UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA DA COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Arquitetura VLSI para Processamentos Morfológicos em Imagens

por: Ilka Marinho Barros

orientador: Prof. Dr. Roberto de Alencar Lotufo

co-orientador Prof. Dr. Neucimar Jerônimo Leite

Este exemplar corresponde à redação final da tese defendida por Ilka Marinho Barros e aprovada pela Comissão Julgadora em 30 set 1994.

Dissertação submetida à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de Mestre em Engenharia Elétrica.

setembro 1994

Resumo

A disseminação e utilização do Processamento de Imagens em inúmeras áreas da ciência e suas aplicações tem levado à intensificação das pesquisas neste ramo. A Morfologia Matemática é um conjunto de ferramentas para o processamento de imagens que tem sido bastante enfocado. Esta dissertação descreve uma arquitetura especialmente desenvolvida para implementação de operações morfológicas sobre imagens binárias e em nível de cinza, com aproveitamento integral do hardware. As operações implementadas de dilatação e erosão são básicas para a composição de diversas operações morfológicas mais complexas. Justifica-se sua implementação em um hardware dedicado devido à quantidade de dados a serem processados e devido à velocidade de operação requisitada. Outras operações morfológicas foram também implementadas através deste mesmo hardware. Usando a linguagem de descrição de hardware VHDL, foi realizada a modelagem, simulação e validação desta arquitetura. Esta arquitetura agrega características importantes de alguns processadores existentes para o mesmo propósito como a utilização integral do hardware em operações numéricas e binárias, alta modularidade, regularidade e baixa complexidade, características importantes para sua implementação VLSI. Temas relacionados a este trabalho como metodologias de projeto, linguagens de descrição de hardware e processadores de imagem são também discutidos.

Abstract

The popularization and the use of Image Processing in several fields of Science and its applications, have increased the importance of the researches in this area. Mathematical Morphology is a set of tools used in image processing which has been widely used. This work describes an architecture specially developed for the execution of morphological operations on binary and gray-level images, using the same hardware structure. Complex morphological operations are composed by the dilation and erosion operations realized by this hardware. Because of the amount of data envolved and of the processing speed required it is used a dedicated hardware to implement these operations. Other morphological operations are also realized by this structure. The hardware description language VHDL was used in the modeling and simulation of the architecture. This architecture is well suited for VLSI implementation due to its high modularity, regularity and low complexity. Subjects related to this work such as methodologies, hardware description languages and image processors are also presented.

Agradecimentos

Agradeço acima de tudo à Deus, que fez com que este trabalho acontecesse, utilizando-o para muito mais do que meu crescimento profissional e pessoal.

Agradeço aos meus pais Rubião e Glorinha por sonharem junto comigo, acreditarem na minha capacidade, investirem amor, tempo e as suas vidas na minha vida.

Agradeço ao Rui pelo amor e pela espera.

Agradeço aos meus parentes e amigos que estiveram torcendo por mais esta vitória.

Agradeço ao meu orientador Prof. Dr. Roberto A. Lotufo pelo seu apoio ao longo deste trabalho, e ao meu co-orientador Prof. Dr. Neucimar J. Leite pela participação, interesse e ajuda.

Agradeço ao Josemir C. Alexandrino pela ajuda e sugestões na arquitetura e no VHDL. Ao Renato Luppi, agradeço a disposição e atenção em fornecer as imagens e dados do Khoros. Ao Eng. Carlos Krüger do Cpqd-Telebrás, pelo auxílio na pesquisa bibliográfica e pelas discussões sobre o VHDL. Ao Leandro Dibb agradeço a ajuda operacional.

Ao pessoal do DSIF, Prof. Vitor Baranauskas, Marcelo F., Marcelo J., Nivaldo, Carla, Eliane, Margareth, Juan, Wilfredo, Chang, Augusto agradeço a convivência, a atenção, a amizade, a ajuda, e o compartilhar das máquinas e do dia-a-dia.

Agradeço aos laboratórios LCA, LCAEE e DSIF pelas instalações e máquinas.

Agradeço à CAPES e à FAPESP pelo apoio financeiro, sem o qual este trabalho não teria sido realizado.

Dedico este trabalho a todos aqueles que me amam e me incentivaram para a realização deste sonho.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O Processamento Digital de Imagens requer alta velocidade de operação devido à grande quantidade de dados envolvidos. Desta forma, uma implementação paralela do hardware que executa estas operações torna-se um fator preponderante no que se refere ao tempo de processamento. Graças à tecnologia de circuitos atual é viável economicamente a construção de circuitos VLSI dedicados à implementação de arquiteturas específicas.

O processamento de imagem de baixo nível é constituído de operações que apresentam características importantes que possibilitam uma abordagem paralela na sua implementação. As operações de dilatação e erosão da Morfologia Matemática fazem parte desta classe de algoritmos. Estas operações são básicas, e muitas operações morfológicas podem ser definidas em função delas. Sua implementação em hardware dedicado possibilita explorar ao máximo as possibilidades de paralelização inerentes a cada operação alcançando uma maior velocidade de processamento.

Visando a implementação através de circuito integrado, foi desenvolvida uma arquitetura enfocando a implementação das operações morfológicas de dilatação e erosão por um elemento estruturante planar, nos modos binário e numérico, com aproveitamento máximo do hardware em ambos os modos. É uma arquitetura sistólica, homogênea, constituída por blocos interligados, através dos quais os dados fluem de maneira síncrona e regular, fornecendo ao final da estrutura o resultado desejado.

Através da modelagem desta arquitetura a partir de uma linguagem de descrição

de hardware - VHDL - foi possível a simulação e a validação do seu funcionamento no ambiente integrado de ferramentas VHDL da Mentor Graphics. Esta linguagem é capaz de descrever conceitos particulares de hardware, permitindo sua descrição em diversos níveis de abstração, possibilitando o uso da metodologia mais adequada ao projeto.

Este trabalho está organizado da seguinte maneira. No capítulo 2 são abordados aspectos de metodologias de projeto. Linguagens de descrição de hardware são apresentadas como a maneira atual de descrição dos circuitos para sua modelagem, simulação e validação. É apresentada a linguagem VHDL e suas características básicas. No capítulo 3 são apresentados alguns processadores de vizinhança de imagens e seus aspectos mais relevantes. No capítulo 4 é proposta uma arquitetura que implementa as operações de Morfologia Matemática sobre imagens binárias e em nível de cinza. O funcionamento da arquitetura é explicado detalhadamente e são discutidas suas vantagens com relação a outras arquiteturas usadas para o mesmo propósito. O capítulo 5 apresenta a utilização da linguagem VHDL na modelagem e simulação da arquitetura. Trata do seu particionamento e composição e discute os resultados obtidos. O capítulo 6 conclui o trabalho destacando seus aspectos principais.

Capítulo 2

Metodologia de Projeto usando VHDL: uma introdução

2.1 Introdução

Este capítulo descreve brevemente metodologias de projeto e aborda seu uso através da linguagem VHDL. A estrutura de funcionamento desta linguagem e seu uso, são também tratados.

2.2 Níveis de abstração de projeto

Um sistema digital pode ser representado através de níveis de projeto, representando diferentes graus de abstração com relação ao sistema físico. Estes níveis podem ser classificados em domínios de descrição comportamental, estrutural ou físico, representados através dos três eixos ortogonais do diagrama de Gajski, mostrado na Figura 2.1. No eixo comportamental se encontram os níveis de projeto que descrevem o comportamento de um sistema. No eixo estrutural cada nível é representado pela interconexão de primitivas que compoem um sistema. E finalmente, sobre o eixo físico, estão os níveis de projeto relacionados à implementação física do projeto, estando intimamente relacionados com a tecnologia

empregada [1],[2],[3].

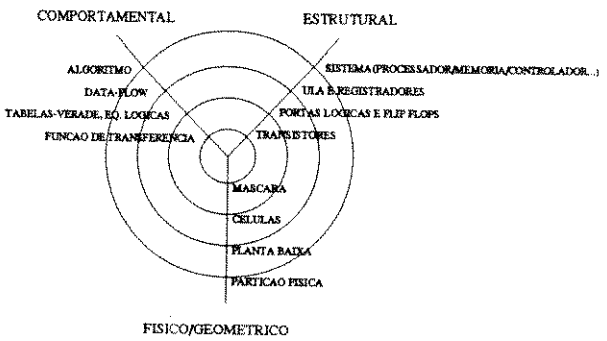

Figura 2.1: diagrama de Gajski

Os círculos concêntricos representam subníveis correspondentes de projeto sobre os três eixos. Quanto mais internos, maior o nível de detalhamento da descrição, ou seja, menor o nível de abstração e, conseqüentemente mais próximos nos encontramos da representação física [2].

2.3 Metodologia de projeto

Metodologia de projeto é a seqüência de transformações que se processam desde a especificação do projeto até a sua descrição final. Estas transformações podem ser de *validação* ou *síntese*. Nas transformações de *síntese*, a partir da descrição em um determinado subnível, é gerada uma descrição em outro subnível pertencente ao mesmo ou à outro eixo (nível) do diagrama de Gajski. Na *validação*, são avaliados e conferidos, a partir da descrição em um determinado subnível, um conjunto de características relativas a este subnível [1]. Segundo Wagner [1], metodologia de projeto pode também ser definida como a seqüência de transições sobre o diagrama de Gajski.

Assim sendo, a metodologia acaba por estabelecer a forma de gerenciamento do projeto, as formas de avaliação e teste (tipos de ferramentas utilizadas) e até sua documentação, afetando diretamente o custo, desempenho, qualidade e tempo de projeto. Todos estes fatores interagem, se completam e conduzem de maneira sistemática ao desenvolvimento do projeto [3]. São estes fatores descritos que distinguem uma metodologia de outra. A metodologia adotada depende, naturalmente, do tipo de projeto sendo desenvolvido [2].

2.4 Finalidade de utilização da metodologia de projeto

A finalidade de utilização de uma metodologia é ter um projeto estruturado [2]. Através de uma maneira sistemática de concepção e validação, sabe-se o que esperar e quanto confiar nos resultados obtidos. Devido à complexidade dos projetos atuais, curto tempo de desenvolvimento para competitividade no mercado, número de projetistas e variedade de ferramentas envolvidas num projeto, torna-se indispensável o uso de uma metodologia de projeto, possibilitando, em princípio um trabalho rápido, sistemático e eficiente e viabilizando sua implementação comercial.

2.5 Descrição de metodologias de projeto

Um projeto pode ser desenvolvido através de metodologias bottom-up (ascendente), top-down (descendente) ou através da combinação de ambas.

2.5.1 Metodologia "Bottom-Up"

Na metodologia "bottom-up", parte-se do desenvolvimento de componentes básicos que vão sendo agrupados hierarquicamente de forma a compor o sistema. Dados relativos à implementação física do projeto, como a tecnologia a ser utilizada, são considerados desde o início. Esta metodologia foi a primeira a ser utilizada pelos projetistas. Gastava-se muito tempo para desenvolver circuitos de tamanho pequeno à mão, e mais tarde com o surgimento de ferramentas que auxiliavam no projeto, possibilitou-se a execução de projetos maiores em tempos menores de desenvolvimento. A metodologia "bottom-up" está arraigada à mentalidade dos projetistas mais antigos, acostumados à filosofia e às ferramentas de projeto disponíveis para ela. A metodologia "bottom-up" não consegue suportar um tipo de projeto onde o número de portas é elevado, e conseqüentemente onde a quantidade de fatores a serem considerados é alta. Não oferece um gerenciamento adequado, sendo considerável o tempo de desenvolvimento do projeto e difícil a garantia de seu funcionamento correto.

2.5.2 Metodologia "Top-Down"

Na metodologia "top-down", parte-se da descrição geral do sistema em direção à descrição do sistema físico, ou seja, dos níveis mais altos de abstração em direção aos níveis mais inferiores. É feita a descrição da idéia do projeto através de um alto nível de abstração, onde não são definidos detalhes referentes à forma de implementação, tecnologia utilizada, etc. À medida que se valida a descrição, maiores detalhamentos são feitos nos níveis inferiores de abstração. Traduzindo isto ao Diagrama de Gajski (figura 2.1), a metodologia "top-down" é a passagem do subnível de abstração mais elevado ao menos elevado dentro de um mesmo nível, ou a passagem entre subníveis correspondentes pertencentes a níveis de abstração diferentes. Percorrem-se os eixos na ordem do comportamental para o estrutural e em seguida ao físico [3].

Um dos benefícios alcançados através desta metodologia é a facilidade de implementação e desenvolvimento de projetos complexos. Esta metodologia dá uma visão geral ao projetista, que passa a ter uma melhor percepção do que é esperado em cada parte do projeto e dele como um todo. Em cada etapa é necessário apenas observar os detalhes referentes a ela e ter domínio sobre estes fatores, o que garante um projeto otimizado passo a passo, sem perder a noção do conjunto. Sendo assim, é possível dominar o espectro de fatores e a quantidade de componentes envolvidos num projeto grande e complexo.

Como a maioria da descrição do projeto é feita independentemente da tecnologia em que este será implementado, partes deste podem ser reaproveitadas em outros projetos. Também o projeto como um todo pode ser implementado de diferentes formas visando, por exemplo, uma otimização [6].

Como um grande investimento é feito na descrição em alto nível, e primitivas de teste podem estar presentes em todos os níveis de descrição, torna-se mais fácil a detecção de erros, reduzindo o tempo e o custo do projeto e aumentando o seu grau de confiabilidade desde o início [4]. Através da metodologia "top-down" é possível a realização da síntese de uma descrição feita em HDL, o que aumenta a produtividade do projeto [5], [6].

Adotar uma metodologia "top-down" significa uma mudança cultural no meio de desenvolvimento de projetos, o qual está ainda ligado à metodologia "bottom-up". Isto implica no desenvolvimento de ferramentas de software para auxílio ao projeto que viabili-

zem um projeto "top-down", e atualmente, no aprendizado de uma linguagem de descrição de hardware que interaja com estas ferramentas.

O que ocorre normalmente é uma combinação das duas metodologias descritas. Num projeto "top-down", estando em um determinado nível de abstração, muitas vezes é necessário retornar ao nível imediatamente superior e modificar conceitos e descrições para que no nível presente se tenha uma melhor forma de concepção. Esta dinâmica na interação entre as etapas do projeto leva a um auto-aprimoramento do produto final.

2.6 Formas de descrição de hardware

Para se descrever um sistema digital, existem à disposição diversos recursos, cada qual mais adaptado aos diferentes níveis de abstração na descrição de um projeto. Podemos citar dentre estes, diagramas de bloco, diagramas de tempo, fluxogramas, autômatos, equações algébricas e linguagens. Para permitir a compreensão e a manipulação de uma descrição por um computador, possibilitando-se então um projeto automatizado, são usadas mais freqüentemente descrições por meio de linguagens [1].

A seguir são apresentados alguns aspectos principais das linguagens de descrição de hardware.

2.7 Linguagens de descrição de hardware - HDL's

Linguagens de Descrição de Hardware - HDL's (do inglês "Hardware Description Languages) - são linguagens formais, feitas especificamente para a descrição de um hardware, seja este um sistema complexo ou um simples circuito [7]. As HDL's fornecem suporte e possibilitam um projeto formal, sistemático e documentado (projeto baseado em uma metodologia). São capazes de descrever aspectos comportamentais e estruturais e geométricos, através de diferentes formas (níveis de abstração). São bastante apropriadas ao projeto "top-down", pois através delas é possível que uma descrição, inicialmente pouco detalhada do projeto (global), possa ser transformada, à medida da sua validação, em uma descrição com alto nível de detalhamento funcional e estrutural.

As HDL's começaram a ter um papel importante no início da década de 80, quando foi introduzida pelos fabricantes de ferramentas de software a possibilidade de simulação de uma arquitetura formada por blocos descritos em distintos níveis de abstração. Através da descrição do sistema em uma HDL tornou-se possível a simulação do projeto desde a especificação a nível de arquitetura até o projeto a nível de portas. No final da década de 80 foi implementada a síntese lógica automática que converte uma descrição feita em HDL em uma descrição à nível de portas lógicas.

O uso de uma HDL significa que através de um conjunto semântico é possível descrever aspectos estruturais, comportamentais e físicos de um sistema complexo. Atualmente, as ferramentas para automação de projetos eletrônicos incluem funções de documentação, síntese, verificação, simulação, descrições de padrões de teste, simulação de falhas, análise de caminhos críticos, análise de temporização, síntese lógica, síntese de layout, otimização, posicionamento e roteamento, baseadas em descrições HDL [8]. A capacidade, o desempenho e as possibilidades disponíveis no uso de uma HDL estão condicionados às ferramentas de projeto do ambiente disponíveis.

Um metodologia de projeto que usa HDL atende necessidades atuais do projeto de circuitos integrados como especificação inicial rigorosa do projeto, simulação e testes de cada etapa e possibilidade de síntese do circuito lógico. Sendo assim, satisfaz três necessidades básicas: uma especificação de projeto simulável, síntese automática em portas lógicas e uma análise de desempenho [8].

2.7.1 Dificuldades na incorporação de HDL's

Há barreiras no uso de linguagens para descrição de hardware pois a cultura atual ainda está associada à representação do hardware através de diagramas de estados, diagramas em bloco, tabelas verdade, esquemáticos e não através de uma linguagem, que apesar de todo seu potencial e recursos, ainda apresenta deficiências técnicas e práticas. Há ainda o custo relativo ao tempo de aprendizado em detrimento do andamento inicial do projeto.

Além do projetista escrever um código funcionalmente correto, deve também escrever um código sintetizável. Atualmente, a síntese automática ainda não é um processo

totalmente dominado pelas ferramentas de automação, não sendo assim, todo código escrito sintetizável.

2.7.2 Sugestões para facilitar o uso de HDL's

Para facilitar o processo de transição da implementação de baixo nível de abstração (esquemáticos, diagramas em bloco, etc.) para a de alto nível de abstração (HDL), as ferramentas de automação devem apresentar também a possibilidade de entrada através dos recursos conhecidos e a partir deles ser feita automaticamente a transformação em alguma linguagem de descrição de hardware.

A verificação e a constatação de uma descrição feita em HDL ser sintetizável ou não, desde o mais alto nível de especificação, é um fator importante para o aceleramento do projeto "top-down" [8].

2.7.3 Vantagens desta incorporação

Com a crescente complexidade dos projetos e devido ao elevado número de pessoas envolvidas, é muito importante obter-se uma definição clara e precisa das especificações, além de uma documentação atualizada, o que pode ser obtido com o uso de uma HDL.

Outro benefício é o aumento da produtividade em que um maior número de portas por intervalo de tempo pode ser projetado. O custo de projeto também pode ser reduzido no que se refere à fase de testes, pois estes podem ser executados num ambiente específico à medida em que se descreve cada nível do sistema [9].

Pela descrição de um circuito através de uma HDL é possível sua síntese automática. Através do resultado obtido pode-se avaliar a relação custo-benefício entre área, velocidade e custo. A partir disto, dentro de um processo de otimização, pode-se alterar a descrição inicial em HDL a fim de se obter melhores resultados, antes de se chegar ao projeto final [8].

2.7.4 Necessidade de um padrão para descrição de hardware

Quando se usa um padrão para descrever um determinado projeto de hardware, faz-se uso de certas vantagens. Ter o projeto descrito de um modo que possa ser entendido universalmente por outros projetistas é uma delas. Com isso, possibilita-se a reutilização do projeto ou de partes deste. É interessante para os fabricantes de ferramentas de automação projetarem seus produtos baseados em um padrão. Assim elimina-se, entre outros, a necessidade de aprendizado de linguagens específicas aos fabricantes de ferramentas.

Com o uso de um padrão, passa a existir uma variedade de ferramentas disponíveis para cada fase do projeto, cada uma atendendo a uma necessidade específica, sendo a descrição do projeto portátil entre todas elas. Formam-se plataformas de suporte ao projeto de hardware que atendem desde o nível mais alto de concepção até o nível de implementação física.

A seguir será discutida a linguagem de descrição de hardware VHDL, adotada como padrão pelo IEEE.

2.8 A linguagem de descrição VHDL - histórico

A linguagem VHDL surgiu no início da década de 80, através do programa VHSIC (Very High Speed Integrated Circuit), do Departamento de Defesa dos Estados Unidos. A intenção inicial da linguagem VHDL - VHSIC HDL - era servir como meio de comunicação e documentação, e assim padronizar a troca dos dados de projetos.

Trabalhando em conjunto com o Departamento de Defesa, diversas empresas da área de computação auxiliaram na especificação desta linguagem, resultando de certa forma num consenso sobre como deve ser uma linguagem de descrição de hardware. Nesta fase de definição foram traçadas as limitações e possibilidades oferecidas pela linguagem.

O desenvolvimento desta linguagem acabou atendendo a uma outra necessidade, além do simples padrão de documentação: ser uma linguagem executável e ser deste modo usada para modelagem, simulação e síntese [1], [3], [10].

Em 1987, após algumas mudanças e revisões, VHDL foi adotada como padrão pelo IEEE (IEEE 1076). Muitas ferramentas foram modificadas e desenvolvidas para aceitar como entrada ou produzir como resultado descrições VHDL. Terminada a fase de desenvolvimento, quando a linguagem foi refinada e melhorada, o VHDL se encontra atualmente na sua fase de disseminação, que tem sido rápida e significativa nos meios industriais, universitários, nas instituições de pesquisa e nos meios comerciais [11].

2.8.1 A linguagem VHDL

A linguagem VHDL suporta a especificação, a descrição e a simulação de estruturas de hardware em diversos níveis de abstração simultâneos (multinível), possibilitando ainda a síntese [7]. Seu poder de descrição e simulação abrange os níveis dos eixos comportamental e estrutural do Diagrama de Gajski, fornecendo descrição executável, portátil e não ambígua.

É sem dúvida altamente direcionada ao projeto de hardware pois suporta conceitos particulares como, por exemplo, concorrência, tempo, componentes, sinais, interfaces, portes. VHDL provê em cada fase do projeto notação adequada para a representação do sistema, não sendo direcionada às características de uma categoria específica de projeto [12],[10].

O projetista trabalhando com a linguagem VHDL tem à sua disposição diversas ferramentas e plataformas de diferentes fabricantes que permitem o desenvolvimento consistente do seu projeto [13].

2.8.2 A descrição em VHDL

A descrição VHDL é feita através de *entidades* e *arquiteturas*, por meio das quais se descreve um bloco de hardware que pode representar desde um microprocessador até uma simples porta lógica.

A *entidade* define a interface do circuito que se descreve com o meio externo a ele através de sinais (PORTES). São ainda definidos parâmetros genéricos e são declarados tipos, sub-tipos e atributos particulares de cada *entidade*.

A cada *entidade* podem estar ligadas uma ou mais arquiteturas (figura 2.2). Na *arquitetura* são descritas as transformações entre entradas e saídas da *entidade*, descrevendo seu funcionamento. Através das diversas arquiteturas pode-se descrever a *entidade* em diferentes níveis de abstração ou diferentes alternativas de implementação da mesma [14].

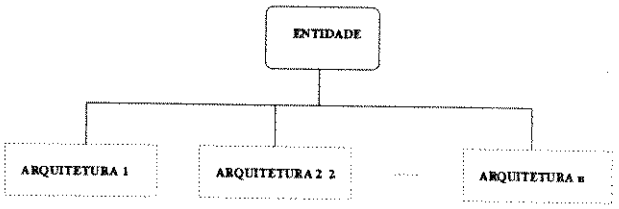

Figura 2.2: estrutura de descrição VHDL

Em cada nível de abstração, o projeto é visto com um enfoque particular e característico, visualizando-se apenas certas propriedades do sistema. Em VHDL, os níveis possíveis de abstração são: comportamental, data-flow (RTL) e lógico ou estrutural [14].

- descrição comportamental: é o mais alto nível de abstração para a descrição de um projeto. É uma descrição algorítmica do comportamento do hardware, como nas linguagens de programação de alto nível, onde a funcionalidade do sistema é descrita através de processos paralelos contendo comandos seqüenciais que transformam entradas em saídas. O modo como esta funcionalidade é implementada não é abordado, sendo independente da arquitetura e da tecnologia [15],[16],[1],[14].
- descrição data-flow ou RTL: é o nivel de descrição abaixo da comportamental. Aqui a funcionalidade é descrita de forma não "procedural" através de atribuições concorrentes entre entradas e saídas do sistema pela transferência de dados de forma síncrona ou assíncrona (aparecendo um esquema de clock). Neste nível começam a se definir as noções da arquitetura, mas como na descrição comportamental, a descrição RTL continua a ser independente da tecnologia utilizada [15],[14],[1],[17].
- descrição lógica ou estrutural: é a descrição imediatamente acima da representação física de um projeto, onde a arquitetura é descrita através de componentes mais primitivos (portas lógicas, por exemplo) e da interconexão entre eles [18]. A descrição neste nível depende da tecnologia a ser utilizada, fazendo uso de informações como por exemplo, atraso das portas [15],[14].

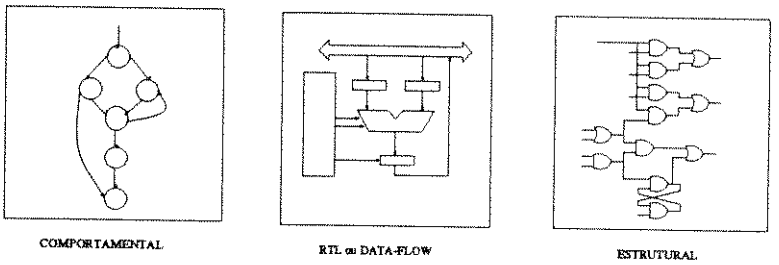

Figura 2.3: estrutura de descrição VHDL

Na modelagem de um sistema VHDL cada *entidade* pode estar sendo descrita por uma *arquitetura* em um nível diferente de abstração, podendo estas três formas de descrição coexistirem na mesma descrição de projeto.

2.8.3 Simulação em VHDL

A possibilidade de se avaliar previamente a idéia, o funcionamento e o desempenho de algo que se planeja, sem ter-se isto fisicamente, é algo extremamente significativo para qualquer tipo de projeto [7]. Tendo-se uma representação ou modelo, isto é possível através da simulação. A modelagem através da linguagem VHDL possibilita simulação desde sistemas até elementos mais básicos de circuito, por meio das ferramentas disponíveis. Deste modo é possível a validação de cada etapa, a reavaliação das decisões e a análise do funcionamento de várias opções de implementação, tornando viável o projeto de circuitos complexos.

Cada vez mais, as ferramentas de projeto baseadas em descrições VHDL facilitam a depuração dos modelos, formando uma interface amigável e interativa com o usuário. Para isto tem-se disponíveis janelas gráficas que exibem o comportamento dos sinais ao longo do tempo, semelhantes a pontas de prova usadas em testes em bancadas. Na depuração do modelo descrito na linguagem, há possibilidade da execução passo à passo, introdução de "breakpoints" e outros recursos usuais na depuração das linguagens convencionais de alto nível (C, Pascal, etc.). As ferramentas de simulação procuram usar todos os recursos oferecidos pela linguagem, penetrando mais profundamente na intenção da descrição e explorando todas as possibilidades de simulação do mesmo. VHDL não apresenta em si dificuldades à implementação da simulação do modelo que se deseja descrever, sendo uma

linguagem fortemente tipada, apropriada à modelagem do hardware. Portanto, a profundidade e abrangência da simulação encontram-se ligadas não à potencialidade da linguagem, mas sim às ferramentas disponíveis.

