Emerson Carlos Pedrino

Arquitetura *pipeline* reconfigurável através de instruções geradas por programação genética para processamento morfológico de imagens digitais utilizando FPGAs

Tese apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

Área de concentração: Processamento de sinais e instrumentação

Orientador: Prof. Dr. Valentin Obac Roda

São Carlos
2008

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Pedrino, Emerson Carlos
P371a Arquitetura *pipeline* reconfigurável através de instruções geradas por programação genética para processamento morfológico de imagens digitais utilizando FPGAs / Emerson Carlos Pedrino ; orientador Valentin Obac Roda. –- São Carlos, 2008.

Tese (Doutorado) - Programa de Pós-Graduação em Engenharia Elétrica e Área de Concentração em Processamento de sinais e instrumentação –- Escola de Engenharia de São Carlos da Universidade de São Paulo.

1. Morfologia matemática. 2. Arquitetura *pipeline*.
3. FPGAs.
4. Programação genética. I. Título.

FOLHA DE JULGAMENTO

Candidato: Bacharel **EMERSON CARLOS PEDRINO**

Tese defendida e julgada em 27/11/2008 perante a Comissão Julgadora:

Aprovado

Prof. Associado **VALENTINOBAC RODA (Orientador)**
(Escola de Engenharia de São Carlos/USP)

Aprovado

Prof. Dr. **MAXIMILIAN LUPPE**
(Escola de Engenharia de São Carlos/USP)

Aprovado

Prof. Dr. **JORGE LUIZ E SILVA**
(Instituto de Ciências Matemáticas e de Computação/USP)

Aprovado

Prof. Dr. **JACQUES FACON**
(Pontifícia Universidade Católica/PUC/PR)

Aprovado

Prof. Associado **JOSÉ HIROKI SAITO**
(Universidade Federal de São Carlos/UFSCar)

Prof. Associado **GERALDO ROBERTO MARTINS DA COSTA**
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica e
Presidente da Comissão de Pós-Graduação

DEDICATÓRIA

Dedico este trabalho a meu pai Antonio (*in memorian*), a minha família, em especial a minha esposa Marly e a minha filha Letícia, ao professor Valentin, meu orientador e grande amigo, com quem tive o imensurável prazer de compartilhar experiências valiosas de vida durante os períodos de mestrado e doutorado.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por todas as bênçãos a mim concedidas até o presente momento. Agradeço aos meus familiares, em especial a minha esposa Marly, por sua compreensão durante este período de trabalho incessante. Agradeço ao meu orientador Professor Valentin, por sua bondade, amizade e dedicada orientação. Também agradeço ao Professor Sergio A. Rohm da UFSCar, por sua amizade, conselhos e ajuda em diversos momentos durante este processo. Agradeço aos Professores Nelson D. A. Mascarenhas da UFSCar e Maria C. Monard da USP por terem participado de minha banca de qualificação de doutorado e por suas valiosas sugestões feitas a este trabalho. Agradeço aos amigos do laboratório pela amizade e apoio, à funcionária Elenise da biblioteca da EESC, que sempre me ajudou nas aquisições dos artigos citados neste trabalho, aos funcionários, amigos e professores do departamento de Engenharia Elétrica que me apoiaram em diversas ocasiões e à UFSCar pelo afastamento me concedido para a realização desta tese.

EPÍGRAFE

“O primeiro lugar cabe aos que nascem com o saber. Vêm a seguir aqueles que têm de estudar para adquiri-lo. Depois, vêm os que só o adquirem à custa de grandes esforços. Quanto aos que não têm nem inteligência nem vontade de aprender, são esses os últimos homens”.

Confúcio

RESUMO

PEDRINO, E. C. **Arquitetura *Pipeline* reconfigurável através de instruções geradas por programação genética para processamento morfológico de imagens digitais utilizando FPGAs.** 2008. 208 f. Tese (Doutorado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

A morfologia matemática fornece ferramentas poderosas para a realização de análise de imagens em baixo nível e tem encontrado aplicações em diversas áreas, tais como: visão robótica, inspeção visual, medicina, análise de textura, entre outras. Muitas dessas aplicações requerem processamento em tempo real e para sua execução de forma eficiente freqüentemente é utilizado *hardware* dedicado. Também, a tarefa de projetar operadores morfológicos manualmente para uma dada aplicação não é trivial na prática. A programação genética, que é um ramo relativamente novo em computação evolucionária, está se consolidando como um método promissor em aplicações envolvendo processamento de imagens digitais. Seu objetivo primordial é descobrir como os computadores podem aprender a resolver problemas sem, no entanto, serem programados para essa tarefa. Essa área ainda não foi muito explorada no contexto de construção automática de operadores morfológicos. Assim, neste trabalho, desenvolve-se e implementa-se uma arquitetura original, de baixo custo, reconfigurável por meio de instruções morfológicas e lógicas geradas automaticamente através de uma aproximação linear baseada em programação genética, visando-se o processamento morfológico de imagens em tempo real utilizando FPGAs de alta complexidade, com objetivos de filtragem, reconhecimento de padrões e emulação de filtros desconhecidos de *software*s comerciais, para citar somente algumas aplicações. Exemplos de aplicações práticas envolvendo imagens binárias, em níveis de cinza e coloridas são fornecidos e seus resultados são comparados com outras formas de implementação.

Palavras-chave: Morfologia matemática. Arquitetura *pipeline*. FPGAs. Programação genética.

ABSTRACT

PEDRINO, E. C. **Reconfigurable pipelined architecture through instructions generated by genetic programming for morphological image processing using FPGAs**. 2008. 208 f. Thesis (Doctoral) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

Mathematical morphology supplies powerful tools for low level image analysis, with applications in robotic vision, visual inspection, medicine, texture analysis and many other areas. Many of the mentioned applications require dedicated *hardware* for real time execution. The task of designing manually morphological operators for a given application is not always a trivial one. Genetic programming is a relatively new branch of evolutionary computing and it is consolidating as a promising method for applications of digital image processing. The main objective of genetic programming is to discover how computers can learn to solve problems without being programmed for that. In the literature little has been found about the automatic morphological operators construction using genetic programming. In this work, the development of an original reconfigurable architecture using logical and morphological instructions generated automatically by a linear approach based on genetic programming is presented. The developed architecture is based on Field Programmable Gate Arrays (FPGAs) and has among the possible applications, image filtering, pattern recognition and filter emulation. Binary, gray level and color image practical applications using the developed architecture are presented and the results are compared with other implementation techniques.

Keywords: Mathematical morphology. Pipelined architecture. FPGAs. Genetic programming.

SUMÁRIO

DEDICATÓRIA **5**

AGRADECIMENTOS **6**

EPÍGRAFE **7**

RESUMO **8**

ABSTRACT **9**

SUMÁRIO **10**

1 INTRODUÇÃO **13**

1.1 OBJETIVOS E MOTIVAÇÕES *17*

1.2 ESTRUTURA DO TRABALHO *18*

2 DISPOSITIVOS DE LÓGICA PROGRAMÁVEL (PLDs) **21**

2.1 CONSIDERAÇÕES INICIAIS *21*
2.1.1 Metodologias de Projeto de Circuitos Integrados 22

2.2 EVOLUÇÃO DOS PLDS *25*

2.3 TECNOLOGIAS DE PROGRAMAÇÃO *27*

2.4 CPLDs *30*
2.4.1 Aplicações de CPLDs 31

2.5 FPGAs *32*
2.5.1 Aplicações de FPGAs 34
2.5.1.1 Computação Reconfigurável 34

2.6 PROCESSO DE PROJETO PARA PLDS *36*

2.7 ESTADO DA ARTE *38*

2.8 CONSIDERAÇÕES FINAIS *40*

3 MORFOLOGIA MATEMÁTICA **41**

3.1 CONSIDERAÇÕES INICIAIS *41*

3.2 OPERADORES BÁSICOS DE MORFOLOGIA MATEMÁTICA *44*
3.2.1 Dilatação 44
3.2.1.1 Adição de Minkowski 45
3.2.2 Erosão 46
3.2.2.1 Subtração de Minkowski 47
3.2.3 Efeitos da Dilatação e da Erosão 48
3.2.4 Abertura e Fechamento 48
3.2.5 Morfologia Matemática para Imagens em Níveis de Cinza 49
3.2.5.1 Dilatação 49
3.2.5.2 Erosão 50
3.2.5.3 Abertura e Fechamento 51

3.3 MORFOLOGIA PARA RETICULADOS *52*
3.3.1 Reticulado Completo 53

3.3.2 Operadores Básicos entre Reticulados 55
3.3.3 Métodos de Ordenação Vetorial 56
3.3.3.1 Processamento Morfológico de Imagens Coloridas 57
3.3.3.1.1 Ordenação por uma Componente 60
3.3.3.1.2 Ordenação por Medida de Distância 61
3.3.3.1.3 Ordenação Canônica 61
3.3.3.1.4 Ordenação Lexicográfica 62
3.3.3.1.5 Outros Métodos de Ordenação Vetorial 63

3.4 CONSIDERAÇÕES FINAIS *64*

4. COMPUTAÇÃO EVOLUCIONÁRIA **67**

4.1 CONSIDERAÇÕES INICIAIS *67*

4.2 ALGORITMOS EVOLUCIONÁRIOS *69*
4.2.1 Componentes dos Algoritmos Evolucionários 71
4.2.1.1 Representação (Definição de Indivíduos) 72
4.2.1.2 Função de Aptidão 72
4.2.1.3 População 73
4.2.1.4 Mecanismo de Seleção de Pais 73
4.2.1.5 Operadores de Variação 74
4.2.1.5.1 Mutação 74
4.2.1.5.2 Cruzamento 75
4.2.1.6 Mecanismo de Seleção Ambiental 75
4.2.1.7 Condição de Parada 76

4.3 PROGRAMAÇÃO GENÉTICA *76*
4.3.1 Passos Preparatórios em Programação Genética 79
4.3.2 Detalhes de Execução 80
4.3.3 Teorema dos Esquemas 87

4.4 CONSIDERAÇÕES FINAIS *90*

5. IMPLEMENTAÇÕES **93**

5.1 CONSIDERAÇÕES INICIAIS *93*

5.2 ARQUITETURAS EXISTENTES PARA MORFOLOGIA MATEMÁTICA *94*
5.2.1 Arquiteturas de Propósito Geral para Processamento de Imagens 94
5.2.2 Estruturas de *Hardware* Especializadas para Processamento Morfológico de Imagens 98
5.2.2.1 Arquiteturas *Pipeline* Morfológicas para Processamento de Imagens Binárias 105

5.3 HARDWARE RECONFIGURÁVEL PARA PROCESSAMENTO MORFOLÓGICO DE IMAGENS COLORIDAS *112*
5.3.1 Arquitetura *Pipeline* para Processamento Morfológico de Imagens Coloridas em Tempo Real 113
5.3.1.1 Sistema RGB para Aquisição e Processamento de Imagens Coloridas Digitais utilizando *Hardware* Reconfigurável 113
5.3.1.1.1 Sensor OV7620 114
5.3.1.1.2 Protocolo SCCBus 115
5.3.1.1.3 Formato RGB *Raw* 119
5.3.1.1.4 Sistema RGB desenvolvido 119
5.3.1.2 Arquitetura Paralela para Processamento Morfológico Rápido de Imagens Coloridas 125
5.3.1.2.1 *Hardware* Desenvolvido 125
5.3.1.2.1.1 Definições Básicas 126
5.3.2.2.1.2 Passos do Algoritmo 126

5.4 CONSTRUÇÃO AUTOMÁTICA DE OPERADORES MORFOLÓGICOS UTILIZANDO PROGRAMAÇÃO GENÉTICA *130*
5.4.1 Construção Automática de Operadores Morfológicos 131
5.4.2 Alguns Exemplos de Aplicação 137
5.4.2.1 Comparações dos Resultados Obtidos com Outros Trabalhos 144

5.5 ARQUITETURA PIPELINE RECONFIGURÁVEL EM HARDWARE POR INSTRUÇÕES GERADAS AUTOMATICAMENTE POR PROGRAMAÇÃO GENÉTICA PARA PROCESSAMENTO MORFOLÓGICO DE IMAGENS EM TEMPO REAL *146*
5.5.1 Estágios da Arquitetura Desenvolvida 147
5.5.1.1 Decodificador de Vídeo 148
5.5.1.2 Arquitetura *Pipeline* 149
5.5.1.3 Conversor D/A 155
5.5.2 Exemplos de Aplicação utilizando o *Hardware* Desenvolvido 156

5.6 CONSIDERAÇÕES FINAIS *166*

6. CONCLUSÕES **169**

6.1 CONTRIBUIÇÕES *171*

6.2 ORIGINALIDADE *173*

REFERÊNCIAS* **175**

APÊNDICE A – Diagrama Esquemático do *Hardware* do Protocolo SCCB **197**

APÊNDICE B – Diagrama Esquemático do *Hardware* do Controlador RGB **198**

APÊNDICE C – Diagrama Esquemático do Circuito Detector de TV **199**

APÊNDICE D – Diagrama Esquemático do Circuito VGA de Saída **200**

APÊNDICE E – Diagrama Esquemático de um Estágio da Arquitetura *Pipeline* **201**

APÊNDICE F – Código em Verilog HDL de um Elemento de Processamento **202**

APÊNDICE G – Algumas Fotos dos Experimentos **205**

APÊNDICE H - Publicações Geradas Durante o Doutorado **207**

1 INTRODUÇÃO

A análise de imagens em baixo nível envolve o uso de estruturas grandes de dados e freqüentemente requer altas taxas de computação. Muitas arquiteturas dedicadas já foram propostas na literatura e construídas para lidar com essas tarefas de processamento de imagens. Logo, é intuitivo pensar em alguma forma de paralelismo para que se possam executar essas tarefas de forma eficiente (PEDRINO, 2003).

De acordo com (ABOTT et al., 1988), em se tratando de paralelismo, as arquiteturas *pipeline* podem algumas vezes oferecer vantagens significativas em relação às arquiteturas de arranjos de elementos de processamento individuais.

O avanço da tecnologia tornou possível a manipulação de grande quantidade de dados e a aquisição de imagens digitais, assim, procedimentos computacionais que visam auxiliar o processamento e a análise de imagens digitais estão sendo cada vez mais utilizados. Nessa abordagem, esses são importantes no auxílio de diagnósticos médicos, em automação industrial, no processamento de documentos, na área de sensoriamento remoto, entre outras aplicações (HIRATA, 2000). Uma técnica que está sendo amplamente utilizada para a implementação dos procedimentos supracitados é a morfologia matemática.

A morfologia matemática é definida como o estudo da forma utilizando ferramentas da teoria de conjuntos e é comumente e eficientemente usada em tarefas de análise e processamento de imagens (FACON, 1996; BROGGI, 1994). A morfologia matemática é uma área não linear em processamento de imagens e se baseia na geometria e teoria da ordem (MATHERON, 1975; SERRA, 1988; SOILLE, 1999). Os primeiros trabalhos nessa área se devem a Minkowsky e Hadwiger com a noção de soma e subtração de Minkowsky. A

reformulação desses trabalhos se daria nos anos 60 através das pesquisas conjuntas de Matheron e Serra. Nesse mesmo período foi criado o centro de morfologia matemática da Escola de Minas de Paris em Fontainebleu (França) (DOUGHERTY, 1992a). As ferramentas básicas de morfologia matemática são a dilatação e a erosão, que podem ser descritas por operadores lógicos e aritméticos. Através dos operadores básicos é possível realizar outras operações mais complexas. Contudo, os operadores clássicos não podem ser aplicados a imagens coloridas. Dessa maneira, a morfologia matemática pode ser descrita de forma algébrica através de operadores sobre reticulados completos (BIRKHOFF, 1984). A teoria de reticulados foi utilizada para formalizar a morfologia para imagens digitais não necessariamente binárias ou em níveis de cinza. De maneira formal, a morfologia matemática estuda mapeamentos entre reticulados completos. Nesse contexto, a noção de ordem é muito importante. Imagens coloridas possuem uma grande quantidade de informações, assim, estas podem ser úteis no processo de análise de imagens baseada em padrões coloridos. Também, essas imagens têm um papel importante no processo de percepção visual do ser humano e estão se tornando amplamente disponíveis nos dias de hoje. A extensão dos operadores morfológicos clássicos para imagens coloridas não é uma tarefa trivial na prática. Em imagens coloridas, não há uma ordem natural para os vetores representando os *pixel*s de imagem, logo, essa extensão requer um estudo específico de ordem em dados multivariados.

A tarefa de projetar operadores morfológicos manualmente que realizem transformações adequadas em imagens, para que se possam extrair das mesmas, informações de interesse, não é trivial na prática. Esta requer do projetista um conhecimento mais abrangente de técnicas de processamento digital de imagens. Assim, tais dificuldades motivaram pesquisadores de morfologia matemática a buscarem técnicas automáticas de projeto de operadores morfológicos. Existem atualmente diversas tentativas para automatizar o projeto de operadores morfológicos. Geralmente, esses procedimentos trabalham em um

espaço fixo de operadores e são baseados em algoritmos que buscam no espaço considerado aqueles operadores que melhor se adequem à descrição dada. As técnicas existentes dividem-se basicamente em: técnicas de inteligência artificial, baseadas em modelagem de imagens e em técnicas de indução a partir de exemplos (SCHIMITT, 1989; SCHONFELD, 1994; DOUGHERTY; ZHAO, 1992; DOUGHERTY, 1992b; DOUGHERTY, 1992c; SARCA et al., 1999). Também foram propostas técnicas adaptativas como algoritmos genéticos ou redes neurais (HARVEY; MARSHALL, 1996; YODA et al., 1999; SARCA et al., 1998; WILSON, 1993). No entanto, todas essas técnicas citadas apresentam limitações de dependência de contexto, imprecisão dos operadores projetados e complexidade de tempo. Uma das ramificações de computação evolucionária que está sendo amplamente utilizada em aplicações de processamento de imagens e que está se consolidando como um método promissor é a programação genética. A programação genética (PG) foi idealizada por John Koza (KOZA, 1989) com base nos trabalhos de John Holland em algoritmos genéticos (AGs) (HOLLAND, 1975) e seu objetivo primordial é descobrir como os computadores podem aprender a resolver problemas sem, no entanto, serem programados para essa tarefa (KOZA, 1992). Essa abordagem estende os domínios dos algoritmos genéticos, nos quais os cromossomos são representados por programas de computador de tamanhos variados. No entanto, essa área ainda não foi muito explorada no contexto de construção automática de operadores morfológicos.

As ferramentas de morfologia matemática são bem adequadas à implementação em arquiteturas *pipeline*. Segundo (HARALICK et al., 1987), uma vez que a morfologia matemática se relaciona diretamente com a forma, torna-se evidente que o processamento natural para tratar problemas de identificação de objetos por visão de máquina ou ainda identificar objetos defeituosos numa linha de montagem é a morfologia matemática. Também, dilatando ou erodindo objetos de forma algorítmica, podem-se detectar bordas, esqueletos,

eliminar ruídos e outras operações. Na literatura foram propostas algumas implementações em *hardware* para lidar com o processamento morfológico de imagens binárias e em níveis de cinza em tempo real. No entanto, as arquiteturas propostas impõem diversas restrições em relação ao tamanho e a forma dos elementos estruturantes utilizados nas operações morfológicas, resolução das imagens processadas e complexidade do *hardware* envolvido. Em (PEDRINO, 2003; PEDRINO; RODA, 2007), propõe-se e implementa-se uma arquitetura *pipeline* reconfigurável através do uso de dispositivos lógicos programáveis de alta complexidade para processamento morfológico de imagens binárias em tempo real. A arquitetura desenvolvida pode lidar com qualquer tamanho e forma de elemento estruturante, dependendo da densidade lógica do dispositivo reconfigurável, e utiliza a técnica de decomposição do elemento estruturante para acelerar o processamento em *hardware*. Também, na literatura não existe um número significativo de arquiteturas voltadas para processamento morfológico de imagens coloridas em *hardware*, talvez devido às diversas limitações existentes a serem abordadas neste texto.

O grande desafio na área de visão computacional é o desenvolvimento de sistemas capazes de realizar o processamento de imagens digitais em tempo-real com o menor custo possível e com possibilidade de uso em aplicações industriais e comerciais. A utilização de computadores de uso geral permite a rápida implementação de sistemas e de vários tipos de algoritmos, simples ou complexos, específicos para o processamento de imagens. Todavia, não é possível obter o maior rendimento exigido com essas máquinas devido a diversas limitações. Uma das maneiras de contornar esse inconveniente é através do uso de arquiteturas dedicadas projetadas em *hardware* para executar os algoritmos de processamento de imagens em tempo real. A utilização de dispositivos lógicos programáveis complexos do tipo CPLDs e FPGAs no desenvolvimento de projetos em *hardware* tem possibilitado o rápido desenvolvimento, análise e implementação de projetos lógicos complexos a um custo

consideravelmente menor quando comparado aos custos de desenvolvimento de uma pastilha VLSI. Os dispositivos atuais possuem dezenas de milhares de portas lógicas, blocos de DSP, memórias internas, CPUs embutidas e outros blocos dedicados, formando-se assim um sistema completo numa única pastilha. Também, sua capacidade de reprogramação em *hardware* tem permitido o desenvolvimento de sistemas de computação reconfigurável de forma dinâmica. Para a descrição de sistemas complexos, já estão disponíveis técnicas de programação em alto nível. Assim, o desenvolvimento de projetos de sistemas específicos para o processamento de imagens de forma customizada tem se beneficiado com essa nova tecnologia, tanto pela diminuição do tempo necessário para a implementação dos projetos e dos gastos com material e mão de obra, quanto pela possibilidade de implementação de sistemas que processem as informações em tempo real.

1.1 OBJETIVOS E MOTIVAÇÕES

O objetivo principal do presente trabalho foi desenvolver e implementar uma arquitetura *pipeline* original e de baixo custo para processamento morfológico de imagens em tempo real, onde seus processadores são configurados por um conjunto de instruções morfológicas e lógicas geradas automaticamente através de uma abordagem linear baseada em programação genética por meio de uma *toolbox* desenvolvida utilizando o *software* Matlab 7.0 da Mathworks. Esta arquitetura é reconfigurável em *hardware* por meio de uma interface USB, através do uso da placa educacional e de desenvolvimento DE2 da Altera (ALTERA, 2008a) que contém um dispositivo FPGA da família Cyclone II. O sistema evolucionário criado é capaz de construir, de forma automática, procedimentos morfológicos e lógicos para processamento de imagens em tempo real, onde este também poderá trabalhar com outros

algoritmos para soluções de problemas práticos envolvendo visão computacional através do uso de *hardware* dedicado. Esta arquitetura lida não somente com imagens binárias e em níveis de cinza, mas também com imagens coloridas. Os dados a serem processados são fornecidos por um *hardware* dedicado utilizando um sensor de vídeo CMOS, capaz de processar 60 quadros/s, desenvolvido neste trabalho para processamento de imagens coloridas RGB. A resolução espacial deste sistema é de 320x240 *pixels*. Também, nos testes, foi utilizada uma câmera comercial de vídeo CMOS com resolução espacial de 640x480 *pixels* e 30 quadros/s. Além dos sistemas supracitados, são propostos: um método de ordenação vetorial de dados RGB e um algoritmo para a extensão dos conceitos de morfologia matemática a imagens coloridas visando sua facilidade de implementação em *hardware*.

Além da motivação de construir uma arquitetura reconfigurável para processamento de imagens em geral em tempo real, a outra motivação do trabalho foi idealizar um método linear de construção automática de operadores morfológicos e lógicos utilizando os conceitos de programação genética, considerando-se que esta abordagem ainda não foi muito explorada no contexto de processamento morfológico de imagens digitais, conforme já abordado. Exemplos práticos de aplicações são propostos e apresentados, permitindo-se assim a verificação e a análise de funcionamento e dos resultados obtidos por esta metodologia. Também, os resultados obtidos são comparados com outras formas de implementação.

1.2 ESTRUTURA DO TRABALHO

No capítulo 1, apresenta-se uma visão geral sobre morfologia matemática e discute-se a possibilidade de construção automática de seus operadores, citando-se as principais técnicas existentes para esta tarefa. Também, aborda-se sobre a possibilidade de extensão desses operadores para imagens coloridas e sobre a motivação de implementação destes através do

uso de *hardware* reconfigurável. Finalmente, são apresentados os objetivos e as motivações do trabalho.

No capítulo 2 é apresentada uma revisão sobre dispositivos lógicos programáveis de alta capacidade, onde estes são inseridos no contexto das demais metodologias alternativas para projeto de sistemas digitais. Também é realizada uma análise sobre a tecnologia de programação dos diversos dispositivos encontrados nos dias de hoje, de sua arquitetura de roteamento e arquitetura de seus blocos lógicos constituintes. O tema *computação reconfigurável* é discutido de forma sucinta. No final do capítulo, apresenta-se o processo de projeto desses dispositivos e seu estado da arte, com especial ênfase nos dispositivos das empresas Altera e Xilinx.

No capítulo 3 é realizada uma revisão sobre morfologia matemática clássica, na qual se destacam os processos de adição e subtração de Minkowski. Também, os operadores clássicos são estendidos para imagens em níveis de cinza e coloridas.

No capítulo 4, realiza-se uma revisão sobre a área de computação evolucionária e de seus operadores clássicos. Uma de suas ramificações, a programação genética, é introduzida nesse capítulo, uma vez que esta é a metodologia utilizada no desenvolvimento dos algoritmos presentes neste trabalho. Também, discute-se, de forma breve, o teorema dos esquemas.

No capítulo 5 são apresentadas as implementações realizadas durante o período de doutorado, onde a principal contribuição foi o desenvolvimento de uma arquitetura *pipeline* para processamento morfológico de imagens digitais em tempo real. Inicialmente é realizada uma revisão sobre as principais arquiteturas em *hardware* voltadas para processamento morfológico de imagens digitais. Em seguida, apresenta-se uma nova proposta de arquitetura

pipeline para processamento morfológico de imagens coloridas em tempo real utilizando um FPGA da empresa Altera. Um novo método de ordenação baseado na distância *city-block* foi proposto para eliminar redundâncias decorrentes do processamento morfológico. Também, é apresentada a implementação de um sistema RGB utilizando um dispositivo CPLD da Altera para pré-processamento de imagens coloridas em tempo real, onde este é baseado num sensor de vídeo CMOS e reconfigurável em *hardware* através do protocolo SCCBus implementado por meio de um sistema microprocessado utilizando um microcontrolador da Atmel. Por fim, apresenta-se o sistema evolucionário desenvolvido no trabalho. Primeiramente é apresentada a implementação de uma *toolbox* utilizando o *software* Matlab 7.0 da Mathworks para construção automática de operadores morfológicos e lógicos por meio de uma abordagem linear baseada em programação genética para aplicações envolvendo filtragem morfológica, reconhecimento de padrões, emulação de filtros desconhecidos, entre outras aplicações. Em seguida, é apresentada a arquitetura *pipeline* reconfigurável em *hardware* pelas instruções geradas através do procedimento genético citado anteriormente, com a finalidade de processar imagens digitais em tempo real. Esta arquitetura foi desenvolvida utilizando a placa de desenvolvimento e educacional DE2 da empresa Altera, que é baseada num dispositivo FPGA da família Cyclone II. Exemplos de aplicações práticas são apresentados e seus resultados são comparados com outras formas de implementação.

Para finalizar, apresentam-se as conclusões do trabalho e comenta-se sobre sua originalidade.

2 DISPOSITIVOS DE LÓGICA PROGRAMÁVEL (PLDs)

2.1 CONSIDERAÇÕES INICIAIS

Dispositivos lógicos programáveis constituem uma ferramenta extremamente poderosa para projetistas de sistemas digitais nos dias de hoje. Um PLD pode ser definido basicamente como um arranjo de portas lógicas que pode ser configurado através de um aplicativo para realizar uma dada função lógica. Sua breve história tem mostrado que suas capacidades lógicas crescem de forma aproximada à mesma taxa que a dos microprocessadores, seguindo a lei de Moore (TROPE, 2004). Os dispositivos atuais podem lidar praticamente com qualquer tarefa computacional. Alguns desses já possuem no mínimo uma CPU embutida na mesma pastilha, formando-se assim um sistema completo num único *chip* programável. As Técnicas de programação desses dispositivos também têm evoluído de *Hardware description languages* (HDLs) básicas (Verilog e VHDL) a métodos de descrição em alto nível, como, por exemplo, Streams-C, Handel-C, entre outros. As pesquisas atuais nessa área se concentram basicamente em novos métodos de programação, novos métodos de modelamento, computação biológica, criptografia, experimentos em sala de aula, processamento de imagens em tempo real, reconhecimento de padrões, só para citar alguns exemplos. Muitos PLDs de última geração possuem a capacidade de serem reprogramados totalmente ou parcialmente no próprio sistema (PELLERIN; THIBAULT, 2005). Isto tem possibilitado a criação de sistemas computacionais reconfiguráveis dinamicamente, entretanto, a tecnologia de computação reconfigurável ainda é uma área incipiente. As ferramentas atuais ainda não suportam completamente esta metodologia. Nas próximas seções, pretende-se fornecer uma visão geral sobre esses dispositivos.

2.1.1 Metodologias de Projeto de Circuitos Integrados

Estimulado pelo desenvolvimento de novos tipos de dispositivos lógicos programáveis (PLDs) mais sofisticados, o processo de projeto de circuitos digitais tem mudado radicalmente nos últimos anos (BROWN; ROSE, 1996). Também, de acordo com (CHAN; MOURAD, 1994), o projeto de circuitos digitais tem sofrido várias evoluções nas últimas décadas. Os componentes dos circuitos evoluíram de transistores individuais a circuitos integrados de integração em muito alta escala [*Very large scale integration* (VLSI)]. As ferramentas *Computer aided design* (CAD) têm acelerado o ciclo de projeto. Não é mais necessário montar diferentes componentes ou desenhar portas lógicas individuais. As linguagens de descrição de *hardware* têm facilitado a descrição de projetos complexos. Ferramentas de síntese lógica automática estão disponíveis para criar circuitos que são rapidamente mapeados na tecnologia em questão. Além das mudanças na tecnologia e técnicas de projeto de sistemas VLSI, o ciclo de vida dos produtos modernos está se tornando mais curto em relação ao ciclo de projeto. Assim, há a necessidade de uma rápida prototipação.

As implementações de *chips* podem ser agrupadas em duas categorias principais: circuitos completamente *customizados* e *semicustomizados*, conforme ilustrado na Figura 2.1.

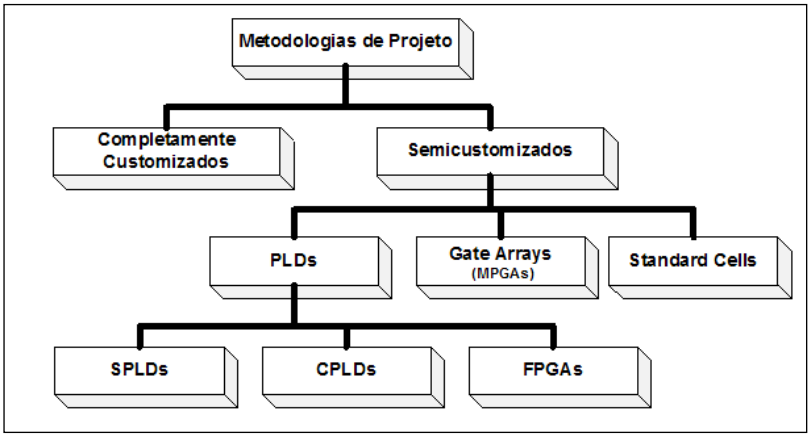


Figura 2.1. Metodologias de projeto de CIs

De acordo com a Figura 2.1, têm-se:

Circuitos integrados (CIs) completamente *customizados*: neste estilo de projeto cada função lógica primitiva é manualmente projetada e otimizada, logo, exigem-se máscaras específicas para cada projeto. Uma vez que o projetista controla todos os estágios de *layout* do *chip*, é possível obter uma máxima flexibilidade de projeto e um alto desempenho. O CI resultante pode ser mais compacto, oferece maior desempenho e um baixo consumo de energia em relação às demais metodologias existentes de projeto. Em contraposição, o tempo de desenvolvimento é longo e os custos envolvidos são extremamente altos. Para grandes volumes de produção, esses CIs fornecem uma alternativa de baixo custo.

***Mask-programmable gate arrays* (MPGAs):** nesta implementação o projeto é geralmente facilitado e agilizado pela existência de uma biblioteca de células. O projeto é mapeado em um arranjo de transistores pré-fabricados numa pastilha. As interconexões do projeto são *customizadas* e feitas durante o processo de fabricação. Logo, há a necessidade de máscaras específicas para as interconexões entre os blocos. O desempenho obtido pode ser muito bom e são mais densos que os PLDs. Pelo fato do arranjo de transistores ser produzido em massa, os custos de fabricação são menores se comparados a CIs completamente *customizados* e *standard cells* (LANDIS, 1996).

***Standard cells*:** como no caso dos MPGAs, a tarefa de projeto é facilitada pelo uso de módulos pré-projetados. Os módulos, *standard cells*, são frequentemente salvos em uma base de dados. O processo de projeto físico é automatizado por ferramentas CAD. Comparados a CIs *customizados*, os circuitos implementados

em *standard cells* são menos eficientes em tamanho e desempenho, entretanto, seu custo de desenvolvimento é mais baixo e o tempo de projeto é menor.

PLDs: termo geral que se refere a qualquer tipo de circuito integrado usado para implementar circuitos digitais onde o *chip* pode ser configurado e reconfigurado pelo usuário final por meio de um *software* específico fornecido pelo seu fabricante. Logo, os projetistas podem produzir CIs *Application specific integrated circuits* (ASICs) sem a necessidade de passar por um processo de fabricação. Também, podem ser reprogramados no próprio sistema pela tecnologia *In-system programmability* (ISP), facilitando-se assim possíveis mudanças de projeto. Possuem como característica principal um ciclo curto de projeto se comparados a CIs completamente *customizados* ou às demais abordagens *semicustomizadas* existentes. Todas as vantagens supracitadas, aliadas a um baixo custo inicial e às altas capacidades lógicas existentes para PLDs, popularizaram seu uso grandemente nos últimos anos (SALSIC; SMAILAGIC, 1997).

Os PLDs se dividem basicamente em:

1. ***Simple PLDs* (SPLDs)**: como exemplo, podem-se citar os dispositivos lógicos programáveis *Programmable logic array* (PLA), atualmente obsoletos, e os *Programmable array logic* (PAL).

2. ***High capacity PLDs* (HCPLDs):** simples acrônimo para ambos, *Complex PLDs* (CPLDs) e *Field programmable gate arrays* (FPGAs).

2.2 EVOLUÇÃO DOS PLDS

O primeiro tipo de *chip* programável pelo usuário que podia implementar circuitos lógicos foi a *Programmable read-only memory* (PROM). Suas linhas de endereçamento podiam ser usadas como entradas do circuito lógico e as linhas de dados como saídas. As funções lógicas, entretanto, raramente requerem mais que alguns termos produto, e uma PROM contém um decodificador completo para seus endereços de entrada, logo, representam uma arquitetura ineficiente para a implementação de circuitos lógicos e raramente são usadas na prática. Posteriormente foram desenvolvidos os PLAs para a implementação de circuitos lógicos. Um PLA possui dois níveis de portas lógicas: um plano de portas AND seguido por um plano de portas OR, ambos programáveis. Cada saída do plano AND pode corresponder a qualquer termo produto de suas entradas. Similarmente, cada saída do plano OR pode ser configurada para produzir a soma lógica de quaisquer saídas do plano AND. Tal estrutura é adequada para a implementação de funções lógicas na forma de soma de produtos. Também, são bastante versáteis, uma vez que ambos os planos, AND e OR, podem ter várias entradas (KATZ, 1993). Quando os PLAs foram introduzidos no início dos anos 70 pela Philips, seus principais problemas eram o alto custo de fabricação e o pobre desempenho de velocidade. Ambas as desvantagens eram devidas aos dois níveis de lógica configurável, pois os planos lógicos eram difíceis de fabricar e introduziam atrasos significantes de propagação. Para superar essas dificuldades, os dispositivos PALs foram desenvolvidos em 1978 pela Monolithic Memories Inc (HOROWITZ; HILL, 1989). Os dispositivos PALs apresentam um único nível de programação, consistindo-se de um plano AND programável seguido por portas OR fixas (KATZ, 1993). Para compensar a falta de generalidade devida ao plano OR fixo, muitas variantes de dispositivos PALs são produzidas com diferentes números de entradas e saídas, e vários tamanhos de portas OR. Também, com esses dispositivos é possível

realizar a implementação de circuitos sequenciais, pois geralmente possuem *flip-flops* conectados às saídas das portas OR. Os dispositivos PALs tiveram um profundo efeito no projeto de *hardware* digital e são a base para dispositivos lógicos programáveis mais complexos. Há outras variantes de arquiteturas semelhantes às dos dispositivos PALs com acrônimos diferentes. Todos os pequenos PLDs, incluindo PLAs, PALs e demais dispositivos similares, são agrupados na categoria de SPLDs, possuindo como características mais importantes um baixo custo e um alto desempenho. Existem duas tecnologias de programação disponíveis para SPLDs, bipolar e *Complementary metal oxide semiconductor* (CMOS). A primeira utiliza fusíveis para as interconexões, sendo, portanto, programável uma única vez, ou seja, caracterizando um dispositivo *One time programmable* (OTP). A última utiliza transistores MOS de *gate* flutuante.

Com o avanço da tecnologia, foi possível produzir dispositivos com maior capacidade que à dos SPLDs. A dificuldade encontrada para aumentar a capacidade dos SPLDs esbarra no fato da estrutura do plano lógico programável crescer muito rapidamente em tamanho à medida que o número de entradas aumenta. Uma maneira de prover dispositivos lógicos programáveis mais complexos baseados em arquiteturas de SPLDs, consiste-se em integrar vários SPLDs em um único *chip* e fornecer interconexões programáveis para conectar seus blocos constituintes.

Os CPLDs foram originalmente desenvolvidos pela Altera, primeiramente em sua família de *chips* conhecida como *Classic Erasable programmable logic devices* (EPLDs) e posteriormente em três séries adicionais: MAX 5000, MAX 7000 e MAX 9000 (ALTERA, 2005). Com o rápido crescimento do mercado de PLDs de alta capacidade, outros fabricantes também desenvolveram dispositivos na categoria de CPLDs. Existe atualmente uma variedade muito grande de opções disponíveis.

Os *chips* lógicos para propósitos gerais de mais alta capacidade disponíveis hoje em dia são os tradicionais *gate arrays* ou MPGAs. Os *gate arrays* motivaram o desenvolvimento dos FPGAs. Assim como esses, os FPGAs são formados por um arranjo bidimensional de blocos lógicos cercados por blocos de I/O programáveis e possuem recursos de interconexões que podem ser programadas pelo usuário final. Também, possuem capacidade superior à dos CPLDs. Logo, oferecem as vantagens de ambos. Os FPGAs foram introduzidos pela Xilinx em 1985 e foram responsáveis pela mudança na maneira de projetar circuitos digitais. Existem diversos fabricantes produzindo FPGAs atualmente. Os três aspectos principais que definem a arquitetura de um FPGA são: tecnologia de programação, arquitetura das células e estrutura de roteamento.

2.3 TECNOLOGIAS DE PROGRAMAÇÃO

O primeiro tipo de comutador programável pelo usuário final foi o fusível. Esta tecnologia foi usada nos PLAs e ainda hoje é utilizada para fabricar SPLDs bipolares (BROWN; ROSE, 1996; KNAPP, 2000). Um fusível é uma conexão metálica que pode ser programada (conexão aberta) fazendo-se passar pela mesma uma corrente de um determinado valor. Portanto, sua tecnologia é OTP. Para dispositivos de alta capacidade, onde a tecnologia CMOS prevalece, foram desenvolvidas diferentes abordagens para a implementação de comutadores programáveis. Para CPLDs, a tecnologia principal utilizada é a de transistores com *gate* flutuante, que são os mesmos utilizados em memórias *Erasable programmable read only memory* (EPROM) e *Electrically erasable programmable read only memory* (EEPROM). Para FPGAs, as tecnologias mais comuns são *Static RAM* (SRAM) e *antifuse*.

As tecnologias EPROM e EEPROM são geralmente utilizadas em SPLDs e CPLDs. As EPLDs da família MAX 5000 da Altera e as da Xilinx, utilizam células EPROM em suas

tecnologias de programação. Um transistor EPROM se parece com um transistor MOS a menos de um segundo *gate*, isolado por uma camada de óxido, inserido entre o *gate* de controle e o canal. Sua célula é tão pequena quanto à de um *antifuse*.

A programação de um transistor EPROM ou EEPROM é mantida mesmo quando a alimentação é desligada, assim, evita-se a necessidade de reprogramar o dispositivo quando a alimentação for religada. Apesar da célula EEPROM ser maior que a célula EPROM, ela oferece a vantagem de poder ser apagada eletricamente. Também, pode ser reprogramada no próprio circuito (ISP).

As conexões programáveis de FPGAs baseadas na tecnologia de programação SRAM são feitas por transistores de passagem ou multiplexadores que são controlados por células SRAMs distribuídas ao longo do *chip* (BROWN; ROSE, 1996; ROSE et al.,1993). A vantagem dessa tecnologia é que ela permite uma rápida reconfiguração no próprio circuito. A principal desvantagem é o tamanho da célula SRAM. Por causa de sua volatilidade, as configurações das células devem ser restabelecidas sempre que o dispositivo for religado, necessitando-se, portanto, de um dispositivo de memória externo. Uma vez que elas podem ser reconfiguradas facilmente, suas configurações podem ser mudadas para consertar *bugs* ou para fazer atualizações. Deste modo, os FPGAs baseados nessa tecnologia fornecem um meio ideal para prototipação (HAUCK, 1998). Também, podem ser usados em situações onde se exigem diversos tipos de configurações.

Outro tipo de comutador programável utilizado em FPGAs é o *antifuse*. Os *antifuses* são dispositivos OTP. Quando programados, têm uma baixa resistência, e quando não, comportam-se como um circuito aberto. Logo, possuem um comportamento oposto ao de um fusível normal. Os *antifuses* possuem resistência e capacitância parasitas menores que às de um transistor de passagem. Logo, reduzem os atrasos RC no roteamento (GREENE et al.,

1993). Apesar das vantagens dos *antifuses* em relação às demais tecnologias, estes necessitam de transistores ocupando largas áreas no dispositivo.

Esses dispositivos se enquadram em duas categorias: *amorphous silicon* e *dieletric*. O primeiro tipo possui uma camada de silício amorfo entre duas camadas de metal, que quando submetida a uma determinada corrente se torna condutiva. Também, possui alguns problemas em relação ao segundo. A Segunda categoria se baseia em um dielétrico colocado entre duas camadas condutoras, *N+ diffusion* e *polysilicon*. Submetido a uma determinada voltagem, esse dielétrico se rompe permitindo a condução do dispositivo. O *antifuse Programmable low-impedance circuit element* (PLICE) da Actel possui um dielétrico constituído por três camadas, chamado *Oxide-nitride-oxide* (ONO). Quando o *antifuse* está desprogramado, esse dielétrico evita a passagem de corrente entre as camadas condutoras. Ao aplicar um pulso de 16V através de seu dielétrico, este se derrete permitindo a formação de um canal condutor entre as camadas de difusão e de silício.

Na Tabela 2.1 é apresentado um resumo das tecnologias de programação discutidas nesta seção. Todos os comutadores programáveis sobreditos ocuparão determinadas áreas no dispositivo e apresentarão resistências e capacitâncias que limitarão o desempenho do dispositivo se comparado a um *gate array* tradicional. Também, os PLDs baseados na tecnologia EEPROM consomem mais energia que os baseados em SRAM ou *Antifuse*.

Tabela 2.1 – Sumário das tecnologias de programação existentes

Nome	Reprogramação	Volatilidade	Tecnologia
Fuse	Não	Não	Bipolar
EPROM	Sim, fora do circuito	Não	UVCMOS
EEPROM	Sim, no circuito	Não	EECMOS
SRAM	Sim, no circuito	Sim	CMOS
Antifuse	Não	Não	CMOS+

Comercialmente, a tecnologia *fuse* é utilizada pela Texas Instruments. A tecnologia EPROM é utilizada pela Xilinx, Altera, AMD, Cypress, Philips, Atmel, ICT e WSI. A tecnologia EEPROM é utilizada pela Altera, AMD e Lattice. A tecnologia SRAM é utilizada pela Xilinx, Altera, Actel, Lucent, Motorola e Dyna*Chip*. A tecnologia *antifuse* é utilizada pela Actel e QuickLogic. A tecnologia FLASH é utilizada pela Cypress.

2.4 CPLDs

Os CPLDs são formados por múltiplos blocos lógicos que se comunicam através de uma estrutura programável de interconexão global. Cada bloco lógico de um CPLD é similar a um SPLD, entretanto, sua estrutura interna é mais sofisticada. Tipicamente, cada bloco lógico contém de 4 a 16 macrocélulas, dependendo de sua arquitetura. Uma macrocélula, nos CPLDs mais modernos, compreende um circuito lógico combinacional formado por soma de produtos mais um flip-flop opcional. Pode ter um grande número de entradas, entretanto, a complexidade de sua função lógica é limitada. As macrocélulas dentro de um bloco lógico geralmente podem ser totalmente conectadas, dependendo da família e fornecedor. Na Figura 2.2 é mostrada uma arquitetura básica de um CPLD (ALTERA, 2007a). Atualmente já existem arquiteturas mais eficientes que a tradicional, como, por exemplo, a arquitetura MAX II de CPLDs de baixo custo e alto desempenho da Altera (ALTERA, 2008b).

Figura 2.2. Arquitetura tradicional de um CPLD (ALTERA, 2007a)

As principais diferenças entre as arquiteturas de CPLDs são: número de termos-produto por macrocélula, se estes podem ser emprestados a outras macrocélulas e se a matriz programável de comutação global pode ser totalmente ou parcialmente conectada. Em algumas arquiteturas, quando o número de termos-produto solicitados excede a capacidade de uma macrocélula, termos adicionais de outras macrocélulas são emprestados, logo, dependendo da arquitetura, estas macrocélulas não podem mais ser utilizadas. Isto aumenta o atraso de propagação interno. Outra diferença nas arquiteturas é o número de conexões dentro da matriz programável de comutação. O número de conexões nesta implicará o quão fácil um projeto se ajustará num dado dispositivo. Para uma matriz de comutação com todas as possibilidades de conexão, um projeto será roteado mesmo com a maioria dos recursos sendo utilizados e após os pinos de I/O já terem sido fixados. Nesse tipo de arquitetura os atrasos internos são fixos e previsíveis. Para as arquiteturas onde a matriz programável é parcialmente conectada, haverá problemas com projetos mais complexos. Também, será difícil realizar uma mudança no projeto mantendo os pinos de I/O fixados, logo, isto implicará numa mudança de *layout* do projeto. Nesse caso, os atrasos internos não serão facilmente previsíveis.

Além da Altera, diversos fabricantes produzem dispositivos categorizados como CPLDs. Entre eles, podem-se citar: Xilinx, Lattice, AMD e ICT. Apesar das diferenças arquiteturais de cada fabricante, tais dispositivos possuem características similares entre si.

2.4.1 Aplicações de CPLDs

Os CPLDs fornecem um caminho de migração natural para projetistas que utilizam SPLDs e que buscam altas densidades para seus projetos. Permitem uma grande integração de circuitos lógicos, reduzindo-se assim o consumo de energia e aumentando a confiabilidade do sistema. Devido ao seu bom desempenho de velocidade pino a pino, geralmente são utilizados

em projetos orientados a controle. As várias entradas de suas macrocélulas os tornam adequados ao projeto de máquinas de estados complexas e de alto desempenho. Também podem ser usados para prototipação de sistemas digitais, controladores de vídeo, controladores de rede, UARTs, controle de *cache*, entre outros. Enfim, com CPLDs é possível realizar a implementação de circuitos complexos que explorem um grande número de portas AND/OR e que não necessitem de um grande número de *flip-flops*. A utilização de CPLDs com recursos de reprogramação no próprio sistema permite reconfigurar o *hardware* sem a necessidade de seu desligamento. Como exemplo, é possível modificar um protocolo para um circuito de comunicação sem o desligamento do mesmo.

2.5 FPGAs

Os FPGAs combinam a arquitetura dos *gate arrays* com a programabilidade dos PLDs. Conforme já abordado anteriormente, um FPGA é formado basicamente por um arranjo de blocos lógicos, circundados por blocos de I/O programáveis, que são conectados por meio de interconexões programáveis. Sua arquitetura é completamente diferente da dos CPLDs e tipicamente oferecem maior capacidade lógica. Nos FPGAs, a lógica a ser implementada usa múltiplos níveis de portas com números de entradas inferiores aos dos SPLDs. Um bloco lógico de um FPGA pode ser tão simples quanto um transistor ou tão complexo quanto um microprocessador. Com FPGAs é possível implementar uma variedade muito grande de funções lógicas combinacionais e sequenciais. A arquitetura de roteamento de um FPGA é formada tipicamente por segmentos metalizados de tamanhos diversificados que podem ser interconectados através de comutadores programáveis eletricamente. A distribuição destes afeta significativamente a densidade e o desempenho obtido por um FPGA (CHOW et al., 1999; ROSE et al., 1993). As tecnologias de programação mais comumente

usadas para implementar comutadores programáveis num FPGA são: SRAM e *Antifuse*. Em ambos os casos, o comutador ocupará uma área maior e apresentará resistência e capacitância parasitas mais altas que as de um típico contato metálico utilizado na *customização* de um MPGA. Quanto à arquitetura, os blocos lógicos de um FPGA se classificam em: blocos de granularidade grossa e blocos de granularidade fina.

Geralmente, as arquiteturas de granularidade grossa são formadas por blocos lógicos grandes contendo dois ou mais *flip-flops* e duas ou mais *look-up tables*. Por exemplo, uma *look-up table* de quatro entradas pode ser imaginada como sendo uma ROM de 16x1. Assim, a tabela verdade de uma função lógica de *K*-entradas é armazenada em uma memória de 2^K x 1. As linhas de endereçamento funcionarão como entradas e a saída fornecerá o valor da função lógica. A vantagem das *look-up tables* (LUTs) é a sua alta funcionalidade. Uma LUT pode implementar qualquer função de *K* entradas, assim, há 2^n funções, onde: $n=2^K$. A desvantagem é que se tornam grandes demais para mais do que cinco entradas. Em (BROWN, 1996), um estudo da complexidade de um bloco lógico de FPGA, assumindo este como uma LUT de *K* entradas, mostrou que para otimizar sua área, o bloco deveria possuir cerca de quatro entradas. Também, para obter um bom desempenho de velocidade, uma LUT com cinco entradas representaria a melhor solução. A desvantagem dos blocos de granularidade grossa consiste no desperdício de seus recursos para implementar funções mais simples.

As arquiteturas de granularidade fina contêm um grande número de blocos lógicos mais simples. Esses blocos geralmente são formados por portas lógicas básicas ou multiplexadores de quatro entradas e um flip-flop. A principal vantagem de utilizar blocos de granularidade fina é que estes são mais bem aproveitados. Sua principal desvantagem é que requerem um número muito grande de segmentos e comutadores programáveis para as interconexões. É preciso mais células lógicas para implementar uma função que seria

implementada com poucas células em uma arquitetura contendo blocos de granularidade grossa. Logo, isto implicará em atrasos e desperdícios de área.

As arquiteturas de FPGAs podem ser classificadas genericamente em quatro categorias disponíveis comercialmente: arranjo simétrico, baseada em linhas, PLD hierárquico e mar de portas. Essas categorias se diferenciam pelo tipo de tecnologia de programação utilizada, pela arquitetura dos blocos lógicos e pela estrutura da arquitetura de roteamento.

2.5.1 Aplicações de FPGAs

Os FPGAs podem ser utilizados em uma faixa muito ampla de aplicações e nas mais distintas áreas. Como exemplos, podem-se destacar: integração de múltiplos SPLDs, controladores de dispositivos, rápida prototipação de projetos para serem posteriormente implementados em *gate arrays*, emulação de grandes sistemas de *hardware*, coprocessadores, criptografia, reconhecimento de padrões, processamento de imagens, processamento de sinais, compressão de dados, arquiteturas paralelas, sistemas de comunicação, entre outros (BOURIDANE et al., 1999; BROWN; ROSE, 1996; CHAN; MOURAD, 1994; DICK; HARRIS, 1998; GUŠTIN, 1999; HAUCK, 1998; SOLDEK; MANTIUK, 1999; THOMAS et al., 1999; VASSÁNYI, 1997; VILLASENOR; SMITH, 1997; DIAZ et al., 2006; GUPTA et al., 2006; CLARK et al., 2006; HENKEL; PARAMESWARAN, 2007; HAUCK; DEHON, 2008).

2.5.1.1 Computação Reconfigurável

Os sistemas reconfiguráveis baseados em FPGAs estão revolucionando o projeto de sistemas computacionais (HAUCK; DEHON, 2008; HENKEL; PARAMESWARAN, 2007).

Sistemas computacionais reconfiguráveis são plataformas cujas arquiteturas podem ser modificadas por *software* em tempo real para se adequarem a uma dada aplicação específica. Para que um certo algoritmo possa obter um máximo rendimento, este deve ser implementado em *hardware*. Um sistema reconfigurável utiliza as partes reprogramáveis dos FPGAs para executar um algoritmo ao invés de compilá-lo para execução numa CPU normal. Esses sistemas tiram vantagem do paralelismo da aplicação enquanto reduzem o *overhead* gerado pelas operações de carga, armazenamento, desvios e decodificação de instruções. De acordo com (TURLEY, 1997), os microprocessadores apresentam diversas limitações e não são adequados para aplicações orientadas a *bits*. Por serem elementos destinados a lidar com processamento genérico, não são tão eficientes quanto os *chips* ASICs para realizar uma tarefa específica. Dessa forma, o *hardware* reconfigurável combina a versatilidade dos microprocessadores com o desempenho do *hardware* dedicado. Um microprocessador é limitado na dimensão espacial e flexível na temporal, assim, suas capacidades nunca mudam, no entanto, a parte do *chip* em uso muda no decorrer do tempo para executar uma diferente função. Sistemas de computação reconfigurável são flexíveis em ambas as dimensões. Logo, sua lógica pode mudar no transcorrer do tempo para se adequar à necessidade em questão. Portanto, sistemas de computação reconfigurável são máquinas que se aproveitam dos aspectos reconfiguráveis dos FPGAs para implementar um dado algoritmo. Todavia, esse novo paradigma requer um modo diferente de pensar para resolver problemas, assim, confunde e mistura os limites de *hardware* e de *software*. Por se tratar de um novo campo, ainda há muitos obstáculos a serem contornados para que essa tecnologia possa ser amplamente adotada, deste modo, há a necessidade do surgimento de novas ferramentas de desenvolvimento que possam fazer a ponte entre a especificação de um projeto e a sua realização em *hardware* para sistemas de reconfiguração dinâmica.

2.6 PROCESSO DE PROJETO PARA PLDS

Embora os primeiros projetos de PLDs eram gerados em grande parte manualmente, a complexidade dos dispositivos atuais exige o emprego de ferramentas CAD. Nesta seção, objetiva-se fornecer uma visão geral sobre os passos e as operações necessárias para a realização de um projeto num PLD. Na Figura 2.3 é possível ver a seqüência típica de operações necessárias para ir desde a descrição de um certo projeto até a programação do *chip*. Os fornecedores comerciais de ferramentas CAD e de companhias de PLDs oferecem ferramentas apropriadas para projetos. Entretanto, as ferramentas *Electronic design automation* (EDAs) tradicionais não suportam todas as etapas do processo de projeto, pois, fornecem-se somente opções de entrada, operações de simulação e uma interface para ferramentas específicas de posicionamento e de roteamento do *chip*.

