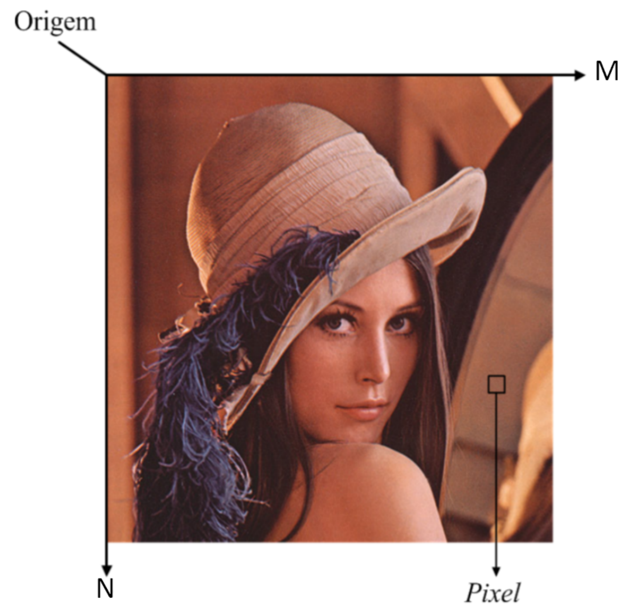


Figura 3.1. Representação de imagens digitais, adaptado de [19]

3.2 TIPOS DE OPERAÇÕES EM UMA IMAGEM

Um sistema de processamento de imagens é constituído de diversas operações em três níveis: baixo, médio e alto [19]. A seguir serão apresentadas descrições de cada nível.

As operações de baixo nível recebem uma imagem em sua entrada e produzem uma imagem em sua saída, este passo é comumente conhecido como pré-processamento, o qual envolve duas categorias principais: (a) métodos que operam no domínio espacial e (b) métodos que operam no domínio da frequência [29]. Uma técnica de pré-processamento simples é a filtragem de imagens, usada para funções tais como: (a) redução de ruídos introduzidos pelos sensores e a correção de distorções geométricas causadas pelo sensor, (b) filtragem para realce das características e atributos das imagens tais como bordas ou texturas e vizinhanças e (c) filtragem para algumas transformações no domínio da frequência [19, 29].

As operações de nível intermediário retratam certos atributos ou características de interesse da imagem [28]. Os objetos precisam ser identificados através de um processo de segmentação que identifique a localização, a topologia e a forma dos objetos [28]. 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 [19].

A função das operações de alto nível têm como objetivo reconhecer, verificar ou inferir a identidade dos objetos a partir das características e representação obtidas pelas etapas anteriores de processamento [19]. Esta interpretação é feita a partir da forma geométrica dos objetos resultante da segmentação [28].

3.2.1 Cadeia geral de um sistema de visão computacional

Na maioria de sistemas de visão computacional busca-se reduzir a quantidade de dados a ser processados. Geralmente, em uma cadeia de processamento de imagens e vídeos, a grande variedade de operações vai das operações mais regulares e com grande fluxo de dados, até os irregulares e com poucos dados no fim da cadeia. Uma típica cadeia de processamento combina os três tipos de operações acima mencionadas (operações de nível baixo, intermediário e alto) como mostrado na Figura 3.2. Nesta figura está-se representando a redução na quantidade de dados em cada etapa para uma imagem de $N \times N$, em um típico exemplo de reconhecimento facial. Na etapa de pré-processamento há uma filtragem para eliminação de ruídos. Na etapa de *segmentação* há uma redução da quantidade de dados, pela transformação da imagem filtrada em uma imagem binária. 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. Por último, na etapa de *interpretação*, é gerada a informação sobre o reconhecimento ou não da face [19].

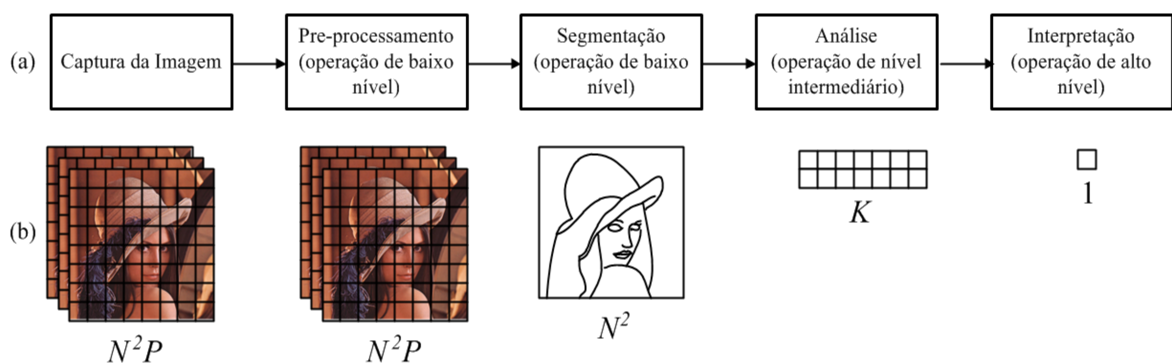


Figura 3.2. (a) Cadeia de processamento e (b) redução na quantidade de dados [19]

3.3 TRANSFORMAÇÕES ESPACIAIS

O processamento de imagens procura modificar e preparar os pixels de uma imagem digital para obter uma forma mais adequada, visando operações posteriores. Há dois grandes tipos de pré-processamento de imagens: (a) o melhoramento e (b) a restauração de imagens [19].

O melhoramento de imagens tenta aperfeiçoar a qualidade da imagem ou realçar aspectos particulares dentro da mesma[19]. As operações para melhorar as imagens podem ser considerados como filtros bidimensionais. Podem ser considerados dois tipos de filtros: lineares e não lineares. Os filtros lineares podem ser desenvolvidos ou implementados no domínio espacial (*spatial operations*) ou no domínio da frequência (*transforms operations*) [30]. A filtragem espacial apenas usa a informação local, ou seja, os filtros espaciais operam transformações pixel a pixel [31]. Filtragem na frequência é uma aplicação importante da transformada de Fourier (Fourier Transform), tomando os espectros de frequência das imagens de entrada e de saída [31]. Pode-se definir a filtragem espacial como a soma dos dados com diferentes pesos, ou seja, os valores dos pesos definem o tipo de filtragem a que a imagem será submetida [19]. Neste contexto, os filtros são classificados em três tipos de filtros: (a) passa-baixa, (b) passa-faixa e (c) passa-alta [28]. O filtro passa-baixa atenua as frequências altas da imagem eliminando as transições abruptas na intensidade. Um exemplo deste tipo de filtro ou do efeito de borramento (*blurring*); o filtro passa-alta recupera os detalhes estruturais da imagem (por exemplo as bordas) e o filtro passa-banda se usa para remover frequências baixas e altas [28]. A maioria de aplicações em processamento de imagens inclui filtragem espacial em 2-D baseado na operação de convolução entre o filtro bidimensional (máscara) e a imagem [1].

Por outro lado, os sistemas de aquisição de imagens tendem a degradar a qualidade das imagens digitais com a adição de ruído, deformação geométrica, movimentos da câmera e/ou fatores do ambiente (turbulência e o aerossol atmosférico). Uma das grandes preocupações do processamento de imagens é diminuir as degradações da qualidade das imagens geradas pelos sistemas de aquisição. As técnicas de restauração de imagens são desenvolvidas a partir do conhecimento *a priori* do fenômeno da degradação com o objetivo de reconstruir ou recuperar as imagens que foram degradadas, aumentando certas características da imagem como detecção de bordas, correlação, contraste, desborramento e diminuir as distorções [30].

3.3.1 Filtragem por convolução

A operação da convolução consiste em deslocar a máscara sobre a imagem (*operação de janelamento*) [19]. Uma operação é realizada formando uma vizinhança com os pixels da imagem de entrada, o valor do resultado é colocado na imagem de saída, geralmente na mesma localização

do centro da máscara da imagem de entrada, e a máscara é deslocado um pixel ao longo da mesma linha para processar a próxima vizinhança dos pixels da imagem de entrada [19]. Quando uma linha de pixels é concluída, a máscara é deslocado uma linha para baixo e o processo é repetido (vide Figura 3.3).

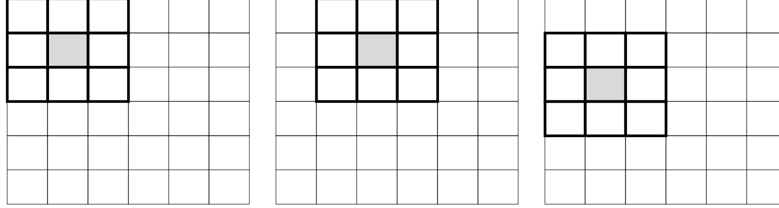


Figura 3.3. Deslocamento da máscara para o filtro espacial

A convolução discreta entre a imagem de entrada e a máscara é escrita matematicamente como:

$$g(x,y) = f(x,y) * h(x,y) = \sum_{s=\frac{-m}{2}}^{\frac{m}{2}} \sum_{t=\frac{-n}{2}}^{\frac{n}{2}} f(s,t) \cdot h(x-s, y-t) \tag{3.2}$$

no qual f é a imagem de entrada, h é a máscara, g é a saída do filtro, e $\otimes$ indica a operação de convolução. Tamanhos típicos para estas operações são de 3×3 ou 5×5, embora máscaras de maior tamanho possam ser utilizados [1]. Quase qualquer função pode ser programada dentro da máscara, por exemplo suavizado, nitidez, detecção de bordas, correlação (reconhecimento de objetos), etc. [31].

Na filtragem por convolução é desejável que o tamanho das imagens de saída e de entrada sejam iguais [31]. Por tanto, quando o centro da máscara situa-se na borda da imagem, alguns dos pesos do filtro ficam fora dos limites da imagem [31]. Isto provoca uma distorção nas bordas da imagem de saída. Uma técnica para manter o mesmo tamanho da imagem é preencher com zeros alguns pixels dentro da vizinhança (*padding*) como é apresentado na Figura 3.4.

3.3.1.1 Complexidade computacional

Desenvolvimento de algoritmos de processamento de imagens são de alto custo computacional para a maioria de processadores sequenciais. Esses algoritmos, muitas vezes, usam operações aritméticas que incluem uma longa sequência de instruções constituindo uma grande carga computacional para o processador [1]. A operação de convolução em 2-D pode ser expressa em termos das operações matemáticas usadas para processar uma imagem, por exemplo multiplicações e adições. Na Tabela 3.1 é mostrado o número total de execuções para uma operação de

Figura 3.4. Operação de janelamento na borda da imagem de entrada

convolução para uma imagem de tamanho $M \times N$ com uma máscara de tamanho $w \times w$. Com o fim de alcançar um desempenho de tempo real (pelo menos 30 fps, *frames per second*- quadros por segundo) é necessária uma capacidade computacional de muitas giga-operações por segundo (*GOPs*) [19, 32]. Por exemplo, uma imagem de 800×525 com uma máscara de 9×9 possui uma capacidade de 135 GOPs aproximadamente.

Tabela 3.1. Carga computacional para a convolução de uma imagem em termos das operações aritméticas [32]

Operação	**Número de execuções**
Multiplicação	$w^2 \times M \times N$
Adição	$[w^2 - 1] \times M \times N$
Escrita/leitura	$[2 \times w^2 + 1] \times M \times N$

3.3.1.2 Exemplos de aplicação

- **Filtro de Suavização ou Passa-Baixas:** O efeito do filtro passa-baixa é o de suavização da imagem à custa de atenuar as regiões de bordas e detalhes finos da imagem. Dentre os filtros passa-baixas mais usados estão o filtro de média e/ou média ponderada e o filtro Gaussiano, e o resultado da convolução é dividido por um fator de normalização. O filtro de média dá pesos iguais a todos os valores da máscara (vide Figura 3.5a), o filtro de

média ponderada dá um peso maior ao valor central da máscara (vide Figura 3.5b) e o filtro Gaussiano obtém os valores da máscara a partir da distribuição discreta da função Gaussiana (vide Figura 3.5c) [28].

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \frac{1}{10}\begin{bmatrix}1 & 1 & 1\\ 1 & 2 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

(a) **(b)** **(c)**

Figura 3.5. Exemplos de filtros passa baixas:(a) vertical, (b) horizontal, (c) diagonal $+45^o$ e (d) diagonal -45°

Na Figura 3.6 mostra-se o resultado da filtragem para a máscara da Figura 3.5c.

(a) Imagem original

(b) Imagem filtrada

Figura 3.6. Filtragem espacial passa-baixa

- **Filtro de aguçamento, ênfase ou Passa-Altas:** Os filtros passa-alta tendem a realçar

os detalhes da imagem, tais como bordas, linhas ou variações de brilho da imagem. Dentre os filtros passa-alta encontrassem os filtros de gradiente ou derivada de imagens e o filtro Laplaciano [28]. Filtros de gradiente permitem detectar as bordas em diferentes direções da imagem e realçá-las (vide Figura 3.7); o filtro Laplaciano utiliza uma máscara onde o valor central tem sinal oposto aos sinais dos valores na periferia (vide Figuras 3.7a e 3.7b) [28].

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

(a)

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

(b)

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

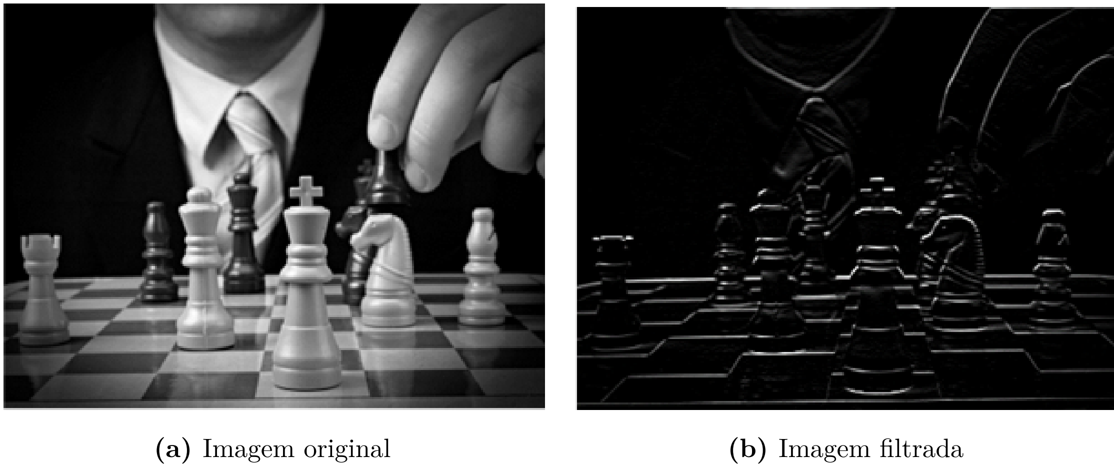
(c)

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

(d)

Figura 3.7. Exemplos de filtros passa-altas:(a) vertical, (b) horizontal, (c) diagonal $+45^o$ e (d) diagonal -45°

Na Figura 3.8 mostra-se o resultado da filtragem para uma máscara mostrada na Figura 3.7b. O resultado da filtragem da imagem original é um forte realce na direção horizontal.



(a) Imagem original **(b)** Imagem filtrada

Figura 3.8. Filtragem espacial passa-alta

3.3.2 Função de espalhamento de ponto (PSF)

A função de espalhamento de ponto, PSF (*Point Spread Function*), representa a resposta de um sistema para uma fonte pontual, se o sistema de aquisição for perfeito o ponto seria equivalente a uma função delta de Dirac em duas dimensões [1]. A Figura 3.9 apresenta a

PSF através de um diagrama esquemático. Analisando um ponto qualquer de um objeto, a luz refletida nele chega até o sistema visual [33]. Quando a imagem é formada, diz-se que a imagem original do objeto foi convoluída com a PSF [33]. A PSF é a representação da função de degradação [34].

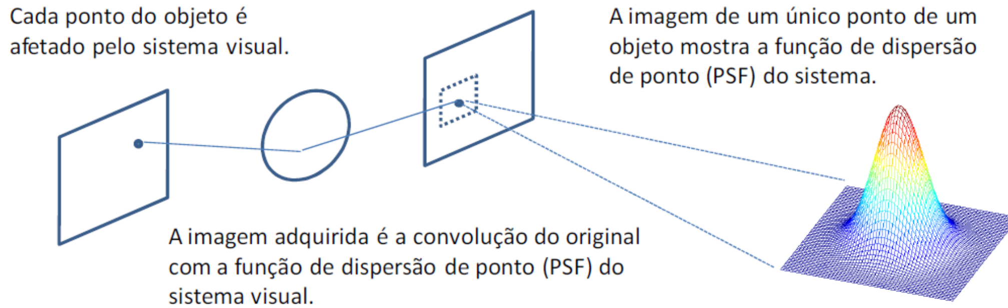


Figura 3.9. Diagrama esquemático da função de dispersão de ponto [33]

Para o caso de imagens degradadas por movimento, assumindo uma cena estática, distante e ignorando os defeitos ópticos do sistema de visão, cada ponto da imagem pode ser modelada como a convolução entre a imagem real e a PSF [35]. Neste caso, a PSF é a máscara que descreve a trajetória de movimento da posição daquele ponto [35]. Uma suposição comum em borramento por movimento é que a PSF é espacialmente invariável [35]. Isto significa que a máscara que descreve o movimento é global, ou seja, os movimentos dos objetos na cena são ignorados [35]. Neste trabalho utilizou-se a PSF como função global, ou seja, o borramento é espacialmente invariante.

3.4 RESTAURAÇÃO DE IMAGENS

A degradação de uma imagem pode ser causada por (a) distorções atmosféricas, (b) aberrações óticas, (c) efeitos do sensor, (d) borramento por movimento, e (e) ruído [34]. A restauração de imagens é muito usada em áreas que envolvem imagens, por exemplo: astronomia, sensoriamento remoto, microscopia, imagens médicas, ou sistemas de *High Definition TV* (HDTV) [34]. O modelo da degradação da imagem pode ser expressado como mostrado na equação 3.3.

$$g(x, y) = h(x, y) * f(x, y) + n(x, y), \tag{3.3}$$

no qual $f(x, y)$ é a imagem original, $g(x, y)$ é a imagem degradada e ruidosa, $h(x, y)$ é a PSF, $n(x, y)$ é o ruído aditivo e $*$ é o operador de convolução [1]. O ruído sempre estaria presente, então se assumem simplificações do problema básico de restauração sem considerar o ruído

aditivo (ruído branco e ruído impulsivo). No processo de restauração da imagem o ruído aditivo nas imagens pode ser reduzido com filtros Gaussianos ou usando o filtro da mediana [6]. Neste contexto, a restauração de imagens usa informações *a priori* sobre as degradações de maneira que seja possível aplicar o processo inverso para remover essa distorção [36].