2.8.4 Síntese em VHDL

Conforme definido anteriormente, uma metodologia consiste em transformações de validação e síntese. VHDL é uma linguagem que possibilita a síntese automática, apesar de não ter sido desenvolvida para este propósito [17]. Entretanto, grande parte das transformações de síntese acabam ocorrendo fora do ambiente de descrição e simulação do VHDL, devido à falta de suporte das ferramentas existentes para fazer isto automaticamente. Entretanto algumas ferramentas já implementam, através da descrição VHDL, a síntese lógica e a RTL.

- síntese RTL: a partir de uma descrição a nível de transferência de registros, onde tem-se definido um clock e um fluxo de dados, é gerada uma descrição em nível lógico, mapeado em uma tecnológica específica. Devido à complexidade dos projetos atuais, a representação a nível de portas é muito grande, o que faz da síntese automática algo significativo em termos de produtividade e custo de projeto.
- síntese lógica: a partir de uma descrição lógica é sintetizada uma descrição física a nível de transistores.

Ainda não existem ferramentas comercialmente disponíveis que realizem a síntese de uma descrição em nível RTL, a partir de uma descrição em nível comportamental [17].

A síntese automática reduz etapas no trabalho do projetista, devendo produzir uma otimização dos fatores pertencentes ao nível de descrição sintetizado. A funcionalidade de um circuito pode ser definida de vários modos e com apenas uma pequena variação na forma de descrição pode-se ter uma mudança significativa no circuito sintetizado. Portanto, a síntese automática ainda possibilita a avaliação e a escolha entre diversas opções de implementação.

Quando se deseja realizar a síntese, além de se ter uma descrição sintetizável é importante também que ela seja eficiente, simples, de fácil de implementação, isto é,

quando sintetizada produza um "bom" hardware [4]. No entanto, nem todo o código VHDL é passível de síntese por grande parte das ferramentas disponíveis, sintetizando cada uma delas um conjunto de instruções VHDL diferente.

2.8.5 Dificuldades de utilização

Além das dificuldades inerentes às linguagens de hardware, VDHL apresenta ainda dificuldades particulares.

A interface do usuário com VHDL é um obstáculo para a sua fácil aceitação. O aprendizado de uma linguagem e a concepção de um hardware através dela impõe uma ruptura na filosofia até então considerada. A descrição do projeto através de uma linguagem textual e não através de representação esquemática é uma das desvantagens da maioria das HDL's, presente na maioria das ferramentas que usam VHDL. Outra característica é a dificuldade de descrição de um código VHDL sintetizável devido a desta linguagem não ter sido desenvolvida com o propósito de síntese automática e também devido à falta de suporte das ferramentas disponíveis a todo conjunto VHDL.

É necessário uma quantidade razoável de código para se descrever um projeto em VHDL. O paralelismo da linguagem, caracterizando os eventos que ocorrem em hardware, torna difícil a depuração de seu código. Algumas ferramentas utilizam subconjuntos diferentes da linguagem VHDL, dificultando a compatibilidade entre elas [12].

Uma característica importante de VHDL é seu bom suporte à metodologia "top-down", permitindo lidar mais facilmente com a complexidade dos grandes sistemas.

A seguir será descrita brevemente a utilização da linguagem VHDL no projeto da arquitetura tratada no Capítulo 4.

2.9 Uso de VHDL no projeto da arquitetura de processamento de imagens

O uso da linguagem VHDL para a modelagem, simulação e validação da Arquitetura de Processamento de Imagens descrita no capítulo 4, se deu no ambiente integrado de projeto VHDL da Mentor Graphics, inicialmente na versão 8.1, e posteriormente na versão 8.2. Instalado numa estação de trabalho do tipo SPARC station 2 da SUN, utilizou-se as seguintes ferramentas deste ambiente:

- Design Architect: ferramenta de modelagem, compilação (através do compilador System-1076) e depuração dos blocos descritos em VHDL.
- Quicksim II: ferramenta utilizada para a simulação e validação dos blocos descritos em VHDL.

A partir do projeto da arquitetura a ser implementada foi feita sua partição funcional em blocos e a descrição VHDL comportamental destes, sendo simulados e validados separadamente. Procurou-se minimizar o número de portes necessários à sua comunicação externa (entradas de dados, controle e saídas), na definição da sua interface. Na modelagem de cada bloco componente, foram utilizados tanto sinais de tipos padronizados VHDL (baseados no pacote IEEE 1076 [19]), sinais de tipos pré-definidos pelas ferramentas Mentor (QSIM-State [19]), como sinais criados especificamente para o projeto da arquitetura, criados a partir dos pacotes citados (definidos no arquivo PACOTE no apêndice A) [20].

A partir de um conjunto reduzido de instruções e comandos VHDL foi possível descrever os diferentes módulos que compõem o circuito de maneira real e concisa. Tudo isto associado ao conhecimento fundamentado da linguagem e de seus recursos.

Para a concepção da arquitetura, utilizou-se da composição esquemática dos blocos validados e de componentes presentes na biblioteca GEN_LIB (uma das bibliotecas digitais da Mentor Graphics). Formou-se assim um modelo estrutural da partição funcional da arquitetura em modelos compotamentais. O recurso de composição esquemática, disponível no ambiente da Mentor Graphics, além de poupar uma etapa de descrição escrita, proporciona a visualização da estrutura física com a qual se trabalha. Foi utilizada uma

metodologia de projeto "top-down", inciando-se com a idéia da arquitetura, sua partição funcional e o projeto destas partes.

A experiência própria mostrou que, pelo fato de ser uma das primeiras pessoas a utilizar esta ferramenta de projeto, assim como a linguagem de descrição de hardware VHDL na Unicamp, e também pela pequena quantidade de bibliografia a respeito do assunto, existiram várias dificuldades que tiveram de ser superadas de forma individual. A versão 8.1 VHDL da Mentor Graphics, usada em grande parte do projeto, apresentava inúmeros 'problemas no compilador e no simulador, impedindo em muitas vezes o prosseguimento do trabalho e em outras vezes forçando mudanças na forma de descrição.

A memória exigida pelas ferramentas Design Architect e principalmente pelo Quicksim representam um ônus bastante grande na memória disponível. São gerados na compilação e na simulação uma grande quantidade de arquivos auxiliares, onerando também espaço em disco disponível. O Quicksim é invocado individualmente e especificamente para cada bloco a ser simulado, gastando para isto um tempo razoável. Seria interessante poder carregar um bloco diferente para a simulação dentro do mesmo Quicksim invocado anteriormente para a simulação de outro bloco. Outra sugestão é que os recursos existentes para manuseio e observação de sinais na ferramenta Quicksim possam ser estendidos para as variáveis. Embora não tenha sido necessário neste projeto, seria interessante a incorporação de uma variedade de recursos gráficos de descrição como novas formas de entrada, como por exemplo, a entrada através de diagrama de estados. Outro ponto também não utilizado aqui, mas que merece consideração, é o aprimoramento e o aumento da compreensão do código por parte das ferramentas de síntese automática e sua aplicação em todos os níveis do projeto, desde a especificação até o "layout". Estas observações descritas acima deverão ser incorporadas nas versões futuras das ferramentas. Volta-se a salientar que os recursos disponíveis da linguagem VHDL estão diretamente relacionados com a exploração destes pelas ferramentas que as implementa, sendo a linguagem em si muito abrangente.

No projeto desenvolvido, a linguagemVHDL teve utilidade na simulação e validação da arquitetura idealizada, modelando seu funcionamento, independentemente do circuito e da tecnologia que a implementam. Não foi utilizada para a síntese automática de circuitos a partir de sua descrição. Para isto seriam necessárias algumas modificações na forma e nos comandos utilizados, para que a descrição fosse compreendida pela ferramenta

sintetizadora.

Capítulo 3

Arquiteturas para Processamento Digital de Imagens

3.1 Introdução

O processamento digital de imagens, caracterizado por uma grande carga computacional, requer, em muitos casos, a execução de operações em tempo real. O volume de dados envolvido neste processamento é geralmente muito alto devido à grande quantidade de pixels presentes na imagem. Além dos avanços na tecnologia de circuitos integrados, considerações sobre a paralelização do hardware são fundamentais para a aceleração do processamento e a viabilização da execução de determinadas operações sobre imagens. Existem arquiteturas específicas que abordam este tipo de problema. De um modo geral elas se classificam em processadores matriciais (processor arrays) e processadores de vizinhança.

Características temporais e espaciais do processamento de baixo nível de imagens propiciam a paralelização. O processamento pipeline explora o fato de que certas classes de processamento podem ser decompostas em uma seqüência de operações, explorando, assim, as características de paralelismo temporal. Os processadores matriciais exploram as características do paralelismo espacial onde uma mesma operação é executada em paralelo por todos os pixels da imagem (paralelismo de dados [21], [22].

3.2 Processamento de imagens

De maneira geral, o processamento digital de imagens consiste na manipulação, codificação e análise de informações pictóricas [23].

A codificação da imagem é a forma como se representa esta imagem, normalmente através de uma base de dados. A forma de armazenamento, recuperação e atualização destes dados se referem à manipulação desta base de dados.

A análise de imagens trata de operações em imagens tais como: extração de linhas, curvas e regiões em imagens, classificação de objetos com base em seu contorno, análise de texturas, análise de seqüência de imagens para a estimativa de movimento de objetos e análise de cenas. Esta fase de processamento transforma a imagem em uma outra imagem ou em uma série de características. Suas aplicações abrangem um largo espectro, desde análise de imagens em biologia celular, reconhecimento de caracteres e diagramas até aplicações industriais envolvendo visão robótica e controle de qualidade.

O modelo computacional normalmente associado ao processamento de imagens consiste de três níveis distintos [24]. O nível mais baixo é responsável por operações do tipo supressão de ruído, detecção de bordas e linhas, etc., os quais podem ser chamados de operações de filtragem num contexto mais amplo. Operam sobre dados organizados na forma de pixels (forma abreviada de "picture elements"), gerando como resultado uma imagem na mesma estrutura de dados. O nível intermediário opera sobre a matriz da imagem e produz como resultado uma lista de informações do tipo: histograma do nível de cinza, contagem de objetos, área, perímetro, linha de contorno, etc., num formato de representação diferente do formato da entrada, eventualmente. O nível mais alto é responsável pelo processamento e classificação da saída do nível intermediário ou de uma matriz imagem, produzindo uma descrição da cena.

A maioria das arquiteturas especiais de processamento de imagens enfocam principalmente o nível mais baixo do processamento por diversas razões: o conjunto das operações baixo nível é melhor definido, as operações são mais simples e os dados em maior quantidade requerem maior velocidade de execução, justificando o uso de estruturas especiais para o processamento.

3.3 Formas de paralelismo em imagens

As formas possíveis de paralelismo em hardware para processamento de imagens são paralelismo pixel-bit, paralelismo de vizinhança, paralelismo de imagem e paralelismo de operador que exploram tanto característica temporais como espaciais [23].

Paralelismo pixel-bit

No paralelismo pixel-bit cada pixel vindo da memória é tratado paralelamente a nível dos seus bits.

Paralelismo de vizinhança

No paralelismo de vizinhança se faz acesso paralelo ao pixel a ser tratado e aos seus pixels vizinhos, resultando em um pixel de saída definido em função da vizinhança.

Paralelismo de imagem

No paralelismo de imagem opera-se paralelamente as partes da imagem. Este paralelismo é típico de algumas classes de algoritmos onde uma imagem pode ser operada simultaneamente por vários processadores. Arquiteturas deste tipo podem ser representadas tanto por CPU's de propósito geral como por simples processadores bit-seriais presentes nas arquiteturas maciçamente paralelas.

Paralelismo de operador

Esta forma de paralelismo explora a possibilidade de fragmentação das operações sobre imagens em sub-operações sucessivas. É característico dos processadores pipeline onde cada operação é realizada por um estágio deste. A cada intervalo de tempo, cada estágio recebe uma entrada diferente, processada no último intervalo pelo elemento anterior. Este elemento realiza sobre os dados a devida operação neste estágio passando os dados

processados ao estágio seguinte. Este sistema explora o paralelismo temporal através da execução paralela de diferentes operações.

O paralelismo de imagem e pixel-bit são característicos dos processadores matriciais, explorando-se características espaciais. Já o paralelismo de vizinhança pode ser encontrado nos processadores pipeline e matriciais, tratados a seguir.

A arquitetura proposta no Capítulo 4 deste trabalho implementa operações de Morfologia Matemática sobre imagens binárias e numéricas, apresentando paralelismo bit-serial, de vizinhança e de operador.

3.4 Processadores matriciais

O desenvolvimento da tecnologia de VLSI ("Very Large Scale Integration") a partir da década de 70, contribuiu para a concepção de processadores dedicados a custos e escalas atrativos. Como exemplo deste desenvolvimento na área de Processamento Digital de Imagens podemos citar o estado operacional, no início dos anos 80, da primeira máquina maciçamente paralela, CLIP 4 [25], [26], baseada no modelo proposto por Unger [27] no final da década de 50. Inspirado nos trabalhos de von Neumann sobre autômatos celulares [28] o modelo de Unger consiste numa associação direta entre os elementos da imagem e as unidades de processamento. Estas últimas, operando simultaneamente, são todas idênticas, cada qual trabalhando sobre um pixel da imagem. É a chamada arquitetura SIMD definida por Flynn [7], usada em vários tipos de processamento de baixo nível de imagens, implementadas nos processadores matriciais.

Nestes processadores uma mesma instrução é propagada a todos os elementos da matriz que opera de maneira síncrona sobre todos os pixels da imagem. Estes elementos se comunicam entre si, trocando informações com seus vizinhos, realizando operações locais paralelamente [29]. A comunicação externa é um dos fatores que limita a velocidade de processamento deste modelo de arquitetura. A velocidade de suprimento de dados é insuficiente para a demanda de processamento, sendo necessário primeiramente carregar a imagem na matriz para depois transformá-la, o que conflita com a natureza serial de entrada da imagem [30]. O custo deste modelo de arquitetura é bastante elevado devido à grande

quantidade de hardware envolvido (uma célula por pixel).

Os Processadores Matriciais bit-seriais são constituídos por células processadoras de um bit, as quais geram serial e acumulativamente o resultado das operações sobre pixels numéricos. Isto proporciona flexibilidade e independência em relação ao tamanho do pixel a ser processado.

Existem ainda os Processadores Matriciais bit-paralelos que para aumentar a eficiência do sistema trabalham com a representação dos dados em paralelo ao contrário dos processadores bit-seriais. O processador VHDAP e o CLIP7 [31] são exemplos desta categoria.

3.4.1 Exemplos de processadores matriciais

A seguir são dados alguns exemplos de processadores matriciais do tipo bit-serial.

Clip4

"Celular Logic Image Processor 4" [25], [26] foi o primeiro processador bit-serial. Desenvolvido em 1974 por uma equipe da University College of London, é constituído por uma matriz de 96x96 processadores bit-serial e um módulo de interface com o exterior (figura 3.1).

Cada processador possui memória RAM local (32 registradores de 1 bit, 2 registradores de propósito geral de 1 bit e um registrador de transporte (carry) de 1 bit), opera de forma bit-serial e está conectado a seus 8 vizinhos próximos (figura 3.2). Realiza processamento booleano e soma completa. Todos os processadores recebem a mesma instrução de controle e operam sincronamente. Apresenta velocidade elevada para a realização das operações lógicas efetuadas sobre uma janela 3x3 [22], [23], [32], [33]. Realiza, por exemplo, operações de erosão ou dilatação binária em um ciclo de 10 μs e operação de adição pontual sobre 8 bits em 80 μs [26].

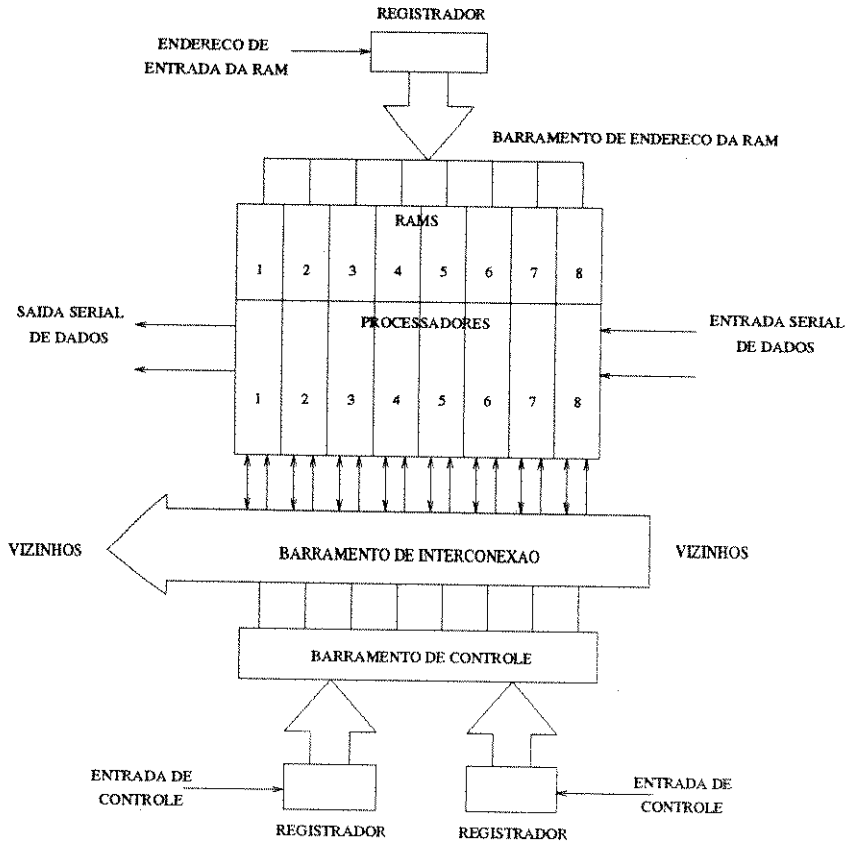

Figura 3.1: configuração do chip CLIP4

MPP

Em 1982 foi concluído o projeto do "Massively Parallel Processor". É uma matriz de 128x128 processadores bit-serial (figuras 3.3, 3.4). Seu projeto foi orientado para a realização de operações aritméticas, possuindo cada processador um somador completo e um registrador de deslocamento de tamanho variável.

A conexão aos 4 vizinhos mais próximos é feita através de um multiplexador. Os dados da imagem são armazenados em uma memória RAM local de alta velocidade e para cada processador é alocado 1K bit de memória [22], [23], [32], [33].

MPP realiza a operação de erosão ou dilatação binária em 1,5 μs e a operação de adição sobre oito bits em 2,5 μs [34].

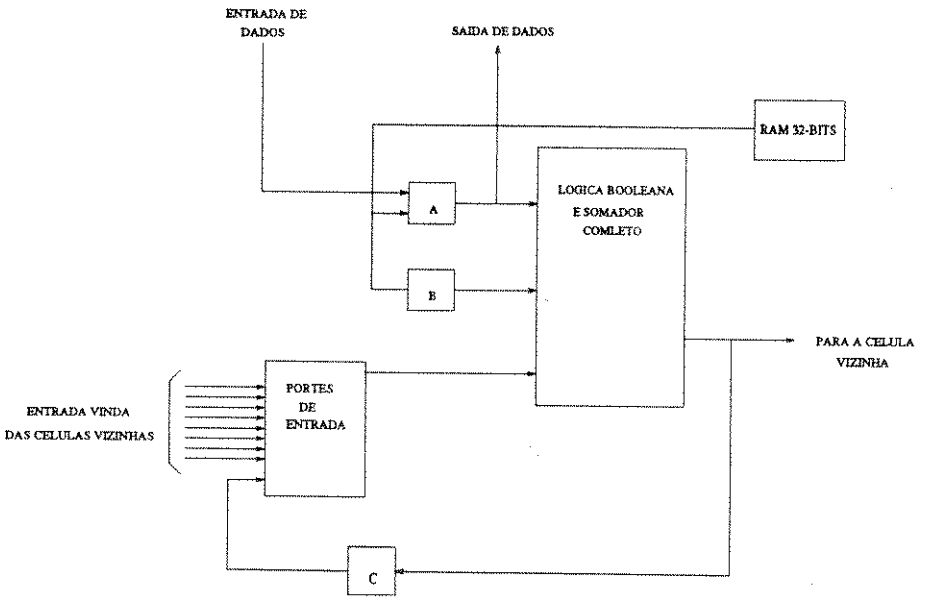

Figura 3.2: célula do processador CLIP4

DAP

O "Distributed Array Processor" foi concebido primeiramente para atender aplicações tais como análise de elementos finitos, modelagem matemática, análise de problemas metereológicos e outras áreas relacionadas. Graças à distribuição espacial das células, o DAP foi naturalmente associado ao tratamento de imagens [34].

Este sistema é constituído por uma matriz de 64x64 processadores bit-seriais. Cada um destes elementos possui 16K bits de memória, somador completo bit-serial e multiplexadores de entrada e saída e comunica-se com os 4 vizinhos mais próximos (figura 3.5). O barramento X-Y (linha-coluna) permite que os dados da imagem se movam diagonalmente na matriz, possibilitando operações sobre uma vizinhança 3x3 [22], [23], [32], [33], [35]. O DAP realiza as operações de erosão ou dilatação e adição sobre oito bits em 5 μs [34].

Gerritsen [36] faz um comparação dos 3 tipos de processadores descritos anteriormente. Outros exemplos de processadores matriciais bit-seriais são os processadores GRID [37], CAAPP [38], NTT [39].

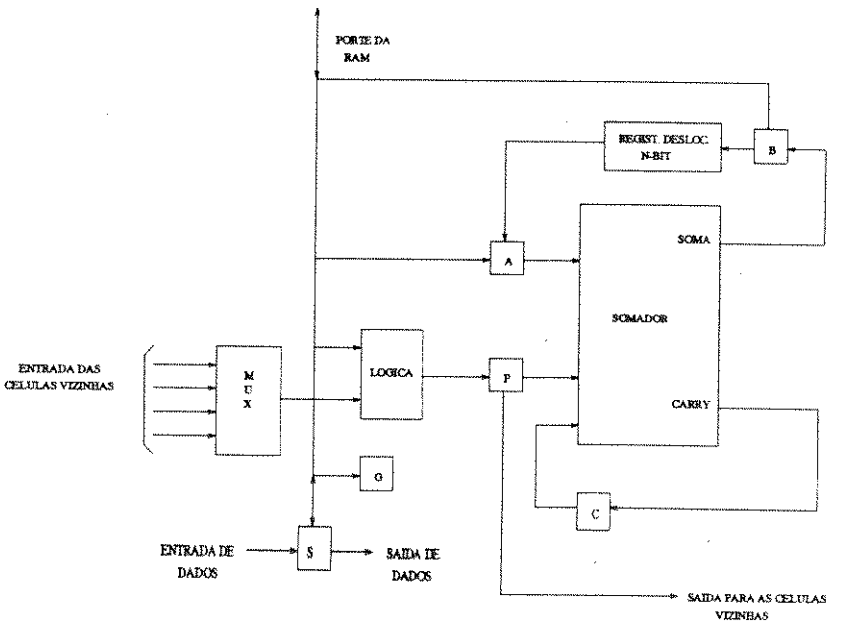

Figura 3.3: célula do processador MPP

3.5 Processadores pipeline

Processadores de vizinhança local são bastante apropriados à implementação de operações cujo pixel resultante é função do pixel de entrada e dos seus elementos vizinhos. Este tipo de operação é realizada a partir de diversos estágios encadeados seqüencialmente, geralmente em que cada estágio executa parte da operação a ser realizada (pipeline). O fluxo de entrada e saída de dados da estrutura é igual e constante, tendo-se simultaneamente a entrada de um pixel e a saída de um resultado referente ao processamento de um pixel anterior. O modo seqüencial de suprimento de dados e de geração dos resultados nesta arquitetura é adequado à comunicação serial que existe entre memória e processador. Os dados de entrada percorrem cada estágio do pipeline e são transformados à medida em que passam por eles, sendo a saída de um estágio a entrada do módulo seguinte.

Os processadores pipeline são convenientes à implementação de algoritmos de baixo nível em que o processamento pode ser decomposto em diferentes operações sucessivas. Um aspecto interessante neste tipo de paralelismo se refere à boa eficiência do sistema, já que cada estágio é construído de forma a atender mais eficientemente a operação a ser realizada. Um exemplo típico de processador pipeline é o Cytocomputer, capaz de realizar

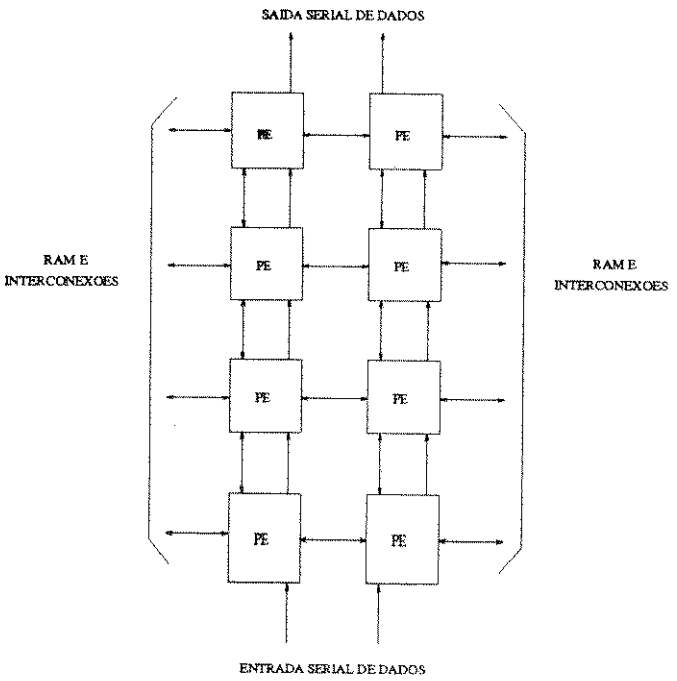

Figura 3.4: configuração do chip MPP

em cada estágio de seu pipeline uma operação morfológica completa [40].

Um estudo comparativo dos processadores matriciais e os processadores do tipo pipeline mostra que se o processamento de uma imagem for considerado a partir de uma matriz cuja dimensão é muito menor que a dimensão da imagem, então uma estrutura do tipo pipeline pode ter um melhor desempenho na execução da mesma tarefa [41].

3.5.1 Cytocomputer

O processador Cytocomputer foi desenvolvido por uma equipe do "Environmental Research Institute of Michigan". O Cytocomputer é um pipeline serial de processadores de vizinhança programáveis (estágios) [40]. É constituído por 88 estágios de processamento ligados em pipeline para o tratamento de imagens binárias, mais 25 estágios para tratamentos de imagens em nível de cinza (8 bits por pixel).