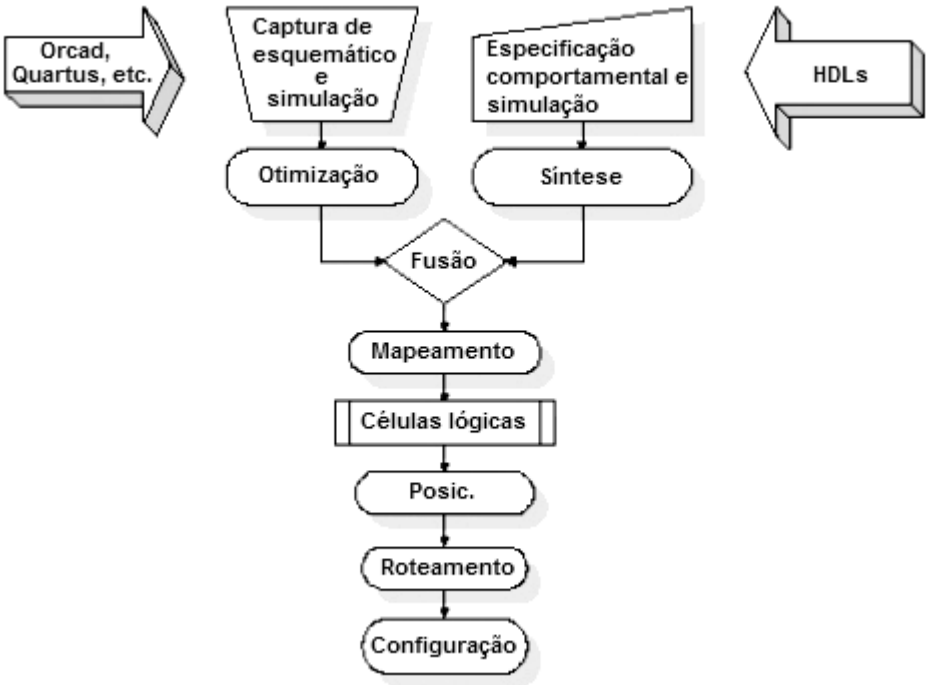


Figura 2.3. Processo de projeto para PLDs

Existe uma variedade muito grande de ferramentas disponíveis para a descrição do processo de entrada de um projeto, conforme ilustrado pelas setas da Figura 2.3. Alguns projetistas preferem usar pacotes de captura de esquemático enquanto outros preferem especificar um projeto através de HDLs ou misturando-se ambos os métodos. Uma alternativa diferente para a entrada de um projeto, consiste na utilização de diagramas de tempo contendo formas de onda desejadas para as entradas e saídas. Também, é possível utilizar *cores* para o processo de entrada de um projeto, reduzindo-se assim o tempo de desenvolvimento e verificação do mesmo. *Cores* são funções pré-definidas especificamente implementadas e verificadas em lógica programável. Independentemente do método utilizado para a entrada do projeto, o próximo passo consiste em converter o projeto em um formato padrão que seja processado por uma ferramenta de otimização lógica. O objetivo da otimização lógica é minimizar as expressões *booleanas* e eliminar redundâncias, portanto, minimizando-se assim a área do circuito final. Também, nesta etapa, realiza-se uma verificação do projeto. Em seguida, o *software* mapeia as portas lógicas básicas em blocos lógicos do PLD de destino. Após a lógica ter sido particionada em vários blocos, o *software* procura pela melhor localização para posicionar os blocos lógicos entre todas as possibilidades existentes. O objetivo principal é reduzir a quantidade de recursos de roteamento requeridos e maximizar o desempenho do sistema. Após o posicionamento, um roteador aloca os segmentos e comutadores disponíveis para as interconexões dos blocos lógicos. Quando os processos de posicionamento e roteamento são finalizados, o *software* gera um arquivo de programação binário necessário para a configuração do dispositivo. A verificação de um projeto ocorre em vários níveis e passos durante as etapas de sua implementação. É possível realizar uma simulação funcional em conjunto com o processo de entrada, mas antes do posicionamento e roteamento, para verificar o funcionamento lógico do projeto. Depois de completados os processos de posicionamento e roteamento, valores exatos de tempos podem ser utilizados

para determinar o desempenho do *chip*. Também é possível obter resultados de simulações funcionais mais precisos, pois são incorporadas informações de atrasos ao arquivo *netlist* gerado. Alternativamente, utilizando-se técnicas de verificação no próprio sistema, o projeto pode ser analisado em alta velocidade.

2.7 ESTADO DA ARTE

O mercado de FPGAs tem se estabelecido em sua maior parte em um estado onde há dois fabricantes principais de FPGAs: Xilinx e Altera. Também, há um número de outras empresas que se distinguem por oferecerem chips com características únicas (WIKIPEDIA, 2008). Assim, nesta seção será realizada uma breve comparação entre os dispositivos mais recentes disponibilizados pela Xilinx e Altera. Cada companhia possui produtos similares, onde as principais diferenças são caracterizadas por componentes adicionais cujo objetivo é otimizar o *chip* para diferentes tipos de aplicações.

A Xilinx possui atualmente algumas famílias principais de produtos envolvendo FPGAs. Algumas dessas famílias serão discutidas a seguir. A família Virtex II Pro possui em um de seus membros básicos um *core* PPC405 e 3008 *slices* lógicos com 6768 elementos lógicos. Esse membro também contém um máximo de 94 Kbits de RAM distribuída, 28 multiplicadores de 18x18 bits, 504 Kbits de blocos de RAM e 4 blocos de *transceiver* RocketIO. A linha Virtex Pro II opera a 1,5 Volt internamente e possui um *clock* máximo de aproximadamente 400 MHz (XILINX, 2007a). Os *transceivers* podem ter taxas de até 6,25 Gb/s. As famílias mais novas, Virtex-4 e Virtex-5, apresentam blocos *Ethernet* MAC e PCI *Express* embutidos. A Xilinx também oferece FPGAs sem *core* de processador embutido, com a vantagem da redução de custos para muitas aplicações, incluindo *Digital Signal Processing* (DSP). Entre as principais plataformas, destacam-se: Spartan-3A DSP, Spartan-

3AN, Spartan-3A, Spartan-3E e Spartan-3. O menor dispositivo da família SPARTAN-3 possui 50000 portas lógicas, 1728 células lógicas, 192 CLBs e 72K de blocos de RAM. Também existem quatro blocos multiplicadores dedicados. A família SPARTAN-3 também possui capacidade de I/O de 622 Mbit/s. A arquitetura dessa família é geral e é adequada para diferentes tipos de aplicações, além de ser sensível a custos (XILINX, 2007b). As Tabelas 2.2 e 2.3 apresentam um resumo das principais características das famílias Virtex II Pro e SPARTAN-3 de FPGAs da Xilinx, respectivamente.

Tabela 2.2 – Membros da Família Virtex II Pro da Xilinx

Dispositivo	*Transceiver*	Processadores PowerPC	Células Lógicas	RAM Distribuída (Kb)	Multiplicador 18x18 bits
XC2VP2	4	0	3168	44	12
XC2VP4	4	1	6768	94	28
XC2VP7	8	1	11088	154	44
XC2VP20	8	2	20880	290	88
XC2VPX20	8	1	22032	306	88
XC2VP30	8	2	30816	428	136
XC2VP40	0, 8 ou 12	2	43632	606	192
XC2VP50	0 ou 16	2	53136	738	232
XC2VP70	16 ou 20	2	74448	1,034	328
XC2VPX70	20	2	74448	1,034	308
XC2VP100	0 ou 20	2	99216	1,378	444

Tabela 2.3 – Sumário de alguns atributos básicos da Família SPARTAN-3 da Xilinx

Dispositivo	Portas Lógicas	Células Lógicas	RAM distribuída (bits)	Multiplicadores
XC3S50	50K	1728	12K	4
XC3S200	200K	4320	30K	12
XC3S400	400K	8064	56K	16
XC3S1000	1M	17280	120K	24
XC3S1500	1.5M	29952	208K	32
XC3S2000	2M	46080	320K	40
XC3S4000	4M	62208	432K	96
XC3S5000	5M	74880	520K	104

A Altera também possui uma linha *standalone* de FPGAs (Família Stratix III) de alta-performance e com *tranceiver* embutido (Stratix II GX), podendo atingir uma taxa de transferência de dados de até 6,375 Gb/s. Também oferece uma opção de baixo custo através de sua família Cyclone III. O *chip* mais avançado da linha Stratix III possui 337.500

elementos lógicos, aproximadamente 20,5 Mbits de RAM, 112 blocos de DSP, 576 multiplicadores de 18x18 bits e um total de 1104 pinos de I/O (ALTERA, 2007b). Os blocos de DSP contêm um grande número de multiplicadores e podem ser utilizados em aplicações envolvendo cálculos baseados na transformada rápida de Fourier, transformada cosseno discreta, filtros de resposta finita e infinita, entre outras aplicações. Na Tabela 2.4, apresenta-se um sumário dessa família. Atualmente, a Altera já está disponibilizando sua família de maior densidade lógica, a Stratix IV (ALTERA, 2008c).

Tabela 2.4 – Características da Família Stratix III de FPGAs da Altera

Dispositivo	Elementos Lógicos	RAM Kbits	Blocos de DSP	Multiplicadores de 18x18 bits	Pinos de I/O
EP3SL50	47500	1836	27	216	480
EP3SL70	67500	2214	36	288	480
EP3SL110	106500	4203	36	288	736
EP3SL150	142000	5499	48	384	736
EP3SL200	198900	7668	72	576	864
EP3SL340	338000	16272	72	576	1104

2.8 CONSIDERAÇÕES FINAIS

Inicialmente os PLDs eram usados somente para prototipação de projetos digitais. Com o amadurecimento desses dispositivos e das ferramentas CAD, os PLDs já são utilizados em diversas aplicações complexas. Atualmente já se dispõem de dispositivos lógicos programáveis com dezenas de milhares de elementos lógicos, com blocos de DSP e de memória, blocos *Ethernet* MAC e PCI *Express*, entre outros recursos disponíveis embutidos no *chip*. Assim, levando-se também em consideração o baixo custo desses dispositivos, a capacidade de reprogramação em campo pelo usuário e o curto ciclo de projeto em relação a outras metodologias tradicionais, pode-se esperar que a lógica programável se consolide como a forma dominante de projeto e implementação de sistemas digitais.

3 MORFOLOGIA MATEMÁTICA

3.1 CONSIDERAÇÕES INICIAIS

A palavra “morfologia” é composta pelas palavras gregas *morphê* (forma) e *logos* (ciência) (FACON, 1996), significando o estudo da forma ou estrutura. Segundo (GONZALEZ; WOODS, 1992), em biologia, a morfologia denota uma área que lida com a forma e a estrutura de animais e plantas. Assim, a forma de uma folha pode ser usada para identificar uma planta. Conforme já abordado, os primeiros trabalhos na área de morfologia matemática se devem a Minkowsky e Hadwiger. A continuação e reformulação desses trabalhos se dariam nos anos 60 através das pesquisas conjuntas de G. Matheron e J. Serra, onde através dessas noções foram criados os operadores básicos de dilatação e erosão morfológicos (DOUGHERTY, 1992a; SERRA, 1982; WEEKS Jr., 1996; ZAMORA, 2002). A morfologia matemática é baseada em conceitos geométricos e na teoria da ordem (MATHERON, 1975; SERRA, 1988; SOILLE, 1999 SONKA et al., 1993). Na teoria clássica de morfologia (morfologia para imagens binárias), uma imagem binária é formalizada como um subconjunto de um grupo “abeliano”, onde seus operadores elementares são a erosão e a dilatação (KIM, 1997). Filtros digitais mais complexos podem ser projetados utilizando esses dois operadores, juntamente com os operadores de complemento, união e intersecção de conjuntos. Esta técnica representa uma ferramenta valiosa para tarefas de visão computacional envolvendo imagens binárias e em níveis de cinza, tais como extração de componentes de imagens que sejam úteis na representação e na descrição da forma de uma determinada região, como, por exemplo, fronteiras, esqueletos e o fecho convexo. Também pode ser utilizada para pré e pós-processamento, como, por exemplo, filtragem morfológica, afinamento e poda. Pode ser aplicada em diversas áreas, entre elas, destacam-se: biologia,

metalografia, medicina, visão robótica, controle de qualidade e reconhecimento de padrões, para citar somente algumas.

A teoria da morfologia matemática foi originalmente desenvolvida como uma técnica experimental para análise de textura e foi mais tarde estendida para imagens em níveis de cinza (HEIJMANS, 1994; SARTOR; WEEKS, 2001). Suas aplicações iniciais foram para análise de imagens geológicas e biomédicas (MARAGOS, 2005). Nos anos 80, extensões da morfologia clássica para outros campos foram realizadas em várias direções por vários grupos de pesquisa ao redor do mundo, incluindo-se: processamento multi-escala de imagens, análise estatística de imagens, projeto de filtros morfológicos ótimos, entre outras. Mais recentemente, a teoria foi generalizada para reticulados completos (SERRA, 1993). De acordo com a teoria de reticulados, operadores morfológicos podem ser construídos para qualquer conjunto que possa ser representado por um reticulado.

A aplicação dos conceitos de morfologia matemática a problemas de visão por computador continua sendo um tema de investigação em nível mundial. A linguagem da morfologia matemática é a teoria de conjuntos (reticulados completos). Os conjuntos em morfologia matemática representam as formas dos objetos numa imagem. Como exemplo, podem-se considerar todos os *pixel*s pretos de uma dada imagem binária como sendo uma descrição completa dessa imagem. Tais conjuntos são membros do espaço bidimensional de números inteiros Z^2. Cada elemento do conjunto é formado por um vetor bidimensional cujas coordenadas são as coordenadas (x,y) dos *pixel*s pretos (por convenção) da imagem. Na Figura 3.1, pode-se ver um exemplo (COSTA; CÉSAR Jr., 2001; HARALICK et al.,1987; MARQUES FILHO; VIEIRA NETO, 1999; ORTIZ et al., 2002).

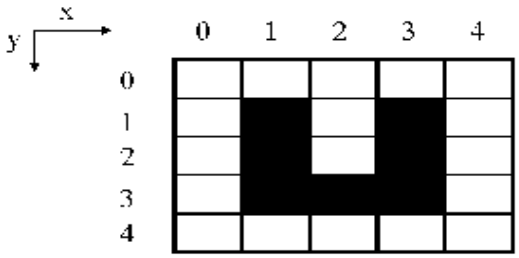


Figura 3.1. Descrição de uma imagem binária através do conjunto formado por seu sistema de coordenadas. O objeto em forma de U pode ser representado completamente pelo conjunto A={(1,1),(1,2),(1,3),(2,3),(3,3),(3,2),(3,1)}

Imagens digitais em níveis de cinza podem ser representadas por conjuntos cujos componentes estejam em Z^3. Nesse caso, dois componentes de cada elemento do conjunto se referem às coordenadas do *pixel* enquanto o terceiro corresponde ao valor discreto de intensidade. Conjuntos em espaços de maiores dimensões podem conter outros atributos de imagens, como, por exemplo, cor e componentes que variem com o tempo.

Conforme já supracitado, imagens coloridas carregam uma grande quantidade de informações, assim, essas informações podem ser úteis na simplificação do processo de análise de imagens, como, por exemplo, na identificação de objetos e extração de características baseadas em padrões coloridos (LI; LI, 2004). Também, sabe-se que as cores representam um papel importante na percepção visual humana e imagens coloridas estão sendo cada vez mais usadas em tarefas de visão computacional, uma vez que sensores de vídeo coloridos estão sendo amplamente disponíveis atualmente. A extensão da morfologia matemática para imagens coloridas não é uma tarefa trivial (COMER; DELP, 1999). A morfologia matemática é baseada na teoria de reticulados, na qual a noção de ordem é muito importante. Em imagens binárias e em níveis de cinza, os *pixel*s são ordenados por seus valores de intensidade, já em imagens coloridas, não há uma ordem natural para os vetores representando os *pixel*s dessas imagens, assim, a extensão dos operadores básicos de

morfologia matemática (dilatação e erosão) de imagens em níveis de cinza para imagens coloridas requer um estudo específico de ordem em dados multivariados (CHANUSSOT; LAMBERT,1998).

Nas próximas seções, são apresentados os principais operadores morfológicos para imagens binárias, em níveis de cinza e coloridas.

3.2 OPERADORES BÁSICOS DE MORFOLOGIA MATEMÁTICA

Nesta seção, são apresentados os principais operadores de morfologia matemática para imagens binárias e em níveis de cinza.

3.2.1 Dilatação

Sejam A e B conjuntos de Z^2 e $\varnothing$ um conjunto vazio, logo, a dilatação de A (imagem original) por B (elemento estruturante) é definida como (GONZALEZ; WOODS, 1992):

$$A \oplus B = \{x \mid (\hat{B})_x \cap A \neq \varnothing\} \qquad (3.1)$$

Assim, o processo de dilatação começa com a obtenção da reflexão de B em torno de sua origem, seguido da translação dessa reflexão por *x*. A dilatação de A por B é então, o conjunto de todos os deslocamentos *x* tais que a reflexão de B e A se sobreponham em pelo menos um elemento não nulo. A equação (3.1) pode ser reescrita como (GONZALEZ; WOODS, 1992):

$$A \oplus B = \{x \mid [(\hat{B})_x \cap A] \subseteq A\} \qquad (3.2)$$

Na Figura 3.2, tem-se um exemplo ilustrativo.

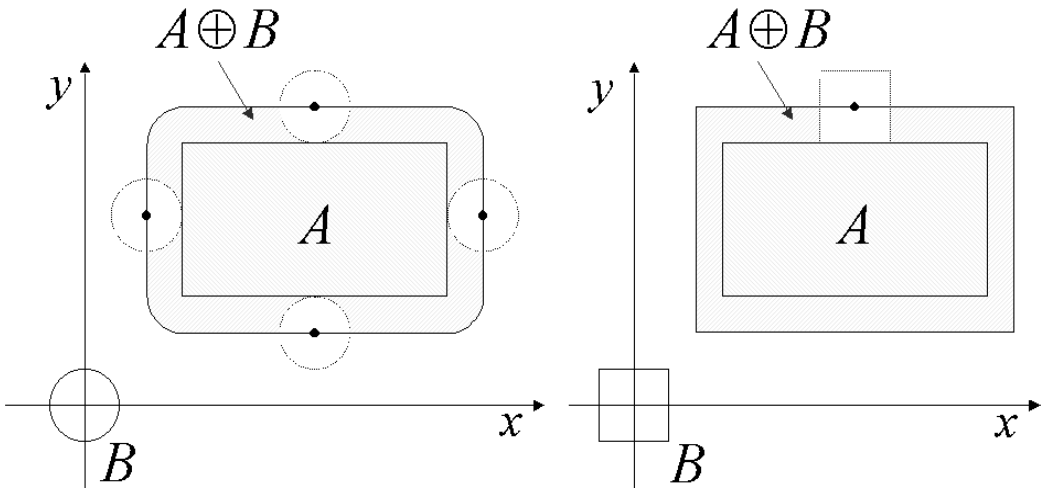


Figura 3.2. Ilustração do processo de dilatação do conjunto A (retângulo) utilizando dois elementos estruturantes de formas geométricas diferentes

3.2.1.1 Adição de Minkowski

A adição de Minkowski de dois conjuntos, A e B, de Z^2, é definida como (DOUGHERTY, 1992):

$$A \oplus B = \bigcup_{b \in B} (A)_b \qquad (3.3)$$

Ou seja, na equação (3.3), pode-se observar que em vez de se transladar o elemento estruturante, deve-se transladar a imagem em relação às posições permitidas do elemento estruturante. Os deslocamentos são realizados em relação ao ponto central de B. Ao contrário das definições anteriores de dilatação, neste caso não há a necessidade de verificação, obtendo-se assim um melhor desempenho computacional. Uma vez que a dilatação é uma transformação comutativa, tem-se (DOUGHERTY, 1992):

$$A \oplus B = \cup\{B + a \mid a \in A\} \qquad (3.4)$$

Assim sendo, através da equação (3.4), é possível efetuar a união de todos os deslocamentos de B em relação aos elementos válidos de A.

Exemplo 3.1: Na Figura 3.3, tem-se um exemplo ilustrativo da dilatação do conjunto A dado por um elemento estruturante B unidimensional utilizando a equação (3.4).

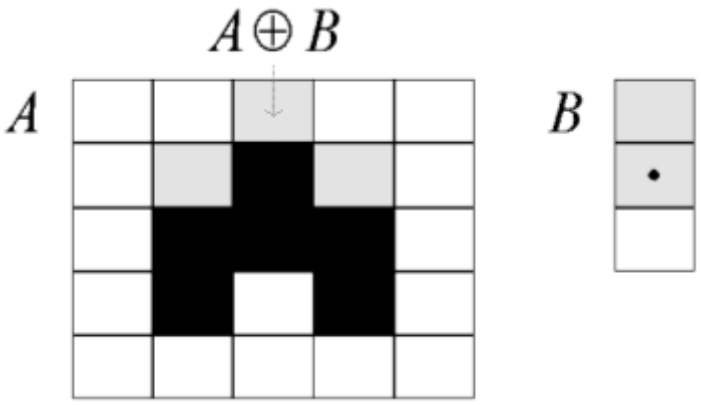


Figura 3.3. Ilustração do processo de dilatação utilizando a equação (3.4). Em cinza, por questões didáticas, têm-se os *pixel*s adicionados após a operação de dilatação

Outras definições de dilatação podem ser encontradas em (FACON, 1996; SERRA, 1986), entretanto, todas são equivalentes.

3.2.2 Erosão

Sejam A e B conjuntos de Z^2, logo, a erosão de A por B é definida como (GONZALEZ; WOODS, 1992):

$$A \Theta B = \{x \mid (B)_x \subseteq A\} \qquad (3.5)$$

Deste modo, a erosão de A por B é o conjunto de todos os pontos *x*, tais que B, quando transladado de *x*, fique contido em A. Como no caso da dilatação, a equação (3.5) não é a única definição de erosão. Na Figura 3.4, ilustra-se a erosão de um retângulo A por um elemento estruturante circular.

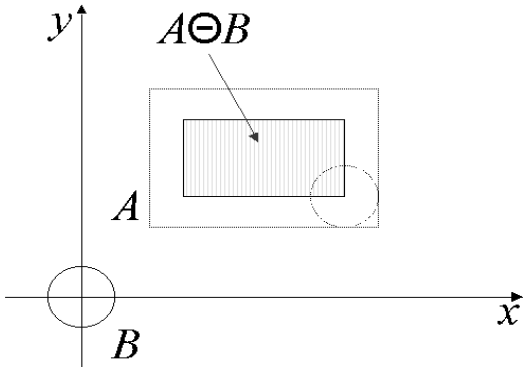


Figura 3.4. Processo de erosão do conjunto A (retângulo) pelo elemento estruturante B (círculo). A área hachurada representa o conjunto erodido

3.2.2.1 Subtração de Minkowski

A subtração de Minkowski de dois conjuntos, A e B, de Z^2, é definida como (DOUGHERTY, 1992):

$$A\Theta(-B) = \cap\{A + b \mid b \in B\} \qquad (3.6)$$

Onde: -B={-*b*|*b*∈B} é a rotação de B de 180° em relação a sua origem. A subtração de Minkowski é a erosão pelo elemento estruturante –B. Portanto, da equação (3.6), vem (DOUGHERTY, 1992):

$$A\Theta B = \cap\{A - b \mid b \in B\} \qquad (3.7)$$

Assim como na dilatação dada pela equação (3.3), também na erosão dada pela equação (3.7) o processo é executado sem verificação, obtendo-se assim um melhor desempenho computacional.

3.2.3 Efeitos da Dilatação e da Erosão

A erosão modifica o conjunto original e este fica menor em todos os casos. Faz desaparecer conjuntos inferiores ao elemento estruturante, aumenta furos internos e separa conjuntos.

A dilatação também modifica o conjunto original e este fica maior em todos os casos. Preenche furos inferiores ao elemento estruturante e conecta conjuntos separados.

3.2.4 Abertura e Fechamento

Conforme visto anteriormente, a dilatação expande uma imagem, enquanto a erosão a reduz. A abertura, em geral, suaviza o contorno de uma imagem, quebra istmos estreitos e elimina proeminências delgadas. O fechamento funde as quebras em golfos estreitos, elimina pequenos orifícios e preenche descontinuidades no contorno.

A abertura de um conjunto A por um elemento estruturante B, é definida como (GONZALEZ; WOODS, 1992):

$$A \circ B = (A \Theta B) \oplus B \qquad (3.8)$$

O fechamento de A por B é definido como (GONZALEZ; WOODS, 1992):

$$A \bullet B = (A \oplus B) \Theta B \qquad (3.9)$$

A teoria da morfologia original para imagens binárias pode ser encontrada em (MATHERON, 1975; SERRA, 1982). Mais exemplos de aplicação para morfologia clássica, bem como de suas propriedades algébricas importantes, podem ser encontrados em (GIARDINA; DOUGHERTY 1988; DOUGHERTY; GIARDINA, 1987; GONZALEZ; WOODS, 1992).

3.2.5 Morfologia Matemática para Imagens em Níveis de Cinza

Serão apresentadas nesta subseção as operações de dilatação, erosão, abertura e fechamento estendidas para imagens em níveis de cinza. Ao longo da discussão, *f(x,y)* corresponderá a uma imagem (função) digital de entrada e *b(x,y)* corresponderá ao elemento estruturante, ou seja, uma subimagem digital. Assume-se também que essas funções sejam discretas, isto é, se Z denotar o conjunto de números inteiros, assume-se que as coordenadas *(x,y)* pertençam a ZxZ e *f* e *b* sejam funções que atribuam um nível de cinza (um número real R) a cada par distinto de *(x,y)*. Se os níveis de cinza também forem inteiros, Z deve substituir R.

3.2.5.1 Dilatação

A dilatação em níveis de cinza de *f* por *b*, denotada, f $\oplus$ b, é definida como (GONZALEZ; WOODS, 1992):

$$(f \oplus b)(s,t) = \max\{f(s-x,t-y) + b(x,y) \mid (s-x),(t-y) \in D_f ; (x,y) \in D_b\} \quad (3.10)$$

Na equação 3.10, D_f e D_b são os domínios de *f* e *b*, respectivamente. Como antes, *b* é o elemento estruturante do processo morfológico. Aqui, diferentemente do caso binário, *f* é deslocada, no lugar do elemento estruturante *b*. Também, este processo pode ser realizado por *b* sendo refletido em relação a sua origem e deslocado pela imagem *f*. Visto que a dilatação se baseia na escolha do valor máximo de *f+b* em uma vizinhança definida pela forma do elemento estruturante, o efeito geral da dilatação de uma imagem em níveis de cinza se desdobra em dois:

- se todos os valores do elemento estruturante forem positivos, a imagem resultante tende a ser mais clara que a imagem de entrada.
- detalhes escuros são reduzidos ou eliminados, dependendo de como seus valores e formas estejam relacionados ao elemento estruturante utilizado.

3.2.5.2 Erosão

A erosão em níveis de cinza, denotada por f Θ b, é definida como (GONZALEZ; WOODS, 1992):

$$(f \Theta b)(s,t) = \min\{f(s+x,t+y) - b(x,y) \mid (s+x),(t+y) \in D_f ; (x,y) \in D_b\} \quad (3.11)$$

Na equação 3.11, D_f e D_b são os domínios de *f* e *b*, respectivamente. Aqui também, *f* é deslocada, e não o elemento estruturante. Também, este processo pode ser realizado por *b* sendo deslocado pela imagem *f*. A erosão se baseia na escolha do valor mínimo de *f-b* em uma

vizinhança definida pela forma do elemento estruturante. Existem dois efeitos gerais decorrentes da erosão de uma imagem:

- se todos os valores do elemento estruturante forem positivos, a imagem de saída tende a ser mais escura que a imagem de entrada.
- detalhes claros na imagem de entrada são reduzidos, sendo que o grau dessa redução é determinado pelos valores dos níveis de cinza desses detalhes e pela forma e valores de amplitude do elemento estruturante.

3.2.5.3 Abertura e Fechamento

As expressões de abertura e fechamento de imagens em níveis de cinza possuem a mesma forma que às análogas do caso binário. A abertura de uma imagem *f* por uma subimagem *b* (elemento estruturante), denotada por $f \circ b$, é dada por (GONZALEZ; WOODS, 1992):

$$f \circ b = (f \ominus b) \oplus b \qquad (3.12)$$

Como no caso binário, a abertura é simplesmente a erosão de *f* por *b* seguida da dilatação do resultado por *b*. De maneira similar, o fechamento de *f* por *b*, denotado por $f \bullet b$, é dado por (GONZALEZ; WOODS, 1992):

$$f \bullet b = (f \oplus b) \ominus b \qquad (3.13)$$

Na prática, as operações de abertura são usualmente aplicadas na remoção de pequenos detalhes claros (em relação ao tamanho do elemento estruturante), enquanto não alteram os níveis de cinza em geral nem os grandes elementos claros. A erosão inicial não só remove os pequenos detalhes como também escurece a imagem. A dilatação subseqüente novamente aumenta a claridade da imagem sem reintroduzir os detalhes removidos pela erosão. O fechamento é geralmente usado na remoção de detalhes escuros em uma imagem, enquanto deixa os elementos claros relativamente inalterados. A dilatação inicial remove os detalhes escuros e clareia a imagem, enquanto a erosão subseqüente escurece a imagem sem reintroduzir os detalhes removidos pela dilatação. A teoria de morfologia para imagens em níveis de cinza, juntamente com algumas aplicações, pode ser encontrada em (BOOMGAARD; SMEULDERS, 1994; BREEN; JONES, 1996; DOUGHERTY; SINHA, 1995a; DOUGHERTY; SINHA, 1995b; DOUGHERTY; ASTOLA, 1994; HARALICK et al., 1987; HEIJMANS, 1991; LI; CHEN, 1991; SONG; DELP, 1990; STERNBERG, 1986; YOUNG at al., 1998). Alguns algoritmos rápidos para morfologia em níveis de cinza podem ser encontrados em (BLEU et al., 1992a; BLEU et al., 1992b).

3.3 MORFOLOGIA PARA RETICULADOS

A morfologia matemática pode ser descrita geometricamente em termos de seus operadores para imagens binárias e em níveis de cinza, conforme visto nas seções anteriores. Tal descrição dependerá do elemento estruturante a ser adotado na operação morfológica em questão. Essa morfologia estrutural é altamente útil na análise e processamento de imagens binárias e em níveis de cinza (PETERS, 1997). Entretanto, os operadores clássicos não podem ser aplicados diretamente a imagens coloridas. Contudo, a morfologia matemática pode ser descrita algebricamente por meio de operadores sobre reticulados completos (BIRKHOFF,

1984). Conforme já citado anteriormente, a teoria de reticulados foi utilizada para formalizar a morfologia para imagens digitais não necessariamente binárias ou em níveis de cinza. Do ponto de vista formal, a morfologia matemática estuda mapeamentos entre reticulados completos. Essa teoria detalhada, juntamente com suas descrições formais de outros operadores mais complexos entre reticulados, pode ser encontrada em (BIRKHOFF; BARTEE, 1970; HEIJMANS, 1994; HEIJMANS; RONSE, 1990; MACLANE; BIRKHOFF, 1967; OVERTURF et al., 1995; RONSE; HEIJMANS, 1991; RONSE, 1996; RONSE, 1990; BANON; BARRERA, 1998; SERRA, 1988; SERRA; VINCENT, 1992).

3.3.1 Reticulado Completo

De forma sucinta, uma relação binária *R* aplicada a um determinado conjunto, toma dois elementos desse conjunto como entrada do processo e retorna *verdadeiro* ou *falso* como saída (JOVANOVIC, 2005). Relações de ordem parcial pertencem à classe de relações binárias.

Definição 3.3.1 (*Poset*): Dado um conjunto não vazio L, uma relação binária < em L é chamada de ordem parcial se as propriedades seguintes forem satisfeitas para $x,y,z \in L$.

a-) $x < x$ (reflexividade);

b-) $x < y$ e $y < x$ implica $x = y$ (anti-simetria);

c-) $x < y$ e $y < z$ implica $x < z$ (transitividade);

Assim, o conjunto L com a ordem parcial < é chamado de conjunto parcialmente ordenado, *partially ordered set* (*poset*), e é denotado por (L, <). É importante perceber que o símbolo < representa uma relação de ordem e não uma desigualdade.

Definição 3.3.2 (Diagrama de Hasse): De acordo com (KIM, 1997; HEIJMANS, 1994), um *poset* L pode ser representado graficamente como segue: se $a < b$, e não existir nenhum elemento $x \in L$ tal que $a < x < b$, então se coloca b acima de a e traça-se uma linha que conecta esses dois elementos. O diagrama resultante é denominado Diagrama de Hasse. A Figura 3.5 ilustra alguns exemplos.

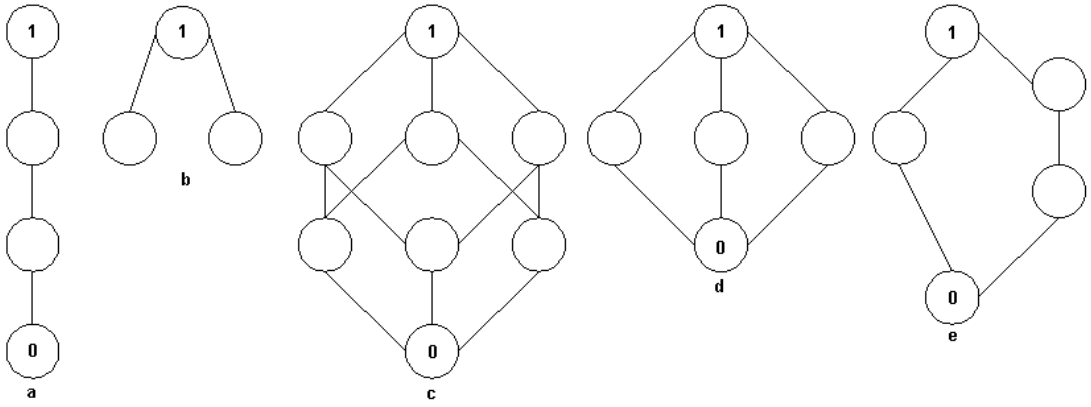


Figura 3.5. Exemplos de Diagramas de Hasse para representação de *posets* (KIM, 1997)

Definição 3.3.3 (Cadeia): Um *poset* (L,<) é totalmente ordenado se $x < y$ ou $y < x$, para todo $x,y \in L$. Um *poset* definido desta maneira é também chamado de *cadeia*. Tal *cadeia* (L,<) será limitada se o conjunto L for finito. Na Figura 3.5a é apresentado um exemplo de *cadeia*.

Definição 3.3.4 (Princípio da dualidade): Se < define uma ordem parcial em L, então, a relação binária > também definirá uma ordem parcial em L chamada de *ordem parcial dual*. Se (L,<) é um *poset*, então, (L,>) será também um *poset* chamado de *poset dual*. Assim, para toda definição, propriedade e proposição em (L,<), existirá uma correspondente dual em (L,>).

Definição 3.3.5 (Mínimo e Máximo): Dado um *poset* L e um subconjunto $M \subset L$, um elemento $a \in M$ é chamado de mínimo de M se $a < x$, para todo $x \in M$. De maneira análoga, $b \in M$ será máximo de M se $b > x$, para todo $x \in M$.

Definição 3.3.6 (Limitantes inferior e superior): Dado um *poset* L e um subconjunto $M \subset L$, um elemento $a \in L$ é chamado de limitante inferior de M se $a < x$, para todo $x \in M$. Assim, $b \in L$ será limitante superior de M se $b > x$, para todo $x \in M$.

Definição 3.3.7 (Ínfimo e Supremo): Dado um *poset* L e um subconjunto $M \subset L$, se o conjunto de limitantes inferiores de M possui um máximo, este será chamado de ínfimo e representado por $\wedge M$. De forma análoga, se o conjunto de limitantes superiores de M possui um mínimo, este será chamado de supremo e representado por $\vee M$.

Definição 3.3.8 (Reticulado): Um *poset* L é chamado de *reticulado* se todo subconjunto finito de L possui um ínfimo e um supremo. Um *poset* L será chamado de *reticulado completo* se todo subconjunto de L possui um ínfimo e um supremo.

Na Figura 3.5b é apresentado um exemplo de *poset* que não é reticulado. Os demais *posets* da Figura são exemplos de reticulados completos e possuem um elemento mínimo e um elemento máximo, 0 e 1, respectivamente, dado que são conjuntos finitos.

3.3.2 Operadores Básicos entre Reticulados

Um mapeamento Ψ entre dois reticulados L_1 e L_2 é chamado de operador (CANDEIAS, 1997).

Definição 3.3.9 (Transformação Crescente e Decrescente): Sejam $(L_1,<)$ e $(L_2,<)$ dois conjuntos parcialmente ordenados. A transformação Ψ de L_1 em L_2 será crescente se $a < b \Rightarrow \Psi(a) < \Psi(b)$ e será decrescente se $a < b \Rightarrow \Psi(b) < \Psi(a)$. Também, se $(L_1,<)$ e $(L_2,<)$ são dois reticulados, então: Ψ é crescente $\Leftrightarrow \Psi(a) \vee \Psi(b) < \Psi(a \vee b)$ e $\Psi(a \wedge b) < \Psi(a) \wedge \Psi(b)$; Ψ é decrescente $\Leftrightarrow \Psi(a) \wedge \Psi(b)$ e $\Psi(a) \vee \Psi(b) < \Psi(a \wedge b)$.

Definição 3.3.10 (Dilatação): Se L_1 e L_2 são dois reticulados finitos, Ψ de L_1 em L_2 será uma dilatação $(L_1,<)$ em $(L_2,<)$ se e somente se $\Psi(a \vee b) = \Psi(a) \vee \Psi(b)$ e $\Psi(0)=0$, onde $(a,b \in L_1)$.

Definição 3.3.11 (Erosão): Se L_1 e L_2 são dois reticulados finitos, Ψ de L_1 em L_2 será uma erosão $(L_1,<)$ em $(L_2,<)$ se e somente se $\Psi(a \wedge b) = \Psi(a) \wedge \Psi(b)$ e $\Psi(1)=1$, onde $(a,b \in L_1)$.

Definição 3.3.12 (Anti-Dilatação): Se L_1 e L_2 são dois reticulados finitos, Ψ de L_1 em L_2 será uma anti-dilatação $(L_1,<)$ em $(L_2,<)$ se e somente se $\Psi(a \vee b) = \Psi(a) \wedge \Psi(b)$ e $\Psi(0)=1$, onde $(a,b \in L_1)$.

Definição 3.3.13 (Anti-Erosão): Se L_1 e L_2 são dois reticulados finitos, Ψ de L_1 em L_2 será uma anti-erosão $(L_1,<)$ em $(L_2,<)$ se e somente se $\Psi(a \wedge b) = \Psi(a) \vee \Psi(b)$ e $\Psi(1)=0$, onde $(a,b \in L_1)$.

3.3.3 Métodos de Ordenação Vetorial

Conforme já mencionado, imagens coloridas possuem uma grande quantidade de informações e essas podem ser úteis no processo de análise de imagens. Também, a extensão dos operadores morfológicos básicos para imagens coloridas não é uma tarefa fácil na prática,

considerando-se que os vetores representando os *pixels* nessas imagens não possuem uma ordenação natural. Assim, nas próximas subseções serão apresentadas as principais técnicas existentes para ordenação de dados multivariados.

3.3.3.1 Processamento Morfológico de Imagens Coloridas

Imagens digitais são tipicamente representadas como composições aditivas de cores que utilizam as três componentes espectrais primárias: vermelho, verde e azul (RGB). Na Figura 3.6, ilustra-se a relação existente entre as componentes RGB de um tubo de raios catódicos colorido encontrado em monitores de vídeo. Na Figura é apresentado o modelo de cores RGB, que é definido pelos níveis de cinza de cada uma das três cores primárias.

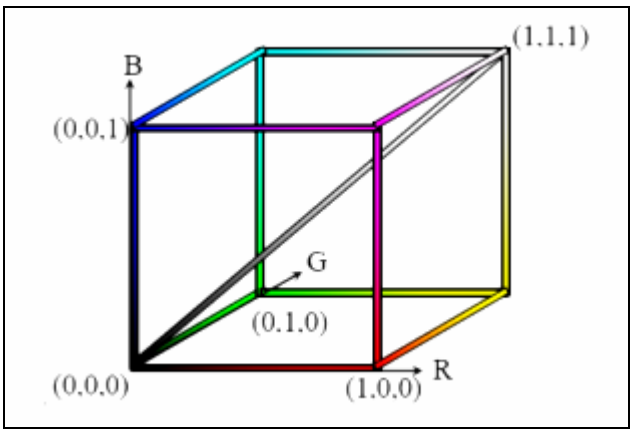


Figura 3.6. Cubo de cores RGB (LIMA et al., 2005)

Para um sistema de vídeo RGB com dados de 8 bits de resolução, a faixa de possíveis valores de níveis de cinza para cada componente é de 0 a 255. Logo, há 256^3 ou 16777216 possibilidades de combinações de vermelho, verde e azul que podem ser mostradas nesse dispositivo. Cada *pixel* é representado por uma coordenada tridimensional dentro do cubo de cores. A linha que vai da origem até o canto oposto do cubo é conhecida como linha de cinza

ou eixo acromático, uma vez que os valores digitais sobre essa linha possuem componentes iguais de vermelho, verde e azul. Monitores RGB são usados extensivamente em processamento digital para mostrar composições de imagens em cor normal, falsa cor e pseudocolorização. Uma composição colorida pode ter seu realce melhorado num monitor RGB através da manipulação do contraste individual de cada uma de suas componentes primárias. Uma alternativa para descrever cores através de componentes RGB é através do uso do sistema *hue-saturation-intensity* (HSI), onde a intensidade se relaciona ao brilho total de uma cor, a matiz a um comprimento de onda referente a essa cor e a saturação ao conceito de pureza da cor. Transformar componentes RGB em componentes HSI antes do processamento pode fornecer mais controle sobre o realce de imagens coloridas. Na Figura 3.7 é apresentado um exemplo do referido modelo. O modelo de cores HSI é muito útil em processamento digital de imagens, pois, a componente de intensidade é separada da informação de cromaticidade e as componentes de matiz e saturação estão relacionadas ao processo pelo qual os seres humanos percebem as cores.

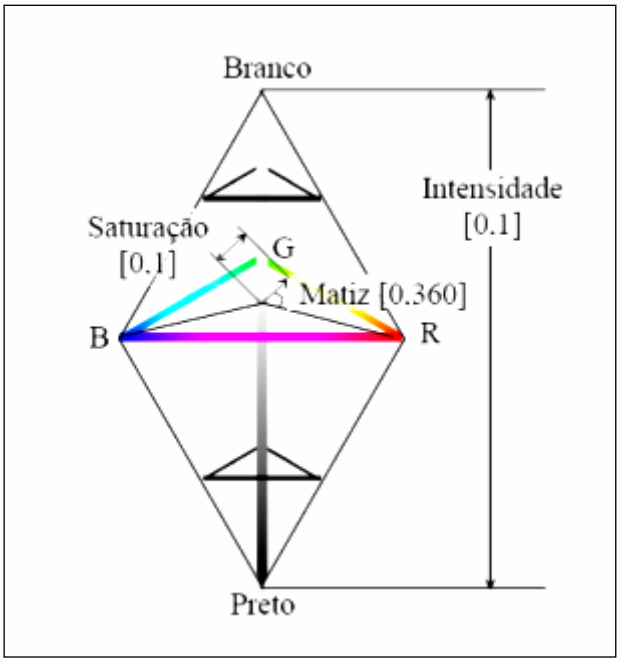


Figura 3.7. Representação de cores no modelo HSI (LIMA et al., 2005)

Uma descrição mais detalhada desses modelos pode ser encontrada em (GONZALEZ; WOODS, 1992). Neste trabalho, devido à facilidade de implementação em *hardware*, o sistema de cores adotado é o RGB.

Em imagens coloridas, os *pixel*s são representados pela seguinte notação vetorial:

$$P(x,y) = [P_1(x,y), P_2(x,y), P_3(x,y)]^T \qquad (3.14)$$

Assim, a aplicação da morfologia matemática a imagens coloridas é dificultada pela natureza vetorial dos dados presentes nessas imagens. A morfologia matemática é baseada na aplicação da teoria de reticulados a estruturas espaciais (ÂNGULO; SERRA, 2005). A definição dos operadores morfológicos requer uma estrutura de reticulado completo, conforme já abordado. Portanto, para se estender os conceitos de morfologia clássica a imagens coloridas, deve-se primeiro escolher uma ordenação adequada de *pixel*s coloridos, um espaço de cores que determine o modo como esses *pixel*s serão representados e a definição dos operadores de supremo e ínfimo no referido espaço de cores adotado. Existem diversas técnicas para ordenação vetorial, conforme será visto a seguir. As duas principais abordagens são: *ordenação marginal* e *ordenação vetorial*. Na ordenação marginal, cada componente P_i (i=1..3) é ordenada independentemente e as operações morfológicas são aplicadas a cada canal separadamente. Infelizmente, esse procedimento possui algumas deficiências, como, por exemplo, a criação de novas cores que não fazem parte da imagem original e pode ser inaceitável em aplicações que utilizem a cor para reconhecimento de objetos. Métodos de ordenação vetorial são mais aconselháveis neste caso. Nesses métodos, realiza-se somente um processamento sobre as três componentes de cor de cada *pixel*. Uma vez estabelecida a forma de ordenação, os operadores morfológicos podem ser definidos de

maneira convencional. Assim, a erosão vetorial E_{nB} de uma imagem colorida *f* em *x*, por um elemento estruturante *B* de tamanho *n*, é dada por:

$$E_{nB}(f)(x) = \{\inf[f(z)], z \in n(B_x)\} \quad (3.15)$$

A dilatação vetorial D_{nB} é obtida pela substituição do operador *inf* pelo operador *sup*, segundo a expressão 3.16. A abertura é uma erosão seguida por uma dilatação e o fechamento é uma dilatação seguida por um erosão.

$$D_{nB}(f)(x) = \{\sup[f(z)], z \in n(B_x)\} \quad (3.16)$$

Diferentes estratégias para ordenação vetorial são analisadas em (BARNETT, 1976). A seguir serão discutidas algumas.

3.3.3.1.1 Ordenação por uma Componente

O método de ordenação vetorial mediante uma componente é o mais simples em relação aos demais existentes. Nesse método, os vetores são ordenados por meio do valor de uma única componente previamente definida como fonte de ordem. Assim, a ordem se reduz a uma comparação escalar. Esse tipo de relação não é antissimétrica, formando-se assim uma relação de pré-ordem. Portanto, nessa abordagem não existe a garantia de unicidade de ínfimo ou supremo de um conjunto. Em (COMER; DELP, 1999), o problema da unicidade é resolvido através de um critério geométrico. A seguir, apresenta-se a definição matemática dessa aproximação:

$$\forall(\mathbf{p} = (x, y, z), \mathbf{q} = (x', y', z')) \in Z^3 \; \mathbf{p} <= \mathbf{q} \Leftrightarrow x < x', \text{onde x : fonte de ordem} \qquad (3.17)$$

3.3.3.1.2 Ordenação por Medida de Distância

Esse método de ordem se baseia no cálculo de uma distância a um *pixel* de referência previamente definido. Na maioria dos casos, o *pixel* de referência é o *pixel* de cor preta, (0,0,0), no espaço de cores RGB. Em (COMER; DELP, 1999), utiliza-se a norma euclideana como método de ordenação de *pixel*s sobre a base RGB, dessa maneira:

$$\mathbf{p} <= \mathbf{q} \Leftrightarrow \sqrt{p(R)^2 + p(G)^2 + p(B)^2} <= \sqrt{q(R)^2 + q(G)^2 + q(B)^2} \qquad (3.18)$$

Esse método também possui uma relação de pré-ordem, pois é evidente que a redução a um escalar por medida de distância não produz escalares únicos para cada vetor.

3.3.3.1.3 Ordenação Canônica

Na estrutura canônica, dois vetores $\mathbf{p}$ e $\mathbf{q} \in Z^n$ são ordenados de acordo com a expressão 3.19.

$$\mathbf{p} <= \mathbf{q} \Leftrightarrow p(i) <= q(i), \forall i \in \{1...n\} \qquad (3.19)$$

A ordem induzida por esta comparação também será parcial. Por exemplo, os vetores (1,2,3) e (3,2,1) não são comparáveis na estrutura canônica. É evidente que esse método de ordenação é muito rígido, pois exige que todas as componentes de um vetor sejam menores ou

maiores do que outro vetor. É possível observar que o método de ordenação por uma componente é um caso particular desse método. A rigidez do reticulado canônico se refletirá no processamento morfológico, assim, somente o vetor com todas as componentes menores ou maiores do que os outros vetores analisados será selecionado como ínfimo ou supremo, respectivamente, do reticulado correspondente. Existem também métodos híbridos que não consideram todas as componentes. Devido a essa rigidez, poderão aparecer descontinuidades na imagem processada.

3.3.3.1.4 Ordenação Lexicográfica

A ordem lexicográfica é análoga à ordem das palavras em um dicionário (TALBOT, 1998). Esse método é baseado na atribuição de prioridades às componentes do vetor, de modo que umas possuam mais peso ou importância que outras no momento de definição da ordem. Assim, a ordem é determinada através da componente de maior prioridade. Se os valores comparados forem iguais, passa-se a comparar a componente seguinte e assim sucessivamente. A ordem lexicográfica é uma ordem total, de modo que todos os vetores sejam comparáveis. Para vetores de três componentes, a ordem lexicográfica se define formalmente de acordo com o procedimento 3.20.

$$\begin{aligned}\mathbf{p} <= \mathbf{q} \Leftrightarrow \{ & \text{se } p(1) < q(1), \\ & \text{senão se } p(1) = q(1) \text{ e } p(2) < q(2), \\ & \text{senão se } p(1) = q(1) \text{ e } p(2) = q(2) \text{ e } p(3) <= q(3)\}\end{aligned} \qquad (3.20)$$

Onde: *p(1)* é a componente de maior prioridade, *p(2)* a de segunda prioridade e *p(3)* a de menor prioridade. De acordo com (ZAMORA, 2002), esse método é orientado a espaços de cores cujas componentes cromáticas possuam uma diferente importância visual para a

percepção humana, como, por exemplo, HSI, HLS, HSV, LCH, L*a*b*, YIQ, entre outros (IWANOWSKI; SERRA, 1999).

3.3.3.1.5 Outros Métodos de Ordenação Vetorial

Outros métodos de ordenação vetorial, levando-se também em consideração outros espaços de cores, são descritos de forma detalhada em (ZAMORA, 2002). Em (CHANUSSOT; LAMBERT, 1998), propõe-se uma estratégia de ordenação vetorial por entrelaçamento de bits. Esse método de ordenação, totalmente simétrico, codifica os bits dos elementos de tal maneira que, através da representação binária dos sinais, é possível formar um escalar de 24 bits ao se misturar os valores de cada componente. Esse método é adequado ao espaço de cores RGB, onde todas as componentes são de mesma natureza e possuem a mesma importância visual. Tal método possui uma relação de ordem total no espaço de cores RGB. Na Figura 3.8 é possível visualizar o procedimento descrito anteriormente.

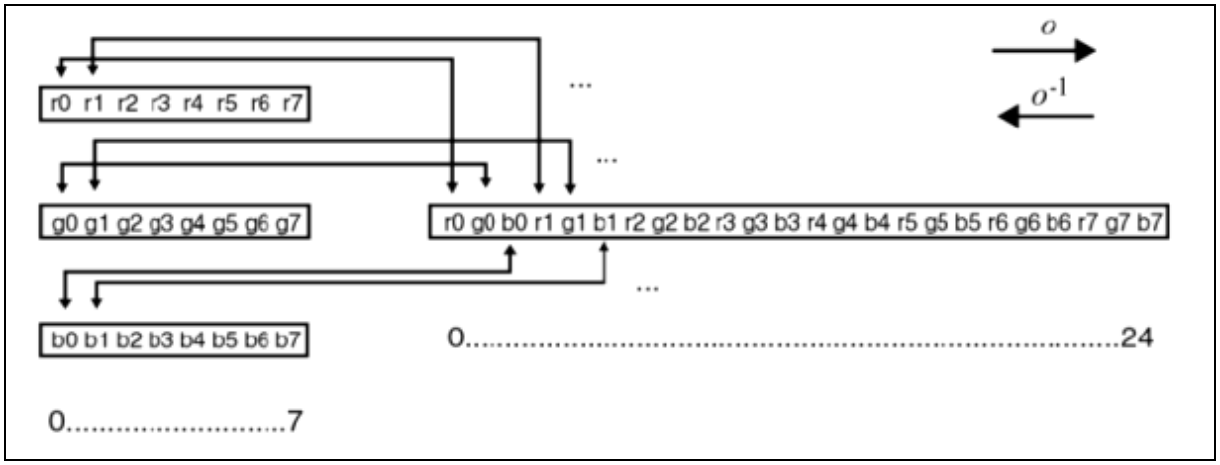


Figura 3.8. Entrelaçamento de bits no espaço de cores RGB (ZAMORA, 2002)

3.4 CONSIDERAÇÕES FINAIS

Em processamento de imagens, a morfologia matemática, que representa um ramo de processamento não linear, permite processar imagens com objetivos de realce, segmentação, detecção de bordas, esqueletização, afinamento, análise de formas e compressão, só para citar algumas aplicações. A morfologia matemática fornece uma abordagem para o processamento de imagens digitais que é baseada na forma. Apropriadamente usadas, as operações de morfologia matemática tendem a simplificar os dados de imagens preservando suas características de forma e eliminando irrelevâncias. De acordo com (GONZÁLEZ et al., 2002), a morfologia matemática lida melhor com algumas características indesejáveis do processamento de imagens linear clássico, uma vez que os filtros lineares geram uma distorção espacial na imagem original. Para propósitos de identificação de objetos ou peças defeituosas, requeridos em aplicações industriais de visão computacional, as operações de morfologia matemática são mais úteis que as de convolução empregadas em processamento de sinais, pois os operadores morfológicos se relacionam diretamente com a forma. A forma e o tamanho do elemento estruturante possibilitam testar e quantificar de que maneira o elemento estruturante está ou não está contido na imagem. Assim, a saída da operação morfológica depende essencialmente da forma do elemento estruturante utilizado nesse processo (GASTERATOS, 2000). Também, a decomposição do elemento estruturante reduz de forma linear o número de operações envolvidas nos processamentos morfológicos. Diversos métodos de decomposição de elementos estruturantes podem ser encontrados em (ANELLI et al., 1998; YANG; LEE, 1996; SHIH; WU, 1992; YANG; CHEN, 1993; SHIH; MITCHELL, 1991; PARK; CHIN, 1995; BROGGI, 1994; RICHARDSON; SCHAFER, 1991; ZHUANG; HARALICK, 1986; PECHT, 1985). Outra grande vantagem da morfologia matemática é a sua simplicidade de implementação, onde, através de suas operações básicas,

dilatação e erosão, por composição destas é possível implementar outros operadores mais complexos, diferentemente das demais técnicas de processamento de imagens que na maioria das vezes não se aproveitam das ferramentas já existentes. Outras abordagens existentes baseadas em morfologia matemática são a morfologia matemática *soft* (KOSKINEN et al., 1991) e a morfologia matemática *fuzzy* (BLOCH; MAITRE, 1995).

4. COMPUTAÇÃO EVOLUCIONÁRIA

4.1 CONSIDERAÇÕES INICIAIS

O objetivo deste capítulo é fornecer uma visão geral da área de computação evolucionária, com especial ênfase em uma de suas ramificações, a programação genética, que é a metodologia utilizada ao longo desta tese no desenvolvimento dos algoritmos propostos. Também, aborda-se, de forma sucinta, o teorema dos esquemas.

A computação evolucionária utiliza modelos computacionais de processos evolucionários baseados na teoria da evolução de Darwin e na biologia molecular com o objetivo de desenvolver sistemas computacionais para resolução de problemas. De acordo com essa teoria, a vida na terra é o resultado de um processo de seleção, realizada pelo meio ambiente, em que somente os organismos mais aptos e adaptados possuem chances de sobreviver e reproduzir-se. Nesse sentido, a computação evolucionária imita o processo de evolução natural, compartilhando uma base conceitual comum de simulação da evolução de estruturas de indivíduos através de processos de seleção e reprodução. Esses processos dependem do desempenho ou aptidão (*fitness*) dessas estruturas conforme determinado por seu ambiente. Mais precisamente, os algoritmos evolucionários mantêm uma população de estruturas que evoluem de acordo com as regras de seleção e outros operadores genéticos, tais como os de recombinação e mutação (SPEARS et al., 1993).

Embora a origem da computação evolucionária remonta à década de 50 (BREMERMANN, 1962; FRIEDBERG, 1958; FRIEDBERG, 1959; BOX, 1957), esse campo permaneceu relativamente desconhecido pela comunidade científica por muitos anos, devido à falta de plataformas computacionais adequadas à época, entre outros fatores. Os trabalhos de

(HOLLAND, 1962; RECHENBERG, 1965; SCHWEFEL, 1968; FOGEL, 1962) serviram para mudar esse cenário e atualmente se observa um crescimento exponencial no número de publicações e conferências nessa área. De acordo com (BÄCK et al., 1997), a computação evolucionária deve ser entendida como um conceito adaptável geral para resolução de problemas complexos, como, por exemplo, problemas de otimização combinatória. A maior parte das implementações existentes dessa área descendem de três abordagens fortemente relacionadas, embora independentes: algoritmos genéticos, programação evolucionária e estratégias evolucionárias.

Os algoritmos genéticos, introduzidos por (HOLLAND, 1975) e subseqüentemente estudados por (DE JONG, 1975; GOLDBERG, 1989; KOZA, 1992; MITCHELL, 1996), para citar somente alguns, foram propostos originalmente como um modelo geral de processos adaptativos, contudo, a maioria das aplicações dessas técnicas se concentra no domínio de otimização.