3.4.1 Algoritmo Richardson-Lucy

O algoritmo Richardson-Lucy (RLA) é um método de deconvolução que consiste em calcular a imagem mais próxima da imagem real a partir de uma imagem observada [37]. O borramento das imagens por movimento pode ser modelado como a convolução da imagem sem borramento e a PSF [37]. O RLA é um método iterativo para restauração de imagens baseado no princípio de máxima verossimilhança usando o modelo de Poisson [37]. Para restaurar uma imagem o RLA é derivado do teorema de Bayes, baseado na ideia de que a imagem de entrada, a PSF e a imagem de saída podem ser consideradas como probabilidades [1]. O teorema de Bayes estabelece uma relação entre distribuições de probabilidade (conforme a equação abaixo) [1]:

$$p(O|I) = \frac{p(I|O)p(O)}{p(I)}, \tag{3.4}$$

no qual $p(O|I)$ é a distribuição de probabilidade da imagem restaurada, $p(I|O)$ é a distribuição de probabilidade da PSF, $p(O)$ é a distribuição de probabilidade da imagem real e $p(I)$ é a distribuição de probabilidade da imagem borrada [1]. A probabilidade da imagem observada esta dada por:

$$p(I|O) = \prod_{x,y} \frac{(h(x,y) * f(x,y))^{g(x,y)}\, exp\,(-(h(x,y) * f(x,y)))}{g(x,y)!}, \tag{3.5}$$

no qual o máximo pode ser calculado tomando a derivada do logaritmo:

$$\frac{\partial ln\ p(I|O)(x,y)}{\partial O(x,y)} = 0, \tag{3.6}$$

no qual assumindo que a PSF esta normalizada, isto leva a uma equação que pode ser resolvida iterativamente segundo Picard [4], como apresentado na equação a seguir :

$$O^{n+1}(x,y) = \left[\frac{I(x,y)}{(P * O^n(x,y))} * P^T(x,y) \cdot O(x,y)\right], \tag{3.7}$$

no qual $P^*(x,y) = P(-x,-y)$, P é a PSF e P^T é a transposta da PSF, I é a imagem observada, O^n é a estimação da imagem real e $*$ é o operador de convolução [1].

É importante ressaltar que o ruído aditivo é amplificado conforme passa cada iteração, assim quando o número de iterações tende ao infinito o resultado da imagem após a restauração é ruído [8]. Na prática podem ser estabelecidos critérios de parada para o número de iterações no intuito de diminuir a quantidade de amplificação do ruído [8]. Um desses critérios pode ser medir a qualidade da imagem restaurada comparando-a com a imagem original.

3.5 MÉTRICAS DE QUALIDADE EM IMAGENS

As métricas de qualidade de imagens são usadas para dimensionar distorções que reduzem a qualidade visual da imagem [33]. As distorções podem ocorrer durante a aquisição, processamento, compressão, armazenamento, transmissão e reprodução das imagens [38]. A avaliação da imagem pode ser realizada usando dois tipos de métodos: objetivos ou subjetivos [38]. Os métodos objetivos usam algoritmos que tentam simular o comportamento do sistema visual humano (SVH) produzindo a avaliação da qualidade [38]. Na prática, um método objetivo pode ser usado dinamicamente para monitorar e ajustar a qualidade da imagem [33]. Além disso, este método pode ser usado na otimização de algoritmos e parâmetros das configurações de sistemas de visão computacional [33]. Nos métodos subjetivos, as notas de qualidade da imagem são produzidas por observadores humanos, resultando em um processo complexo, que pode levar a resultados mais precisos que os métodos objetivos [33]

O processo de avaliação para métricas objetivas podem ser classificados como se segue: métricas de referência completa (FR- *Full Reference*), métricas de referência reduzida (RR- *Reduced Reference*) ou métricas sem Referência (NR- *No Reference*) [38, 39]. Nas métricas FR, é medida a fidelidade entre uma imagem distorcida e uma imagem original, ou seja, uma imagem que não está contaminada com erros ou distorcida [38]. A métrica NR analisa a imagem processada sem a necessidade de uma referência [38]. Este método sempre necessita fazer suposições sobre o conteúdo da imagem e/ou sobre as distorções de interesse [38]. As métricas RR utilizam certa quantidade de características da imagem de referência, como por exemplo detalhes espaciais, para executar a comparação com a imagem de teste [38].

Com o intuito de avaliar a qualidade das imagens são usadas duas abordagens: (a) medição da fidelidade da sinal (neste caso os pixels de uma imagem) e (b) os métodos inspirados nas características da percepção humana [38]. Os dois métodos de medição de fidelidade mais usados e de simples implementação são: (a) erro médio quadrático (*Mean Squared Error*-MSE) e (b) razão sinal-ruído de pico (*Peak Signal-to-Noise Ratio*-PSNR) [38, 39]. No entanto, estes dois métodos não estão adaptados ao sistema de qualidade visual percebido, ou seja, não podem dar uma estimativa real da qualidade das imagens baseada nas características desta [39]. Neste contexto, uma das métricas FR mais populares é o índice de similaridade estrutural (*Structural*

Similarity-SSIM). A seguir será descrita a métrica de avaliação objetiva SSIM.

O MSE não é uma boa métrica para a avaliação de imagens devido ao fato de comparar pixel a pixel ao invés de avaliar as características da imagem (estrutura, contraste e luminância). A Figura 3.10 ilustra a diferença entre o MSE e o SSIM na avaliação da qualidade de imagens distorcidas [39]. Para as três imagens distorcidas, o MSE calculado foi aproximadamente o mesmo. Entretanto com o SSIM, a imagem (d) apresenta uma qualidade muito inferior às imagens (b) e (c). O seja, a métrica de qualidade SSIM se assemelha mais à percepção humana da qualidade do que o MSE.

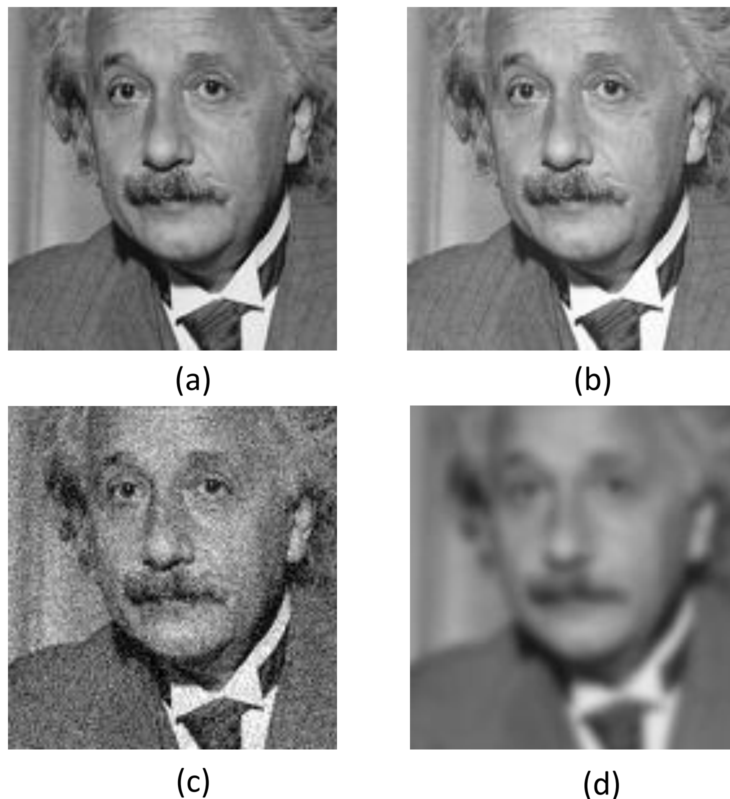


Figura 3.10. Comparação de métricas de qualidade para imagens distorcidas [39]. (a) Einstein original. MSE=0 e SSIM=1, (b) Einstein distorcido com mudança de luminância. MSE=309 e SSIM=0,987, (c) Einstein distorcido com ruído Gaussiano. MSE=309 e SSIM=0,576, (d) Einstein distorcido por borramento. MSE=308 e SSIM=0,641

3.5.1 *Structural similarity index* (SSIM)

O SSIM é definido como uma métrica objetiva FR que estima a qualidade de um imagem utilizando aspectos da percepção visual humana. Esta métrica considera as características estruturais da imagem (desvio, média, correlação) [40]. As distorções podem ser de origem estrutural

e não-estrutural [39]. Dentre as distorções estruturais encontram-se (a) borramento, (b) contaminação por ruído, (c) compressão JPEG, entre outras [39]. As distorções não-estruturais são (a) mudanças na iluminação, (b) mudança de contraste, (c) distorção gamma e (d) mudança espacial [39]. Uma boa medida de mudança estrutural indica o grau de aproximação de distorção da imagem adquirida [33, 40]. A SSIM usa a combinação de três componentes: (a) luminância, (b) contraste e (c) estrutura (vide Equações 3.8, 3.9, 3.10), as quais são comparadas e combinadas para gerar um escore de similaridade entre duas imagens [39, 40]. Na Figura 3.11, é apresentada a métrica SSIM proposta por [40], onde x é sinal original (sem perda de qualidade) e y é a imagem degradada.

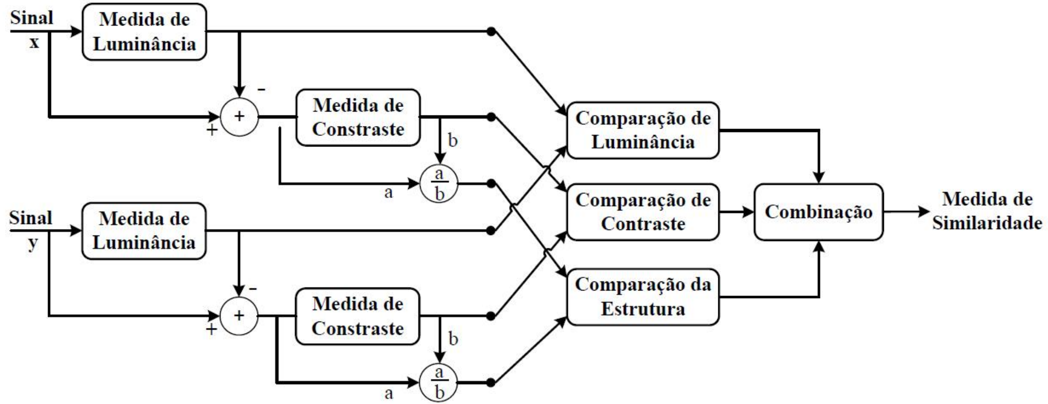


Figura 3.11. Diagrama esquemático da medida do índice de similaridade estrutural (SSIM) [38]

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{3.8}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{3.9}$$

$$s(x,y) = \frac{2\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}, \tag{3.10}$$

no qual μ_x, μ_y, σ_x^2, σ_y^2 e σ_{xy} são a média de x e y, a variância de x e y e a covariância cruzada de x e y, respectivamente [38]. Os termos C_1, C_2 e C_3 são constantes definidas como $C_1 = (K_1L)^2$, em que L é um valor dinâmico de pixel (255 para imagens em tom cinza de 8 *bits*) e $K_1 \ll 1$; $C_2 = (K_2L)^2$ e $K_2 \ll 1$; $C_3 = \frac{C_2}{2}$[38]. A expressão geral do índice SSIM é expresso como:

$$SSIM = [l(x,y)]^\alpha \cdot [c(x,y)]^\beta \cdot [s(x,y)]^\gamma, \tag{3.11}$$

no qual α, β e γ são usados para simplificar a equação do SSIM. Substituindo as equações 3.8, 3.9 e 3.10 em 3.11 obtemos a seguinte expressão:

$$SSIM == \left(\frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}\right)^\alpha \left(\frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}\right)^\beta \left(\frac{2\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}\right)^\gamma, \tag{3.12}$$

usando $\alpha = \beta = \gamma = 1$, o que resulta na seguinte expressão:

$$SSIM = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x\sigma_y + C_2)} \tag{3.13}$$

A métrica SSIM tem diversas extensões, entre as quais estão o MS-SSIM (*the multiscale SSIM index*), o IW-SSIM (*the information content weighted SSIM index*), o FSIM (*the feature similarity index*) e o SR-SIM (*Spectral Residual Based Similarity*) [41]. Dentre os quais, o SR-SIM possui uma baixa complexidade computacional e está baseado no modelo da saliência visual espectral residual (*Spectral Residual Visual Saliency*-SRVS) [41]. O SR-SIM possui duas funções: (a) avaliar a qualidade das regiões de uma imagem e (b) criar uma ponderação para avaliar a importância de uma região para o sistema visual humano. A SRVS permite a detecção de objetos a partir de seu fundo usando um mapa de saliência visual. *Zhang e Li* propõem utilizar o mapa SRVS para calcular o mapa de similaridade local entre uma imagem de referência e a imagem distorcida [41]. A desvantagem do SRVS é que é débil para destacar as transições de contraste da imagem. Uma solução para este problema foi aplicar o operador gradiente sobre a imagem e calcular seu módulo (G). Assim, a implementação da métrica SR-SIM faz uso do SRVS e do G como é apresentado nas equações a seguir:

$$S_V = \frac{2R_1 \cdot R_2 + C_1}{R_1^2 + R_2^2 + C_1}, \tag{3.14}$$

$$S_G = \frac{2G_1 \cdot G_2 + C_2}{G_1^2 + G_2^2 + C_2}, \tag{3.15}$$

no qual R_1 e R_2 são os mapas SRVS das imagens, G_1 e G_2 são os gradientes das imagens usando como operadores o gradiente de Scharr (vide Equações 3.16 e 3.17) e C_1 e C_2 são constantes [41].

$$G_x = \frac{1}{16}\begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix} * f \tag{3.16}$$

$$G_y = \frac{1}{16}\begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} * f \tag{3.17}$$

$$G = \sqrt{G_x^2 + G_y^2} \tag{3.18}$$

Assim, S_V e S_G são usadas para obter a similaridade entre duas imagens [41]. A expressão é descrita tal como mostrado na equação 3.19.

$$S = S_V \cdot [S_G]^{\alpha}, \tag{3.19}$$

no qual α é usada para ajustar as duas componentes de S. Finalmente, a expressão para o SR-SIM é definida a seguir.

$$SRSIM = \frac{\sum S_V \cdot [S_G]^{\alpha} \cdot R_m}{\sum R_m}, \tag{3.20}$$

no qual $R_m = max(R_1, R_2)$ é considerado como o peso de S na similaridade global.

Segundo [39] a métrica SR-SIM oferece uma melhor avaliação, um baixo custo computacional e pode ser implementável em um sistema embarcado. Com tudo, o SR-SIM pode ser usado para aplicações em tempo real como sensoriamento remoto, reconhecimento aéreo, câmeras localizadas em satélites, entre outras. Esta métrica será usada neste trabalho como métrica de avaliação das imagens.

3.6 TRABALHOS CORRELATOS À IMPLEMENTAÇÃO DE RESTAURAÇÃO DE IMAGENS EM *HARDWARE*

Na restauração de imagens, alguns artigos reportam implementações de deconvoluções em 2-D, principalmente para aplicações em sistemas embarcados com requerimentos de tempo real. Existem arquiteturas para restauração de imagens baseados no RLA, embora sejam usadas em outras plataformas que não o FPGA.

Na referência [1] pode-se encontrar a implementação em FPGA da combinação de dois filtros separáveis para obter a resposta de um filtro não separável, usando o *Singular Value Decomposition* (SVD). O SVD permite obter uma rápida convergência da imagem restaurada e poupar o número de recursos lógicos. Esta arquitetura usa menos recursos de *hardware* que uma convolução em 2D. No entanto, esta estrutura elimina os efeitos nas bordas diminuindo assim o tamanho da imagem restaurada. A arquitetura implementada está baseada na equação 3.21.

$$I^{n+1} = I^n \cdot \left(PSF * \frac{Observation}{PSF * I^n} \right)^\beta, \tag{3.21}$$

no qual PSF é a função de espalhamento de ponto, *Observation* é a imagem borrada, I^n é a imagem restaurada na iteração n, β é a constante para a aceleração da convergência e $*$ é o operador de convolução [1]. O expoente β é usado com valores entre 1 e 3 (na primeira iteração), gerando um decremento no número de iterações a serem executadas. O sistema usa uma máscara de 11×11 pixels e pode alcançar uma saída de 30 fps com imagens de 640×480 pixels. Esta implementação alcança uma taxa de 60 MP/s (milhões de pixels por segundo) e uma frequência máxima de 63 MHz.

Na referência [8] é proposto um método para acelerar o RLA e estabelecer um critério de parada. A implementação é feita em *software* para resoluções de imagens de 256×256 pixels. A técnica é denominada amortecimento do algoritmo Richardson-Lucy (*Damped Lucy-Richardson Algorithm-DRLA*). O DRLA maximiza a probabilidade de restaurar a imagem degradada. A desvantagem deste algoritmo é que ele tem melhor desempenho quando a PSF é conhecida, ou seja, com informação *a priori*. O DRLA apresenta uma melhor restauração das imagens degradadas, além de acelerar a convergência reduzindo o número de iterações.

Em [42], Diewald *et al.* apresentam uma modificação do RLA adaptado para uma imagem com distribuição *chi-quadrado*. O algoritmo foi implementado para uso em imagens que capturadas por radares de trânsito com a finalidade de estimar a largura dos veículos e dos pedestres. Os resultados dos testes mostram que o algoritmo melhora a relação sinal-ruído e permite estimar a largura depois da deconvolução. Outra abordagem para a aceleração do RLA é feita em [43], usando o método *Scaled Heavy-Ball* para a convergência mais rápida do algoritmo em imagens 3D para imagens microscópicas.

Um método de aceleração das iterações fazendo uma estimativa do vetor de movimento é proposto na referência [9], usando a função da equação 3.21. Esta técnica permite calcular o parâmetro de aceleração das iterações e aumentar a velocidade de convergência da imagem restaurada. O processamento das imagens capturadas é feito usando o formato fornecido pela câmera para resoluções de imagem de 128×128 pixels. No entanto, Prato *et al.* assegura que este método não apresenta resultados confiáveis, tendo um desvio na estimativa da trajetória [44]. Neste trabalho é feita a implementação dos algoritmos: (a) RLA, (b) *Scaled gradient projection* (SGP) e (c) *Ordered subset expectation maximization* (OSEM). Além disso, Prato *et al.* reduz o problema dos efeitos de borda nas imagens restauradas. As implementações foram feitas em uma GPU (CUDA) e uma CPU.

Em [45], Angelopoulou *et al.* apresenta uma arquitetura para acrescentar o desempenho e a resolução de um sistema de visão em tempo real. A resolução espacial de uma imagem depende

do tamanho do pixel, da qualidade do sensor do sistema de aquisição e também da distância entre o sensor e o alvo [46]. O incremento na resolução espacial traz consigo a obtenção de uma imagem mais detalhada. Isto gera uma redução da resolução temporal, ou seja, o sistema de visão necessita de mais tempo para adquirir e processar a imagem, em consequência a imagem sofre de borramento. Neste caso, a restauração de imagens está baseada na técnica de Super-Resolução (*Super-Resolution-SR*), onde usa-se uma sequência de $frames$ que estão degradados para obter assim uma imagem sem degradação. A FPGA tem uma interface com o sensor adaptativo de imagem para localizar as regiões que têm sofrido borramento, processando-as em tempo-real. A saída do sistema está em torno aos 25 fps, alcançando uma frequência máxima de 80 MHz, para resoluções de imagem de 480×640 pixels.

Em [47], ZHEN apresenta uma arquitetura para a restauração de imagens via *software* e usando uma FPGA. A função de espalhamento de ponto (PSF) é estimada usando um acelerômetro e o NIOS II. A PSF é armazenada em uma *SD Card* junto com a imagem capturada. A deconvolução é feita utilizando o algoritmo *Sparse Prior* (SP) em outra plataforma computacional para testar a validez da PSF. O processamento das imagens capturadas é feito usando o formato fornecido pela câmera para resoluções de imagem de 800×480 pixels.

A distorção de uma imagem pode ser assumida sendo linear e espacialmente invariante, ou seja, a degradação é uniforme em toda a imagem [35]. Na referência [35] é apresentado uma modificação do RLA para restaurar imagens com degradações espacialmente variantes (vide Figura 3.12).