Cada estágio representa um processador de vizinhança programável e é formado basicamente por 2 registradores de deslocamento de tamanho fixo (509 estágios), uma sub-

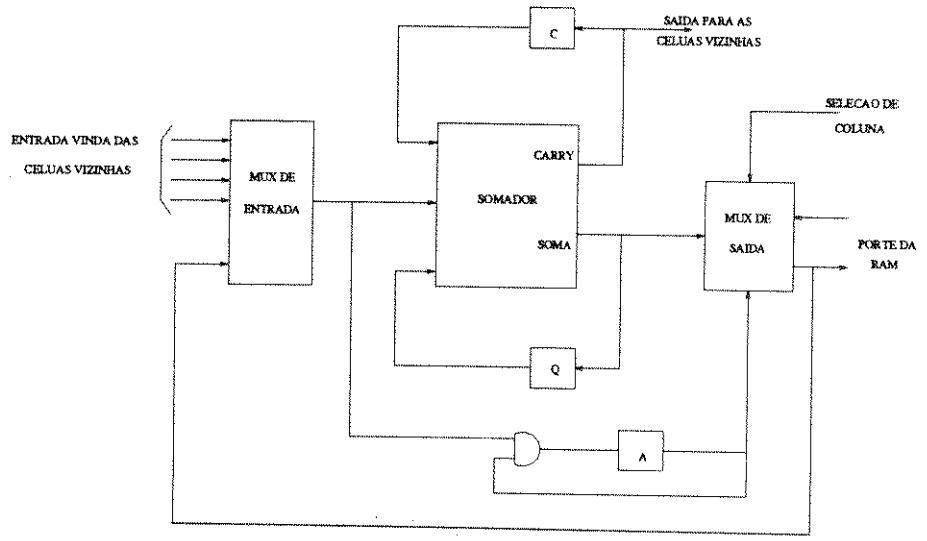

Figura 3.5: célula do processador DAP

matriz 3x3 de registradores os quais são acessados paralelamente por uma lógica de cálculo de operações de vizinhança. Esta lógica é uma "look-up table" que traduz a função a ser executada no estágio. Assim, conforme a programação das "look-up tables" na seqüência de estágios, define-se o algoritmo executado pelo Cytocomputer.

Dentro de cada estágio, os pixels entram serialmente se deslocando de maneira síncrona na estrutura de memória. Estando esta completa, os pixels vizinhos ao elemento a ser calculado (vizinhança 3x3) se encontram presentes na sub-matriz e são acessados simultaneamente pela lógica de cálculo. Há um fluxo de entrada e de saída de pixels constante, ocorrendo em cada intervalo o cálculo de um pixel resultante [42], [33].

Este processador opera sobre imagens de tamanho 512 x infinito, sendo sua aplicação interessante na implementação de operações que possam ser divididas em um número de sub-operações (etapas) igual ao número de estágios do processador [35]. A velocidade do Cytocomputer é de aproximadamente 1.6 milhões 8-bits pixels/s [40].

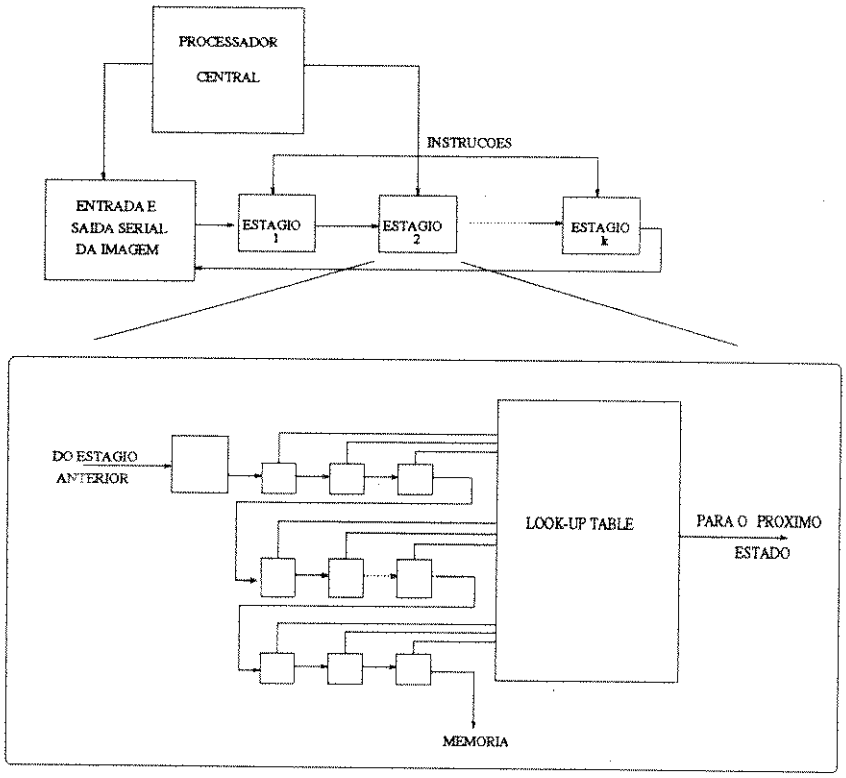

Figura 3.6: arquitetura do Cytocomputer

3.6 Outros modelos de arquiteturas para processamento de imagens

3.6.1 Processadores piramidais

Conforme descrito em [33], um processador piramidal é formado por uma base, composta de uma matriz de processadores, e por camadas sucessivamente menores de matriz de processadores ou outra topologia, formando uma estrutura tri-dimensional. Há subordinação hierárquica dos níveis inferiores com relação aos superiores. Outras características desta estrutura são o uso de células simples e de tipos pouco variados, controle e fluxo de dados regular e simples e conseqüentemente interconexões também simples e regulares.

Como exemplo desta categoria citamos os processadores PLCA [43], SPHINX [44] e PAPIA [45]. Este último é constituído por 5 processadores (pai e 4 filhos) dispostos em

pirâmide. Cada processador é um elemento bit-serial com processador booleano, somador completo, conjunto de registradores de 1 bit, 2 registradores de comprimento programável, 1 registrador comparador e 256 bits de memória local [32].

3.6.2 Processadores associativos

Processadores associativos são aqueles que fazem uso de memória associativa, ou seja memórias endereçáveis pelo conteúdo. Para se implementar este tipo de estrutura é necessário lógica adicional de manipulação dos dados para leitura e armazenamento, em compensação há, naturalmente, um ganho no desempenho do processador.

Podemos citar o STARAN [46] e o SCAPE [47] como pertencentes a esta classe de processadores. O SCAPE apresenta 256 processadores que possuem processamento boleano e associativo, sendo a rede de interconexão entre eles reconfigurável [32].

3.6.3 Processadores sistólicos

As arquiteturas sistólicas foram desenvolvidas inicialmente na Universidade de Carnegie-Mellon e encontraram aplicação na área de processamento de imagens devido à alta capacidade e velocidade de processamento e a uma eficiência nas características do hardware resultante [48]. Um sistema sistólico é uma matriz geralmente uni ou bidimensional de processadores ou células. Estas células são simples e homogêneas e são agrupadas conforme a estrutura de processamento desejada. A comunicação entre elas é local, simples, regular e síncrona, o que contrasta com a maioria dos sistemas formados por módulos agrupados. Deste modo, o fluxos de dados flui sincrona e regularmente através da estrutura com os dados interagindo dentro de cada uma das células. Esta característica gera uma estrutura de controle simples em que a única preocupação é a introdução de dados numa extremidade e a retirada dos resultados na extremidade oposta. São características importantes para implementação em VLSI do modelo sistólico a modularidade, o controle simples, sincronicidade e a regularidade de dados e da estrutura [49]. GAPP [50] e PSC [51] e NPP [52] são exemplos deste tipo de processador.

3.6.4 Microprocessadores

Problemas de processamento de imagens podem ser tratados através de um conjunto de microprocessadores interconectados, realizando múltiplas instruções sobre dados diferentes(MIMD). Este tipo de estrutura é bastante apropriada ao paralelismo a nível de regiões, onde cada uma destas pode ser associada a um processador diferente [48]. A dificuldade neste tipo de configuração está na partição das operações entre os microprocessadores e na comunicação entre eles. O número de processadores envolvidos, o modo de interconexão e a memória são fatores determinantes no que se refere à eficiência de processamento.

O Transputer (INMOS, 1983) é um microprocessador modular de 32 bits projetado para conexão numa grande variedade de configurações MIMD. Cada Transputer apresenta 4 canais independentes para comunicação com o exterior e 4Kbytes de memória de programa e de dados [53].

Capítulo 4

Arquitetura para Operações de Morfologia Matemática sobre Imagens

4.1 Introdução

Neste capítulo é descrita uma arquitetura sistólica que implementa filtros não-lineares (filtros de morfologia matemática) sobre imagens binárias e em nível de cinza. A arquitetura é regular e modular, permitindo uma implementação VLSI eficiente. A arquitetura é apropriada para operações de dilatação e erosão por um elemento estruturante planar, nos modos binário e numérico, com aproveitamento máximo do hardware em ambos os modos. Foram consideradas também as operações morfológicas de threshold, inversão, subtração, máximo e mínimo. A simulação desta arquitetura foi realizada através da linguagem de descrição de hardware VHDL. Comparações entre esta arquitetura e outras são realizadas, destacando-se os seus aspectos importantes.

4.2 Morfologia matemática

Morfologia matemática é o estudo da estrutura e da forma dos objetos, sob o ponto de vista matemático, e a aplicação desta teoria na transformação de imagens para extração de informações. É um método de análise de imagens em que estruturas são compreendidas através da melhoria da imagem e do reconhecimento de padrões [54].

No início do século, Minkowski desenvolveu princípios de álgebra espacial, a partir dos quais, após 1964, Matheron [55] e Serra [54] fundamentaram teoricamente o assunto, suas propriedades, operações sobre imagens e aplicações básicas. Sternberg introduziu esta teoria em imagens biomédicas e industriais [56], [57].

A morfologia matemática tem como operações básicas a dilatação e a erosão que combinadas produzem operações mais complexas tais como aberturas, fechamentos, gradientes, esqueletos. Encontramos sua aplicação em áreas como geologia, sensoriamento remoto, astronomia, automação industrial, biomedicina, metalografia, etc., ou seja, em todas as áreas que trabalham com informações pictóricas.

Implementando estas operações em hardware dedicado é possível explorar ao máximo as possibilidades de paralelização inerentes a elas e alcançar maior simplicidade de hardware. O crescente desenvolvimento da Morfologia Matemática nos últimos anos tem levado vários grupos de pesquisa a implementar suas operações básicas em hardware [52], [58], [59], [60], [61].

A arquitetura que será descrita neste capítulo, implementa operações morfológicas de vizinhaça de dilatação e erosão. Estas operações trabalham com uma matriz estruturante (ou elemento estruturante) que corresponde a uma forma dada como parâmetro a um operador morfológico. Numa operação de vizinhança, o resultado de um pixel é função dos pixels da sua vizinhança indicados pelo elemento estruturante.

4.2.1 Definições e notação

Seja Z o conjunto de inteiros. Seja E um retângulo de Z^2 e K um intervalo $[0,k]$ de Z, com $k > 0$. As imagens formam um conjunto de funções de E para K. Este

conjunto é denominado K^E e f,g são elementos genéricos de K^E. Se $k = 1$, diremos que a imagem é binária e se $k > 1$ chamamos de imagem em nível de cinza [62].

Seja B um subconjunto de Z^2 chamado conjunto estruturante. B_h indica a translação de B por qualquer vetor h em Z^2, isto é, $B_h = \{x + h : x \in B\}$.

A dilatação de f por B é a função $\delta_B(f)$ em K^E dada na equação 4.1, para qualquer x em E:

$$\delta_B(f)(x) = max\{f(y) : y \in B_x^t \cap E\} \tag{4.1}$$

onde:

$B^t = \{-x, x \in B\}$ é a transposta de B.

A erosão de f por B é a função $\varepsilon_B(t)$ em K^E dado na equação 4.2, para qualquer x em E:

$$\varepsilon_B(f)(x) = min\{f(y) : y \in B_x \cap E\} \tag{4.2}$$

Seja W um retângulo de Z^2 onde é definida a função M em W^K, denominada matriz estruturante, associada ao conjunto estruturante B, dada na equação 4.3, para qualquer x em W:

$$M(x) = \begin{cases} k & se\ x \in B \\ 0 & se\ x \notin B \end{cases} \tag{4.3}$$

e,

$$M(y - x) = \begin{cases} k & se\ y \in B_x \\ 0 & se\ y \notin B_x \end{cases} \tag{4.4}$$

Utilizando-se a equação da dilatação 4.1, chega-se à seguinte formulação em função de M:

$$\delta_B(f)(x) = max\{min\{f(y), M(-y-x)\} : y \in W_x \cap E\}, \tag{4.5}$$

$$\delta_B(f)(x) = max\{min\{f(y+x), M(-y)\} : y \in W \ e \ y+x \ \in E\} \tag{4.6}$$

analogamente,

$$\varepsilon_B(f)(x) = min\{min\{f(y), M(y-x)\} : y \in W_x \cap E\}, \tag{4.7}$$

$$\varepsilon_B(f)(x) = min\{min\{f(y+x), M(y)\} : y \in W \ e \ y+x \in E\} \tag{4.8}$$

onde:

W é o domínio da matriz estruturante.

$M(x)$ são os elementos da matriz estruturante, de dimensão $m \times n$ (linhas por colunas), supondo que a sua origem está no elemento central. $M(x)$ assume os valores 0 e k (planar).

E é o domínio da imagem de entrada.

$f(x)$ é a imagem a ser operada.

x e y são elementos de Z^2.

É importante notar que as definições acima se aplicam tanto para imagens binárias como para imagens numéricas. Na dilatação, é considerada a matriz transposta $M(-y)$.

4.2.2 Exemplo

$$\mathrm{M(y)} = \begin{bmatrix} M[-1,-1] & M[-1,0] & M[-1,1] \\ M[0,-1] & M[0,0] & M[0,1] \\ M[-1,-1] & M[1,0] & M[1,1] \end{bmatrix}$$

Para uma matriz estruturante $M(y)$ 3x3 y assume as coordenadas possíveis para este formato: $[-1,-1]$, $[-1,0]$, $[-1,1]$, $[0,-1]$, $[0,0]$, $[0,1]$, $[1,-1]$, $[1,0]$, $[1,1]$. Operando-se esta matriz com uma imagem $f(j,i)$, chega-se a partir de 4.6 e 4.8, nas seguintes equações ((4.9, 4.10) para a dilatação e para a erosão:

$$\begin{aligned}\delta_B(f)(x) = \; & max\{min\{f(j-1,i-1), M(-1,-1)\}, min\{f(j-1,i), M(-1,0)\},\\ & min\{f(j-1,i+1), M(-1,1)\}, min\{f(j,i-1), M(0,-1)\},\\ & min\{f(j,i), M(0,0)\}, min\{f(j,i+1), M(0,1)\},\\ & min\{f(j+1,i-1), M(1,-1)\}, min\{f(j+1,i), M(1,0)\},\\ & min\{f(j+1,i+1), M(1,1)\}\}\end{aligned} \tag{4.9}$$

$$\begin{aligned}\varepsilon_B(f)(x) = \; & min\{min\{f(j-1,i-1), M(-1,-1)\}, min\{f(j-1,i), M(-1,0)\},\\ & min\{f(j-1,i+1), M(-1,1)\}, min\{f(j,i-1), M(0,-1)\},\\ & min\{f(j,i), M(0,0)\}, min\{f(j,i+1), M(0,1)\},\\ & min\{f(j+1,i-1), M(1,-1)\}, min\{f(j+1,i), M(1,0)\},\\ & min\{f(j+1,i+1), M(1,1)\}\}\end{aligned} \tag{4.10}$$

A matriz estruturante opera com uma imagem $f(x)$ centrada sobre o pixel corrente, ou seja, o elemento $M[0,0]$ da matriz estruturante opera com $f(j,i)$ e os demais operam com os outros pixels desta vizinhança.

Ilustrando este formalismo através de um exemplo numérico de dilatação de uma imagem binária 4.11 por uma matriz estruturante 4.12, tem-se a imagem resultado 4.13:

Imagem de Entrada =

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.11}$$

Matriz Estruturante =

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.12}$$

Imagem Dilatada =

$$\begin{bmatrix} X & X & X & X & X & X & X & X \\ X & 0 & 1 & 0 & 1 & 1 & 0 & X \\ X & 1 & 1 & 1 & 1 & 1 & 1 & X \\ X & 1 & 1 & 1 & 1 & 1 & 1 & X \\ X & 0 & 1 & 1 & 1 & 1 & 1 & X \\ X & 1 & 1 & 1 & 1 & 1 & 0 & X \\ X & 0 & 1 & 1 & 1 & 0 & 0 & X \\ X & X & X & X & X & X & X & X \end{bmatrix} \tag{4.13}$$

Executando esta dilatação para uma imagem em níveis de cinza 4.14, tem-se a imagem 4.16:

Imagem de Entrada =

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 23 & 103 & 7 & 12 & 0 & 0 \\ 0 & 0 & 255 & 13 & 41 & 21 & 0 & 0 \\ 0 & 0 & 33 & 1 & 231 & 21 & 0 & 0 \\ 0 & 0 & 15 & 111 & 235 & 63 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.14}$$

Matriz Estruturante =

$$\begin{bmatrix} 0 & 255 & 0 \\ 255 & 0 & 255 \\ 0 & 255 & 0 \end{bmatrix} \tag{4.15}$$

Imagem Dilatada =

$$\begin{bmatrix} X & X & X & X & X & X & X & X \\ X & 0 & 23 & 103 & 7 & 12 & 0 & X \\ X & 0 & 255 & 13 & 41 & 21 & 0 & X \\ X & 255 & 33 & 255 & 231 & 41 & 21 & X \\ X & 33 & 255 & 231 & 235 & 231 & 21 & X \\ X & 15 & 111 & 235 & 231 & 235 & 63 & X \\ X & 0 & 15 & 111 & 235 & 63 & 0 & X \\ X & X & X & X & X & X & X & X \end{bmatrix} \tag{4.16}$$

Os valores representados por X representam os pixels de borda da imagem.

Nas Figuras 4.1 e 4.2 tem-se visualmente mostrada a dilatação e a erosão de uma imagem (em preto) por um elemento estruturante (esfera) resultando na imagem representada em cinza.

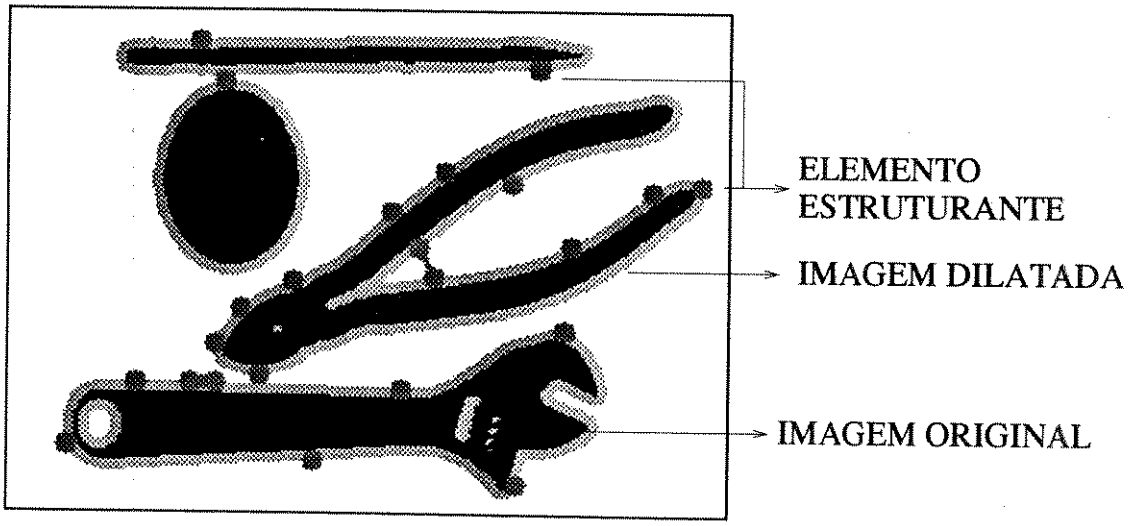

Figura 4.1: dilatação

4.3 Arquitetura SAMM

A arquitetura apresentada aqui - SAMM (Systolic Architecture for Mathematical Morphology) - possui características de modularidade e regularidade em sua estrutura. Isto é alcançado através da decomposição da operação morfológica (equações 4.6 e 4.8) em sub-operações realizadas por blocos de circuito (elementos morfológicos) que se encadeiam para realizar a operação total. Cada elemento morfológico é responsável pelo cálculo de um fator do resultado. Os dados são introduzidos na estrutura, e o resultado é gerado gradativa e acumulativamente durante a passagem destes através dos diferentes blocos.

De acordo com a equação 4.6, a operação de dilatação é fragmentada em vários mínimos e o máximo entre eles, sendo que o máximo é calculado acumulativamente à medida que os mínimos são gerados. O correspondente ocorre para a operação de erosão, formada pelo mínimo dos mínimos. Portanto, na dilatação, o elemento morfológico é formado por um circuito de cálculo de mínimo ligado à entrada de um circuito de cálculo de máximo e um "flip-flop" de defasagem do resultado (Figura 4.3). Na erosão, o elemento morfológico é composto por dois circuitos de cálculo de mínimo e um "flip-flop". Nesta arquitetura um elemento morfológico serve tanto para dilatação como para erosão pela seleção da função MÍNIMO/MÁXIMO.

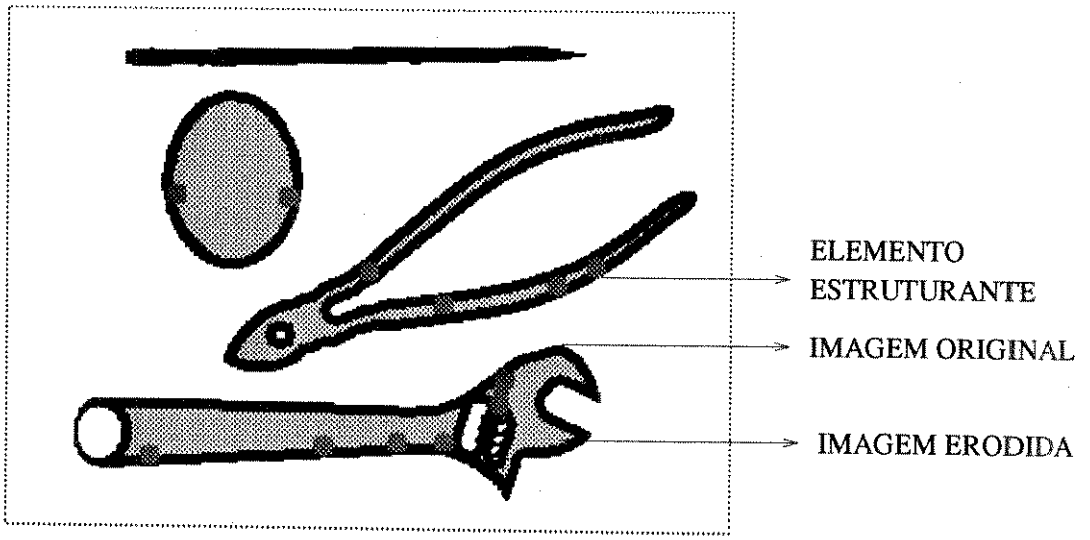

Figura 4.2: erosão

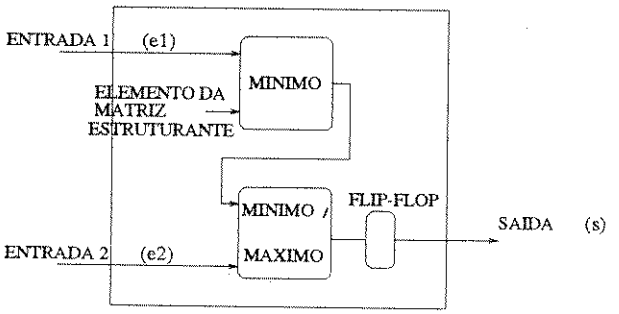

Figura 4.3: elemento morfológico

Na arquitetura SAMM não há distinção entre processador e elementos de memória, estando parte de ambos presentes na estrutura do elemento morfológico (circuitos de máximo e mínimo e "flip-flop") - Figura 4.3.

A Tabela 4.1 apresenta o diagrama tempo-espaço transitório da erosão de uma imagem $f(x)$ por uma matriz estruturante 1×3 (W), implementada através da estrutura da Figura 4.4. A matriz estruturante determina a dimensão e o formato da estrutura de processamento, estando cada um dos seus elementos armazenado em um elemento morfológico (registrador). O cálculo do pixel $\varepsilon_B(f)(j,i)$ é dado pela equação 4.17

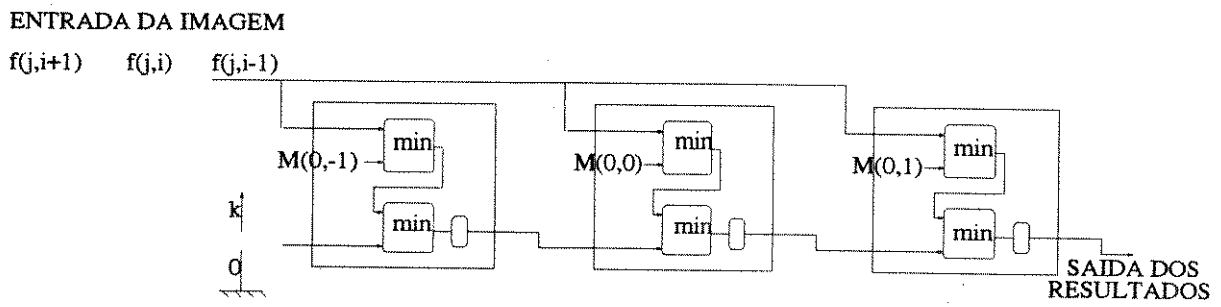

Figura 4.4: estrutura 1x3

$$\varepsilon_B(f)(j,i) = min\{min(f(j,i-1),M(0,-1)),k,$$

$$min(f(j,i),M(0,0)),min(f(j,i+1),M(0,1))\} \quad (4.17)$$

Como mencionado anteriormente, o cálculo de cada pixel ocorre gradativa e acumulativamente quando da passagem dos dados através da estrutura. O pixel da imagem de entrada é fornecido a todos os elementos morfológicos simultaneamente, e tratado em cada um por um coeficiente diferente da matriz estruturante presente no respectivo elemento. Na erosão, os mínimos vão sendo calculados em cada elemento morfológico assim que um novo pixel se encontra disponível. Este valor de mínimo é aproveitado dentro do mesmo elemento morfológico em que foi calculado para o cálculo do mínimo parcial. Este último cálculo de mínimo agrega o mínimo recém calculado com um valor vindo do elemento morfológico anterior. O resultado final é formado à medida que os pixels são propagados pela estrutura, chegando à sua saída totalmente definidos. Nota-se então que o cálculo é fragmentado, sendo executado cada uma de suas partes por um elemento morfológico. A estrutura apresenta comunicação local e síncrona, sendo composta por elementos simples e regulares, com taxa de comunicação constante com o meio externo, características estas típicas de um sistema sistólico [64].

Na operação de erosão (dilatação) a entrada 2 (Figura 4.3) do primeiro elemento morfológico é forçada a k (0) como condição inicial para a garantia do cálculo correto do primeiro máximo parcial [Tabela 4.1].