A programação evolucionária introduzida por (FOGEL, 1964) e estendida em (BURGIN, 1973; ATMAR, 1976; FOGEL, 1992), entre outros, foi inicialmente proposta como uma tentativa de criar inteligência artificial. Essa abordagem foi utilizada para evolução de máquinas de estado finito.

As estratégias evolucionárias foram desenvolvidas por (RECHENBERG, 1973; SCHWEFEL, 1975) e estendidas por (HERDY, 1992; KURSAWE, 1991; OSTERMEIER, 1994; RUDOLPH, 1990), para citar somente alguns, com o objetivo de resolver difíceis problemas de otimização de parâmetros nos domínios discreto e contínuo.

Durante a década de 80, avanços no desempenho de plataformas computacionais permitiram a aplicação de algoritmos evolucionários na resolução de problemas práticos de otimização e essas soluções receberam uma grande atenção da comunidade científica.

4.2 ALGORITMOS EVOLUCIONÁRIOS

Existem muitas variantes diferentes de algoritmos evolucionários (AE), no entanto, a idéia básica por trás dessas técnicas é sempre a mesma: dada uma população de indivíduos, a pressão ambiental faz prevalecer a seleção natural, sobrevivência dos indivíduos mais aptos, e isto causa um aumento na aptidão da população (EIBEN; SMITH, 2003). Por exemplo, dada uma função a ser maximizada, pode-se criar de forma aleatória um conjunto de soluções candidatas, isto é, elementos do domínio da função, e aplicar esta como uma medida de aptidão abstrata. Baseando-se nessa aptidão, os melhores candidatos são escolhidos para o processo de reprodução da próxima geração através da aplicação dos operadores de recombinação e/ou mutação a estes. A recombinação ou cruzamento é um operador aplicado a dois ou mais candidatos (cromossomos pais) selecionados com o objetivo de criar um ou mais novos candidatos (cromossomos filhos). A mutação é aplicada a um candidato e resulta em um novo candidato (cromossomo). A execução dos operadores genéticos de cruzamento e mutação leva a um conjunto de novos candidatos (descendentes) que irão competir com os mais velhos, de acordo com a aptidão e possível idade de cada um, por um lugar na próxima geração. Esse processo é então repetido até que um cromossomo com qualidade suficiente (solução) seja encontrado ou uma condição de parada adequada seja satisfeita. Nesse processo, existem duas forças fundamentais que governam a base dos sistemas evolucionários: os operadores genéticos de cruzamento e mutação, que criam a diversidade necessária da população, e a seleção que atua como uma força impulsionando a qualidade. A

aplicação combinada desses operadores geralmente tende a melhorar os valores de aptidão da população nas gerações subseqüentes. Alternativamente, a evolução pode ser vista como um processo de adaptação. Assim, o processo evolucionário tende a melhorar a capacidade de adaptação dos indivíduos de uma dada população ao seu ambiente.

Muitos dos componentes presentes no processo evolucionário são estocásticos. Durante a seleção, os indivíduos mais bem adaptados possuem maiores chances de serem selecionados em relação aos menos adaptados, entretanto, estes também terão chances de se tornarem pais ou de sobreviverem. Também, os operadores de cruzamento e mutação atuam de forma aleatória nos cromossomos da população de indivíduos. O esquema geral de um algoritmo evolucionário pode ser visto na Figura 4.1. Na Figura 4.2, tem-se um diagrama em blocos do referido algoritmo.

```
INICIO
        INICIALIZAR população com soluções aleatórias de indivíduos;
        AVALIAR cada indivíduo;
        ENQUANTO (condição de parada não satisfeita) FAÇA
                APLICAR operadores genéticos;
                AVALIAR novos indivíduos;
                SELECIONAR indivíduos para a próxima geração;
        FIM ENQUANTO
FIM
```

Figura 4.1. Esquema geral de um algoritmo evolucionário

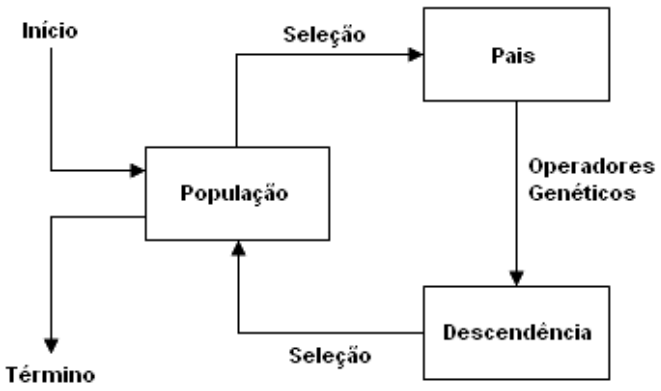


Figura 4.2. Diagrama em blocos do algoritmo dado pela Figura 4.1

Os algoritmos evolucionários se inserem na família de métodos de “geração e teste” ou ciclo “criar-testar-modificar”, de acordo com a Figura 4.3, que é análogo à forma na qual os seres humanos desenvolvem seus programas (RODRIGUES, 2002). Os vários dialetos de computação evolucionária seguem basicamente as idéias expostas acima, diferindo-se somente em detalhes técnicos.

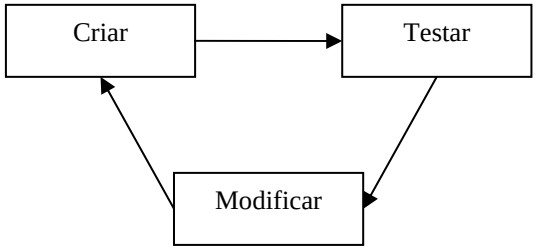


Figura 4.3. Ciclo criar-testar-modificar (RODRIGUES, 2002)

4.2.1 Componentes dos Algoritmos Evolucionários

Os algoritmos evolucionários possuem um certo número de componentes, procedimentos ou operadores que devem ser especificados de antemão. Os componentes mais importantes são (RAYWARD-SMITH et al., 1996; REEVES, 1993; GALVÃO; VALENÇA, 1999):

- representação (definição de indivíduos);
- função de avaliação (ou função de aptidão);
- população de indivíduos (cromossomos);
- mecanismos de seleção;
- operadores de variação;
- mecanismos de substituição;

Além dos passos supracitados, deve-se também definir um procedimento de inicialização e uma condição de término.

4.2.1.1 Representação (Definição de Indivíduos)

O primeiro passo na definição de um algoritmo evolucionário é fazer uma ponte entre o contexto do problema original e seu espaço de soluções onde o processo de evolução ocorrerá. Os objetos formando possíveis soluções dentro do contexto do referido problema são denominados de fenótipos e os indivíduos codificados do AE são chamados de genótipos. Assim, a representação se consiste num mapeamento de fenótipos num conjunto de genótipos, como, por exemplo, vetores de números inteiros ou reais, cadeias de bits, árvores sintáticas e máquinas de estado finito. É importante notar que o espaço de fenótipos pode ser muito diferente do espaço de genótipos, e que todo o processo evolucionário ocorre no espaço de genótipos. Uma solução, um bom fenótipo, é obtida pela decodificação do melhor genótipo após a condição de término. No espaço de genótipos, um indivíduo é chamado de cromossomo, um cromossomo é formado por genes e o valor de cada gene é denominado alelo.

4.2.1.2 Função de Aptidão

A função de aptidão atribui uma medida de qualidade aos genótipos. Assim, esta avalia a distância de um cromossomo em relação à solução ideal. Freqüentemente, o problema a ser resolvido por um algoritmo evolucionário é de otimização, nesse caso, essa função recebe o nome de função objetivo.

4.2.1.3 População

A função da população é manter a representação de possíveis soluções. A primeira população ou geração, geralmente é criada de forma aleatória. A população é formada por múltiplos conjuntos de genótipos. Também, nesse contexto é possível haver cópias de alguns indivíduos presentes nessa população. A população forma a unidade da evolução. Para uma dada representação é necessário definir o tamanho da população como um dos parâmetros do algoritmo evolucionário corrente. Na maior parte dos algoritmos evolucionários, o tamanho da população não se altera durante o processo de evolução. Contrariamente aos operadores de variação que atuam sobre um ou dois cromossomos pais, os operadores de seleção operam a nível populacional. A diversidade de uma população é uma medida do número de soluções diferentes presentes. Não existe uma única medida para a diversidade, tipicamente se utilizam as seguintes medidas: número de valores diferentes de aptidão, número de diferentes fenótipos, número de diferentes genótipos, entropia, entre outros.

4.2.1.4 Mecanismo de Seleção de Pais

A importância da seleção de pais é fazer uma separação entre indivíduos baseando-se em suas qualidades, em particular, permitir que os melhores indivíduos se tornem pais na próxima geração. Um indivíduo é considerado pai se foi selecionado para ser submetido a um operador de variação com a finalidade de gerar um descendente (filho). Os mecanismos de seleção (de escolha de pais e de substituição) são responsáveis por melhorarem a qualidade do processo evolucionário. Em computação evolucionária, a seleção de pais é tipicamente probabilística. Dessa maneira, os indivíduos de maior aptidão ou maior qualidade terão uma probabilidade maior de se tornarem pais do que os de menor aptidão. Todavia, aos indivíduos

de baixa qualidade é dada uma chance positiva de se reproduzirem, senão o processo de busca tenderá a se tornar muito ganancioso e poderá convergir para ótimos locais.

4.2.1.5 Operadores de Variação

A função dos operadores de variação é criar novos indivíduos a partir de indivíduos mais velhos. No espaço correspondente de fenótipos, isto equivale a gerar novas soluções candidatas. Sob a perspectiva do método de "geração e teste", os operadores de variação desempenham o papel de gerar. Em computação evolucionária, os operadores de variação são divididos em dois tipos, conforme descrito a seguir.

4.2.1.5.1 Mutação

Em computação evolucionária, um operador de variação unário é comumente chamado de mutação. Aplicando-se este a um genótipo, gera-se um mutante (filho ou descendente desse genótipo). Um operador de mutação é sempre estocástico. O papel do operador de mutação é diferente nos vários dialetos de computação evolucionária existentes. É importante notar que os operadores de variação formam a implementação evolucionária dos passos elementares no contexto do espaço de busca. Assim, gerar um filho, implica saltar para um novo ponto do espaço de busca. Sob essa ótica, o operador de mutação também possui um papel teórico: a garantia do espaço ser conectado. A importância dessa abordagem está em concordância com alguns teoremas que afirmam que um algoritmo evolucionário encontrará, após certo tempo, um ótimo global de um dado problema, baseando-se na propriedade de que um genótipo representando uma possível solução possa ser alcançado pelos operadores de variação. A forma mais simples de satisfazer essa condição é permitir que o operador de

mutação possa saltar para qualquer lugar do espaço de busca. Entretanto, muitos pesquisadores acreditam que essas provas têm uma importância prática limitada e muitas implementações de AE não possuem tal propriedade.

4.2.1.5.2 Cruzamento

Um operador de variação binário é chamado de recombinação ou cruzamento. Conforme o próprio nome sugere, tal operador combina informações de dois cromossomos pais gerando um ou dois cromossomos filhos. Similarmente à mutação, o operador de cruzamento também é estocástico. Novamente, o papel da recombinação é diferente em cada dialeto existente. Operadores de recombinação de vários pais são matematicamente possíveis e fáceis de implementar, embora não possuam equivalência biológica. O princípio da recombinação é simples, assim, do cruzamento de dois indivíduos com características desejáveis, produz-se um descendente que combina as características de ambos os pais que o geraram. Com isso se espera que o descendente possua uma aptidão melhor que a de seus pais. Muitas vezes este pode não ser melhor nem pior que seus pais. Embora a biologia do planeta terra sugere, salvo raras exceções onde poucos organismos se reproduzem de forma assexuada e muitos de forma sexuada, que a recombinação é a forma superior de reprodução, os operadores em computação evolucionária são usualmente aplicados probabilisticamente, assim, existe uma chance de não serem aplicados.

4.2.1.6 Mecanismo de Seleção Ambiental

A função da seleção ambiental é similar à função da seleção de pais, mas esse mecanismo é utilizado em um estágio diferente do ciclo de evolução. Após a criação de

descendentes, deve-se decidir quais indivíduos deverão permanecer entre os demais da próxima geração. Essa decisão é usualmente baseada no valor de aptidão de cada indivíduo, favorecendo assim os que possuem melhor qualidade, embora o conceito de idade muitas vezes seja também utilizado. Nesse contexto, o mecanismo de seleção é determinístico. Assim, escolhem-se quais indivíduos deverão ser apagados para darem lugar a novos indivíduos com a finalidade de manter o tamanho da população constante.

4.2.1.7 Condição de Parada

Algoritmos evolucionários são estocásticos e na maioria das vezes não existe garantia de que uma solução ótima será obtida. Nesse caso, o algoritmo nunca convergirá. Para que isso não ocorra, além da condição de parada, outras opções são utilizadas, assim, destacam-se: número máximo de gerações, diversidade da população, entre outras.

4.3 PROGRAMAÇÃO GENÉTICA

A programação genética (PG) utiliza conceitos de genética e da seleção natural de Darwin para gerar e evoluir programas de computador. É um ramo relativamente novo de computação evolucionária e gradualmente está se consolidando como um método promissor em aplicações de reconhecimento de padrões, problemas de classificação, modelamento de sistemas complexos, síntese de filtros eletrônicos, amplificadores e outros circuitos (NILSSON, 1998).

Conforme já abordado, o objetivo primordial da programação genética é descobrir como os computadores podem aprender a resolver problemas sem, no entanto, serem programados para essa tarefa (KOZA, 1992; KOZA, 1994; KOZA et al., 1999; BANZHAF,

1998). Esse paradigma foi idealizado por John Koza (KOZA, 1989) com base nos trabalhos de John Holland em algoritmos genéticos (AGs) (HOLLAND, 1975). A programação genética é uma extensão de algoritmos genéticos na qual os cromossomos são representados por programas de computador com tamanhos variados. Nessa abordagem, tais programas são representados por árvores sintáticas ao invés de linhas de código. Por exemplo, considere a representação da expressão min(x*2,x+2*y), de acordo com a Figura 4.4.

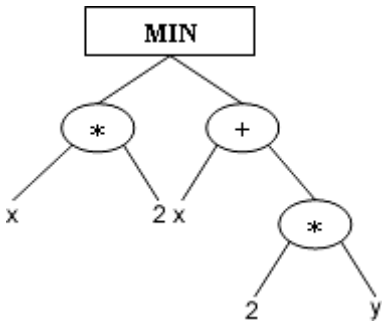


Figura 4.4. Árvore sintática representando a expressão min(x*2,x+2*y)

O conjunto de nós internos de uma árvore sintática é representado por funções e seu conjunto de nós terminais é formado por variáveis e constantes. As funções e os terminais são escolhidos de acordo com uma dada aplicação. O algoritmo básico de busca utilizado em programação genética se baseia no algoritmo genético clássico com seus operadores de cruzamento e mutação operando sobre estruturas de árvores. Este algoritmo trabalha da maneira descrita a seguir. Inicialmente são gerados programas pertinentes ao problema, de forma estocástica, segundo a estrutura de árvore dada pela Figura 4.4. Então, o procedimento genético é controlado por uma função de aptidão que avalia a qualidade dos programas (indivíduos) gerados, onde se torna necessário executar cada programa dentro desse ambiente, assim, os melhores são selecionados e modificados para produzir uma nova população que

será avaliada na próxima geração. Portanto, esse processo é repetido até que uma condição de parada seja satisfeita. Tal processo pode ser visualizado na Figura 4.5. Nas próximas seções, esses passos serão descritos pormenorizadamente.

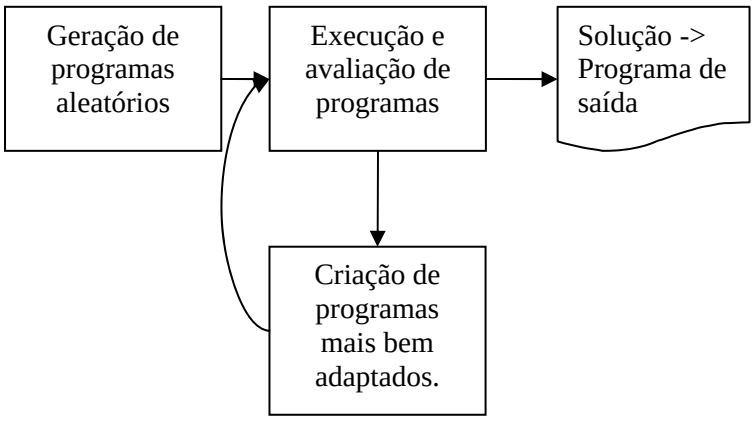


Figura 4.5. Estrutura principal do paradigma de programação genética

Nas formas mais avançadas de programação genética, os programas podem ser compostos de múltiplos componentes, como, por exemplo, sub-rotinas (KOZA; POLI, 2003). Nesse caso, a representação usada em programação genética será um conjunto de árvores (ramos) agrupadas abaixo do nó raiz, conforme a Figura 4.6. O número e o tipo de ramificação em um programa, juntamente com certas características estruturais, formam a arquitetura do programa.

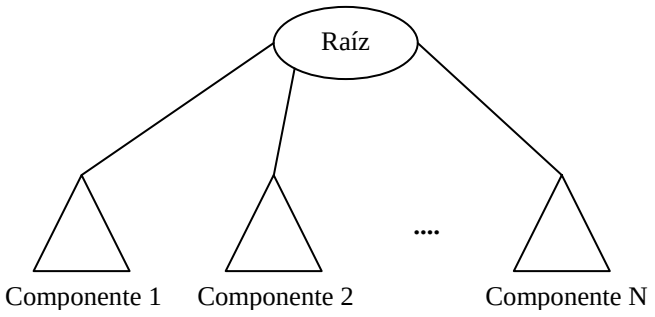


Figura 4.6. Representação de múltiplas árvores de programa (sub-rotinas) (KOZA e POLI, 2003)

Árvores sintáticas em programação genética e suas expressões correspondentes podem equivalentemente ser representadas em notação prefixa, como, por exemplo, expressões-S de LISP.

4.3.1 Passos Preparatórios em Programação Genética

O processo de programação genética tem início por meio de uma especificação de requisitos em alto nível para um dado problema e tenta produzir um programa de computador para solucionar o mesmo. A única interação do ser humano com o processo evolucionário se dá através da especificação de alto nível do problema e se baseia em alguns passos preparatórios bem definidos. Os principais passos preparatórios da versão clássica de programação genética exigem que o usuário especifique:

- o conjunto de terminais, como, por exemplo, variáveis independentes do problema, constantes, etc.,
- o conjunto de funções primitivas, como, por exemplo, +, -, *, % (divisão protegida), seno, cosseno, etc.,
- a função de aptidão,
- certos parâmetros para controle de execução e
- o critério de término.

Nos dois primeiros passos se especificam os ingredientes que serão utilizados para criar os programas de computador. A identificação dos conjuntos de funções e terminais para um dado problema é geralmente um processo direto. Para algumas classes de problemas, o conjunto de funções consistirá meramente de funções aritméticas e do operador condicional *if*.

O conjunto de terminais pode conter variáveis independentes e constantes numéricas. Para muitos outros problemas, os ingredientes incluem funções especializadas e terminais. Por exemplo, para a síntese de circuitos elétricos analógicos, o conjunto de funções pode conter: transistores, capacitores e resistores. Uma vez que o usuário tenha identificado os ingredientes primitivos para um problema de síntese de circuitos, o mesmo conjunto de ingredientes pode ser utilizado para sintetizar automaticamente um amplificador, um circuito computacional, um filtro ativo ou qualquer outro circuito composto desses ingredientes. A função de aptidão, que é o conceito mais importante e mais difícil em PG, especifica o que de fato precisa ser feito. Esta determina o quanto um programa está apto a resolver um dado problema. A avaliação de aptidão depende do domínio do problema e pode ser medida de diversas maneiras. Os dois últimos passos sobreditos são administrativos. Um dos parâmetros de controle mais importantes é o tamanho da população. Os outros parâmetros incluem probabilidades de execução dos operadores genéticos, o tamanho máximo dos programas do espaço de soluções do problema e outros detalhes de execução. A condição de parada pode incluir um número máximo de gerações ou pode interromper o laço de repetição do processo evolutivo quando uma solução satisfatória for encontrada.

4.3.2 Detalhes de Execução

Os passos de um algoritmo de PG são independentes do tipo de problema sob consideração. Inicialmente, gera-se uma população de programas aleatoriamente. Em seguida, o procedimento genético transforma a população de indivíduos criados em uma nova geração através da aplicação de operadores genéticos. Estes operadores são aplicados probabilisticamente a cromossomos da população corrente, gerando-se assim uma nova população de indivíduos (programas representando soluções para o problema). Este processo

é então repetido por muitas gerações até que uma solução ótima seja obtida. Em suma, são necessários os seguintes passos:

1. Criação de forma aleatória de uma população (geração 0) de indivíduos compostos de funções e terminais do espaço de soluções do problema.
2. Execução de forma iterativa dos passos seguintes (gerações) até que uma condição de parada seja satisfeita:
    a. Execução de cada programa da população e verificação de seu valor de aptidão usando a medida de aptidão adotada para o problema.
    b. Seleção de um ou mais indivíduos (programas) da população com probabilidades baseadas em seus desempenhos (*fitness*) para participarem das operações genéticas em *c*.
    c. Criação de novos programas por meio dos operadores genéticos dados a seguir com probabilidades especificas:
        i. **Reprodução:** Cópia de indivíduos selecionados para a nova população.
        ii. **Cruzamento:** Criação de programas filhos para a nova população por meio de recombinação de partes escolhidas aleatoriamente de indivíduos pais.
        iii. **Mutação:** Criação de um novo programa filho por meio de mutação de uma parte escolhida aleatoriamente de um indivíduo selecionado.
3. Após a condição de parada ter sido satisfeita, o melhor programa ou o mais próximo da solução ideal é apresentado como resultado da execução.

Na Figura 4.7 é apresentado um fluxograma contendo todos os passos básicos de PG descritos anteriormente.

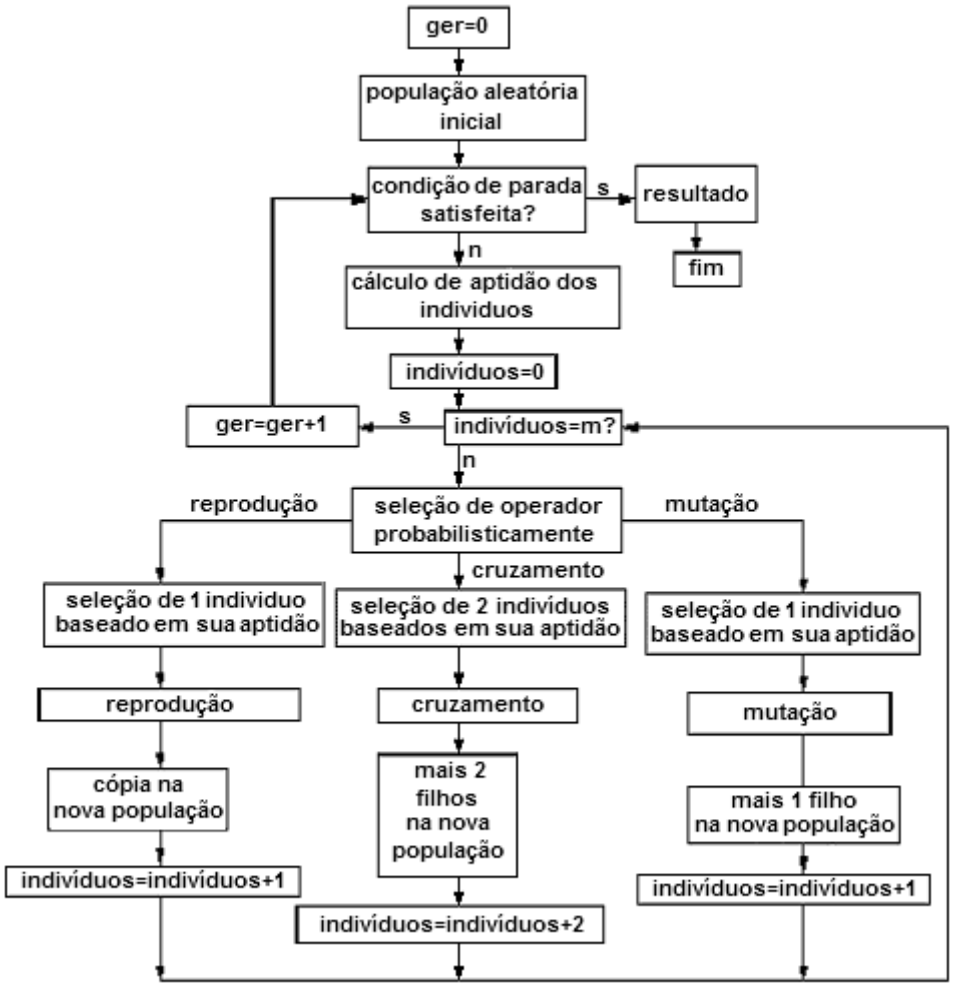


Figura 4.7. Fluxograma básico de programação genética (Adaptado de GENETIC PROGRAMMING, 2006)

Os indivíduos iniciais geralmente são gerados sujeitos a um tamanho máximo pré-estabelecido pelo usuário. Basicamente, dois métodos se destacam para esta tarefa, conforme descrito a seguir. No método de inicialização *full*, os nós são tomados do conjunto de funções até que uma profundidade máxima de árvore seja alcançada. Além dessa profundidade, somente os terminais podem ser escolhidos. Na Figura 4.8 é mostrado esse método. Uma variante do método descrito anteriormente, o método de inicialização *grow*, permite selecionar os nós do conjunto primitivo completo até que a profundidade limite seja

alcançada. Esse método está ilustrado na Figura 4.9. Em geral, depois da fase de inicialização, os programas da população possuem tamanho e forma diferenciados.

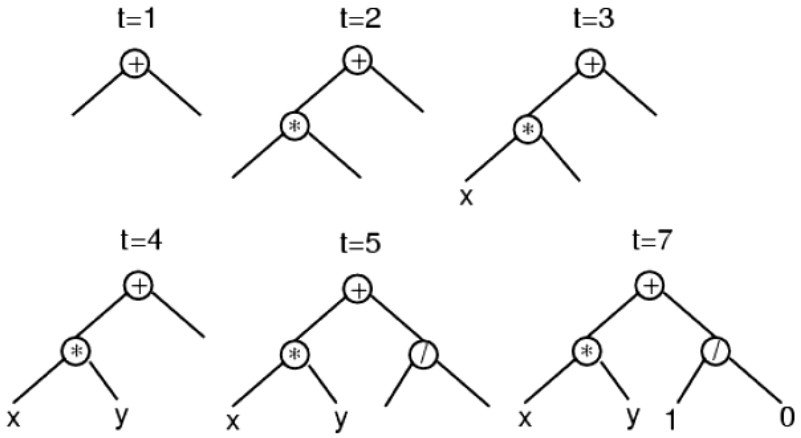


Figura 4.8. Método de inicialização *Full* (KOZA; POLI, 2003)

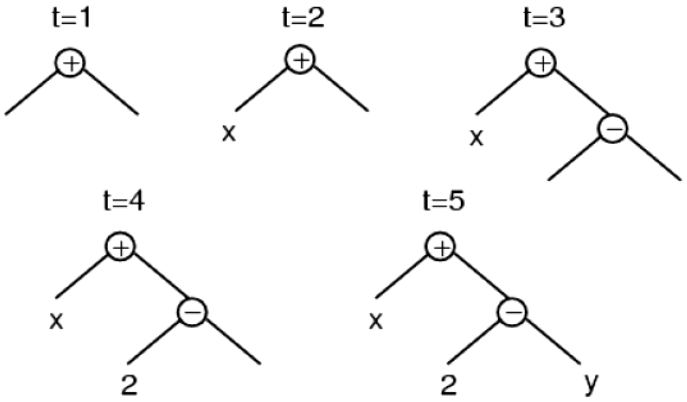


Figura 4.9. Método de inicialização *Grow* (KOZA; POLI, 2003)

Cada indivíduo da população é medido ou comparado de acordo com seu desempenho no ambiente do problema. Normalmente, a avaliação de aptidão requer a execução dos programas contidos numa dada população. Interpretar uma árvore de programa significa executar os nós dessa árvore numa ordem que garanta que estes não sejam executados antes

dos valores de seus argumentos serem conhecidos. Isto é realizado geralmente de forma recursiva, conforme a direção indicada na Figura 4.10. No exemplo mostrado na referida Figura, o valor da variável X é “-1” e o valor final da avaliação da expressão em questão é “2”.

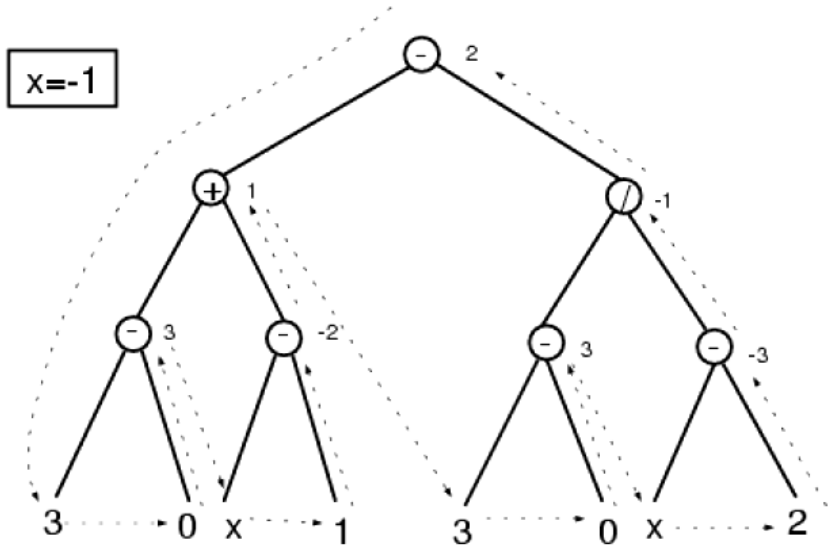


Figura 4.10. Exemplo de interpretação de uma árvore sintática (KOZA; POLI, 2003)

Independentemente da estratégia de execução adotada, a aptidão de um programa pode ser medida de muitas maneiras diferentes, incluindo, por exemplo, a quantidade de erro entre sua saída e a saída desejada, a quantidade de tempo necessária para levar um sistema a um estado desejado, a precisão de um programa em reconhecer padrões ou classificar objetos em classes específicas, entre outros exemplos. A execução dos programas dentro desse contexto pode retornar um ou mais valores explícitos. Alternativamente, através da execução desses programas é possível realizar certas ações.

Dados dois indivíduos pais, o operador de cruzamento tipicamente seleciona aleatoriamente um ponto de corte em cada um desses indivíduos e permuta as sub-árvores

abaixo dos pontos selecionados, conforme ilustrado na Figura 4.11. Freqüentemente, os pontos de corte não são selecionados com probabilidade uniforme.

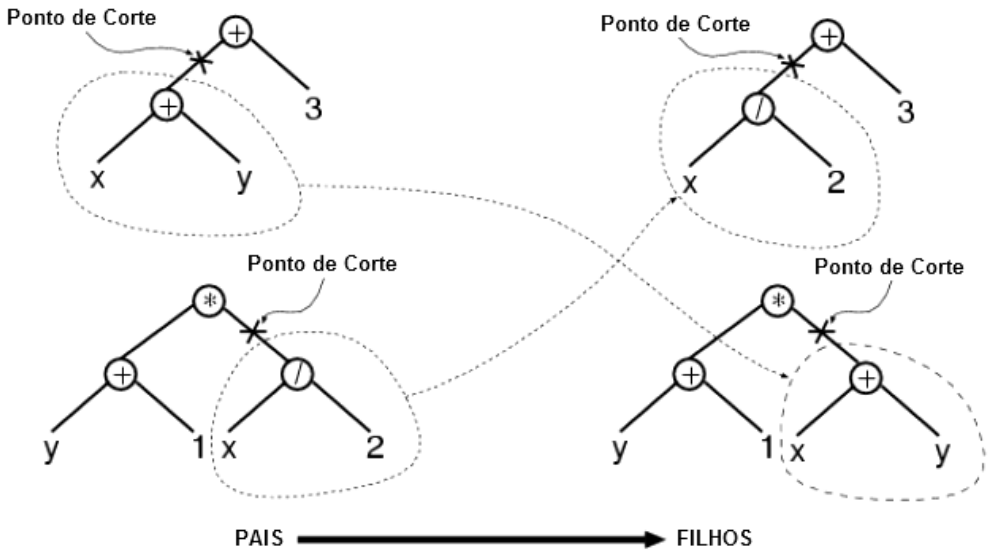


Figura 4.11. Exemplo do operador de cruzamento entre árvores sintáticas (Adaptado de KOZA; POLI, 2003)

O operador de mutação escolhe um ponto de corte aleatório da árvore sob consideração e substitui o ramo abaixo desse ponto por outra sub-árvore gerada aleatoriamente. A Figura 4.12 apresenta um exemplo do operador de mutação.

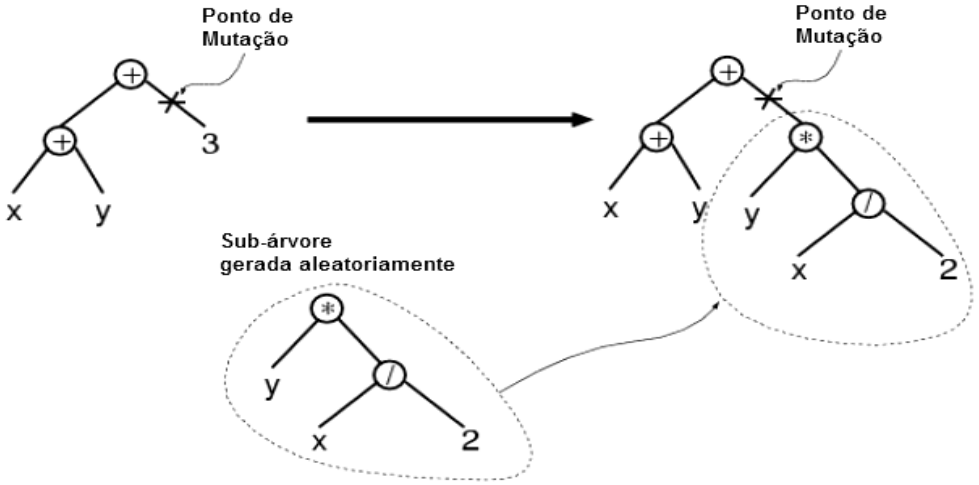


Figura 4.12. Exemplo do operador de mutação (Adaptado de KOZA; POLI, 2003)

O operador de reprodução simplesmente copia certos indivíduos na nova população. Os operadores descritos anteriormente são aplicados a indivíduos que são probabilisticamente selecionados de acordo com seu valor de aptidão. Nesse processo probabilístico, os melhores indivíduos são favorecidos em relação aos piores. Contudo, nem sempre os melhores indivíduos são selecionados. Na prática foram propostos diversos métodos de seleção, entre eles, podem-se destacar: seleção proporcional (HOLLAND, 1975), seleção por torneio (GOLDBERG, 1991), seleção por truncamento (MUHLENBEIN, 1993), para citar somente alguns. Depois dos operadores genéticos serem aplicados à população corrente, a população de descendentes substituirá a mesma. Esse processo iterativo é então repetido por muitas gerações. Portanto, o melhor indivíduo (solução) ou o indivíduo que melhor se aproxima da solução ideal é apresentado logo após a condição de parada ter sido satisfeita. Todos os programas dentro dessa abordagem devem ser sintaticamente válidos. De acordo com (KOZA, 1992; RODRIGUES, 2002), para garantir a viabilidade de árvores sintáticas, Koza definiu a propriedade de fechamento, dessa maneira, cada função do domínio de soluções deve aceitar como argumento (ou argumentos) qualquer valor que possa ser retornado por qualquer função ou terminal. Assim, há a garantia de que qualquer árvore gerada será avaliada corretamente. Um exemplo típico é a operação de divisão. Para esta operação, Koza criou a função de divisão protegida, que retornará o valor “1” se ocorrer uma divisão por zero, caso contrário, retornará o quociente da divisão. Para garantir a convergência do algoritmo genético, Koza definiu a propriedade de suficiência, onde o espaço de soluções utilizado pelo problema deverá ser capaz de representar soluções concretas dentro desse ambiente. Dependendo do problema, esta propriedade poderá ser trivial ou exigir algum conhecimento a respeito das possíveis soluções.

4.3.3 Teorema dos Esquemas

A programação genética é uma técnica de busca que explora o espaço de programas de computador. Conforme já discutido, a busca por soluções de um determinado problema começa a partir de um grupo de pontos (programas aleatórios) no espaço de busca. Os pontos que estão acima da média de qualidade são usados para gerar uma nova geração de pontos através dos operadores genéticos de cruzamento, mutação, reprodução e possivelmente outras operações. Então, esse processo é repetido até que uma condição de parada seja satisfeita. Se fosse possível visualizar tal processo, inicialmente se enxergaria uma nuvem de pontos espalhados aleatoriamente, e com o passar das gerações, essa nuvem deveria seguir uma trajetória bem definida no espaço de soluções. Considerando-se que esse mecanismo é estocástico, em diferentes execuções seria possível visualizar diferentes trajetórias. Entretanto, normalmente é impossível visualizar o espaço de busca devido à sua dimensionalidade e complexidade. Assim, uma maneira de superar essa dificuldade seria através do registro de variações de certos descritores numéricos, entre eles, a aptidão média dos indivíduos numa população, o tamanho médio dos programas, etc. Essa forma também é sujeita a erros, considerando-se que esse sistema é complexo, adaptativo e possui muitos graus de liberdade. Assim, para se compreender de forma precisa o comportamento desses algoritmos, torna-se necessário definir um modelo matemático de busca evolucionária. A teoria do esquema está entre as mais antigas e prováveis classes conhecidas de modelos de algoritmos evolucionários.

O teorema dos esquemas foi desenvolvido por (HOLLAND, 1975), e busca, de forma teórica, descrever o comportamento dos algoritmos genéticos (AGs). De acordo com (GALVÃO; VALENÇA, 1999), sua compreensão pode auxiliar na construção de aplicações eficientes de procedimentos genéticos. Segundo Holland, os AGs manipulam determinados

segmentos de cadeia de bits. A esses segmentos, dá-se o nome de esquemas. Um esquema é formado pelos símbolos (0,1,*), onde o valor do caractere * denotará um estado irrelevante, podendo ser 0 ou 1. Por exemplo, os esquemas H1=1****, H2=**10* e H3=*0*01 estão contidos no cromossomo 10101. Este cromossomo pode ter até 2^5 esquemas. Os cromossomos 11001, 11011 e 10101 possuem o esquema 1****. O comprimento de um esquema, C(H), é a diferença entre a última posição e a primeira posição ocupadas por bits 0 ou 1. A ordem de um esquema, O(H), é definida como o número de bits 0 ou 1 desse esquema. A variação do número de esquemas H entre duas gerações consecutivas pode ser dada pela equação a seguir:

$$NE = \frac{MA}{MT} CR \qquad (4.1)$$

onde: NE: número esperado de cópias contendo o esquema H;

MA: média das aptidões dos cromossomos contendo o esquema H;

MT: média das aptidões de toda a população;

CR: número de cromossomos da população atual que contém o esquema H.

Pela equação 4.1 é possível verificar que o número de esquemas H irá aumentar na próxima população se MA>MT, ou seja, se o esquema H estiver contido em bons indivíduos. Entretanto, o esquema H poderá ser destruído pelos operadores de cruzamento e mutação. De acordo com (GOLDBERG, 1989), esquemas possuindo comprimentos menores que estejam contidos em bons cromossomos irão aumentar de forma exponencial nas próximas gerações, enquanto que esquemas que estejam contidos em cromossomos ruins tenderão a desaparecer nas próximas gerações do processo evolucionário. Bons esquemas de comprimentos reduzidos

recebem o nome de blocos de construção. A informação contida em um bloco de construção, quando combinada com informações de outros blocos, ao longo das gerações, produz indivíduos de altas aptidões. Tal afirmação é decorrente da hipótese dos blocos de construção. Problemas que não obedecem a essa hipótese são denominados de AG-Deceptivos. Também, de acordo com Holland, apesar de serem manipulados N cromossomos, a quantidade de esquemas será muito maior, da ordem de $O(N^3)$ esquemas. Essa propriedade é denominada de paralelismo implícito, ou seja, o procedimento genético manipulará uma grande quantidade de informações em paralelo somente com N cromossomos. Quando se incorporam bons blocos de construção num indivíduo, sua aptidão é melhorada. Logo, a estrutura de codificação de um indivíduo determinará a formação de blocos de construção.

A expressão 4.2, dada a seguir, mostra uma versão ligeiramente diferente do teorema original de Holland, quando o operador de cruzamento é aplicado a dois indivíduos da população intermediária:

$$E[m(H,t+1)] \geq m(H,t).\frac{f(H,t)}{f'(t)}.(1-p_m)^{O(H)}.[1-p_c.\frac{L(H)}{N-1}.(1-\frac{m(H,t).f(H,t)}{M.f'(t)})] \quad (4.2)$$

onde: m(H,t): número de cromossomos contendo o esquema H na geração t;

f(H,t): aptidão média dos cromossomos que contém o esquema H;

f’(t): aptidão média dos cromossomos na população;

p_m: probabilidade de mutação por bit;

p_c: probabilidade de cruzamento;

N: número de bits no cromossomo;

M: número de cromossomos na população;

E[(m(H,t+1)]: número esperado de cromossomos contendo o esquema H na geração t+1;

L(H): comprimento do esquema;

O(H): ordem do esquema.

O termo entre colchetes na expressão 4.2, contendo a taxa de cruzamento, representa a probabilidade de um esquema ser destruído na geração t devido ao operador de cruzamento. Tal probabilidade depende da freqüência desse esquema na população intermediária, mas também da fragilidade intrínseca do esquema L(H)/(N-1).

A extensão dessas idéias para programação genética não é uma tarefa trivial na prática. Uma das dificuldades encontradas é o tamanho variável das estruturas presentes em PG, assim, muitas alternativas foram propostas na literatura tentando solucionar este e outros problemas. Nessas abordagens, os esquemas são definidos como sendo compostos de uma ou muitas árvores ou fragmentos de árvores. Para uma descrição mais detalhada dessa teoria aplicada à programação genética, recomendam-se os trabalhos de (POLI; LANGDON, 1997; POLI; LANGDON, 1998; POLI; McPHEE, 2001; LANGDON; POLI, 2002).

4.4 CONSIDERAÇÕES FINAIS

A idéia de utilizar computadores para resolver problemas de forma automática é central em inteligência artificial (TURING, 1950; SAMUEL, 1983). A programação genética é um método sistemático, no qual os computadores buscam resolver automaticamente um problema, partindo de uma especificação de alto nível do que realmente precisa ser feito. Também, é independente do domínio de aplicação. O procedimento genético utilizado em PG procura explorar pontos inteiramente novos em seu espaço de busca, bem como intensificar

essa busca em torno de regiões promissoras. Esse mecanismo de diversificação e intensificação (*exploration e explotation*) é obtido no procedimento genético através do correto uso dos operadores genéticos (SILVA, 2005). O desempenho do procedimento genético também é fortemente influenciado pelos parâmetros utilizados em um determinado problema, entre eles, citam-se: tamanho da população, tamanho da árvore de programa, taxas de cruzamento e mutação, tipos de instruções utilizadas, número de gerações e condição de parada. Também, o processo de inicialização da população influencia a qualidade da solução, pois, como no caso biológico, não há evolução sem variedade, desse modo, é importante que a população de programas inicial cubra a maior parte possível do espaço de busca de soluções. Assim, os indivíduos da população inicial poderiam ser gerados por uma determinada heurística. A obrigatoriedade da propriedade de fechamento limita a aplicação ampla da programação genética. A necessidade dessa propriedade é devida ao uso irrestrito dos operadores genéticos. Com a finalidade de contornar esses problemas, alguns pesquisadores utilizam gramáticas, assim, além da especificação dos conjuntos de funções e terminais, informam-se também as regras de formação de programas (RODRIGUES, 2002; GRUAU, 1996).

5. IMPLEMENTAÇÕES

5.1 CONSIDERAÇÕES INICIAIS

Neste capítulo, conforme já abordado, são apresentadas as implementações realizadas no projeto de doutorado. Inicialmente, realiza-se uma revisão sobre as principais arquiteturas em *hardware* dedicado existentes para processamento morfológico de imagens. Em particular, é apresentado o desenvolvimento de uma nova arquitetura para processamento morfológico de imagens coloridas em tempo real utilizando um FPGA da empresa Altera e a linguagem Verilog HDL, considerando-se que na literatura não há um número significativo de arquiteturas morfológicas em *hardware* voltadas para processamento de imagens coloridas. Para esta arquitetura foi proposto um novo método de ordenação vetorial, baseado na distância *city-block*, para eliminar redundâncias decorrentes do processamento morfológico. Os resultados deste método são comparados com outras implementações. Como outra contribuição deste trabalho, apresenta-se a implementação de um sistema RGB original utilizando um CPLD da Altera para pré-processamento de imagens digitais em tempo real, onde este é baseado num sensor de vídeo CMOS e reconfigurável em *hardware* pelo protocolo SCCBus. Este protocolo foi implementado através de um sistema microprocessado utilizando o microcontrolador 2051 da Atmel. Além das aplicações sobreditas, o sistema RGB desenvolvido serve também de plataforma à aquisição de dados e pode ser utilizado para conversão de padrões de vídeo. Por fim, é apresentado o sistema evolucionário original desenvolvido no trabalho. Primeiramente, apresenta-se o desenvolvimento de uma *toolbox* no Matlab 7.0, denominada *morph_gen*, para construção automática de operadores morfológicos e lógicos por meio de uma abordagem linear baseada em programação genética visando aplicações de visão computacional. Em seguida, apresenta-se uma arquitetura *pipeline*

reconfigurável em *hardware* pelas instruções geradas pela *toolbox morph_gen*, com a finalidade de processar imagens em tempo real. Esta arquitetura foi desenvolvida utilizando a placa de desenvolvimento e educacional DE2 da Altera. Também, são mostrados exemplos de aplicações práticas através da arquitetura desenvolvida, nos quais seus resultados são comparados com outras formas de implementação.

5.2 ARQUITETURAS EXISTENTES PARA MORFOLOGIA MATEMÁTICA

5.2.1 Arquiteturas de Propósito Geral para Processamento de Imagens

Segundo (GASTERATOS, 2002 apud FLYNN, 1966; TANENBAUM, 1990), as arquiteturas de computadores são classificadas em quatro categorias principais: *Single Instruction, Single Data* (SISD), *Multiple Instruction, Single Data* (MISD), *Single Instruction, Multiple Data* (SIMD) e *Multiple Instruction, Multiple Data* (MIMD). Em processamento de imagens, devido a grande quantidade de dados envolvida, o paralelismo se torna necessário. As arquiteturas paralelas também podem ser agrupadas em duas categorias principais: síncronas e MIMD. Entretanto, esse agrupamento não é único e seus domínios podem ser confundidos. Assim, para a categoria síncrona, têm-se as seguintes arquiteturas: processadores vetoriais, SIMD e arquiteturas sistólicas. Para a categoria MIMD, têm-se: arquiteturas com memória distribuída e com memória compartilhada. Processadores vetoriais utilizam múltiplos elementos de processamento acoplados em múltiplos barramentos. Com eles é possível realizar operações lógicas ou aritméticas, como, por exemplo, a adição de dois vetores de entrada produzindo um vetor de saída como resultado. Em arquiteturas SIMD, uma unidade de controle difunde as instruções, que são executadas paralelamente por todos os processadores, cada qual usando seu próprio dado de sua própria memória. Essas arquiteturas

são adequadas para processamento rápido de imagens em baixo nível, no qual se enquadra a morfologia matemática. Como exemplo de arquitetura SIMD, pode-se citar o sistema AIS-5000 (GASTERATOS, 2002), formado por um arranjo de até 1024 elementos de processamento com memória local e conexões com dois elementos vizinhos (Figura 5.1a). Nesta arquitetura, os elementos de processamento são programáveis através de uma tabela verdade e podem executar operações *booleanas*, aritméticas e de vizinhança. Na Figura 5.1b, mostra-se um exemplo de erosão binária. Na Figura 5.1c, apresentam-se os dados utilizados no exemplo. Outros exemplos de arranjos paralelos são o Illiac IV, que contém um arranjo de 8x8 elementos de processamento e o MPP, contendo 16384 processadores em uma matriz de 128x128 (GASTERATOS, 2002).

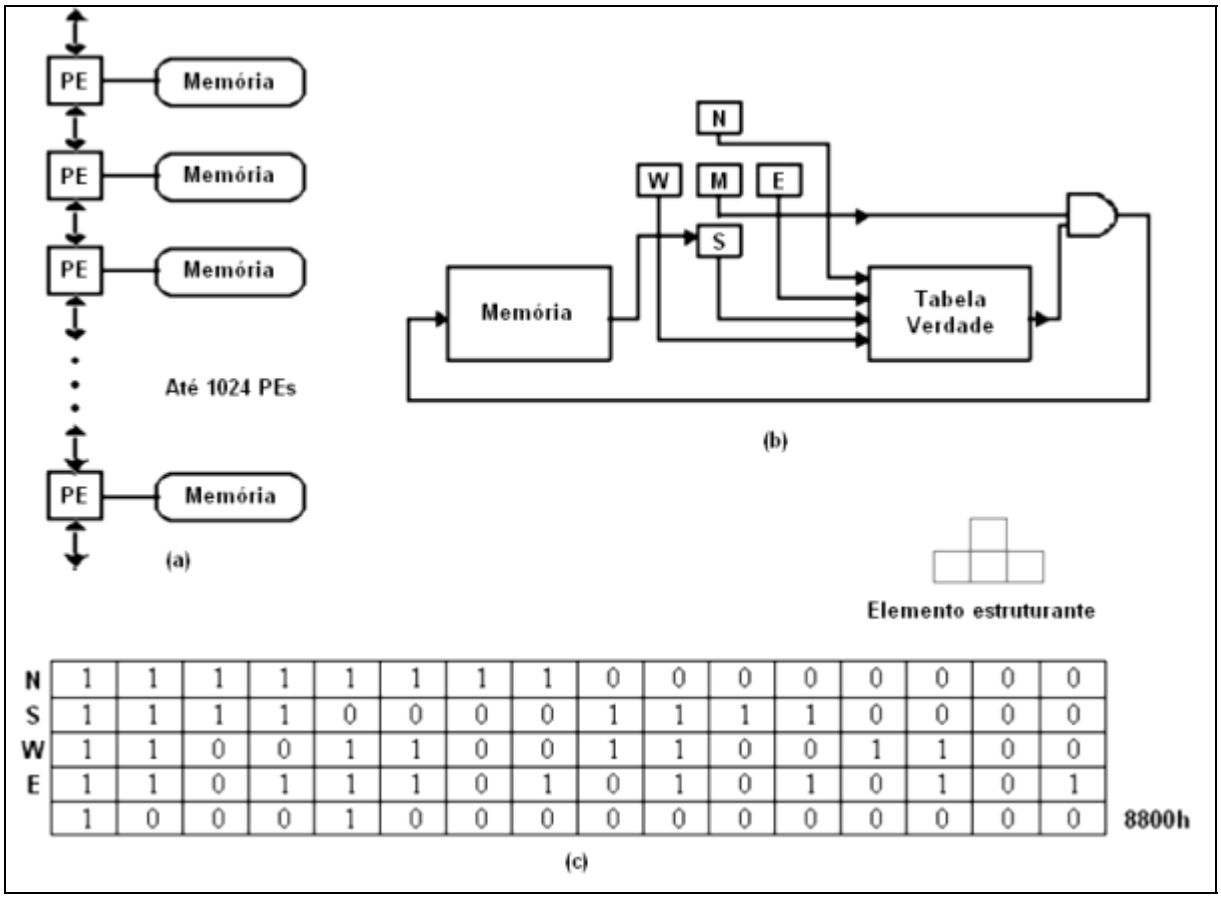


Figura 5.1. Sistema AIS-5000. (a) Topologia do arranjo paralelo. (b) Implementação de erosão binária. (c) Elemento estruturante e Tabela verdade utilizada no exemplo

Arquiteturas sistólicas ou arranjos sistólicos são as arquiteturas mais amplamente utilizadas para processamento de imagens em baixo nível. Tais arranjos são utilizados para implementar um certo algoritmo em *hardware*. Nesses arranjos, os elementos de processamento são rígidos. Os dados fluem pela memória de forma rítmica, passando pelos elementos de processamento e retornando novamente à memória. A sincronização dos dados é obtida através de um *clock* global que controla todos os elementos de processamento.

As arquiteturas MIMD são divididas basicamente em arquiteturas de memória-distribuída (anel, malha, pirâmide e hipercubo) e arquiteturas de memória-compartilhada. As arquiteturas de memória-distribuída possuem diversos processadores interconectados de várias maneiras. Cada processador se comunica com sua própria memória local e também com processadores adjacentes. Na Figura 5.2, apresentam-se alguns exemplos de arquiteturas de memória-distribuída. Na Figura 5.2a, tem-se uma arquitetura com topologia em anel. Esta topologia é apropriada para um número limitado de nós e é adequada para algoritmos que não demandem grande quantidade de dados. Na Figura 5.2b, apresenta-se uma arquitetura com topologia mais sofisticada, que é a topologia em malha. Esta pode ser aplicada a algoritmos que envolvam matrizes, tais como os usados em processamento de imagens 2D. Uma das desvantagens dessa topologia é a necessidade de comunicação entre dois nós que não pertençam à mesma vizinhança. Uma extensão para esta abordagem é a topologia piramidal. Esta é formada por diversos arranjos em malha, sobrepostos e de resoluções menores (Figura 5.2c).

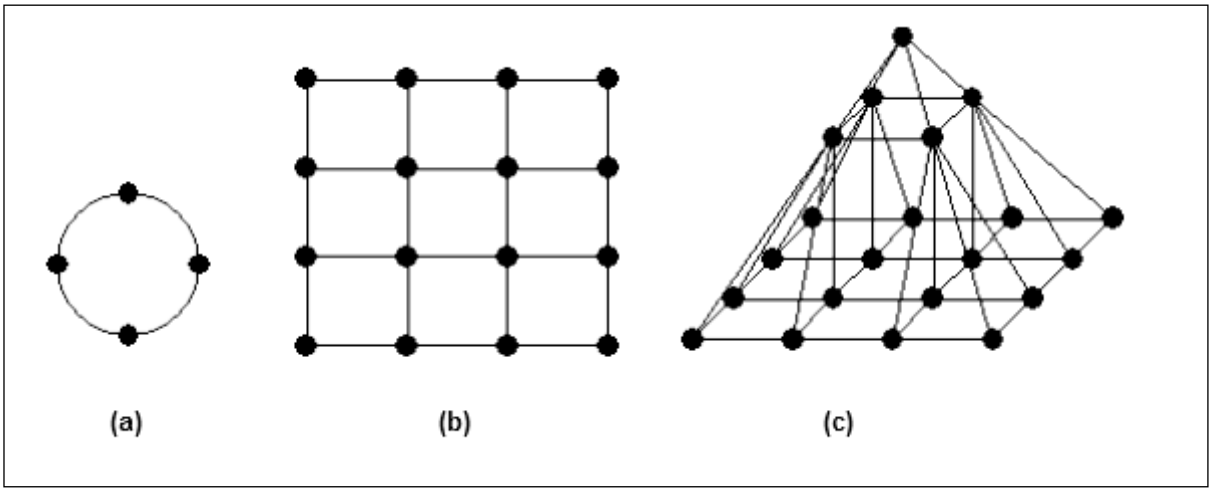


Figura 5.2. Topologias de memória distribuída. (a) Topologia em anel. (b) Topologia em malha. (c) Topologia piramidal

A topologia piramidal resolve o problema encontrado na topologia em malha, de acordo com a discussão anterior. Outra topologia mais complexa é a de hipercubo. Nessa topologia, os nós são localizados nos cantos de um cubo de dimensão *n*. Uma máquina de dimensão *d*, construída nessa arquitetura, têm 2^d nós, com cada um deles sendo conectado aos seus *d* nós vizinhos. Na Figura 5.3, apresentam-se alguns exemplos.