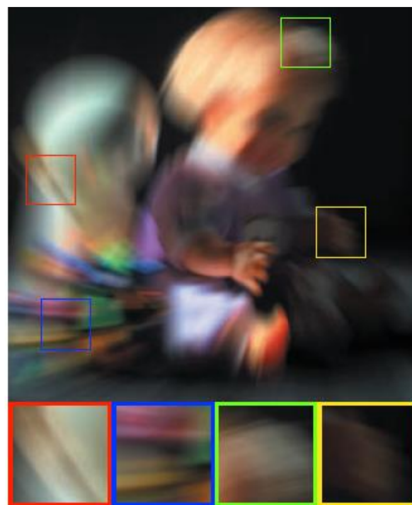


Figura 3.12. Imagem degradada por movimento espacialmente variante [35]

O resumo dos resultados obtidos nos trabalhos correlatos pode ser observado na Tabela 3.2. Neste caso, pode ser observado que os algoritmos de restauração de imagens baseados no algo-

Tabela 3.2. Comparação das diferentes implementações do RLA

Ref.	Método	Tamanho da imagem	Q/seg	Plataforma	Métrica	Acc.	Iter.
[1]	RLA	640×480	30	Virtex 2 XC2VP50	Visual	sim	2
[44]	RLA SGP OSEM	256×256 512×512	Não	GPU CPU	error	sim	10000
[43]	RLA	$672 \times 712 \times 104$ $256 \times 256 \times 128$ $1004 \times 712 \times 51$	Não	*software*	PSNR SSIM	sim	200
[35]	RLA	512×512	Não	*software*	RMS	sim	5000
[42]	RLA	Não	Não	*software*	RMSE	não	1
[9]	RLA	128×128	Não	*software*	MSE	sim não	250 10000
[8]	DRLA	256×256	Não	*software*	linearidade fotométrica	sim	400
[45]	SR	256×256 480×640	25	Virtex 4 ADMXRC4SX	RMSE SNR	sim	41
[47]	*Sparse Prior*	800×480	30	Cyclone IV-E software	Visual	não	não

ritmo RLA (implementados em *software*) necessitam de muitas iterações para obter a solução da imagem real. Assim, suas aplicações em tempo real são limitadas pelo quesito de desempenho.

No caso do item *aceleração* (vide coluna 7 da Tabela 3.2) pode ser observada que a mesma pode ser obtida via *software* (com modificações do algoritmo, por exemplo usando aproximações de cálculo numérico. Por outro lado, as acelerações obtidas via módulos de *hardware* foram feitas via FPGA (vide referências [1], [45] e [47]) ou usado GPUs (vide referência [44]). Entretanto, no uso do algoritmo RLA pode ser observado que somente um trabalho usou um FPGA como módulo de aceleração (vide referência [1]). Neste caso, pode ser observado que o número de iterações reportado nesse trabalho foi de 2. Uma desvantagem desse trabalho foi que nenhuma métrica objetiva foi reportada para verificar a qualidade da imagem restaurada. Uma outra limitação que pode ser observada é que a técnica usada não é baseada em convolução, mas em SVD. Isto pode trazer alguns problemas devido ao uso de uma decomposição baseada em filtros separáveis, o qual implica na necessidade de se ter uma aproximação da PSF. Esta aproximação permite um decremento dos recursos de *hardware* necessários no mapeamento do algoritmo.

Entretanto, traz uma perda na qualidade da solução obtida.

A discussão apresentada no parágrafo anterior deixa aberta a possibilidade de explorar soluções para o RLA onde as interações no algoritmo clássico possam ser mapeadas em *hardware* via um *pipeline* (*unrolling loop*), fazendo um estudo criterioso sobre a qualidade da solução (medida via uma métrica adequada) e o número de iterações necessárias para se obter uma imagem com um bom grau de restauração.

3.7 CONCLUSÕES DO CAPÍTULO

Neste capítulo foram explicados os conceitos básicos sobre imagens e o seu processamento assim como o algoritmo que será implementado para a restauração e a avaliação das imagens neste trabalho.

O algoritmo Richardson-Lucy é um algoritmo de fácil implementação mas seu grande inconveniente é a amplificação de ruído e o seu critério de parada. Uma solução para este problema é a utilização de filtros no começo da restauração ou durante a restauração da imagem. Por outro lado, o RLA é um algoritmo iterativo (é necessário armazenar mais de uma imagem) e de alto custo computacional (devido a suas duas convoluções), a solução para este problema é o mapeamento em uma arquitetura paralela e a divisão das iterações em um laço aberto (*loop unrolling*).

4 IMPLEMENTAÇÃO DO ALGORITMO PARA A RESTAURAÇÃO DE IMAGENS

Neste capítulo será descrita a implementação da arquitetura para restauração de imagens, baseada no algoritmo Richardson-Lucy. A arquitetura proposta foi mapeada em um dispositivo FPGA da família Cyclone IV-E da Altera. A implementação foi validada por meio de experimentos computacionais, permitindo assim realizar avaliações de qualidade das imagens fornecidas pela arquitetura implementada em *hardware* e comparadas com as suas imagens de referência (imagem restaurada em *software*).

4.1 CONSIDERAÇÕES INICIAIS

Algoritmos iterativos podem aumentar o seu nível de paralelismo usando uma técnica de paralelização de seus laços denominada *loop unrolling*. Esta técnica organiza a arquitetura em um arranjo de N estágios de processamento (vide Figura 4.1). Como observado na Figura 4.1, a arquitetura pode ser replicada N vezes, de tal forma que a cada ciclo de relógio ela forneça um pixel útil para a seguinte iteração. O RLA necessita de duas imagens de entrada (observada e restaurada) de $m \times n \times 8$ bits e de várias iterações, como visto na seção 3.4.1. Porém, devido ao tamanho das imagens é preciso salvar estas duas imagens em memórias externas, a cada iteração. Assim, a arquitetura fica limitada pelo tamanho das imagens em memória e pela sincronização entre o FPGA e a memória externa. Com tudo, aproveitando o nível de paralelismo que possui o FPGA, a arquitetura do RLA pode ser replicada e organizada em um *pipeline* sem o uso de memórias externas.

4.1.1 Função de espalhamento de ponto (PSF)

Devido ao fato deste trabalho focar no problema de restauração de imagens, desconsi-derou-se o ruído aditivo (ruído gerado pelo sistema de captura) do modelo da imagem (vide Equação 3.3). Por outro lado, as máscaras usadas para a convolução com a imagem (vide seções 3.3.1 e 3.3.2) foram calculadas *offline* usando a função *fspecial* de *Matlab*. O *toolbox* de processamento de

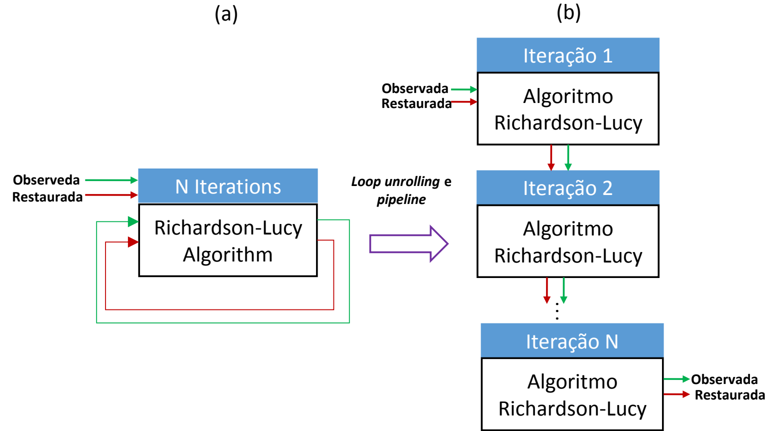


Figura 4.1. Exemplo do *loop unrolling* do RLA. (a) Implementação em *software* e (b) implementação em *hardware*

imagens de *Matlab* disponibiliza a função *fspecial*, esta função fornece filtros bidimensionais a partir do tipo de filtragem desejado e uns parâmetros de entrada (para o caso de borramento, deslocamento x e ângulo θ). Os tamanhos das máscaras podem ser desde 1×3 até 9×9 pixels, para deslocamentos entre três e nove pixels. A Tabela 4.1 mostra a estrutura geral da função *fspecial* do *Matlab*. Neste trabalho foi usado a degradação por movimento (*motion*) para gerar a PSF.

Tabela 4.1. Alguns filtros gerados usando a função *fspecial* do *toolbox* de *Matlab*

Tipo	Parâmetro	Máscara
Motion	Deslocamento (x) ângulo (θ)	para $x = 3$ e $\theta = 30^o \rightarrow \begin{bmatrix} 0 & 0 & 0,1647 \\ 0,1647 & 0,3414 & 0,1647 \\ 0,1647 & 0 & 0 \end{bmatrix}$
Average	Tamanho da máscara ($n \times n$)	para $n = 3 \rightarrow \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
Laplacian	Controle do contorno (α)	para $\alpha = 0,5 \rightarrow \begin{bmatrix} 0,3333 & 0,3333 & 0,3333 \\ 0,3333 & -2.6667 & 0,3333 \\ 0,3333 & 0,3333 & 0,3333 \end{bmatrix}$
Gaussian	Tamanho da máscara ($n \times n$) Desvio padrão (σ)	para $n = 3$ e $\sigma = 0,5 \rightarrow \begin{bmatrix} 0,0113 & 0,0838 & 0,0113 \\ 0,0838 & 0.6193 & 0,0838 \\ 0,0113 & 0,0838 & 0,0113 \end{bmatrix}$

A Figura 4.2(a) mostra a abordagem usada para o carregamento da vizinhança para a ope-

ração de convolução, via multiplexação. O sistema a ser projetado deve disponibilizar a possibilidade de se trabalhar com máscaras de tamanhos variados (3×3, 5×5, 7×7 e 9×9). Isto é importante porque o RLA é sensível ao nível de borramento da imagem. Por exemplo, se o borramento é pequeno (produzido por um deslocamento pequeno, de poucos pixels) a máscara a ser usada na convolução pode ser de 1×3. Borramento envolvendo mais pixels precisariam de máscaras maiores.

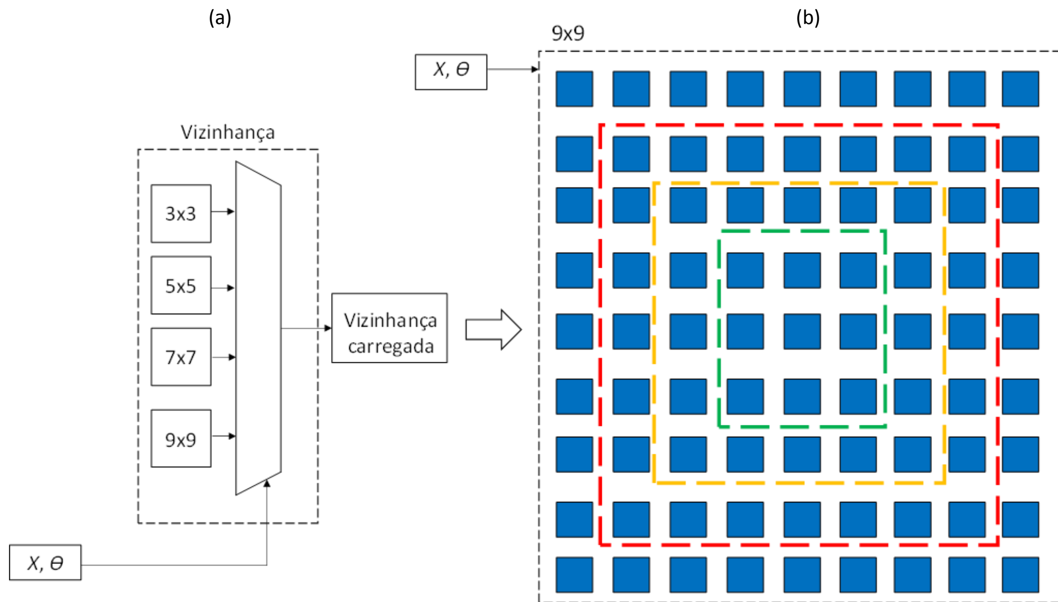


Figura 4.2. Configuração das máscaras para a PSF. (a) Primeira abordagem e (b) Segunda abordagem

Desta maneira é necessário a disponibilização de uma arquitetura de *hardware* que possibilite a escolha do tamanho da máscara, dependendo da necessidade específica. Esta informação do deslocamento é proveniente do percurso da câmera (deslocamento e ângulo, vide seção 3.3.2). Neste caso, esta informação está relacionada com a PSF, a qual está implícita nos coeficientes das respectivas máscaras.

Entretanto, o uso de multiplexadores para viabilizar o uso de máscaras de diferentes tamanhos requer a implementação de carregadores de vizinhanças separados para cada tipo de máscara (vide figura Figura 4.2(a)). Adicionalmente, para executar a somatória implícita em cada convolução são necessários circuitos de sincronização, tendo em conta que cada convolução vai ter tempos de execução diferentes, dependendo do tamanho da máscara.

Para resolver os problemas supracitados, optou-se por um sistema de carregamento de vizinhança, tal com apresentado na Figura 4.2(b). Neste caso, os blocos ressaltados na figura 4.2(b) têm as seguintes configurações para as máscaras: (a) cor verde, 1×3 até 3×3, (b) cor amarela, 1×5 até 5×5, (c) cor vermelha, 1×7 até 7×7 e (d) cor preta, 1×9 até 9×9 pixels.

Neste sentido, uma única vizinhança máxima de tamanho 9×9, o que permite trabalhar com máscaras menores, de acordo com o deslocamento que gerou o borramento da imagem.

Na figura 4.2(b) pode ser observado que existe uma entrada (x, θ) a qual tem o potencial de selecionar o tamanho da máscara, considerando que uma vizinhança de tamanho $n \times n$ já está previamente carregada. No caso de se precisar de uma máscara de 3×3, somente os valores correspondentes a este tipo de máscara serão diferentes de zero (vide área verde da Figura 4.2(b)). Na arquitetura proposta, os valores para cada tipo de máscara (relacionada com cada PSF possível) estão previamente armazenados na memória interna da FPGA (calculados *off-line*). Por exemplo, se o deslocamento for de $X = 5$ pixels e ângulo θ a máscara gerada é mostrada na Figura 4.3(a), assim, os valores na área verde são preenchidos com os valores da máscara de 5×5 (vide Figura 4.3(b)) e os demais valores fora da área amarela são iguais a zero. Se o deslocamento corresponde a $X = 3$ pixels e ângulo θ somente os valores na área verde são preenchidos com a máscara de 3×3 (vide área verde da Figura 4.3(b) e (c)) e os demais valores fora da área verde são igual a zero.

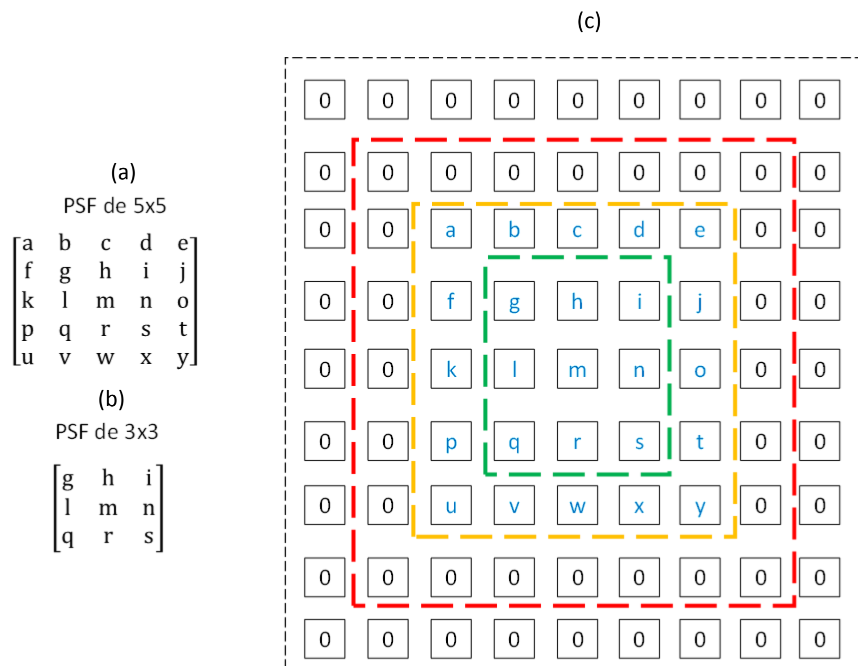


Figura 4.3. Preenchimento dos valores da PSF para uma máscara global de 9×9 pixels. (a) PSF para deslocamentos de $X = 5$ pixels, (b) PSF para deslocamentos de $X = 3$ pixels e (c) máscara global

4.2 IMPLEMENTAÇÃO DO ALGORITMO RICHARDSON-LUCY EM *SOFTWARE*

Nesta seção a implementação realizada do RLA em *software* será descrita. A implementação do citado algoritmo foi codificada em *Matlab*, que inclui várias funções já predefinidas, que podem ser utilizadas na implementação direita do algoritmo. O RLA é apresentado no Algoritmo 1.

Algoritmo 1: Implementação em *Matlab* do RLA (Equação 3.21 com $\beta = 1$)

Entrada: Imagem observada (*Observation*) em escala de cinza de tamanho $m \times n$
Função de espalhamento de ponto (psf)
Número de iterações ($iter$)
Imagem restaurada na iteração n (I^n), para $n = 1$ $I^n = ones(m, n)$

Saída : Imagem restaurada I^{n+1} em escala de cinza

1 $Observation = rgb2gray(Observation)$; `// conversão de RGB para escala de cinza`

2 **for** *n=1* **to** *iter* **do**

3 $imag_obser = imfilter(I^n, psf)$; `// convolução entre a imagem restaurada na iteração anterior e a PSF`

4 $imag_fator = Observation./(imag_obser + 10^{-12})$; `// divisão entre a imagem observada e a imagem convoluída na linha anterior`

5 $imag_corre = imfilter(imag_fator, psf)$; `// convolução da imagem do resultado anterior com a psf`

6 $I^{n+1} = imag_corre. * I^n$; `// imagem restaurada na iteração n+1`

A implementação para o RLA tem três entradas e uma saída; sendo que nas entradas encontram-se: (a) a imagem observada (Im) em formato de cor RGB, (b) a função de espalhamento de ponto fornecida por *Matlab* e (c) o número de iterações ($iter$). A saída do algoritmo é a imagem recuperada (*restaurada*) em escala de cinza, depois de *n*-iterações. Esse é o resultado que se espera obter com a implementação em *hardware*. Inicialmente, a escala de cores da imagem de entrada é reduzida para escala de cinza com a função *rgb2gray* de *Matlab*. A operação de convolução descrita na seção 3.3.1 é realizada de modo direto em *Matlab*, por meio da função *imfilter*.

4.3 IMPLEMENTAÇÃO DO ALGORITMO RICHARDSON-LUCY EM *HARDWARE*

Nesta seção apresentam-se os passos seguidos para a implementação do algoritmo RLA. O Algoritmo 2 apresenta o pseudocódigo do RLA baseado na Equação 3.21, no qual é usado o conceito de convolução (vide Equação 3.2), usando tamanhos de máscaras desde 1×3 até 9×9 pixels. Este algoritmo começa com a informação da imagem observada (*Observation*) em escala de cinza de tamanho $m \times n$, assim como a função de espalhamento de ponto (*psf*), produzindo como saída a imagem restaurada (I^{n+1}), em escala de cinza do mesmo tamanho da imagem *Observation*.