TEMPOS		ELEMENTOS		
		elemento 1	elemento 2	elemento 3
T 1	entrada 1	$f(j,i-1)$	$f(j,i-1)$	$f(j,i-1)$
	entrada 2	k	X	X
	saída	**min(min($f(j,i-1)$,$M(0,-1)$),k))**	X	X
T 2	entrada 1	$f(j,i)$	$f(j,i)$	$f(j,i)$
	entrada 2	k	min(min($f(j,i-1)$,$M(0,-1)$),k))	X
	saída	min(min($f(j,i)$,$M(0,-1)$),k))	**min(min($f(j,i-1)$,$M(0,-1)$),k, min($f(j,i)$,$M(0,0)$))**	X
T 3	entrada 1	$f(j,i+1)$	$f(j,i+1)$	$f(j,i+1)$
	entrada 2	k	min(min($f(j,i)$,$M(0,-1)$),k))	min(min($f(j,i-1)$,$M(0,-1)$),k, min($f(j,i)$,$M(0,0)$),
	saída	min(min($f(j,i+1)$,$M(0,-1)$),k)	min(min($f(j,i)$,$M(0,0)$),k min($f(j,i+1)$,$M(0,1)$))	**min(min($f(j,i-1)$,$M(0,-1)$),k, min($f(j,i)$,$M(0,0)$), min($f(j,i+1)$,$M(0,1)$))**

Tabela 4.1: Diagrama tempo-espaço transitório da erosão para a estrutura 1×3

4.3.1 Configurações da arquitetura

A complexidade de SAMM é diretamente proporcional ao tamanho da matriz estruturante (W) e independe do tamanho da imagem. Para operar com uma matriz estruturante de $m \geq 2$ (onde m é o número de linhas da matriz estruturante), são necessários elementos de atraso entre uma linha e outra da estrutura para armazenar o pixel até o instante da sua utilização pela próxima linha. Para operar com uma imagem de N colunas por uma matriz estruturante 3×1, temos a estrutura mostrada na Figura 4.5 e seu funcionamento representado no diagrama tempo-espaço da tabela 4.2.

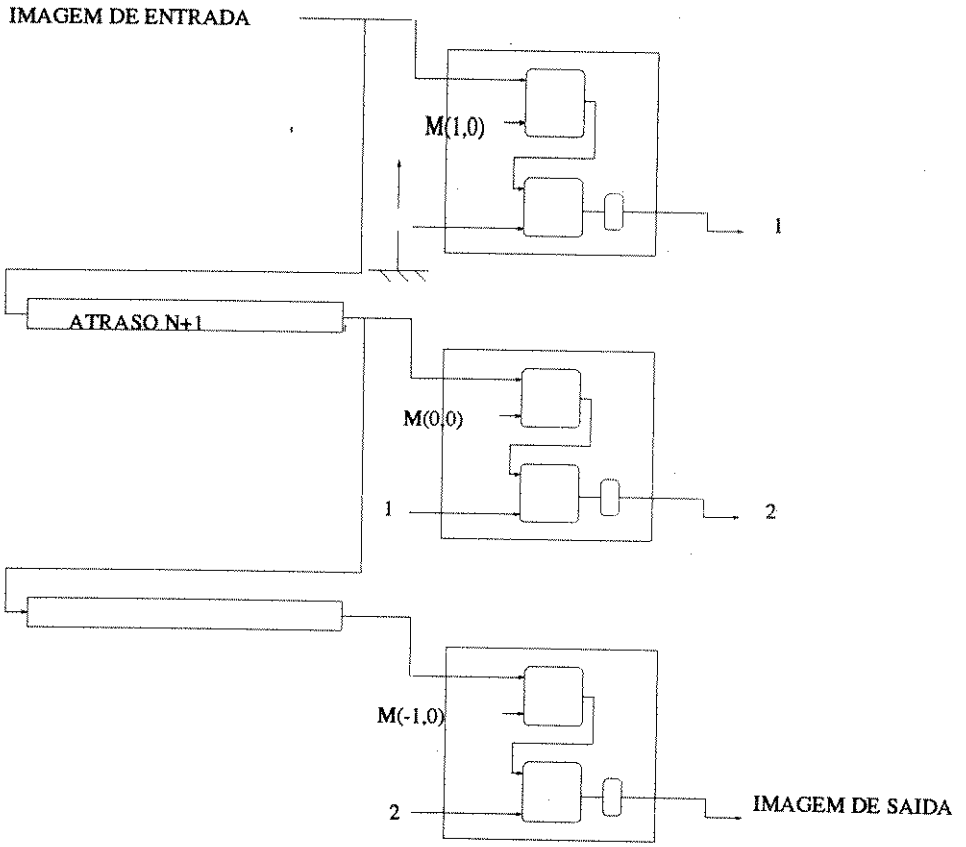

Figura 4.5: estrutura 3x1

Assim, considerando:

$$\text{Matriz Estruturante} = \begin{bmatrix} M[-1,0] \\ M[0,0] \\ M[1,0] \end{bmatrix}$$

$$\text{Matriz Imagem} = \begin{bmatrix} & & \vdots & & \\ .... & f(j-1,i-1) & f(j-1,i) & f(j-1,i+1) & .... \\ .... & f(j,i-1) & f(j,i) & f(j,i+1) & .... \\ .... & f(j+1,i-1) & f(j+1,i) & f(j+1,i+1) & .... \\ & & \vdots & & \end{bmatrix},$$

e como unidimensionais os índices da matriz imagem, tem-se a matriz a seguir:

$$\text{Matriz Imagem} = \begin{bmatrix} f(0) & .... & .... & .... & f(N-1) \\ & & \vdots & & \\ .... & f(k-N-1) & f(k-N) & f(k-N+1) & .... \\ .... & f(k-1) & f(k) & f(k+1) & .... \\ .... & f(k+N-1) & f(k+N) & f(k+N+1) & .... \\ & & \vdots & & \end{bmatrix}$$

A equação 4.12 enfoca a formação do pixel $\varepsilon_B(f)(k)$.

$$\begin{aligned} \varepsilon_B(f)(k) &= min\{min(f(k-N), M(-1,0)), k, \\ & \qquad min(f(k), M(0,0)), min(f(k+N), M(1,0))\} \end{aligned} \tag{4.18}$$

A matriz estruturante é sempre posicionada no processador com uma rotação 180 graus em relação ao seu eixo horizontal, tornando-se isto visível neste caso. Assim, no cálculo de um pixel, é obtido, na primeira linha da estrutura, o resultado parcial referente à operação dos pixels de entrada com relação à linha inferior da matriz estruturante original.

Devido à forma de propagação do pixel, o qual é entregue simultaneamente a todos os elementos de processamento da linha da estrutura, e ao registrador de deslocamento

TEMPOS		ELEMENTOS		
		elemento 1	elemento 2	elemento 3
T_{k+N}	e1	$f(k+N)$	$f(k-1)$	$f(k-N-2)$
	e2	k	min(min($f(k+N-1)$,$M(1,0)$),k)	min(min($f(k-3N-2)$,$M(1,0)$),k min($f(k-2N-2)$,$M(-1,0)$))
	s	**min(min($f(k+N)$,$M(1,0)$),k)**	min(min($f(k+N-1)$,$M(1,0)$),k, min($f(k-1)$,$M(0,0)$))	min(min($f(k-3N-2)$,$M(1,0)$),k, min($f(k-2N-2)$,$M(0,0)$)) min($f(k-N-2)$,$M(0,0)$))
T_{k+N+1}	e1	$f(k+N+1)$	$f(k)$	$f(k-N-1)$
	e2	k	min(min($f(k+N)$,$M(1,0)$),k)	min(min($f(k+N-1)$,$M(1,0)$),k min($f(k-1)$,$M(1,0)$))
	s	min(min($f(k+N+1)$,$M(1,0)$),k)	**min(min($f(k+N)$,$M(1,0)$),k, min($f(k)$,$M(0,0)$))**	min(min($f(k+N-1)$,$M(1,0)$),k min($f(k-1)$,$M(1,0)$), min($f(k-N-1)$,$M(-1,0)$))
T_{k+N+2}	e1	$f(k+N+2)$	$f(k+1)$	$f(k-N)$
	e2	k	min(min($f(k+N+1)$,$M(1,0)$),k)	min(min($f(k+N)$,$M(1,0)$),k, min($f(k)$,$M(0,0)$))
	s	min(min($f(k+N+2)$,$M(1,0)$),k)	min(min($f(k+N+1)$,$M(1,0)$),k, min($f(k+1)$,$M(0,0)$))	**min(min($f(k+N)$,$M(1,0)$),k, min($f(k)$,$M(0,0)$), min($f(k-N)$,$M(-1,0)$))**

Tabela 4.2: Diagrama em regime tempo-espaço da erosão para a estrutura 3×1

da linha seguinte, o tempo de armazenamento do pixel na arquitetura SAMM é de $N + n$ períodos (onde N e n são o número de colunas da imagem e da matriz estruturante, respectivamente). Como o resultado parcial de uma linha leva n períodos para ser formado e estar disponível na saída, são necessários, no registrador de deslocamento, além de N estágios para armazenar a linha de entrada, mais n estágios para tornar disponível o pixel para o próximo processamento.

A Figura 4.6 mostra a implementação bidimensional desta arquitetura para a operação com uma matriz estruturante 3x3. Nesta configuração o resultado do pixel em questão é função de seus pixels na sua vizinhança 3x3.

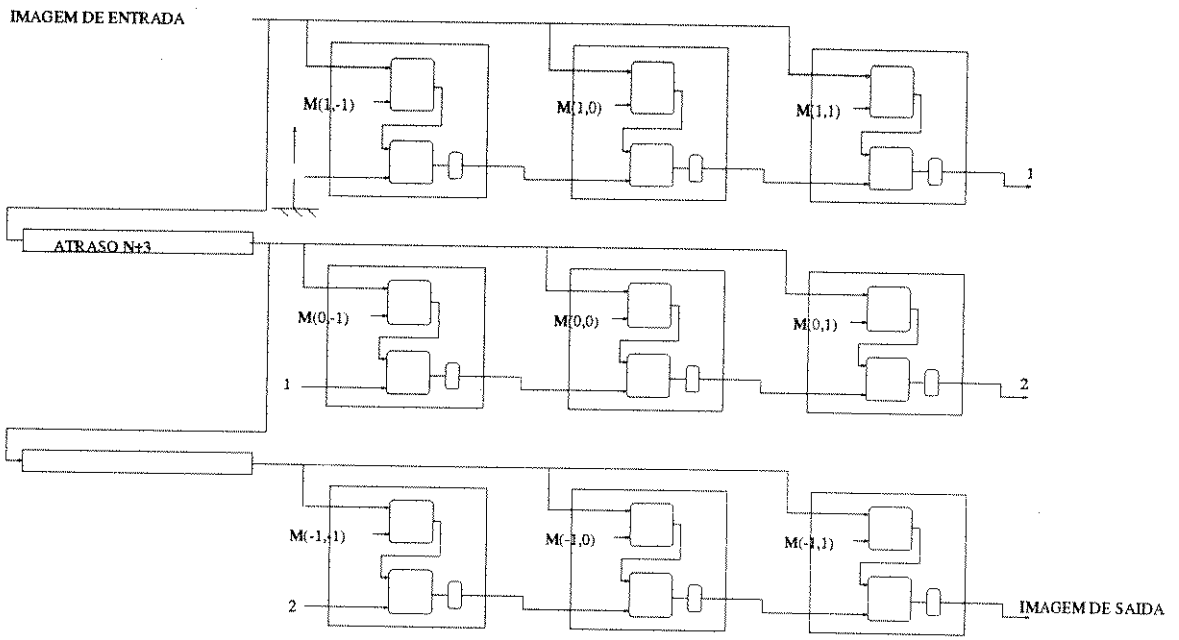

Figura 4.6: implementação de uma estrutura 3x3 para operações morfológicas

4.3.2 Modos numérico e binário de processamento

Esta arquitetura baseia-se na proposta de partição feita em [61] no que diz respeito ao aproveitamento integral do hardware usado em operações sobre imagens em níveis de cinza, como será visto a seguir.

Decompondo os circuitos de cálculo de máximo e mínimo para uma imagem

numérica em "fatias" binárias, tem-se o elemento morfológico binário mostrado na Figura 4.7.

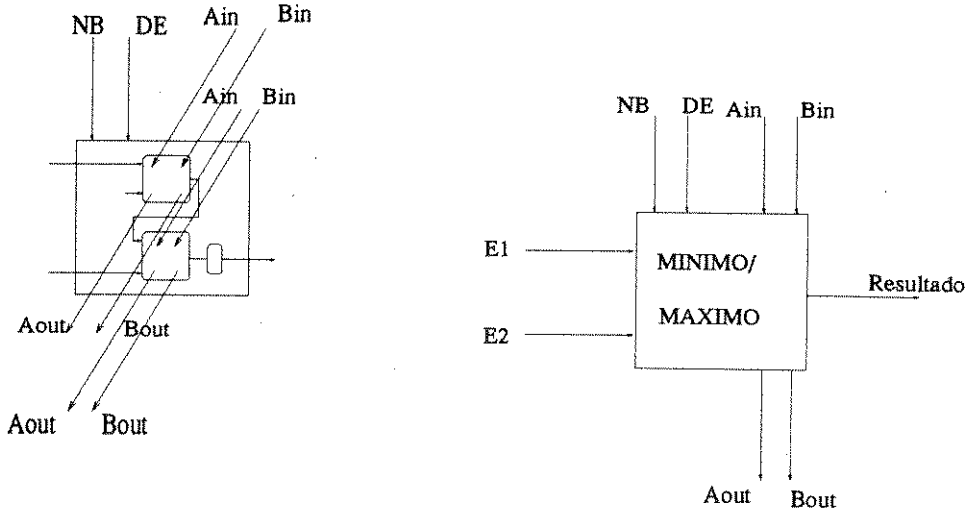

Figura 4.7: elemento morfológico binário e circuito de máximo

Para que o cálculo seja feito apropriadamente sobre o pixel numérico a partir deste elemento binário, é necessária a interligação de um conjunto destes através de dois tipos de sinais, A e B, formando um elemento morfológico numérico que opera sobre um byte (Figura 4.8). Os sinais *Ain* e *Aout* indicam se foi possível determinar o maior ou menor bit (conforme o caso de máximo ou mínimo, respectivamente), através das entradas binárias de cada elemento morfológico binário ($E1$ e $E2$). Os sinais *Bin* e *Bout* indicam qual das duas entradas binárias foi selecionada para a saída como resultado do máximo ou do mínimo. Tem-se ainda a entrada NB que indica se a operação será executada sobre uma imagem numérica ou binária, e a entrada Max que indica se a operação é de erosão ou dilatação. Foi montada a tabela verdade das entradas e saídas do circuito de máximo/mínimo decomposto em partes binárias para seu funcionamento quando interconectado a outros (modo numérico) ou isolado (modo binário). Pela minimização lógica desta tabela através de um software apropriado - MultPlex [65] -, chegou-se às seguintes equações.

$$Resultado = C1 + C2 + C3 + C4 + C5 + C6 + C9$$
$$Aout = C0 + C1 + C2 + C3 + C7 + C8 + C10$$
$$Bout = C0 + C3 + C5 + C8$$

onde os mintermos são dados por:

$C0 = NB.E2.\bar{E}1.\bar{D}E.\bar{A}in$
$C1 = E2.\bar{E}1.DE.\bar{A}in$
$C2 = NB.E2.Ain.\bar{B}in$
$C3 = NB.\bar{E}1.\bar{A}in.Bin$
$C4 = \bar{N}B.E1.DE$
$C5 = E1.DE\bar{A}in$
$C6 = \bar{N}B.E2.DE$
$C7 = NB.\bar{E}2.E1$
$C8 = NB.Ain.Bin$
$C9 = E2.E1$
$C10 = NB.Ain$

Estas equações conduzem ao cálculo de máximo ou mínimo numérico (quando $NB = 1$), fazendo com que os elementos morfológicos operem integradamente, trocando os sinais A e B entre si, formando um elemento morfológico numérico (Figura 4.8). Como os sinais A e B percorrem o módulo numérico do elemento morfológico binário associado ao bit mais significativo para o menos significativo, na estabilização da lógica combinacional de cálculo, tem-se na saída um dos bytes da entrada (representando o valor de máximo ou mínimo numérico). É necessário forçar Ain e Bin a zero no elemento morfológico associado ao bit mais significativo.

Para a operação sobre uma imagem binária ($NB = 0$), os sinais *Ain*, *Bin*, *Aout*, *Bout* são todos iguais a zero, o que significa uma não comunicação entre os elementos morfológicos binários que passam a operar de forma independente.

A célula básica para operações morfológicas 3x3, mostrada nas Figuras 4.9, 4.8), é composta por nove elementos morfológicos binários encadeados e dois registradores de deslocamento. Para operação no modo numérico, $k = 255$ (byte), são utilizadas oito células básicas com seus elementos morfológicos binários interligados conforme descrição anterior (Figura 4.8).

Apenas uma célula básica é suficiente para a realização de uma operação morfológica binária completa. Para o aproveitamento do hardware, o modo binário é configurado com oito células básicas encadeadas seqüencialmente, conforme a Figura 4.9. Esta configuração permite a realização de até oito operações morfológicas diferentes e sucessivas na imagem binária.

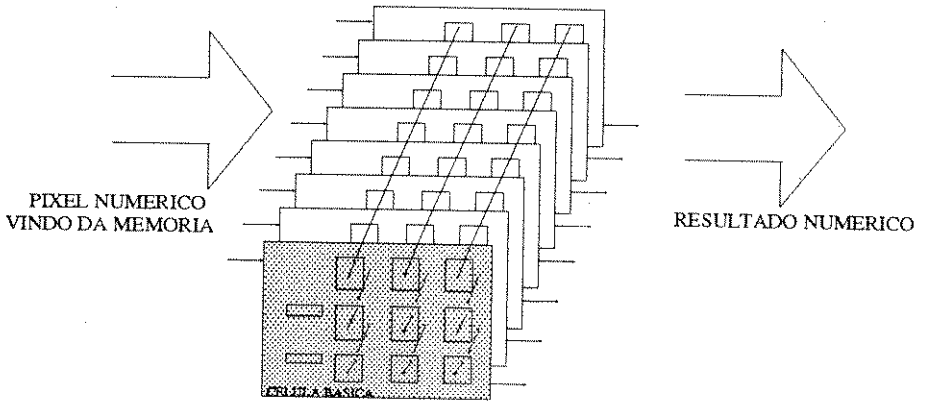

Figura 4.8: configuração das células no modo numérico

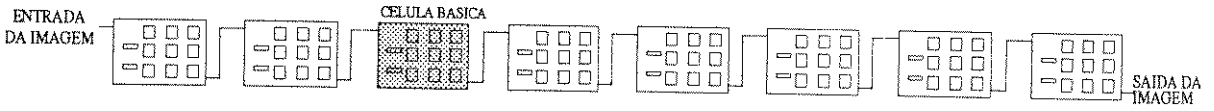

Figura 4.9: configuração das células da arquitetura no modo binário

A arquitetura SAMM apresenta tipos de paralelismo característicos dos processadores de imagens existentes. Distingue-se o paralelismo pixel-bit em que os bits de cada pixel são operados paralelamente, caracterizado naturalmente pela implementação de operações sobre vizinhanças da imagem. Neste caso é necessário ter-se disponível os pixels de uma determinada vizinhança para o cálculo de cada pixel. O paralelismo de operador é representado pela fragmentação da operação a ser implementada em diversas sub-operações sucessivas.

4.3.3 Outras operações associadas à SAMM

Além das clássicas operações morfológicas de dilatação e erosão, a arquitetura pode realizar as seguintes operações morfológicas:

- infimum: $f_1(x) \wedge f_2(x)$

  Para imagens binárias é a operação lógica AND, e para imagens numéricas é a operação de mínimo aplicadas entre duas imagens pixel à pixel.

- supremum: $f_1(x) \vee f_2(x)$

  Para imagens binárias é a operação lógica OR e para imagens numéricas é a operação de máximo aplicadas entre duas imagens pixel à pixel.

- inversão: $k - f(x)$

  Nada mais é do que a subtração entre k e cada pixel da imagem (para imagens binárias $k = 1$ e para imagens numéricas de oito bits $k = 255$).

- subtração: $f_1(x) - f_2(x)$ se $f_2(x) \leq f_1(x)$

  0 em caso contrário

  Tanto no modo binário como no numérico é a subtração entre os pixels correspondentes de duas imagens. Para valores de pixel da imagem subtratora menores que o pixel correspondente da imagem subtraída, o pixel resultado assume valor 0.

- threshold: k se $f_1(x) \leq f_2(x)$

  0 em caso contrário

  Operação também executada pixel à pixel. Sendo o valor do pixel da imagem 2 maior que o correspondente da imagem 1, o pixel resultado assume o máximo valor possível k (imagens binárias $k = 1$, imagens numéricas de oito bits $k = 255$). Caso contrário, o resultado é 0.

- dilatação condicional: $\delta x_B(f) \wedge fx_2(x)$

  É a operação de infimum aplicada a uma imagem dilatada.

- erosão condicional: $\epsilon x_B(f) \vee fx_2(x)$

  É a operação de supremum aplicada a uma imagem erodida.

Com exceção das operações condicionais, que são função das operações de dilatação e erosão, as demais são operações pontuais. Isto significa que a vizinhança do pixel em questão não influi no cálculo do pixel resultado. Para implementar estas operações são necessários, além dos circuitos de máximo e mínimo existentes, circuitos subtrator, inversor

e threshold. Estes circuitos operam tanto no modo binário como no modo numérico, apresentando a mesma versatilidade (partição numérica/binária) de operação que os circuitos de máximo e mínimo descritos na Figura 4.7. Estes circuitos são agregados ao primeiro elemento morfológico da estrutura, que se diferencia dos demais (Figura 4.10) e apresenta uma ligação direta com a saída do processador.

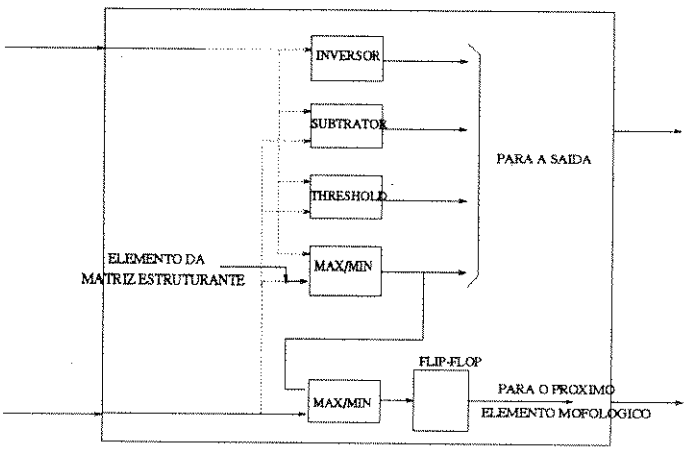

Figura 4.10: primeiro elemento morfológico implementado para as operações pontuais

As operações sobre duas imagens fazem uso da entrada para o elemento morfológico na dilatação/erosão (entrada 2) como entrada para a segunda imagem (Figura 4.10).

Para a realização das operações condicionais, é inserido um circuito de máximo/mínimo no final da estrutura de dilatação/erosão que opera sobre as imagens dilatadas ou erodidas.

4.3.4 Comunicação com a memória

O processador SAMM apresenta duas entradas de imagem - uma para a imagem operando e outra para a imagem operador - e uma saída. Para realizar as operações de inversão, dilatação e erosão simples, é necessário a entrada de apenas uma imagem enquanto as demais operações pontuais trabalham sobre duas imagens. À princípio na operação modo binário, das oito operações realizáveis apenas uma pode ser considerada sobre duas imagens já que o processador aceita apenas duas imagens distintas como entrada.

Há duas possibilidades de acesso ao pixel binário. A primeira é quando este pixel é representado por apenas um bit do byte lido, desconsiderando-se os valores dos demais bits. A outra possibilidade é de compactar-se os pixels binários num byte. O byte lido da memória é então introduzido seqüencialmente no processador, alcançando assim maior eficiência na leitura. A Figura 4.11 ilustra a entrada de dados nos modos numérico e binário.

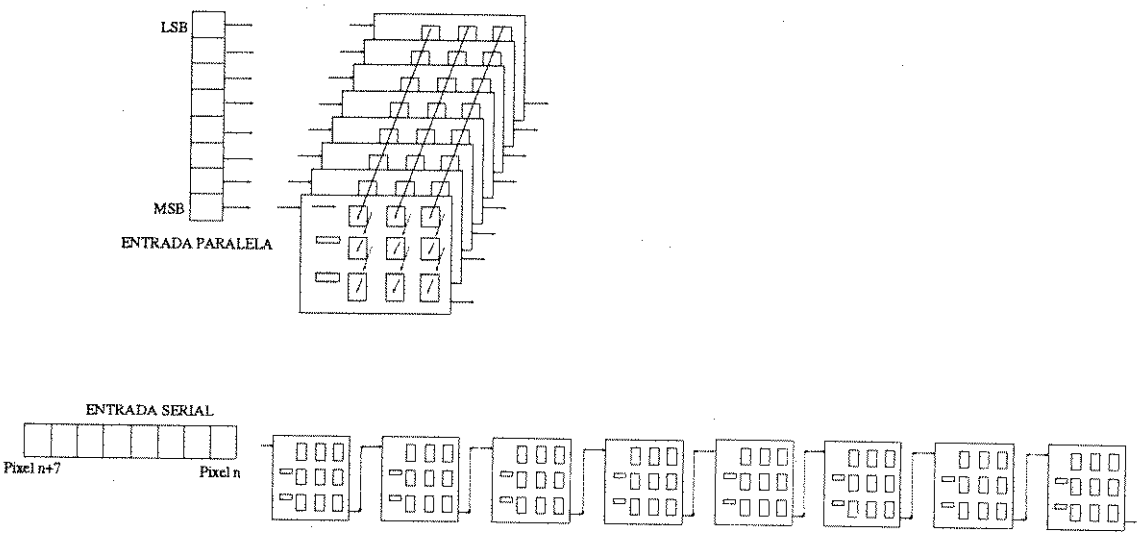

Figura 4.11: entrada dos pixels na forma numérica e na forma binária compactada

A comunicação com a memória é um fator limitante da velocidade de operação dos sistemas. Para amenizar este gargalo, pode-se considerar, por exemplo, a utilização de duas memórias fisicamente distintas, acessadas por barramentos distintos e uma memória contendo a imagem operando e a outra a imagem operador (no caso de operações sobre duas imagens) e a imagem resultado. Deste modo é necessário apenas um ciclo de leitura para operações sobre duas imagens, diminuindo o tempo de espera do processador pelos dados.

4.3.5 Uso de matrizes estruturantes não planares

Inicialmente, a arquitetura SAMM opera sobre matrizes estruturantes planares, ou seja, seus elementos assumem apenas os valores máximo ou mínimo possíveis - 0 ou k (para 8 bits k=255). Para operar sobre matrizes não planares, podendo esses elementos assumir toda a faixa de valores de b-bits, pode ser substituído, no caso das operações numéricas, o circuito superior de mínimo do elemento morfológico por um somador/subtrator diferenci-

ado (Figura 4.12).

Neste caso o somador/subtrator deve apresentar quatro entradas: entrada de seleção da operação (soma/subtração), entrada do pixel da imagem, entrada do elemento da matriz morfológica e entrada para habilitação ou não da operação. Esta entrada de controle indica se o elemento da matriz estruturante apresenta valor 0. Caso isto ocorra, a soma/subtração não é habilitada pois o pixel em questão não deve ser considerado no cálculo do pixel de saída, devendo a saída do circuito ser levada a 0. Para evitar a adição de mais uma entrada no processador, a entrada correspontente do elemento da matriz estruturante (8 bits) tem um de seus bits utilizado como sinal de controle. Portanto, o elemento não planar passa a ter sua faixa de valores limitada em 7 bits (variando entre 0 e 127).

No caso das operações binárias, o mínimo continuará a ser implementado através do AND bit à bit.