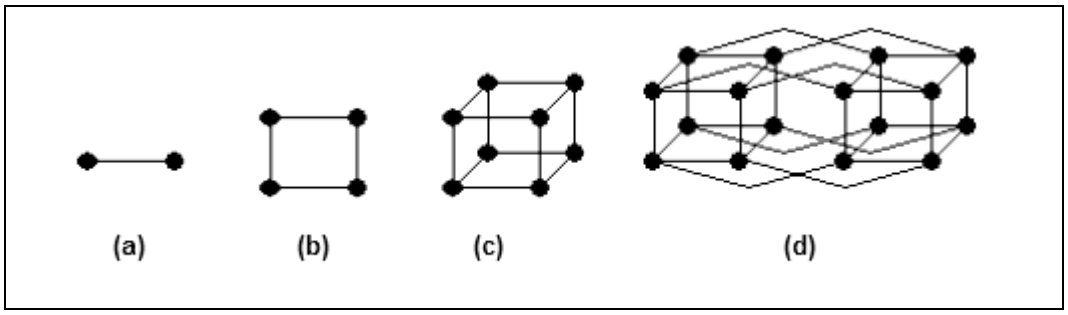


Figura 5.3. Topologias hipercubo. (a) Hipercubo 1D. (b) Hipercubo 2D. (c) Hipercubo 3D. (d) Hipercubo 4D

Em arquiteturas de memória-compartilhada, os elementos de processamento são controlados por uma memória comum. Basicamente, existem dois modos de interfacear os elementos de processamento com a memória. O modo mais simples consiste de um barramento rápido onde os elementos de processamento e a memória são ligados (Figura

5.4a). O modo mais sofisticado possui uma rede de interface multi-camada, que é responsável por interconectar os elementos de processamento com pedaços de memória (Figura 5.4b). A principal vantagem de máquinas com memória-compartilhada é a facilidade de implementação dessas. Elas são usadas em processamento de imagens em alto nível. Embora essas máquinas sejam aplicáveis ao processamento paralelo de imagens, no qual cada elemento de processamento é responsável por uma dada fração de imagem, elas não são tão suficientes quanto às SIMD. Essas arquiteturas não são eficientes quanto à velocidade.

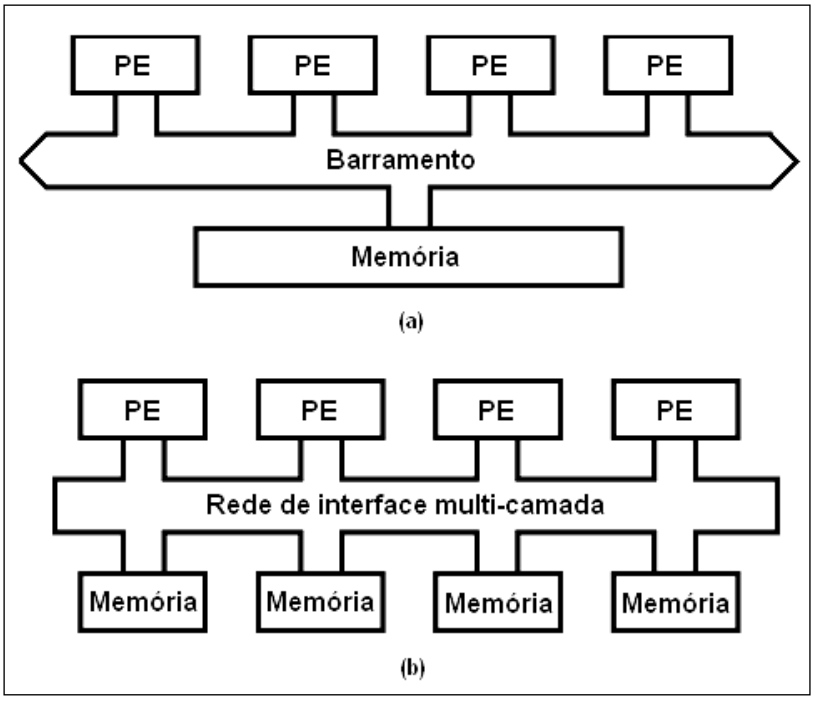


Figura 5.4. Arquiteturas de memória-compartilhada. (a) Barramento rápido. (b) rede de interface multi-camada

5.2.2 Estruturas de *Hardware* Especializadas para Processamento Morfológico de Imagens

Além das arquiteturas de propósito geral discutidas na subseção anterior, outras arquiteturas particularmente dedicadas para processamento morfológico de imagens foram sugeridas na literatura. As técnicas para implementação de estruturas de *hardware*

especializadas em processamento morfológico podem ser classificadas nas seguintes categorias:

- Implementações digitais: técnica de decomposição por *threshold*, técnica *bit serial*, implementações de arranjos sistólicos e implementações assíncronas.
- Implementações analógicas.
- Implementações Ópticas.

A metodologia da técnica de decomposição por *threshold*, introduzida por (FITCH et al., 1985), levou ao desenvolvimento da teoria de filtros *stack*. Um filtro *stack* é qualquer filtro que pode ser definido através da decomposição por *threshold* e de uma função *booleana* positiva (PBF). De acordo com esta aproximação, um sinal multinível que obtém seus valores no conjunto {0, 1,..., M-1}, é decomposto em M-1 sinais binários. Cada um desses sinais é filtrado através de uma PBF. A mesma PBF é aplicada a cada sinal binário. A saída do filtro *stack* é composta pela soma de todos os dados binários após a filtragem. Uma PBF é uma função *booleana* que é implementada como uma soma lógica de produtos lógicos. Na Figura 5.5, ilustra-se essa técnica. No exemplo, um sinal x unidimensional é decomposto em três sequências binárias, x_1, x_2 e x_3. Cada seqüência é filtrada por um filtro de *max* binário com uma janela de tamanho 3x1, isto é, uma porta OR de três entradas. Assim, as saídas binárias, y_1, y_2 e y_3 , são obtidas respectivamente de acordo com a fórmula: $y_i(t)=x_i(t-1)+x_i(t)+x_i(t+1)$, i=(1..3). As saídas são então somadas por colunas e o resultado final, y, é obtido.

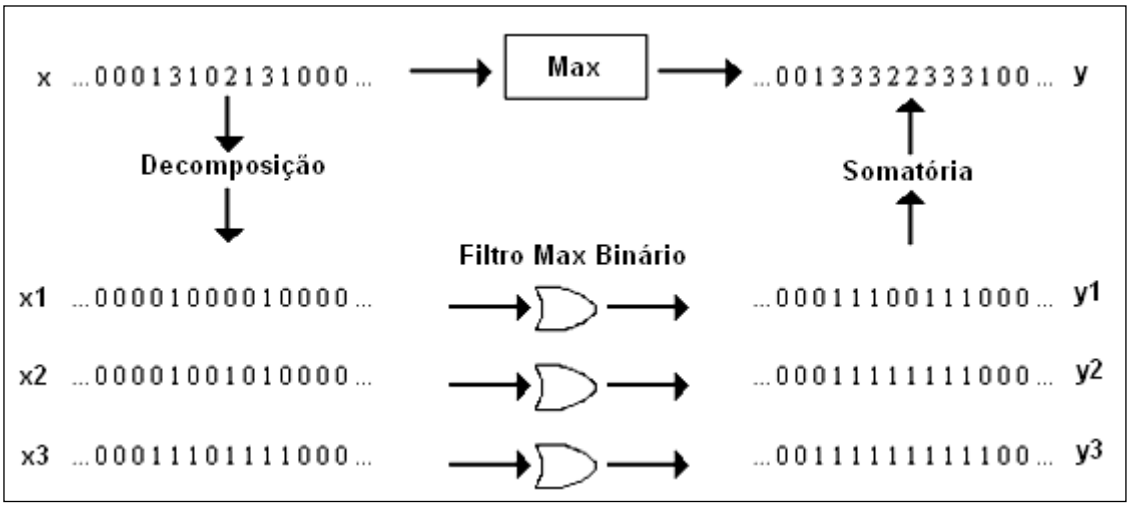


Figura 5.5. Técnica de decomposição por *threshold* para um sinal unidimensional

Essa técnica foi utilizada por (SHIH; MITCHELL, 1989) em morfologia clássica com a finalidade de decompor problemas complexos em problemas mais simples, ou seja, imagens em níveis de cinza são decompostas em múltiplas imagens binárias, onde estas são processadas em paralelo por meio de um *hardware* dedicado, aumentando-se assim o desempenho do processamento morfológico. Como exemplo dessa técnica, considere o algoritmo de dilatação que funciona da seguinte maneira: primeiramente é preciso encontrar os valores máximos de níveis de cinza na imagem e no elemento estruturante. Assim, seja *P* o maior valor de intensidade na imagem e *Q* o maior valor de intensidade no elemento estruturante. A imagem deve ser decomposta em *P* imagens binárias e o elemento estruturante em *Q+1* imagens binárias. Então, os pares de conjuntos binários resultantes da decomposição devem ser inseridos no estágio de dilatação da seguinte forma: a primeira imagem binária e o Q_{th} elemento estruturante decomposto; a segunda imagem e o Q_{th} elemento estruturante, a segunda imagem e o elemento estruturante *(Q-1)*; a terceira imagem e o Q_{th} elemento estruturante, a terceira imagem e o $(Q\text{-}1)_{th}$ elemento estruturante, a terceira imagem e o $(Q\text{-}2)_{th}$ elemento estruturante, e assim por diante. Considerando-se que cada par é processado independentemente, essas operações podem ser implementadas em paralelo. Após a saída do

estágio de dilatação, os conjuntos resultantes devem ser somados *bit* a *bit*, obtendo-se assim *P* saídas. Se $P \geq Q$, então, acrescenta-se 1 a cada posição da $(Q+2)_{th}$ saída, acrescenta-se 2 a cada posição da $(Q+3)_{th}$ saída, e assim por diante. O resultado final é obtido através da seleção dos valores máximos de cada posição das *P* saídas e do acréscimo de *Q* a cada posição. Uma implementação em ASIC para dilatações em níveis de cinza, de acordo com a técnica descrita anteriormente, é apresentada em (ANDREADIS et al., 1996). Nesta implementação, as resoluções de imagem e do elemento estruturante são limitadas a 4 bits. Este é um dos principais problemas dessa técnica, no qual a complexidade do *hardware* cresce de acordo com $O(2^{2b}+2^{b})$, onde *b* é a resolução de *pixel*s. Portanto, a aplicação dessa técnica para imagens de alta resolução se torna impraticável. Outras técnicas surgiram com o propósito de reduzir a complexidade do *hardware* envolvido, como, por exemplo, a técnica híbrida. Nesta, a decomposição por *threshold* não é aplicada a ambos, o elemento estruturante e a imagem; primeiramente é realizada uma operação aritmética entre o elemento estruturante e a imagem, e a partir do resultado dessa operação se aplica a técnica de decomposição por *threshold*. Portanto, a complexidade do *hardware* é reduzida a menos da metade.

Outra técnica para encontrar de forma eficiente os valores de máximo ou de mínimo em uma operação morfológica, conhecida como *bit serial*, é apresentada em (CHEN, 1989). Muitos pesquisadores implementaram este algoritmo em VLSI para executar as operações básicas de morfologia matemática (DIAMANTARAS; KUNG, 1997; GASTERATOS et al., 1996; LUCK; CHAKRABATY, 1995). De acordo com o algoritmo original, os diferentes *bits* de *n* números são processados por elementos de processamento dedicados em diferentes estágios de um arranjo sistólico. Suponha que se deseje encontrar os valores de máximo ou de mínimo entre *n* números apresentados. No primeiro estágio do processo, os *bits* mais significativos desses números são processados. Os números cujos *bits* mais significativos são 1 ou 0, são candidatos a serem máximo ou mínimo, assim, desprezam-se os demais números

para os próximos estágios do processo. Um sinal de *flag*, *f*, classifica os números em duas categorias. Mais especificamente, se *f*=1, o número é um candidato para o próximo estágio do processo, caso contrário, o número é excluído do processo. Este procedimento é então repetido nos estágios subseqüentes do processo para os *bits* de menor ordem dos números candidatos. Uma ilustração deste procedimento para encontrar o valor máximo de cinco números de 4 *bits* pode ser vista na Tabela 5.1. Nesta, $x_i(i=1..5)$ são os cinco números, $b_{j,i}$ é o j_{th} bit da representação binária de x_i e $f_{j,i}$ é o flag do i_{th} número no j_{th} estágio.

TABELA 5.1 – Exemplo ilustrativo do algoritmo *bit serial*

i	x_i	$f_{0,i}$	$b_{1,i}$	$f_{1,i}$	$b_{2,i}$	$f_{2,i}$	$b_{3,i}$	$f_{3,i}$	$b_{4,i}$	$f_{4,i}$
1	3	1	0	0	0	0	1	0	1	0
2	5	1	0	0	1	0	0	0	1	0
3	11	1	1	1	0	0	1	0	1	0
4	12	1	1	1	1	1	0	1	0	1
5	8	1	1	1	0	0	0	0	0	0
m=	12	m_1=	1	m_2=	1	m_3=	0	m_4=	0	

As arquiteturas mais comuns em processamento morfológico de imagens são as *pipeline*. Uma arquitetura *pipeline* é adequada para elementos estruturantes que obedeçam à regra da cadeia, uma vez que cada estágio da máquina *pipeline* opera através de um elemento estruturante pequeno. Como exemplos de arquiteturas morfológicas *pipeline*, têm-se:

- *Texture Analyser System* (TAS) (KLEIN; SERRA, 1972), que podia executar operações binárias morfológicas em um *pipeline* de três estágios. Originalmente era suportado um único elemento estruturante hexagonal, mas as outras versões podiam executar operações lógicas mais gerais.
- Cytocomputer: desenvolvido no *Environmental Research Institute of Michigan* para processamento de imagens biomédicas (STERNBERG, 1983). O Cytocomputer é constituído de um *pipeline* serial onde cada estágio executa uma

> transformação simples de uma imagem inteira. Oitenta e oito estágios executam operações lógicas e vinte e cinco outros executam processamento numérico. Registradores de deslocamento dentro de cada estágio armazenam duas linhas adjacentes de *n pixel*s e registradores de janela mantêm os dados que constituem a entrada 3x3 necessária à função de transição. Esses registradores endereçam uma LUT com 512 possibilidades de valores que serve para implementar a função de transição (GERRITSEN; VERBEEK, 1984; PRESTON, 1983).

Nas próximas subseções, alguns exemplos de arquiteturas *pipeline* serão destacados mais detalhadamente.

Implementações analógicas para as operações morfológicas podem ser muito atrativas, considerando-se que estas oferecem altas velocidades em *chips* mais compactos, além do mais, não necessitam de conversões A/D e D/A, logo, podem ser acopladas diretamente nos sensores e câmeras, assim, é possível construir instrumentos mais inteligentes (SISKOS et al., 1998).

Uma arquitetura opto-eletrônica para processamento morfológico de imagens foi proposta em (MICHAEL; ARRATHOON, 1997). Essa arquitetura usa o princípio de *pipelining*, onde seus estágios são implementados através de PLAs baseados em fibra óptica (OPLAs). Tais arranjos são caracterizados por altos fatores de *fan-in* e *fan-out*, assim, é possível implementar operações morfológicas com elementos estruturantes grandes sem a necessidade de decomposição, logo, a arquitetura projetada possuirá poucos estágios, entretanto, tal abordagem ainda é cara. Os OPLAs são baseados em operações lógicas OR, onde estas são implementadas por meio de fibras, detectores e inversores ópticos. Na Figura 5.6, apresenta-se a Tabela de programação de um OPLA para a operação de erosão binária por um elemento estruturante 2x2.

Bits de Controle				Entradas				
C1	C2	C3	C4	Ai,j	Ai,j+1	Ai+1,j	Ai+1,j+1	Ei,j
0	0	0	1	—	—	—	1	1
0	0	1	0	—	—	1	—	1
0	0	1	1	—	—	1	1	1
0	1	0	0	—	1	—	—	1
0	1	0	1	—	1	—	1	1
0	1	1	0	—	1	1	—	1
0	1	1	1	—	1	1	1	1
1	0	0	0	1	—	—	—	1
1	0	0	1	1	—	—	1	1
1	0	1	0	1	—	1	—	1
1	0	1	1	1	—	1	1	1
1	1	0	0	1	1	—	—	1
1	1	0	1	1	1	—	1	1
1	1	1	0	1	1	1	—	1
1	1	1	1	1	1	1	1	1

Figura 5.6. Tabela de programação do OPLA para a operação de erosão binária por um elemento estruturante 2x2

Sejam $a_{i,j}$ e $e_{i,j}$ *pixel*s na imagem original e na erodida, respectivamente na linha *i* e na coluna *j*. Assim, o *pixel* $e_{i,j}$ será 1 (isto é, claro) se os *pixel*s claros do elemento estruturante e da imagem coincidirem. Para um padrão 2x2, é possível obter 16 elementos estruturantes diferentes. Quatro bits de controle são utilizados para escolher o elemento estruturante requerido. Na Figura 5.7, mostra-se o padrão de conexão em fibra-óptica para a operação de erosão descrita anteriormente.

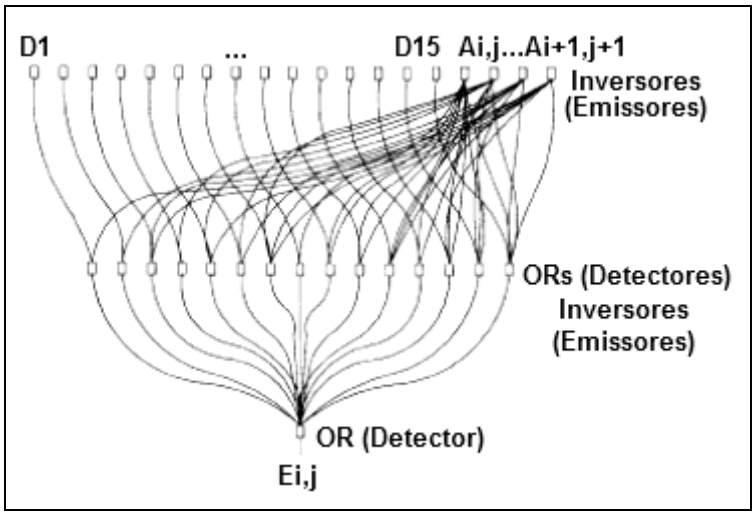


Figura 5.7. Padrão de conexão em fibra-óptica para a operação de erosão

A arquitetura *pipeline* opto-eletrônica proposta para processamento morfológico de imagens binárias pode ser vista na Figura 5.8.

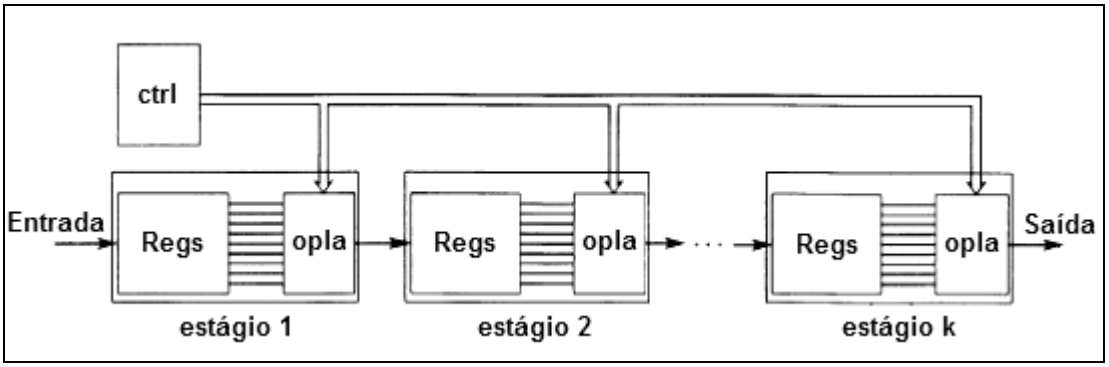


Figura 5.8. Arquitetura *pipeline* utilizando OPLAs, proposta para processamento morfológico de imagens binárias

5.2.2.1 Arquiteturas *Pipeline* Morfológicas para Processamento de Imagens Binárias

Em uma arquitetura *pipeline* para processamento de vídeo, é conveniente apresentar os dados no formato *raster*, de acordo com o padrão de varredura de uma câmera de vídeo convencional. Em (ABBOTT et al., 1988), propõe-se uma arquitetura *pipeline* para processamento morfológico de imagens binárias, onde cada estágio opera através de um elemento estruturante consistindo da origem e de um ponto arbitrário *(r,c)* no plano de imagem. Para essa arquitetura, têm-se as seguintes definições:

- *R:* $E^2 \rightarrow E^1$: função que mapeia uma imagem A_{NxM}, onde *N* é o número de linhas e *M* é o número de colunas, em um arranjo *R[A]* de *pixel*s.
- *R[A](t):* representação de um *pixel* particular.
- *R[A](rM+c):* representação do valor do *pixel* da *r*-ésima linha e *c*-ésima coluna de *A*, onde: $0 \leq r \leq N-1$ e $0 \leq c \leq M-1$.
- *R[A](t-k)*: imagem *A* deslocada de *k* unidades de tempo.
- *R[A](u):* duração de *R[A]*, onde: $0 \leq u \leq NM-1$.

Assim, a dilatação de uma imagem binária por um elemento estruturante composto pela origem e por um ponto arbitrário *(r,c)* no plano de imagem pode ser realizada da seguinte maneira: deslocam-se os dados de entrada por $|Mr+c|$ unidades de tempo; se $Mr+c>0$ (caso causal), a saída é obtida através de um OR lógico da versão deslocada e mascarada da entrada de dados com a versão não deslocada de entrada, seguido do janelamento do resultado pela duração da imagem original; se $Mr+c<0$ (caso não causal), a imagem deslocada se tornará a imagem primária; logo, efetuando-se um OR lógico da versão deslocada com a versão mascarada da imagem não deslocada, seguido do janelamento do resultado pela duração da entrada de dados deslocada, o resultado desejado é obtido. Também, a idéia apresentada anteriormente pode ser aplicada à erosão, assim, a erosão de uma imagem binária por um elemento estruturante composto pela origem e por um ponto arbitrário *(r,c)* no plano de imagem pode ser efetuada da seguinte maneira: deslocam-se os dados de entrada por $|Mr+c|$ unidades de tempo; se $Mr+c>0$ (caso não causal), o resultado desejado é obtido através de um AND lógico da versão mascarada dos dados de entrada com a entrada deslocada, seguido do janelamento do resultado pela duração da imagem deslocada; se $Mr+c<0$ (caso causal), o resultado desejado é obtido por meio de um AND lógico da entrada não deslocada com a versão mascarada de entrada, seguido do janelamento do resultado pela duração da imagem de entrada. A arquitetura *pipeline* conceitual para realizar essas operações está ilustrada na Figura 5.9. Tais idéias podem ser generalizadas para elementos estruturantes de *n*-pontos, entretanto, a complexidade do *hardware* envolvido também aumenta.

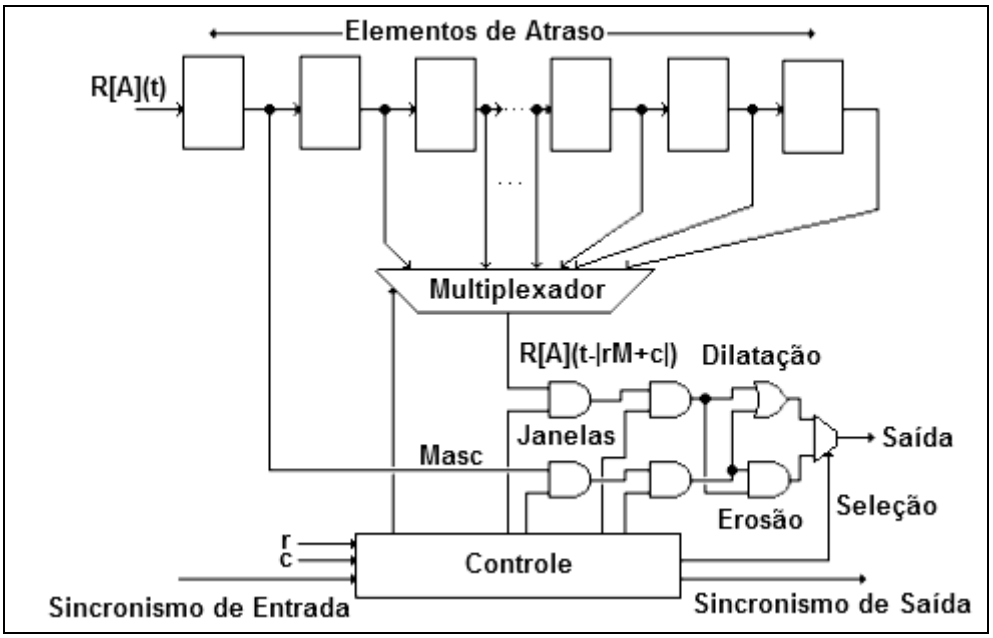


Figura 5.9. Arquitetura *pipeline* conceitual para operações morfológicas

Em (KOJIMA et al., 1994), apresenta-se uma arquitetura para processamento morfológico de imagens binárias onde a complexidade das operações básicas de morfologia pode ser reduzida através do uso de um algoritmo de decomposição dimensional do elemento estruturante 2-D. O algoritmo apresentado possui as seguintes características:

- Não há restrições quanto ao tamanho do elemento estruturante.
- Possibilidade de o elemento estruturante possuir qualquer forma arbitrária.
- Alta velocidade de processamento.

Considerando-se um elemento estruturante circular como exemplo, o algoritmo supracitado se divide nos seguintes passos:

- De acordo com a Figura 5.10(a), o elemento estruturante circular deve ser aproximado através de retângulos, formando-se assim o elemento estruturante *B*.
- Conforme mostrado na Figura 5.10(b), os retângulos são definidos como b_i (i=1,2,..n), onde: $B = \bigcup_{i=1}^{n} b_i$.

- Conforme ilustrado na Figura 5.10(c), cada b_i é decomposto em dois elementos estruturantes unidimensionais, assim, $b_i = b_{ix} \oplus b_{iy}$.

- Consequentemente, $B = \bigcup_{i=1}^{n}(b_{ix} \oplus b_{iy})$. Portanto, o elemento estruturante original B é decomposto em $2n$ elementos estruturantes unidimensionais.

De acordo com as discussões acima, tem-se: $A \oplus B = \bigcup_{i=1}^{n}(A \oplus b_{ix} \oplus b_{iy})$ e $A \Theta B = \bigcap_{i=1}^{n}(A \Theta b_{ix} \Theta b_{iy})$. Na Figura 5.10(c), l_{ix} e l_{iy} são os comprimentos dos elementos estruturantes unidimensionais; s_{ix} e s_{iy} representam a distância entre a origem e o ponto extremo do elemento estruturante 1-D mais próximo à origem. Tais parâmetros são utilizados pelo *hardware*.

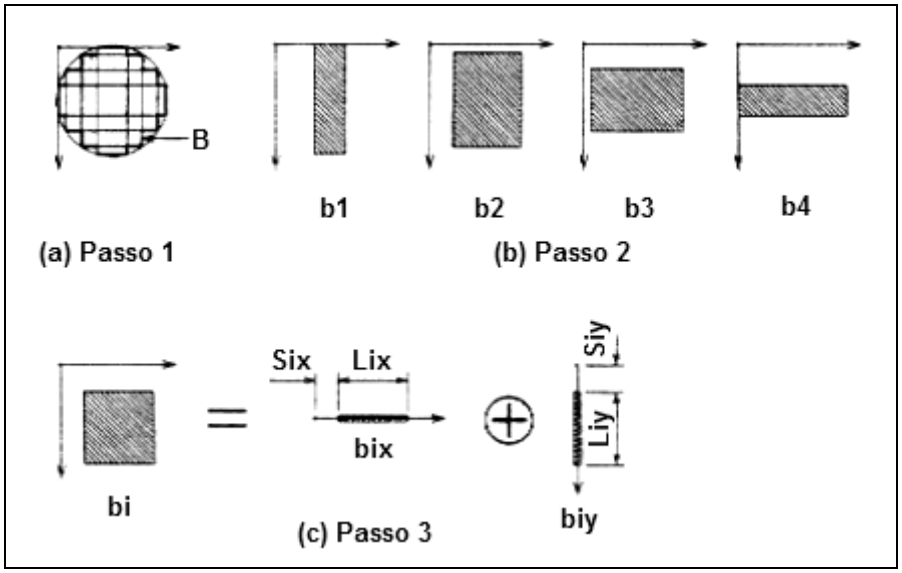


Figura 5.10. Decomposição do elemento estruturante

Nas Figuras 5.11 e 5.12 estão ilustrados, respectivamente, o circuito de dilatação 1-D e o circuito conversor X-Y de direção de varredura. Na Figura 5.13 é apresentado um exemplo contendo um diagrama de tempo de um sinal de entrada e o respectivo sinal de saída do

circuito de dilatação 1-D. Assim, o sinal de saída permanecerá em nível alto quando o sinal de entrada deslocado for alto e durante a contagem decrescente do contador, determinada pelo parâmetro *l*, após a descida do sinal de entrada.

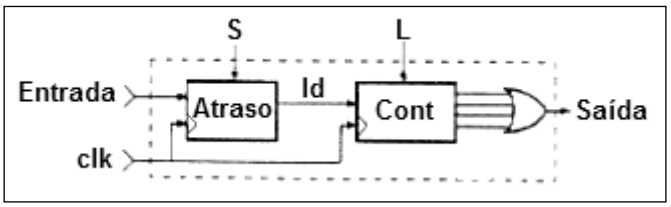


Figura 5.11. Circuito de Dilatação 1-D

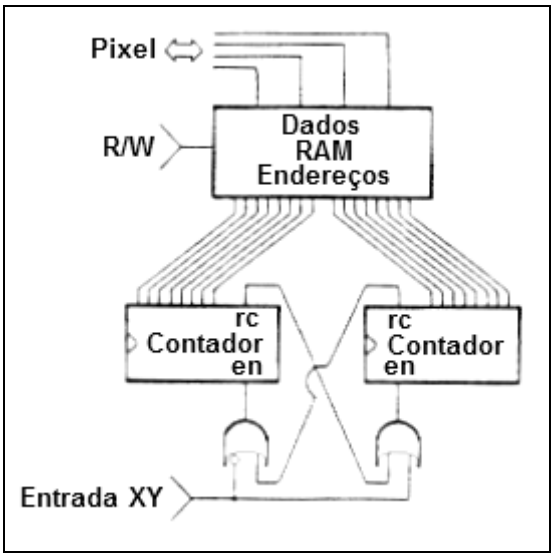


Figura 5.12. Circuito conversor de direção de varredura

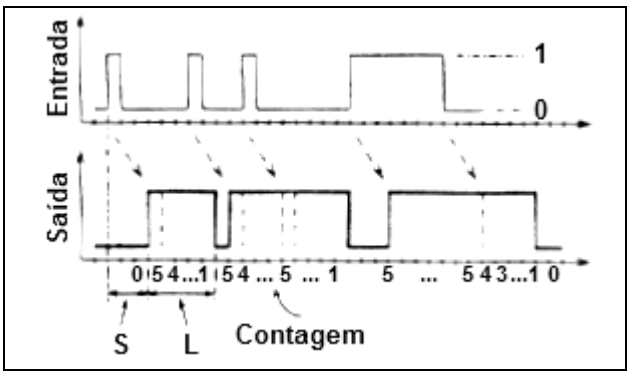


Figura 5.13. Exemplo de Diagrama temporal do circuito de dilatação 1-D

Quando o circuito conversor de varredura se encontra no modo *X*, têm-se: $S=s_{ix}$ e $L=l_{ix}$. Neste caso, a varredura é realizada na direção *x*. Idem para a direção *y*. No circuito de dilatação 1-D, o sinal de entrada sequencial é deslocado pela variável de atraso *S*, e então é transferido à entrada *ld* do contador decrescente. Quando *ld* for igual a 1, o valor de *L* é carregado no contador decrescente, e quando este mudar para nível 0, a saída do contador será decrementada através de cada pulso de *clock* até que a sua saída seja 0. Assim, a saída do circuito de dilatação 1-D permanecerá em nível lógico 1 após a carga de *ld* e manterá este estado até que o valor do contador seja nulo. O circuito conversor de varredura faz a varredura da imagem nas direções *x* e *y*. Dois contadores de 8 bits são utilizados para gerar os endereços às RAMs. Assim, é possível lidar com imagens de 256x256 *pixel*s. A comutação de direção é realizada através da troca dos bytes mais e menos significativos do barramento de 16 bits da RAM. De acordo com a Figura 5.12, esta operação pode ser implementada através do uso de contadores de 8 bits com pinos de *ripple carry* (*rc*) e *enable* (*en*). A dilatação 2-D é obtida paralelizando-se *n*-sistemas de dilatação 1-D e memórias conforme a Figura 5.14. Nesta Figura, a imagem de entrada é varrida na direção *x*, dilatada pelos elementos estruturantes unidimensionais e os resultados obtidos são armazenados nas RAMs. Em seguida, o processo de varredura das RAMs se dá na direção *y* e novamente as imagens são dilatadas pelos circuitos de dilatação unidimensionais. Finalmente, o resultado da dilatação 2-D pode ser obtido através de uma operação binária OR entre cada saída dos circuitos de dilatação 1-D. O processo de erosão é obtido de maneira análoga, de acordo com a Figura 5.15. Um dos problemas encontrados por esse algoritmo é que se a forma do elemento estruturante necessitar de muitos retângulos para a sua aproximação, o que não é difícil na prática, seu desempenho computacional diminuirá.

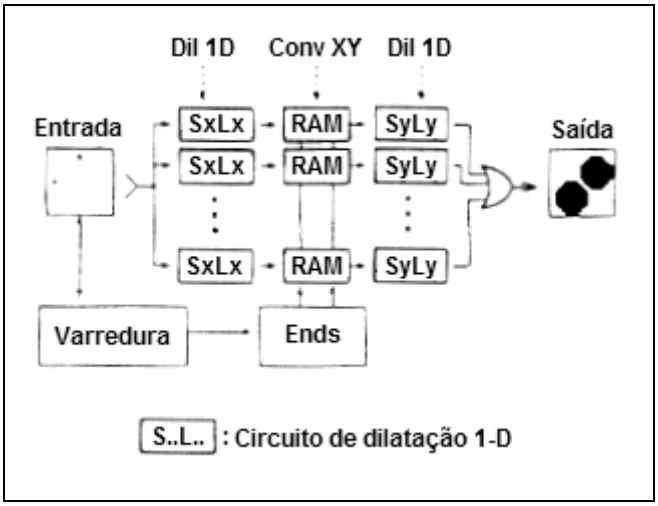


Figura 5.14. Arquitetura para dilatação binária 2-D

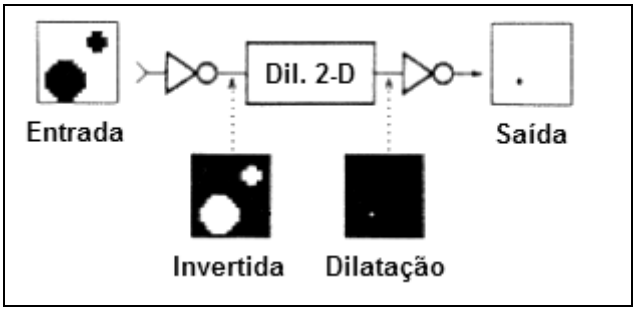


Figura 5.15. Circuito para erosão binária

Em (PEDRINO, 2003; PEDRINO; RODA, 2004a), apresenta-se a implementação de uma arquitetura *pipeline* dedicada para dilatação e erosão de imagens binárias em tempo real utilizando CPLDs e FPGAs. A arquitetura desenvolvida é capaz de processar imagens binárias de 512x512 *pixel*s de resolução espacial. As imagens digitais são obtidas através da digitalização de um sinal de vídeo composto padrão fornecido por uma câmera CCD comercial (PEDRINO; RODA, 2004b). Cada estágio da arquitetura é responsável pelo processamento de um subconjunto menor de um dado elemento estruturante arbitrário. Esses estágios foram implementados através de PLDs, permitindo-se assim a reprogramação da forma e tamanho dos elementos estruturantes e também do número de estágios da arquitetura. Uma unidade de controle, implementada através de um CPLD, gera os sinais de sincronismo e

controle para cada estágio da arquitetura, assim, é possível realizar operações morfológicas mais complexas. A arquitetura descrita está ilustrada na Figura 5.16.

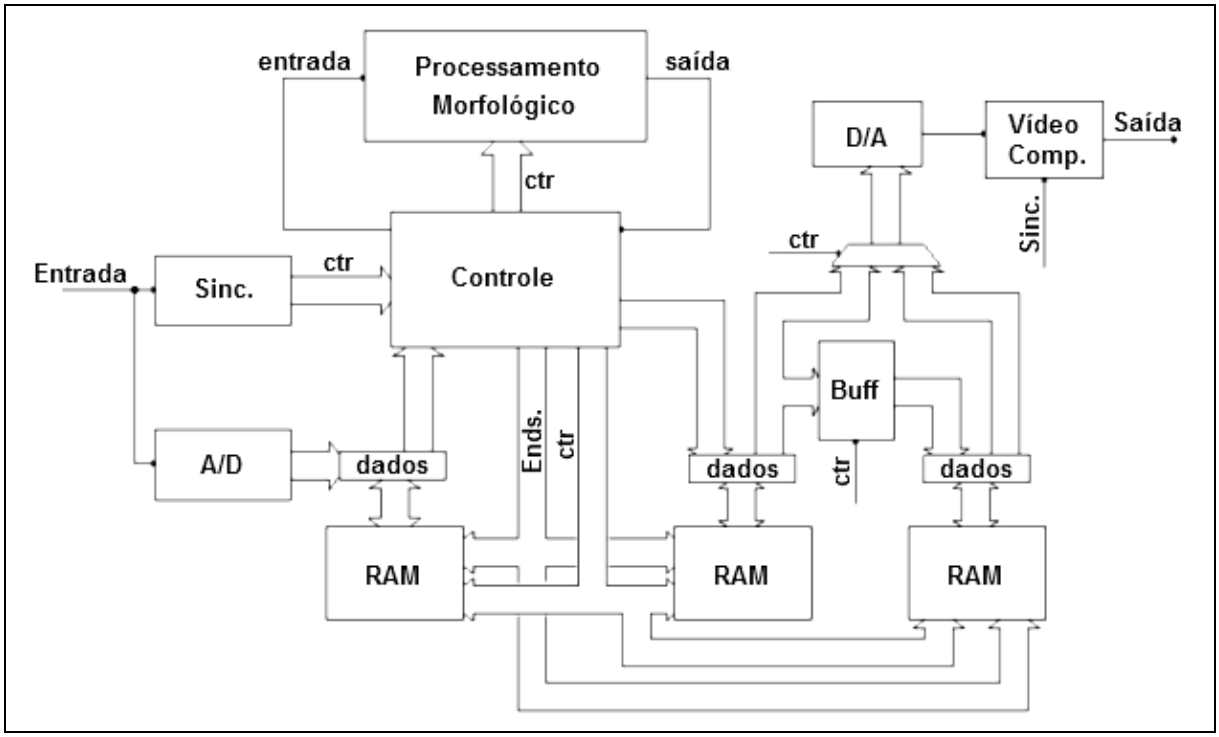


Figura 5.16. Diagrama em blocos da arquitetura *pipeline* reconfigurável para processamento morfológico de imagens binárias em tempo real (PEDRINO, 2003)

5.3 *HARDWARE* RECONFIGURÁVEL PARA PROCESSAMENTO MORFOLÓGICO DE IMAGENS COLORIDAS

Na próxima subseção é proposta a idealização de uma arquitetura paralela para processamento morfológico de imagens coloridas em tempo real utilizando *hardware* reconfigurável. O algoritmo desenvolvido utiliza a distância *city-block* (GONZALEZ; WOODS, 1992) para a ordenação dos vetores de imagem, considerando-se que sua implementação em hardware é facilitada pela ausência da raiz quadrada presente na distância euclidiana. Também, apresenta-se o desenvolvimento de um sistema RGB para processamento de vídeo colorido.

5.3.1 Arquitetura *Pipeline* para Processamento Morfológico de Imagens Coloridas em Tempo Real

A arquitetura *pipeline* reconfigurável em *hardware* para processamento morfológico de imagens coloridas proposta nesta subseção utiliza o espaço de cores RGB, assim, neste trabalho também foi desenvolvido um sistema RGB para fornecer os *pixels* de imagem a esta arquitetura.

5.3.1.1 Sistema RGB para Aquisição e Processamento de Imagens Coloridas Digitais utilizando *Hardware* Reconfigurável

O sistema RGB para processamento de vídeo digital desenvolvido neste trabalho, foi baseado no sensor CMOS OV7620 de baixo custo da OmniVision (ELECTRONICS123, 2006). Por meio de uma interface dedicada, foi possível implementar esta plataforma através do uso do CPLD EPM7128SLC84-15 da família MAX 7000S da Altera. Com esta arquitetura, é possível implementar algumas tarefas simples envolvendo visão computacional, como, por exemplo, filtragem, limiarização, operações aritméticas e geométricas, entre outras operações. Neste projeto, também foi utilizado o *software* Quartus II *Web Edition Full 6.0* da Altera durante a fase de desenvolvimento. Por meio de um processo de interpolação de *pixel*s, foi possível dobrar a freqüência de quadros do sensor para 60 quadros/s com o objetivo de utilizar o mesmo em aplicações requerendo processamento de vídeo em tempo real. Além desta plataforma, foi desenvolvido um sistema microprocessado baseado no microcontrolador AT89C2051-24 da Atmel, com teclado e display de cristal líquido, para programação dos registradores internos ao sensor através da interface digital SCCBus.

5.3.1.1.1 Sensor OV7620

Na Figura 5.17 é possível ver uma ilustração do sensor colorido CMOS OV7620 utilizado neste projeto. Este sensor possui um arranjo de 664 x 492 *pixel*s, onde o tamanho de *pixel* é de 7.6 x 7.6 µm. Sua área efetiva de imagem é de 4.86 mm x 3.64 mm. Também, pode ser configurado para trabalhar com varredura progressiva ou entrelaçada. Possui exposição eletrônica de 500:1, 128 curvas de ajuste para correção *gamma*, iluminação mínima de 0.5 lux e potência de 120mW. Seus *pixel*s podem ser obtidos através de duas portas digitais de 8 bits de largura de dados. Essas portas suportam os seguintes formatos: YCrCb 4:2:2 de 60Hz e 16/8 bits, ZV, RGB *raw* de 16/8 bits e CCIR601/656.

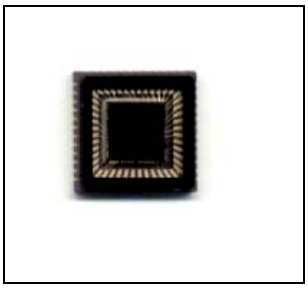


Figura 5.17. Sensor colorido CMOS OV7620

Uma interface SCCB implementada internamente ao *chip* fornece um modo de controle para suas funcionalidades. Na Figura 5.18 é possível ver o diagrama em blocos do sensor. De acordo com a Figura, o sensor possui um processador de sinais analógico, dois conversores AD de 10 bits de resolução, três multiplexadores de vídeo analógicos, um formatador digital de dados e duas portas de saída de vídeo (Y e UV). Os controles digitais incluem um bloco de temporização, um bloco de exposição e um bloco de balanço de branco. O filtro de cores do sensor é formado por um padrão de cores primárias RG/GB arranjadas em linhas alternadas.

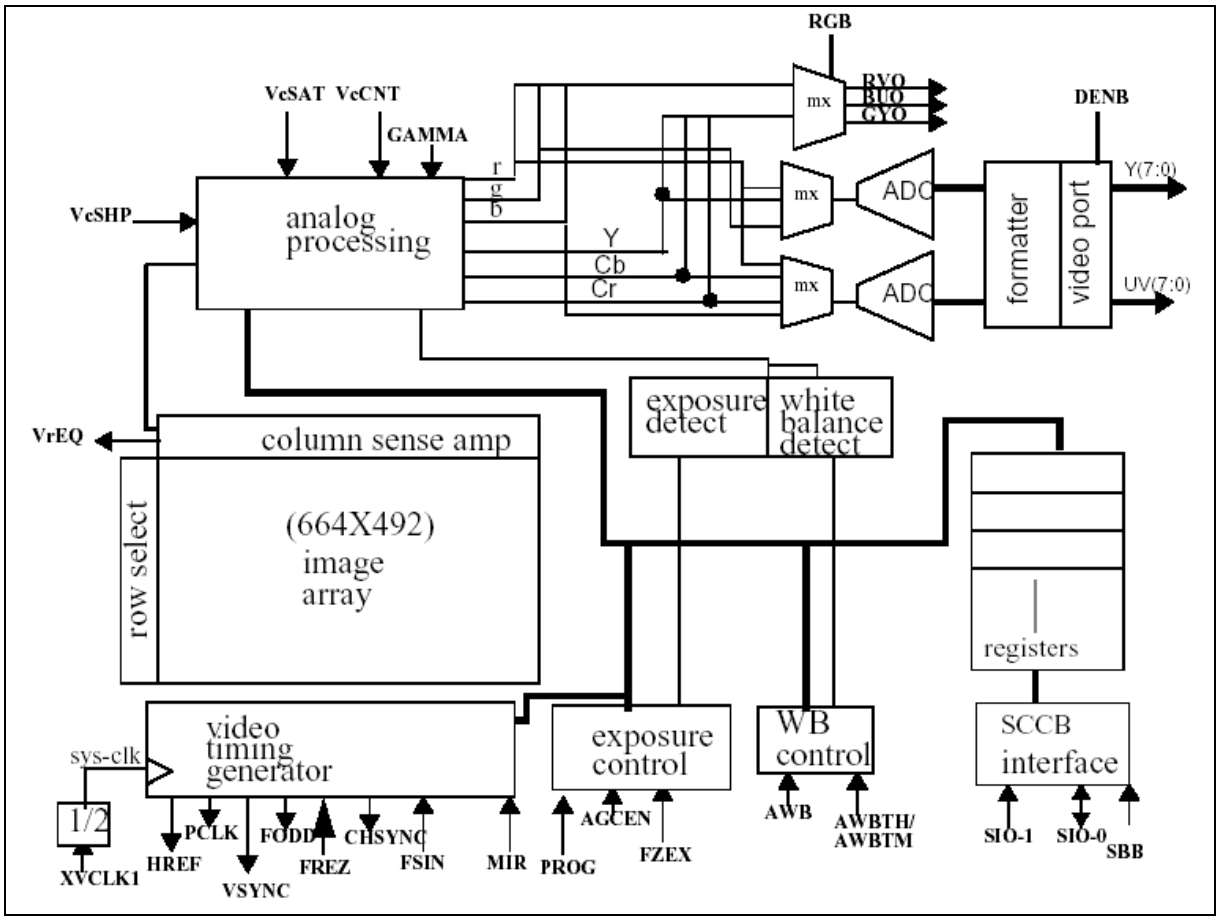


Figura 5.18. Diagrama em blocos do sensor OV7620 (OMNIVISION, 2000)

5.3.1.1.2 Protocolo SCCBus

O protocolo *Serial Camera Control Bus* (SCCB) foi definido e desenvolvido pela OmniVision Technologies para controlar a maioria das funções de sua família de sensores de vídeo (OMNIVISION, 2003). Os sensores da OmniVision somente trabalham como dispositivos escravos. Para controlar um ou mais dispositivos escravos é necessário que um dispositivo mestre seja conectado ao barramento SCCB. Na Figura 5.19, apresenta-se o diagrama funcional deste barramento.

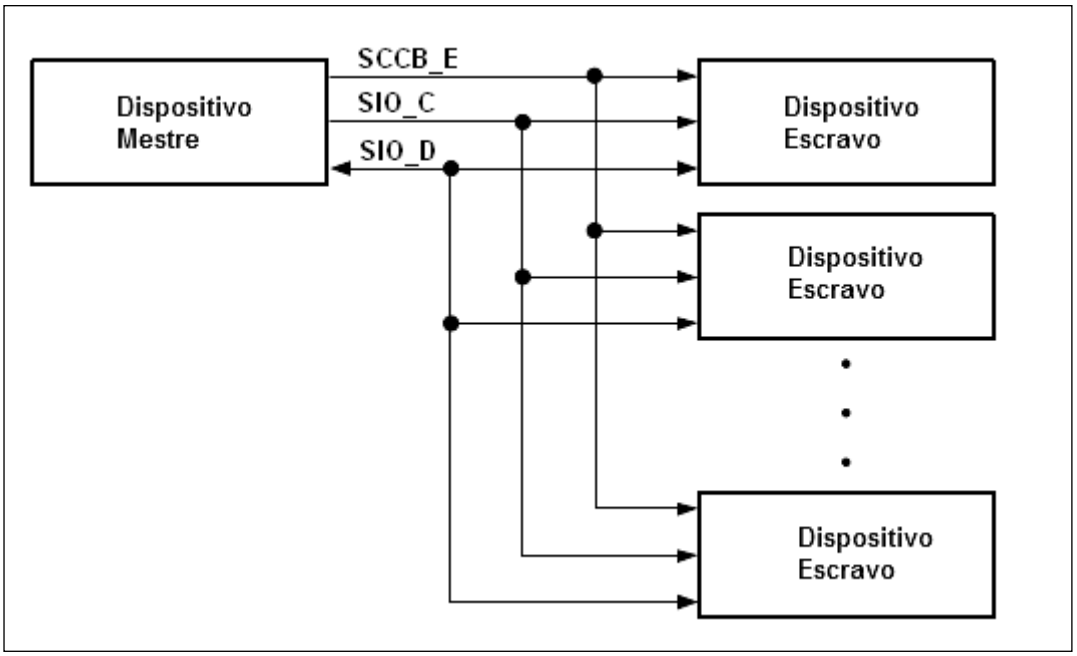


Figura 5.19. Diagrama funcional em blocos do barramento SCCB (Adaptado de OMNIVISION, 2003)

Neste projeto, conforme já sobredito, o dispositivo mestre foi implementado através do microcontrolador AT89C2051-24 da Atmel (ATMEL, 2006). De acordo com a Figura 5.19, o protocolo SCCB possui um barramento serial de três fios. O sinal *SCCB_E* controlado pelo dispositivo mestre é unidirecional e ativo baixo. Este sinal indica as condições de início e parada da transmissão de dados. Uma transição de nível alto para baixo indica o início de uma transmissão, enquanto uma transição de nível baixo para alto indica o fim de uma transmissão. Durante uma transmissão de dados, o sinal *SCCB_E* deverá permanecer em nível 0, caso contrário, o barramento se tornará ocioso. O sinal *SIO_C* controlado pelo dispositivo mestre também é unidirecional e ativo alto. O objetivo deste sinal de controle é indicar a transmissão de cada bit. Quando o barramento está ocioso, o dispositivo mestre atribui nível lógico 1 a este sinal. O sinal de dados *SIO_D* é bidirecional e pode ser controlado pelo dispositivo mestre e também pelos dispositivos escravos. Este sinal permanecerá flutuando quando o barramento estiver ocioso. Cada bit transmitido deverá estar estável durante o período de nível alto do sinal *SIO_C*. Na Figura 5.20 é possível visualizar o diagrama de tempos de transmissão de dados deste protocolo.

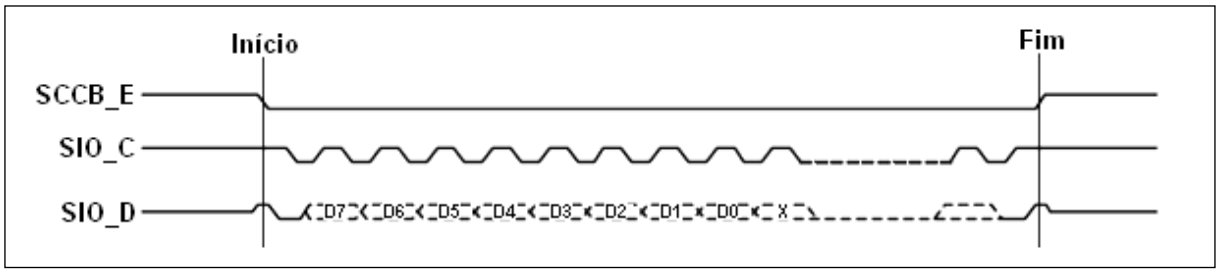


Figura 5.20. Diagrama de tempos de transmissão de dados do protocolo SCCB

Neste protocolo, um elemento básico de transmissão de dados é chamado de fase. Uma fase contém um total de 9 bits. O 8º bit é o bit de direção, indicando se o processo é de escrita ou de leitura. O 9º bit é irrelevante. Para cada transmissão é possível transmitir até 3 fases. O bit mais significativo de informação de fase é sempre transmitido em primeiro lugar. O ciclo de escrita é descrito através de 3 fases, conforme a Figura 5.21. O *ID Address* especifica o endereço do dispositivo escravo que deverá ser acessado pelo dispositivo mestre. O *Sub-address* especifica a localização do registrador do dispositivo escravo especificado. Os dados a serem escritos no registrador endereçado são especificados em *Write Data*.

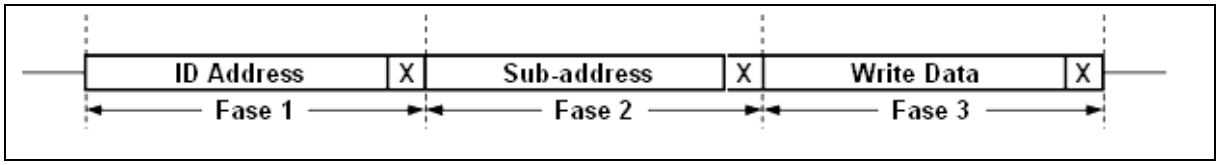


Figura 5.21. Ciclo de escrita do protocolo SCCB

O ciclo de leitura do protocolo SCCB é precedido por um ciclo de escrita de 2 fases que tem por objetivo identificar o registrador que deverá ser lido no dispositivo escravo. Na Figura 5.22, apresenta-se o ciclo de leitura deste protocolo. Entre as fases de escrita e leitura se insere uma seqüência de *restart*.

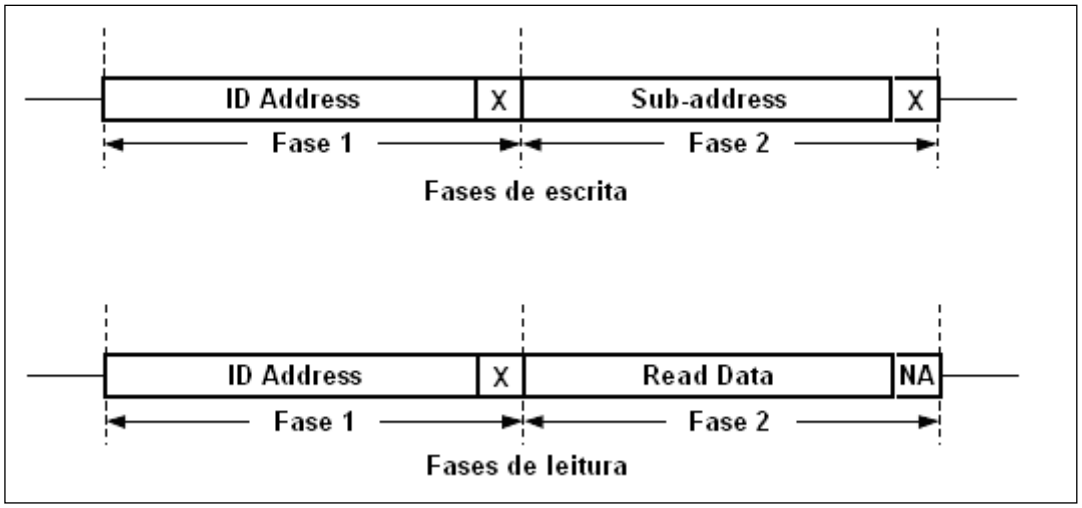


Figura 5.22. Ciclo de leitura do protocolo SCCB

O sensor OV7620 é um dispositivo escravo que suporta taxas de até 400 kbits/s. Neste projeto foram utilizados os seguintes endereçamentos *ID*: 42h para escrita e 43h para leitura. O protocolo foi implementado em linguagem C através do compilador gratuito SDCC para microcontroladores (SDCC, 2006). Na Figura 5.23 é possível ver uma ilustração do *hardware* desenvolvido. O diagrama esquemático desse *hardware* pode ser visto no Apêndice A.