Algoritmo 2: Pseudocódigo para uma iteração do RLA (Equação 3.21 com $\beta = 1$)

Entrada: Imagem observada (*Observation*) em escala de cinza de tamanho $m \times n$
Função de espalhamento de ponto (psf)
Imagem restaurada na iteração n (I^n), para $n = 1$ então $I^n \leftarrow 1$

Saída : Imagem restaurada I^{n+1} na iteração n+1

1 $imag_prem \leftarrow Observation * \alpha$; `// pré-multiplicação da imagem observada por um valor` α `para acrescentar o valor do numerador na equação 3.21`

2 $imag_obse \leftarrow I^n \otimes psf$; `// convolução entre a imagem restaurada na iteração anterior e a psf`

3 **if** *imag_obse=0* **then**

4 $\quad imag_fator \leftarrow 255$; `// se o denominador da equação 3.21 for zero então toma o valor de 255`

5 **else**

6 $\quad imag_fator \leftarrow imag_prem \div imag_obse$; `// divisão entre a imagem pré-multiplicada e a imagem obtida da convolução anterior`

7 $imag_corre \leftarrow imag_fator \otimes psf$; `// convolução entre o a imagem obtida na linha anterior e a psf`

8 $imag_resta \leftarrow I^n * imag_corre$; `// imagem restaurada na iteração n+1`

9 $I^{n+1} \leftarrow imag_resta \div \alpha$; `// pós-divisão para obter a imagem restaurada de 8 bits`

Observa-se que o RLA é um algoritmo iterativo e sequencial; porém para aumentar o seu nível de paralelismo a arquitetura foi mapeada em um *pipeline* de N estágios (vide Figura 4.1).

A Figura 4.4 mostra a arquitetura proposta para um estágio do algoritmo. A imagem de entrada é recebida em forma de *streaming* em escala de cinza, em uma taxa de um pixel por ciclo de relógio. Como observado na linha 1 do Algoritmo 2, a imagem em tons de cinza é multiplicada por uma constante $\alpha = 10^6$ (pre-multiplicação); sendo este passo necessário para

que o formato dos números usados nos blocos subsequentes seja do tipo inteiro. Os valores das máscaras usadas são convertidos em formato inteiro e salvas em memórias internas do FPGA. Finalmente, o resultado obtido é dividido por a mesma constante usada na pre-multiplicação (vide entrada para o bloco de pós-divisão, na figura 4.4), obtendo assim a imagem restaurada. A seguir, são apresentadas as implementações em *hardware* das unidades usadas no RLA. O estágio mostrado na figura 4.4 deverá ser replicado N vezes a fim de implementar N interações do algoritmo RLA. No bloco da divisão o tamanho da palavra do dividendo é de 28 bits e o tamanho do divisor é de 22 bits, porém o resultado da divisão é de 28 bits. No entanto, o valor máximo desse resultado é de 17 bits ($255 * 10^3$, vide Equação 4.1).

$$quociente = \frac{V_max_div}{V_min_dsr} = \frac{255 * 10^6}{1 * 10^3} = 255 * 10^3, \tag{4.1}$$

no qual o V_{max_div} é o valor máximo do dividendo e o V_{min_dsr} é o valor mínimo do divisor. Assim, o valor maior mais próximo para esse tamanho da palavra que disponibiliza o Quartus II é de 24 bits.

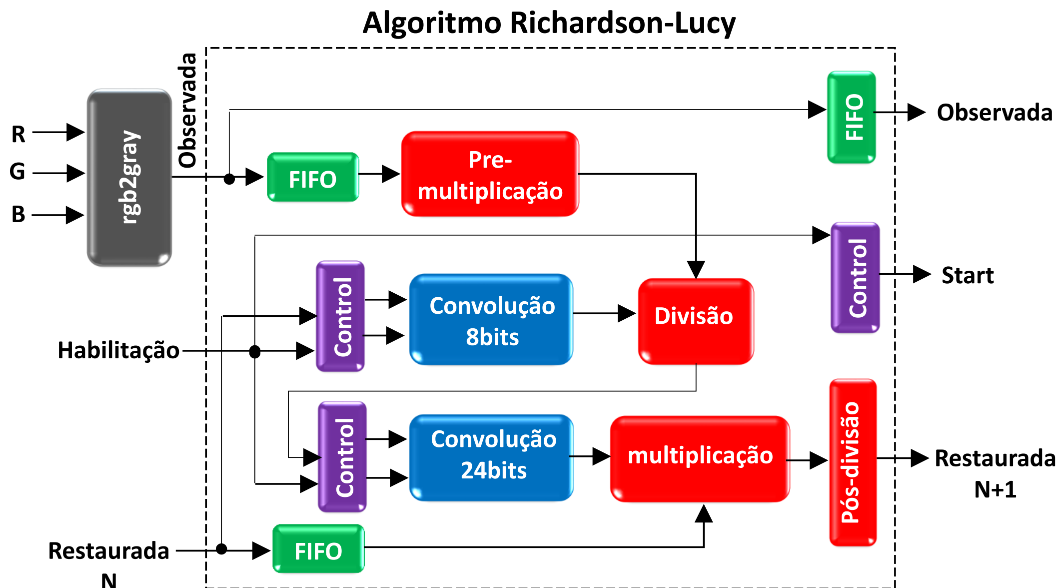


Figura 4.4. Arquitetura geral do RLA

4.3.1 Unidade de redução de cores

Por questões de simplificação, a técnica desenvolvida neste trabalho está baseada no uso de imagens monocromáticas. As imagens monocromáticas podem ser consideradas como uma simplificação das cores reais de uma imagem em apenas um plano de cor [19]. Por tanto, foi

implementado o código de conversão para tons de cinza que permite transformar os três canais de cores da imagem em apenas um canal de 8 bits. Utilizando a seguinte equação:

$$I = 0,2989 * R + 0,5870 * G + 0,1140 * B \tag{4.2}$$

no qual R é o plano vermelho (*Red*), G é o plano verde (*Green*) e B é o plano azul (*Blue*). Essa equação é usada por *Matlab* para simplificar a imagem.

Algoritmo 3: Pseudocódigo do algoritmo de conversão de imagens RGB para imagens em escala de cinza (Equação 4.2)

input: Níveis R,G,B da imagem observada

Output: Imagem em escala de cinza I

1 $I \leftarrow 2989 * R$

2 $I \leftarrow 5870 * G + I$

3 $I \leftarrow 1140 * B + I$

4 $I \leftarrow 105 * I$

5 $I \leftarrow I \div 2^{20}$

6 **return** I

A Figura 4.5 apresenta a implementação em *hardware*, referenciada como *rgb2gray* no Algoritmo 2. As constantes da Equação 4.2 estão em formato decimal e são transformadas em frações equivalentes para que as operações subsequentes fiquem em formato inteiro (por exemplo 0,2989= 2.989/10.000). Em seguida, é feita uma soma dos resultados das multiplicações entre os numeradores de cada fração e os pixels dos canais RGB. O resultado desta soma é dividido pelo denominador das frações (10.000). Para obter o resultado em formato inteiro, o resultado da soma é dividido pelo valor do denominador das frações (10.000). Este processo é realizado multiplicando-se o resultado da soma por uma constante (neste caso 105) e dividindo-se por uma potência de dois (2^{20}). Isto elimina a divisão por uma constante por um simples deslocamento (*shift register*) e um multiplicador, evitando o uso de um circuito divisor mais complexo.

4.3.2 Unidade de filtragem por convolução

O RLA faz bastante uso da filtragem por convolução. Neste trabalho, o tamanho da imagem de saída é igual ao tamanho da imagem de entrada, o que não acontece em trabalhos similares [19, 26], e também desenvolvidos no LEIA-UnB (*Laboratório de Sistemas Embarcados e Aplicações de Circuitos Integrados*). Nesses trabalhos a convolução produzia uma imagem de maior tamanho que a original. Ou seja, bordas são acrescentadas à imagem original, sendo que o tamanho das bordas vai depender do tamanho da máscara usada na convolução. Neste trabalho, foi necessário

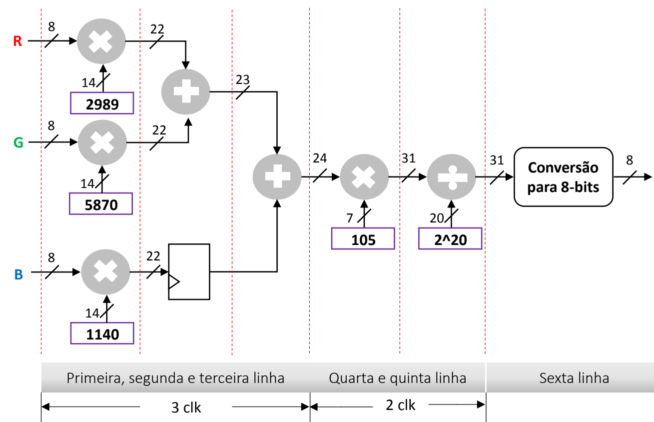


Figura 4.5. Unidade de conversão de RGB para escala de cinza

implementar uma mudança nas arquiteturas convencionais de processamento de imagens que usam a convolução [19, 26], a fim de se ter imagens sem distorções dimensionais. Para tanto, foi desenvolvido um circuito especial, que detecta os problemas de deslizamento da máscara sobre a imagem. Os problemas acontecem especificamente quando a máscara está se deslizando sobre as bordas de uma linha específica da imagem. Como o RLA usa dois passos de convolução durante o processamento, as distorções por dimensionalidade criadas pelas arquiteturas de convolução tradicionais são interpretadas como ruido. Este fato, se não for tratado por uma nova arquitetura de convolução, pode inviabilizar o processo de desborramento que é o principal objetivo deste trabalho.

Neste trabalho, foram desenvolvidos três passos necessários da filtragem por convolução sem distorções de dimensionalidade, que são apresentados a seguir:

(a) **Carregamento da vizinhança:** para o processamento de uma vizinhança é necessário que todos os dados da vizinhança estejam disponíveis.

(b) **Disponibilização da vizinhança:** disponibiliza valores dos pixels da imagem que vão ser processados.

(c) **Convolução:** os valores da vizinhança são multiplicados e somados por seus respectivos pesos da máscara.

A Figura 4.6 mostra o diagrama de blocos, implementado no Quartus II, utilizado no processo de filtragem por convolução. Esta arquitetura desenvolvida baseia-se em três blocos: (a) arquitetura de vizinhança, (b) disponibilização da vizinhança e (c) convolução. Os módulos supracitados serão detalhados a seguir.

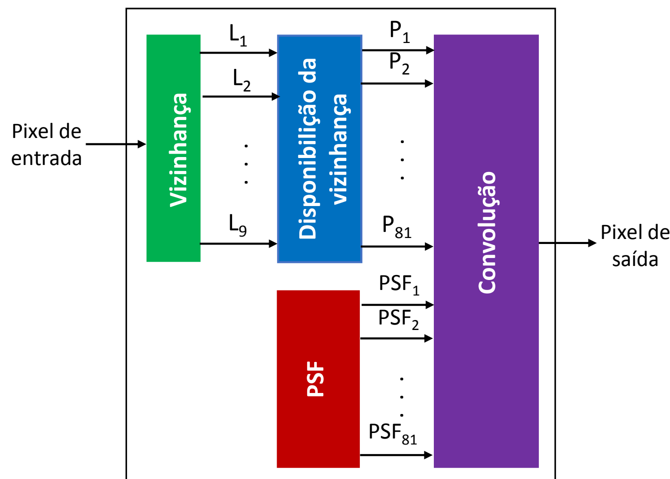


Figura 4.6. Filtragem por convolução

4.3.2.1 Arquitetura de vizinhança

A arquitetura usada para a operação de janelamento é mostrada na Figura 4.7. A arquitetura é baseada na definição matemática da filtragem por convolução; porém, em lugar do deslocamento da máscara sobre a imagem, a imagem é que é deslocada através de um *pipeline* (vide Seção 3.3.1). Para realizar esse processo em *hardware* é necessário utilizar estruturas para o armazenamento temporário de linhas da imagem e de registradores de deslocamento.

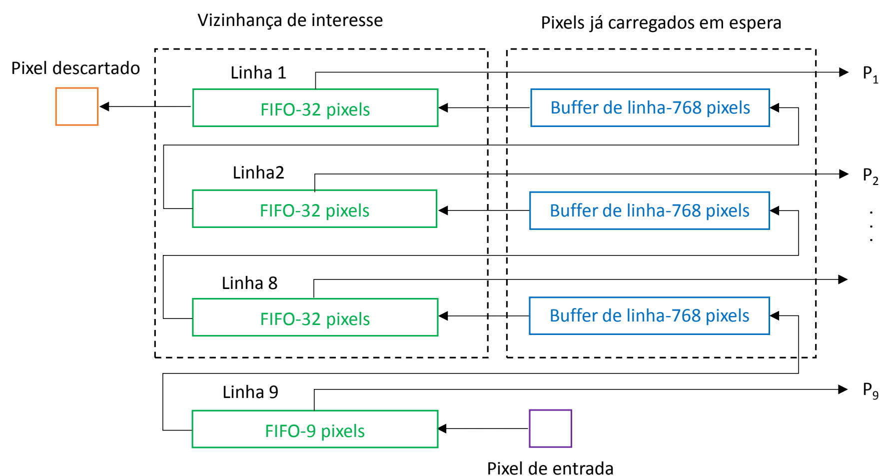


Figura 4.7. Arquitetura para a operação de janelamento para uma imagem de 800×525 pixels

O bloco verde ressaltado na figura 4.7 corresponde a uma FIFO (*First In, First Out*); ou seja, o primeiro pixel a entrar é o primeiro pixel a sair. A FIFO funciona como um banco de

registradores de deslocamento, permitindo assim o acesso aos pixels desejados da vizinhança (na Figura 4.6 é referenciada como *Fcarreg*). Após o pixel sair da *Fcarreg* o mesmo é carregado temporariamente no *buffer* de linha, em cada ciclo de relógio os pixels são deslocados na mesma direção, sendo o pixel mais antigo descartado, e o mais novo é carregado(bloco azul ressaltado, vide Figura 4.7) [19]. Os *buffers* de linha são implementados como arquiteturas do tipo *shift registers* síncronos utilizando *block-RAMs* do FPGA.

Geralmente, o processo de carregamento disponibiliza toda a vizinhança (9×9, ou seja, 81 pixels). Neste trabalho, a disponibilização foi dividida em um único pixel por cada linha, sendo os pixels enviados simultaneamente desde a nona posição dos registradores de deslocamento. Tendo em vista que o *pipeline* deve estar cheio para desenvolver a primeira convolução, este processo tem uma latência inicial (vide Equação 4.3), para uma imagem de $M \times N$ e uma máscara de $L \times P$ [19]. Neste trabalho, imagens de 800×525 estão sendo processadas utilizando uma máscara de 9×9. Portanto, a latência inicial do processo de carregar todos os dados para a primeira convolução é de 3205 ciclos de relógio.

$$Latência = (P - \alpha)M + L - \beta \tag{4.3}$$

no qual α *e* β são apresentadas a seguir:

$$\alpha = \frac{L+1}{2} \tag{4.4}$$

$$\beta = 1 - \alpha \tag{4.5}$$

O Quartus II disponibiliza a ferramenta *MegaWizard Plugin Manager*, que contém a operação desejada (*shift registers*), onde o valor mais próximo do tamanho da linha da imagem é de 768 pixels, porém o tamanho da *Fcarreg* é de 32 pixels (768 + 32=800). Os *buffers* de linha junto com os *shift registers* com a *Fcarreg* são equivalentes exatamente a uma linha da imagem.

4.3.2.2 Arquitetura de disponibilização da vizinhança

A implementação da arquitetura para disponibilizar a vizinhança é mostrada na Figura 4.8. Simultaneamente, cada pixel da nona posição de *Fcarreg* é carregado em uma FIFO de nove posições (*Fdisp*), sendo multiplexado para dois conjuntos de registradores ($BReg_1$ e $BReg_2$) de igual tamanho que *Fcarreg* (vide Figura 4.8 e Figura 4.9(b)).

Um fator importante é que a câmera envia os dados de acordo com a sua captura, onde os pixels formam um fluxo continuo (*streaming*) (vide Figura 4.9(a)). Porém, a arquitetura

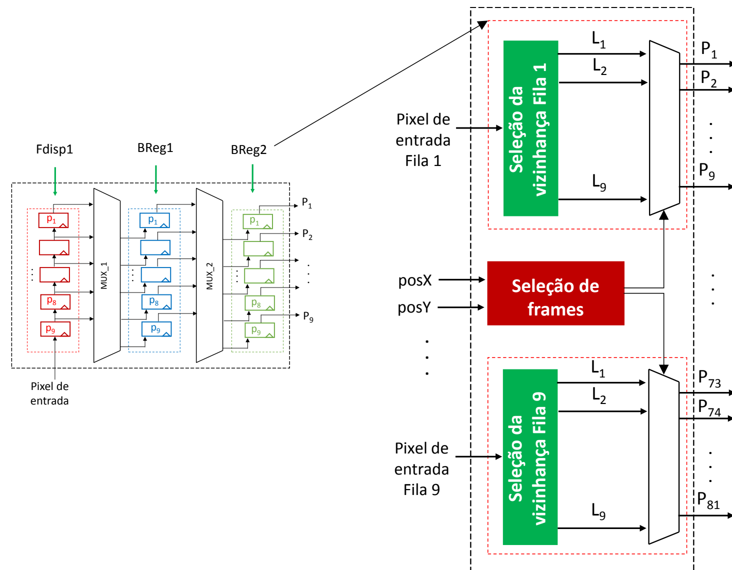


Figura 4.8. Arquitetura para a disponibilização dos pixels entre linhas da imagem e entre *frames* do vídeo

proposta separa os pixels entre as transições de linhas da imagem e entre os *frames*, no caso de processamento por vídeo. A disponibilização dos pixels é como segue:

(a) **Caso 1:** pixels da posição 1 até 9 da linha n da imagem são transferidos através do *MUX1* para o primeiro banco de registradores. Observe-se que na Figura 4.9(c) os pixels entre transições de linhas ficam misturados na mesma vizinhança, porém é necessário separar os pixels que correspondem à linha n para que sejam processados. Neste caso, só os pixels de cor verde são disponibilizados para a convolução.

(b) **Caso 2:** pixels da posição 792 até 800 da linha *n-1* da imagem são transferidos através do *MUX2* para o segundo banco de registradores. Da mesma forma que no caso anterior, é necessário separar só os pixels que vão ser processados (pixel de cor verde na Figura 4.9(d)).

(c) **Caso 3:** pixels da posição 10 até 791 da linha n da imagem são transferidos diretamente e disponibilizados para o processo de convolução.

Com tudo, os pixels formam um fluxo múltiplo em que temos os pixels referentes a cada linha da vizinhança sendo enviados simultaneamente. Estes fluxos terão como destino o processo de convolução.

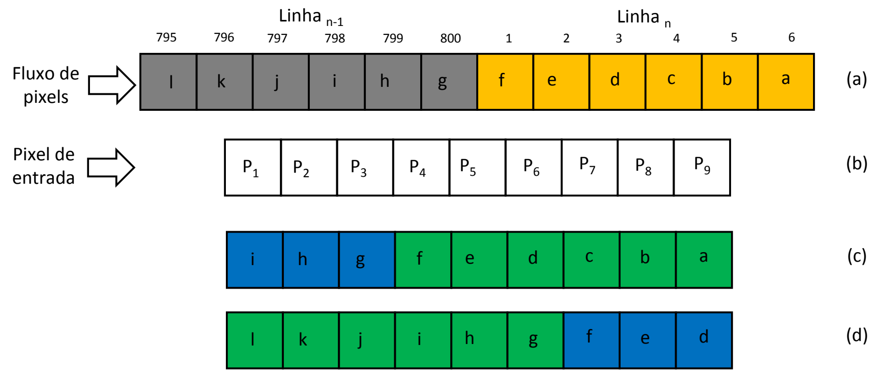


Figura 4.9. Exemplo dos casos para disponibilizar

4.3.2.3 Arquitetura de convolução

O processo de convolução 2D está baseada na definição (vide equação 3.2). O processamento ocorre simultaneamente varrendo toda a vizinhança carregada, fornecendo assim um pixel a cada ciclo de relógio (vide Figura 4.10). A disponibilização da vizinhança, que foi descrita na seção 4.3.2.2, é representada pela área (a) da Figura 4.10, onde as implementações dos multiplicadores e dos somadores foram omitidas (vide áreas (b) e (c) respectivamente).