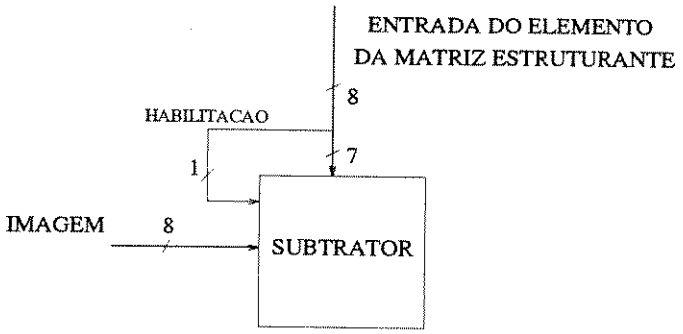

Figura 4.12: subtrator para operação sobre matrizes estruturantes não planares

4.4 Apresentação de outras arquiteturas

4.4.1 Arquitetura "Bit-Plane"

O processador apresentado em [61] foi desenvolvido especialmente para a realização de operações de Morfologia Matemática em PLCA ("Progammable Logic Cell Array"). É uma arquitetura programável, construída a partir de células básicas binárias que permite uma configuração ótima do hardware no tratamento de imagens binárias e

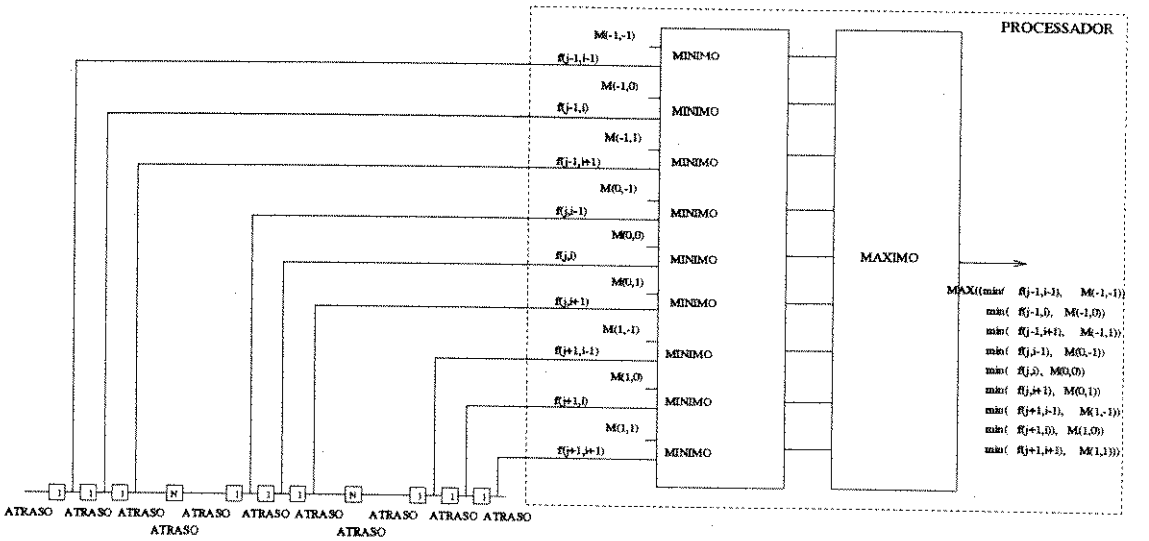

Figura 4.13: arquitetura direta de cada célula básica do processador "bit-plane" para operação com uma matriz estruturante 3×3.

numéricas. Cada célula é um processador de arquitetura direta como o processador da Figura 4.13 que corresponde à aplicação direta das equações 4.6 e 4.8. Nela os pixels a serem operados em cada instante estão presentes na memória interna e são simultaneamente acessados por um bloco de lógica binária que os processa. Para a operação de dilatação, por exemplo, realiza-se em um determinado instante, todos os mínimos envolvidos no cálculo e o máximo entre todos estes mínimos. A Figura 4.13 mostra a estrutura para operação com uma matriz estruturante 3×3.

Cada célula deste processador realiza uma operação morfológica binária completa, várias delas podem ser encadeadas, realizando vários processamentos em pipeline sobre a imagem binária de entrada, como a partição implementada em SAMM. Quando a imagem a ser processada é numérica, as células trabalham paralelamente, cada uma sobre um "bit-plane" da imagem, sendo portanto, uma estrutura "bit-slice".

Cada célula deste processador é um bloco lógico único, não havendo, portanto, modularidade a nível interno da célula.

4.4.2 O processador NPP

O processador NPP (non-linear pipeline processor), desenvolvido para a implementação de filtros não lineares, é um processador sistólico. Sua arquitetura consiste em dois tipos de unidades básicas de cálculo (unidade de dilatação e unidade de erosão - Figura 4.14), concebidas a partir das equações 4.20, 4.19, que definem as operações de erosão e de dilatação numéricas não planares [52].

$$(f \oplus g)(x) = max\{f(x-z) + g(z)\} \forall z \in G, x - z \in F \tag{4.19}$$

$$(f \ominus g)(x) = min\{f(x+z) - g(z)\} \forall z \in G, x + z \in F \tag{4.20}$$

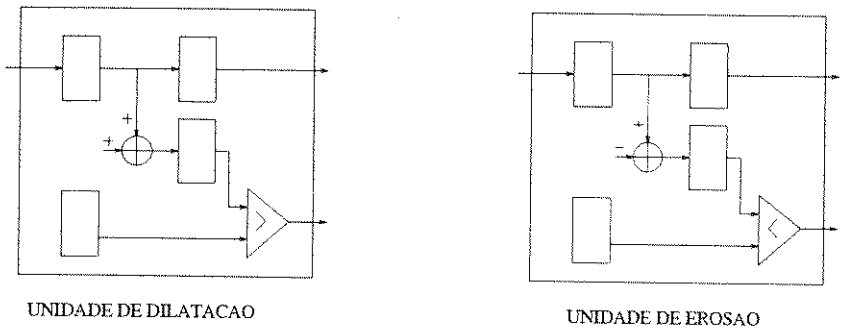

Figura 4.14: unidades básicas do processador NPP

Unidades de dilatação (erosão) são compostas basicamente por quatro registradores de deslocamento de b-bits, somador (subtrator) e comparador e quando encadeadas implementam as respectivas operações. Estas unidades podem ser transformadas em unidades binárias, bastando substituir-se o somador (subtrator) por uma porta AND (OR) e o comparador por uma porta OR (AND). A comunicação entre unidades é local, simples, regular e síncrona. O fluxo dos dados através da estrutura é também síncrono e regular.

4.4.3 Decomposição threshold

Outra possibilidade na implementação de operações morfológicas é a aplicação da decomposição "threshold" nas operações morfológicas numéricas [66], [67]. Através desta

propriedade é possível decompor sinais numéricos (b-bits) em múltiplos sinais binários (b) os quais são tratados paralelamente. Apesar de sua lógica combinacional diferente, esta arquitetura se assemelha à da Figura 4.13.

4.5 Comparações e discussões

O princípio de partição binária/numérica considerada na arquitetura SAMM é o utilizado na arquitetura "bit-plane" [61]. Em ambas, o hardware que implementa a operação numérica é fragmentado em "fatias" binárias, operando interligadamente ou independentemente sobre imagens numéricas ou binárias, respectivamente. O dado numérico não necessita sofrer nenhum processamento anterior ou posterior ao seu tratamento, bastando replicar o número de células básicas conforme o tamanho do pixel e entregar um dos seus bits a uma delas. Neste sentido, aproveita-se integralmente o hardware do modo numérico no processamento binário através do encadeamento de várias operações binárias.

Fazendo uso da decomposição threshold, uma vez decomposto o pixel numérico em módulos binários, estes são tratados paralela e independentemente. Somente após o final do processamento os dados interagem, constituindo novamente o pixel numérico (resultado). Neste caso, portanto, há necessidade de um particionamento do pixel numérico e de sua posterior recomposição. O hardware que implementa a decomposição threshold não é modular, sendo necessárias grandes modificações quando da variação do tamanho do pixel a ser processado, o que gera uma estrutura bastante irregular.

Na arquitetura SAMM há modularidade em dois níveis. O primeiro está relacionado com a estrutura interna da célula básica, composta por elementos morfológicos binários (módulos) encadeados, que são partes da memória e do processador. O outro nível de modularidade se refere à estrutura de mais alto nível da arquitetura, que é composta por células básicas que se combinam diferenciadamente conforme o modo de operação (numérico ou binário). A arquitetura "bit-plane" não apresenta o primeiro tipo de modularidade, sendo sua célula básica uma implementação da arquitetura direta. Neste caso há distinção entre unidade de processamento e memória de pixels.

Estruturalmente, a arquitetura SAMM é semelhante à arquitetura NPP [52],

apresentando vantagens importantes. Ambas são arquiteturas modulares e flexíveis. Seu tamanho é diretamente proporcional à dimensão do elemento estruturante e independente do tamanho da imagem a ser processada, podendo formar estruturas de operação unidimensionais e bidimensionais com a facilidade proporcionada por um sistema modular. Em ambas, o processador e a memória encontram-se presentes em cada elemento morfológico (SAMM) e unidade básica (NPP), ou seja, não são arquiteturas de implementação direta como "bit-plane".

No sincronismo dos dados, a unidade básica da arquitetura NPP necessita de mais "flip-flops" que o elemento morfológico da SAMM, devido ao modo de propagação dos pixels vindos da memória. Em NPP os pixels são entregues à próxima unidade básica somente após passarem por dois atrasos, ficando temporariamente armazenados no seu interior. Isto possibilita a realização paralela sobre dados distintos das duas operações implementadas pela unidade básica, duplicando assim a velocidade com relação à SAMM. Na arquitetura SAMM, o atraso unitário ("flip-flop") presente em cada elemento morfológico, para o sincronismo do fluxo de dados, garante o armazenamento temporário do resultado parcial após o cálculo dos dois circuitos (dois mínimos - erosão - ou um mínimo e um máximo - dilatação). Apesar de uma velocidade de processamento menor (equações 4.21, 4.22), a arquitetura SAMM apresenta uma menor latência dos resultados. Considerando:

$T_{M/m}$ - tempo de realização do máximo/mínimo

T_a - tempo de realização da adição

T_{df} - atraso do "flip-flop"

T_{em} - perído mínimo do processador

tem-se para SAMM:

$$T_{em} = 2T_{M/m} + T_{df} \tag{4.21}$$

e para NPP:

$$T_{em} = T_a + T_{df} \tag{4.22}$$

Outro ponto distinto refere-se à formação de estruturas que operam com uma matriz estruturante com número de colunas maior que 1. Em NPP é necessário, ao final de cada linha da estrutura, uma lógica adicional para combinar os resultados parciais gerados em cada linha e obter o resultado final. Em SAMM ocorre a passagem do resultado parcial de uma linha superior para uma entrada no início da linha seguinte, que vai sendo agregado, ao longo da estrutura, a outros resultados parciais. Isto se dá sucessivamente entre uma linha e outra, sendo o resultado final gerado gradativa e acumulativamente.

O projeto da arquitetura NPP não enfoca o problema da implementação de operações binárias e numéricas na mesma arquitetura, ponto importante na arquitetura SAMM. Entretanto NPP considera operações morfológicas através de matrizes estruturantes planares e não planares. Até o momento, a arquitetura SAMM considera somente as matrizes estruturantes planares, que são os casos mais usuais na prática [54].

Na implementação da decomposição threshold os resultados são obtidos paralelamente, atingindo grande velocidade de processamento. Entretanto, o hardware que a implementa é bastante complexo e não modular. Nos processadores SAMM e NPP a complexidade do hardware cresce linearmente com o número de bits. O hardware que executa uma decomposição threshold apresenta um crescimento exponencial da complexidade com relação ao número de bits (tabela 4.3).

Considerando:

B - número de portas de cada elemento morfológico = 48,

n, m - dimensões da matriz estruturante,

M, N - dimensões da imagem de entrada,

b - número de bits,

F - número de portas em cada "flip-flop",

N_B - número de portas dos blocos da estrutura,

N_S - número de portas dos registradores de deslocamento,

N_R - número de portas dos registradores do elemento da matriz estruturante em cada bloco básico,

N_{tot} - número total de portas,

Para a arquitetura SAMM temos a complexidade do hardware calculada por:

$$N_B = Bmnb \tag{4.23}$$

$$N_S = (N + n)(m - 1)bF \tag{4.24}$$

$$N_R = 3mnb \tag{4.25}$$

$$N_{tot} = N_B + N_S + N_R \tag{4.26}$$

Visando estabelecer uma comparação coerente entre as arquiteturas no cálculo dos dados da tabela 4.3, não foram incluídos o número de portas dos registradores de deslocamento nem dos registradores do elemento da matriz estruturante. Os dados referentes à NPP e à decomposição threshold foram extraídos de [52] e se referem a estruturas que operam com matrizes planares e não planares. Estima-se que SAMM continue a apresentar o menor número de portas, mesmo quando implementada para operações não-planares.

ARQUITETURA	TAMANHO DA MATRIZ ESTRUT.	NUMERO DE BITS		
		1-bit	4-bits	8-bits
THRESHOLD	3x3		1214	326.7K
	4x4		2054	555.1K
	5x5		3134	848.9K
NPP	3x3		2205	4401
	4x4		3920	7824
	5x5		6125	12.2K
SAMM	3x3	432	1728	3456
	4x4	768	3072	6144
	5x5	1200	4800	9600

Tabela 4.3: Comparação da complexidade do hardware para operações de dilatação/erosão

Capítulo 5

Simulações e Resultados

5.1 Modelagem da arquitetura

A modelagem da arquitetura foi realizado através da linguagem VHDL, simulada no ambiente da Mentor Graphics.

O objetivo deste trabalho foi a descrição e a validação de uma estrutura, formada por blocos básicos encadeados sistolicamente, que implementa as operações de dilatação e erosão. Particionando esta arquitetura em blocos, foi feita a descrição de cada uma destes através de uma *arquitetura* ligada a uma *entidade*. Estes blocos descritos foram agrupados em uma representação esquemática simulável, formando a arquitetura completa. Como foi utilizada a metodologia "top-down", a próxima etapa de projeto seria a descrição a nível mais detalhado de cada bloco componente e, finalmente, sua descrição através de portas lógicas para a implementação física.

As operações de dilatação e erosão, dependentes do valor dos pixels da vizinhança em questão, têm seu resultado gerado através de diversos blocos encadeados. As operações pontuais dependem apenas do pixel em questão, não sendo necessário, portanto, ser fragmentada em partes sendo implementada por somente um bloco de circuito o qual foi agregado ao primeiro bloco da estrutura.

5.2 Simulação da arquitetura

Os blocos descritos através de *entidades* e *arquiteturas* foram simulados e validados separadamente antes de sua junção para compor a arquitetura SAMM. A descrição VHDL deles se encontra no Apêndice A. O teste de cada *entidade-arquitetura* foi realizado através da aplicação de estímulos nos portes de entrada e da monitoração do comportamento dos sinais nos portes de saída, avaliando-se assim o comportamento interno do modelo. Os vetores de teste foram descritos na forma de programas (arquivos .do) aplicados sobre as entidades. Estes arquivos de teste são usados a fim de facilitar a interação entre ferramenta e usuário e se encontram no Apêndice B.

A simulação visa validar o funcionamento dos blocos para uma posterior validação da funcionalidade da arquitetura SAMM. Não estando em questão detalhes referentes à sua implementação física. Não se chegou ao nível de descrição lógica dos blocos componentes da arquitetura, sendo as operações realizadas por eles, independente dos circuitos que as implementam, e conseqüentemente, de seu tempo de resposta e tecnologia utilizada. O CLOCK aplicado nas simulações foi meramente um sincronizador, e seu valor de freqüência arbitrário. No Apêndice C são mostradas partes das simulações extraídas do ambiente simulador.

5.2.1 Blocos constituintes da arquitetura

A arquitetura SAMM mostrada na Figura 5.1 foi particionada em blocos, simulados e validados separadamente, os quais foram então compostos via descrição esquemática na estrutura completa (mostrada na Figura 5.1). Os blocos formadores da arquitetura são os seguintes:

- bloco_beh: descreve a parte superior do primeiro elemento morfológico e as operações por ele implementadas. Este elemento se diferencia dos demais por também implementar as operações de não vizinhança. Agrupa os modos de operação binário e numérico em si, sendo que no modo binário implementa a partição em um circuito "bit-slice".

- bloco_sup: descreve a parte superior dos demais elementos morfológicos e a operação de mínimo por ele implementada, tanto no modo numérico como no modo binário (partição "bit-slice").
- bloco_spl: descreve a parte inferior de todos os elementos morfológicos e as operações implementadas de máximo e mínimo nos modos numérico/binário (partição "bit-slice").
- flipflop: descreve os flip-flips presentes na saída de cada elemento morfológico.
- registrador: descreve os registradores que armazenam os elementos da matriz estruturante das operações de vizinhança.
- chave2: descreve um multiplexador 2x1 com um sinal de seleção de oito bits. Usada para direcionar corretamente o fluxo dos dados conforme alguns modos de operação.
- chave_spl2: descreve um multiplexador 2x1 com um sinal de seleção de um bit, também usado para direcionamento do fluxo de dados.

No apêndice A se encontram os arquivos vhdl de descrição de cada um destes blocos. No apêndice B estão os arquivos .do utilizados para a aplicação automática dos estímulos durante a simulação.

5.2.2 Arquitetura

Os portes de entrada e saída desta arquitetura são:

- clock: entrada do relógio
- in_1(8 : 1), in_2(8 : 1): entradas vindas da memória da imagem. Pela in_2 são introduzidos nos registradores os valores dos elementos da matriz estruturante ou a imagem operadora (segunda imagem).
- saída(8 : 1): saída da imagem resultado para a memória.
- ld_reg1 – ld_reg9: sinais de habilitação de escrita em cada um dos registradores ligados a cada elemento morfológico.

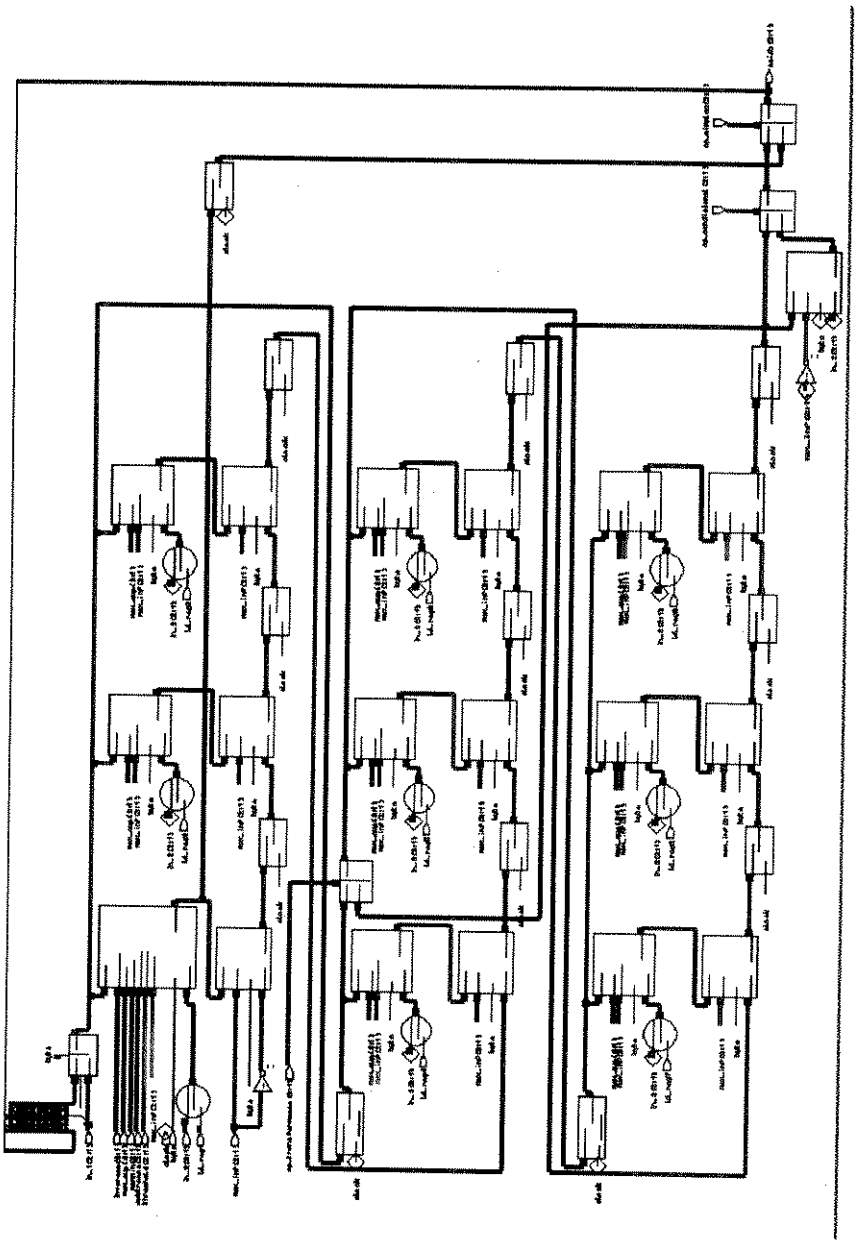

Figura 5.1: arquitetura de processamento de imagens implementada no Quicksim

- max_sup(8 : 1), max_inf(8 : 1): sinais de controle indicando operação de máximo ou mínimo aos blocos superiores e inferiores de cada elemento morfológico.
- inversão(8 : 1), subtração(8 : 1), maxmin(8 : 1), threshold(8 : 1): sinais de controle habilitando a inversão, subtração, máximo/mínimo e threshold no bloco superior do primeiro elemento morfológico.
- byte: sinal de controle que indica o modo binário ou numérico de operação.
- op_condicional(8 : 1): sinal de controle que habilita o fluxo de resultados das operações condicionais para a saída (dilatação e erosão condicional).
- op_simples(8 : 1): sinal de controle que habilita o fluxo de resultados das operações simples para a saída (subtração, inversão, máximo, mínimo, threshold).

Para cada operação testada na arquitetura foi aplicada uma seqüência de arquivos .do (apêndice B), gerando os sinais nos portes de entrada necessários para a observação, através dos portes de saída, do funcionamento da estrutura.

Para a validação dos resultados da simulação, as operações foram implementadas através da caixa de ferramentas de Morfologia Matemática do Khoros [62] e estes resultados comparados com os da simulação VHDL.

Assim como na modelagem, não foram implementados aspectos referentes a implementação física da arquitetura e, conseqüentemente, na simulação não foram observados aspectos tais como tempo de resposta da estrutura. Observou-se simplesmente se os resultados obtidos estavam corretos e sincronizados com o cadenciador do sistema (o sinal de CLOCK), comprovando desta forma o funcionamento da arquitetura. No apêndice C encontram-se os resultados das simulações extraídos do simulador Quicksim. Em [20] é tratado detalhadamente a modelagem e a simulação desta arquitetura.

Capítulo 6

Conclusão

Neste trabalho foi desenvolvida uma arquitetura para a implementação de operações de Morfologia Matemática, enfocando as operações básicas de dilatação e erosão. Esta arquitetura trabalha sobre imagens binárias e em nível de cinza, aproveitando integralmente o mesmo hardware. Isto foi alcançado através da decomposição do hardware em "fatias" binárias ("bit-slice") e sua operação conjunta ou dissociada conforme o modo numérico ou binário de operação, respectivamente. Desta forma o hardware utilizado para a realização de um operação morfológica sobre uma imagem numérica é utilizado para realizar várias operações binárias sucessivamente encadeadas.

A descrição desta arquitetura, feita através da linguagem de descrição de hardware VHDL, foi utilizada para sua simulação e validação funcional. As maiores dificuldades encontradas neste sentido, não foram na utilização da linguagem VHDL, mas sim na utilização da ferramenta que a implementa. Pelo fato de ser um software complexo de última geração, apresenta ainda diversos problemas de instalação e execução. Como perspectivas futuras vê-se a disseminação de descrições de hardware através de linguagens, das quais VHDL apresenta-se como um padrão emergente. É uma linguagem consistente, fortemente tipada, capaz de caracterizar adequadamente conceitos de hardware. A modelagem analógica, a síntese automática desde o nível de especificação, o aproveitamento da linguagem em testes e o uso da totalidade da linguagem pelas ferramentas são pontos a serem explorados futuramente [11].

Comparando-se a arquitetura SAMM com outras implementadas com a mesma finalidade, ressaltam-se alguns aspectos. SAMM agrega, na mesma arquitetura, características de importantes de implementação tais como utilização integral do hardware em operações numéricas e binárias, alta modularidade, regularidade e baixa complexidade, que a tornam bem adaptada à implementação em VLSI. A operação sobre matrizes estruturantes não planares é uma sugestão para trabalhos futuros.

Ainda como extensão da funcionalidade do processador, podemos considerar a possibilidade de implementação de operações recursivas na estrutura de base da SAMM. Podemos propor uma investigação sobre a introdução de outros operadores de filtragem bastante utilizados em processamento de imagens e formalizados a partir dos conceitos de Morfologia Matemática, tais como o filtro da mediana, filtro da ordem, filtro dos k-vizinhos mais próximos, etc. pode ser considerada. A adequação das descrições VHDL da arquitetura aqui desenvolvida para a síntese automática do hardware também é uma sugestão para trabalhos futuros.