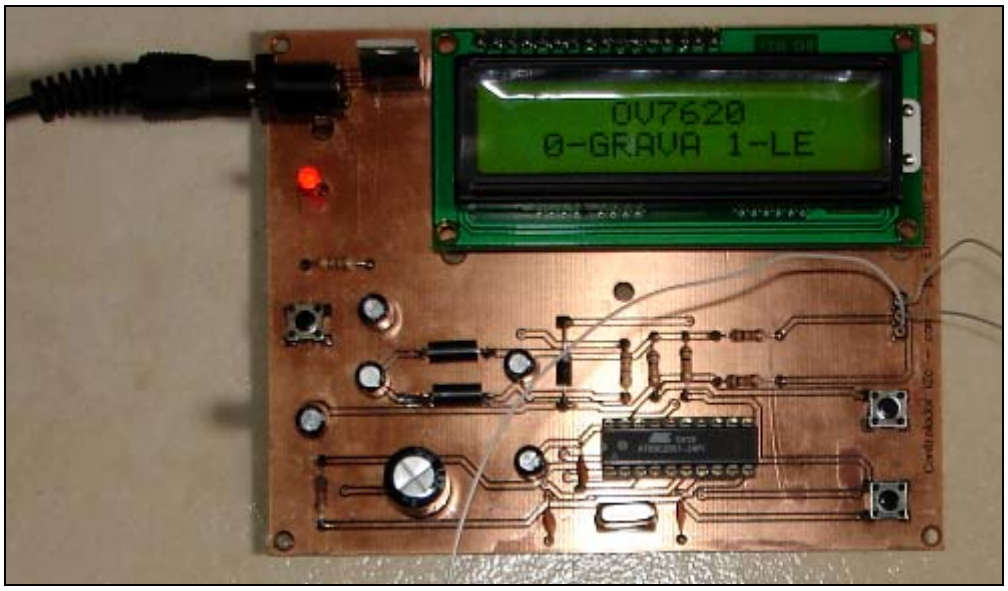


Figura 5.23. *Hardware* responsável pelo gerenciamento do protocolo de comunicação do sensor de vídeo

5.3.1.1.3 Formato RGB *Raw*

Através da configuração do registrador 12h é possível obter o padrão RGB *Raw* nas saídas das portas de vídeo do sensor, que foi o padrão adotado neste trabalho. Assim, no formato de 16 bits de dados (portas Y e UV), a seqüência de saída será correspondente ao padrão de Bayer de filtro de cor, isto é, o formato de saída do canal UV será: GRGRGR... e o formato de saída do canal Y será BGBGBG.... Este padrão pode ser visto na Figura 5.24.

L/C	1	2	3	4	.	641	642	643	644
1	B	G	B	G		B	G	B	G
2	G	R	G	R		G	R	G	R
3	B	G	B	G		B	G	B	G
4	G	R	G	R		G	R	G	R
5	B	G	B	G		B	G	B	G
.									
481	B	G	B	G		B	G	B	G
482	G	R	G	R		G	R	G	R
483	B	G	B	G		B	G	B	G
484	G	R	G	R		G	R	G	R
485	B	G	B	G		B	G	B	G

Figura 5.24. Padrão de Bayer

5.3.1.1.4 Sistema RGB desenvolvido

Este sistema é baseado no modo entrelaçado de 16 bits do sensor CMOS OV7620 e no formato RGB *Raw* discutido na subseção anterior. O *hardware* desenvolvido possui taxa de 60 quadros/s, onde esta foi obtida através de um processo de interpolação de *pixel*s. A resolução adotada para este sistema foi de 320x240 *pixel*s (QVGA), visando-se aplicações em tempo real. Este *hardware* foi baseado no CPLD EPM7128SLC84-15 de baixo custo da família MAX 7000S da Altera (MAX 7000, 2005). Também foi utilizado o *software* Quartus II *Web Edition Full 6.0* da Altera para o processo de descrição do projeto por esquemático e

por linguagem de descrição de *hardware*. Na Figura 5.25, apresenta-se o diagrama esquemático do controlador RGB desenvolvido.

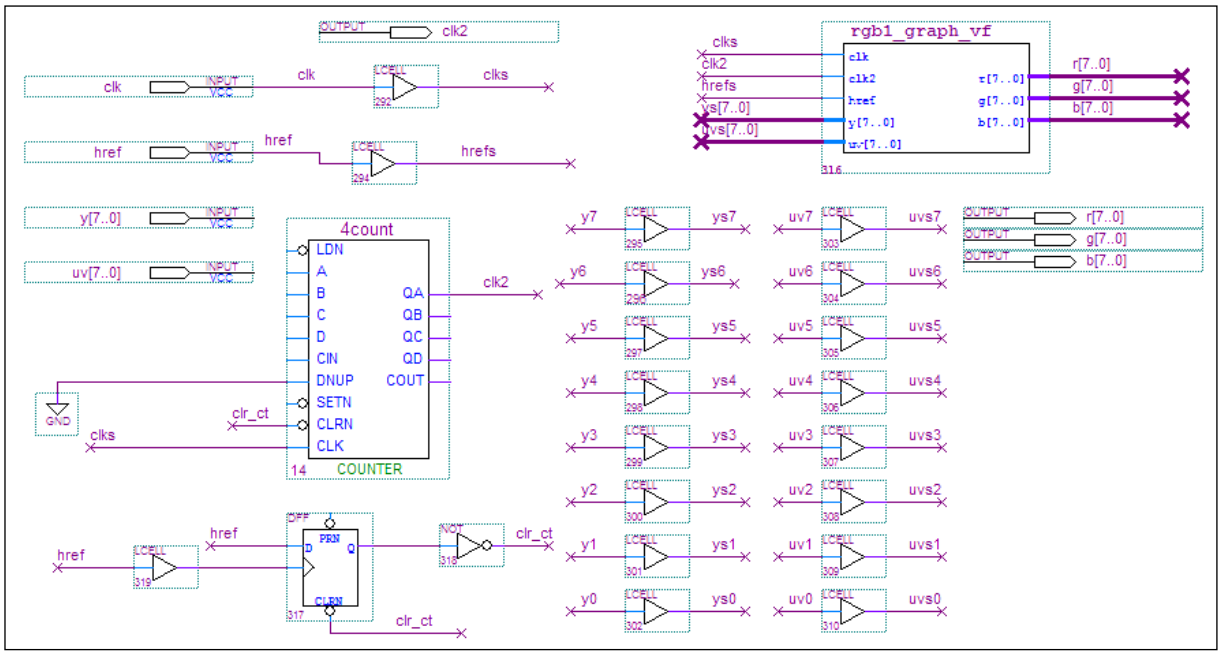


Figura 5.25. Diagrama esquemático do controlador RGB desenvolvido

O funcionamento básico do controlador se baseia no módulo *rgb1_graph_vf* ilustrado na Figura 5.25. Este módulo foi desenvolvido inteiramente em Verilog HDL. No formato RGB *Raw* de 16 bits, os dados se apresentam nos barramentos UV e Y de acordo com a Figura 5.26, assim, o funcionamento do módulo *rgb1_graph_vf* é dividido em duas fases: fase de armazenamento e fase de exibição. Na primeira fase, quando o sinal HREF for igual a nível lógico 1 e após a subida do pulso de *clock* PCLK, são armazenados os *pixel*s G e B. Na segunda fase se armazenam os *pixel*s R e G, dessa maneira, os dados se tornam estáveis para serem apresentados à saída RGB do controlador. O mesmo processo se repete durante os dois campos (ímpar e par) de varredura do sensor. A simulação do processo de interpolação de *pixel*s pode ser vista na Figura 5.27.

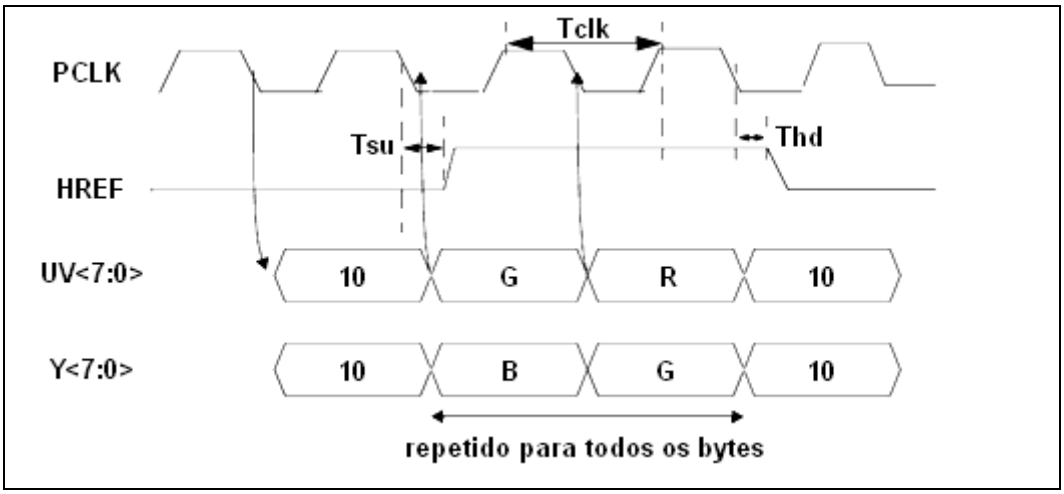


Figura 5.26. Formato RGB *Raw* de 16 bits

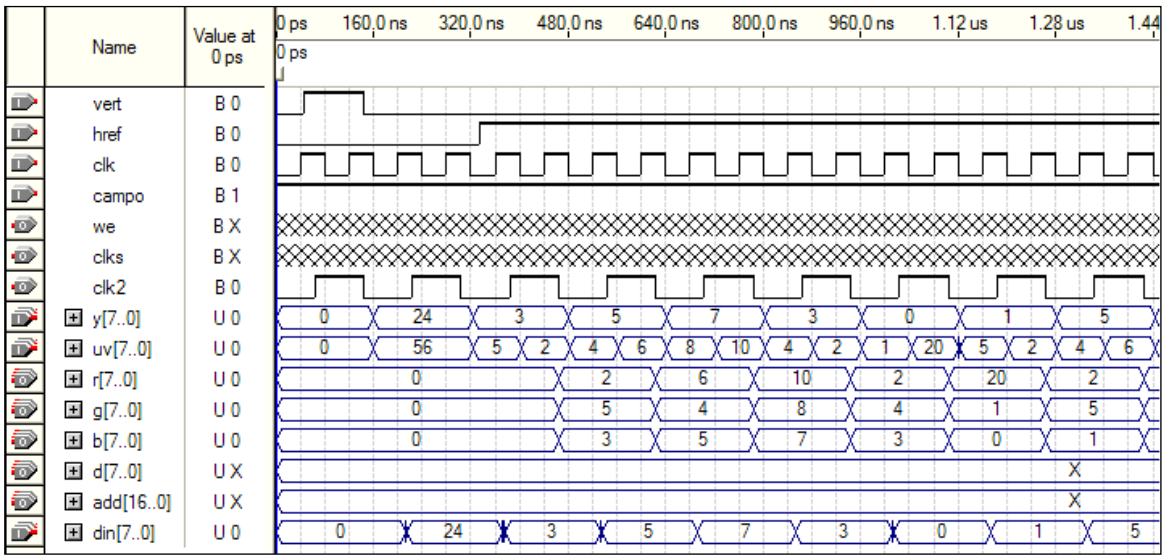


Figura 5.27. Processo de interpolação de *pixel*s do controlador RGB desenvolvido

O diagrama em blocos completo do *hardware* desenvolvido pode ser visto na Figura 5.28. Após a configuração do registrador 12h com o valor 2Ch, o sensor fornece em suas portas digitais os sinais G e B seguidos por R e G. Após o processo de interpolação de *pixel*s supracitado, o CPLD fornece em seus barramentos de saída os dados no formato RGB digital. Estes dados são então convertidos para um formato analógico através do conversor ADV7125 da Analog Devices (ANALOG DEVICES, 2002), que é um conversor de três entradas de 8 bits e alta velocidade utilizado em aplicações de vídeo digital. Finalmente, os dados

analógicos são enviados ao *encoder* AD724 da Analog Devices (ANALOG DEVICES, 1999), que converterá os mesmos de RGB para o padrão NTSC colorido, formando-se assim o sinal de vídeo composto na saída do sistema. Este sinal é muito utilizado em sistemas convencionais de vídeo analógico (PEDRINO et al., 2005). O sinal de vídeo composto também é conhecido na literatura como CVBS, acrônimo para *color*, *vídeo*, *blanking* e *sync*. Este sinal combina as informações de brilho, cor e sincronismo num mesmo cabo. Sua forma de onda típica pode ser vista na Figura 5.29. Cada linha deste sinal é composta de uma porção ativa de vídeo e uma porção de apagamento horizontal. A porção ativa de vídeo contém informações de luminância e crominância. A informação de brilho é representada pela amplitude instantânea do sinal em qualquer ponto no tempo. A unidade de amplitude é dada em IRE, onde: 140 IRE = 1 Vp-p. O nível de preto do sinal geralmente é de 54 mV acima do nível de apagamento. A informação de cor é adicionada ao topo do sinal de luminância e é formada por sinais senoidais onde as cores são identificadas através da diferença de fase entre cada sinal e o sinal de sincronismo de cor (*burst*). De acordo com a Figura 5.30, a amplitude da modulação é proporcional à quantidade de cor (saturação) enquanto que a informação de fase denota a matiz da cor. A porção de apagamento horizontal contém o pulso de sincronismo horizontal e também o sinal de sincronismo de cor localizado após a borda de subida do pulso de sincronismo (*back porch*) (NINCE, 1991).

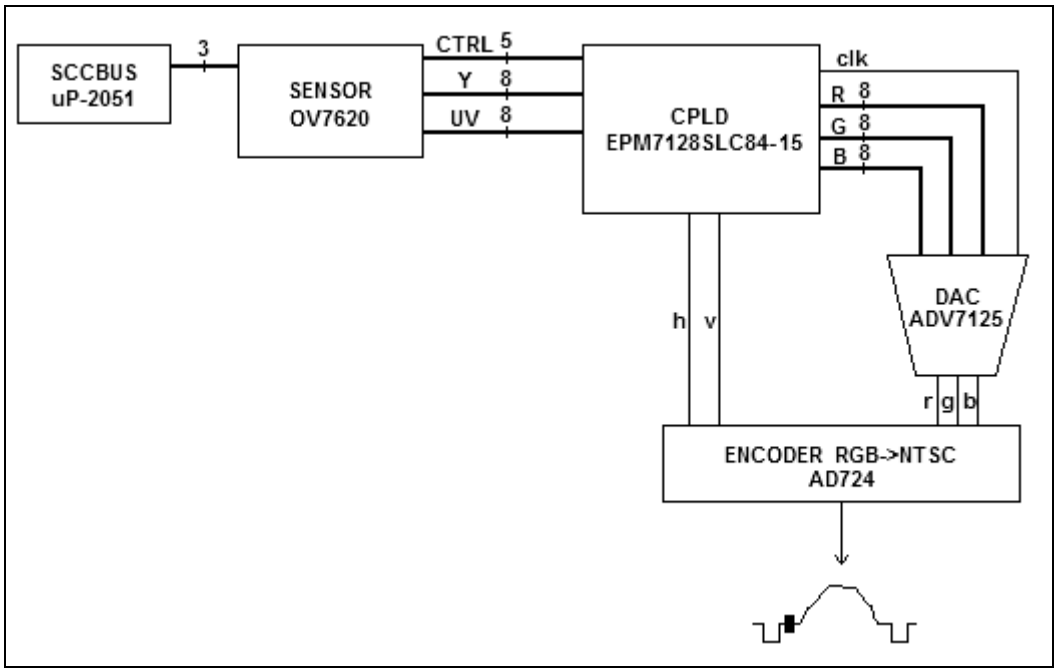


Figura 5.28. Diagrama em blocos do controlador RGB desenvolvido

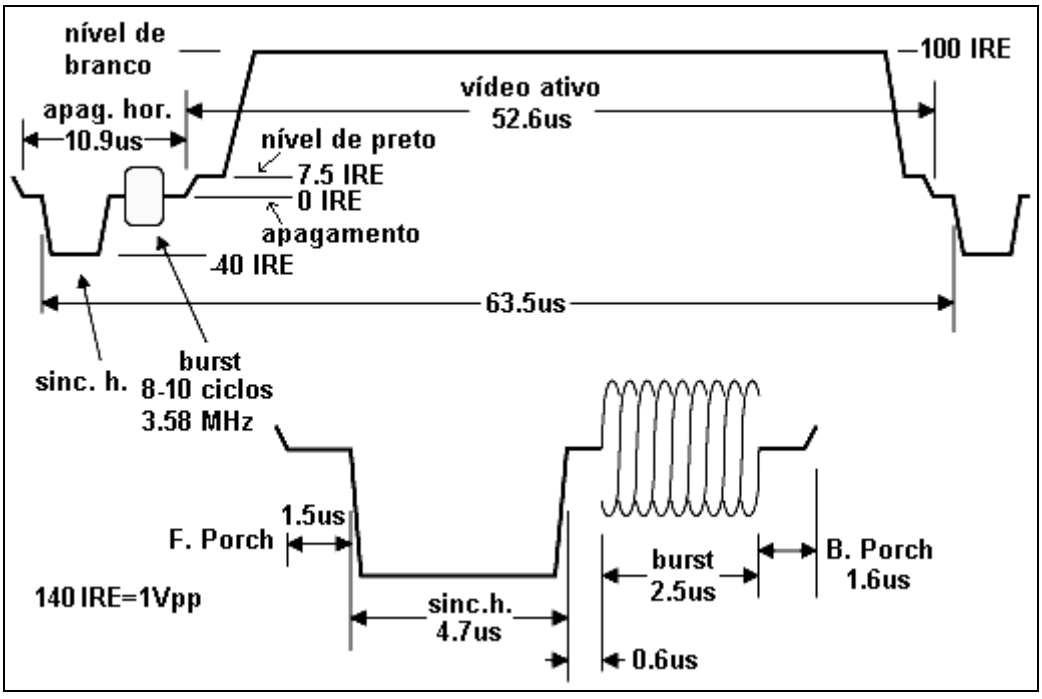


Figura 5.29. Sinal de vídeo composto NTSC (Adaptado de MAXIM, 2001)

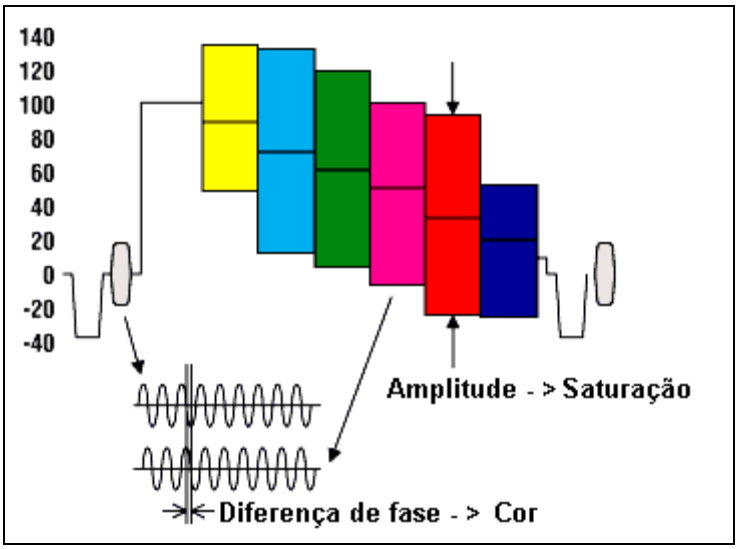


Figura 5.30. Sinal de vídeo composto colorido (Adaptado de MAXIM, 2001)

Na Figura 5.31 é possível ver uma foto do sistema RGB desenvolvido. A foto colorida da Figura 5.32 foi capturada por este sistema. O diagrama esquemático deste sistema pode ser visto no Apêndice B.

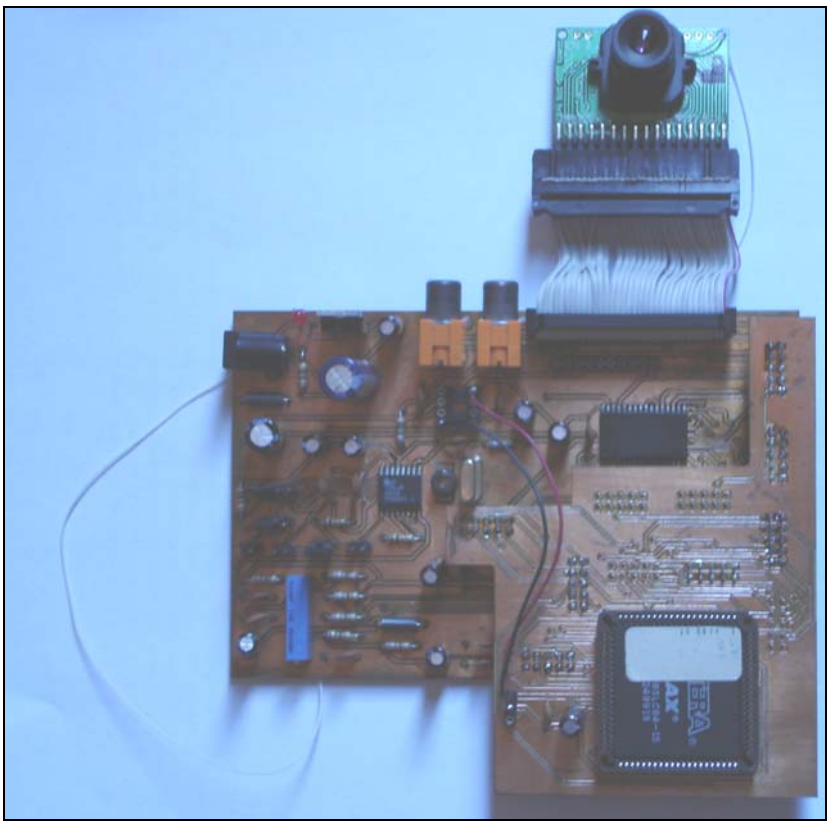


Figura 5.31. Sistema RGB desenvolvido

Figura 5.32. Exemplo de imagem colorida (QVGA) capturada pelo sistema RGB desenvolvido

5.3.1.2 Arquitetura Paralela para Processamento Morfológico Rápido de Imagens Coloridas

Nesta subseção será apresentada a descrição de uma arquitetura *pipeline* para processamento morfológico de imagens coloridas em tempo real através do uso de *hardware* reconfigurável, considerando-se que na literatura há um número muito reduzido de arquiteturas em *hardware* voltadas para o processamento morfológico de imagens coloridas, talvez devido às dificuldades existentes já mencionadas neste texto. A arquitetura implementada foi desenvolvida inteiramente em Verilog HDL com o auxílio do *software* Quartus II *Web Edition Full 6.0* da Altera. Nesta abordagem, as componentes de imagem são ordenadas através da técnica de ordenação por uma componente ou pela distância *city-block* quando ocorrerem relações de pré-ordem (PEDRINO; RODA, 2006a).

5.3.1.2.1 *Hardware* Desenvolvido

Nesta implementação foi utilizado o espaço de cores RGB para simplificar a descrição do *hardware*. Apesar do espaço de cores HSI ser mais próximo da interpretação humana de

cores, de acordo com (ZAMORA, 2002), não existe um modelo ótimo para todas as aplicações. Os passos do algoritmo proposto são apresentados nas próximas subseções.

5.3.1.2.1.1 Definições Básicas

Uma cor *c* no espaço de cores RGB será representada como um vetor $F_c=[R_c,G_c,B_c]$. *F* será uma imagem colorida e *B* será uma janela contendo a vizinhança do *pixel c* sob consideração. Neste contexto, *B* será um elemento estruturante plano. Assim, a erosão e a dilatação morfológica de uma imagem colorida *F* por um elemento estruturante *B* será definida segundo as equações 3.15 e 3.16, respectivamente. Portanto, o algoritmo contém os seguintes passos:

1. determinação de uma janela *B* sobre o *pixel c*,
2. ordenação dos dados correspondentes a *B* em *F* usando o método proposto de ordenação vetorial,
3. determinação do ínfimo e do supremo.

5.3.2.2.1.2 Passos do Algoritmo

Os passos do algoritmo desenvolvido são apresentados a seguir:

1. deslocar *B* por *F*, e para cada posição de *B*, ordenar os dados correspondentes a *B* em *F* utilizando a técnica de ordenação vetorial por uma componente,
2. se não houver redundância de dados, os valores de *min* e *max* são determinados para a posição atual de *B*,

a. senão, avalia-se a distância *city-block* de cada dado redundante em relação à origem,

b. se não houver redundâncias em (a), os valores de *min* e *max* são determinados para a posição atual de *B*,

c. senão, calcula-se a distância *city-block* de cada dado redundante em relação a um *pixel* de referência de *B* e escolhe-se a menor distância como saída do processo para a posição atual de *B*,

d. os valores de *min* e *max* são então armazenados nas respectivas posições de saída de acordo com a posição central de *B*.

3. repetir os passos 1 e 2 para a nova posição de *B*.

Nas Figuras 5.33 e 5.34, pode-se ver uma simulação realizada no *software* Quartus II *Web Edition Full 6.0* da Altera para a arquitetura proposta. Neste exemplo, foi realizada uma erosão de uma imagem colorida hipotética de 6 bits de resolução por meio de um elemento estruturante plano e quadrado de dimensão 2x2. No entanto, dependendo da capacidade lógica do *chip* reconfigurável utilizado, esta arquitetura pode ser reconfigurada para trabalhar com qualquer resolução de imagem colorida, forma e tamanho de elemento estruturante plano. Na Figura 5.35, apresenta-se o resultado de uma simulação deste algoritmo realizada no *software* Matlab 7.0 da Mathworks para as operações morfológicas de dilatação e erosão de uma imagem colorida de entrada por um elemento estruturante plano e quadrado de dimensão 3x3.

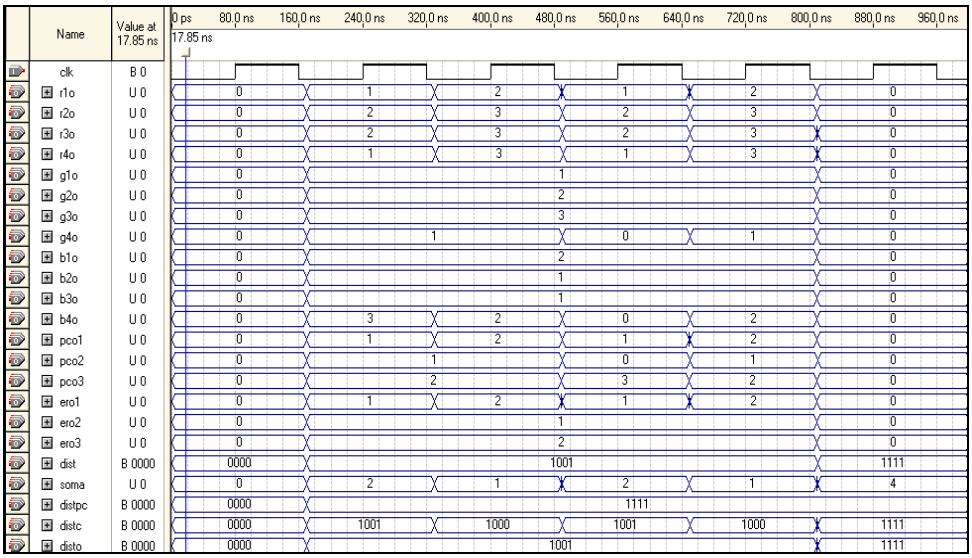


Figura 5.33. Erosão de uma imagem colorida hipotética de 6 bits por um elemento estruturante plano e quadrado de dimensão 2x2

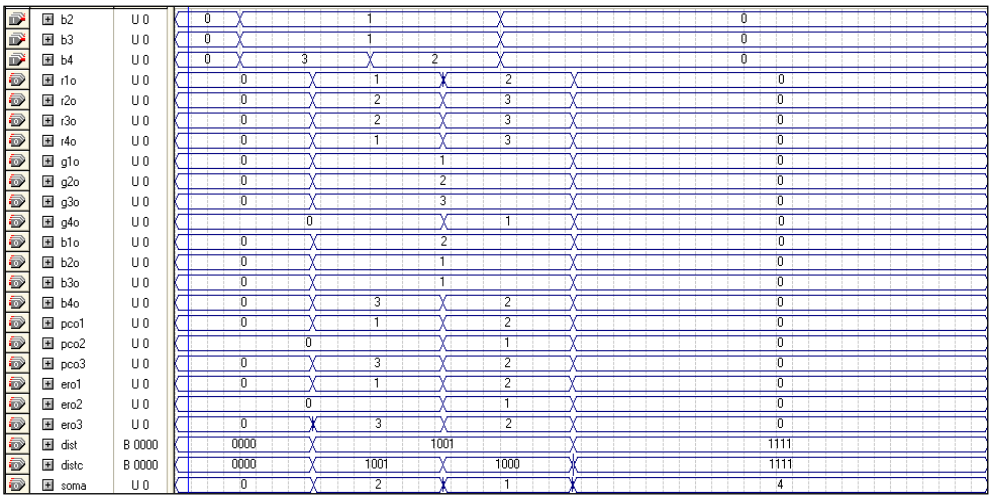


Figura 5.34. Erosão de uma imagem colorida hipotética de 6 bits por um elemento estruturante plano e quadrado de dimensão 2x2

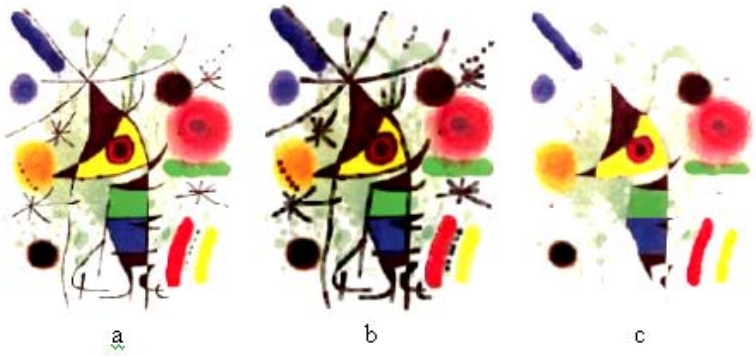


Figura 5.35. a-) Imagem original (MIRO - "Le Chanteur" by Joan Miro); b-) resultado da erosão morfológica; c-) resultado da dilatação morfológica

Na Figura 5.33, os *pixel*s rgb_1=[1,1,2], rgb_2=[2,2,1], rgb_3=[2,3,1] e rgb_4=[1,1,3] estão sendo considerados num processo de erosão morfológica. O *pixel* rgb_1 foi adotado como origem no primeiro ciclo de *clock*. De acordo com a técnica de ordenação por uma componente, os *pixel*s escolhidos neste caso serão: rgb_1 e rgb_4, considerando-se que suas primeiras componentes são iguais. Então, devido à relação de pré-ordem existente, cada *pixel* redundante é comparado em relação a sua distância *city-block* à origem, selecionando-se neste exemplo, como saída do processo de erosão, o *pixel* rgb_1. No segundo pulso de *clock*, o *pixel* resultante será igual a [2,1,2], considerando-se que sua primeira componente é menor que as demais. Na Figura 5.34, após o processo de erosão, o *pixel* [1,0,3] foi escolhido como saída, devido a sua distância *city-block* ser menor que as demais em relação ao *pixel* de origem, satisfazendo-se assim o passo 2c do algoritmo proposto. As mesmas idéias são repetidas para o processo de dilatação morfológica de uma imagem colorida. Os resultados obtidos na Figura 5.35, juntamente com outros resultados obtidos pelo procedimento descrito, foram comparados através de um processo de diferença de imagens com outros resultados de trabalhos similares existentes na literatura, assim, não houve diferenças entre os resultados comparados.

Na Tabela 5.2 são apresentados os recursos utilizados pelo dispositivo FPGA adotado. Uma célula lógica (LC) descreve um bloco de construção básica de um dispositivo da Altera.

Tabela 5.2 – Resumo dos recursos utilizados pelo dispositivo FPGA

Dispositivo	Pinos de entrada	Pinos de saída	Pinos Bidir	% Mem	LCs	% Total
EPF10K10TC144-3	33	56	0	0	146	25 %

5.4 CONSTRUÇÃO AUTOMÁTICA DE OPERADORES MORFOLÓGICOS UTILIZANDO PROGRAMAÇÃO GENÉTICA

Nesta seção é apresentada uma nova metodologia de construção automática de operadores morfológicos utilizando uma abordagem linear baseada em programação genética para projeto de filtros não lineares, emulação de filtros desconhecidos e reconhecimento de padrões, entre outros exemplos, conforme já sobredito. Para esta finalidade foi criada uma *toolbox*, denominada *morph_gen*, no Matlab 7.0, onde as soluções geradas por esta são expressas por meio dos operadores básicos de morfologia matemática, dilatação e erosão, juntamente com operadores lógicos e aritméticos. Por fim, estas soluções são convertidas em *opcodes* da arquitetura *pipeline* reconfigurável em *hardware* para processamento morfológico de imagens em tempo real a ser descrita na seção subseqüente. Esta *toolbox* permite o processamento de imagens binárias, em níveis de cinza e coloridas. Conforme já abordado neste trabalho, o projeto de operadores morfológicos mais complexos para uma dada aplicação não é uma tarefa trivial na prática, assim, é necessário um conhecimento abrangente para selecionar de forma adequada os elementos estruturantes e os operadores morfológicos a serem utilizados numa dada aplicação. Em (YODA et al., 1999; HARVEY; MARSHALL, 1996; BALA; WECHSLER, 1993) são utilizados algoritmos genéticos para a finalidade de geração automática de procedimentos morfológicos, tentando-se assim reduzir as dificuldades do processo de projeto. Contudo, os trabalhos supracitados têm se limitado a projetar filtros ótimos, nos quais as seqüências de operadores morfológicos são de tamanho fixo e os elementos estruturantes utilizados são limitados a formas clássicas (QUINTANA et al., 2002). Na literatura, somente alguns poucos trabalhos utilizam os conceitos de programação genética aplicados ao processamento de imagens. Assim, neste trabalho é proposta uma nova metodologia para gerar procedimentos automáticos de morfologia matemática por meio de

uma variação da técnica de programação genética com a possibilidade de explorar um espaço muito grande (não fixo) de possíveis algoritmos morfológicos com elementos estruturantes de formas diferenciadas. Também, o algoritmo desenvolvido permite utilizar ou acrescentar qualquer tipo de instrução, além das morfológicas, de acordo com uma dada aplicação. Assim, o método apresentado permite minimizar as dificuldades inerentes do processo de projeto de operadores morfológicos para uma dada aplicação e também suas instruções foram idealizadas para serem implementadas em *hardware*. Para finalizar, são mostrados alguns exemplos de aplicação do método proposto, nos quais a eficiência do processo pôde ser avaliada.

5.4.1 Construção Automática de Operadores Morfológicos

Nesta subseção, apresenta-se a metodologia de construção automática de operadores morfológicos utilizando programação genética (PEDRINO; RODA, 2006b). O algoritmo proposto utiliza uma abordagem linear de programação genética (OLTEAN; GROSAN, 2004; BRAMEIER; BANZHAF, 2007). O algoritmo desenvolvido trabalha basicamente da seguinte maneira: dadas duas imagens amostras de entrada, uma original e outra possuindo somente características desejadas a serem extraídas pelo procedimento genético, buscam-se seqüências de operadores morfológicos no espaço de algoritmos de morfologia matemática que satisfaçam o requisito supracitado. Tais operadores são procedimentos pré-definidos para trabalharem com determinados tipos de elementos estruturantes, podendo-se criar novos operadores quando necessário. A saída do programa é representada por uma estrutura linear contendo os tipos de operadores utilizados no algoritmo produzido (melhor indivíduo) juntamente com seus elementos estruturantes especificados. O resultado de uma operação é o argumento do operador subseqüente, e assim sucessivamente. Os parâmetros do algoritmo

genético são fornecidos pelo usuário, através de uma interface gráfica (*GUI*), dentre os quais se destacam: tamanho máximo de cada programa (cromossomo), número de gerações, número de cromossomos, probabilidade de cruzamento, probabilidade de mutação, probabilidade de reprodução, erro esperado e tipos de instruções adequadas para um determinado problema. A função de aptidão utilizada para avaliar cada cromossomo (procedimento) é o erro médio absoluto (EMA) entre a imagem de saída do processo e a imagem objetivo, de acordo com a equação 5.1. Nesta equação, *a* representa uma imagem resultante gerada por um cromossomo particular da população de indivíduos e *b* corresponde à imagem objetivo.

$$d(a,b) = \frac{1}{XY}\sum_{i}^{X}\sum_{j}^{Y}\left|a(i,j) - b(i,j)\right| \qquad (5.1)$$

As instruções são codificadas por cadeias binárias de tamanhos variáveis. Na Figura 5.36, apresenta-se o fluxograma de operação do sistema proposto. Seu funcionamento é análogo ao de um algoritmo genético. Entretanto, no procedimento linear proposto neste trabalho, os genes são representados por instruções e os cromossomos possuem tamanhos variados, conforme descrito anteriormente, ao contrário do algoritmo genético clássico, no qual os cromossomos possuem tamanho fixo e os genes são representados por dados. O método de seleção adotado foi o de torneio entre três indivíduos da população corrente escolhidos de forma aleatória. Também foi utilizada a técnica de substituição geracional de cromossomos e a estratégia de elitismo.

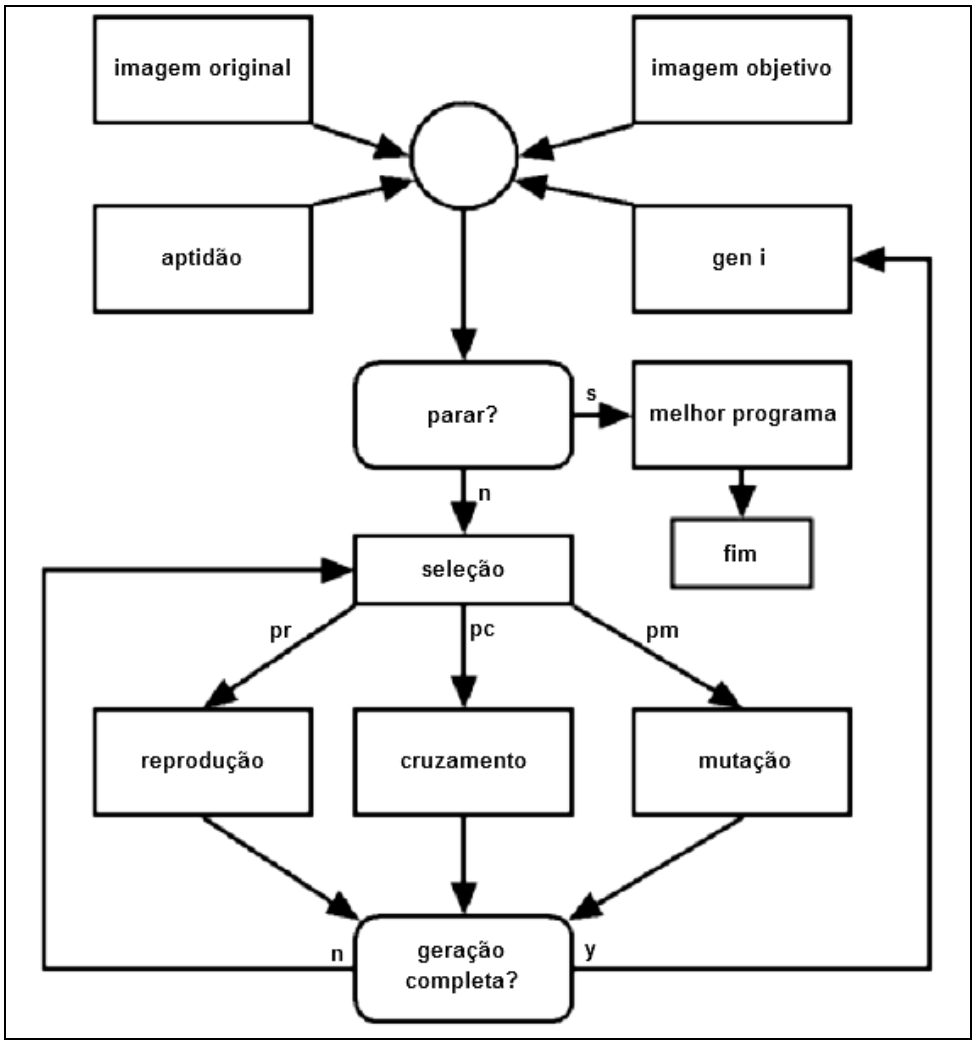


Figura 5.36. Fluxograma de operação do sistema proposto

Na Figura 5.37, apresenta-se a janela principal do programa. A janela contendo as instruções que podem ser selecionadas para uma dada aplicação pode ser vista na Figura 5.38. Essas instruções possuem um único argumento de entrada, onde o tipo de operação morfológica a ser utilizada, bem como o tamanho e a forma de seu elemento estruturante, já estão predefinidos na base de dados do projeto. Também, a *toolbox* desenvolvida permite a inclusão de novas instruções. Na Figura 5.39 é possível ver a janela de parâmetros genéticos. O parâmetro *Profundidade* se refere ao tamanho limite da estrutura linear utilizada (tamanho máximo de programa). Alguns parâmetros comuns a alguns problemas clássicos já são predefinidos como padrão.

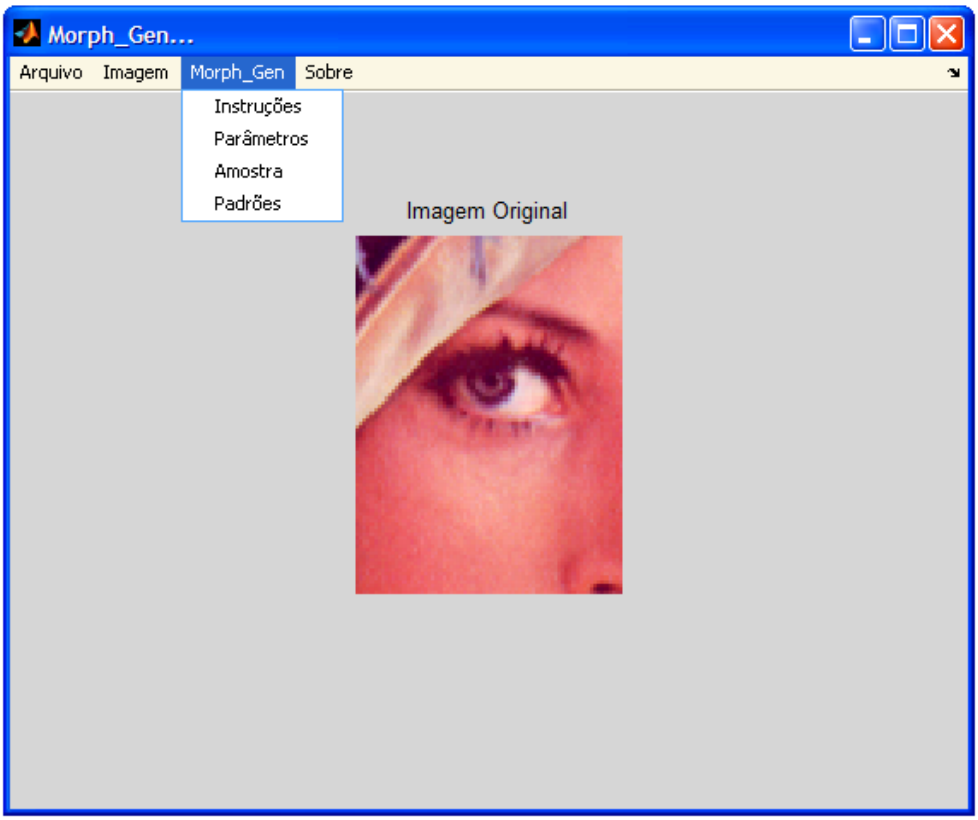


Figura 5.37. Tela principal do programa *Morph_Gen*

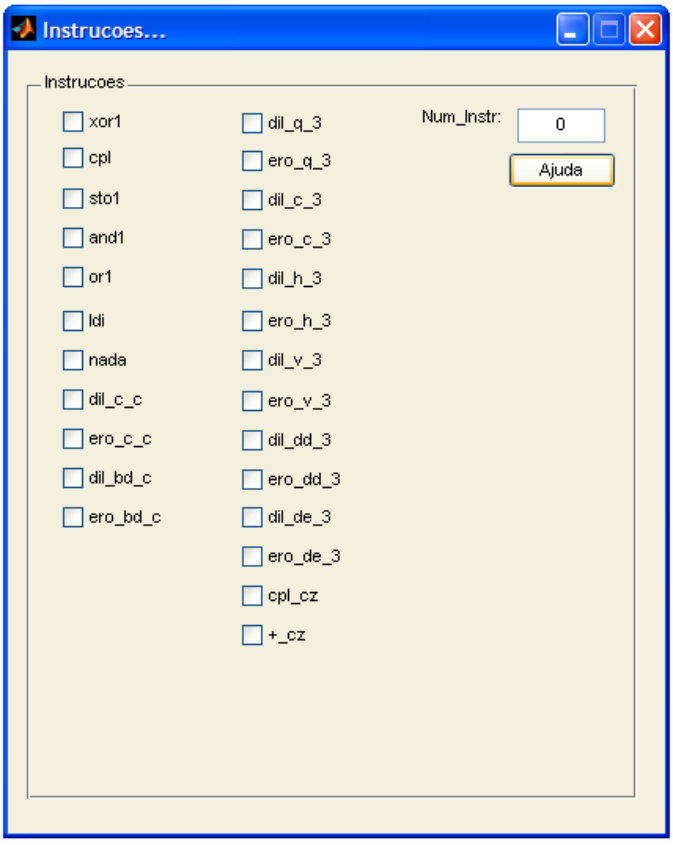


Figura 5.38. Janela de instruções do programa *Morph_Gen*

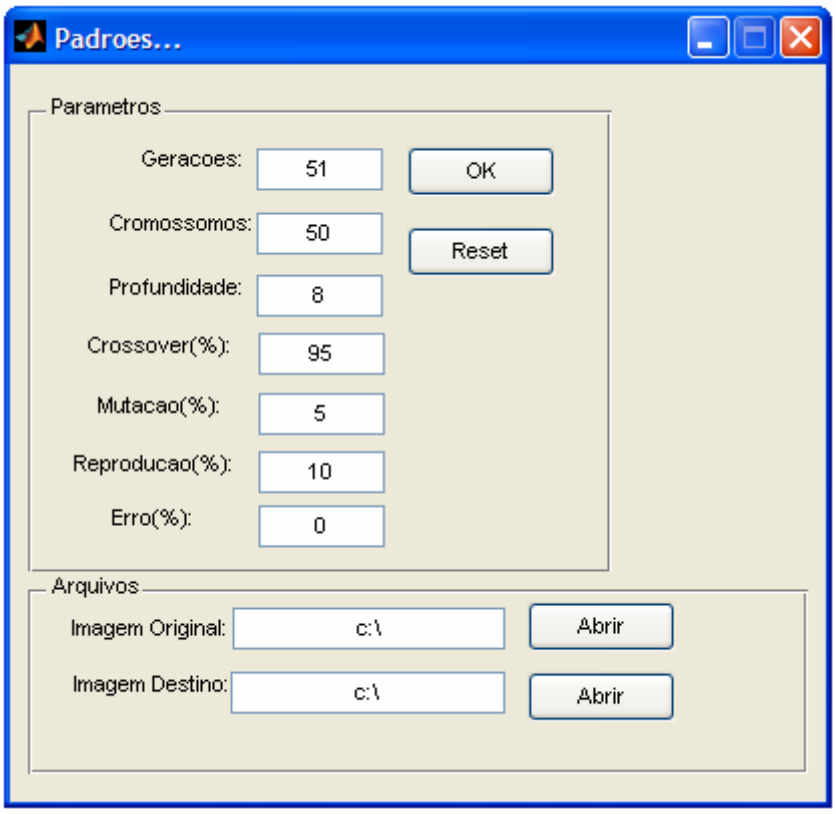


Figura 5.39. Janela de parâmetros genéticos do programa *Morph_Gen*

Na Figura 5.40 é possível ver um exemplo de avaliação de um cromossomo da população de indivíduos (programas). Uma vez que os cromossomos são codificados por cadeias binárias, por exemplo, se o usuário selecionar as instruções: *and* (operador lógico E), *sto* (armazenamento do resultado atual), *ero* (erosão morfológica) e *cpl* (operador lógico NÃO), o primeiro operador será codificado como "00_2", o segundo como "01_2", o terceiro como "10_2" e o último como "11_2". Se o tamanho máximo de programa escolhido for igual a 4, por exemplo, o cromossomo "00011011_2" poderia ser criado. Portanto, neste caso, a imagem original sofreria um processo de erosão por um elemento estruturante quadrado de dimensão 3x3 seguido por um processo de complementação lógica.

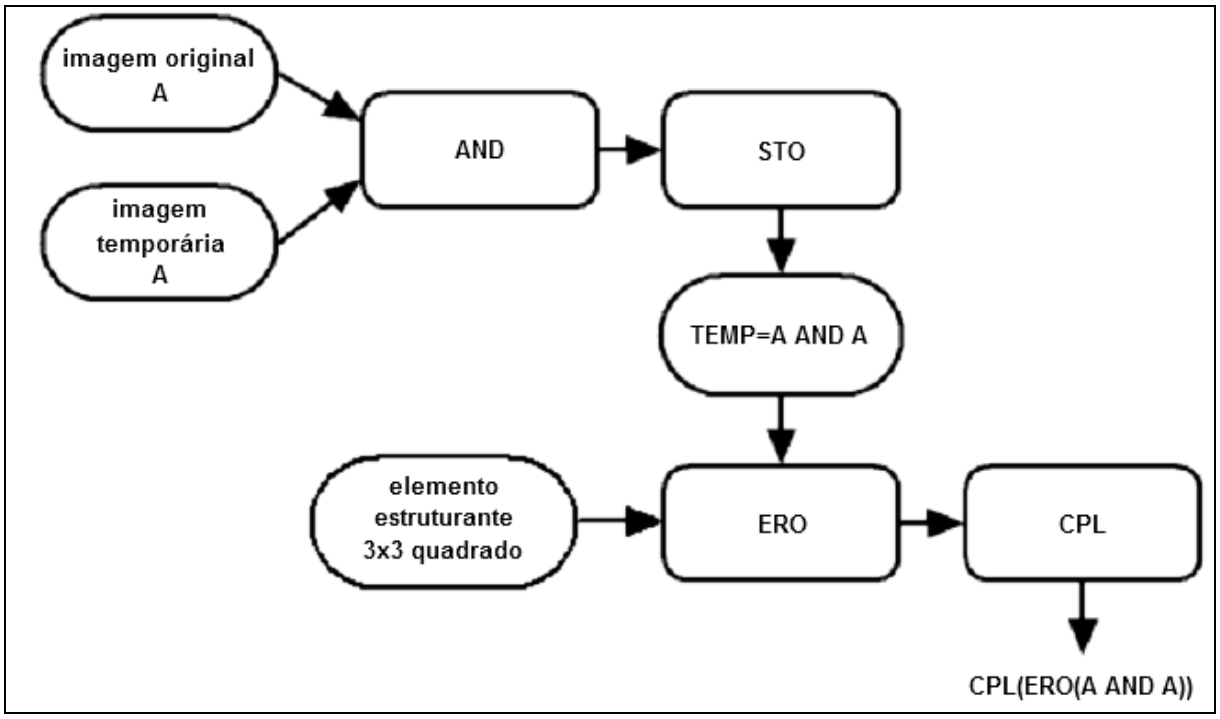


Figura 5.40. Avaliação do cromossomo "00011011_2" referente ao programa "and->sto->ero->cpl" aplicado à imagem original A

Na operação de cruzamento, de acordo com a probabilidade de cruzamento, dois indivíduos pais geram dois indivíduos filhos a serem inseridos na nova população através da troca de seqüências de instruções selecionadas em pontos aleatórios dos indivíduos pais. O operador de mutação troca uma seqüência de instruções de um indivíduo por outra gerada aleatoriamente, de acordo com a probabilidade de mutação. O operador de reprodução copia um indivíduo da população atual para a nova população de acordo a probabilidade de reprodução. Estes operadores são aplicados a indivíduos que venceram o processo de seleção por torneio de acordo com suas aptidões. Na figura 5.41 são apresentados exemplos dos operadores de cruzamento e mutação utilizados neste trabalho.

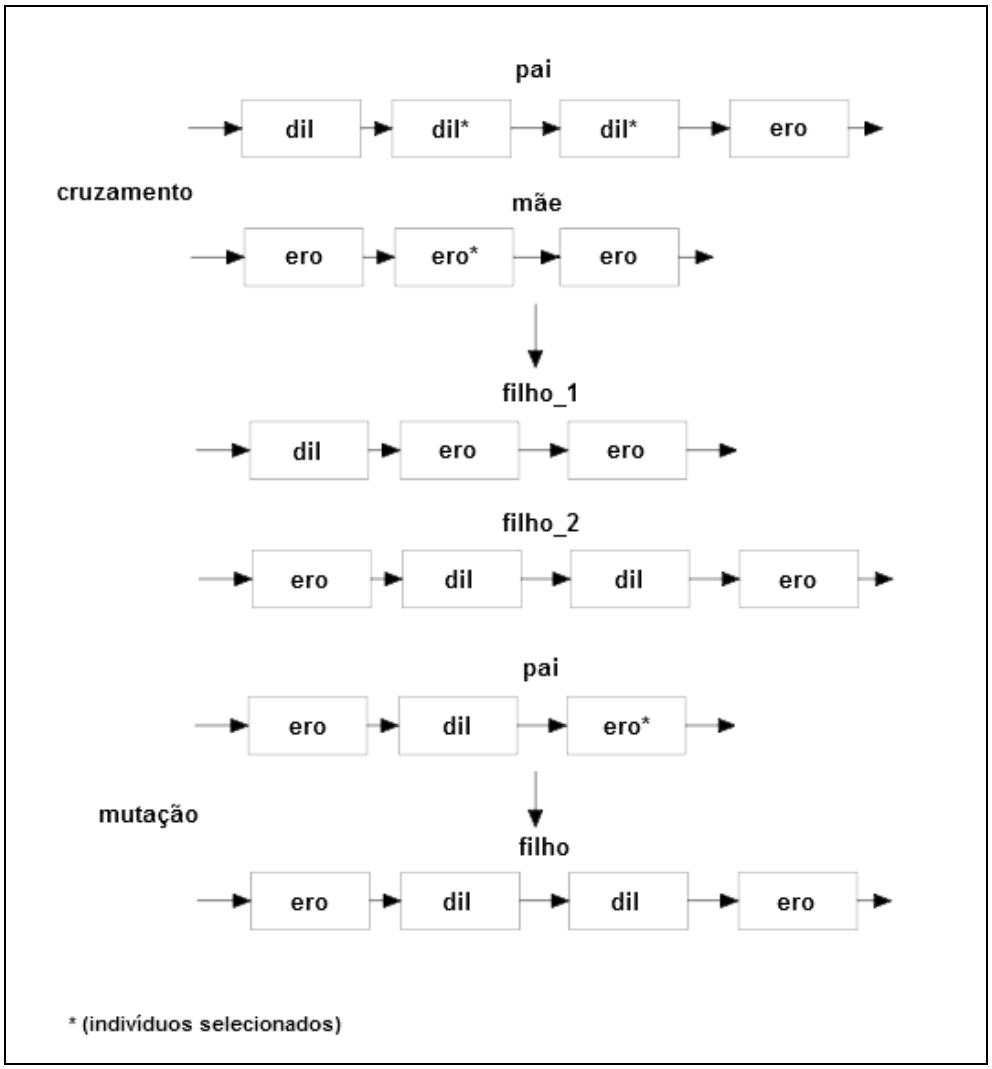


Figura 5.41. Operadores de cruzamento e mutação

5.4.2 Alguns Exemplos de Aplicação

Nesta subseção serão apresentados alguns exemplos de aplicação utilizando o algoritmo evolucionário desenvolvido. Também, os resultados obtidos são comparados a outras formas de implementação. Na Figura 5.42, a imagem original foi corrompida com o ruído ‘sal e pimenta’ de densidade: 0.04. Para a construção automática de um filtro morfológico para eliminação deste ruído, foram utilizadas as instruções *ero_q_3* e *dil_q_3*, onde estas correspondem a uma erosão e a uma dilatação por um elemento estruturante

quadrado de dimensão 3x3, respectivamente. Após a apresentação das amostras de treinamento, o procedimento genético evolucionário de construção automática de operadores morfológicos convergiu antes da 5ª geração criando o filtro: “ero_q_3->dil_q_3->dil_q_3->ero_q_3”, sendo este um filtro clássico na literatura de morfologia matemática. Assim, a imagem de entrada sofreu um processo de erosão seguido por dois processos de dilatação e por um de erosão. Para esta tarefa, foram utilizados os seguintes parâmetros genéticos: 51 gerações, 50 cromossomos, tamanho máximo de programa igual a 4, taxa de cruzamento de 95%, taxa de mutação de 5% e de reprodução de 10%. O erro médio absoluto (EMA) encontrado foi de aproximadamente 0,061%. O tempo de treinamento para gerar a solução do problema foi menor que 8,38s e o de execução foi menor que 0,16s. Neste exemplo e nos demais a seguir, foi utilizado um *notebook* de 512MB de memória RAM com processador Athlon AMD-64 de 1,8GHz para a execução das tarefas em *software*.