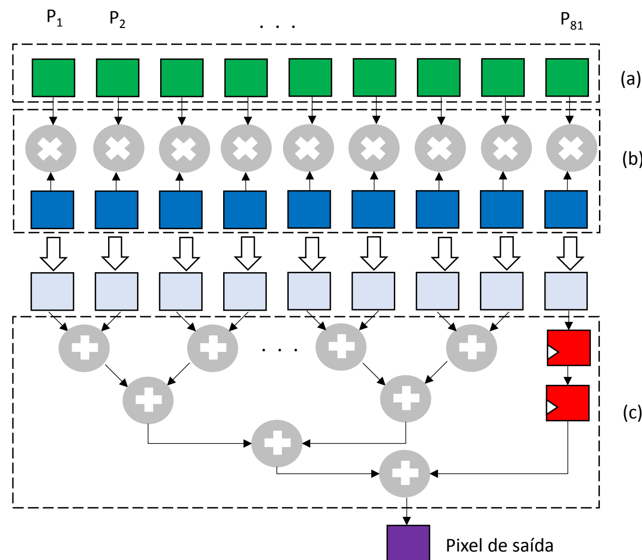


Figura 4.10. Arquitetura para a convolução

Usando o *MegaWizard Plugin Manager*, foi implementada uma memória ROM de uma saída, sincronizada a cada ciclo de relógio. Um *script* foi criado para gerar vários arquivos que armazenam valores da PSF para diferentes deslocamentos da câmera. Neste caso, é necessário passar um endereço para escolher os valores de PSF para cada deslocamento da câmera; sendo este

endereço a entrada para cada memória ROM (*Address*)(vide Figura 4.11). Assim sendo, quando um deslocamento é efetuado os valores de PSF são carregados e usados na convolução. Cada valor de PSF calculado em *Matlab* é transformado em número inteiro, uma vez que a arquitetura foi projetada para trabalhar com este tipo de número.

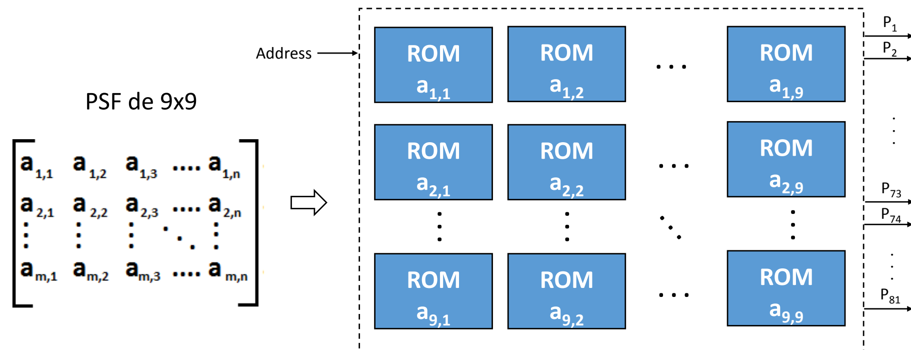


Figura 4.11. Vinculação dos valores da PSF à memória ROM

Cada elemento da vizinhança é multiplicado com o valor da máscara (PSF). Todas as multiplicações são feitas em paralelo entre a vizinhança e a máscara (vide figura 4.10(b)). Depois da multiplicação em paralelo, os resultados são somados usando uma árvore binária de soma (vide figura 4.10(c)). As somas são realizadas aos pares e em um *pipeline*, e o resultado desta somatória é o pixel processado.

4.3.3 Unidades de operações aritméticas

O RLA utiliza a operação matemática da divisão e a multiplicação, tais funções poderiam ser implementadas diretamente em VHDL. Como já mencionado, o Quartus II disponibiliza a ferramenta *MegaWizard Plug-In Manager* que contém diversas funções parametrizáveis e otimizadas para os FPGAs da Altera (vide figura 4.12). Os blocos de cor vermelho na Figura 4.4 (*pre-multiplicação*, *divisão* e *multiplicação*) foram implementados usando estas funções.

4.4 VERIFICAÇÃO FUNCIONAL E AVALIAÇÃO DE QUALIDADE

A verificação funcional é a fase que antecede a sintetização em um FPGA na qual demonstra-se que a funcionalidade do projeto está de acordo com sua especificação [48]. O ambiente de verificação da arquitetura ou DUT (*Design Under Test*) é conhecido como *testbench*. O DUT receberá estímulos, gerando saídas que serão comparadas com as saídas desejadas [48]. Este é um processo realizado a partir de simulação, usando a ferramenta *ModelSim* desenvolvida pela

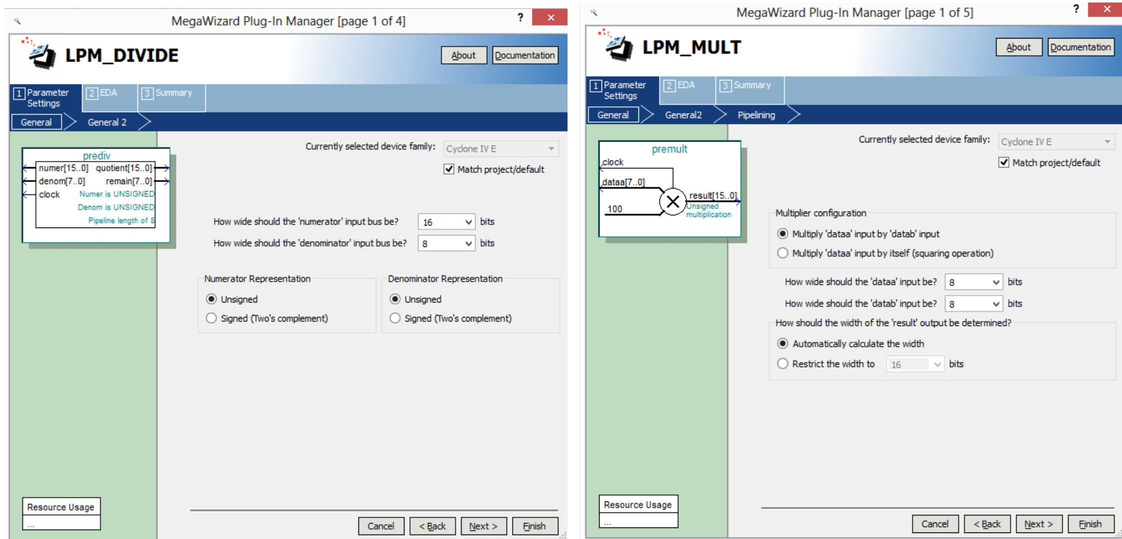


Figura 4.12. Tela do *MegaWizard Plug-In Manager* para configuração dos blocos da multiplicação e a divisão

Mentor Graphics [48]. Esta ferramenta simula o código a nível de RTL (*Register Transfer Level*) e *Gate Level*. Em nível RTL o circuito é analisado a nível de comportamento dos registradores, enquanto que em *Gate Level* o circuito é analisado a nível de *netlist* (com inclusão de atrasos das portas lógicas) [48]. A estrutura do *testbench* usada é apresentada na Figura 4.13.

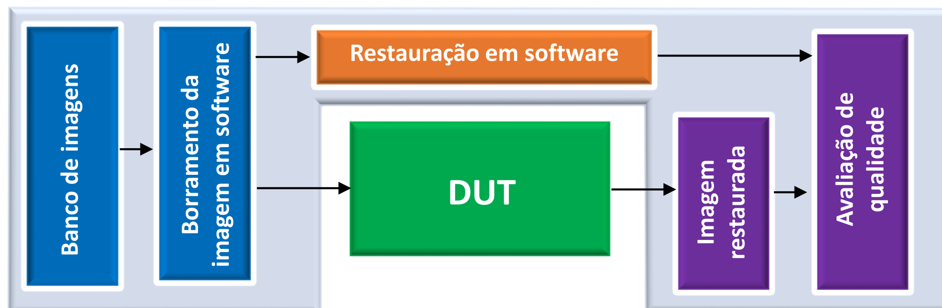


Figura 4.13. Diagrama de *testbench*

Nesta metodologia, o *testbench* possui cinco elementos:

(a) **sequência de imagens**: Na captura de imagens por uma câmera não é possível prever quais serão os valores de todos os pixels de uma imagem. A verificação da arquitetura deve ser testada com entradas conhecidas. Desta forma, as imagens a serem processadas são obtidas a partir de uma sequência de imagens e os valores da função de espalhamento de ponto (calculados usando a função *fspecial*).

(b) **borramento da imagem**:Um *script* foi criado para gerar um arquivo para a degradação das imagens em *Matlab*. As imagens que são arranjos bidimensionais foram convertidas para arranjos unidimensionais (vide figura 4.14). Isto é feito para simular a entrada da câmera. Estas imagens correspondem aos estímulos para a simulação da DUT e para a restauração em *software*.

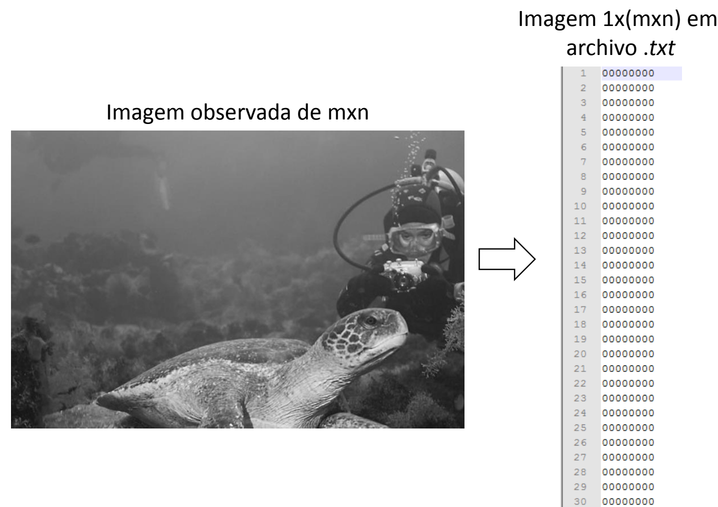


Figura 4.14. Transformação de uma imagem bidimensional em um arranjo unidimensional

(c) **restauração**: O DUT é a arquitetura que está sendo testada.

(d) **imagem restaurada**: Outro *script* foi gerado para salvar os resultados da simulação.

(e) **Avaliação de qualidade**: A avaliação da qualidade compara a saída da DUT com a referência (implementação em *software*). O algoritmo de avaliação de qualidade usado foi implementado em *Matlab* por [41], este método foi discutido na seção 3.5.1.

4.5 VALIDAÇÃO DA ARQUITETURA

Para os testes da arquitetura proposta, foi implementada una plataforma de captura, processamento e visualização de imagens digitais. A arquitetura geral do sistema de captura de imagens é mostrada na Figura 4.15. O sistema foi fornecido pelo fabricante do kit DE2-115, o qual inclui:(a) o controlador e a configuração do sensor CMOS, (b) a conversão do formato RAW (saída fornecida pela câmera) para RGB e (c) a sincronização e o controlador do display LCD (utilizando o controlador da memória SDRAM) [19]. O bloco de cor verde é a arquitetura integrada, ou seja, o algoritmo Richardson-Lucy.

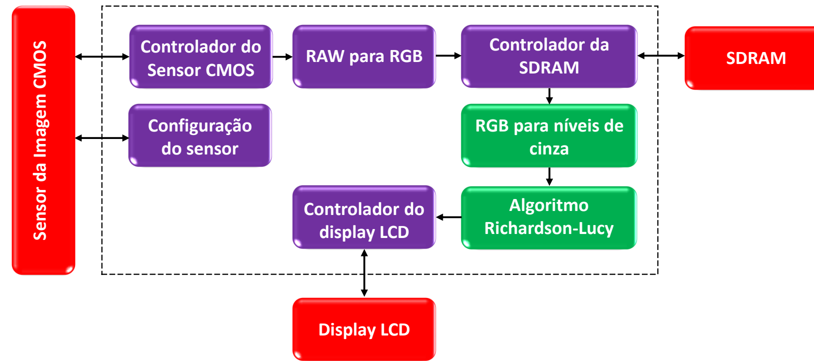


Figura 4.15. Arquitetura geral do sistema

Para captura das imagens foi utilizada uma câmera digital de 800×480 pixels de resolução, com conexão própria para o kit DE2-115, modelo D5M (*Terasic Inc*), Figura 4.16. Algumas características dessa câmera são:

Figura 4.16. Kit de desenvolvimento modelo VEEK-MT, composto principalmente pela câmera e o LCD

(a) Controle de tempo de exposição do sensor, configurável via registradores.

(b) Sensor CMOS de 5 Mega Pixel, com resolução configurável (máximo de 2592×1944 pixels ativos) e conversor AD de 12 bits por canal de cor.

(c) Barramento de dados de saída paralelo, com sinais separados R, G e B.

(d) Taxa de aquisição variável, de acordo com parâmetros de resolução e tempo de exposição do sensor.

A visualização das imagens, processadas ou não, foi feita utilizando um *display* LCD, modelo TFT-LCD, da *Terasic Inc* , disponível no kit de desenvolvimento VEEK-MT (vide Figura 4.16). Esse *display* possui uma interface para conexão no kit DE2-115, e as seguintes características:

(a) Interface de dados paralela RGB de 24 bits.

(b) Correção de brilho e contraste configuráveis.

(c) Resolução de $800 \times 480 \times RGB$ pixels.

4.6 CONCLUSÃO DO CAPÍTULO

As arquiteturas de convolução e do algoritmo Richardson-Lucy foram apresentadas neste capítulo. Aproveitando a capacidade da FPGA de fazer múltiplas operações em paralelo, foi implementada a operação de convolução (com preservação das bordas) que é a parte fundamental do RLA. Foi apresentada a modificação para obter a imagem de saída igual ao tamanho da imagem original, caraterística que não possuem outras implementações em *hardware.*

A característica do algoritmo influenciou na escolha da arquitetura implementada em *hardware*, apresentando algumas limitações como por exemplo as imagens que precisam ser armazenadas por cada iteração. Esta exigência requer o uso de memórias externas, no entanto, a maior das memórias disponíveis no kit de desenvolvimento (SDRAM) está sendo utilizada para realizar a sincronização entre a câmera e o *display.* Além disso, a capacidade de armazenamento da outra memória externa (SRAM, 512KBytes) não é suficiente para armazenar as imagens.

Para a solução deste problema, foi usada a técnica de *loop unrolling*, replicando a arquitetura em cada iteração. Isto gera uma grande quantidade de uso de recursos lógicos da FPGA mas permite restaurar imagens com um nível grande de desempenho. Finalmente, o método de verificação do algoritmo foi apresentado, usando o *testbench* e sua avaliação da qualidade implementada em *Matlab.*

5 TESTES DE IMPLEMENTAÇÃO E RESULTADOS

Neste capítulo são apresentados os resultados referentes à arquitetura proposta para a operação de convolução, assim como do algoritmo Richardson-Lucy. As arquiteturas são mostradas separadamente para, primeiramente, obter uma análise individual de cada uma e ao final, realizar uma comparação com as implementações em *software*. Para medir a qualidade das imagens são comparados os resultados obtidos com a ferramenta *Matlab* R2013b, e os resultados obtidos da simulação das arquiteturas implementadas em *VHDL*; calculando assim o índice de similaridade SR-SIM. Adicionalmente, são apresentados resultados da síntese lógica do algoritmo implementado, realizando uma estimativa do consumo de recursos da arquitetura.

As arquiteturas *hardware* propostas foram desenvolvidas na linguagem de descrição de *hardware* (VHDL) usando como plataforma de desenvolvimento o *Quartus II* 14.1 da *Altera*. O dispositivo FPGA usado foi o *Cyclone IV-E* EPC4C115F29C7 da Altera incluído no kit de desenvolvimento DE2-115 do fabricante Terasic.

5.1 ARQUITETURA DO ALGORITMO RICHARDSON-LUCY

Os resultados da síntese foram coletados usando a opção de *synthesis aggressive* a qual visa minimizar a área do circuito a ser implementado no dispositivo FPGA. Os resultados obtidos pela arquitetura são mostrados nas Tabelas 5.1 e 5.2. As tabelas exibem uma série de resultados referentes à compilação da arquitetura para várias iterações, usando dois tipos de FPGAs da Altera. Nestas tabelas pode-se observar os resultados para consumo de elementos lógicos (LEs) e/ou módulos adaptativos lógicos (*Adaptive Logics Modules*-ALMs), os multiplicadores embarcados (DSPs) e a frequência máxima de operação da arquitetura (Fmax). As FPGAs usadas foram a Cyclone IV-E EPC4C115F29C7 (FPGA de baixo custo) e a Stratix V 5SGXMB9R1H43C2 (FPGA de alto desempenho).

Como visto na seção 3.4.1, o algoritmo possui duas convoluções e, como consequência disto, pode-se observar na Tabela 5.2 que a arquitetura exige uma grande quantidade de consumo

Tabela 5.1. Recursos de *hardware* para a arquitetura RLA proposta na Cyclone IV-E Altera

Iterações	LEs[1]	DSP [2]	Fmáx (MHz)
1	37.695 (33%)	501 (94%)	71,15
2	96.981 (65%)	532 (100%)	69,12

[1] Elementos lógicos

[2] Multiplicadores embarcados

[%] Porcentagem de elementos usados no FPGA

Tabela 5.2. Recursos de *hardware* para a arquitetura RLA proposta na Stratix V da Altera

Iterações	ALMs[1]	DSP[2]	Fmáx (MHz)
1	11.869 (3%)	170 (48%)	60,00
2	23.759 (7%)	340 (97%)	61,21
3	37.256 (12%)	352 (100%)	56,66
4	61.391 (19%)	352 (100%)	67,33
5	85.663 (27%)	352 (100%)	59,39
6	112.114 (35%)	352 (100%)	63,20
7	138.622 (44%)	352 (100%)	65,53
8	164.160 (52%)	352 (100%)	63,26
9	177.986 (56%)	352 (100%)	61,53
10	202.932 (64%)	352 (100%)	61,32

[1] Módulos adaptativos lógicos

[2] Multiplicadores embarcados

[%] Porcentagem de elementos usados no FPGA

de recursos em *hardware* para cada iteração. Já a partir da terceira iteração, o uso de DSPs é de 100%; em função disto, a partir desta iteração é necessário usar elementos lógicos para implementar as operações de multiplicação, o que pode trazer uma perda de desempenho no quesito frequência de operação. A causa principal do aumento dos multiplicadores é o processo de convolução de tamanho de 24 bits, tamanho que foi escolhido para conservar a precisão da imagem restaurada (vide seção 4.3). Outro fator importante para ressaltar é que a frequência máxima de operação atinge o valor exigido pelo sistema de aquisição (50 MHz), frequência que é fornecida pela câmera.

5.2 SIMULAÇÃO COMPORTAMENTAL

A arquitetura foi simulada usando a ferramenta computacional ModelSim-Altera Start Edition. Para simular a sequência de vídeo da câmera (*streaming*, vide Figura 5.1a), todos os canais de cores (*Reed*, *Green* e *Blue*) das imagens usadas foram transformadas em arranjos unidimensionais consecutivos (vide Figura 5.1b e 5.1c). O tamanho da imagem de entrada foi de 800×525 pixels e os valores da PSF calculados para cada valor de deslocamento foram salvos em memórias internas da FPGA. As imagens de teste foram borradas, usando deslocamentos entre três e nove pixels, e valores de ângulos de 0°, 30°, 60° e 90°. Onde o deslocamento mais significativo, ou seja, o deslocamento que permite degradar com maior nível visual a imagem e testar a arquitetura proposta, foi de $X = 9$ pixels e $\theta = 60^o$. Finalmente, as imagens obtidas pela operações de redução de cores, filtragem por convolução (para borramento) e restauração das mesmas (obtidas no *Matlab*) foram comparadas com as respectivas imagens obtidas usando a arquitetura (vide Figura 5.1d). Estas comparações foram realizadas usando a métrica SR-SIM (vide Capítulo 3, onde é feita a justificativa do uso desta métrica).