Apêndice A

Descrições VHDL

A seguir são mostrados os arquivos VHDL de descrição dos blocos formadores da arquitetura

```
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;

PACKAGE pacote IS

        CONSTANT Taa: TIME:= 100ns; -- Address access time (max)
        CONSTANT Toh: TIME:= 10ns; -- Output hold from address change  (min)
        CONSTANT Tdw: TIME:= 30ns; -- Data Hold Time
        CONSTANT Twc: TIME:=100ns;  -- Write cycle time (min)
        CONSTANT Trc: TIME:=100ns;  -- Read cycle time (min)
        CONSTANT clock_width: TIME:= 50ns;
        CONSTANT plano: INTEGER:=7;

        CONSTANT inv_delay: TIME:=10ns;
        CONSTANT max_delay: TIME:=64ns;
        CONSTANT subtr_delay: TIME:=64ns;
        CONSTANT thresh_delay: TIME:=74ns;
        CONSTANT porta1_delay: TIME:=4ns;
        CONSTANT porta2_delay: TIME:=8ns;
        CONSTANT ff_delay: TIME:=4ns;
        CONSTANT reg_delay: TIME:=4ns;

        TYPE ar IS ARRAY (7 DOWNTO 0) OF BIT;
        TYPE arr IS ARRAY (8 DOWNTO 0) OF BIT;

        TYPE qsim_state IS ('0','1','Z');

        SUBTYPE vec5 IS QSIM_STATE_VECTOR(5 DOWNTO 1);
        SUBTYPE vec4 IS QSIM_STATE_VECTOR(4 DOWNTO 1);
        SUBTYPE word IS QSIM_STATE_VECTOR(8 DOWNTO 1);
        SUBTYPE resword IS  QSIM_STATE_RESOLVED_X_VECTOR(8 DOWNTO 1);
        SUBTYPE add IS QSIM_STATE_VECTOR(10 DOWNTO 0);
        SUBTYPE resbit IS QSIM_STATE_RESOLVED_X_VECTOR(0 DOWNTO 0);

        FUNCTION intval(val:word) RETURN INTEGER;
        FUNCTION vecval(int:INTEGER) RETURN word;

END pacote;

PACKAGE BODY pacote IS
        FUNCTION intval (val:word)
                RETURN INTEGER IS
                VARIABLE sum:INTEGER:=0;
        BEGIN
                FOR n IN val'LOW TO val'HIGH
                LOOP
                IF val(n)='1' THEN
                sum:=sum+(2**n);
                END IF;
                END LOOP;
        RETURN sum;
        END intval;

        FUNCTION vecval (int:INTEGER)
                RETURN  QSIM_STATE_VECTOR IS
                VARIABLE vec:word;
                VARIABLE v:INTEGER;
        BEGIN
                v:=int;
                FOR i IN 0 TO 7
                LOOP
                        IF ((v REM 2)=1) THEN
                                vec(i):='1';
                        ELSE
                                vec(i):='0';
                        END IF;
                        v:=v/2;
                END LOOP;
        RETURN vec;
        END vecval;


        FUNCTION maximo_bit (i1,i2:BIT)
                RETURN BIT IS
                VARIABLE maxbit: BIT;
        BEGIN
                IF (i1>=i2) THEN
                        maxbit:=i1;
                ELSE
                        maxbit:=i2;
                END IF;
        RETURN maxbit;
        END maximo_bit;


        FUNCTION maximo_byte (i1,i2: resword)
                RETURN resword IS
                VARIABLE maxbyte: resword;
        BEGIN
                IF (i1>=i2) THEN
                        maxbyte:=i1;
                ELSE
                        maxbyte:=i2;
                END IF;
        RETURN maxbyte;
        END maximo_byte;

END pacote;
```

```
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;

USE work.pacote.ALL;

ENTITY blocosup_beh IS
        PORT (in_1:IN  resword; --resword;
                       in_2:IN word;    --word;
                       byte:IN BIT;
                       Max:IN word;
                       Max_inf: IN word;
                       saida: OUT word);
END blocosup_beh;

ARCHITECTURE b0 OF blocosup_beh IS
BEGIN
PROCESS (in_1,in_2,byte,Max)

BEGIN

IF (byte='1') THEN

        --maximo numerico
        IF (Max="11111111") THEN
                IF (word(in_1) >= in_2) THEN
                        saida<= word(in_1);
                ELSE
                        saida<= in_2;
                END IF;

        --minimo numerico
        ELSIF (Max="00000000" ) THEN

                --op.erosao e el. da matriz estrut.= 0
                --(imagem nao deve ser considerado)
                IF (in_2="00000000" AND Max_inf="00000000") THEN
                saida<="11111111";
                ELSIF (word(in_1) <= in_2) THEN
                        saida<= word(in_1);
                ELSE
                        saida<= in_2;
                END IF;
        END IF;

ELSE    --imagem binaria

        FOR i IN 8 DOWNTO  1 LOOP

        --maximo binario
        IF (Max(i)='1') THEN
                IF (in_1(i) >= in_2(i)) THEN
                        saida(i)<= in_1(i);
                ELSE
                        saida(i)<= in_2(i);
                END IF;

        --minimo binario
        ELSIF (Max(i)='0') THEN
                IF (in_2(i)='0' AND Max_inf(i)='0') THEN
                        saida(i)<='1';
                ELSIF (in_1(i) <= in_2(i)) THEN
                        saida(i)<= in_1(i);
                ELSE
                        saida(i)<= in_2(i);
                END IF;

        END IF;
        END LOOP;
END IF;

END PROCESS;
END b0;
```

```
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;

USE work.pacote.ALL;

ENTITY blocospl_beh IS
      PORT (in_1:IN resword;
                  in_2:IN word;
                  byte:IN BIT;
                  Max:IN word;
                  saida: OUT word);
END blocospl_beh;

ARCHITECTURE b0 OF blocospl_beh IS
BEGIN
PROCESS (in_1,in_2,byte,Max)

BEGIN

IF (byte='1') THEN

        --maximo numerico
        IF (Max="11111111") THEN
                IF (in_2="XXXXXXXX") THEN
                        saida<=word(in_1);
                ELSIF (in_1="XXXXXXXX") THEN
                        saida<=word(in_2);
                ELSIF (word(in_1) >= in_2) THEN
                        saida<= word(in_1);
                ELSE
                        saida<= in_2;
                END IF;

        --minimo numerico
        ELSIF (Max="00000000") THEN
                IF (in_2="XXXXXXXX") THEN
                        saida<=word(in_1);
                ELSIF (in_1="XXXXXXXX") THEN
                        saida<=word(in_2);
                ELSIF (word(in_1) <= in_2) THEN
                        saida<= word(in_1);
                ELSE
                        saida<= in_2;
                END IF;
        END IF;
ELSE

        FOR i IN 8 DOWNTO  1 LOOP

        --maximo binario
        IF (Max(i)='1') THEN
                IF (in_2="X") THEN
                        saida(i)<= '0';
                ELSIF (in_1="X") THEN
                        saida(i)<= '0';
                ELSIF (in_1(i) >= in_2(i)) THEN
                        saida(i)<= in_1(i);
                ELSE
                        saida(i)<= in_2(i);
                END IF;

        --minimo binario
        ELSIF (Max(i)='0') THEN
                IF (in_2="X") THEN
                        saida(i)<= '0';
                ELSIF (in_1="X") THEN
                        saida(i)<= '0';
                ELSIF (in_1(i) <= in_2(i)) THEN
                        saida(i)<= in_1(i);
                ELSE
                        saida(i)<= in_2(i);
                END IF;

        END IF;
        END LOOP;
END IF;

END PROCESS;
END b0;
```

```
--testado (25-11)

LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;

USE work.pacote.ALL;

ENTITY bloco_beh IS
        PORT (in_1:IN resword;
                        in_2:IN word;
                        byte:IN BIT;
                        Max:IN word;
                        Max_inf: IN word;
                        threshold,inversao,maxmin,subtracao: IN word;
                        saida: OUT word);
END bloco_beh;

ARCHITECTURE b0 OF bloco_beh IS
BEGIN
PROCESS (in_1,in_2,byte,Max,threshold,inversao,maxmin,subtracao)

BEGIN

IF (byte='1') THEN

        --subtracao numerica
        IF (subtracao="11111111") THEN
                IF (word(in_1) < in_2) THEN
                saida<="00000000" AFTER 0ns;
                ELSE
                saida<=word(in_1)-in_2 AFTER 0ns;
                END IF;

        --inversao numerica
        ELSIF (inversao="11111111") THEN
                saida<=NOT(word(in_1));

        --maximo numerico
        ELSIF (maxmin="11111111" AND Max="11111111") THEN
                IF (word(in_1) >= in_2) THEN
                        saida<= word(in_1);
                ELSE
                        saida<= in_2;
                END IF;

        --minimo numerico
        ELSIF (maxmin="11111111" AND Max="00000000") THEN

                --op.erosao e el.da matriz estr.=0 (imagem
                --nao deve ser considerado)
                IF (in_2="00000000" AND Max_inf="00000000") THEN
                        saida<="11111111";
                ELSIF (word(in_1) <= in_2) THEN
                        saida<= word(in_1);
                ELSE
                        saida<= in_2;
                END IF;

        --threshold numerico
        ELSIF (threshold="11111111") THEN
                IF (word(in_1) <= in_2) THEN
                        saida<= "11111111";
                ELSE
                        saida<= "00000000";
                END IF;

        END IF;

ELSE
        FOR i IN 8 DOWNTO 1 LOOP

        --subtracao binaria
        IF (subtracao(i)='1') THEN
                IF (in_1(i) < in_2(i)) THEN
                saida(i)<='0';
                ELSE
                saida(i)<=in_1(i)-in_2(i);
                END IF;

        --inversao binaria
        ELSIF (inversao(i)='1') THEN
                saida(i)<=NOT(in_1(i));

        --maximo binario
        ELSIF (maxmin(i)='1' AND Max(i)='1') THEN
                IF (in_1(i) >= in_2(i)) THEN
                        saida(i)<= in_1(i);
                ELSE
                        saida(i)<= in_2(i);
                END IF;

        --minimo binario
        ELSIF (maxmin(i)='1' AND Max(i)='0') THEN
                IF (in_2(i)='0' AND Max_inf(i)='0') THEN
                        saida(i)<='1';
                ELSIF (in_1(i) <= in_2(i)) THEN
                        saida(i)<= in_1(i);
                ELSE
                        saida(i)<= in_2(i);
                END IF;

        --threshold binario
        ELSIF (threshold(i)='1') THEN
                IF (in_1(i) <= in_2(i)) THEN
                        saida(i)<= '1';
                ELSE
                        saida(i)<= '0';
                END IF;
        END IF;
        END LOOP;
END IF;
END PROCESS;
END b0;
```

```
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;

USE work.pacote.ALL;

ENTITY registrador IS
        PORT (entrada:IN resword;
                        saida:OUT word;
                        load:IN QSIM_STATE);
END registrador;

ARCHITECTURE r0 OF registrador IS
BEGIN

opera:PROCESS(load,entrada)
        VARIABLE dado:word;
BEGIN

IF(load='1') THEN
   dado:=word(entrada);
        saida<=word(dado) AFTER reg_delay;

ELSE
        saida<=dado;

END IF;
END PROCESS opera;

END r0;
```

```
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;

USE work.pacote.ALL;

ENTITY flipflop IS
       PORT (entrada: IN word;
                      saida:OUT word;
                      clock:IN QSIM_STATE);
END flipflop;

ARCHITECTURE f0 OF flipflop IS
BEGIN

opera:PROCESS(clock,entrada)
                VARIABLE dado1:word;
                VARIABLE dado2:word;
               BEGIN

               --armazena novo dado e saida dado antigo
               IF (clock='0') THEN
                       dado1:=entrada;
                       saida<=dado2;

               --saida dado armazenado
               ELSIF (clock='1') THEN
                       dado2:=dado1;
                       saida<=dado2 AFTER ff_delay;

               END IF;
END PROCESS opera;

END f0;
```

```
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;

USE work.pacote.ALL;

ENTITY chave_spl2 IS
        PORT (selecao: IN BIT;
                       i1,i2: IN QSIM_STATE_VECTOR (8 DOWNTO 1);
                       o: OUT QSIM_STATE_VECTOR (8 DOWNTO 1));
END chave_spl2;

ARCHITECTURE c0 OF chave_spl2 IS
        BEGIN

PROCESS (i1,i2,selecao)
BEGIN

CASE selecao IS
        WHEN '0'=>
                o<=i1;
        WHEN '1'=>
                o<=i2;
        WHEN OTHERS=>
                o<="ZZZZZZZZ";
END CASE;

END PROCESS;
END c0;
```

```
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;

USE work.pacote.ALL;

ENTITY chave2 IS
      PORT (selecao: IN  QSIM_STATE_VECTOR (8 DOWNTO 1);
                    i1,i2: IN QSIM_STATE_VECTOR (8 DOWNTO 1);
                    o: OUT QSIM_STATE_VECTOR (8 DOWNTO 1));
END chave2;

ARCHITECTURE c0 OF chave2  IS
      BEGIN

PROCESS (i1,i2,selecao)
BEGIN

FOR i IN 8 DOWNTO 1 LOOP
CASE selecao(i) IS
      WHEN '0'=>
            o(i)<=i1(i);
      WHEN '1'=>
            o(i)<=i2(i);
      WHEN OTHERS=>
            o(i)<='Z';
END CASE;
END LOOP;

END PROCESS;
END c0;
```

Apêndice B

Arquivos de Estímulos .DO

A seguir encontram-se os arquivos de descrição dos estímulos aplicados sobre os blocos formadores da arquitetura (nome_do_bloco.do), os arquivos .DO das operações sobre a arquitetura e o arquivo Package (pacote).

Para operar sobre as imagens em nível de cinza tem-se a seguinte sequência de arquivos .DO:

- para as operações de dilatação e erosão: 1num.DO, 2num.DO, cinza1.DO
- para as operações de dilatação e erosão condicionais: 1num.DO, 2num.DO, cinza1-2def.DO
- para as operações de não vizinhança: clock.DO, 2num.DO, cinza1-2spl.DO

Para as imagens binárias tem-se:

- para as operações de dilatação e erosão: 1bin.DO, 2bin.DO, bin1.DO
- para as operações de dilatação e erosão condicionais: 1bin.DO, 2bin.DO, bin1-2def.DO
- para as operações de não vizinhança: clock.DO, 2bin.DO, bin1-2.DO

```
// seta periodo do clock
        SET clock Period 100;
        FORCE clock   '1'   0 -Repeat;
        FORCE clock   '0'  50 -Repeat;
        RUN 100;

// OPERACOES BYTE
// calculo do maximo
        FORCE in_1 11;
        FORCE in_2 30;
        FORCE byte 1;
        FORCE Max 255;
        RUN 50;

        FORCE in_2 0;
        RUN 50


// calculo do minimo
        FORCE Max 0;
        FORCE Max_inf 255;
        RUN 100;

        FORCE Max_inf 0;
        RUN 100;

// OPERACOES BIT
// calculo de maximo (bits 8 e 5-1) // e minimo (bits 7 e 6)
        FORCE byte 0;
        FORCE Max_inf 255;
        FORCE in_1 170;
        FORCE in_2 163;
        FORCE Max 159;
        RUN 100;

// acerto da erosao (minimo no bloco sup.( bits 7 e 6) e minino no inf.(bit 7)
        FORCE Max_inf 191;
        RUN 100;
```

```
// seta periodo do clock
        SET clock Period 100;
        FORCE clock    '1'    0 -Repeat;
        FORCE clock    '0'   50 -Repeat;
        RUN 100;

// OPERACOES BYTE
// calculo do maximo
        FORCE in_1 11;
        FORCE in_2 30;
        FORCE byte 1;
        FORCE Max 255;
        RUN 100;

// calculo do minimo
        FORCE Max 0;
        RUN 100;

        FORCE in_1 30;
        FORCE in_2 11;
        RUN 100;

// OPERACOES BIT
// calculo de maximo (bits 8 e 5-1) // e minimo (bits 7 e 6)
        FORCE byte 0;
        FORCE in_1 170;
        FORCE in_2 227;
        FORCE Max 159;
        RUN 100;
```

```
// seta periodo do clock
        SET clock Period 100;
        FORCE clock    '1'   0 -Repeat;
        FORCE clock    '0'  50 -Repeat;
        RUN 100;

// OPERACOES BYTE
// calculo do maximo
        FORCE in_1 11;
        FORCE in_2 30;
        FORCE byte 1;
        FORCE Max 255;
        FORCE maxmin 255;
        RUN 100;

// calculo do minimo
        FORCE Max 0;
        RUN 100;

// calculo do threshold
        FORCE Max 1;
        FORCE maxmin 0;
        FORCE threshold 255;
        RUN 100;

        FORCE in_1 30;
        FORCE in_2 11;
        RUN 100;

// calculo da inversao
        FORCE threshold 0;
        FORCE inversao 255;
        RUN 100;

// calculo da subtracao
        FORCE inversao 0;
        FORCE subtracao 255;
        RUN 100;

        FORCE in_1 11;
        FORCE in_2 30;
        RUN 100;

// OPERACOES BIT
// calculo de maximo (bits 8 e 5) // e minimo (bits 7 e 6)
// threshold (bits 4 e 3)
// inversao (bits 2 e 1)

        FORCE in_1 170;
        FORCE in_2 227;

        FORCE byte 0;
        FORCE subtracao 0;
        FORCE Max 159;
        FORCE maxmin 240;
        FORCE threshold 12;
        FORCE inversao 3;
        RUN 100;
```

```
// calculo da subtracao
        FORCE inversao 0;
        FORCE subtracao 3;
        RUN 100;

        FORCE in_1 11;
        FORCE in_2 30;
        RUN 100;
```

```
FORCE entrada FF;
RUN 100;

FORCE entrada 10;
FORCE load 1;
RUN 100;

FORCE load 0;
FORCE entrada 20;
RUN 100;
```

```
FORCE entrada 22;
SET clock Period 1 00;
FORCe clock    '1'    0 -Repeat;
FORCe clock    '0'   50 -Repeat;
RUN 100;

FORCE entrada 11;
RUN 100;

FORCE entrada 72;
RUN 100;
```

Apêndice C

Simulações VHDL

A seguir são mostras os resultados das simulações VHDL das operações de morfologia matemática binárias e numéricas na arquitetura tratada, extraídos do ambiente Quicksim de simulação.