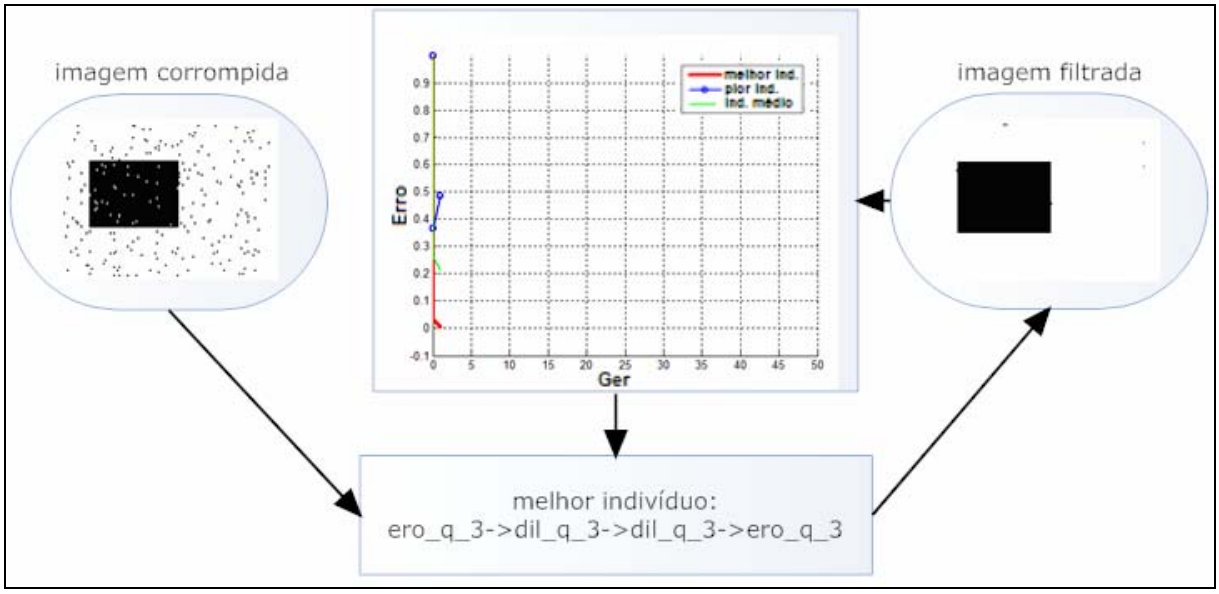


Figura 5.42. Filtro gerado automaticamente para eliminação do ruído ‘sal e pimenta’ de densidade ‘0.04’ presente na imagem original

Na Figura 5.43 é apresentada uma seqüência de operadores morfológicos e lógicos gerada automaticamente pelo sistema evolucionário para a tarefa de detecção de bordas em imagens. Após a apresentação das amostras de treinamento, o processo convergiu antes da 5ª geração criando o programa: “sto->and1->ero_aleat->cpl->and1->nada”. A instrução *nada* é utilizada como um recurso para variar o tamanho da estrutura cromossômica linear gerada, onde esta repete em sua saída o argumento presente em sua entrada. A instrução *sto* armazena a imagem original temporariamente e em seguida é realizado um *and* lógico entre a imagem armazenada e a imagem original, neste caso, resultando na própria imagem original. A seguir, esta sofre um processo de erosão por um elemento estruturante escolhido aleatoriamente seguido por um processo de complementação lógica e, por fim, realiza-se um *and* lógico entre o resultado obtido após a complementação e a imagem original, gerando, portanto, a borda da imagem de entrada. Para esta tarefa, foram utilizados os seguintes parâmetros genéticos: 50 gerações, 25 cromossomos, tamanho máximo de programa igual a 8, taxa de cruzamento de 90% e taxas de mutação e reprodução de 20%. O erro médio absoluto (EMA) encontrado foi de aproximadamente 0,01%. O tempo de treinamento para gerar a solução do problema foi menor que 5s e o de execução foi menor que 0,07s. Estas instruções também foram aplicadas à imagem da Figura 5.44, onde a instrução *ero_aleat* foi trocada por *ero_q_3* e também foi acrescentada a instrução dil_q_3 ao final do processo. O resultado obtido foi comparado ao de um *software* comercial usado para a mesma tarefa, onde o erro EMA encontrado foi menor que 4,3%.

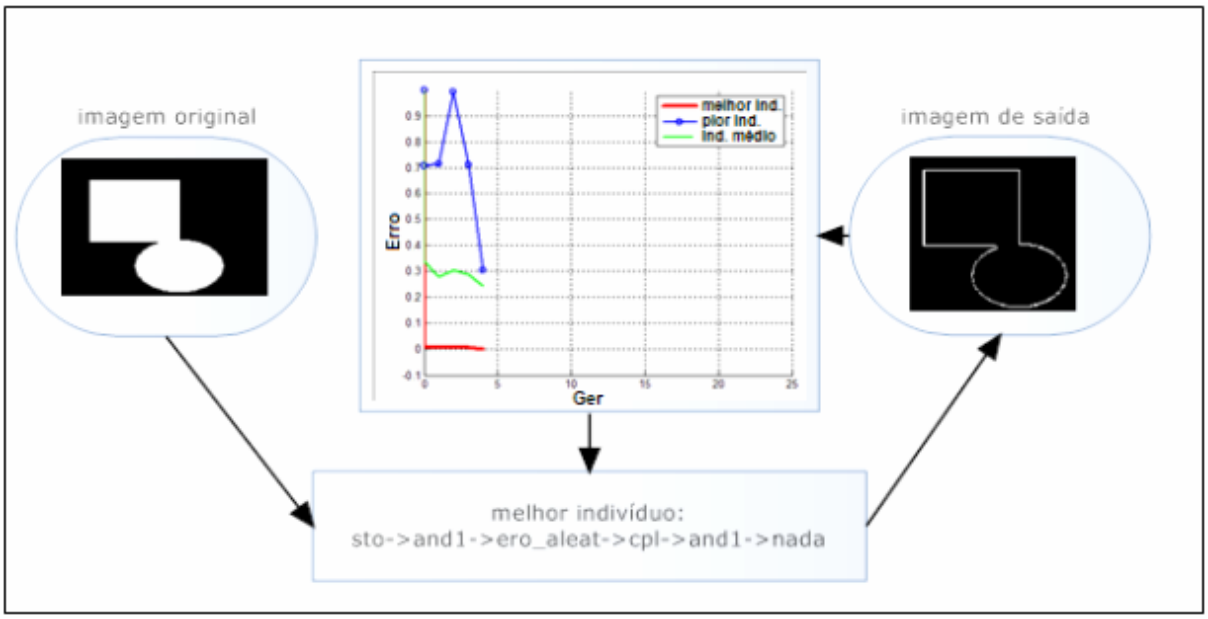


Figura 5.43. Detector de bordas criado automaticamente

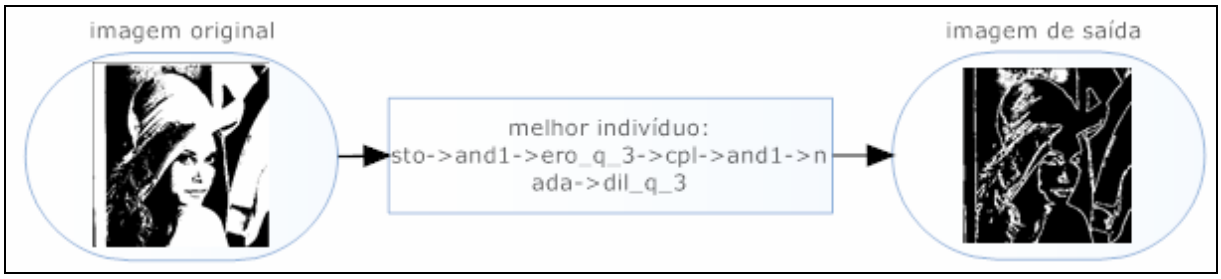


Figura 5.44. Detector de bordas aplicado à imagem Lenna

Na Figura 5.45 é apresentado um exemplo de extração automática de padrões musicais presentes numa dada partitura musical. Para esta tarefa foram apresentadas ao programa a imagem original e a imagem objetivo contendo características desejadas a serem extraídas pelo procedimento evolucionário. O melhor programa encontrado pelo sistema foi: "dil_dd_3->dil_q_3->nada->ero_q_3->nada->ero_q_3". A instrução *dil_dd_3* corresponde a uma dilatação por um elemento estruturante diagonal de dimensão 3x3. Para esta tarefa, foram utilizados os seguintes parâmetros genéticos: 50 gerações, 25 cromossomos, tamanho máximo de programa igual a 6, taxa de cruzamento de 90% e taxas de mutação e reprodução de 20%. O erro médio absoluto (EMA) encontrado foi menor que 0,7%. O tempo de treinamento para gerar a solução do problema foi de aproximadamente 71s e o de execução foi menor que

0,02s. O programa encontrado também foi aplicado à imagem da Figura 5.46 produzindo um bom resultado.

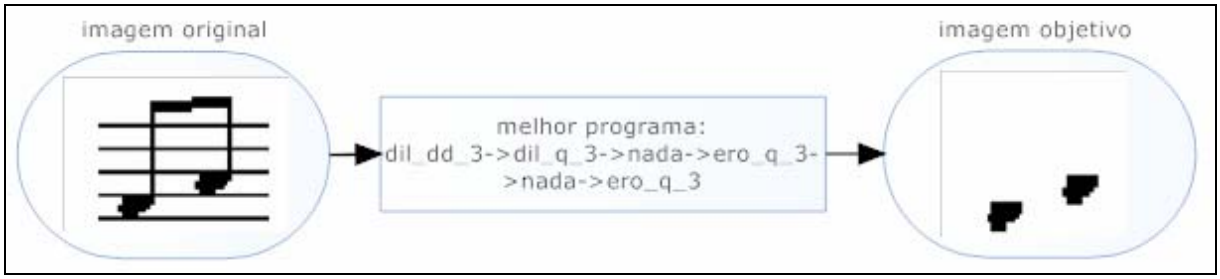


Figura 5.45. Procedimento criado automaticamente para extração de padrões musicais

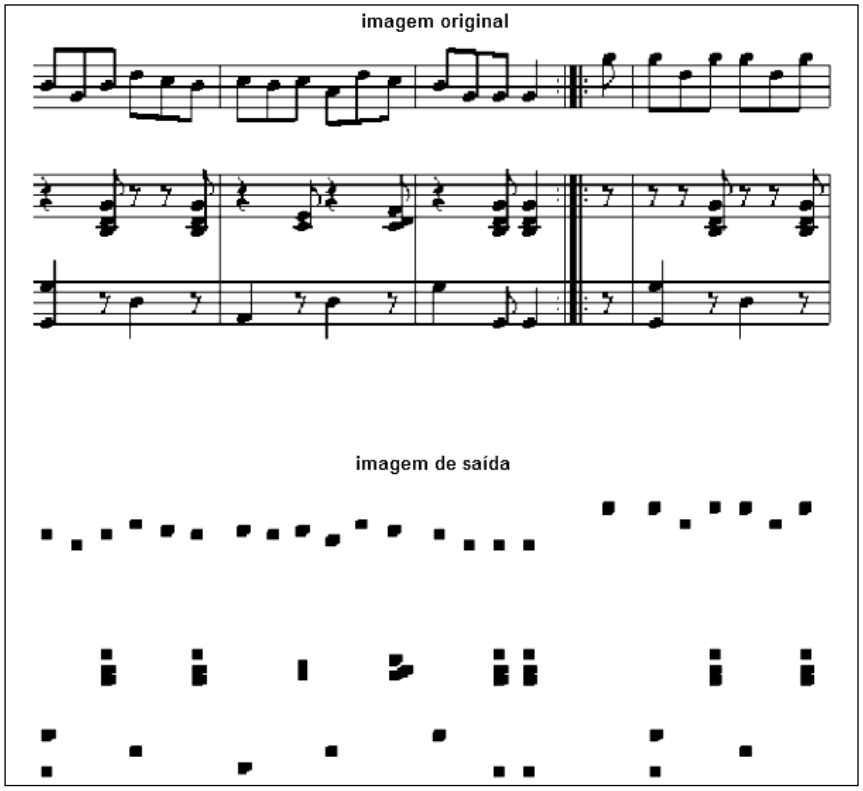


Figura 5.46. Exemplo de extração de padrões musicais

Na Figura 5.47 a imagem original (em níveis de cinza) foi corrompida com o ruído ‘sal e pimenta’ de densidade: 0.05. Para a construção de um filtro automático para remoção deste ruído foram utilizadas as instruções *dil_q_3* e *ero_q_3*. Neste exemplo, foram utilizados os seguintes parâmetros genéticos: 25 gerações, 12 cromossomos, tamanho máximo de programa igual a 4, taxa de cruzamento de 90% e taxas de mutação e reprodução de 20%. O

erro médio absoluto (EMA) encontrado foi menor que 2%. O tempo de treinamento para gerar a solução do problema foi de aproximadamente 2,4min e o de execução foi menor que 0,6s.

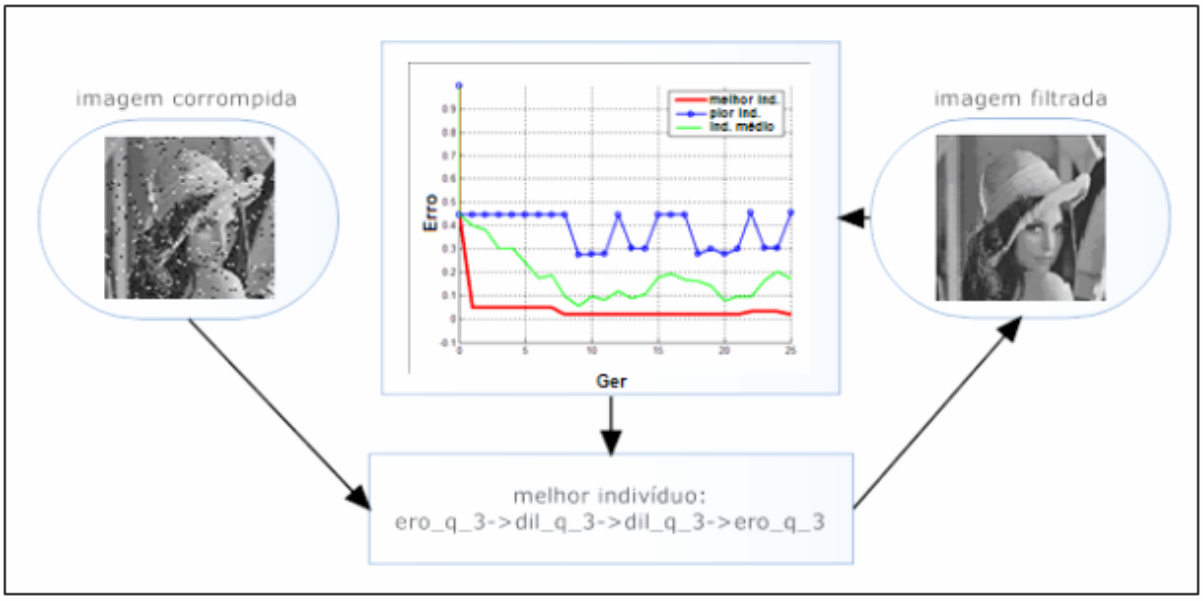


Figura 5.47. Exemplo de construção automática de um filtro morfológico simples para eliminação do ruído 'sal e pimenta' de densidade: 0.05

Na Figura 5.48, o sistema evolucionário foi utilizado para emular um filtro desconhecido implementado por um *software* comercial. Primeiramente, a imagem original foi processada através do filtro *glowing edges* do *software* comercial gerando a imagem objetivo para o procedimento genético. Após a apresentação de ambas as imagens ao sistema evolucionário, o melhor programa encontrado foi: "ero_q_3->comp_cz->add->sto->ero_q_3->dil_q_3->add". A instrução *comp_cz* corresponde à função complemento para imagens em níveis de cinza e a instrução *add* soma o resultado corrente do programa ao conteúdo da variável *img_temp* (variável de armazenamento temporário de imagens) que poderá corresponder ao último resultado previamente armazenado pela instrução *sto* ou à imagem original. Após a aplicação do filtro gerado automaticamente à imagem original, o resultado obtido foi comparado ao resultado proveniente do filtro comercial e o erro EMA calculado foi inferior a 1,9%. Na Figura 5.49 é apresentado o resultado de um exemplo similar ao anterior,

só que aplicado a uma imagem colorida. O erro EMA encontrado para este exemplo foi inferior a 1,7%. Neste exemplo, a instrução *dil_21_c* corresponde a uma dilatação para imagens coloridas através de um elemento estruturante quadrado de dimensão 2x2. Esta dilatação é baseada na abordagem de ordenação por uma componente e na distância *city-block* desenvolvida neste trabalho, assim, há a preservação das cores presentes na imagem original. Na Tabela 5.3 é apresentado um sumário de todos os resultados obtidos pelos exemplos apresentados anteriormente.

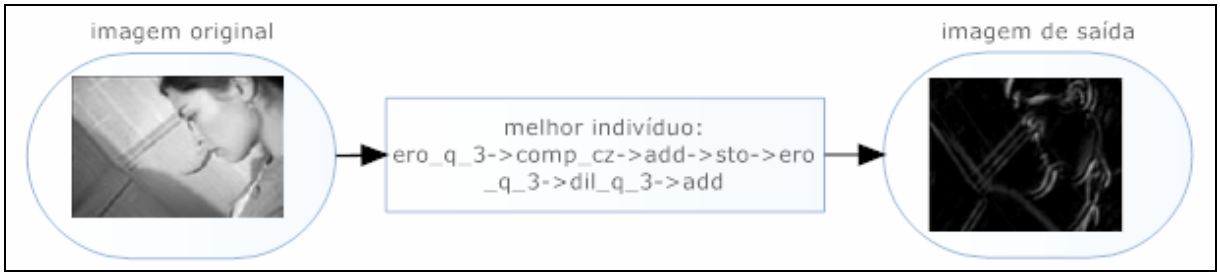


Figura 5.48. Emulação do filtro *glowing edges* de um *software* comercial

Figura 5.49. Emulação do filtro *glowing edges* colorido

Tabela 5.3. Sumário dos resultados

Exemplo	Gerações	Cromossomos	Tamanho	Cruzamento	Mutação	Reprodução	Erro EMA	Treinamento	Execução
filtro binário	50	25	8	90%	20%	20%	0,4%	4,3s	0,15s
detector de bordas	50	25	8	90%	20%	20%	0,01%	5s	0,07s
extrator de padrões musicais	50	25	6	90%	20%	20%	0,7%	71s	0,02s
filtro (níveis de cinza)	25	12	4	90%	20%	20%	2%	2,4min	0,6s
glowing edges (níveis de cinza)	50	25	7	90%	20%	20%	1,9%	2,22min	0,03s
glowing edges (colorido)	50	25	8	90%	20%	20%	1,7%	3min	0,38s

5.4.2.1 Comparações dos Resultados Obtidos com Outros Trabalhos

Em (YODA et al., 1999), o erro encontrado pelo algoritmo genético utilizado para a extração do mesmo padrão musical presente na Figura 5.45 foi de 11,8% para um cromossomo de tamanho igual a 14 formado por operadores e elementos estruturantes básicos de morfologia matemática. O tempo de treinamento foi de 2min em uma estação de trabalho *IRIS INDIGO 2* para imagens de 64x64 *pixels* e o número de gerações foi igual a 50. Nesse trabalho, não há informações sobre o tempo de execução dos procedimentos. Em (MIYAO; NAKANO, 1995) foi utilizada uma rede neural com retropropagação para o reconhecimento (baseado em regiões) do mesmo padrão musical descrito anteriormente. O erro encontrado foi de 0,8% e o tempo de processamento variou de 40 a 100s numa estação de trabalho *SPARC II*. A resolução das imagens utilizadas foi de 400dpi. O presente trabalho também oferece uma abordagem mais intuitiva (baseada em linhas de código) em relação ao algoritmo (baseado em árvores sintáticas) apresentado em (QUINTANA et al., 2002), sendo este último um dos poucos algoritmos na literatura que explora o uso de programação genética aplicada à morfologia matemática binária. No entanto, esse trabalho não traz aplicações envolvendo imagens em níveis de cinza ou coloridas. No referido trabalho foram utilizados os seguintes parâmetros genéticos: 50 indivíduos, 100 gerações, taxa de cruzamento de 90% e de mutação de 20%. No entanto, tal algoritmo não utiliza o operador genético de reprodução. O tamanho máximo das imagens de treinamento foi de 64x64 *pixels*. Os erros encontrados para a extração dos mesmos padrões musicais utilizados no corrente trabalho foram maiores que 20%. Também, nesse trabalho não são feitas considerações em relação aos tempos de treinamento e de execução do algoritmo proposto. Em (HARVEY; MARSHALL, 1996), utiliza-se um algoritmo genético para a construção de um filtro morfológico para eliminação do ruído ‘sal e pimenta’ de densidade não especificada. No entanto, o algoritmo

possui uma estrutura cromossômica fixa de tamanho igual a 4 e sua convergência é muito lenta, ocorrendo após centenas de gerações. O erro EMA encontrado nesse trabalho foi de 10,59%. Nos testes, foram utilizadas imagens em níveis de cinza apenas. Também não são feitas considerações sobre os tempos de treinamento e de execução do algoritmo utilizado. (KIM, 1997; KIM, 2004) utiliza um modelo de aprendizagem PAC (provavelmente aproximadamente correta) generalizada e vizinho mais próximo para a construção automática de operadores. Também, nessa abordagem, utilizam-se pares de imagens de entrada e de saída como exemplos de treinamento. Em um dos exemplos apresentados, uma imagem em níveis de cinza foi corrompida com o ruído ‘sal e pimenta’ de densidade: 0,25. Para a eliminação do ruído, foram utilizadas duas abordagens. Na primeira, o erro EMA encontrado foi de 4,96% e os tempos de treinamento e de aplicação foram de 1s, respectivamente. Na segunda, o erro EMA encontrado foi de 6,42%, o tempo de treinamento foi de 1s e o de aplicação foi de 36s. Nesse trabalho, foi utilizado um computador Pentium de 100MHz com 32MB de memória. Também, todas as rotinas foram implementadas em C++. As imagens utilizadas para as amostras foram de 256x256 *pixels* de 8 bits cada. Empregando o sistema evolucionário desenvolvido no presente trabalho para a eliminação do mesmo ruído, o erro EMA encontrado foi de 2,2%, o tempo de treinamento foi de aproximadamente 7min para 51 gerações de 50 cromossomos de profundidade igual a 8. O tempo de aplicação foi de 0,9s. Neste exemplo, foram utilizados apenas operadores básicos de morfologia matemática com elementos estruturantes planos contendo formas básicas. Além das aplicações sobreditas, foram realizadas outras aplicações e comparações com outros trabalhos existentes na literatura, assim, foi possível comprovar a versatilidade e a qualidade dos resultados obtidos pela abordagem adotada neste trabalho em relação às demais implementações. Na Tabela 5.4 é apresentado um sumário das comparações realizadas para os procedimentos genéticos avaliados.

Tabela 5.4. Sumário das comparações de erros realizadas para os procedimentos genéticos avaliados

Exemplo	Trabalho	Tamanho do Cromossomo	Erro
Extração de Padrões Musicais	Yoda et al., 1999	14	11,8%
Extração de Padrões Musicais	Quintana et al., 2002	-	>20%
Extração de Padrões Musicais	Tese	6	0,7%
Filtro	Harvey; Marshall, 1996	4	10,59%
Filtro	Tese	4	2,2%

5.5 ARQUITETURA *PIPELINE* RECONFIGURÁVEL EM *HARDWARE* POR INSTRUÇÕES GERADAS AUTOMATICAMENTE POR PROGRAMAÇÃO GENÉTICA PARA PROCESSAMENTO MORFOLÓGICO DE IMAGENS EM TEMPO REAL

Nesta seção, apresenta-se o desenvolvimento de uma arquitetura original, de baixo custo e reconfigurável por instruções morfológicas e lógicas geradas automaticamente pela abordagem linear de programação genética discutida na seção anterior, com o objetivo de processar imagens digitais em tempo real através do uso de FPGAs de alta complexidade. Conforme já mencionado, seus processadores são configurados através de um arquivo contendo os *opcodes* da arquitetura para cada aplicação, onde este é gerado pela *toolbox* desenvolvida neste trabalho. O sistema foi implementado através da linguagem Verilog HDL e através da técnica de descrição de projeto por diagrama esquemático usando o *software* Quartus II *Web Edition Full 6.0* da Altera. Para processamento das imagens, foram implementados 32 processadores em *pipeline*, onde cada um pode ter até 256 instruções. Cada instrução é executada em um único ciclo de *clock* com freqüência de 27 MHz. O processo de reconfiguração da arquitetura é realizado através da interface USB da placa educacional e de desenvolvimento DE2 da Altera, que contém um dispositivo FPGA da família Cyclone II. Os *pixels* das imagens a serem processadas são fornecidos por dois sistemas, sendo o primeiro

deles o sistema RGB baseado no sensor de vídeo CMOS OV7620, desenvolvido neste trabalho para processar imagens digitais à taxa de 60 quadros/s, onde cada quadro de imagem possui uma resolução espacial de 320x240 *pixels*. O segundo sistema é baseado numa câmera de vídeo comercial, também CMOS, com resolução de 640x480 *pixels* e taxa de 30 quadros/s. Também, são apresentados exemplos de aplicações práticas para a arquitetura implementada através do uso de imagens binárias, em níveis de cinza e coloridas. Para finalizar, os resultados obtidos através do *hardware* reconfigurável são comparados com outras formas de implementação.

5.5.1 Estágios da Arquitetura Desenvolvida

Na Figura 5.50, apresenta-se um diagrama em blocos simplificado do sistema de *hardware* reconfigurável desenvolvido neste trabalho. Os *pixels* de imagem são fornecidos pelos sistemas descritos anteriormente e o resultado do processamento de uma dada aplicação é mostrado num monitor RGB. A seguir, são descritas as etapas do referido sistema.

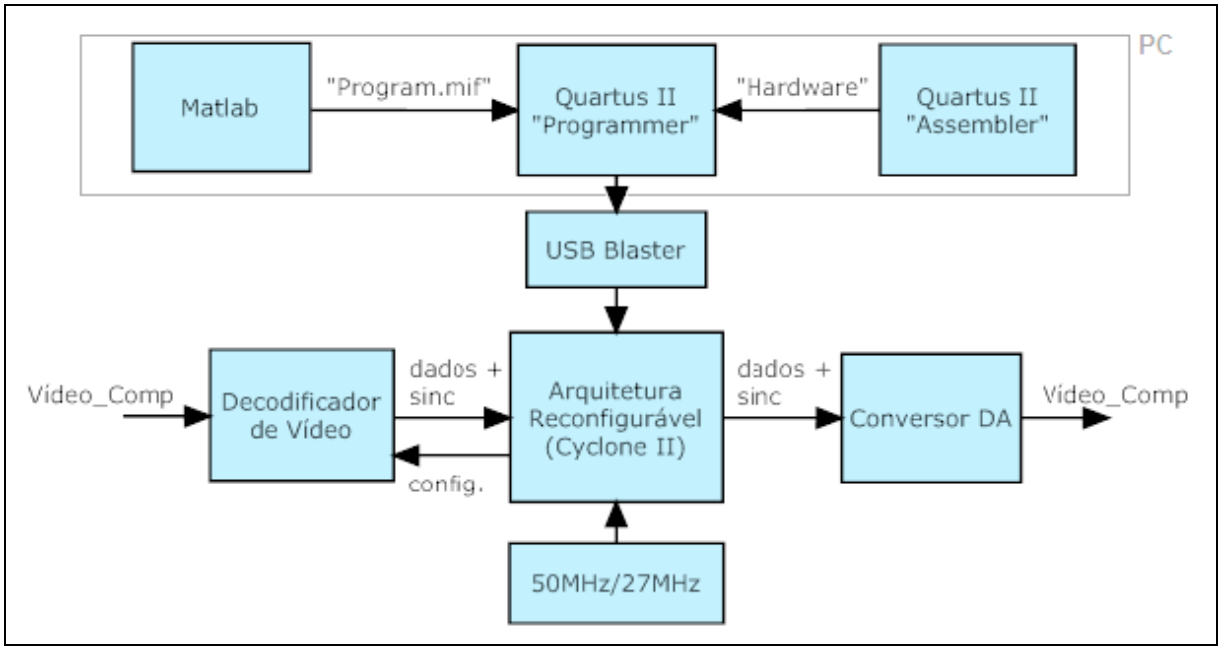


Figura 5.50. Diagrama em blocos do sistema de *hardware* desenvolvido

5.5.1.1 Decodificador de Vídeo

O objetivo principal do sistema de decodificação de vídeo implementado é converter um sinal de vídeo composto padrão de entrada em uma saída RGB digital de 10 bits de resolução radiométrica por canal, juntamente com seus sinais de sincronismo e controle: *VGA_HS* , *VGA_VS*, *VGA_SYNC*, *VGA_BLANK* e *VGA_CLK*. Para esta tarefa foi utilizado o dispositivo ADV7181 (decodificador de TV) da Analog Devices, presente na placa DE2 da Altera. Este *chip* detecta automaticamente e converte sinais padrões de vídeo analógico em componentes de vídeo 4:2:2 compatíveis com o formato digital CCIR601/656 de 16/8 bits. Seus registradores são configurados através do protocolo serial I2C. O diagrama esquemático do circuito detector de TV utilizado neste trabalho pode ser visto no Apêndice C. Na Figura 5.51, apresenta-se o diagrama em blocos do estágio detector de vídeo implementado no dispositivo FPGA da família Cyclone II da Altera. Este estágio funciona basicamente da maneira descrita a seguir. Primeiramente, o *chip* ADV7181 é configurado através do protocolo I2C implementado em Verilog HDL. Neste processo, são configurados os registradores responsáveis pela detecção do sinal de vídeo composto padrão de entrada e pela geração dos parâmetros corretos do formato digital CCIR601/656 citado anteriormente, de acordo com o exemplo para uma freqüência de 27MHz de entrada descrito no manual do dispositivo (ANALOG DEVICES, 2005). O decodificador ITU-R656 fornece em sua saída as componentes de vídeo YCrCb(4:4:4), dadas as componentes provenientes do decodificador de TV. Este sinal, ainda entrelaçado, é desentrelaçado por meio de um *buffer*, responsável por dobrar a freqüência de *pixel*. Também, o sinal de sincronismo horizontal é dobrado de acordo com o padrão VGA. Por fim, os dados YCrCb(x2) são convertidos em RGB através do decodificador YCrCb->RGB mostrado na figura.

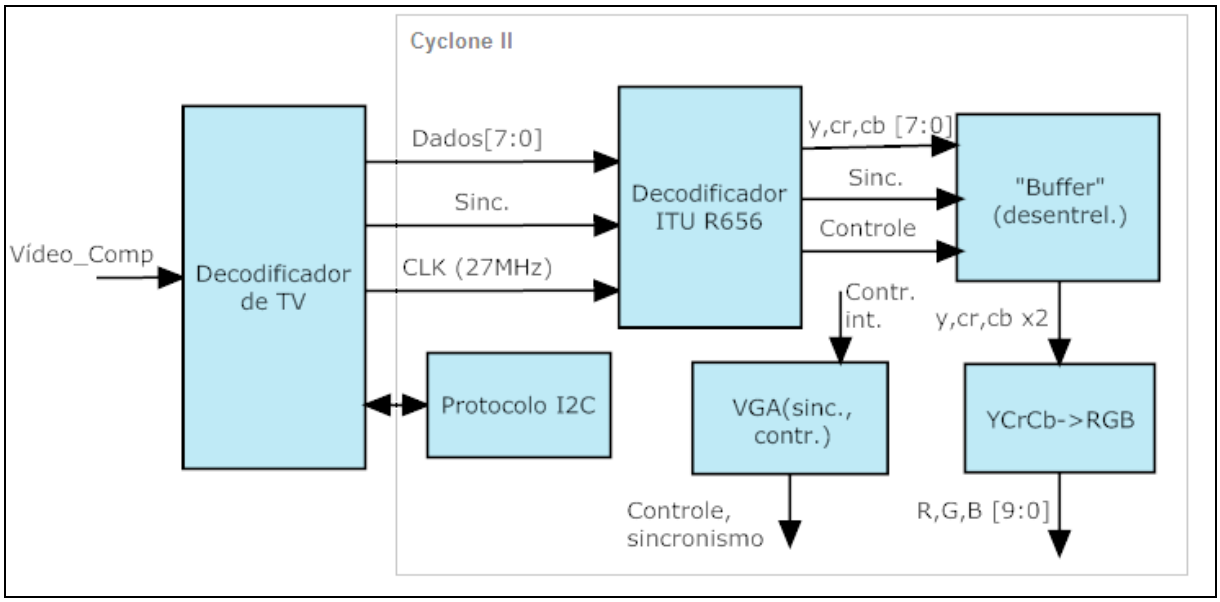


Figura 5.51. Diagrama em blocos do estágio de detecção de vídeo

5.5.1.2 Arquitetura *Pipeline*

A arquitetura *Pipeline* reconfigurável em *hardware* desenvolvida neste trabalho foi baseada no dispositivo FPGA 2C35 da família Cyclone II da Altera. Este *chip* possui como principais características: 33216 elementos lógicos, 483840 bits de memória RAM e 475 pinos de E/S (ALTERA, 2005). A arquitetura desenvolvida foi implementada através da linguagem Verilog HDL e da técnica de descrição de projeto por diagrama esquemático. Seu principal objetivo é o processamento morfológico e lógico de imagens digitais em tempo real, onde estas são fornecidas pelo estágio de decodificação de vídeo descrito anteriormente. Neste projeto, foram implementados 32 processadores em *pipeline*, conforme já supracitado, onde cada um pode ter até 256 instruções e cada instrução é executada em um único ciclo de *clock* de 27 MHz. Também, a arquitetura é configurada através da interface USB da placa DE2. O programa *program.mif* gerado automaticamente pela *toolbox* implementada é transferido para o *chip* juntamente com os demais arquivos de projeto durante o processo de programação do dispositivo através do *software* Quartus II. Na Figura 5.52 é apresentado o

diagrama em blocos da arquitetura *pipeline* implementada. Cada processador da arquitetura é responsável pelo processamento de uma única instrução do programa *program.mif* que contém os *opcodes* da arquitetura para cada aplicação.

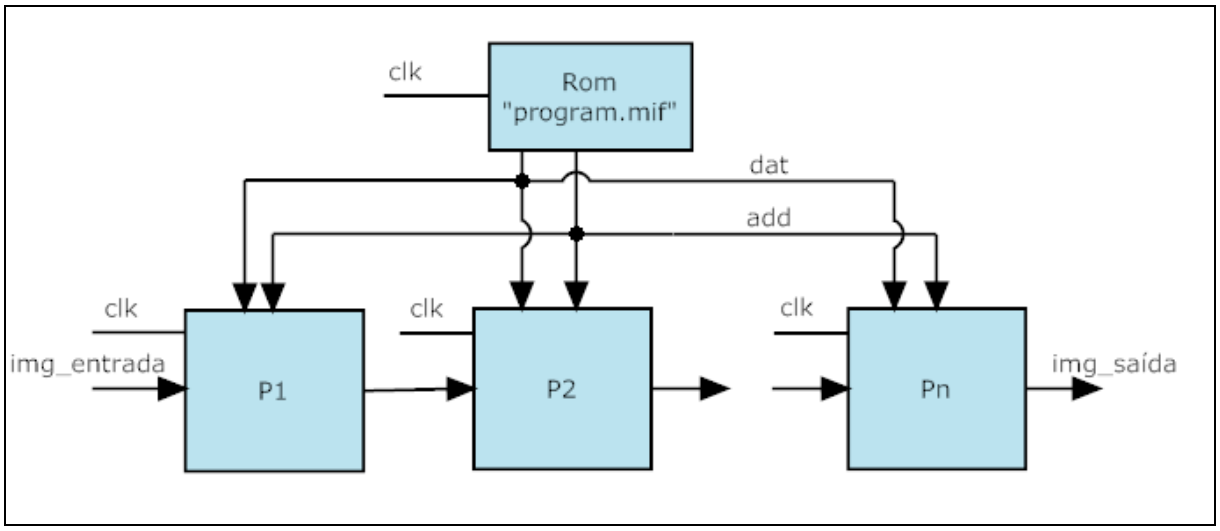


Figura 5.52. Diagrama em blocos da arquitetura *pipeline* desenvolvida

A unidade ROM, presente na Figura 5.52, foi implementada através de uma máquina de estados finitos desenvolvida em Verilog HDL e é responsável pelo carregamento das instruções provenientes do arquivo *program.mif* em cada estágio da arquitetura. Seu funcionamento é descrito conforme a seguir. Após o processo de *reset* da arquitetura, a máquina de estados entra no estado de carregamento gerando os endereços de cada estágio no barramento de endereços e as instruções referentes a cada endereço no barramento de dados. Assim, cada processador armazena as instruções referentes a seu endereço em seu respectivo registrador de instruções para que a mesma possa ser decodificada. Após o estado de carregamento, a máquina de estados sinaliza o início do período de processamento de vídeo e permanece neste estado até o próximo *reset*. Na Figura 5.53, apresenta-se o diagrama de estados da máquina de estados implementada na unidade ROM e na Figura 5.54 é apresentado

um exemplo de simulação de seu funcionamento. O diagrama esquemático desta unidade pode ser visto na Figura 5.55.

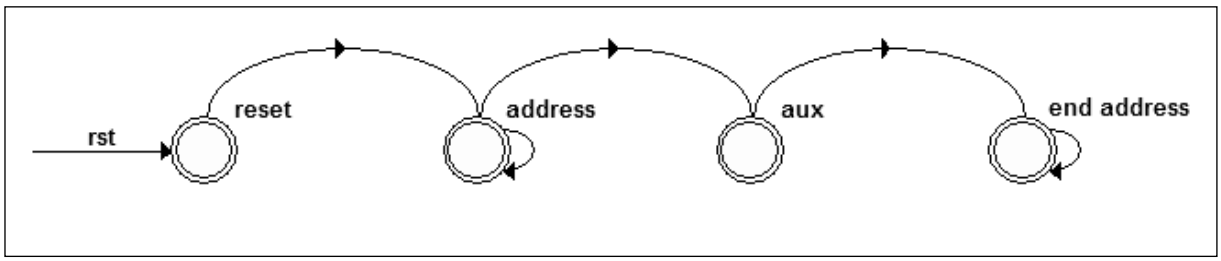


Figura 5.53. Diagrama de estados da máquina de estados da unidade ROM

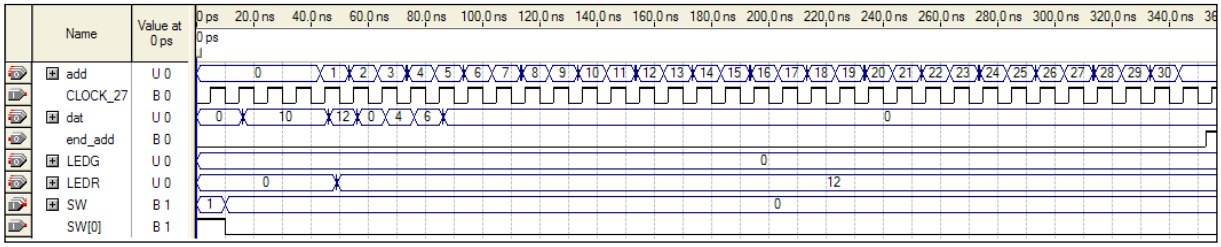


Figura 5.54. Exemplo de simulação de operação da máquina de estados implementada em ROM. Neste exemplo é possível verificar as instruções referentes a cada estágio (representado por seu endereço *add*) no barramento *dat.* Após o estado de carregamento, o pino *end_add* sinaliza o início do período de processamento de vídeo

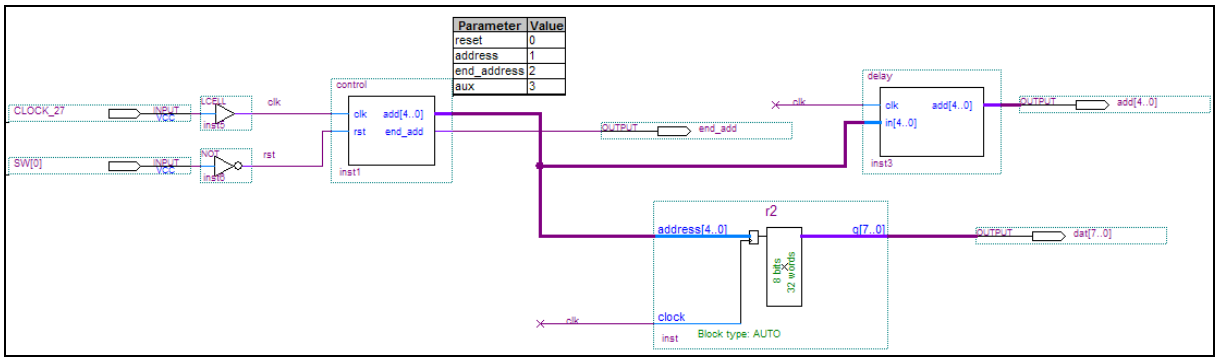


Figura 5.55. Diagrama esquemático da unidade ROM. O bloco *control* corresponde à máquina de estados implementada e o bloco *r2* corresponde à memória de programa

A avaliação em *hardware* de um cromossomo pela arquitetura *pipeline* é realizada conforme o exemplo apresentado na Figura 5.40. As imagens de entrada de cada estágio são apresentadas no formato *raster* e cada estágio da arquitetura desenvolvida possui registradores

de deslocamento, criados através de *megafunctions* customizadas pelo *software* Quartus II, para armazenagem de duas linhas adjacentes de 640 *pixels*. Registradores de janela mantêm os dados que constituem a entrada 3x3 necessária às operações morfológicas e lógicas envolvidas em cada aplicação. Os *opcodes* implementados em cada estágio de processamento foram desenvolvidos inteiramente em Verilog HDL. Além das instruções morfológicas e lógicas, também foram implementadas instruções aritméticas necessárias às operações envolvendo imagens em níveis de cinza e coloridas. Na Figura 5.56 são apresentados os símbolos de um elemento de processamento para imagens binárias, um para imagens em níveis de cinza e um para imagens coloridas desenvolvidos neste trabalho. Na Figura 5.57 é apresentada uma ilustração simplificada do diagrama esquemático da arquitetura desenvolvida contendo 32 elementos de processamento interligados. Além das unidades já descritas, a arquitetura também possui blocos de *bypass* e de limiarização digital necessária ao ajuste da iluminação ambiente que podem ser controlados pelo usuário em tempo real. Na Tabela 5.5 é apresentado um resumo das principais instruções implementadas.

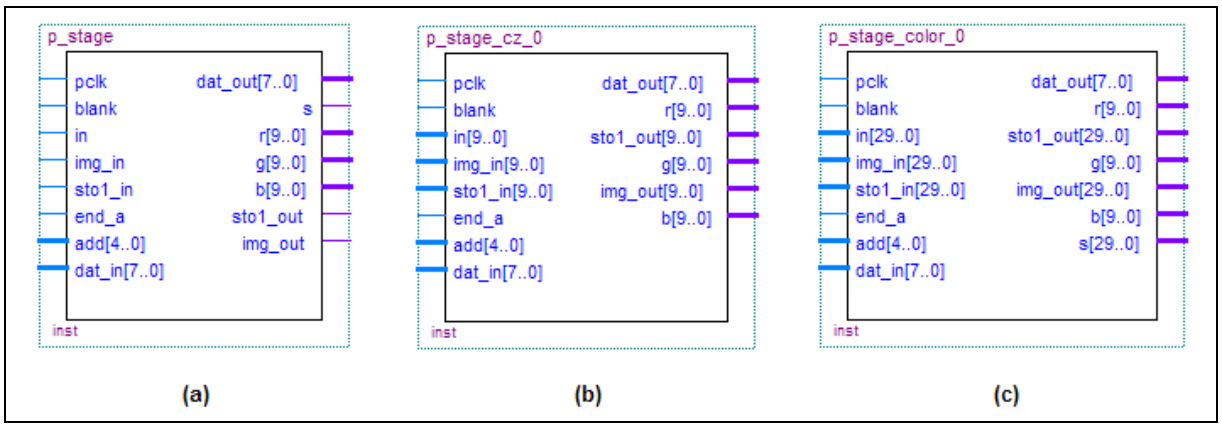


Figura 5.56. Símbolos dos elementos de processamento desenvolvidos. Em (a), *in* corresponde à entrada de *pixels* binários provenientes do estágio anterior de processamento e *s* corresponde à saída processada. Em (b), *in[9..0]* corresponde à entrada de *pixels* em níveis de cinza e a saída processada pode ser obtida em *r[9..0]*, *g[9..0]* ou *b[9..0]* . Em (c), *in[29..0]* corresponde à entrada de um *pixel* colorido (10 bits por componente), onde internamente é realizado um processo de entrelaçamento de *pixels* necessário para a formação de uma estrutura de reticulado completo requerida pelas operações morfológicas em imagens coloridas. A saída processada é obtida em *r[9..0]*, *g[9..0]* e *b[9..0]*

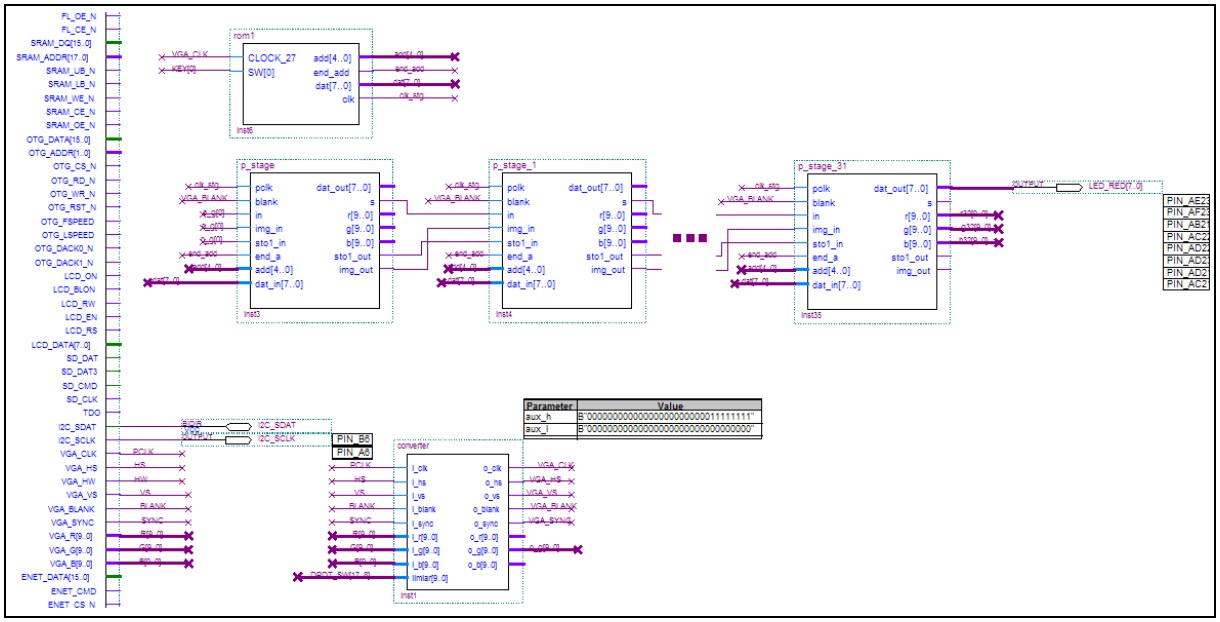


Figura 5.57. Exemplo simplificado de interligação dos elementos de processamento da arquitetura *pipeline* desenvolvida

Tabela 5.5. Sumário dos principais *opcodes* implementados

Opcode	Nome	Comentário
0x00	nada	Cópia da imagem de entrada na saída do elemento de processamento correspondente.
0x01	dil_q_3	Dilatação da imagem de entrada por um elemento estruturante quadrado de dimensão 3x3.
0x02	ero_q_3	Erosão da imagem de entrada por um elemento estruturante quadrado de dimensão 3x3.
0x03	dil_c_3	Dilatação da imagem de entrada por um elemento estruturante circular de dimensão 3x3.
0x04	ero_c_3	Erosão da imagem de entrada por um elemento estruturante circular de dimensão 3x3.
0x05	dil_h_3	Dilatação da imagem de entrada por um elemento estruturante horizontal de dimensão 1x3.
0x06	ero_h_3	Erosão da imagem de entrada por um elemento estruturante horizontal de dimensão 1x3.
0x07	dil_v_3	Dilatação da imagem de entrada por um elemento estruturante vertical de dimensão 3x1.
0x08	ero_v_3	Erosão da imagem de entrada por um elemento estruturante vertical de dimensão 3x1.
0x09	dil_dd_3	Dilatação da imagem de entrada por um elemento estruturante diagonal direito de dimensão 3x3.
0x0A	ero_dd_3	Erosão da imagem de entrada por um elemento estruturante diagonal direito de dimensão 3x3.
0x0B	dil_de_3	Dilatação da imagem de entrada por um elemento estruturante diagonal esquerdo de dimensão 3x3.
0x0C	ero_de_3	Erosão da imagem de entrada por um elemento estruturante diagonal esquerdo de dimensão 3x3.
0x0D	xor1	Ou-Exclusivo entre a imagem de entrada e uma imagem previamente armazenada proveniente de um resultado obtido anteriormente.
0x0E	cpl	Complementação lógica da imagem de entrada.
0x0F	sto1	Armazenamento temporário da imagem de entrada.
0x10	and1	And lógico entre a imagem de entrada e uma imagem previamente armazenada proveniente de um resultado obtido anteriormente.
0x11	or1	Or lógico entre a imagem de entrada e uma imagem previamente armazenada proveniente de um resultado obtido anteriormente.
0x12	ldi	Carregamento da imagem original de entrada.
0x13	cpl_cz	Operador de complementação para imagens em níveis de cinza ou coloridas.
0x14	add_cz	Soma aritmética entre a imagem de entrada e uma imagem previamente armazenada proveniente de um resultado obtido anteriormente.

Na Figura 5.58 é apresentado um exemplo hipotético de um processo de erosão binária da imagem original implementada por 2 processadores interligados de acordo com a Figura 5.59, no qual o primeiro foi programado para realizar um *bypass* (instrução *nada*) da imagem de entrada e o segundo para realizar a erosão (instrução *ero_q_3*) desta imagem por um elemento estruturante quadrado de dimensão 3x3. O resultado desta operação pode ser visto no pino de saída *s* da Figura 5.63 e a imagem de entrada correspondente pode ser vista no pino *in*.

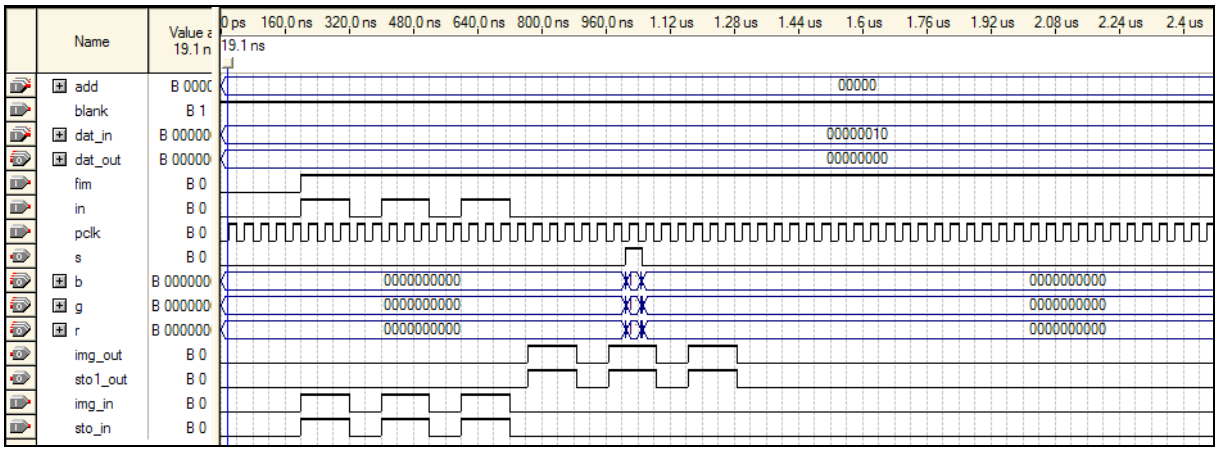


Figura 5.58. Exemplo hipotético de um processo de *bypass* da imagem original seguido por um processo de erosão morfológica por um elemento estruturante quadrado de dimensão 3x3

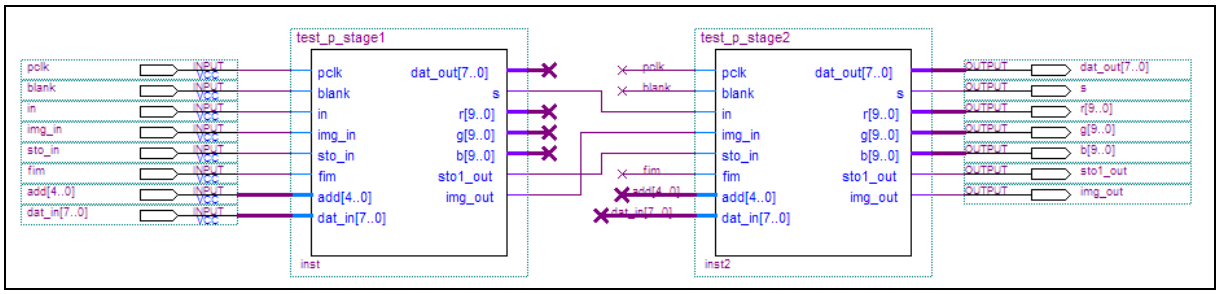


Figura 5.59. Exemplo de interligação de 2 elementos de processamento para realizar as operações dadas pelos *opcodes*: 0x00->0x02 (nada->ero_q_3)

5.5.1.3 Conversor D/A

Neste projeto foi utilizado o conversor D/A triplo de 10 bits e alta velocidade ADV7123 da Analog Devices (ANALOG DEVICES, 1998), presente na placa DE2, necessário para produzir os sinais analógicos de vídeo (RGB) de saída. A placa também possui um conector D-SUB (saída VGA) de 16 pinos. O diagrama esquemático deste circuito pode ser visto no Apêndice D. Assim, os dados digitais de vídeo processados, juntamente com os sinais VGA de sincronismo e controle são fornecidos a este circuito por meio da arquitetura *pipeline* desenvolvida. Na Figura 5.60 é apresentado o diagrama temporal básico de uma linha (horizontal) de vídeo que é mostrada num monitor VGA. Um pulso ativo em nível baixo de duração *a* é aplicado à entrada de sincronização horizontal do monitor indicando o final de uma linha de dados e o início da próxima. Durante o intervalo de tempo *b*, os dados RGB devem permanecer desligados (0V). Este período é chamado de *back porch* e ocorre logo após o pulso de sincronismo horizontal visto anteriormente. Os *pixels* de imagem são mostrados durante o intervalo de tempo *c*. Após este período, durante o intervalo de tempo *d*, chamado de *front porch*, desligam-se novamente os sinais RGB antes do próximo pulso de sincronismo horizontal. A sincronização vertical ocorre de forma análoga à horizontal, exceto que um pulso de sincronismo vertical corresponderá ao final de um quadro de imagem e ao início do próximo. Os intervalos de tempo compreendidos por *a*,*b*,*c* e *d* podem ser vistos na Tabela 5.6 para uma imagem de resolução espacial VGA (640x480 *pixels*).

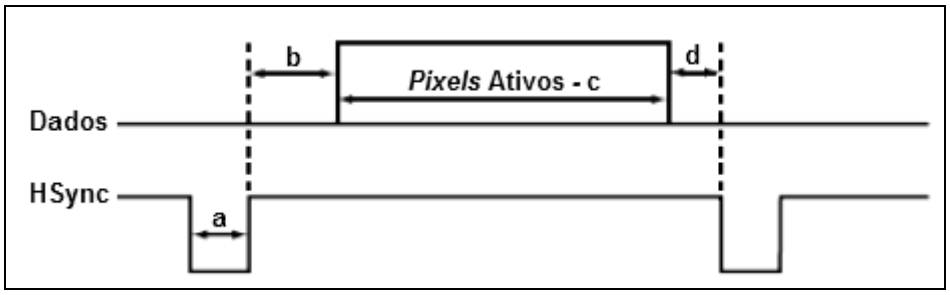


Figura 5.60. Diagrama temporal de uma linha horizontal de vídeo (VGA)

Tabela 5.6. Especificação temporal do padrão VGA adotado no trabalho

Configuração	Resolução (H x V)	a	b	c	d
VGA (60 Hz) Horizontal	640 x 480	3.8 µs	1.9 µs	25.4 µs	0.6 µs
VGA (60 Hz) Vertical	640 x 480	2 linhas	33 linhas	480 linhas	10 linhas

5.5.2 Exemplos de Aplicação utilizando o *Hardware* Desenvolvido

Em todas as aplicações descritas a seguir, as imagens envolvidas em cada caso foram obtidas através do sistema de aquisição de vídeo *Play TV Box 4* e da placa de conversão de padrões de vídeo *Game Show*, ambas da PixelView. Também, a resolução espacial da câmera de vídeo utilizada foi de 640x480 *pixels*. Nas Figuras 5.61 e 5.62 são apresentados dois resultados obtidos pelo sistema de *hardware* desenvolvido para a extração automática de padrões musicais em tempo real. Após a apresentação das imagens de treinamento, o melhor programa encontrado pelo sistema evolucionário foi: “dil_dd_3->dil_de_3->dil_dd_3->dil_v_3->dil_v_3->dil_dd_3->dil_v_3->ero_q_3->ero_v_3->ero_q_3->ero_c_3”. Para esta tarefa, foram utilizados os seguintes parâmetros genéticos: 50 gerações, 50 cromossomos, tamanho máximo de programa igual a 12, taxa de cruzamento de 97%, taxa de mutação de 3% e de reprodução de 10%. O erro médio absoluto (EMA) encontrado foi menor que 1,1%. O

tempo de treinamento para gerar a solução do problema foi de aproximadamente 12min e o processamento foi realizado em tempo real pela arquitetura desenvolvida.