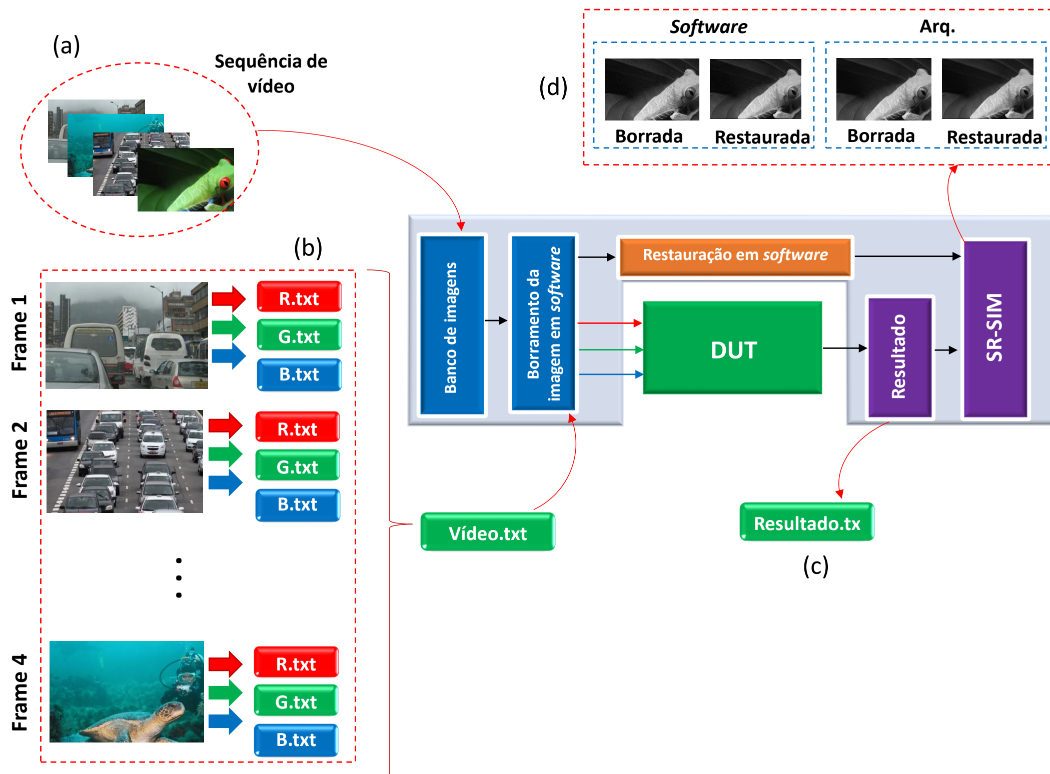


Figura 5.1. Metodologia da verificação comportamental do algoritmo, usando ModelSim

Como parte da metodologia usada para verificar os resultados usando a arquitetura proposta foram usados os seguintes passos:

- Sobre uma imagem definida (por exemplo, vide Figura 5.2) é feita a operação desejada (redução de cores, filtragem por convolução e/ou restauração).

- Sobre uma imagem com a operação desejada foi feito um *zoom* a fim de verificar mais detalhes da mesma (vide figura 5.2b).

(a) SR-SIM=1 **(b)** SR-SIM=0,9999 **(c)** SR-SIM=0,9999

Figura 5.2. Resultados da verificação comportamental e avaliação da qualidade para trânsito 1, usando como referência as imagens obtidas de *Matlab*. (a) Imagem original, (b) Zoom da imagem em escala de cinza e (c) Zoom da imagem distorcida por borramento

Como foi explicado na Capítulo 4, as imagens são aplicadas à arquitetura proposta (como dados de entrada) e o ModelSim gera os dados das novas imagens (que são armazenados em arquivos), produto das operações realizadas sobre as imagens originais. Estes dados das imagens de saída são visualizados no *Matlab*, a fim de fazer uma comparação das mesmas (avaliação objetiva e visual). Neste caso, se as imagens processadas, tanto do *Matlab* como da arquitetura fossem igual, a métrica SR-SIM seria igual a 1.

Deve ser observado, que as imagens produzidas pelo ModelSim correspondem a um modelo de *simulação comportamental*, onde não são tidos em conta os efeitos de retardos nos dispositivos eletrônicos (neste caso, dos FPGAs), vinculados aos caminhos críticos do circuito e dos efeitos de roteamento global e local efetuado pelo Quartus II.

5.2.1 Redução de cores

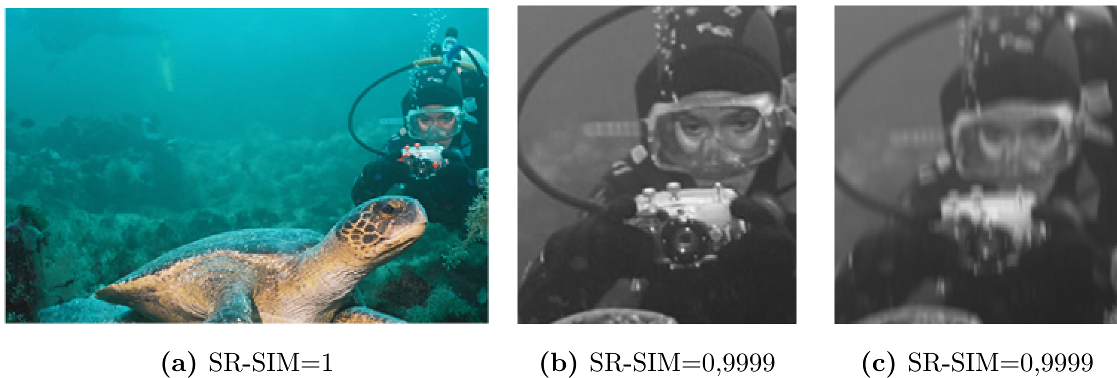
Na seção 4.3.1 foi descrita a arquitetura desenvolvida para realizar a transformação de cores da imagem, antes da mesma ser restaurada. Este bloco é de baixo consumo de recursos, tendo em conta que, para isto foi desenvolvida a operação apresentada na seção 4.3.1.

As Figuras 5.2b, 5.3b, 5.4b e 5.5b mostram os resultados das verificações comportamentais, realizadas no ModelSim. Neste caso, a operação de redução de cores é simples, constando só de três multiplicações e duas somas. Na mesma figura são mostrados os resultados da métrica usada

neste trabalho, a fim de comparar as imagens com redução de cores obtidas no *Matlab* com as correspondentes imagens obtidas através da arquitetura. Este resultado mostra que as operações de redução de cores tanto do *Matlab* como da arquitetura, são bem similares. A pequena diferença pode ser devida a quesitos de precisão nas respectivas representações numéricas no *Matlab* e na arquitetura proposta.

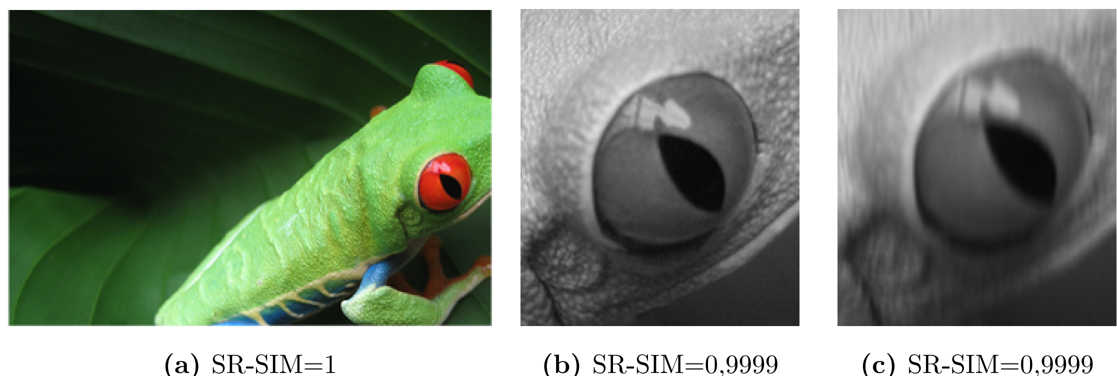
5.2.2 Filtragem por convolução

Os resultados da simulação comportamental, com respeito às operações de filtragem por convolução (para borramento), são mostrados nas Figuras 5.2c, 5.3c, 5.4c e 5.5c. Pode-se observar que as imagens filtradas tornam-se borradas, simulando o borramento que pode ser conseguido com o deslocamento da câmera durante o processo de captura.



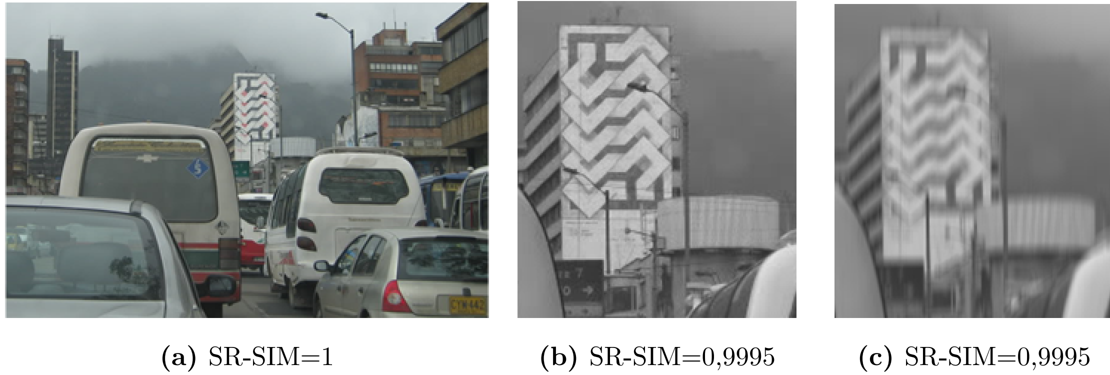
(a) SR-SIM=1 **(b)** SR-SIM=0,9999 **(c)** SR-SIM=0,9999

Figura 5.3. Resultados da verificação comportamental e avaliação da qualidade para tartaruga, usando como referência as imagens obtidas de *Matlab*. (a) Imagem original, (b) Zoom da imagem em escala de cinza e (c) Zoom da imagem distorcida por borramento



(a) SR-SIM=1 **(b)** SR-SIM=0,9999 **(c)** SR-SIM=0,9999

Figura 5.4. Resultados da verificação comportamental e avaliação da qualidade para sapo, usando como referência as imagens obtidas de *Matlab*. (a) Imagem original, (b) Zoom da imagem em escala de cinza e (c) Zoom da imagem distorcida por borramento



(a) SR-SIM=1 **(b)** SR-SIM=0,9995 **(c)** SR-SIM=0,9995

Figura 5.5. Resultados da verificação comportamental e avaliação da qualidade para trânsito 2, usando como referência as imagens obtidas de *Matlab*. (a) Imagem original, (b) Zoom da imagem em escala de cinza e (c) Zoom da imagem distorcida por borramento

A implementação do RLA depende fortemente da convolução, assim, uma boa qualidade da imagem obtida por convolução pode dar uma boa uma imagem restaurada. Para avaliar a qualidade das imagens filtradas da arquitetura utilizaram-se como imagens de referência as imagens filtradas em *Matlab*. Estas imagens apresentam uma boa avaliação se comparadas com as imagens de referência (vide resultado do SR-SIM das Figuras 5.2, 5.3, 5.4 e 5.5).

Com respeito ao quesito de desempenho da arquitetura (vazão -*throughput*), segundo a equação 4.3, os resultados do processamento (latência e tempo inicial) da arquitetura são apresentados na Tabela 5.3. Após o tempo de latência inicial, a arquitetura de convolução fornece um pixel por cada ciclo de relógio.

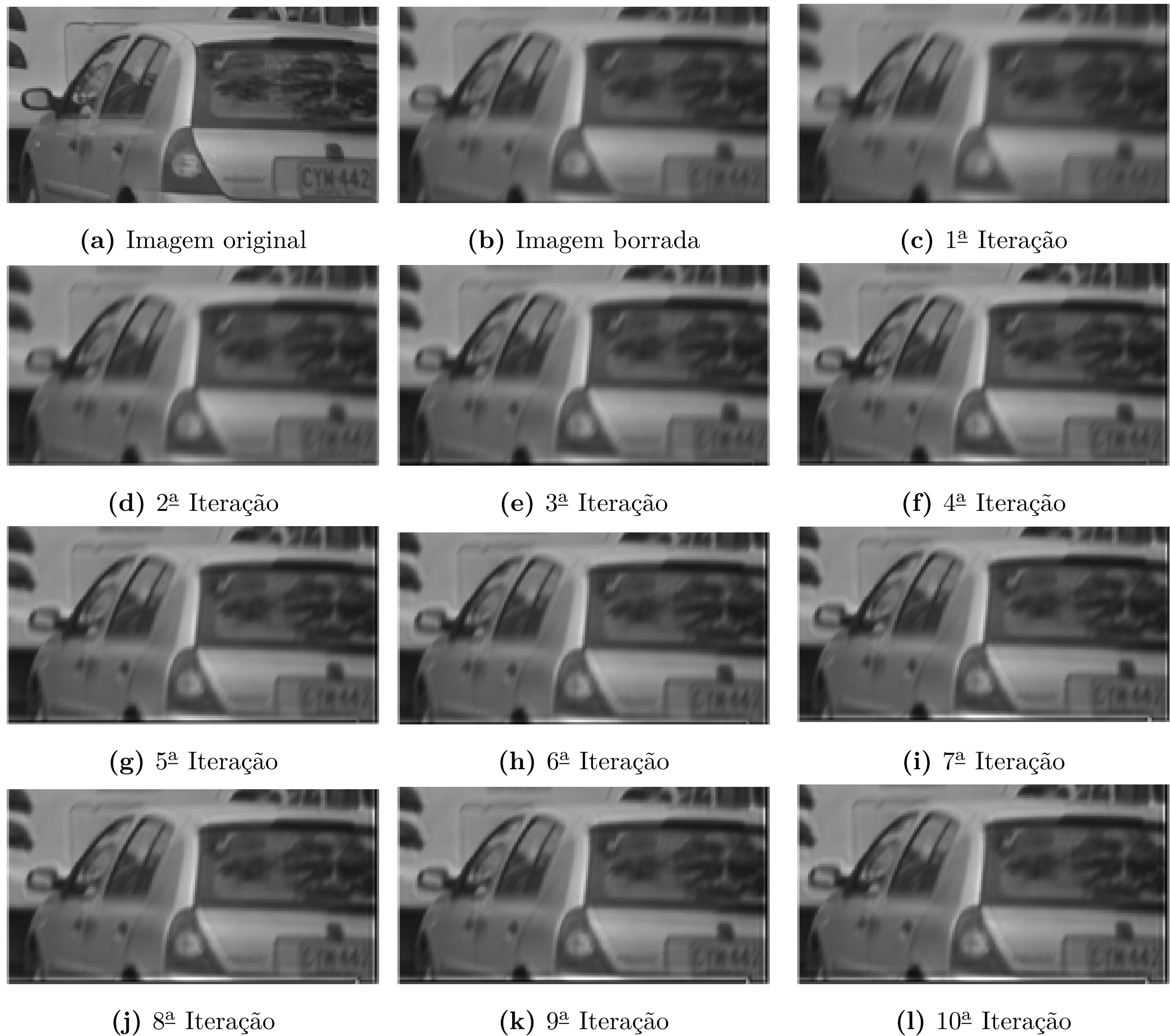
Tabela 5.3. Resultados da simulação para a arquitetura de convolução

Tamanho da máscara	**Latência (ciclos)**	**Frequência de operação (MHz)**	**Tempo de latência (μs)**
9×9	3205	50	64,1

5.2.3 Algoritmo Richardson-Lucy

Para simular a degradação das imagens devido ao deslocamento da câmera, as imagens degradadas foram geradas em *Matlab* utilizando a função *imfilter* e os valores da função de PSF com a função *fspecial*. A função *imfilter* é a encarregada de fazer a operação de convolução entre a imagem e a PSF. Esta função disponibiliza que a imagem resultante seja do mesmo tamanho da imagem que esta sendo convoluída, característica que esta sendo implementada neste trabalho.

Na Figura 5.6, pode-se observar o resultado da verificação comportamental da arquitetura.



(a) Imagem original **(b)** Imagem borrada **(c)** 1ª Iteração

(d) 2ª Iteração **(e)** 3ª Iteração **(f)** 4ª Iteração

(g) 5ª Iteração **(h)** 6ª Iteração **(i)** 7ª Iteração

(j) 8ª Iteração **(k)** 9ª Iteração **(l)** 10ª Iteração

Figura 5.6. Zoom da sequência de restauração para trânsito 2

A Figura 5.6a e 5.6b são a imagem original (sem borramento) e com borramento (o borramento foi obtido usando o *Matlab*), respectivamente. A partir das Figuras 5.6c até a 5.6l são mostradas as sequências de iterações para a restauração da imagem (usando a arquitetura proposta).

Pode ser observado que a imagem obtida melhora a qualidade conforme avança a cada iteração. No entanto, dez iterações não são suficientes para obter uma qualidade objetiva e visual ótima.

Para cada iteração n, a imagem restaurada em *Matlab* foi comparada com a imagem restaurada pelo *hardware* (vide Figura 5.7). Como visto na Figura 5.7, a diferença entre as imagens restauradas no *Matlab* e na arquitetura aumenta conforme o valor da interação. Isto é devido a que arquitetura não possui o nível de precisão nas operações matemáticas (multiplicação e divisão) de *Matlab*, originando um erro que é propagado para os blocos subsequentes, assim como para as iterações subsequentes. O tamanho de deslocamento e ângulo da imagem usado

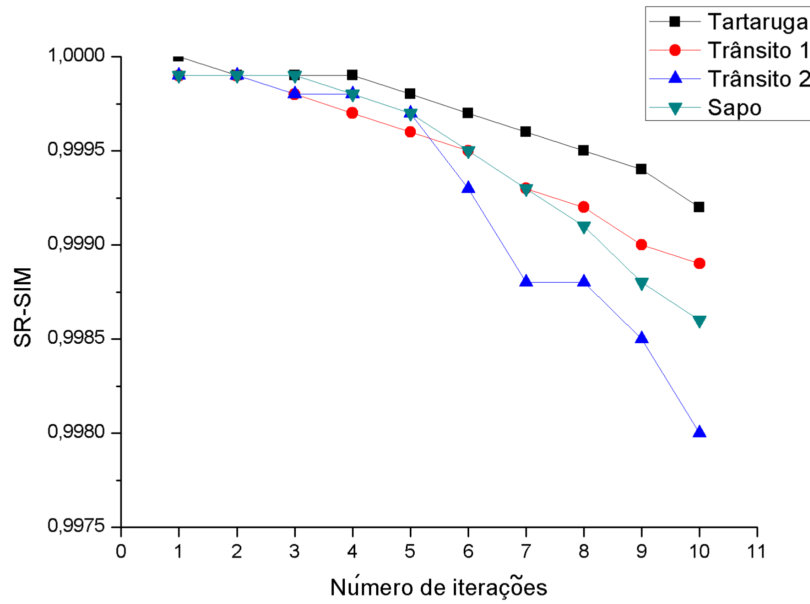


Figura 5.7. SR-SIM para várias imagens restauradas em *hardware* comparadas com a imagem obtida em *Matlab*

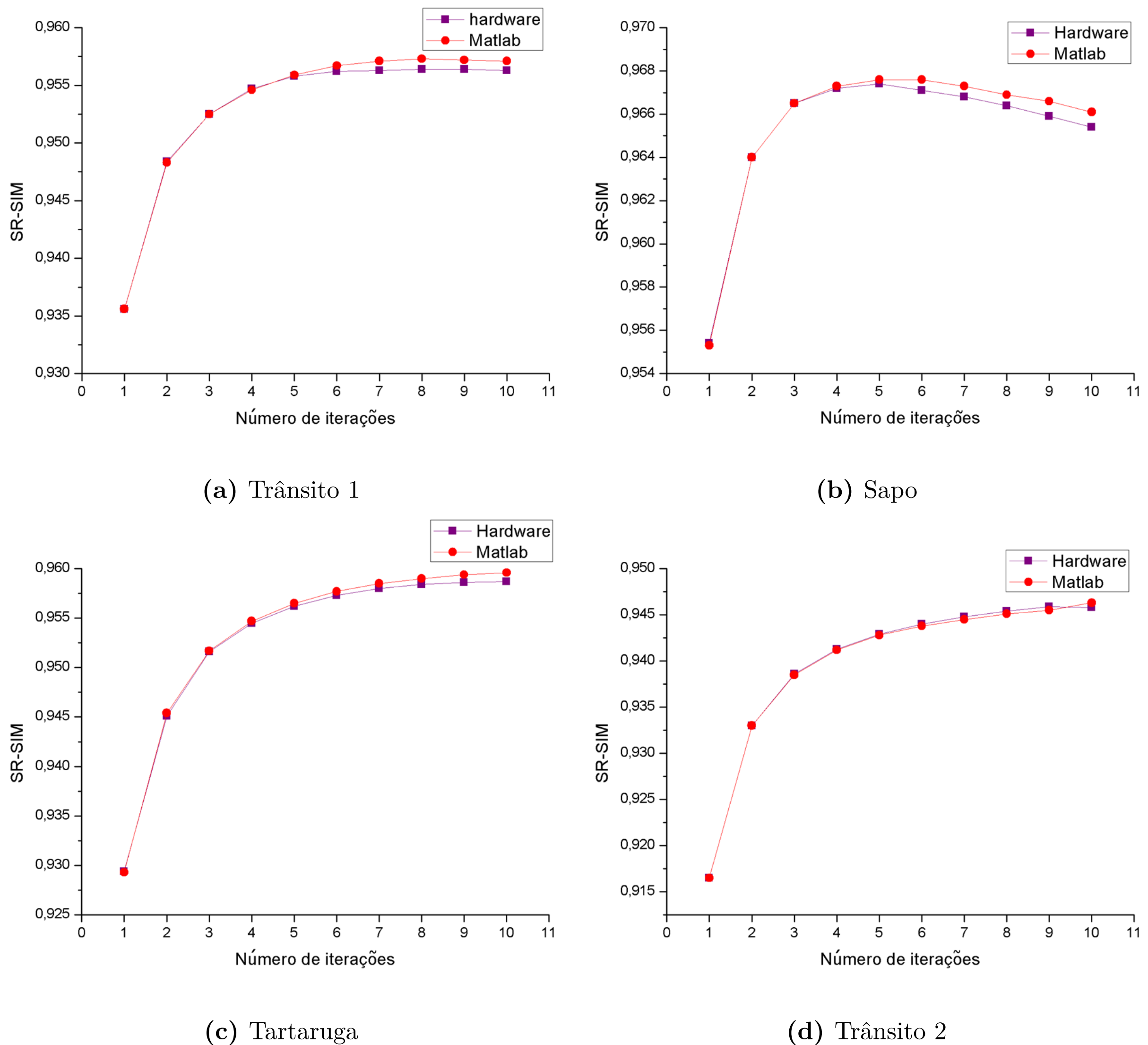
neste trabalho, permite obter uma boa avaliação da qualidade das imagens pois as mesmas não apresentam um borramento considerável.

Tabela 5.4. SR-SIM para as imagens restauradas em *Matlab* e *hardware* comparadas com a imagem sem degradação

Iter.	**Trânsito 1**		**Trânsito 2**		**Sapo**		**Tartaruga**	
	Hard.	***Matlab***	**Hard.**	***Matlab***	**Hard.**	***Matlab***	**Hard.**	***Matlab***
1	0,9356	0,9356	0,9165	0,9165	0,9554	0,9553	0,9294	0,9293
2	0,9484	0,9483	0,9330	0,9330	0,9640	0,9640	0,9451	0,9454
3	0,9525	0,9525	0,9386	0,9385	0,9665	0,9665	0,9516	0,9517
4	0,9547	0,9546	0,9413	0,9412	0,9672	0,9673	0,9545	0,9547
5	0,9558	0,9559	0,9429	0,9428	0,9674	0,9676	0,9562	0,9565
6	0,9562	0,9567	0,9440	0,9438	0,9671	0,9676	0,9573	0,9577
7	0,9563	0,9571	0,9448	0,9445	0,9668	0,9673	0,9580	0,9585
8	0,9564	0,9573	0,9454	0,9451	0,9664	0,9669	0,9584	0,9590
9	0,9564	0,9572	0,9459	0,9455	0,9659	0,9666	0,9586	0,9594
10	0,9563	0,9571	0,9458	0,9463	0,9654	0,9661	0,9587	0,9596

Pode-se observar na Tabela 5.4 os resultados da avaliação da qualidade das imagens restauradas (*Matlab* e *hardware*), usando como referencia as imagens sem degradação (esta figura é

referente às imagens mostradas nas Figuras 5.2, 5.3, 5.4 e 5.5). É importante ressaltar que conforme é incrementado o número de iterações a qualidade objetiva aumenta e converge lentamente (vide Tabela 5.4 e Figura 5.8). Isto é pelo fato das caraterísticas que possui o algoritmo, onde os efeitos nas bordas da imagem (e dos objetos que a mesma possui) criam distorções nas imagens restauradas (vide referência [44]). Da mesma maneira, a qualidade da imagem restaurada é sensível também ao erro das operações matemáticas.



(a) Trânsito 1

(b) Sapo

(c) Tartaruga

(d) Trânsito 2

Figura 5.8. SR-SIM para as imagens restauradas em *Matlab* e *hardware* comparadas com a imagem sem borramento

Como já mencionado, o sistema foi projetado para ter uma convergência lenta (vide Figura 5.8), pois o algoritmo não possui a aceleração das iterações (vide seção 3.4.1). Este fator é limitante devido ao fato de que a implementação para cada iteração consume uma grande quantidade de recursos.

Para exemplificar visualmente o funcionamento do algoritmo, foram coletados os resultados

obtidos através da simulação comportamental (usando o ModelSim), sendo assim apresentadas as imagens restauradas, como mostrado na Figura 5.9. Como era de se esperar, dez iterações não foram suficientes para obter uma excelente qualidade visual e objetiva. Neste caso, o algoritmo deverá ser modificado e otimizado usando técnicas de aceleração discutidas no Capítulo 3.

Os resultados do processo de restauração para um *frame* (latência e tempo de processamento) da arquitetura são apresentados na Tabela 5.5. Porém, para n iterações o tempo de processamento é de $n \times 8,4\ ms$ mais o tempo de latência inicial.

Tabela 5.5. Resultados da simulação para a arquitetura de restauração

Tamanho da imagem	**Latência (ciclos)**	**Frequência de operação (MHz)**	**Tempo de latência (μs)**	**Tempo de restauração (ms)**
800×525	6488	50	129,76	8,4

As câmeras comuns (filmadoras e fotográficas) adquirem imagens estaticamente ou em vídeos, com uma taxa de 30 fps (33,33 ms). Porém, a arquitetura proposta consegue executar quatro iterações entre *frames* de um vídeo. Na captura das imagens (estáticas ou vídeos), vários *frames* consecutivos podem apresentar um borramento gradual (borramento lento) das imagens. Porém, o tempo do conjunto de *frames* consecutivos pode abrir a possibilidade de executar mais iterações para restaurar um determinado *frame*. Contudo, o deslocamento da câmera deve ser medido para a estimação da função de espalhamento de ponto (PSF). Assim, este processo (usando técnicas de estimação de trajetória ou sensores) leva consigo um tempo determinado, e finalmente, este tempo determina o frame que deve ser processado e quantas iterações podem ser executadas. Outra possibilidade pode ser incrementar a frequência de operação da arquitetura a fim de aumentar o número de iterações executadas entre *frames*.

Para comparar a arquitetura no quesito tempo de execução, o RLA foi implementado em *software* usando a biblioteca OpenCV 2.4.9.0 (*Open Source Computer Vision*) para processamento de imagens. Esta biblioteca possui funções otimizadas para processamento de imagens. A plataforma usada foi um PC com processador:Intel Core i3 CPU550 a 3,20GHz, 7,6GB RAM e um sistema operacional de 64 bit (*Kernel Linux 3.11.0-26-generic*). Uma aplicação foi executada durante cada teste para evitar uma sobrecarga da CPU. Além da implementação em C++ usando a biblioteca de OpenCV, o algoritmo foi codificado em linguagem C e os resultados foram comparados com os tempos de execução da arquitetura, usando tamanhos de máscara de 9×9 .

A Figura 5.10 mostra o tempo de execução usado pelo FPGA para diferentes iterações, operando baixo um relógio com frequência de 50 MHz e comparado com a implementação em *software*. Na Figura 5.10, observa-se que a arquitetura mantêm sempre um desempenho abaixo do *software* (C++), no quesito tempo de execução. No entanto, levando em consideração o tempo

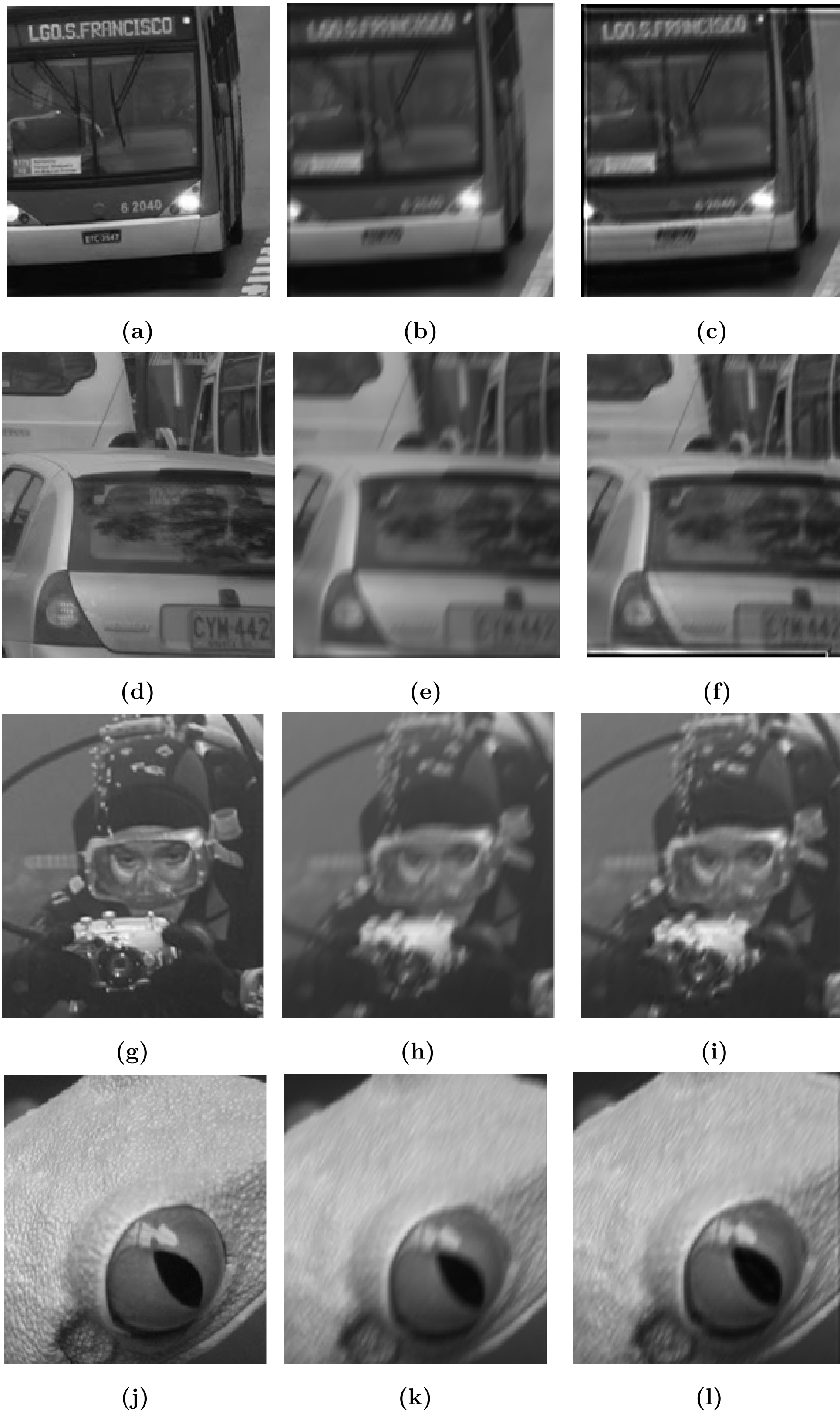


Figura 5.9. Restauração das imagens em *hardware*. (a), (d), (g) e (j) Imagens originais. (b), (e), (h) e (k) Imagens borradas. (c), (f), (i) e (l) Imagens restauradas

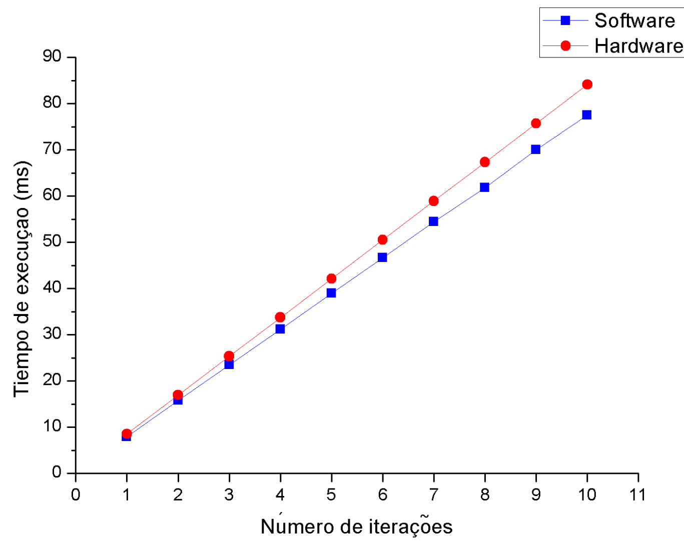


Figura 5.10. Desempenho do algoritmo rodando em *software* (usando um código C++, e OpenCV em um PC a 3.2 GHz) e rodando na arquitetura proposta

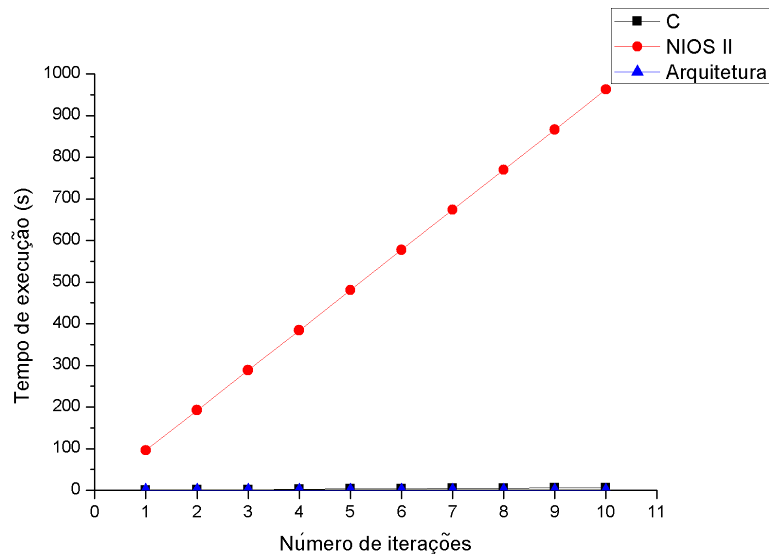


Figura 5.11. Desempenho do algoritmo rodando em *software* (usando um código C em um PC a 3.2 GHz e em NIOS II a 100 MHz) e rodando na arquitetura proposta

de execução de cada implementação (*software* e *hardware*), a arquitetura em *software* (C++) é aproximadamente 0,91 vezes mais rápida que a implementação em *hardware*. Entretanto, a arquitetura em *hardware* é 81 vezes mais rápida que a implementação em C (vide Equação 5.1). Esta diferença pode trazer uma vantagem do *hardware*, no quesito consumo de potência, para a

implementação em um sistema embarcado com restrições de portabilidade.

$$Ganho_{velocidade} = \frac{T_s}{T_h}, \tag{5.1}$$

no qual T_s de execução em *software* e T_h é o tempo de execução do *hardware*. O código C do algoritmo de restauração (RLA) foi embarcado no processador NIOS II, e a frequência de operação do sistema foi de 100 MHz. O tempo de execução para cada iteração é apresentado na Figura 5.11. Pode-se observar que, comparado com o NIOS II (rodando a frequência de 100 MHz), o desempenho no quesito tempo de execução do *hardware* é de aproximadamente 12.425 vezes mais rápido. A diferença em desempenho é devido à grande quantidade de GOPs que o NIOS deve executar para as operações de convolução (escrita/leitura, multiplicações e adições) rodando a uma frequência do mesmo ordem do *hardware* (50 MHz).

Na Tabela 5.6 é apresentada a comparação entre os tempos de execução do algoritmo para diferentes tamanhos de máscaras e dez iterações, implementado em *software* (C++) e *hardware*. Como mencionado no Capítulo 4, é de esperar que para máscaras menores o tempo de execução em *software* comparado com o *hardware* seja menor pois deve-se considerar que o tamanho da máscara da arquitetura é constante (9×9) e a frequência de operação do *software* é de 3,20GHz.

Tabela 5.6. Desempenho do algoritmo rodando em *software* (C++) e em *hardware* para diferentes tamanhos de máscara

Tamanho da máscara	**Tempo de execução (ms)**	
	Hardware	***Software***
3x3	84,1298	59,8695
5x5	84,1298	71,0301
7x7	84,1298	76,8726
9x9	84,1298	77,0401

5.3 VALIDAÇÃO DA ARQUITETURA

A arquitetura foi compilada para diferentes tipos de FPGAs da Altera (vide Tabela 5.2). No entanto, a Cyclone IV-E é o kit com FPGA Cyclone IV disponível no LEIA (Laboratório de Sistemas Embarcados e Aplicações de Circuitos Integrados). Esta placa possui um sistema embarcado para a aquisição de imagens com vídeo (vide Figura 4.15). Porém, como visto na Tabela 5.2 o número de iterações alcançado para a FPGA Cyclone IV-E (FPGA de médio porte) foi de duas iterações, sendo que para a Stratix (FPGA de alto desempenho) foi de dez iterações. Contudo, o sistema de captura e visualização na Cyclone IV-E foi feito usando somente duas

iterações e o resultado obtido é mostrado na Figura 5.12.

A Figura 5.12a mostra a implementação usando o kit de desenvolvimento VEEK-MT, nesta figura mostra-se a imagem borrada; as Figuras 5.12b e 5.12c apresentam a imagem em nível de cinza e a imagem restaurada após duas iterações respetivamente. Pode-se observar que na Figura 5.12c a imagem restaurada apresenta uma leve restauração pois é de esperar que a qualidade para esta iteração não seja o suficientemente boa. Adicionalmente, foi implementada a arquitetura usando uma interface VGA e uma câmera analógica, visando o uso dos diferentes sistemas de captura e visualização de imagens (vide Figura 5.12d).

5.4 CONCLUSÕES DO CAPÍTULO

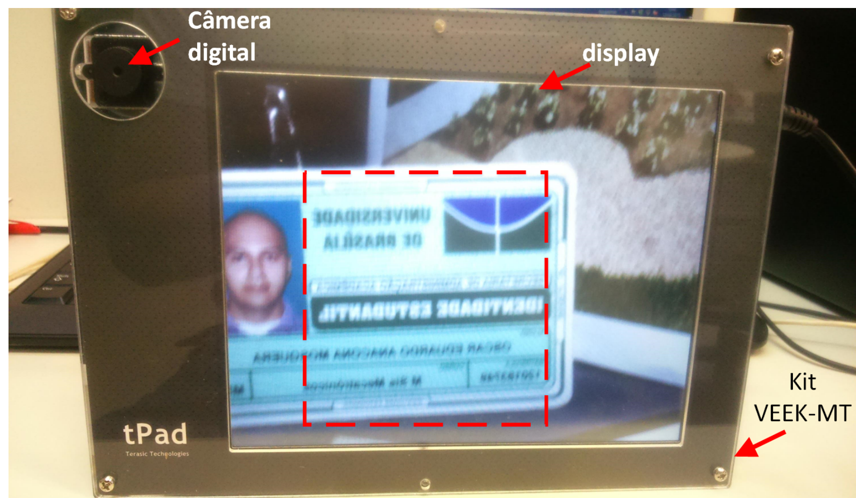
Neste capítulo foram vistos os diferentes resultados para as arquiteturas propostas de implementação do RLA e da convolução. Primeiramente, mostrou-se os resultados de recurso de *hardware* para várias iterações. Viu-se para este resultado os recursos consumidos em termos de elementos lógicos, DSPs e frequência máxima de operação das arquiteturas.

Foi mostrado também o método de verificação e os resultados da verificação comportamental de tais arquiteturas. Para a implementação da convolução, a vantagem está no tamanho da imagem obtida, sendo igual ao tamanho da imagem observada. Além disso, foi disponibilizada uma sequência de imagens simulando a entrada de aquisição de vídeo em tempo real.

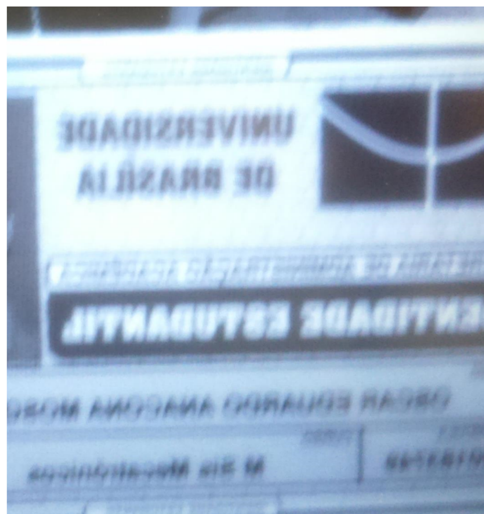
Sendo que a arquitetura da convolução é a parte fundamental do RLA, verificou-se também que a imagem obtida apresenta uma boa avaliação da qualidade, usando a métrica SR-SIM. Os resultados mostram que a filtragem por convolução atinge as características do algoritmo de restauração.

O tamanho de deslocamento da imagem usado neste trabalho, permite obter uma boa nota de qualidade. Porém, o desafio está na implementação da arquitetura para tamanhos de máscaras maiores, o que traz aumento de recursos lógicos, e na otimização do método de convolução para poupar a maior quantidade de recursos sem perder qualidade no resultado.

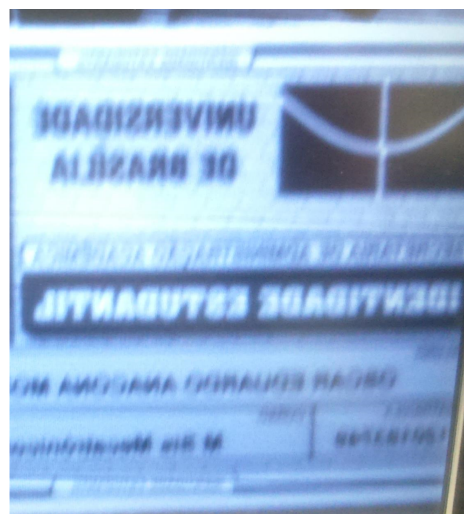
A arquitetura proposta neste trabalho alcança uma velocidade de processamento de até 1,09 vezes mais lento com relação ao processador utilizado na comparação (usando OpenCV), e de 12.425 vezes mais rápido que o processador NIOS II rodando em uma frequência de 100 MHz.



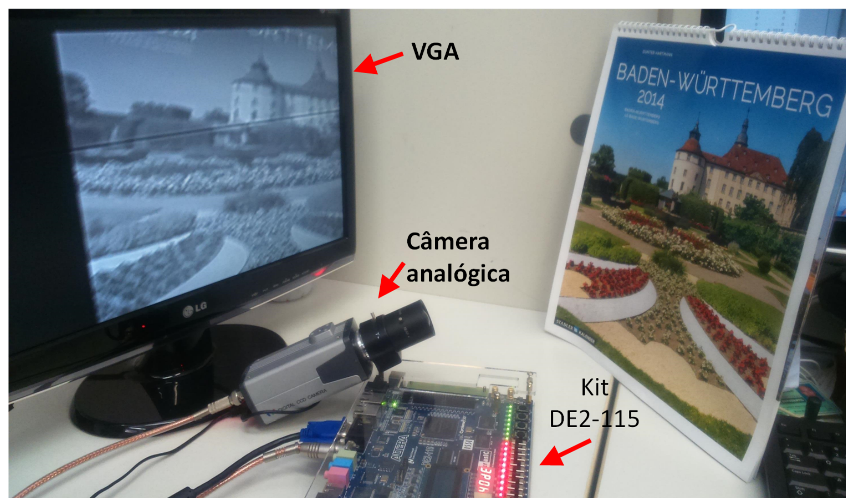
(a) Imagem borrada em escala de RGB



(b) Imagem borrada em escala de cinza



(c) Imagem desborrada



(d) Implementação usando uma interface VGA e câmera analógica

Figura 5.12. Validação da arquitetura do RLA.

6 Conclusões e Trabalhos Futuros

6.1 Comentários Finais e Conclusões

Neste trabalho foi apresentada uma arquitetura de *hardware* (baseada em FPGA) para resolver o problema de borramento de imagens. Primeiramente, foi desenvolvida uma arquitetura para a filtragem por convolução, sendo que esta arquitetura é a parte fundamental da arquitetura para a deconvolução de imagens. A implementação consegue realizar filtragem usando tamanhos de máscaras desde 3×3 até 9×9.

Para provar o funcionamento apropriado da arquitetura, um método de verificação foi realizado. Verificou-se que os resultados da arquitetura foram semelhantes com a implementação em *software*. Foi utilizada uma métrica de avaliação de qualidade para imagens (SR-SIM) para avaliar o desempenho das duas abordagens. Os resultados apontaram para um desempenho satisfatório da unidade de filtragem por convolução.

Na sequência, foi apresentada a arquitetura para restauração de imagens baseada no algoritmo de Richardson-Lucy. Esta arquitetura explora eficazmente sua característica de paralelismo assim como a implementação de um *pipeline* de n-estágios. Houve uma preocupação com o consumo de recursos lógicos da arquitetura para uma iteração do algoritmo, pois esta arquitetura é replicada em cada estagio do *pipeline*. O mesmo método de verificação e avaliação foi usado para avaliar o funcionamento desta arquitetura, mostrando que o método é uma boa alternativa para a restauração de imagens.

A convergência do algoritmo implementado neste trabalho é lenta e necessita ser otimizada. Porém, a implementação das arquiteturas para a aceleração da convergência é mais complexa, exigindo um algoritmo mais sofisticado e com maior consumo de recursos.

O sistema foi projetado para trabalhar como uma arquitetura de tempo real em forma de *pipeline*, fornecendo um pixel por ciclo de relógio (50 MHz) depois de um período de latência, sendo capaz de processar imagens de 800×525 com uma taxa de 120 fps (um pixel processado cada 20 ns). A abordagem levou a uma velocidade de processamento de 1,09 vezes mais lento, quando comparada com a implementação da arquitetura em *software* em um processador Intel

Core i3 CPU550 a 3,20GHz, 7,6GB RAM e um sistema operacional de 64 bit (*Kernel Linux 3.11.0-26-generic*). Como a arquitetura *hardware* trabalha com uma tecnologia de pequena área de silício, concluímos que a arquitetura proposta é apta ou apropriada para aplicações de tempo real com restrições de portabilidade (tamanho e peso).

6.2 Propostas de Trabalhos Futuros

Neste trabalho não foram consideradas algumas situações para a implementação da arquitetura em *hardware*. O sistema foi implementado para restaurar imagens com convergência lenta e grande consumo de recursos de hardware.

Assim, como trabalhos futuros sugere-se:

- Considerar a implementação de uma unidade de filtragem visando a poupar a maior quantidade de recursos.
- Desenvolver uma arquitetura para a aceleração do algoritmo usando um critério de parada.
- Implementar um algoritmo para o cálculo *online* dos parâmetros da função de espalhamento de ponto.
- Analisar o consumo de potência da arquitetura proposta. Esta análise é importante para validar a eficácia do algoritmo proposto para a restauração de imagens.
- Pode-se implementar o algoritmo para máscaras maiores às usadas neste trabalho (por exemplo 11×11, 13×13, etc). Esta implementação permitiria restaurar imagens que apresentem maior nível de borramento.
- Otimizar a arquitetura para aplicações de tempo real que usam taxas de mais de 150 fps, onde a presença dos gargalos (baixo desempenho no quesito de processamento dos *frames*) pode levar ao colapso do processamento.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] O. Sims, *Efficient implementation of video processing algorithms on FPGA*. Engd thesis, University of Glasgow, June 15, 2007.

[2] M. Liu, Z. Lu, W. Kühn, and A. Jantsch, "Fpga-based particle recognition in the hades experiment," *Design Test of Computers, IEEE*, vol. 28, pp. 48–57, July 2011.

[3] R. Raskar, A. Agrawal, and J. Tumblin, "Coded exposure photography: Motion deblurring using fluttered shutter," *ACM Trans. Graph.*, vol. 25, pp. 795–804, July 2006.

[4] J. L. Starck, E. Pantin, and F. Murtagh, "Deconvolution in Astronomy: A Review," *Publications of the Astronomical Society of the Pacific*, vol. 114, pp. 1051–1069, 2002.

[5] R. M. Parton and I. Davis, *Cell Biology*, ch. Lifting the Fog: Image Restoration by Deconvolution, pp. 187 – 200. Burlington, USA: Elsevier Academic Press, third edition ed., 2006.

[6] J. M. Cecilia, F. J. Moreno, and J. M. García, "Acelerando algoritmos de deconvolución para imagenes de microscopía usando gpus," in *Proc. of the II Workshop en Aplicaciones de Nuevas Arquitecturas de Consumo y Altas Prestaciones (ANACAP09)*, (Madrid, Spain), November, 2009.

[7] J.-L. Wu, C.-F. Chang, and C.-S. Chen, "An improved richardson-lucy algorithm for single image deblurring using local extrema filtering," in *Intelligent Signal Processing and Communications Systems (ISPACS), 2012 International Symposium on*, pp. 27–32, Nov 2012.

[8] R. L. White, "Image restoration using the damped richardson-lucy method," in *Instrumentation in Astronomy VIII*, vol. 2198, pp. 1342–1348, June, 1994.

[9] D. S. C. Biggs and M. Andrews, "Acceleration of iterative image restoration algorithms," *Appl. Opt.*, vol. 36, pp. 1766–1775, Mar 1997.

[10] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 5th ed., 2013.

[11] T. Lenart, *Design of Reconfigurable Hardware Architectures for Real-time Applications.* PhD thesis, Lund University, 2008.

[12] R. J. Tocci, *Sistemas digitais: princípios e aplicações*, ch. Arquiteturas de dispositivos lógicos programáveis, pp. 727 – 751. São Paulo: Pearson Prentice Hall, 10a ed., 2007.

[13] J. A. GARCÍA, "Implementação em gpga de uma biblioteca parametrizável para inversão de matrizes baseada no algoritmo gauss-jordan, usandoo respresentação em ponto flutuante," mestrado, Departamento de Engenharia Mecânica, UNB, Universidade de Brasília, Brasília, DF, Setembro 2010.

[14] B. Padilha, G. D. Bonis, and L. Kayo, "Gpu," 2010. Monografia de organização de computadores, Instituto de Matemática e Estatística, USP, Universidade de São Paulo, São Paulo.

[15] J.-O. Jeong, "Hybrid fpga and gpp implementation of ieee 802.15.4 physical layer," master's thesis, Virginia Polytechnic Institute and State University, July 30, 2012.

[16] K. A. Yelick, "Ten ways to waste a parallel computer.," in *ISCA* (S. W. Keckler and L. A. Barroso, eds.), p. 1, ACM, 2009.

[17] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[18] S. Borkar and A. A. Chien, "The future of microprocessors," *Commun. ACM*, vol. 54, pp. 67–77, May 2011.

[19] C. S. FERREIRA, "Implementação do algoritmo de subtração de fundo para detecção de objetos em movimento, usando sistemas reconfiguráveis," mestrado, Departamento de Engenharia Mecânica, UNB, Universidade de Brasília, Brasília, DF, Março 2012.

[20] L. J. Baumstark and L. Wills, "Exposing data-level parallelism in sequential image processing algorithms," in *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*, (Virginia, USA), pp. 245–254, October, 2002.

[21] S. Kagami, T. Komuro, and M. Ishikawa, "A software-controlled pixel-level a-d conversion method for digital vision chips," in *2003 IEEE Workshop on Charge-Coupled Devices and Advanced Image Sensors*, (Elmau, Germany), May, 2003.

[22] R. COLLETT, *Application Specific Integrated Circuit (ASIC) Technology*, ch. Market Dynamics of the ASIC Revolution, pp. 7 – 25. Academic Press, 1991.

[23] A. D. Gonsales, “Projeto de uma nova arquitetura de fpga para aplicações bist e dsp,” mestrado, Instituto de Informática, UFRGS, Universidade Federal do Rio Grande do Sul, Agosto 2002.

[24] F. Vahid, *Sistemas digitais: projeto, otimização e HDLs.* Porto Alegre: Bookman, 2008.

[25] A. C. de Oliveira Souza Aragão, “Uma arquitetura sistólica para solução de sistemas lineares implementada com circuitos fpgas,” mestrado, Instituto de Ciências Matemáticas e de Computação-ICMC, USP, Universidade de São Paulo, Dezembro 1998.

[26] J. Y. M. A. D. SILVA, “Implementação de técnicas de processamento de imagens no domínio espacial em sistemas reconfiguráveis,” mestrado, Departamento de Engenharia Mecânica, UNB, Universidade de Brasília, Brasília, DF, Janeiro 2010.

[27] L. A. Casillo, “Projeto e implementação em fpga de um processador com conjunto de instrução reconfigurável utilizando vhdl,” mestrado, Departamento de Informática e Matemática Aplicada, UFRN, Universidade Federal do Rio Grande do Norte, Natal, RN, Maio 2015.

[28] H. Pedrini and W. Schwartz, *Análise de imagens digitais: princípios, algoritmos e aplicações.* São Paulo: Thomson Learning, 2008.

[29] I. A. Esquef, “Processamento digital de imagens,” mestrado, Centro Brasileiro de Pesquisas Físicas, Rio de Janeiro, RJ, Dezembro 2002.

[30] I. Pitas, *Digital Image Processing Algorithms and Applications.* New York, NY, USA: John Wiley & Sons, Inc., 1st ed., 2000.

[31] R. A. Schowengerdt, ed., *Remote Sensing*, ch. Spectral transforms, pp. 183 – 228. Burlington: Academic Press, third edition ed., 2007.

[32] C. Torres-Huitzil and M. Arias-Estrada, “Real-time image processing with a compact fpga-based systolic architecture,” *Real-Time Imaging*, vol. 10, no. 3, pp. 177 – 187, 2004.

[33] R. C. C. de Souza, “Avaliação de imagens através de similaridade estrutural e do conceito de mínima diferença de cor perceptível,” mestrado, Departamento de Eletrônica e Telecomunicações, UERJ, Universidade de Rio de Janeiro, Outubro 2009.

[34] B. K. B. K. Gunturk and X. Li, eds., *Image restoration : fundamentals and advances.* Digital imaging and computer vision, Boca Raton, FL: CRC Press, 2012.

[35] Y.-W. Tai, P. Tan, and M. Brown, “Richardson-lucy deblurring for scenes under a projective motion path,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, pp. 1603–1618, Aug 2011.

[36] M. Stelzer, “Restauração de imagens de microscopia confocal utilizando técnicas pocs,” mestrado, Departamento de Computação, UFSCAR, São Carlos, SP, Setembro 2013.

[37] M. Tanaka and T. Takahama, “Restoration of motion-blurred line drawings by differential evolution and richardson-lucy algorithm,” in *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pp. 1–8, June 2012.

[38] W. B. D. SILVA, *Métodos sem referência baseados em características espaço-temporais para avaliação objetiva de qualidade de vídeo digital.* Doutorado, Departamento Acadêmico de Eletrônica, UTFPR, Universidade Tecnológica Federal do Paraná, Març 2013.

[39] Z. Wang and A. Bovik, “Mean squared error: Love it or leave it? a new look at signal fidelity measures,” *Signal Processing Magazine, IEEE*, vol. 26, pp. 98–117, Jan 2009.

[40] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *Image Processing, IEEE Transactions on*, vol. 13, pp. 600–612, April 2004.

[41] L. Zhang and H. Li, “Sr-sim: A fast and high performance iqa index based on spectral residual,” in *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pp. 1473–1476, Sept 2012.

[42] F. Diewald, J. Klappstein, and J. Dickmann, “An adaption of the lucy-richardson deconvolution algorithm to noncentral chi-square distributed data,” in *Proceedings of the IAPR Conference on Machine Vision Applications (IAPR MVA 2011), Nara Centennial Hall, Nara, Japan, June 13-15, 2011*, pp. 389–392, 2011.

[43] H. Wang and P. Miller, “Scaled heavy-ball acceleration of the richardson-lucy algorithm for 3d microscopy image restoration,” *Image Processing, IEEE Transactions on*, vol. 23, pp. 848–854, Feb 2014.

[44] Prato, M., Cavicchioli, R., Zanni, L., Boccacci, P., and Bertero, M., “Efficient deconvolution methods for astronomical imaging: algorithms and idl-gpu codes,” *Astronomy & Astrophysics manuscript*, vol. 539, p. A133, October 2012.

[45] M. E. Angelopoulou, C.-S. Bouganis, P. Y. K. Cheung, and G. A. Constantinides, “Robust real-time super-resolution on fpga and an application to video enhancement,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, pp. 22:1–22:29, Sept. 2009.

[46] N. M. Figueira and L. C. de Oliveira, “Super-resolução: Técnicas existentes e possibilidade de emprego às imagens do vant vt-15,” *Revista Militar de Ciência e Tecnologia*, vol. 30, pp. 3–19, 2013.

[47] R. Zhen, “Enhanced raw image capture and deblurring,” mastersś thesis, Electrical Engineering, University of Notre Dame, Notre Dame, Indiana, April 2013.

[48] K. R. G. da Silva, *Uma metodologia de verificação Funcional para circuitos Digitais*. Doutorado, Departamento de Engenharia Elétrica, UFCG, Universidade Federal de Campina Grande, Campina Grande, PB, Fevereiro 2007.