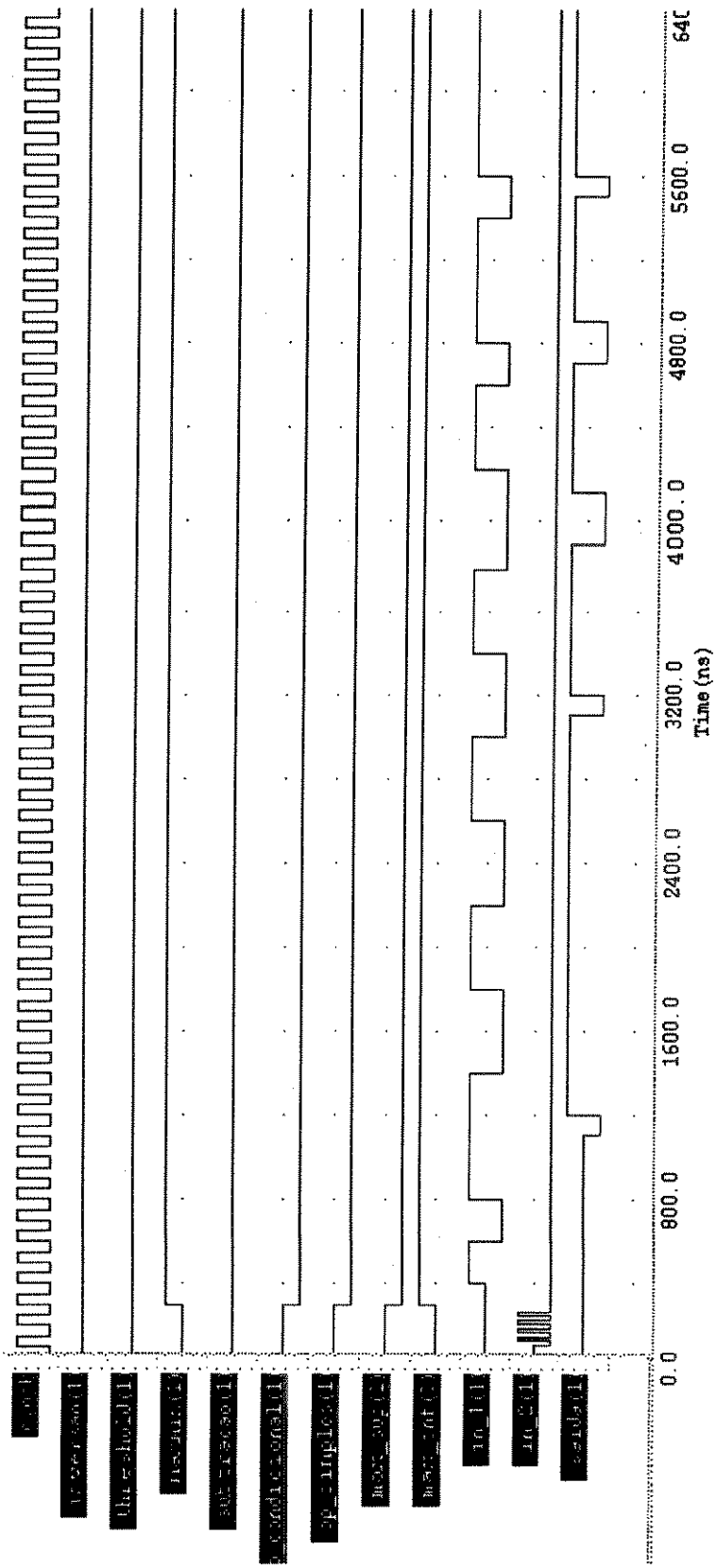

Figura C.1: operação de dilatação sobre uma imagem binária

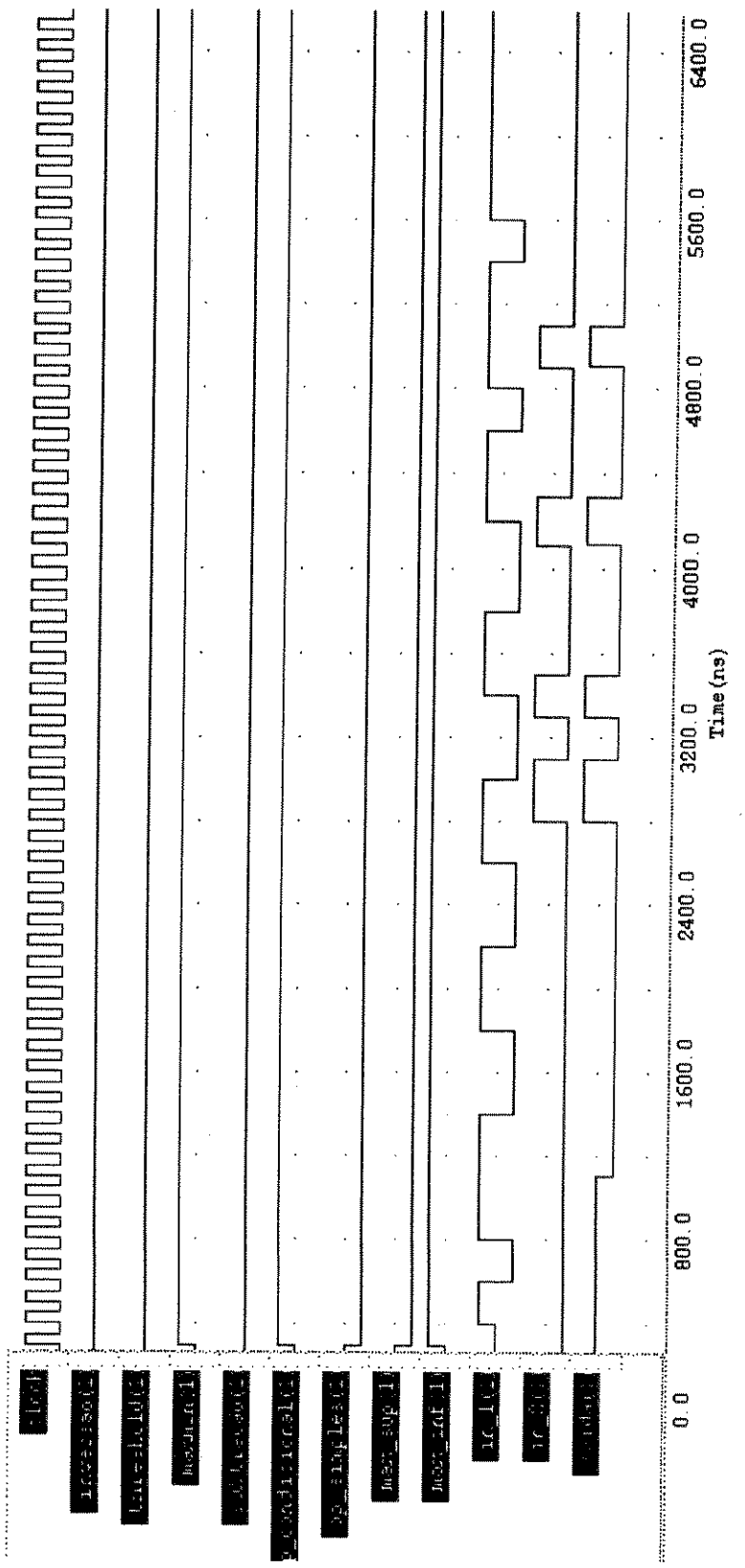

Figura C.2: operação de dilatação condicional sobre imagens binárias

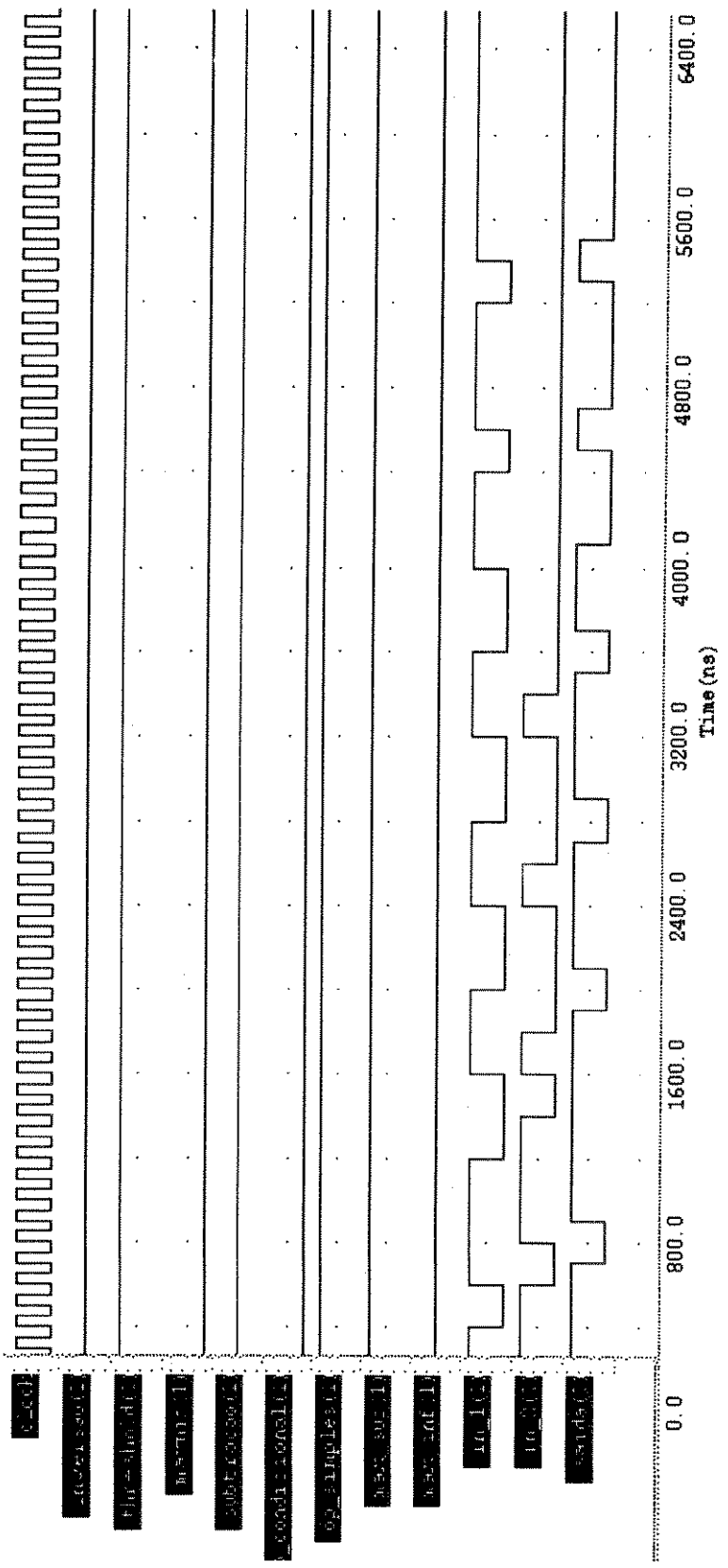

Figura C.8: operação de threshold sobre imagens binárias

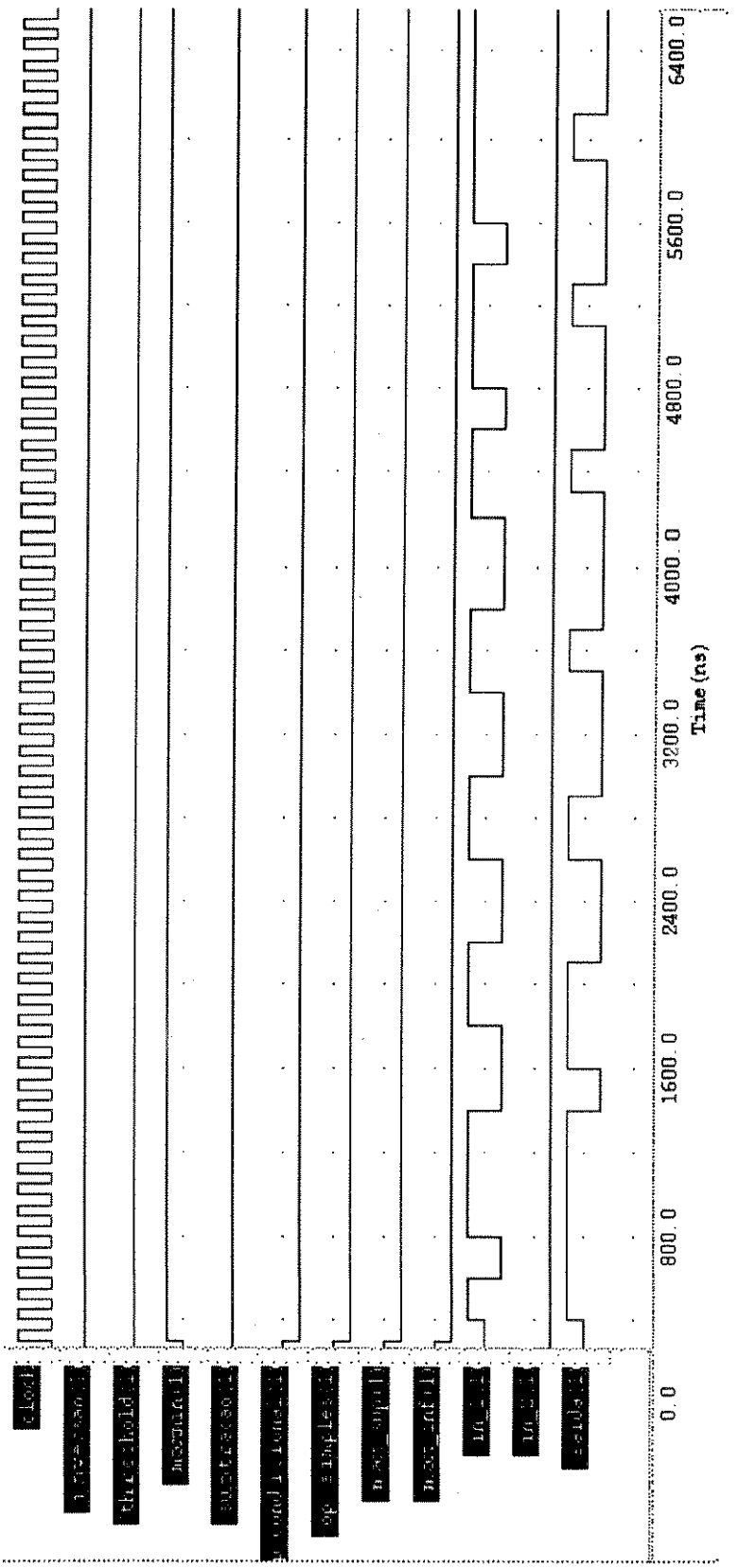

Figura C.3: operação de erosão sobre uma imagem binária

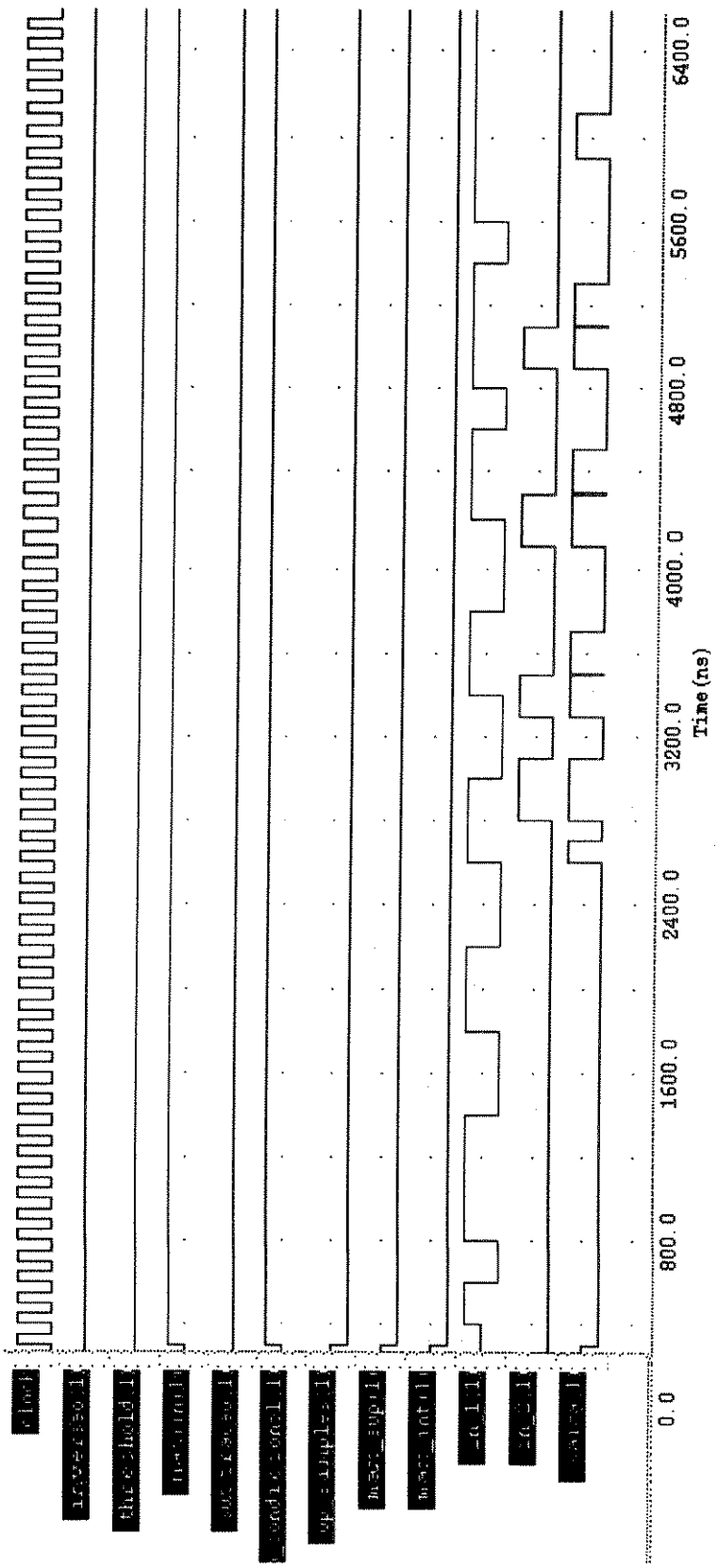

Figura C.4: operação de erosão condicional sobre imagens binárias

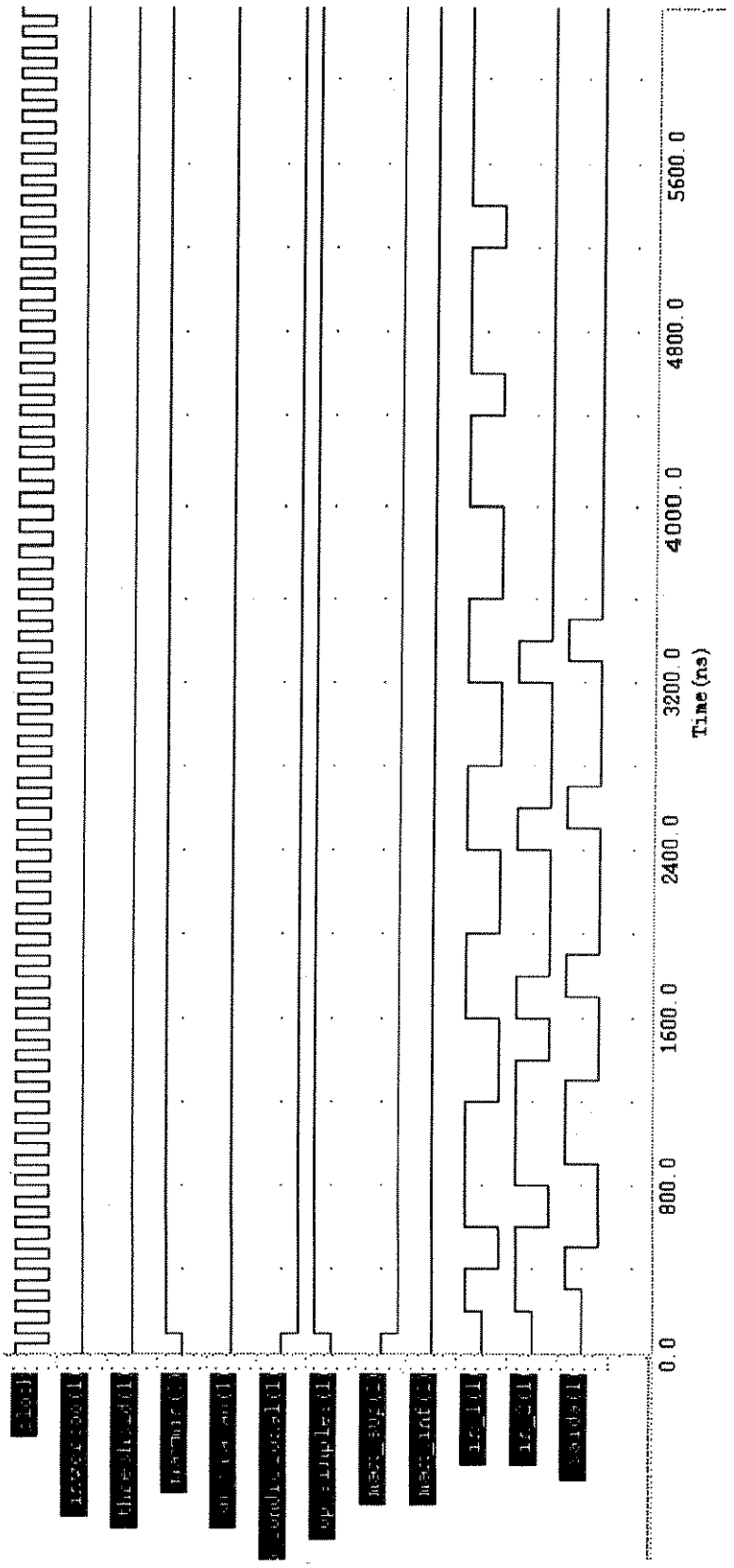

Figura C.5: operação de mínimo entre duas imagens binárias

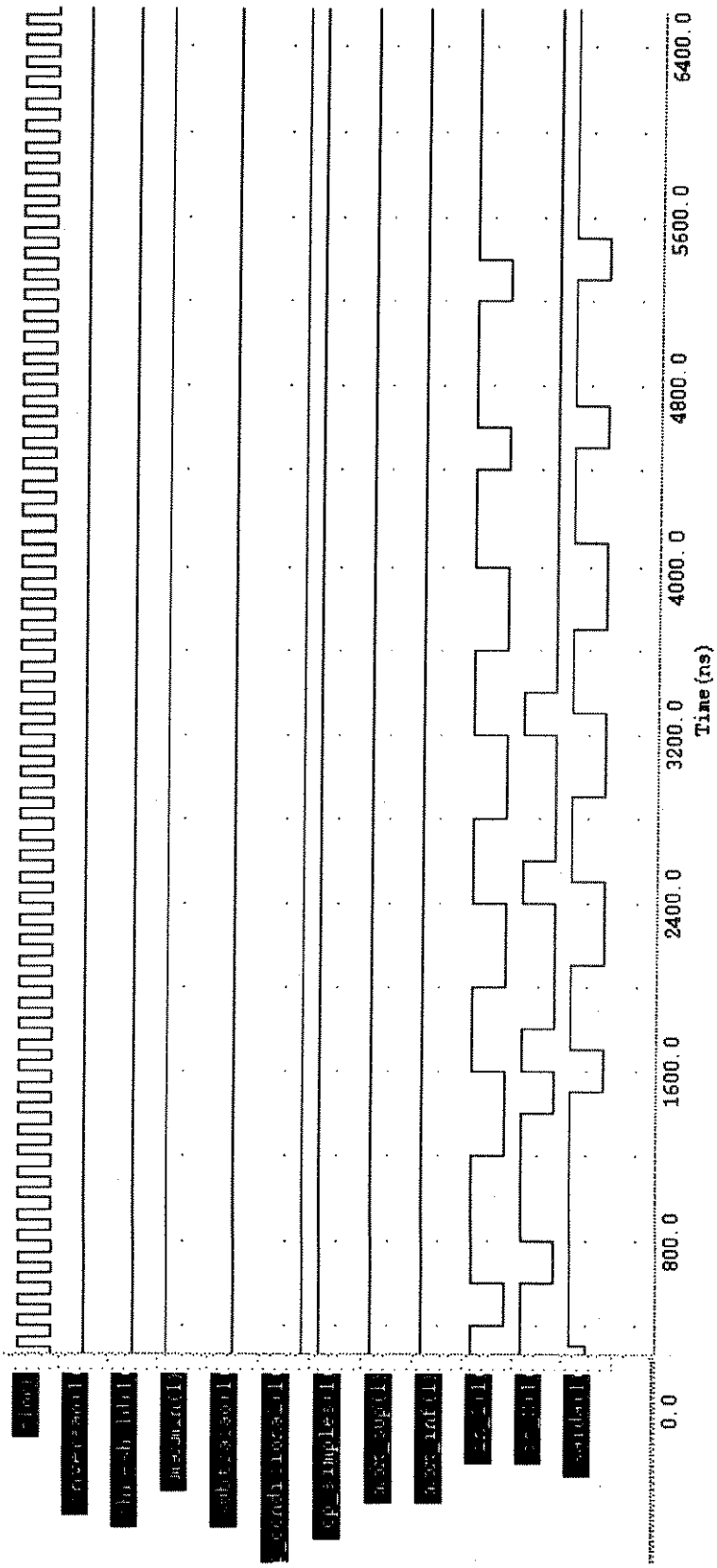

Figura C.6: operação de mínimo entre duas imagens binárias

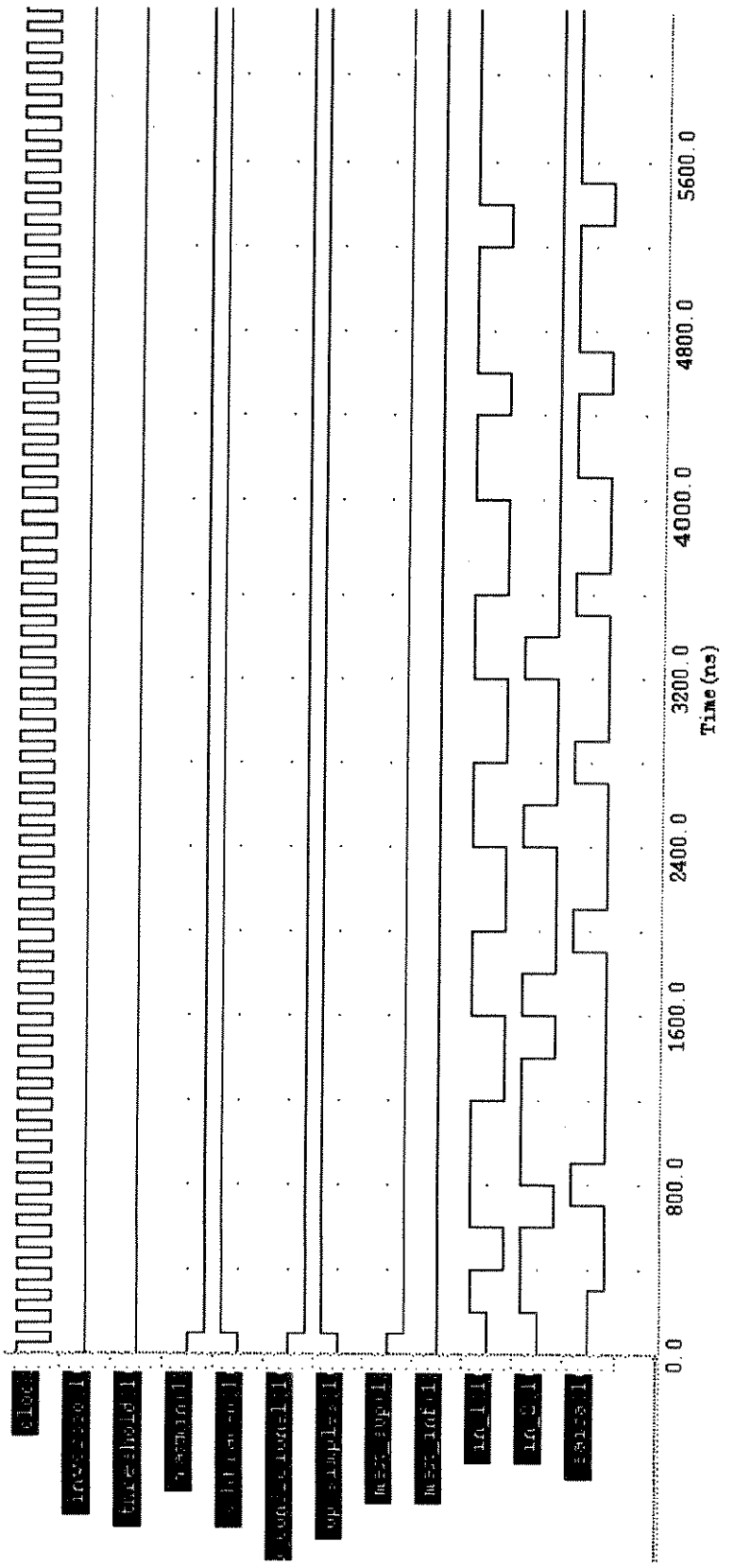

Figura C.7: operação de subtração entre duas imagens binárias

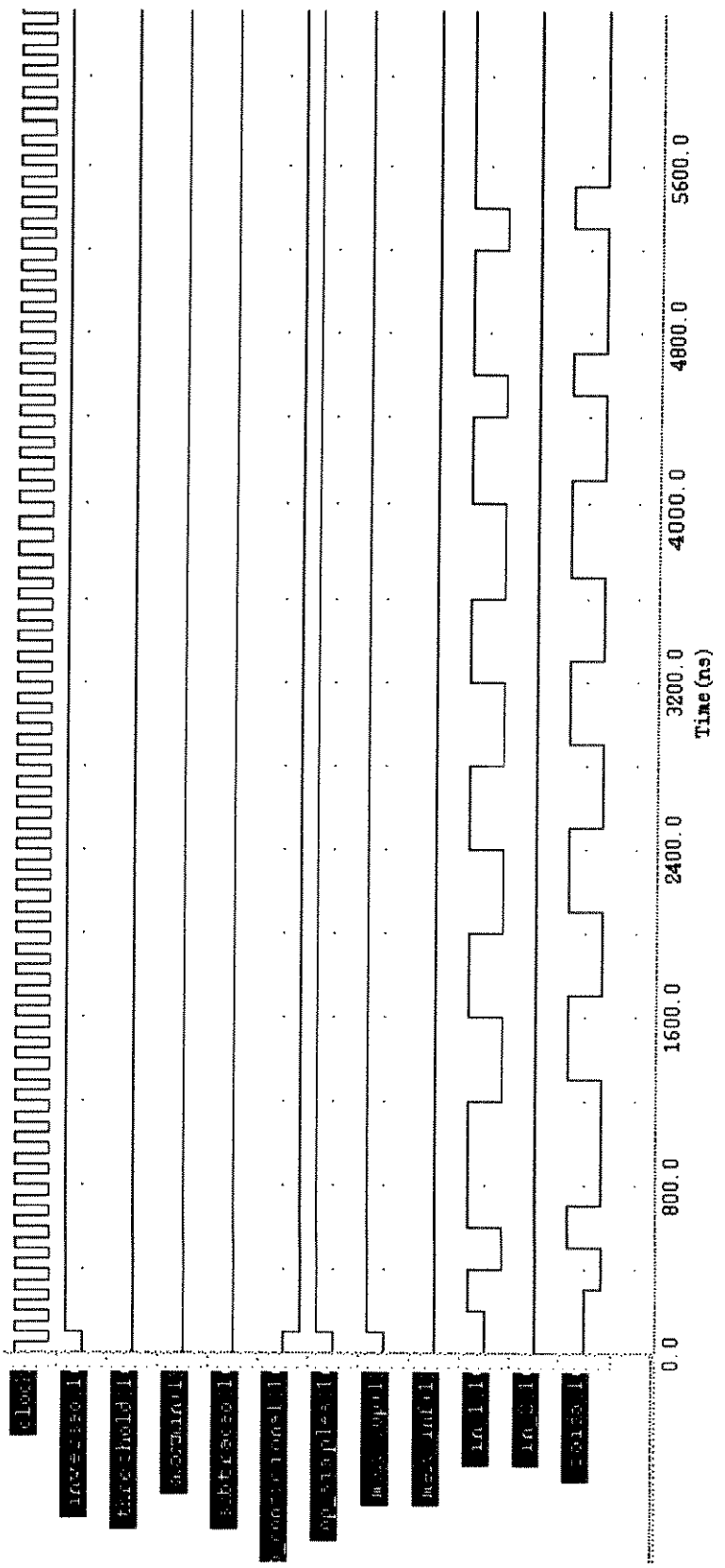

Figura C.9: operação de inversão sobre uma imagem binária

Figura C.10: operação de dilatação sobre uma imagem numérica

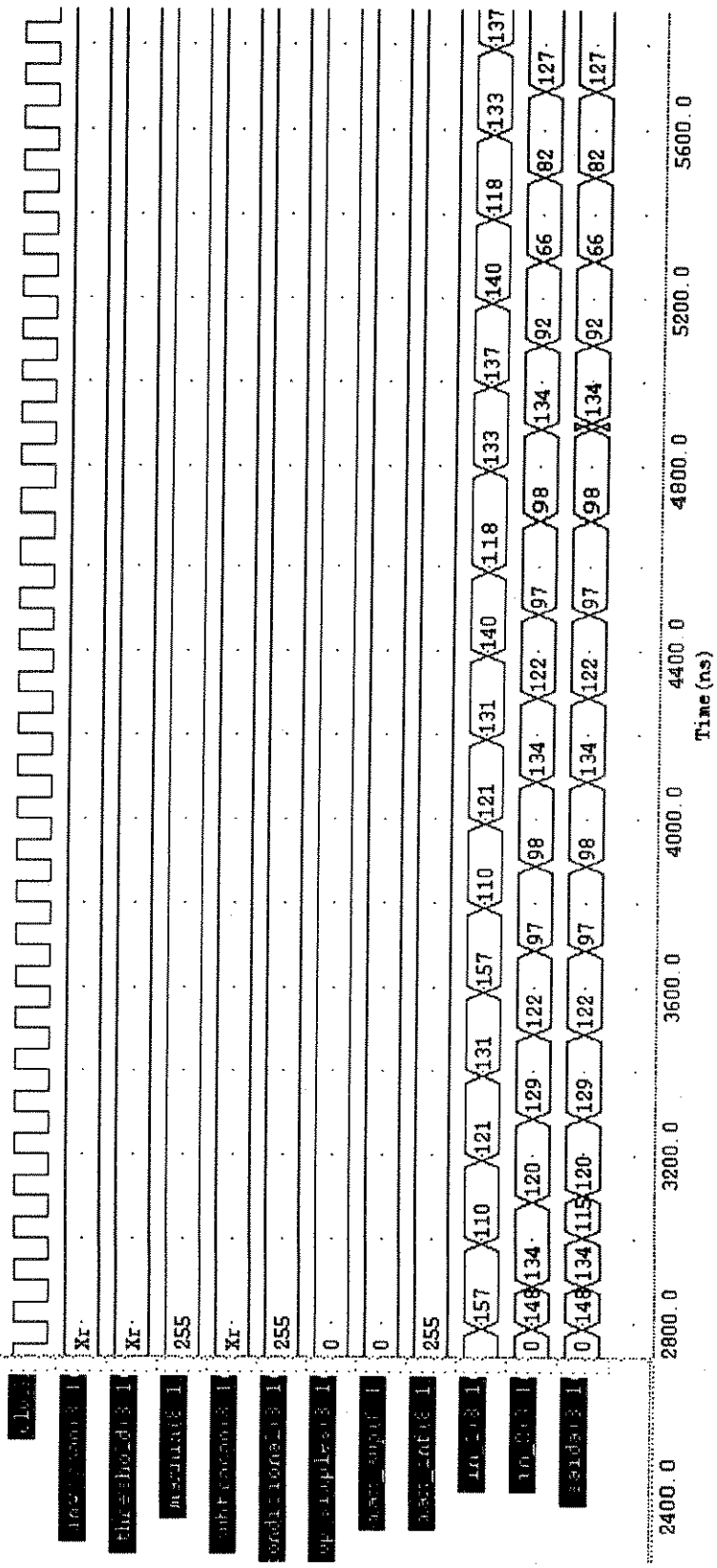

Figura C.11: operação de dilatação condicional sobre imagens numéricas

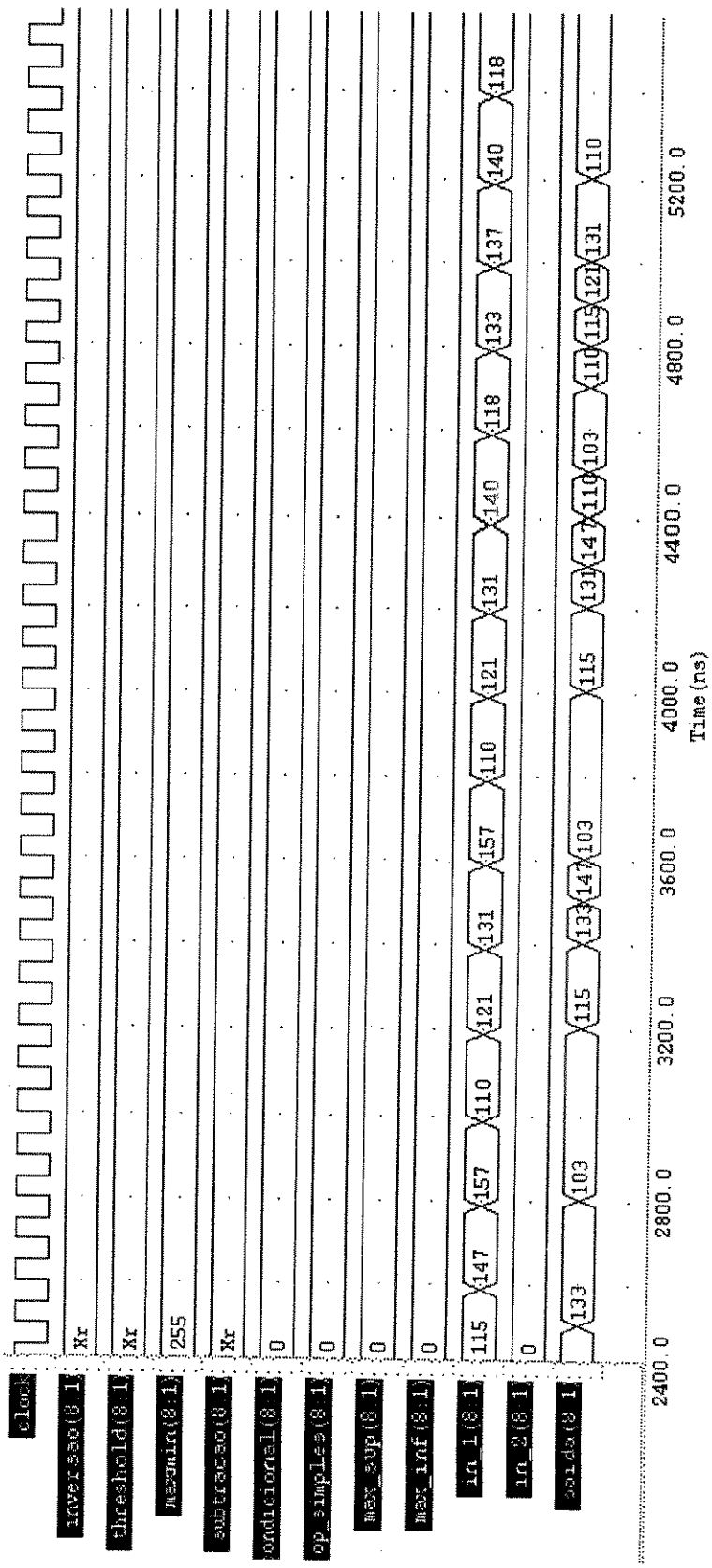

Figura C.12: operação de erosão sobre uma imagem numérica

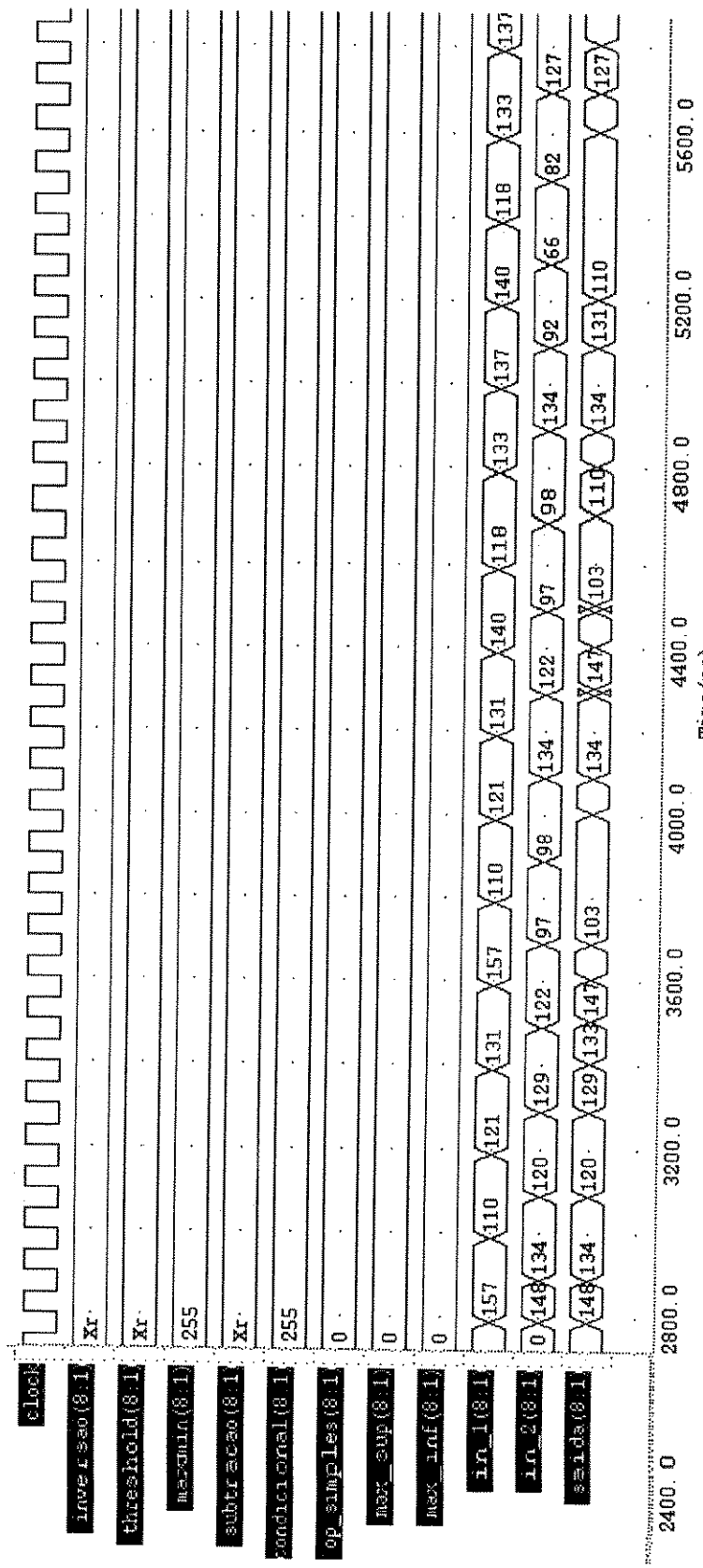

Figura C.13: operação de erosão condicional sobre imagens numéricas

Figura C.14: operação de mínimo entre duas imagens numéricas

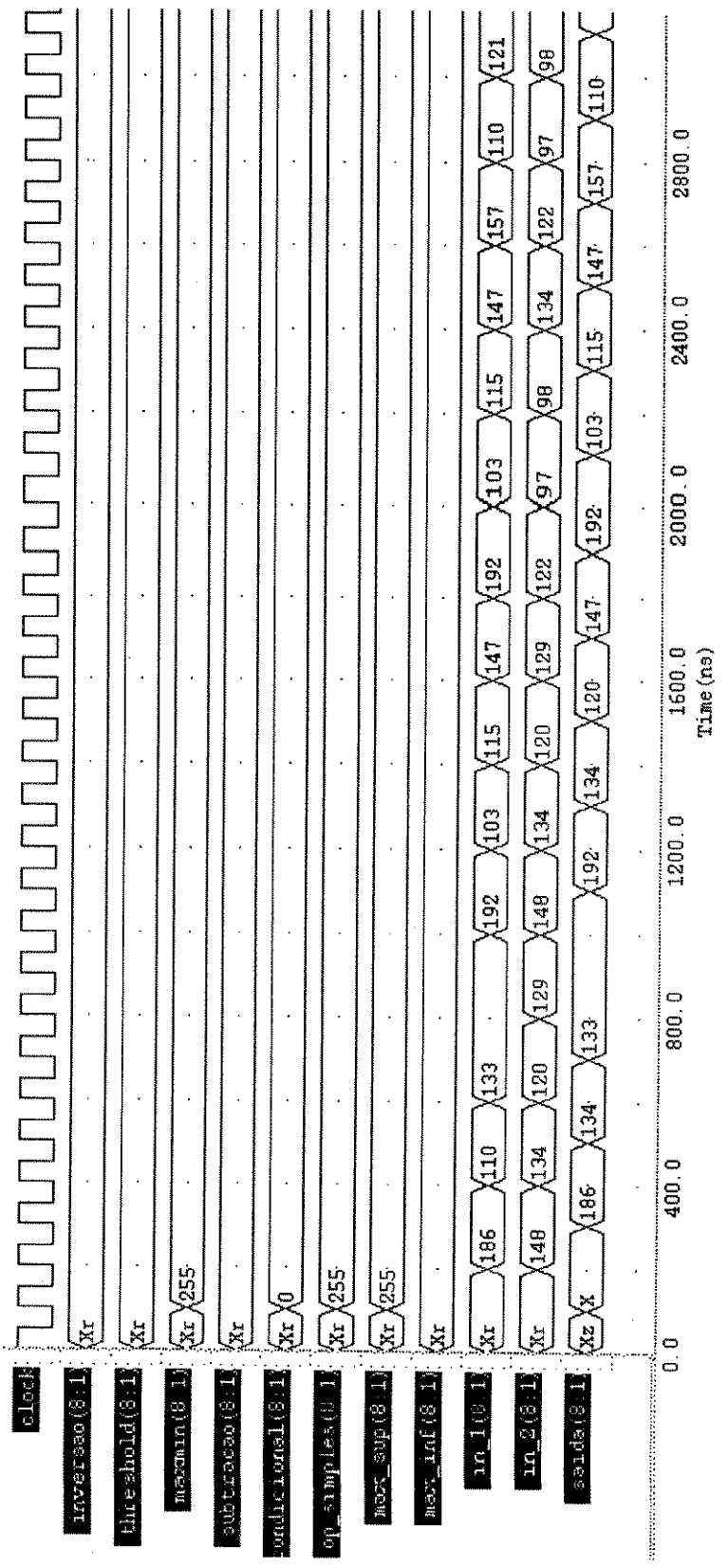

Figura C.15: operação de mínimo entre duas imagens numéricas

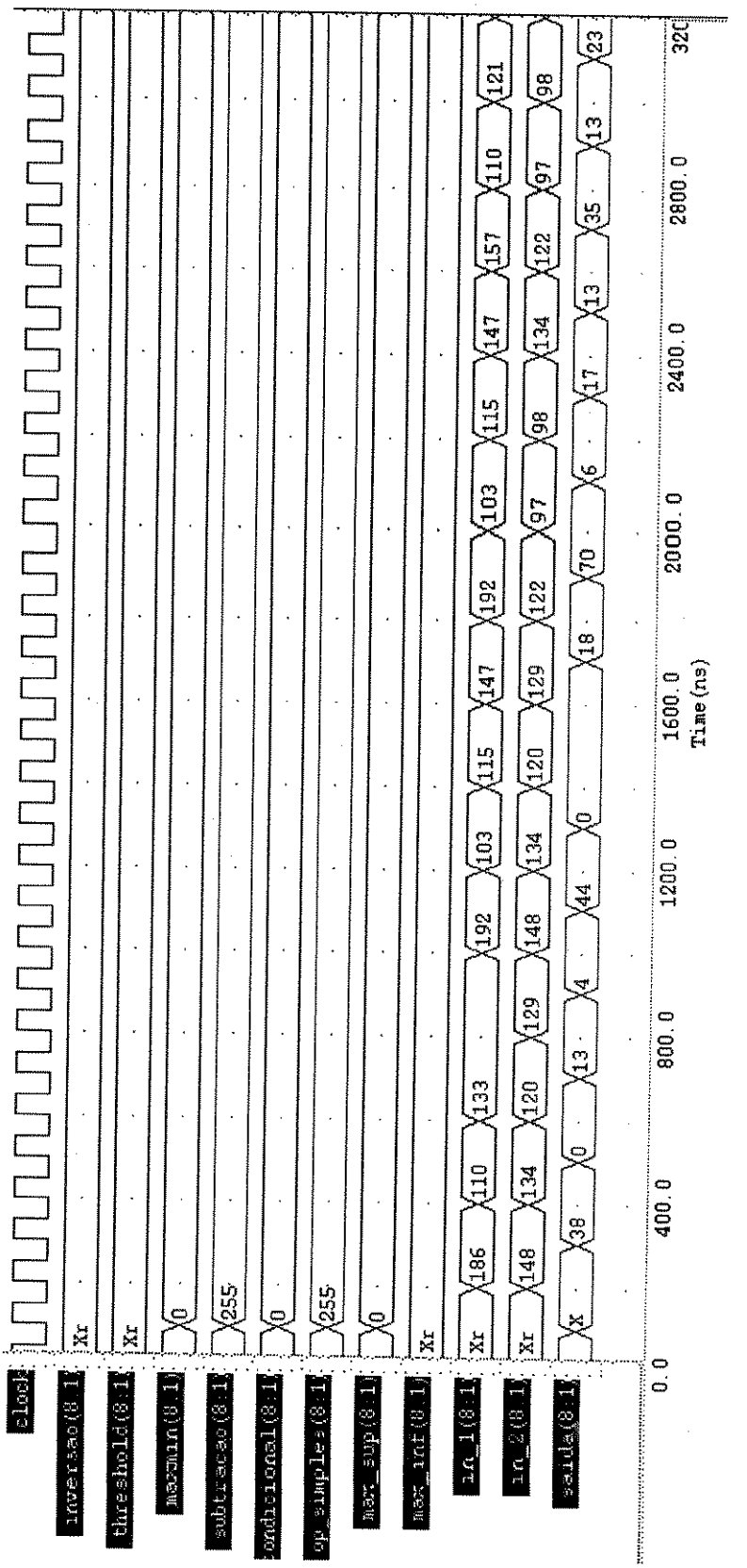

Figura C.16: operação de subtração entre duas imagens numéricas

Figura C.17: operação de threshold sobre imagens numéricas

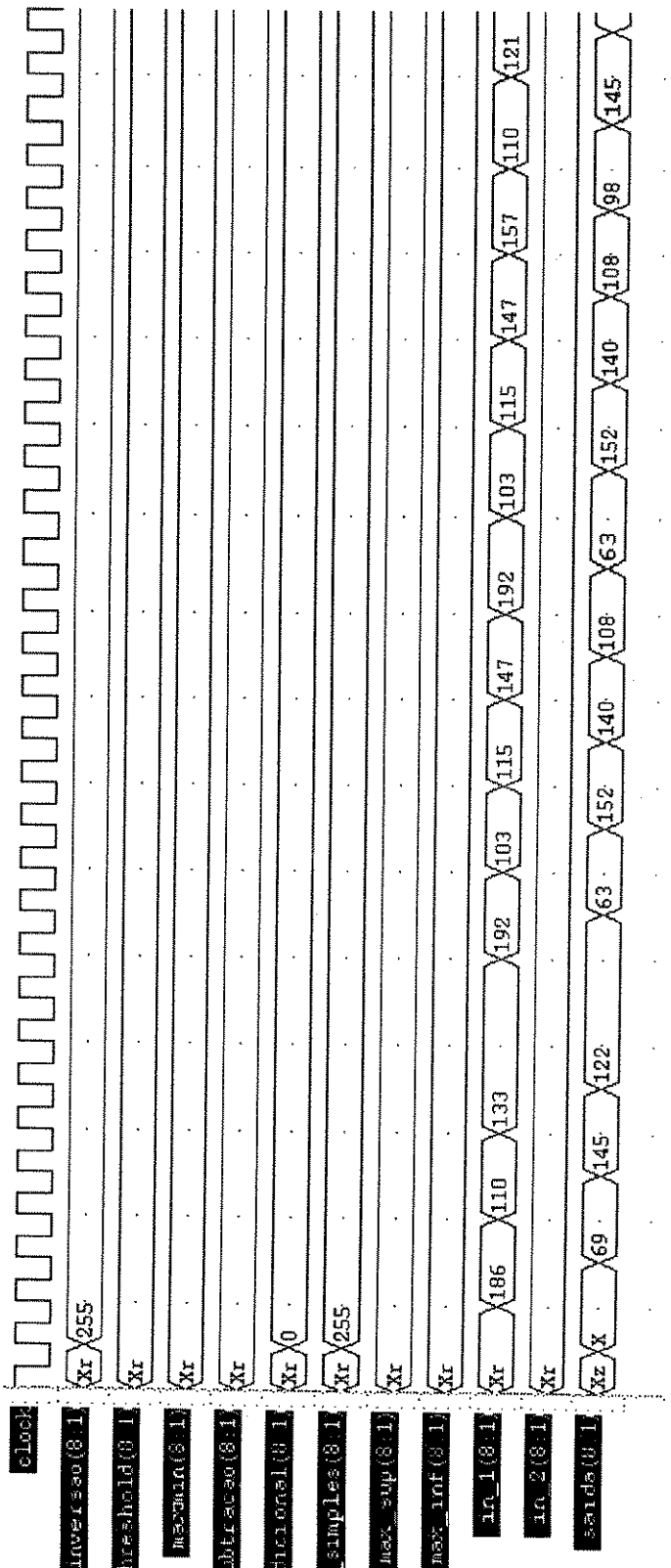

Figura C.18: operação de inversão sobre uma imagem numérica

BIBLIOGRAFIA

[1] **Wagner,F., Porto,I., Weber,R., Weber,T.**,*Métodos de Validação de Sistemas Digitais*, VI Escola de Computação, Campinas, 1988.

[2] **Piguet, C. and Dijkstra, E.**, *Design Methodologies and CAD Tools*, INTEGRATION, the VLSI journal 10, pp. 219-250, 1991.

[3] **Krüger, C.G.**, *ASIC e VHDL: Um estudo de metodologias de projeto visando reusabilidade de hardware*, Tese de Mestrado, Departamento de Ciência da Computação-Unicamp, Maio,1993.

[4] **Donlin, M.**, *Top-Down Methodologies Infiltrate System Designs*, Computer Design, Jan. 1992.

[5] **Mentor Graphics Corporation**, *Autologic VHDL Synthesis Guide.*

[6] **Mentor Graphics Corporation**, *Introduction to Top-Down Design Training Workbook.*

[7] **Dasgupta, S.**, *Computer Architecture - A Modern Synthesis*, vol 2, 1989

[8] **Vahtra, K.**, *HDL design methods simplify specs and boost productivity*, EDN, vol. 35, n.13, June, 1990.

[9] **Donlin, M.**, *ASIC complexity fuels drive to HDL design*, Computer Design, May, 1991.

[10] **Smith, S. and Larson, J.**,*A High Performance VHDL Simulator with Integrated Switch and Primitive Modeling*, in Computer Hardware Description Languages and their Applications, Darringer, J.A. and Ramming, F.J., 1990.

[11] **Dewey, A., Geus, A.J.**, *VHDL: Toward a Unified View of Design*, IEEE Design and Test of Computers, June, 1992.

[12] **Gunn, L.**, *VHDL: An EDA Standard Slowly Emerges*, Electronic Design, Mar., 22, 1990.

[13] **Costa, R.M. et al.** *Experience reveals VHDL's questions and answers*, Electonic Design, vol. 39, n. 23, Dec., 1991.

[14] **Mentor Graphics Corporation**, *Mentor Graphics Introduction to VHDL.*

[15] **Carlson, S. and Girczyc, E.**, *Understanding synthesis begins with knowing terminology*, EDN, vol. 37, Sept., 1992.

[16] **Mentor Graphics Corporation**, *An Introduction to Digital Simulation*, 1989.

[17] **Nagasamy, V., Berry, N., Dangelo, C.**, *Specification, Planning and Synthesis in a VHDL Design Environment*, IEEE Design and Test of Computers, vol. 9 n. 3, Sept. 1992.

[18] **Armstrong, J.**, *Chip-Level Modeling with VHDL.*

[19] **Mentor Graphics Corporation**, *System-1076 Reference Software*, 1992.

[20] **Barros, I.M.**, *Relatório Técnico de Simulações VHDL de uma Arquitetura para Morfologia Matemática*, Relatório Técnico DCA-FEE-UNICAMP, 1994.

[21] **Sunwoo, M.H., Aggarwal, J.K.**, *Flexibly Coupled Multiprocessors for Image Processing*, Journal of Parallel and Distributed Computing 10, pp. 115-129, 1990.

[22] **Fountain, T.J.**, *Processor Arrays - Architectures and Applications*, Academic Press, London, 1988.

[23] **Danielson, P. and Levialdi, S.**, *Computer Architectures for Pictorial Information Systems*, Computer Magazine IEEE, Nov. 1981.

[24] **Reeves, A.** *Meshes and Hypercubes for Computer Vision*, in Multicomputer Vision, Levialdi, S. (ed), 1988.

[25] **Duff, M.J.B**, *CLIP4: A Large Scale Integrated Circuit Array Parallel Processor*, in Proc. 3rd International Joint Conference on Pattern Recognition, Coronado, Ca., pp. 728-733.

[26] **Duff, M.J.B**, *Cellular Logic Image Processing*, Academic Press, London, 1986.

[27] **Unger, S.H.**, *A Computer Oriented Toward Spacial Problems*, Pro. fo the IRE, vol. 46, 1958, pp. 1744-1750.

[28] **Burks, A. W.**, *Essays on Cellular Automata*, University of Illinois Press, Urbana, 1970.

[29] **Azriel, R.** , *Parallel Image Processing Using Cellular Arrays*, Computer Magazine IEEE, Jan. 1983.

[30] **Young and Liu**, *VLSI Arrays for Pattern Recognition and Image Processing: I/O Bandwith Considerations*, in VLSI for Pattern Recognition and Image Processing, King-sun Fu (ed), 1984

[31] **Fountain, T.J.** *The Development of the CLIP7 Image Processing System*, Patt. Recongnition Lett. 1(5-6), pp. 331-339.

[32] **Fountain, T.J.**, *An Evaluation of Some Chips for Image Processing*, in Evaluation of Multicomputers for Image Processing, Uhr, L., Levialdi, S., Preston, K, Duff, M.J.B (eds.), Academic Press, 1986.