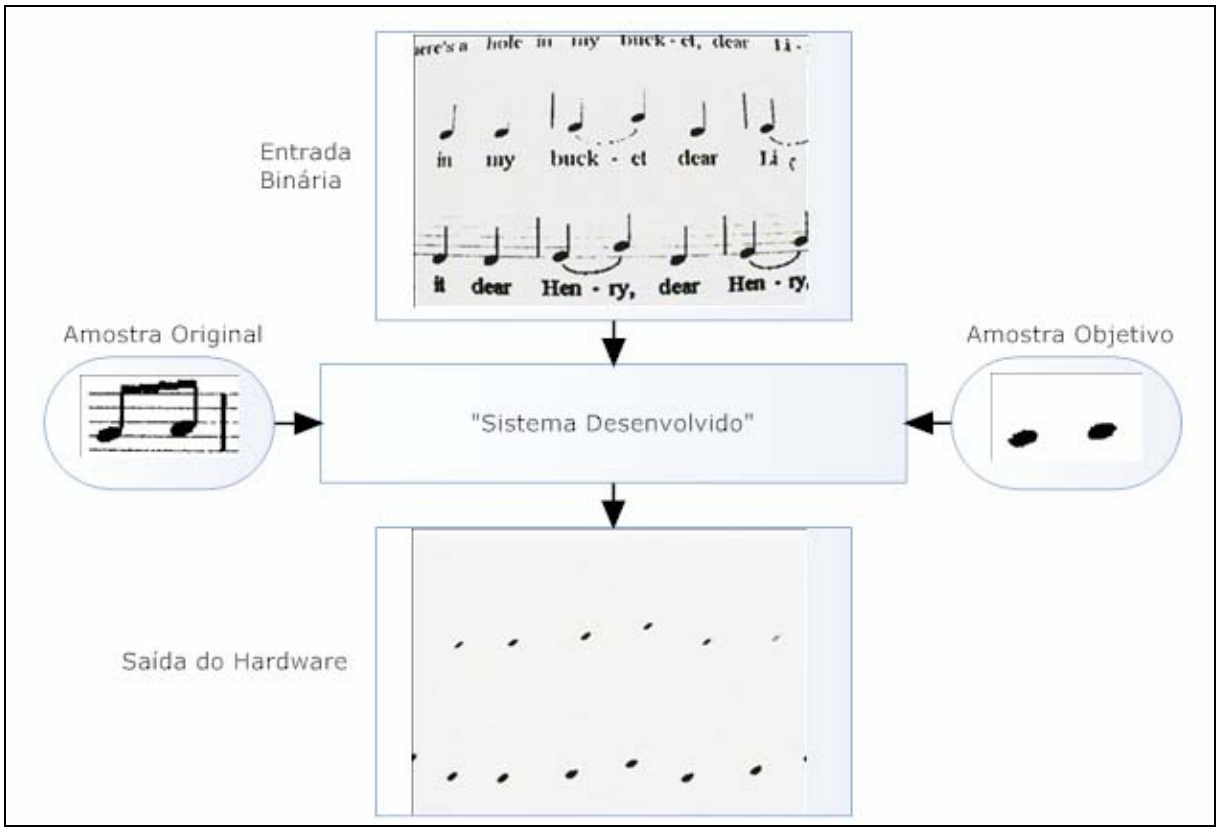


Figura 5.61. Resultado obtido pelo *hardware* desenvolvido para a extração de padrões musicais em tempo real

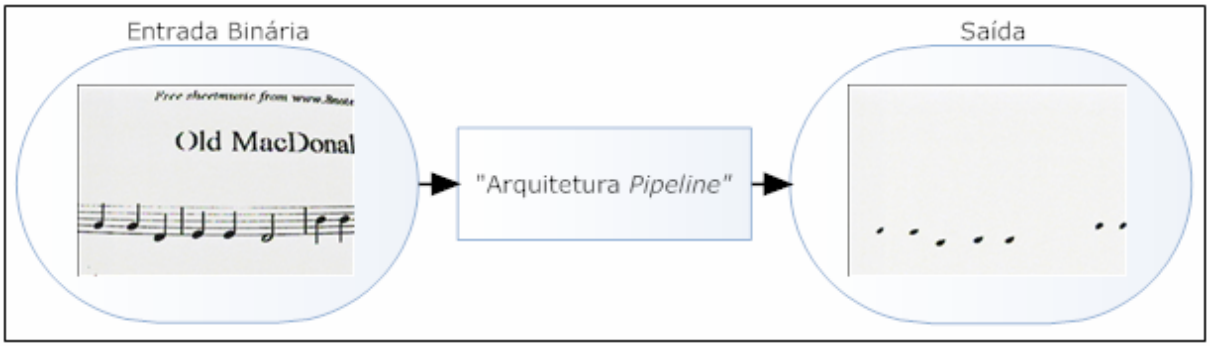


Figura 5.62. Resultado obtido pelo *hardware* desenvolvido para a extração de padrões musicais em tempo real utilizando uma partitura diferente da utilizada para o treinamento do sistema evolucionário

Nas Figuras 5.63 e 5.64 são apresentados alguns resultados obtidos através do *hardware* desenvolvido para a emulação do filtro *Trace Contour* do Photoshop 7.0. Após a apresentação das amostras de treinamento, o melhor programa encontrado pelo sistema evolucionário foi: “ero_c_3->sto1->sto1->dil_q_3->dil_c_3->cpl->cpl->dil_c_3->ero_q_3->cpl->or1->cpl->cpl”. Para esta tarefa, foram utilizados os seguintes parâmetros genéticos: 50 gerações, 90 cromossomos, tamanho máximo de programa igual a 16, taxa de cruzamento de 97%, taxa de mutação de 3% e de reprodução de 10%. O erro médio absoluto (EMA) encontrado foi menor que 2,86%. O tempo de treinamento para gerar a solução do problema foi de aproximadamente 18min e o processamento foi realizado em tempo real pelo *hardware* desenvolvido.

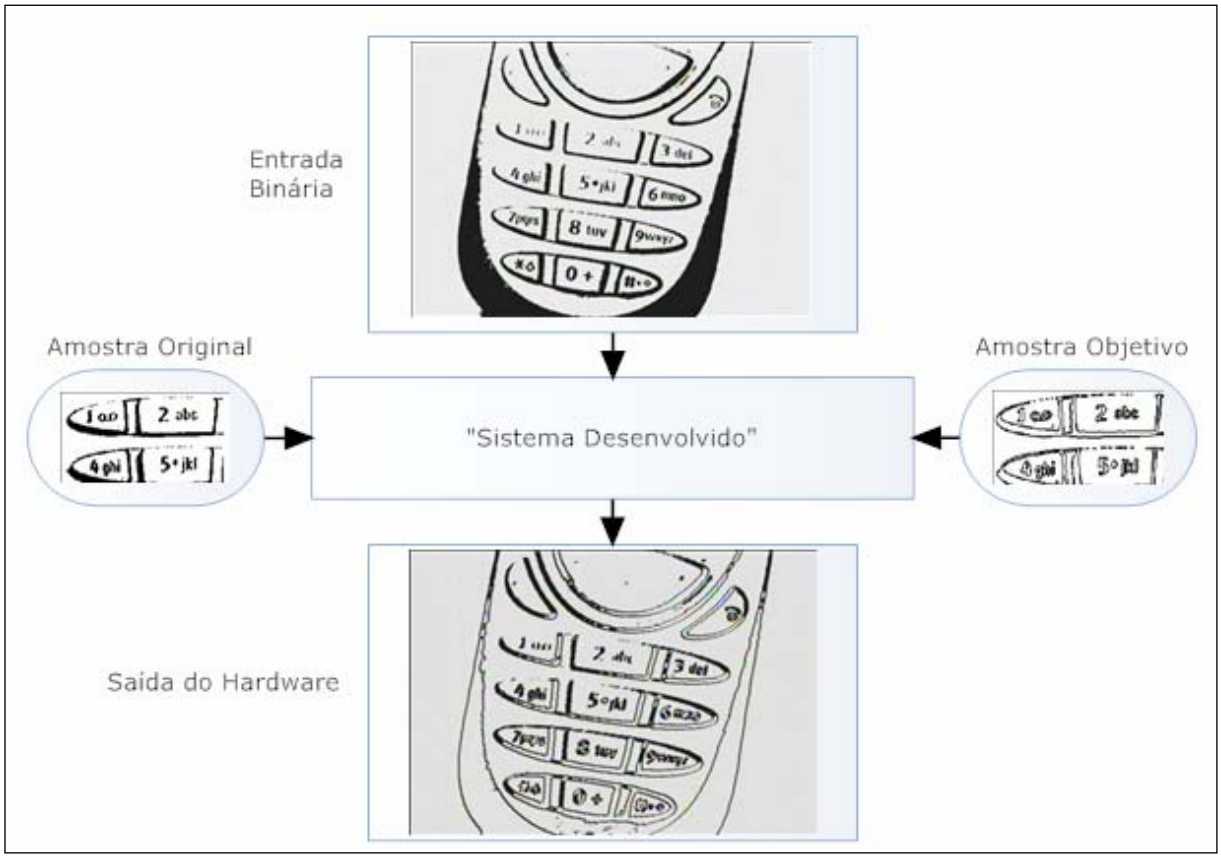


Figura 5.63. Resultado da emulação em *hardware* do filtro *Trace Contour* do Photoshop 7.0

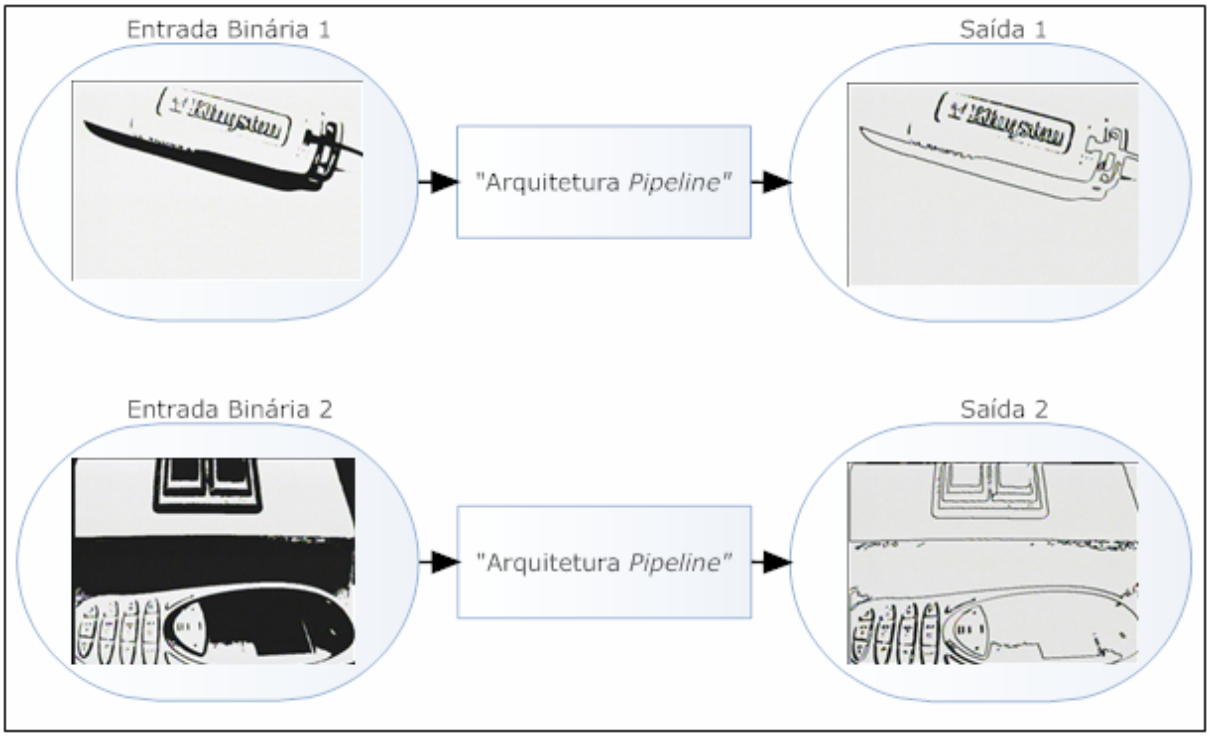


Figura 5.64. Resultados obtidos da emulação em *hardware* do filtro *Trace Contour* do Photoshop 7.0

Nas Figuras 5.65 e 5.66 são apresentados alguns resultados obtidos através da arquitetura desenvolvida para a segmentação de texto em imagens de jornais ou revistas. O melhor programa encontrado pelo sistema evolucionário para esta aplicação foi: “dil_c_3->dil_c_3->dil_c_3->cpl->dil_c_3->dil_c_3->dil_c_3->dil_c_3->dil_c_3->dil_c_3-> dil_c_3->or1”, na qual foram utilizados os seguintes parâmetros genéticos: 51 gerações, 50 cromossomos, tamanho máximo de programa igual a 12, taxa de cruzamento de 95%, taxa de mutação de 5% e de reprodução de 10%. O erro médio absoluto (EMA) encontrado foi menor que 0,3%. O tempo de treinamento para gerar a solução do problema foi de aproximadamente 13min e o processamento foi realizado em tempo real pela arquitetura desenvolvida.

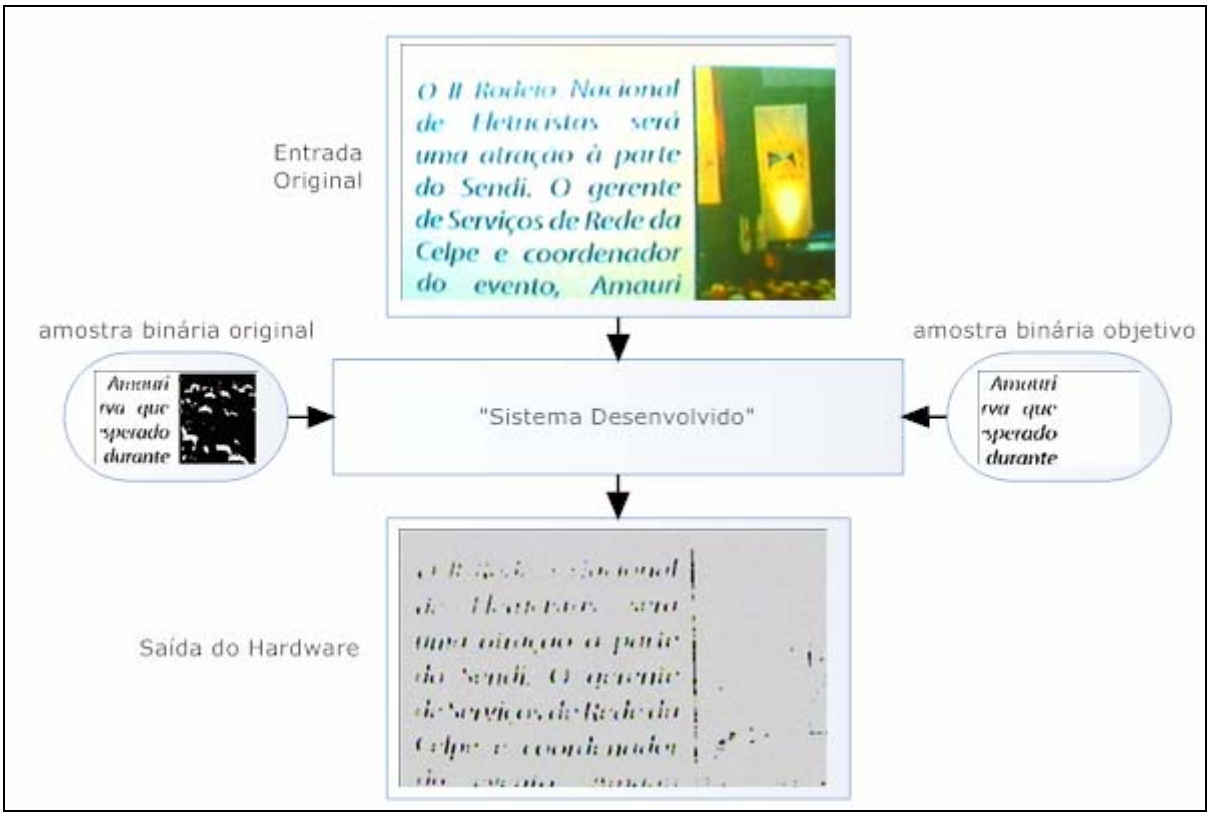


Figura 5.65. Exemplo de segmentação de texto em tempo real utilizando a arquitetura desenvolvida

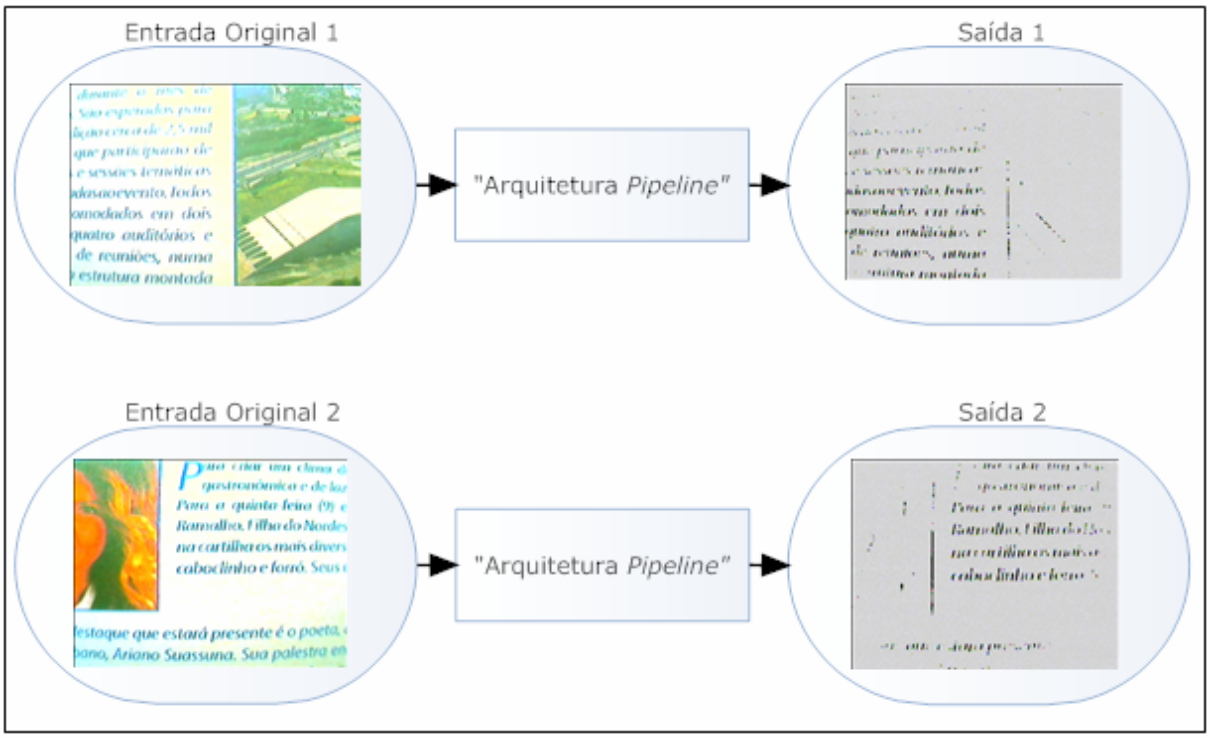


Figura 5.66. Resultados obtidos da segmentação de texto em tempo real através da arquitetura desenvolvida

Nas Figuras 5.67 e 5.68, apresentam-se alguns resultados obtidos da emulação em *hardware* do filtro *Glowing Edges* do Photoshop 7.0, para imagens em níveis de cinza, utilizando a arquitetura desenvolvida. Após a apresentação das amostras de treinamento, o melhor programa encontrado pelo sistema evolucionário foi: “mais_1->mais_1->dil_c_3->sto1->comp_cz->dil_c_3->dil_c_3->mais_1->sto1”. Para esta tarefa, foram utilizados os seguintes parâmetros genéticos: 51 gerações, 70 cromossomos, tamanho máximo de programa igual a 9, taxa de cruzamento de 95%, taxa de mutação de 20% e de reprodução de 10%. O erro médio absoluto (EMA) encontrado foi menor que 6,2%. O tempo de treinamento para gerar a solução do problema foi de aproximadamente 10,6min e o processamento foi realizado em tempo real pelo *hardware* desenvolvido. Na Figura 5.69 são apresentados alguns resultados obtidos através da arquitetura desenvolvida para a mesma emulação de filtragem descrita anteriormente aplicada a imagens coloridas. Nesta tarefa foram utilizados elementos de processamento iguais ao descrito na Figura 5.56c. Também, neste caso, foram utilizadas imagens coloridas de 512 cores, levando-se em consideração a capacidade lógica do dispositivo reconfigurável utilizado.

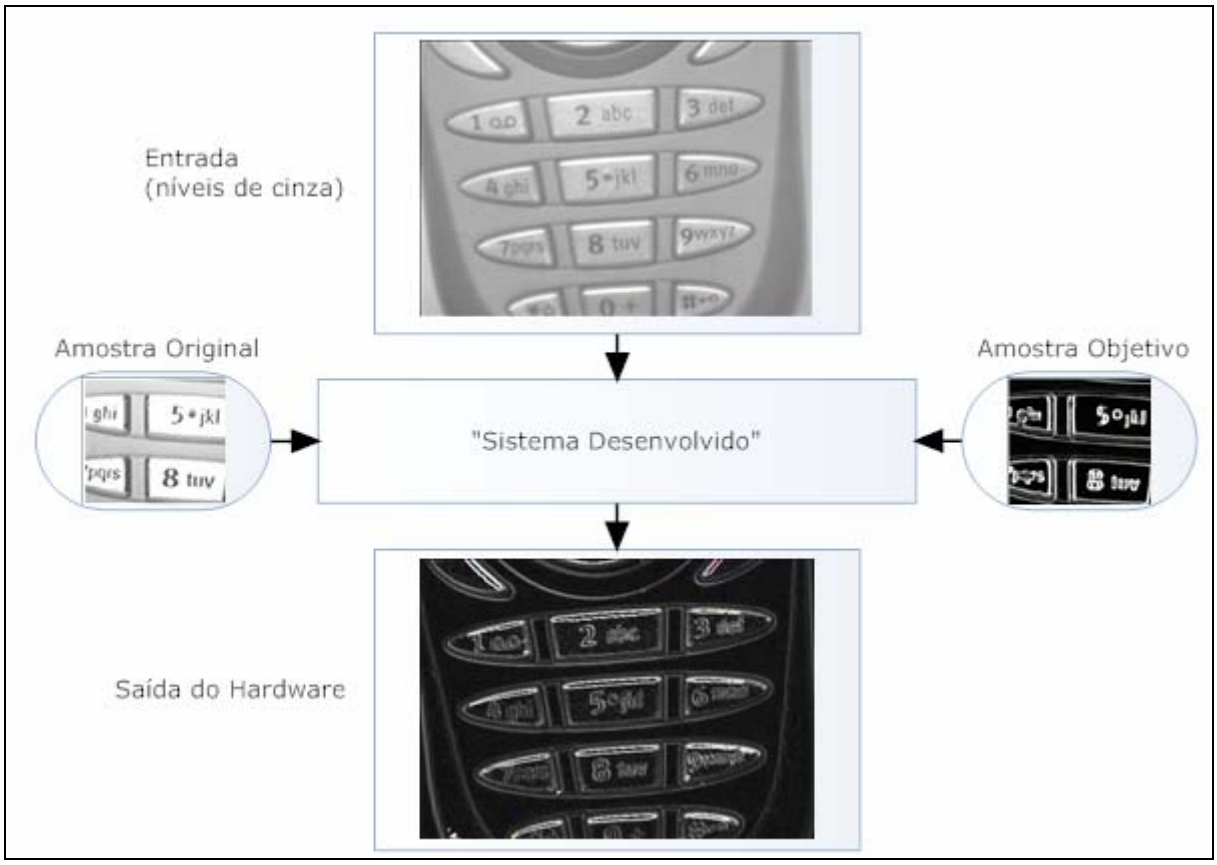


Figura 5.67. Resultado da emulação em *hardware* do filtro *Glowing Edges* do Photoshop 7.0

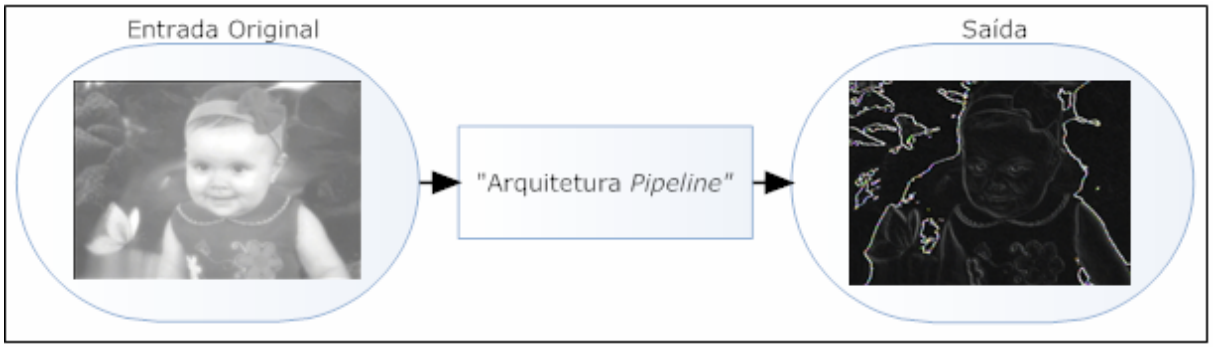


Figura 5.68. Resultado obtido através da arquitetura desenvolvida para a emulação em tempo real do filtro *Glowing Edges* do Photoshop 7.0

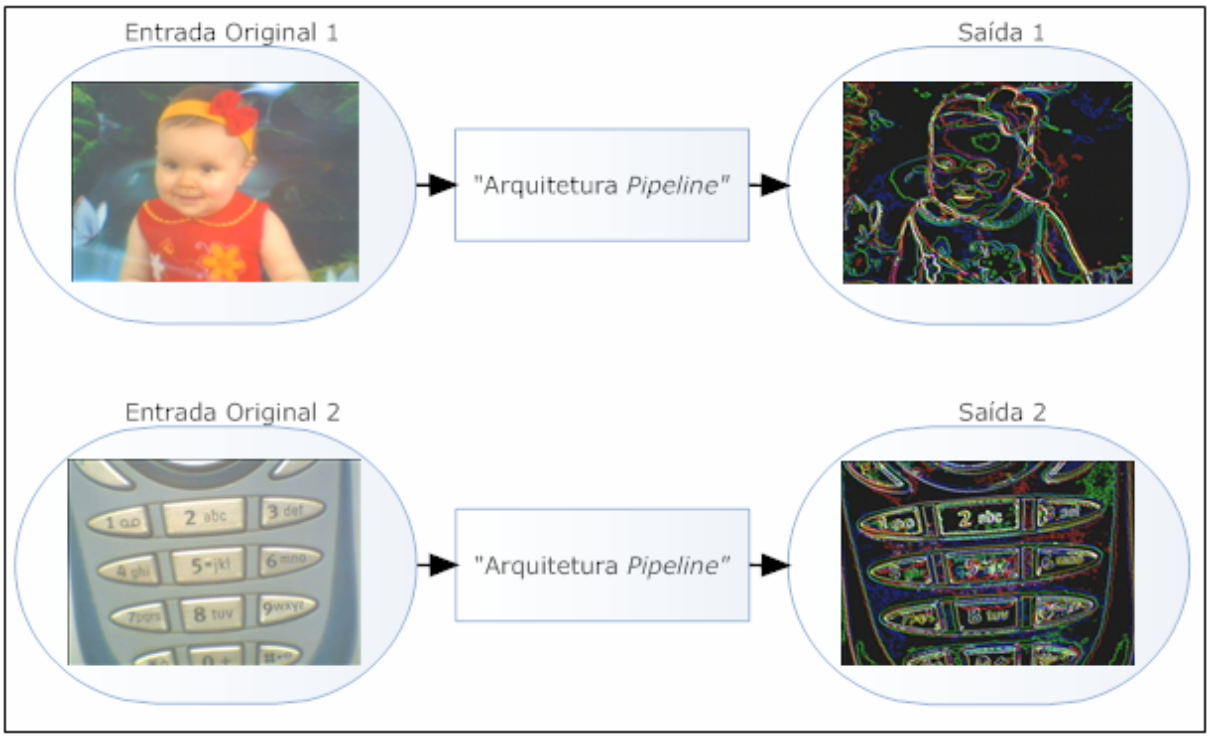


Figura 5.69. Resultados da emulação em *hardware* do filtro *Glowing Edges* colorido do Photoshop 7.0

Nos exemplos seguintes, o sistema evolucionário foi treinado para a tarefa de detecção de faces em fotografias. Para esta aplicação foi adotado o espaço de cores YCrCb, considerando-se que os *pixels* da face de um indivíduo qualquer formam um agrupamento compacto e independente da cor de pele neste domínio. Nas Figuras 5.70 e 5.71 são apresentados alguns resultados obtidos pela arquitetura desenvolvida para esta tarefa. Inicialmente foram apresentadas ao sistema as amostras de treinamento no espaço de cores YCrCb, e após a convergência do algoritmo, o melhor programa encontrado foi: “dil_q_3-> dil_q_3->ero_q_3->dil_q_3->dil_q_3->dil_q_3->dil_c_3->ero_q_3->ero_q_3->ero_q_3 ->ero_c_3“. Nesta aplicação, foram utilizados os seguintes parâmetros genéticos: 51 gerações, 25 cromossomos, tamanho máximo de programa igual a 14, taxa de cruzamento de 95%, taxa de mutação de 5% e de reprodução de 10%. O erro médio absoluto (EMA) encontrado foi menor que 1,52%. O tempo de treinamento para gerar a solução do problema foi de aproximadamente 22,6min e o processamento foi realizado em tempo real pelo *hardware*

desenvolvido. No trabalho apresentado em (SAWANGSRI et al., 2005), o menor erro encontrado para a mesma aplicação sobredita foi de aproximadamente 6,2%. Nesse trabalho os autores não abordam sobre o tempo de processamento das operações envolvidas no processo de detecção de face. Tal algoritmo é baseado em limiares nos espaços de cores RGB e YCrCb e em um processo de pós filtragem através de um filtro morfológico para eliminação de ruídos. Também, as imagens são provenientes de uma *WebCam*.

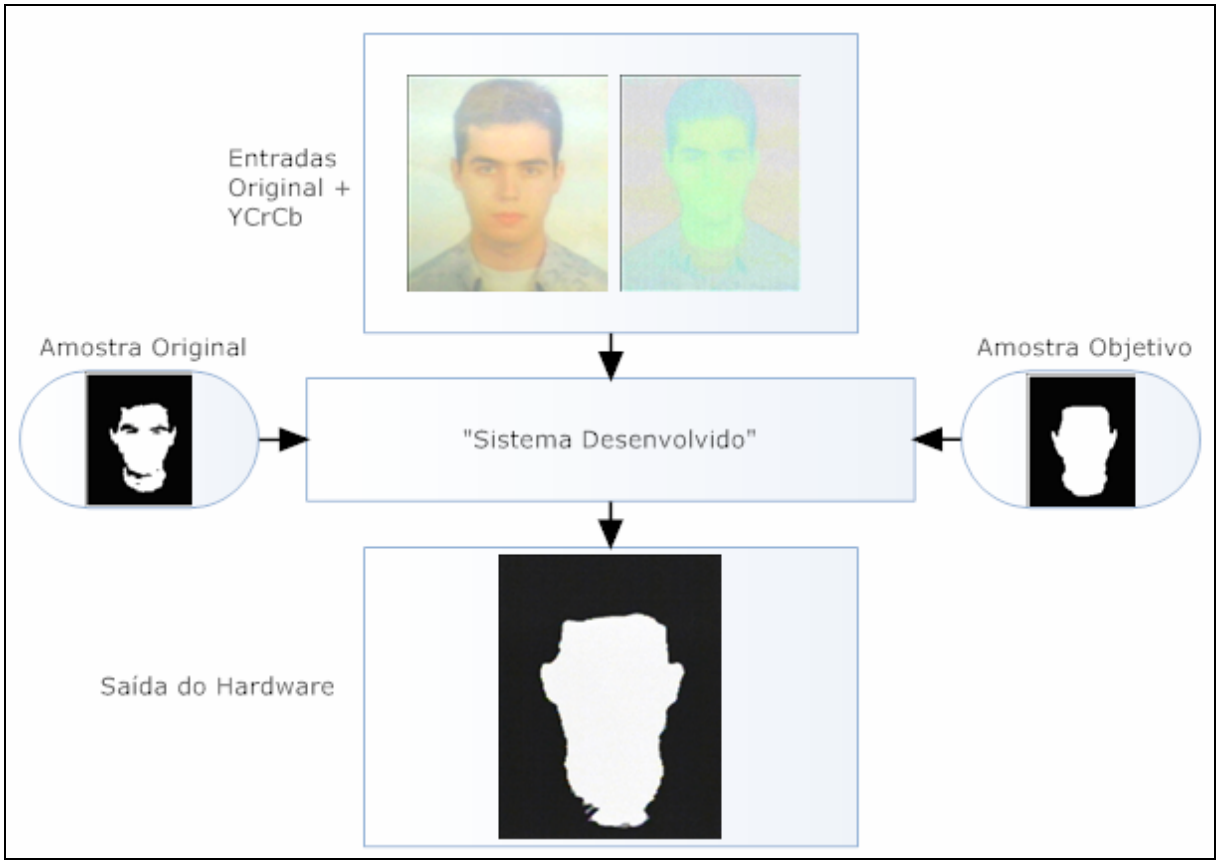


Figura 5.70. Resultado obtido da detecção de face utilizando o sistema em *hardware* desenvolvido

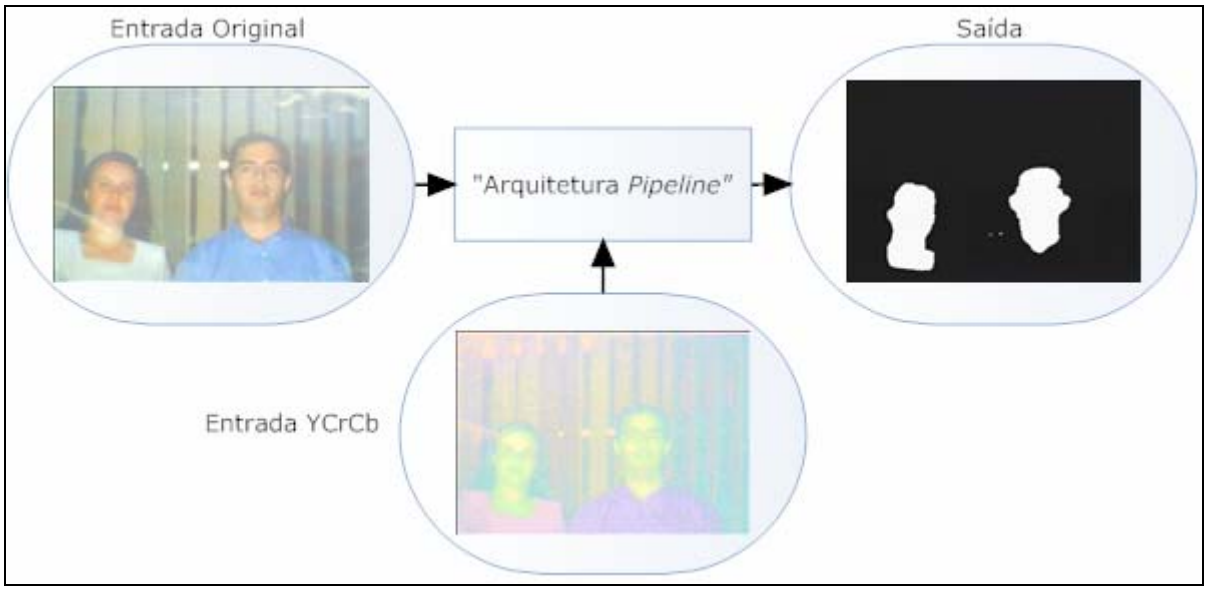


Figura 5.71. Resultado obtido da detecção de faces em tempo real através da arquitetura *pipeline* desenvolvida utilizando o mesmo algoritmo encontrado para as amostras de treinamento da Figura 5.70

Na Tabela 5.7, apresenta-se um sumário dos resultados obtidos anteriormente para fins de comparação com os resultados obtidos através da Tabela 5.3 e dos trabalhos citados na subseção 5.4.2.1. Na Tabela 5.8 são apresentados os recursos de *hardware* utilizados pelo dispositivo reconfigurável para cada aplicação dada.

Tabela 5.7. Sumário dos resultados obtidos em *hardware*

Exemplo	Gerações	Cromossomos	Tamanho	Cruzamento	Mutação	Reprodução	Erro EMA	Treinamento	Execução
extrator de padrões musicais	50	50	12	97%	3%	10%	1,1%	12min	tempo real
trace contour	50	90	16	97%	5%	10%	2,86%	18min	tempo real
segmentação de texto	51	50	12	95%	5%	10%	0,3%	13min	tempo real
glowing edges (níveis de cinza)	51	70	9	95%	20%	10%	6,2%	10,6min	tempo real
glowing edges (colorido)	51	70	9	95%	20%	10%	6,2%	10,6min	tempo real
detector de faces	51	25	14	95%	5%	10%	1,52%	22,6min	tempo real

Tabela 5.8 – Resumo de recursos de *hardware* utilizados pelo dispositivo FPGA para cada aplicação

Dispositivo Cyclone II EP2C35F672C6 / Aplicação	**Pinos Utilizados**	**Bits de Memória**	**LEs (Elementos Lógicos)**
reconhecimento de Padrões musicais	125 (26%)	134208 (28%)	2702 (8%)
trace contour	125 (26%)	134208 (28%)	2702 (8%)
segmentação de texto	125 (26%)	134208 (28%)	2702 (8%)
glowing edges (níveis de cinza)	125 (26%)	264944 (55%)	5672 (17%)
glowing edges (colorido)	125 (26%)	233032 (48%)	5249 (16%)
detector de faces	125 (26%)	88272 (18%)	1729 (5%)

5.6 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentadas as implementações em *software* e em *hardware* do presente trabalho, de acordo com os objetivos propostos, juntamente com os respectivos resultados obtidos para cada caso, nos quais estes foram comparados com outras formas de implementação existentes para fins de validação do sistema evolucionário desenvolvido. No Apêndice E, apresenta-se o diagrama esquemático de um estágio da arquitetura *pipeline* implementada e no Apêndice F, apresenta-se o código em Verilog HDL de um elemento de processamento desenvolvido neste projeto. Em todas as implementações realizadas foram utilizados elementos estruturantes planos, visando-se a simplificação das implementações em *hardware* envolvidas. Também, para operações mais complexas, estes são decompostos ao longo dos estágios da arquitetura *pipeline*. Para garantir a homogeneidade da iluminação durante os experimentos, foi utilizada uma lâmpada fluorescente de baixa potência com reator eletrônico de alta freqüência. Também, para o processo de binarização das imagens provenientes da câmera de vídeo, foi desenvolvido um sistema de limiarização digital

controlado pelo usuário em tempo real através de 10 chaves presentes na placa de desenvolvimento utilizada neste trabalho.

6. CONCLUSÕES

Nas últimas décadas, a morfologia matemática tem fornecido ferramentas poderosas para a tarefa de análise de imagens em baixo nível e tem encontrado aplicações em diversas áreas do conhecimento. Muitas dessas aplicações requerem processamento de vídeo em tempo real e geralmente são implementadas através de um *hardware* dedicado. Considerando-se também que a tarefa de projetar operadores morfológicos manualmente para uma dada aplicação não é trivial na prática, várias abordagens foram propostas na literatura para a construção automática desses operadores, entretanto, tais técnicas apresentaram diversas limitações que foram citadas neste trabalho. Assim, uma vez que a programação genética, que é um ramo relativamente novo em computação evolucionária, ainda não foi muito explorada no contexto de construção automática de operadores morfológicos e na literatura não há um número significativo de arquiteturas dedicadas em *hardware* para o processamento morfológico de imagens em geral, conforme já mencionado, neste trabalho foi idealizada e implementada uma arquitetura original, de baixo custo e reconfigurável através de instruções morfológicas e lógicas geradas automaticamente por meio de uma aproximação linear baseada em programação genética, visando-se o processamento morfológico de imagens digitais em tempo real através de FPGAs de alta complexidade para propósitos de filtragem, reconhecimento de padrões e emulação de filtros desconhecidos de *software*s comerciais, entre outras aplicações.

A arquitetura desenvolvida pode ser configurada para trabalhar com imagens binárias, em níveis de cinza e coloridas. Sua configuração é realizada através de um arquivo contendo as instruções (*opcodes* da arquitetura) para uma dada aplicação, onde este é gerado através da *toolbox morph_gen* desenvolvida em Matlab para treinamento do sistema evolucionário

proposto, através de pares de imagens (exemplos de treinamento) contendo a entrada original e a saída desejada para uma dada aplicação, conforme descrito na seção 5.4. A arquitetura desenvolvida foi implementada na placa DE2 da Altera, sendo esta baseada no dispositivo FPGA EP2C35F672C6 da família Cyclone II e configurada através da interface USB. Todas as implementações deste trabalho foram realizadas através da linguagem Verilog HDL e da técnica de descrição por diagrama esquemático usando o *software* Quartus II *Web Edition Full 6.0* da Altera. Neste projeto, foram implementados 32 processadores em *pipeline*, de até 256 instruções, onde cada uma é executada em um único ciclo de *clock* de 27 MHz. Os *pixels* das imagens a serem processadas pela arquitetura podem ser fornecidos por dois sistemas, sendo o primeiro deles um sistema RGB baseado num sensor de vídeo CMOS, desenvolvido neste trabalho para processar imagens digitais à taxa de 60 quadros/s, onde cada quadro de imagem possui uma resolução espacial de 320x240 *pixels* e o segundo é baseado numa câmera de vídeo comercial, também CMOS, com resolução de 640x480 *pixels* e taxa de 30 quadros/s. O resultado de um dado processamento pode ser visualizado num monitor RGB.

Também, foram apresentados exemplos de aplicações práticas para o sistema evolucionário proposto, nos quais foi possível verificar, através de comparações com outros trabalhos existentes na literatura, o bom desempenho obtido pela arquitetura *pipeline* desenvolvida para o processamento de imagens digitais em geral. Em todas as aplicações apresentadas neste trabalho, o processamento dos dados foi realizado em tempo real. Também, foi possível verificar a boa qualidade dos resultados obtidos pela metodologia evolucionária adotada neste trabalho na solução de problemas práticos de visão computacional. Em todos os exemplos de aplicações apresentados, e em diversos testes realizados, os erros médios encontrados em cada caso foram inferiores aos das demais implementações da literatura consultada para cada problema avaliado. Em algumas aplicações o sistema evolucionário encontrou resultados idênticos aos descritos na literatura e em outras

encontrou formas alternativas de solução para o mesmo problema, através do uso de outros tipos de operadores, que muitas vezes não são notados pelo projetista. Destaca-se também a versatilidade do sistema evolucionário desenvolvido, podendo-se este ser utilizado para uma gama muito grande de aplicações em processamento de vídeo digital. No entanto, foi possível verificar, através de muitos experimentos realizados, que o sucesso desta abordagem depende da escolha adequada dos parâmetros genéticos que são fornecidos pelo usuário, sendo esta a única intervenção humana neste processo. Parâmetros mal escolhidos podem influenciar na qualidade dos resultados obtidos. Também, dependendo da complexidade do problema, o tempo de treinamento do sistema pode ser lento, considerando-se que no processo evolucionário é necessário um número muito grande de gerações para se obter o melhor resultado para uma dada aplicação.

6.1 CONTRIBUIÇÕES

Considerando-se que na literatura há poucas arquiteturas voltadas para o processamento morfológico de imagens coloridas, conforme já citado, e sabendo-se que essas imagens possuem uma quantidade grande de informações, uma das contribuições desta tese, foi a proposta de uma arquitetura *pipeline* para o processamento morfológico de imagens coloridas em tempo real utilizando dispositivos lógicos programáveis de alta complexidade. Esta arquitetura pode ser reconfigurada para trabalhar com qualquer tamanho e forma de elemento estruturante plano e também com qualquer resolução de cor, dependendo do dispositivo reconfigurável utilizado. Também, a arquitetura proposta preserva as cores originais contidas na imagem de entrada. Outra contribuição desta tese foi a idealização e a implementação de um sistema RGB de baixo custo, baseado num sensor de vídeo CMOS

comercial, para aquisição e processamento de imagens coloridas em *hardware* reconfigurável. Através de um processo de interpolação de *pixel*s, foi possível dobrar a freqüência de quadros do sistema desenvolvido para 60 quadros/s, podendo-se assim o mesmo ser utilizado em aplicações que necessitem de processamento de vídeo em tempo real. Também, a resolução espacial adotada para este sistema foi de 320x240 *pixel*s (QVGA). A frequência utilizada neste projeto foi de 13,5 MHz. Além deste equipamento, foi desenvolvido um sistema microprocessado para a programação dos registradores internos do sensor de vídeo através da interface digital SCCBus. Outra contribuição do presente trabalho foi a idealização de uma metodologia para a geração automática de procedimentos morfológicos a serem aplicados a imagens digitais por meio de uma aproximação linear baseada em programação genética, para propósitos de filtragem, detecção de bordas, reconhecimento de padrões, entre outras aplicações. Esta metodologia foi implementada através da *toolbox morph_gen* criada com o auxílio do Matlab 7.0 para a geração de procedimentos morfológicos e lógicos a serem processados em *hardware* reconfigurável. Assim, foi possível diminuir as dificuldades inerentes do processo de projeto de operadores morfológicos para uma dada aplicação. Também, o algoritmo proposto pode ser utilizado para outras aplicações de processamento digital de imagens e permite o uso de vários tipos de instruções. Como contribuição final, destaca-se o desenvolvimento e a implementação de uma arquitetura *pipeline* de baixo custo e reconfigurável em *hardware* para o processamento morfológico de imagens em geral em tempo real. Esta arquitetura é configurada através dos procedimentos gerados automaticamente pela *toolbox morph_gen*, conforme descrito no capítulo 5. Para processar imagens coloridas, também foi desenvolvido um processador baseado na teoria de reticulados, discutida neste texto, através do qual as cores das imagens processadas são preservadas, de acordo com a seção 5.5. Esta arquitetura também pode ser utilizada para outros tipos de operações comuns em processamento de vídeo digital. No Apêndice G são apresentadas

algumas fotos dos experimentos e No Apêndice H são listadas as publicações geradas durante o doutorado.

6.2 ORIGINALIDADE

Esta proposta de trabalho é original no sentido de que, após uma exaustiva revisão bibliográfica, não foi encontrada nenhuma implementação em *hardware* com capacidade de reconfiguração de suas instruções (*opcodes*) através de programação genética para o processamento morfológico de imagens em geral em tempo real.

REFERÊNCIAS*

ABBOTT, A. et al. Pipeline Architectures for Morphologic Image Analysis. **Machine Vision and Applications**, v. 1, n. 1, p. 23-40. 1988.

ALTERA, Corp. **MAX 7000**: Programmable Logic Device Family Datasheet. 2005.

_____. **Cyclone II Device Family Datasheet**. 2005.

ALTERA, Corp. **MAX II Low-Cost Architecture**. Disponível em: < http://www.altera.com/products/devices/cpld/max2/overview/architecture/mx2-architecture.html>. Acesso em: 19 dez. 2007.

_____. **Stratix III Devices**. Stratix III Device Handbook. 2007.

ALTERA, Corp. **Altera's Development and Education Board**. Disponível em: < http://www.altera.com/education/univ/materials/boards/unv-de2-board.html>. Acesso em: 17 fev. 2008.

_____. **MAX II Product Backgrounder**. 2008.

_____. **Stratix IV Device Handbook**, v. 1, 2008.

ANALOG DEVICES, Inc. **ADV7123 – CMOS, 240 MHz Triple 10-Bit High Speed Video DAC**. 1998.

ANALOG DEVICES, Inc. **AD724 – RGB to NTSC/PAL Encoder**. 1999.

ANALOG DEVICES, Inc. **ADV7125 – CMOS, 330 MHz Triple 8-Bit High Speed Video DAC**. 2002.

ANALOG DEVICES, Inc. **ADV7181B – MultiFormat SDTV Video Decoder**. 2005.

ANDREADIS, I. et al. An ASIC for fast grey-scale dilation. **Microprocessors and Microsystems,** v. 20, p. 89-95. 1996.

ANELLI, G.; BROGGI, A.; DESTRI, G. Decomposition of Arbitrarily Shaped Binary Morphological Structuring Elements using Genetic Algorithms. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 20, n. 2, p. 217-224, feb. 1998.

ANGULO, J.; SERRA, J. Morphological Coding of Color Images by Vector Connected Filters. Centre de Morphologie Mathématique – Ecole des Mines de Paris. 2005.

ATMAR, J. W. **Speculation on the Evolution of intelligence and its possible realization in machine form**. 1976. Ph.D. thesis. New Mexico State University, 1976.

ATMEL. **AT89C2051 - 8-Bit Microcontroller with 2K Bytes Flash**. 2006.

BACK, T.; HAMMEL, U.; SCHWEFEL, H.-P. Evolutionary computation: comments on the history and current state. **IEEE Transactions on Evolutionary Computation**, v. 1, n. 1, p. 3-17, apr. 1997.

BALA, J. W.; WECHSLER, H. Shape analysis using genetic algorithms. **Pattern Recognition Letters**, v. 14, n. 12, p. 965-973, 1993.

BANZHAF, W et al. **Genetic Programming**: an introduction. Morgan Kaufmann, 1998.

BANON, G.J.F.; BARRERA, J. **Bases da Morfologia Matemática para Análise de Imagens Binárias**. 2 ed. 1998.

BARNETT, V. The ordering of multivariate data. **Journal of the Statistical Society of America A**, v. 139, n. 3, p. 318-354, 1976.

BIRKHOFF, G. **Lattice Theory**. 3rd Ed. Rhode Island: AMS Colloquium Publications, 1984.

BIRKHOFF, G.; BARTEE, T. C. **Modern Applied Algebra**. McGraw-Hill, 1970.

BLEAU, A.; DE GUISE, J.; LEBLANC, A. R. A new Set of fast Algorithms for Mathematical Morphology I. Idempotent Geodesic Transforms. **Computer Vision, Graphics and Image Processing: Image Understanding**, v. 56, n. 2, p. 178-209, 1992.

_____. A New Set of fast Algorithms for Mathematical Morphology II. Identification of Topographic Features on Grayscale Images. **Computer Vision, Graphics and Image Processing: Image Understanding**, v. 56, n. 2, p. 210-229, 1992.

BLOCH, I.; MAITRE, H. Fuzzy Mathematical Morphologies: A comparative Study. **Pattern Recognition**, v. 28, p. 1341-1387, 1995.

BOOMGAARD, R.; SMEULDERS, A. The morphological structure of images: The differential equations of morphological scale-space. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 16, n. 11, p. 1101-1113, nov. 1994.

BOURIDANE, A. et al. A high level FPGA-based abstract machine for image processing. **Journal of System Architecture**, v. 45, p. 809-824, 1999.

BOX, G. E. P. Evolutionary operation: A method of increasing industrial productivity. **Applied Statistics**, v. 6, n. 2, p. 81-101, 1957.

BRAMEIEIR, M. F.; BANZHAF, W. **Linear Genetic Programming**. Springer, 2007.

BREEN, E. J.; JONES, R. Attribute openings, thinnings and granulometries. **Computer Vision and Image Understanding**, v. 64, p. 377-389, nov. 1996.

BREMERMANN, H. Optimization through evolution and recombination. **Self-Organizing Systems**, p. 93-106, 1962.

BROGGI, A. Speeding-up Mathematical Morphology Computations with Special-Purpose Array Processors. **In Procs of the 27th HICSS**, v. 1, p. 321-330, 1994.

BROWN, S. FPGA Architectural Research: A Survey. **IEEE Design & Test of Computers**, v. 13, n. 4, p. 9-15, 1996.

BROWN, S.; ROSE, J. FPGA and CPLD Architectures: A Tutorial. **IEEE Design & Test of Computers**, v. 13, n. 2, p. 42-57, 1996.

BURGIN, G. H. Systems identification by quasilinearization and evolutionary programming. Journal of Cybernetics, v.3, n.2, p. 56-75, 1973.

CANDEIAS, A. L. B. **Aplicação da morfologia matemática à análise de imagens de sensoriamento remoto**. 1997. 188 p. Tese (Doutorado) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 1997.

CHAN, P. K.; MOURAD, S. **Digital Design Using Field Programmable Gate Arrays**. Prentice Hall, 1994.

CHANUSSOT, J., LAMBERT, P. Total ordering based on space filling curves for multivalued morphology. In: PROCEEDINGS OF THE 4TH INTERNATIONAL SYMPOSIUM ON MATHEMATICAL MORPHOLOGY AND ITS APPLICATIONS, 1998. Amsterdam, 1998. p. 51-58.

CHEN, K. Bit-Serial Realizations of a Class of Nonlinear Filters Based on Positive Boolean Functions. **IEEE Transactions on Circuits and Systems**, v. 36, n. 6, p. 785-794, jun. 1989.

CHOW, P. et al. The Design of an SRAM-Based Field-Programmable Gate Array – Part I: Architecture. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 7, n. 2, jun. 1999.

CLARK, C. R. et al. An FPGA-based network intrusion detection system with on- *chip* network interfaces. **International Journal of Electronics**, v. 93, p. 403-420, jun. 2006.

COMER, M.; DELP, E. Morphological Operations for colour image processing. **Journal of Electronic Imaging**, v. 8, p. 279-289, 1999.

COSTA, L. F.; CESAR Jr., R. M. **Shape Analysis and Classification**: Theory and Practice. CRC Press, 2001.

DE JONG, K. E. **An analysis of the behavior of a class of genetic adaptive systems**. University of Michigan Press, Ann Arbor, 1975.

DIAMANTARAS, I.; KUNG, S. Y. A Linear Systolic Array for Real-Time Morphological Image Processing. **Journal of VLSI Signal Processing**, v. 17, p. 43-55, 1997.

DIAZ, J. et al. FPGA-based real-time optical-flow system. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 16, p. 274-279, feb. 2006.

DICK, C.; HARRIS, F. Virtual signal processors. **Microprocessors and Microsystems**, v. 22, p. 135-148, 1998.

DOUGHERTY, E. R. **An Introduction to Morphological Image Processing**. Washington, SPIE, 1992.

_____. Optimal Mean-Square N-Observation Digital Morphological Filters I: Optimal binary filters. **CVGIP - Image Understanding**, v. 55, n. 1, p. 36-54, 1992.

_____. Optimal Mean-Square N-Observation Digital Morphological Filters II: Optimal Gray-Scale Filters. **CVGIP - Image Understanding**, v. 55, n. 1, p. 55-72, 1992.

DOUGHERTY, E. R.; ASTOLA, J. **An Introduction to Nonlinear Image Processing**. SPIE Press, 1994.

DOUGHERTY, E. R.; GIARDINA, C. R. **Matrix Structured Image Processing**. New Jersey : Prentice-Hall, 1987.

DOUGHERTY, E. R., SINHA, D. Computational gray-scale mathematical morphology on lattices (a computer-based image algebra). Part I: architecture. **Real-Time Imaging**, v. 1, p. 69-85, 1995.

_____. Computational gray-scale mathematical morphology on lattices (a comparator-based image algebra) part 2: Image operators. **Real-Time Imaging**, v. 1, p. 283-295, 1995.