[33] **Uhr, L.**,*Pyramid Multi-computer Structures, and Augmented Pyramids*, in Computing Structures for Image Processing, M.J.B.Duff (ed).

[34] **Leite, N.J**, *Quelques Modéles d'Automates Cellulaires pour le Traitement d'Images*, Thèse de Doctorat de l'Université Paris 6, Mai, 1993.

[35] **Preston**, *Cellular Logic Array for Image Processing*, Handbook of Pattern Recognition and Image Processing, 1986.

[36] **Gerritsen, F.A**, *A Comparison of the CLIP4, DAP and MPP Processor-Array Implementations*, Computing Structures for image Processing, Duff, M.J.B.(ed), 1983.

[37] **Arvind, D.K., Robinson, I.N., and Parker, I.N.**, *A VLSI Chip for Real-Time Image Processing*, Proc. IEEE International Symposium on Circuits and Systems, pp. 405-408, 1983.

[38] **Weems, C and Lawton, D.T.**, *Incorporating Content Addressable Array Processors into Computer Vision System*, Private Communication, 1984.

[39] **Sudo, T., Nakashima,T., Aoki,M. and Kondo, T. 82**, *An LSI Adaptive Array Processor*, IEEE Int. Solid-State Circuits Conference, pp. 122,123,307.

[40] **Loocheed, R.M. et alli**, *Cytocomputers: Architectures for Parallel Image Processing*, Proc. Workshop Picture Data Descr. and Management, Pacific Grove, CA, 1980, pp. 281-286.

[41] **Duin, R.P.W. and Jonker, P.P.**, *Processor Arrays Compared to Pipelines for Cullular Image Operatioins*, in S. Levialdi (ed.), Multicomputer Vision, Academic Press, 1988.

[42] **Sternberg, S.R.**, *Pipeline Architectures for Image Processing - Algorithms and Programs*, in Preston, K. and Uhr, L. (eds), Multicomputers and Image Processing Algorithms and Programs, 1982.

[43] **Tanimoto, S.L.**, *A Pyramidal Approach to Parallel Processing*, Proc. 10th Annual International Symposium on Computer Architecture, pp. 372-378, Stockholm, 1983.

[44] **Mérigot, A. et alli**, *A Pyramidal System for Image Processing*, in Pyramidal Systems for Computer Vision, Cantoni, V. and Levialdi, S. (eds.), SpringerVerlag, Berlin, 1986.

[45] **Cantoni, V., Ferreti, M., Levialdi, S. and Maloberti, F.**, *A Pyramidal Project using Integrated Technology*, in Integrated Technology by Parallel Image Processing (ed Levialdi, S.) pp. 121-132, Academic Press, London, 1984.

[46] **Illiffe, J.K.**, *Computing in Store*, Internal Report, Queen Mary College, London, 1977.

[47] **Lea, R.M.**, *SCAPE: A Single Chip Array Processing Element for Image Analysis*, in VLSI 1983, Elsevier Science Publishers, Anceau, F. and Aas, E.J. (eds.), pp. 285-294, Amsterdam, 1983.

[48] **Basille, J-L., Castan, S. and Rozz, M.**, *Parallel Architectures adapted to Image Processing, and their Limits*, Computing Structures for image Processing, Duff, M.J.B.(ed), 1983.

[49] **Kung and Picard**, *One-dimensional Systolic Arrays for Multidimensional Convolution and Resampling*, in VLSI for Pattern Recognition and Image Processing, King-sun Fu (ed), 1984

[50] **Davis, R. and Thomas, D.**, *Geometric Arithmetic Parallel Processor - Systolic Array Chip Meets the Demands of Heavy-Duty Processing*, Electronic Design, Oct. 31, 1984.

[51] **Fisher, A.L. and Kung, H.T.**, *Design of the PSC: A Programmable Systolic Chip*, Proc. 3rd. CALTECH Conf. on VLSI, pp. 287-302.

[52] **Loui, A., Venetsanopoulos, A.N. and Smith, K.C**, *Flexible Architectures for Morphological Image Processing and Analysis*, IEEE Transaction on Circuits and Systems for Video Technology, vol. 2, n. 1, March 1992.

[53] **Lotufo, R.A.**, *Processamento de Imagens em máquinas basedas à Transputer*, IV SIBGRAPI, Jul., 1991, São Paulo.

[54] **Serra, J.** *Image Analysis and Mathematical Morphology*, vol. 1, 1982.

[55] **Matheron, G.** *Random Sets and Integral Geometry*, New Yor, Willey, 1975.

[56] **Pratt, W.K.**, *Digital Image Processing*, 1991

[57] **Jesus, E.O. , Lotufo, R.A.**, *Morfologia Aplicada à Visão Computacional*, Anais do VI Sibgrapi, 1992.

[58] **Jonker, P.P. and Komen, E. R.**, *A Scalable Real-Time Image Processing Pipeline*, 11th, IAPR International Conf. on Patter Recognition, The Hague, The Netherlands, 1992, pp. 142-146.

[59] **Person, E.**, *A Pipelined Image Analysis System Using Custom Integrated Circuits*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 10, n. 1, Jan 1988, pp. 110-116.

[60] **Djunatan, M., Mengko, T.**, *A Programmable Real-Time Systolic Processor Architecture for Image Morphological Operation, Binary Template Machine and Min/Max Filtering*, Proc. Int. Conf. on Acoustics Speech and Signal Proc., 1991, pp. 65-67.

[61] **Klein, J.C, Collange, F., Bilodeau, M.**, *A Bit Plane Architecture for an Image Analysis Processor Implemented with P.L.C.A. Gate Array*, First European Conf. on Computer Vision, Lecture Notes on Computer Science, Apr. 1990, Antibes, France.

[62] **Barrera, J., Banon, G.J.F. e Lotufo, R.A.**, *A Mathematical Morphology Toolbox for the Khoros System*, Relatório Técnico RT-MAC-9403, DCC-IME-USP, Jan 1994.

[63] **Gerritsen, F.A. and Verbeek, P.W.**, *Implementation of Cellular-Logic Operators Using 3x3 Convolution and Table Lookup Hardware*, Computer Vision, Graphics and Image Processing, 27, 1984.

[64] **Santos, E.T.**, *Um Sistema Sistólico para Interpolação de Imagens*, Anais do VI SIBGRAPI (1993), pp. 133-138.

[65] **Silva, A.C.R.**, *Contribuição à Síntese de Circuitos Digitais Utilizando Programação Linear Inteira 0 e 1*, Tese de Doutorado DT-FEE-UNICAMP, Set. 1993.

[66] **Shih, F. Y.-C., Mitchell, O. R.**, *Threshold Decomposition of Gray-Scale Morphology into Binary Morphology*, IEE Transactions on Patter Analysis and Machine Intelligence, vol. 11 n. 1, Jan. 1989.

[67] **Chang, Y., Ansari, N.**, *A Practical VLSI Realization of Mophological Operations*, SPIE Vol 1606 Visual Communications and Image Processing '91 Image Processing.