DOUGHERTY, E. R.; ZHAO, D. Model-Based Characterization Of Statistically Optimal Design For Morphological Shape Recognition Algorithms Via The Hit-Or-Miss Transform. **Visual Communication and Image Representation**, v. 3, n. 2, p.147-160, jun. 1992.

EIBEN, A. E.; SMITH, J. E. **Introduction to Evolutionary Computing**. Springer, 2003.

ELECTRONICS123. **OV7620 - CMOS Image Sensor**. Disponível em: <http://www.electronics123.com>. Acesso em: 16 set. 2006.

FACON, J. **Morfologia Matemática**: Teoria e Exemplos. Curitiba: Editora Universitária Champagnat da Pontífica Universidade Católica do Paraná, 1996.

FITCH, J. P. et al. Threshold Decomposition of Multidimensional Ranked Order Operations. **IEEE Transactions On Circuits And Systems**, v. 32, p. 445-450, 1985.

FLYNN, M. J. Very High Speed Computing Systems. **Proceedings of the IEEE**, v. 56, n. 12, p. 1901-1909, 1966.

FOGEL, L.J. Autonomous automata. **Industrial Research**, v. 4, p. 14-19, 1962.

FOGEL, L. J. **On the organization of intellect**. 1964. PhD thesis, University of California, 1964.

FOGEL, D. B. **Evolving Artificial Intelligence**. 1992. Ph.D. thesis, University of California, San Diego, 1992.

FRIEDBERG, R. M. A learning machine: Part I. **IBM Journal of Research and Development**, v. 2, n. 1, p. 2-13, jan. 1958.

FRIEDBERG, R. M.; DUNHAM, B.; NORTH, J**.** H. A learning machine, part ii. **IBM Journal of Research and Development**, p. 282-287, jul. 1959.

GALVÃO, C. O.; VALENÇA, M. J. S. **Sistemas inteligentes**: aplicações a recursos hídricos e ciências ambientais. Porto Alegre: Ed. Universidade/UFRGS, 1999.

GASTERATOS, A. et al. Improvement of the Majority Gate Algorithm for Grey-Scale Dilation/Erosion. **Electronics Letters**, v. 32, p. 806-807, 1996.

GASTERATOS, A. **Specialized *Hardware* Structures for Morphological Image Processing**. 2000. Disponível em: <http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/GASTERATOS>. Acesso em: 10 jul. 2006.

GASTERATOS, A. Specialized *Hardware* Structures for Morphological Image Processing. 2002. Disponível em: <http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/GASTERATOS>. Acesso em: 16 set. 2006.

GENETIC PROGRAMMING. **The GP Tutorial**. Disponível em: < http://www.geneticprogramming.com/Tutorial>. Acesso em: 03 ago 2006.

GERRITSEN, F. A.; VERBEEK, P. W. Implementation of Cellular-Logic Operators Using 3*3 Convolution and Table Lookup *Hardware*. **Computer Vision, Graphics, and Image Processing**, v. 27, p. 115-123, aug. 1984.

GIARDINA, C. R.; DOUGHERTY, E. R. **Morphological Methods in Image and Signal Processing**. Prentice-Hall, Englewood Cliff, NJ, 1988.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Addison –Wesley, 1989.

GOLDBERG, D. E.; DEB, K. A comparative analysis of selection schemes used in genetic algorithms. **Foundations of Genetic Algorithms (FOGA)**, p. 69-93, 1991.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 3.ed. Addison-Wesley, 1992.

GONZÁLEZ, F. et al. (2002). Morphological processor for real-time image applications. **Microelectronics Journal**, v. 33, p. 1115-1122, jul. 2002.

GREENE, J. et al. Antifuse Field Programmable Gate Arrays. **Proceedings of the IEEE**, v. 81, n. 7, jul. 1993.

GRUAU, F. On syntactic constraints with genetic programming. **Advances in Genetic Programming II**, p. 377-394, 1996.

GUPTA, A. K. et al. Realizing Low-Cost High-Throughput General-Purpose Block Encoder for JPEG2000. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 16, p. 843-858, jul. 2006.

GUŠTIN, V. An FPGA extension to ALU functions. **Microprocessors and Microsystems**, v. 22, p. 501-508, dec. 1998.

HARALICK, R. M. et al. Image Analysis Using Mathematical Morphology. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 9, n. 4, p. 532-550, jul. 1987.

HARVEY, N. R.; MARSHALL, S. The use of genetic algorithms in morphological filter design. **Signal Processing: Image Communication**, v. 8, n. 1, p. 55-72, jan. 1996.

HAUCK, S. The Roles of FPGA's in Reprogrammable Systems. **Proceedings of the IEEE**, v. 86, n. 4, p. 615-638, apr. 1998.

HAUCK, S; DEHON, A. **Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing**. Morgan Kaufman, 2008.

HEIJMANS, H. Theoretical aspects of gray-scale morphology. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 13, n. 6, p. 568-582, 1991.

HEIJMANS, H. **Morphological Image Operators**. New York: Academic Press, 1994.

HEIJMANS, H.; RONSE, C. The algebraic basis of mathematical morphology - part I: Dilations and erosions. **Computer Vision, Graphics and Image Processing**, v. 50, p. 245-295, 1990.

HENKEL, J.; PARAMESWARAN, S. **Designing Embedded Processors. A Low Power Perspective**. Springer Verlag, 2007.

HERDY, M. Reproductive isolation as strategy parameter in hierarchically organized evolution strategies. **Parallel Problem Solving from Nature**, p. 207-217, 1992.

HIRATA, N. S. T. **Projeto Automático de Operadores**: Explorando Conhecimentos a Priori. 2000. 198 p. Tese (Doutorado) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2000.

HOLLAND, J. H. Outline for a logical theory of adaptive systems. **JACM**, v. 3, n. 9, p. 297-314, 1962.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. Ann Arbor: Univ. of Michigan Press, 1975.

HOROWITZ, P.; HILL, W. **The Art Of Electronics**. 2.ed., Cambridge University Press, 1989.

IWANOWSKI, M.; SERRA, J. Morphological interpolation and color images. In: PROCEEDINGS OF INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, 1999, Vennice, Italy.

JOVANOVIC, N. Lattice Tutorial. 2005. Disponível em: < http://www.infosys.tuwien.ac.at/Staff/enji/>. Acesso em: 10 jul. 2006.

KATZ, R. H. **Contemporary Logic Design**. University of California: Benjamin Cummings/Addison Wesley Publishing Company, 1993.

KIM, H. Y. Construção Automática de Operadores Morfológicos por Aprendizagem Computacional. 1997. 204 p. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 1997.

KIM, H. Y. Projeto de Operadores pela Aprendizagem, Difusão Anisotrópica e Marca d'Água de Autenticação. 2004. 202 p. Tese (Livre-Docência) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2004.

KLEIN, J. C.; SERRA, J. The texture analyser. Journal of Microscopy, v. 95, p. 349-356, apr. 1972.

KNAPP, S. **Frequently-Asked Questions (FAQ) About Programmable Logic**. 2000. Disponível em: <http://www.optimagic.com/faq.html>. Acesso em: 28 maio 2002.

KOJIMA, S. et al. Fast Morphology *Hardware* Using Large-Sized Structuring Element. **Systems and Computers in Japan**, v. 25, n. 6, p. 41-49, 1994.

KOSKINEN, L. et al. Soft Morphological Filters. **Proceedings of SPIE Symposium on Image Algebra and Morphological Image Processing**, p. 262-270, 1991.

KOZA, J. R. Hierarchical genetic algorithms operating on populations of computer programs. **Proceedings of the Eleventh International Joint Conference on Artificial Intelligence**, v. 1, p. 768-774, 1989.

KOZA, J. R. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**. Cambridge Massachusetts: MIT Press, 1992.

KOZA, J. R. **Genetic Programming II: Automatic Discovery of Reusable Programs**. Cambridge Massachusetts: MIT Press, 1994.

KOZA, J. R.; BENNET, F. H.; ANDRE, D.; KEANE, M. A. **Genetic Programming III: Darwinian Invention and Problem Solving**. Morgan Kaufmann Publishiers, Inc., 1999.

KOZA, J. R. ; POLI, R. **A Genetic Programming Tutorial**. Stanford University, 2003.

KURSAWE, F. A Variant of Evolution Strategies for Vector Optimization. **Parallel Problem Solving from Nature**, p. 193-197, 1991.

LANDIS, D. **Handbook of Components for Electronics**. 2.ed., H. Jones and C. Harper editors, McGraw-Hill, Inc., 1996.

LANGDON, W. B.; POLI, R. **Foundations of Genetic Programming**. Springer-Verlag, 2002.

LI, J., LI, Y. Multivariate Mathematical Morphology based on Principal Component Analysis: Initial Results in Building Extraction. In: ISPRS, 2004, Istanbul.

LI, D.; CHEN, X. Automatically Generating Triangulated Irregular Digital Terrain Model Networks by Mathematical Morphology. **Journal of Photogrammetry and Remote Sensing**, v. 46, p. 283-295, 1991.

LIMA, D. L. et al. Aplicação do modelo de cores IHS na detecção de plantas aquáticas imersas. In: XII SIMPÓSIO BRASILEIRO DE SENSORIAMENTO REMOTO, 2005, Goiânia, Brasil. INPE, p. 4115-4122, 2005.

LUCK, L.; CHAKRABATY, C. A Digit-Serial Architecture For Gray-Scale Morphological Filtering. **IEEE Transactions on Image Processing**, v. 4, p. 387-391, 1995.

MACLANE, S.; BIRKHOFF, G. **Algebra**. MacMilan, 1967.

MARAGOS, P. Lattice Image Processing: A Unification of Morphological and Fuzzy Algebraic Systems. **Journal of Mathematical Imaging and Vision**, v. 22, p. 333-353, 2005.

MARQUES FILHO, O.; VIEIRA NETO, H. Processamento Digital de Imagens. Rio de Janeiro: Brasport, 1999.

MATHERON, G. **Random sets and integral geometry**. New York: Wiley, 1975.

MAXIM. **Video Basics**. 2001.

MICHAEL, N.; ARRATHOON, R. Optoelectronic pipeline architecture for morphological image processing. **Applied Optics**, v. 36, n. 8, p. 1718-1725, mar. 1997.

MITCHELL, M. **An Introduction to Genetic Algorithms**. Cambridge, MA: The MIT Press, 1996.

MIYAO, H.; NAKANO, Y. Head and stem extraction from printed music scores using a neural network approach. **ICDAR**, p. 1074-1079, 1995.

MUHLENBEIN, H.; SCHIERKAMP-VOOSEN, D. Predictive models for the breeder genetic algorithms. **Evolutionary Computation**, v. 1, n. 1, p. 25-49. MIT Press, 1993.

NILSSON, N. J. **Artificial Intelligence**: A New Synthesis. San Francisco: Morgan Kaufmann Publishers, 1998.

NINCE, U. S. **Sistemas de Televisão e Vídeo**. 2. ed. Rio de Janeiro: LTC, 1991.

OLTEAN, M.; GROSAN, C. Encoding Multiple Solutions in a Linear Genetic Programming Chromosome. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE, p. 1281-1288, 2004.

OMNVISION. **OV7620 Single-Chip CMOS VGA Color Digital Camera**. 2000.

OMNVISION. **OmniVision Serial Camera Control Bus (SCCB) Functional Specification**. 2003.

ORTIZ, F., TORRES, F., DE JUAN, E., CUENCA, N. Colour Mathematical Morphology for Neural Image Analysis. **Real Time Imaging**, v. 8, p. 455-465, 2002.

OSTERMEIER, A.; GAWELCZYK, A.; HANSEN, N. Step-size adaptation based on non-local use of selection information. **Parallel Problem Solving from Nature**, p. 189-198, 1994.

OVERTURF, L. A.; COMER, M. L.; DELP, E. J. Color Image Coding Using Morphological Pyramid Decomposition. **IEEE Transactions on Image Processing**, v. 4, n. 2, p. 177-185, feb. 1995.

PARK, H.; CHIN, R. T. Decomposition of Arbitrarily Shaped Morphological Structuring Elements. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 17, n. 1, p. 2-15, jan. 1995.

PECHT, J. Speeding-up successive Minkowski operations with bit-plane computers. **Pattern Recognition Letters**, v. 3, p. 113-117, mar. 1985.

PEDRINO, E. C. **Arquitetura *pipeline* para processamento morfológico de imagens binárias em tempo real utilizando dispositivos de lógica programável complexa**. 2003. 143 p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2003.

PEDRINO, E. C.; RODA, V. O. Arquitetura *pipeline* para processamento morfológico de imagens binárias em tempo real. In: SIMPÓSIO LATINO AMERICANO EM APLICAÇÕES DE LÓGICA PROGRAMÁVEL E PROCESSADORES DIGITAIS DE SINAIS EM PROCESSAMENTO DE VÍDEO, VISÃO COMPUTACIONAL E ROBÓTICA, 2004, São Carlos.

_____. Sistema de aquisição, armazenamento e exibição de imagens monocromáticas utilizando CPLDS. In: SIMPÓSIO LATINO AMERICANO EM APLICAÇÕES DE LÓGICA PROGRAMÁVEL E PROCESSADORES DIGITAIS DE SINAIS EM PROCESSAMENTO DE VÍDEO, VISÃO COMPUTACIONAL E ROBÓTICA, 2004, São Carlos.

PEDRINO, E. C. et al. Sinal de Vídeo Composto Padrão. **Multiciência**, v. 6, p. 93-98, 2005.

PEDRINO, E. C.; RODA, V. O. Pipeline architecture for morphological color image processing in real time. In: II SOUTHERN CONFERENCE ON PROGRAMMABLE LOGIC, 2006, Mar Del Plata, Argentina. **FPGA Based Systems**, Madrid, ES: Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2006.

_____. Construção automática de operadores morfológicos utilizando programação genética. In: WVC'2006 – II WORKSHOP DE VISÃO COMPUTACIONAL, 2006, Universidade de São Paulo, São Carlos.

PEDRINO, E. C.; RODA, V. O. Real-time morphological pipeline architecture using high-capacity programmable logical devices. **Journal of Electronic Imaging**, v. 16, p. 023002-023009, 2007.

PELLERIN, D.; THIBAULT, S. **Practical FPGA programming in C**. 1.ed., Massachusetts: Prentice Hall PTR, 2005.

PETERS II, A. Mathematical morphology for angle-valued images. **Proceedings of SPIE, Non-Linear Image Processing VIII**, v. 3026, p. 84-94, 1997.

POLI, R.; LANGDON, W. B. A New Schema Theory for Genetic Programming with One-Point Crossover and Point Mutation. **Proceedings of the Second International Conference on Genetic Programming**, p. 278-285, jul. 1997.

POLI, R.; LANGDON, W. B. Schema Theory for Genetic Programming with one-point crossover and point mutation. **Evolutionary Computation**, v. 6, n. 3, p. 231-252, 1998.

POLI, R; MCPHEE, N. F. Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. **Genetic Programming, Proceedings of EuroGP**, 2001.

PRESTON Jr., K. Cellular Logic Computers for Pattern Recognition. **Computer**, v. 16, p. 36-47, jan. 1983.

QUINTANA, M. I. et al. Genetic programming for mathematical morphology algorithm design on binary images. **M. Proceedings of the International Conference KBCS**, p. 161-171, 2002.

RAYWARD-SMITH, V. J. et al. **Modern Heuristic Search Methods**. Chichester: Wiley, 1996.

RECHENBERG, I. Cybernetic solution path of an experimental problem. R**oyal Aircraft Establishment**, Farnborough, page Library Translation 1122, UK, aug. 1965.

RECHENBERG, I. **Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution**. Stuttgart: Fommann-Holzboog, 1973.

REEVES, C.R. **Modern Heuristic Techniques for Combinatorial Problems**. New York: John Wiley & Sons, Inc., 1993.

RICHARDSON, C. H.; SCHAFER, R. W. A Lower Bound for Structuring Element Decompositions. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 13, n. 4, p. 365-369, apr. 1991.

RODRIGUES, E. L. M. **Evolução de funções em programação genética orientada a gramáticas**. 2002. 104 p. Dissertação (Mestrado) – Universidade Federal do Paraná, Curitiba, 2002.

RONSE, C.; HEIJMANS, H. The algebraic basis of mathematical morphology - part II: Openings and closings. **Computer Vision, Graphics and Image Processing: Image Understanding**, v. 54, p. 74-97, 1991.

RONSE, C. A lattice-theoretical morphological view on template extraction in images. **Journal of Visual Communication & Image Representation**, v. 7, n. 3, p. 273-295, 1996.

RONSE, C. Why mathematical morphology needs complete lattices. **Signal Processing**, v. 21, n. 2, p. 129-154, 1990.

ROSE, J. et al. Architecture of Field-Programmable Gate Arrays. **Proceedings of the IEEE**, v. 81, n. 7, p. 1013-1029, jul. 1993.

RUDOLPH, G. Global Optimization by Means of Distributed Evolution Strategies. **Parallel Problem Solving from Nature**, p. 209-213, 1990.

SALSIC, Z.; SMAILAGIC, A. **Digital Systems Design and Prototyping Using Field Programmable Logic**. Boston: Kluwer Academic Publishers, 1997.

SAMUEL, A. L. AI: Where it has been and where it is going. **Proceedings of the Eighth International Joint Conference on Artificial Intelligence**, p. 1152-1157, 1983.

SARCA, O. V. et al. Optimal Binary Filters with Linearly Separable Preprocessing. **Nonlinear Image Processing, Proc. of SPIE**, jan. 1998.

SARCA, O. V. et al. Two-stage binary filters. **Electronic Imaging**, v. 8, n. 3, p. 219-232, jul. 1999.

SARTOR, L.; WEEKS, A. Morphological operations on colour images. **Journal of Electronic Imaging**, v. 10, p. 548-559, apr. 2001.

SAWANGSRI, T. et al. Face Segmentation using Novel Skin-Color Map and Morphological Technique. **Proceedings of World Academy of Science, Engineering and Technology**, v. 2, p. 56-59, 2005.

SCHMITT, M. Mathematical morphology and artificial intelligence: An automatic programming system. **Signal Processing**, v. 16, p. 389-401, 1989.

SCHONFELD, D. Optimal structuring elements for the morphological pattern restoration of binary images. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 16, p. 589-601, 1994.

SCHWEFEL, H. P. Projekt MHD-Staustrahlrohr: Experimentelle Optimierung einer Zweiphasenduse, Teil I. **Technischer Bericht 11.034/68**, AEG Forschungsinstitut, Berlin, 35, oct. 1968.

SCHWEFEL, H. P. **Evolutionsstrategie und numerische Optimierung**. 1975. PhD thesis, Technische Universitat Berlin, Germany, 1975.

SDCC. SDCC Small Device C Compiler. 2006. Disponível em: <http://sdcc.sourceforge.net/>. Acesso em: 16 set. 2006.

SERRA, J. **Image Analysis and Mathematical Morphology**. Califórnia: Academic Press Inc., 1982.

SERRA, J. **Image Analysis and Mathematical Morphology**: Theoretical Advances, v.2. Academic Press Inc., 1988.

SERRA, J. (1986). Introduction to Mathematical Morphology. **Computer Vision, Graphics, and Image Processing**, v. 35, p. 283-305, mar. 1986.

SERRA, J. Anamorphoses and function lattices (multivalued morphology). **Mathematical Morphology in Image Processing**, p. 483-523, 1993.

SERRA, J.; VINCENT, L. An Overview of Morphological Filtering. **Circuits Systems Signal Process**, v. 11, n. 1, p. 48-107, 1992.

SHIH, F. Y. C.; MITCHELL, O. R. Threshold Decomposition of Gray-Scale Morphology into Binary Morphology. **IEEE Transactions On Pattern Analysis And Machine Intelligence**, v. 11, n. 1, p. 31-42, jan. 1989.

SHIH, F. Y. C.; MITCHELL, O. R. Decomposition of grayscale morphological structuring elements. **Pattern Recognition**, v. 24, n. 3, p. 195-203, 1991.

SHIH, F. Y.; WU, H. Decomposition of geometric-shaped structuring elements using morphological transformations on binary images. **Pattern Recognition**, v. 25, n. 10, p. 1097-1106, feb. 1992.

SILVA, A. J. M. **Implementação de um algoritmo genético utilizando o modelo de ilhas**. 2005. 73 p. Dissertação (Mestrado) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2005.

SISKOS, S. et al. Analog Implementation of Fast Min/Max Filtering. **IEEE Transactions on Circuits and Systems**, v. 45, n. 7, p. 913-918, jul. 1998.

SOILLE, P. **Morphological Image Analysis. Principles and Applications**. Berlin: Springer-Verlag, 1999.

SOLDEK, J.; MANTIUK, R. (1999). A reconfigurable processor based on FPGAs for pattern recognition, processing, analysis and synthesis of images. **Pattern Recognition Letters**, v. 20, p. 667-674, 1999.

SONG, J.; DELP, E. J. The analysis of morphological filters with multiple structuring elements. **Comput. Vis., Graph., Image Process.**, v. 50, p. 308-328, 1990.

SONKA, M., HLAVAC, V., BOYLE, R. **Image Processing, Analysis and Machine Vision**. Chapman & Hall Computing, 1993.

SPEARS, W. M. et al. **An Overview of Evolutionary Computation**. 1993. Disponível em: <http://www.cs.uwyo.edu/~wspears/overview/ecml93.all.html>. Acesso em: 31 jul. 2006.

STERNBERG, S. R. Biomedical Image Processing. **Computer**, v. 16, p. 22-34, 1983.

STERNBERG, S.R. Grayscale morphology. **Computer Vision, Graphics and Image Processing**, v. 35, p. 333-335, 1986.

TALBOT, H., EVANS, C.; JONES, R. (1998). Complete Ordering and Multivariate Mathematical Morphology. **Mathematical Morphology and Its Applications to Image and Signal Processing**, p. 27-34, 1998.

TANENBAUM, A. S. **Structured Computer Organization**. London: Prentice- Hall International (UK) Limited, 1990.

THOMAS, F. et al. (1999). A *hardware*/*software* codesign for improved data acquisition in a processor based embedded system. **Microprocessors and Microsystems**, v. 24, p. 129-134, sep. 1999.

TROPE, M. FPGAs: past, present, future. **Electrical Engineering and Computer Science**. University of Kansas, may. 2004.

TURING, A. M. Computing machinery and intelligence. **Mind**, v. 59, p. 433-460, 1950.

TURLEY, J. Soft computing reconfigures designer options. **Computer Design**, p. 76-82, apr. 1997.

VASSÁNYI, I. FPGAs and cellular algorithms: Two implementation examples. **Journal of System Architecture**, v. 43, p. 23-26, 1997.

VILLASENOR, J.; SMITH, W. H. M. Configurable Computing. **Scientific American**. 1997. Disponível em: <http://www.sciam.com/0697issue/0697villasenor.html>. Acesso em: 28 Maio 2002.

WEEKS Jr., A. R. (1996). **Fundamentals of Electronic Image Processing**. Washington: SPIE, 1996.

WIKIPEDIA. **Field-programmable gate array**. 2008. Disponível em: < http://en.wikipedia.org/wiki/FPGA>. Acesso em: 24 fev. 2008.

WILSON, S. S. Training Structuring Elements in Morphological Networks. **Mathematical Morphology in Image Processing**, 1993.

XILINX. **Virtex-II Pro and Virtex-II Pro X Platform FPGAs**: Complete Data Sheet. Product Specification, 2007.

_____. **Spartan-3 FPGA Family**: Complete Data Sheet. Product Specification, 2007.

YANG, J. Y.; CHEN, C. C. Decomposition of additively separable structuring elements with applications. **Pattern Recognition**, v. 26, n. 6, p. 867-875, 1993.

YANG, H. T.; LEE, S. J. Optimal decomposition of morphological structuring elements. **Proceedings of IEEE International Conference on Image Processing**, v. 3, p. 1-4, 1996.

YOUNG, I.T.; GERBRANDS, J.J.; VAN VLIET, L.J. **Fundamentals of Image Processing**. Printed in The Netherlands at the Delft University of Technology, 1998.

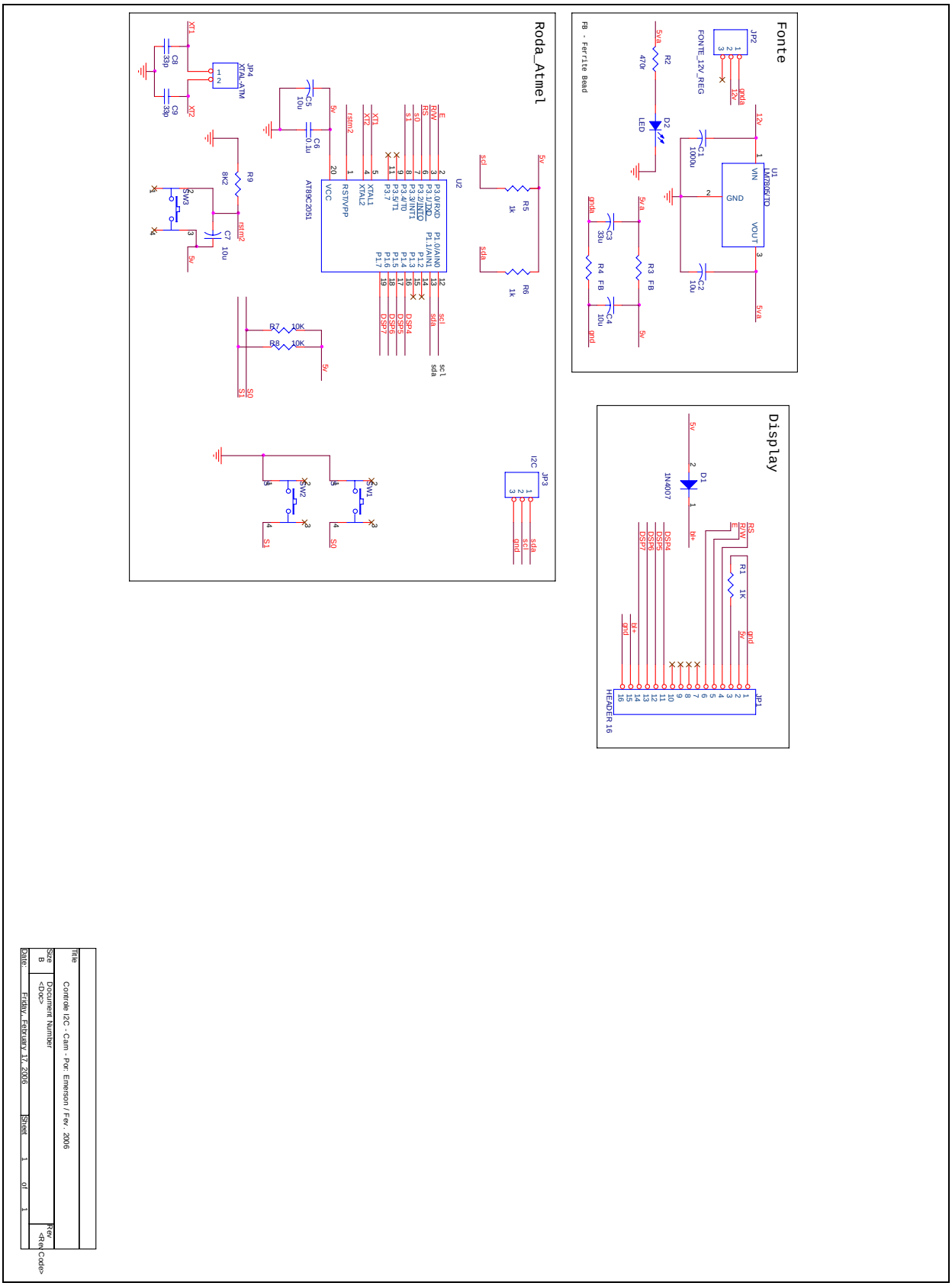
YODA, I. et al. Automatic acquisition of hierarchical mathematical morphology procedures by genetic algorithms. **Image and Vision Computing**, v. 17, n. 10, p. 749-760, 1999.

ZAMORA, F. G. O. **Procesamiento Morfológico De Imágenes En Color. Aplicación A La Reconstrucción Geodésica**. 2002. 215 p. Tese (Doutorado) – Escuela Politécnica Superior, Universidad de Alicante, Alicante, 2002.

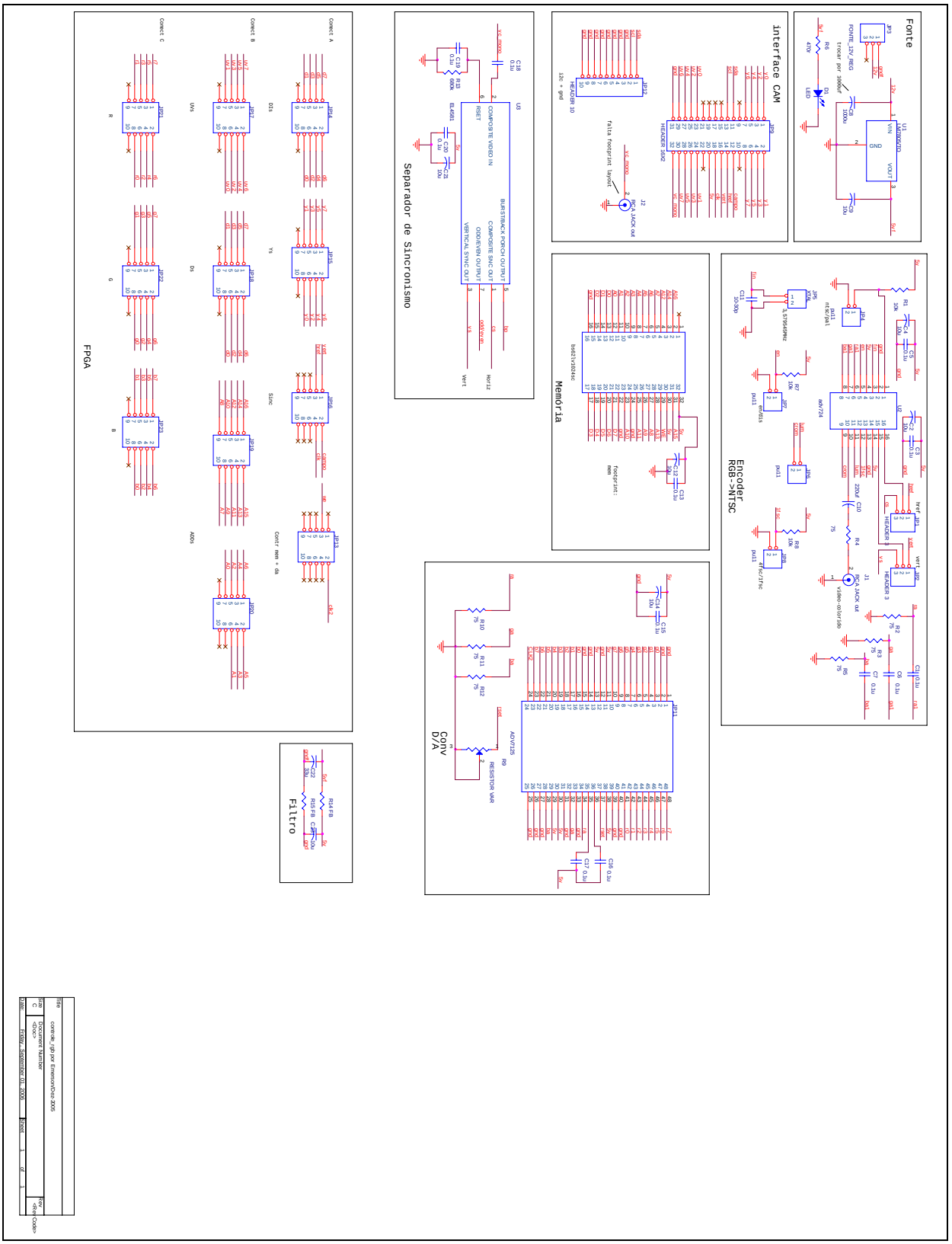
ZHUANG, X.; HARALICK, R. M. Morphological Structuring Element Decomposition. **Computer Vision, Graphics, and Image Processing**, v. 35, p. 370-382, sep. 1986.

* De acordo com:
ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023**: informação e documentação: referências: elaboração. Rio de Janeiro, 2002.

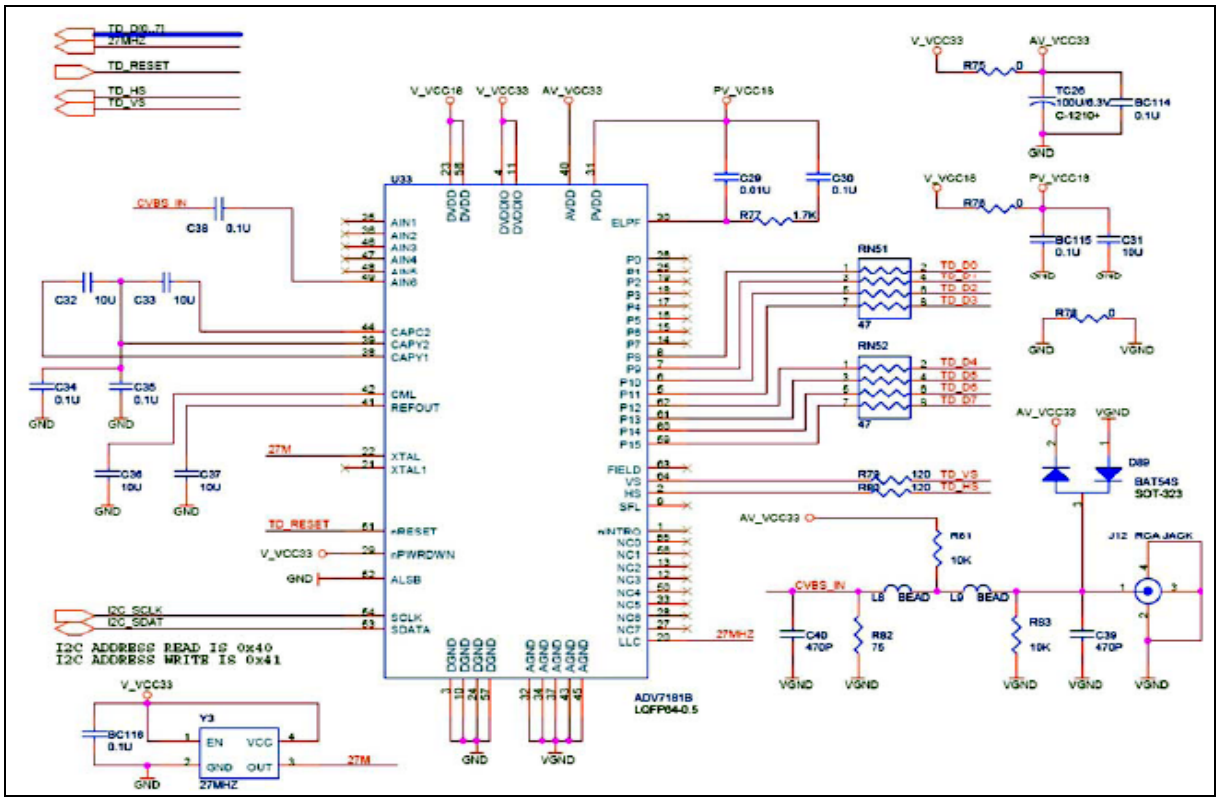
APÊNDICE A – Diagrama Esquemático do *Hardware* do Protocolo SCCB



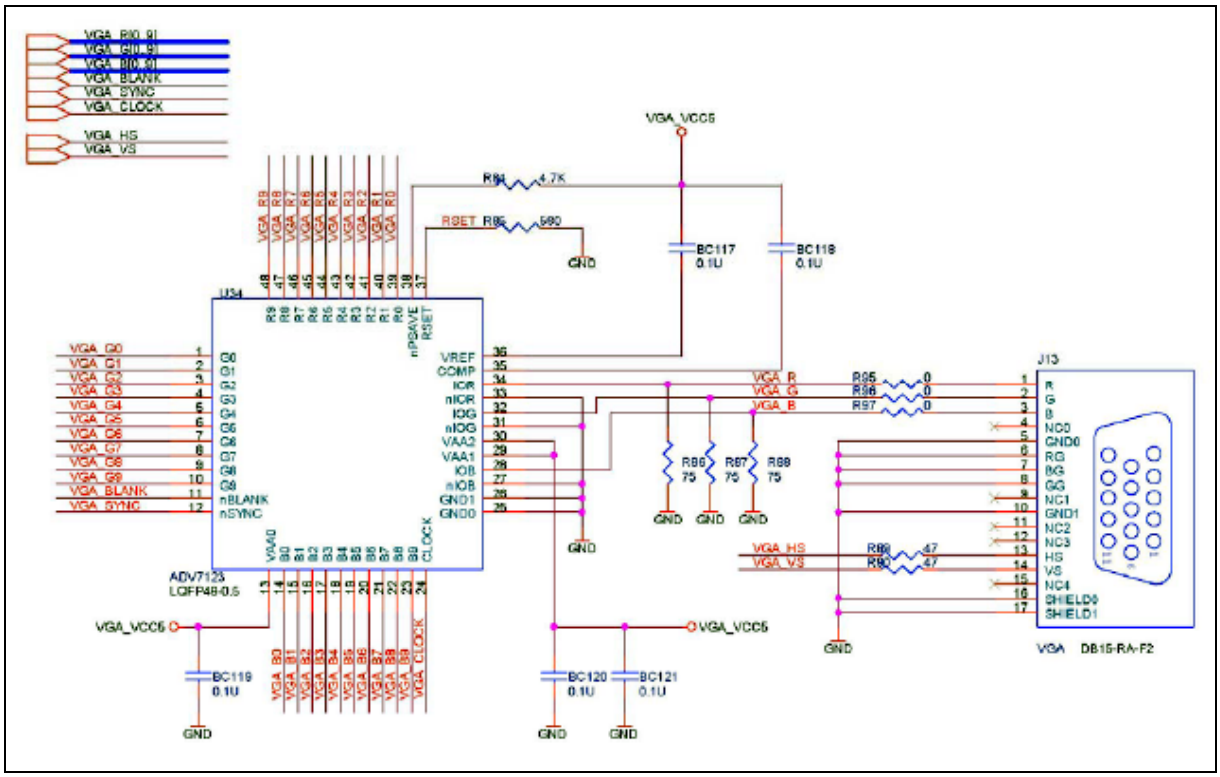
APÊNDICE B – Diagrama Esquemático do *Hardware* do Controlador RGB



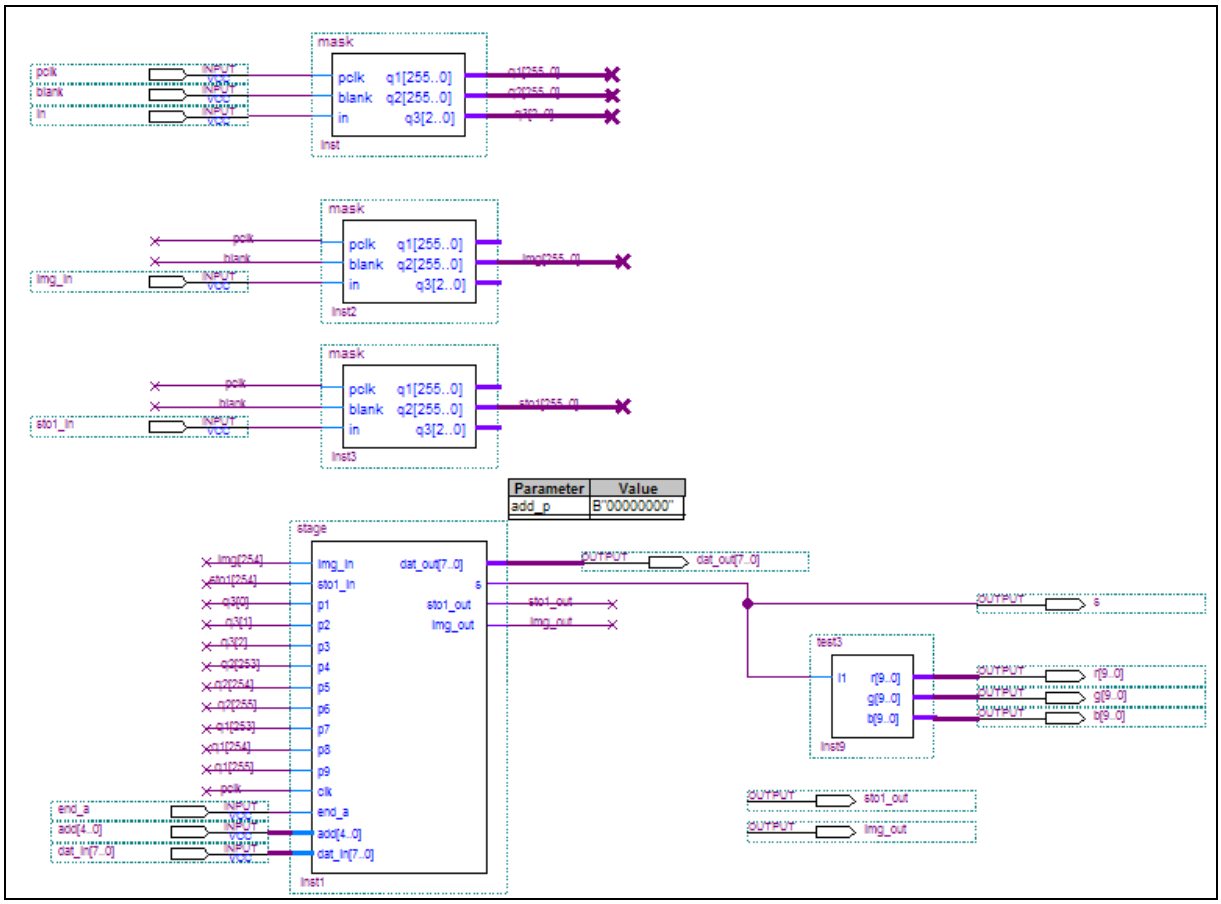
APÊNDICE C – Diagrama Esquemático do Circuito Detector de TV



APÊNDICE D – Diagrama Esquemático do Circuito VGA de Saída



APÊNDICE E – Diagrama Esquemático de um Estágio da Arquitetura *Pipeline*



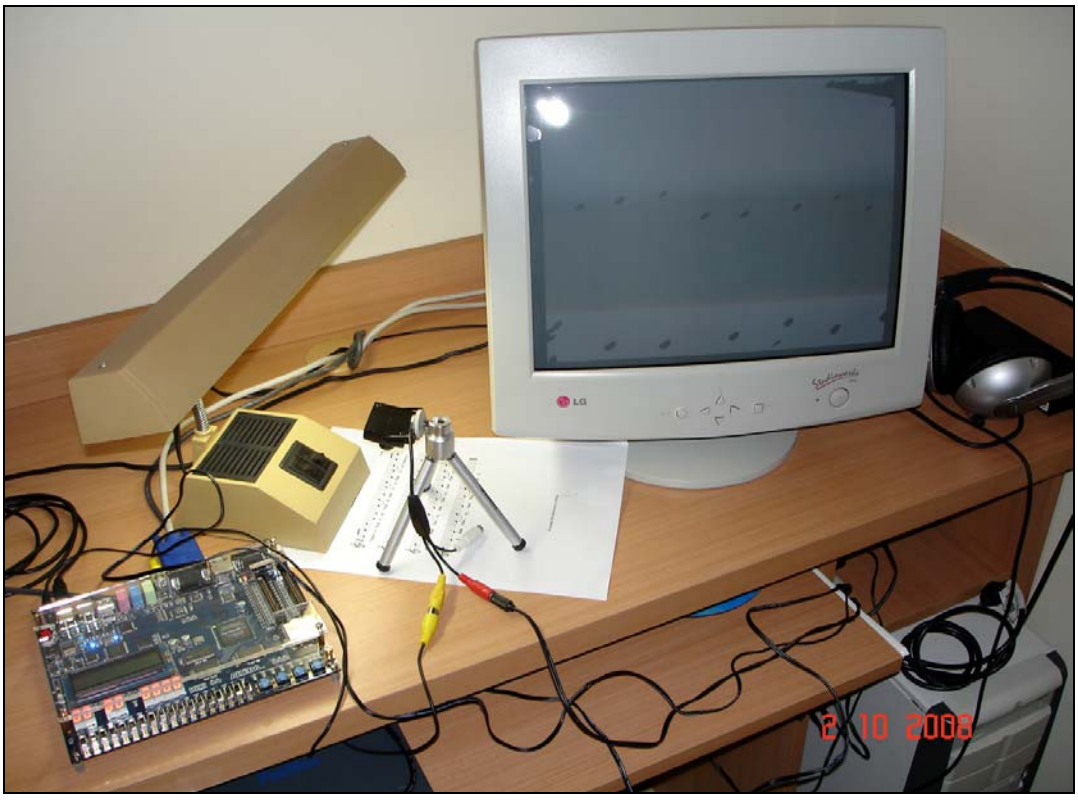
APÊNDICE F – Código em Verilog HDL de um Elemento de Processamento

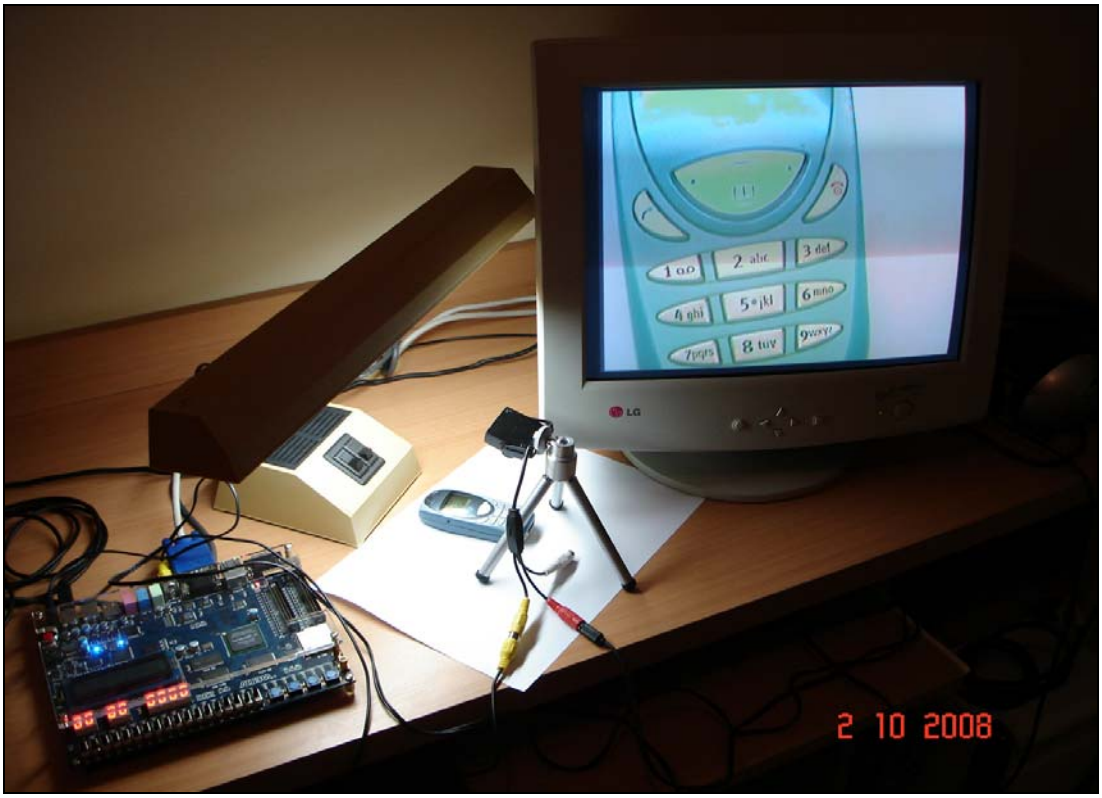
```
// Simple Computer Design
// By: Emerson C Pedrino, Ago/2007...

module stage (img_in,sto1_in,p1,p2,p3,p4,p5,p6,p7,p8,p9,clk,end_a,add,dat_in,dat_out,s,sto1_out,img_out);
  input p1,p2,p3,p4,p5,p6,p7,p8,p9,sto1_in,img_in;
  input clk;
  input end_a;
  input [4:0] add;
  input [7:0] dat_in;
  output [7:0] dat_out;
  output s;
  reg s;
  reg [7:0] dat_out, temp;
  output sto1_out;
  reg sto1_out;
  output img_out;
  reg img_out;
  parameter add_p=8'b00000000;

  always @(negedge clk)
   begin
     if (~end_a) begin
       if (add==add_p) begin
        temp=dat_in;
                        dat_out=temp;
                  end
                end
                else
                        begin
                                case (temp)
                                        8'b00000000: begin //Instr nada (0)...
                                                                s<=p5;
                                                                sto1_out<=sto1_in;
                                                                img_out<=img_in;
                                                        end

                                        8'b00000001: begin //Instr dil_q_3 (1)...
                                                                s<=0 | p1 | p2 | p3 | p4 | p5 | p6 |
p7 | p8 | p9;
                                                                sto1_out<=sto1_in;
                                                                img_out<=img_in;
                                                        end

                                        8'b00000010: begin //Instr ero_q_3 (2)...
                                                                s<=1 & p1 & p2 & p3 & p4 & p5
& p6 & p7 & p8 & p9;
                                                                sto1_out<=sto1_in;
                                                                img_out<=img_in;
                                                        end

                                    8'b00000011: begin //Instr dil_c_3 (3)...
                                                                s<= 0 | p2 | p4 | p5 | p6 | p8;
                                                                sto1_out<=sto1_in;
                                                                img_out<=img_in;
                                                        end
```

```
                8'b00000100: begin //Instr ero_c_3 (4)...
                                s<= 1 & p2 & p4 & p5 & p6 &
p8;
                                sto1_out<=sto1_in;
                                img_out<=img_in;
                             end

                8'b00000101: begin //Instr dil_h_3 (5)...
                                s<= 0 | p4 | p5 | p6;
                                sto1_out<=sto1_in;
                                img_out<=img_in;
                             end

                8'b00000110: begin //Instr ero_h_3 (6)...
                                s<= 1 & p4 & p5 & p6;
                                sto1_out<=sto1_in;
                                img_out<=img_in;
                             end

                8'b00000111: begin //Instr dil_v_3 (7)...
                                s<= 0 | p2 | p5 | p8;
                                sto1_out<=sto1_in;
                                img_out<=img_in;
                             end

                8'b00001000: begin //Instr ero_v_3 (8)...
                                s<= 1 & p2 & p5 & p8;
                                sto1_out<=sto1_in;
                                img_out<=img_in;
                             end

                8'b00001001: begin //Instr dil_dd_3 (9)...
                                s<= 0 | p3 | p5 | p7;
                                sto1_out<=sto1_in;
                                img_out<=img_in;
                             end

                8'b00001010: begin //Instr ero_dd_3 (10)...
                                s<= 1 & p3 & p5 & p7;
                                sto1_out<=sto1_in;
                                img_out<=img_in;
                             end

                8'b00001011: begin //Instr dil_de_3 (11)...
                                s<= 0 | p1 | p5 | p9;
                                sto1_out<=sto1_in;
                                img_out<=img_in;
                             end

                8'b00001100: begin //Instr ero_de_3 (12)...
                                s<= 1 & p1 & p5 & p9;
                                sto1_out<=sto1_in;
                                img_out<=img_in;
                             end

                8'b00001101: begin //Instr xor1 (13)...
                                s<= p5 ^ sto1_in;
                                sto1_out<=sto1_in;
                                img_out<=img_in;
```

```
                                        end

                                8'b00001110: begin //Cpl (14)...
                                                s<=~p5;
                                                sto1_out<=sto1_in;
                                                img_out<=img_in;
                                        end

                                8'b00001111: begin //Instr sto1 (15)...
                                                sto1_out<=p5;
                                                img_out<=img_in;
                                                s<=p5;
                                        end

                                8'b00010000: begin //Instr and1 (16)...
                                                sto1_out<=sto1_in;
                                                img_out<=img_in;
                                                s<=p5 & sto1_in;
                                        end

                                8'b00010001: begin //Instr or1 (17)...
                                                sto1_out<=sto1_in;
                                                img_out<=img_in;
                                                s<=p5 | sto1_in;
                                        end

                                8'b00010010: begin //Instr ldi (18)...
                                                sto1_out<=sto1_in;
                                                img_out<=img_in;
                                                s<=img_in;
                                        end

                                default: begin //Instr nada (0)...
                                                s<=p5;
                                                sto1_out<=sto1_in;
                                                img_out<=img_in;
                                        end
                        endcase
                end
    end


endmodule
```

APÊNDICE G – Algumas Fotos dos Experimentos




2 10 2008


6 10 2008

APÊNDICE H - Publicações Geradas Durante o Doutorado

PEDRINO, E. C.; RODA, V. O; OGASAWARA, O. FPGA based real time non linear image processor. In: 2008 4th Southern Conference on Programmable Logic, 2008, S. C. de Bariloche, Argentina. Proceedings 2008 4th Southern Conference on Programmable Logic. Madrid: School of Engineering - Universidad Autónoma de Madrid, p. 57-60, 2008.

PEDRINO, E. C.; RODA, V. O. Real-time morphological pipeline architecture using high-capacity programmable logical devices. Journal of Electronic Imaging (Springfield), v. 16, p. 023002-023009, 2007.

PEDRINO, E. C.; RODA, V. O.; JORGE, L.; DE FRANCISCO, C. A. Color Digital Video Acquisition and Processing System using Reconfigurable Logic. In: 3rd Southern Conference on Programmable Logic, 2007, Mar del Plata, Argentina. 3rd Southern Conference on Programmable Logic. Madrid: School of Engineering - Universidad Autónoma de Madrid, p. 29-32, 2007.

PEDRINO, E. C.; MARINHO, M.; RODA, V. O.; OGASAWARA, O. Arquitetura para Crescimento de Regiões de Imagens Binárias. In: 3rd Southern Conference on Programmable Logic, 2007, Mar del Plata, Argentina. 3rd Southern Conference on Programmable Logic. Madrid: School of Engineering - Universidad Autónoma de Madrid, p. 33-36, 2007.

PEDRINO, E. C.; RODA, V. O. Reconhecimento automático de padrões musicais utilizando operadores morfológicos e programação genética. In: WVC'2007 - III Workshop de Visão Computacional, p. 88-93, 2007, São José do Rio Preto.

PEDRINO, E. C.; RODA, V. O. Automatic pattern recognition of binary image objects using mathematical morphology. In: WVC'2007 - III Workshop de Visão Computacional, p. 180-185, 2007, São José do Rio Preto.

PEDRINO, E. C. Introdução à Visão de Máquina. Revista Mecatrônica Fácil, jan. 2007.

PEDRINO, E. C.; RODA, V. O. Pipeline Architecture for Real Time Morphological Color Image Processing. In: II SOUTHERN CONFERENCE ON PROGRAMMABLE LOGIC, 2006, Mar Del Plata, Argentina. FPGA Based Systems, Madrid, ES: Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2006.

PEDRINO, E. C.; RODA, V. O. Construção Automática de Operadores Morfológicos utilizando Programação Genética. In: WVC'2006 – II WORKSHOP DE VISÃO COMPUTACIONAL, 2006, Escola de Engenharia de São Carlos, Universidade de São Paulo.

PEDRINO, E. C.; RODA, V. O. Extração de Descritores em Imagens Binárias utilizando Técnicas de Morfologia Matemática Clássica. Multiciência, v. 7, p. 65-80, 2006.

DE SANTI, R. M.; PEDRINO, E. C.; MILANI, D. M. Inspeção em Placas de Circuito Impresso utilizando Métodos de Morfologia Matemática. Multiciência, v. 7, p. 173-187, 2006.

PEDRINO, E. C. Morfologia Matemática Binária Aplicada ao Processamento de Imagens Digitais. In: II Jornada de Matemática, 2005, UNICEP, São Carlos, SP.

PEDRINO, E. C. Arquitetura para Extração de *Features* em Imagens Monocromáticas. Multiciência, v. 6, p. 79-92, 2005.

PEDRINO, E. C.; KLEIN, C.; JORDAO, J. A. Sinal de Vídeo Composto Padrão. Multiciência, v. 6, p. 93-98, 2005.

PEDRINO, E. C.; RODA, V. O. Arquitetura *Pipeline* para Processamento Morfológico de Imagens Binárias em Tempo Real. In: I Simpósio Latino Americano em Aplicações de Lógica Programável e Processadores Digitais de Sinais em Processamento de Vídeo, Visão Computacional e Robótica, 2004, São Carlos, SP, Brasil.

PEDRINO, E. C.; RODA, V. O. Sistema de Aquisição, Armazenamento e Exibição de Imagens Monocromáticas utilizando CPLDs. In: I Simpósio Latino Americano em Aplicações de Lógica Programável e Processadores Digitais de Sinais em Processamento de Video, Visão Computacional e Robótica, 2004, São Carlos, SP.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo