Uma Implementação em VLSI para Reconhecimento de Padrões de Imagens

Josemir da Cruz Alexandrino

Uma Implementação em VLSI para Reconhecimento de Padrões de Imagens

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Josemir da Cruz Alexandrino e aprovada pela Comissão Julgadora.

Campinas, 15 de abril de 1994.

Prof. Dr. Mario Lúcio Côrtes
Orientador

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Uma Implementação em VLSI para Reconhecimento de Padrões de Imagens[1]

Josemir da Cruz Alexandrino[2]

Departamento de Ciência da Computação
IMECC — UNICAMP

Banca Examinadora:

- Nelson Castro Machado (Suplente)[3]
- Mario Lúcio Côrtes(Orientador)[3]
- Furio Damiani[4]
- Ricardo de Oliveira Anido[3]

[1]Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação da UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

[2]Formado em Engenharia Elétrica pela Universidade Federal da Bahia.

[3]Professor do Departamento de Ciência da Computação - IMECC - UNICAMP.

[4]Professor do Departamento de Semicondutores, Instrumentos e Fotônica - FEE - UNICAMP.

A memória de minha avó
Laudelina Sta Rosa da Cruz
(1911 - 1993)

Agradecimentos

Ao CNPq e à FAPESP pelo apoio financeiro recebido

À Mentor Graphics pelo convênio que possibilitou o uso de suas ferramentas de EDA

A Mario Lúcio Côrtes pela orientação deste trabalho

Ao Professor Ricardo O. Anido pelas sugestões de Projeto

Ao Professor José Raimundo de Oliveira pela concessão de valiosas horas nos equipamentos do LCAEe

À José Marcos Lanna por sua contribuição no projeto físico e validação do sistema

A todos os colegas do DCC pela amizade e solidariedade

A Maria do Lago e todos os outros primos e amigos de Campinas pela recepção carinhosa e apoio com o qual sempre pude contar

A Terezinha Sta Rosa pelo apoio, carinho, amizade, otimismo, confiança ...

Aos meus pais Emilia e José por tudo que fizeram por mim

A Mirian pelo amor, carinho, apoio e principalmente pela paciência

A todos os familiares e amigos de Salvador pela confiança e incentivo

"O homem se torna muitas vezes o que ele próprio acredita que é. Se eu insisto em repetir para mim mesmo que não sou capaz de realizar alguma coisa, é possível que realmente me torne incapaz de fazê-la. Ao contrario, se tenho convicção de que posso fazê-la, certamente adquirirei a capacidade de realizá-la mesmo que não a tenha no começo."

Gandhi.

Resumo

Este trabalho está organizada em duas partes. Um estudo de técnicas e arquiteturas dedicadas ao reconhecimento de padrões de imagens é abordado na primeira parte. A segunda parte apresenta uma implementação em VLSI de um sistema para o reconhecimento de imagem utilizando a arquitetura de rede neural baseada em memórias RAMs.

O estudo de técnicas e arquiteturas inclui exemplos de métodos estatísticos, arquiteturas altamente paralelas e redes neurais. Os métodos estatísticos evidenciam a ineficiência das máquinas monoprocessadas convencionais de propósito geral no reconhecimento de padrões, especialmente nas aplicações de tempo real. Arquiteturas multiprocessadas, especificamente concebidas para esta tarefa, apresentam um desempenho elevado mas não conseguem manipular dados como aqueles encontradas nas imagens obtidas na prática, as quais geralmente apresentam algum conteúdo de ruído. Redes neurais são completamente diferentes de arquiteturas baseadas em processadores programáveis. A execução de um programa é substituído pelo treinamento da rede através de um conjunto apropriado de estímulos. Sua capacidade de generalização, isto é, de fornecer respostas adequadas à estímulos para os quais a rede não foi treinada, possibilita o processamento de dados com algum conteúdo de ruído. Elas foram concebidas a partir da observação dos sistemas nervosos naturais e seu funcionamento. Os neurônios podem ser modelados através de computadores convencionais ou dispositivos eletrônicos. A modelagem direta em dispositivos eletrônicos é mais eficiente e pode ser implementada em VLSI.

O sistema proposto é constituído de um cartão para IBM PC composto de uma PLD de controle e um número variável de ASICs que implementam as redes neurais. Os ASICs são agrupados em uma matriz, que pode ser dimensionada de acordo com as necessidades da aplicação, com capacidade máxima de 64 *chips*. As alternativas de projeto relevantes são apresentadas juntamente com a descrição do sistema e seu princípio de funcionamento, o qual foi validado através de um programa simulador escrito em Pascal. Os resultados de simulação obtidos com esse programa validaram o princípio de funcionamento e possibilitaram o dimensionamento de estruturas tais como barramentos e RAMs. Os aspectos de arquitetura do sistema e dos ASICs foram modelados e validados sobre uma descrição em alto nível escrita em VHDL. O uso de metodologias e ferramentas de EDA apropriadas, onde o projeto é dividido em vários níveis hierárquicos com diferentes graus de abstração possibilitou um maior controle do desenvolvimento do projeto, reduzindo as possibilidades de erro de projeto e reduzindo o tempo para sua realização. A arquitetura permite uma implementação física bastante elegante e regular do ASIC, realizada em CMOS 1,2 μm. A dissertação é concluída com um resumo dos trabalhos, extensões futuras e algumas considerações a respeito das características e limitações do sistema proposto.

Abstract

This work is organized in two parts. A study of specific architectures and techniques for image pattern recognition is presented in the first part. The second part presents a VLSI implementation for a image recognition system using the RAM-based neural network architecture.

The study of specific architecture and techniques includes examples of statistical methods, highly parallel architectures and neural networks. The statistical methods show the inefficiency of conventional general purpose single-CPU machines in pattern recognition, specially in real time applications. Multiprocessor systems, specifically designed for this task, present high performance but are not well fitted to handle data with a reasonable noise content, such as those found in real world images. Neural networks are entirely different from program based processor architectures. The function executed by processor programming is replaced by training of the neural net with a convenient set of stimulus. Their generalization capability to produce appropriate answers for stimulus out of training set facilitates noisy data processing. Neural networks emulate the nervous systems functionality by modeling their structures. The neurons can be modeled by simulation in conventional computer or by electronic devices. Straight modeling by electronic devices is more efficient and can be implemented in VLSI.

The proposed system consists of a PCB composed by a control PLD and a variable number of ASICs to implement the neural network. The ASICs are grouped into an array that can be sized according to the type of application, up to 64 chips. The relevant design alternatives are presented along with the system description and its functional principle, which was validated by a simulator program written in Pascal. The simulation results obtained with this program validated the functional principle and made possible the correct sizing of structures such as bus and RAMs. The system and ASICs architectural aspects were modeled and validated using a high level description written in VHDL. The use of adequate EDA tools and methodologies and the fact that the project was organized hierarchically with different levels of abstraction, allowed a good control of project development, reducing the chances of design error and shortening the development time. The architecture allows a very regular and elegant physical implementation for the ASIC, designed in 1.2 μm CMOS. The dissertation concludes with a summary presenting considerations about the proposed system advantages and limitations.

Conteúdo

Lista de Tabelas

Lista de Figuras

Glossário

ASIC: *Application Specific Integrated Circuits*

ATE: *Automated Test Equipment*

BAM: *Bidirectional Associative Memories*

CI: Circuito Integrado

CMOS: *Complementary Metal Oxide Silicon*

CPLD: *Complex Programmable Logic Device*

CPU: *Central Processing Unit*

CTI: Centro Tecnológico para Informatica

DMA: *Direct Memory Access*

EDA: *Electronic Design Automation*

EPROM: *Erasable Programmable Read Only Memory*

FPGA: *Field Programmable Gate Array*

GNU: *General Neural Unit*

GRAM: *Generic Random Access Memory*

I/O: Refere-se a dispositivos de entrada e saída (*Input/Output*)

IEEE: *The Institute of Electrical and Electronics Engineers, Inc.*

LFSR: *Linear Field Back Register*

LSB: *Least Significant Bit*

LSI: *Low Scale Integration*

LVS: *Layout Versus Schematic*

MDC:

MOS: *Metal Oxide Silicon*

MSB: *Most Significant Bit*

PAL: *Programmable Array Logic*

PC-AT: Referente ao computador pessoal da IBM que utiliza o processador 80386 da INTEL

PE: *Processor element*

PLA: *Programmable Logic Array*

PLD: *Programmable Logic Device*

PLN: *Probabilistic Logical Network*

PMU: Projeto Muti Usuário

RAM: *Random Access Memory*

VHDL: *VHSIC Hardware Description Language*

VHSIC: *Very High Speed Integrated Circuit*

VLSI: *Very Large Scale Integration*

PCB: *Printed Circuit Board*

Capítulo 1

Introdução

O crescente volume de pesquisas na área de modelagem e síntese de sistemas "inteligentes" aliado ao desenvolvimento tecnológico de dispositivos eletrônicos dos últimos anos, e as necessidades de processamento em tempo real da maioria das aplicações de sistemas de reconhecimento de imagem (como por exemplo na automação do controle de qualidade em indústrias), demandam a concepção de sistemas de reconhecimento de imagem cada vez mais eficientes.

A necessidade de computadores mais potentes para o processamento em sistemas que manipulam grande quantidade de informação levou ao desenvolvimento de computadores não convencionais tais como computadores com múltiplos processadores, matriz de processadores, redes neurais etc. Em geral, o aumento de eficiência nestes tipos de computadores se dá devido ao processamento em paralelo dos seus processadores.

Sistemas baseados em arquiteturas convencionais são ineficientes no processamento de imagens devido à complexidade das operações de seus algoritmos e a grande quantidade de informação a ser processada. Se por exemplo, aplicarmos a técnica de classificação pela mínima distancia [Ree82], (computada através da equação 1.1), ao reconhecimento de padrões de imagens binárias de n^2 pixels num computador convencional, teremos um tempo de execução proporcional a n^2. Estima-se que o tempo de execução de um IBM PC-386 com clock de 33Mhz, seja de 3,04 ms para duas imagens de 128 × 128 *pixels*. Admitindo que a imagem pode estar deslocada tanto horizontalmente como verticalmente em no máximo j pixels o tempo de execução sobe para um valor proporcional a j^2n^2. Para $j = 10$ o tempo de processamento no PC-386 é de 1,34 s. Uhr [Ree82] estima que o tempo de execução em uma matriz de processadores binários neste ultimo caso, é de 618 μs.

$$mdc = \sum_{i=1}^{n^2} x_i \oplus y_i \tag{1.1}$$

Estas estimativas sugerem que o reconhecimento e descrição de objetos numa imagem simples, tipicamente necessita de muitos bilhões de instruções e a maioria das aplicações exigem um processamento em tempo real da ordem de dezenas de mili-segundos. A limitação de velocidade da tecnologia do silício descarta o uso de computadores convencionais para esses tipos de aplicações. Tais computadores, classificados como seriais de propósito geral, são projetados para atender as mais diversas aplicações, mantendo uma arquitetura complexa e bastante flexível penalizando o

desempenho da máquina. Outra classe de arquitetura é a orientada ao problema, denominada de arquiteturas de propósito específico. Elas possuem estruturas mais simples, projetadas especificamente para uma dada aplicação, resultando em maior desempenho. É mais um caso de compromisso entre desempenho e flexibilidade. As arquiteturas de propósito especifico podem ser implementadas em processadores seriais ou paralelos. O conceito de "paralelismo" foi incorporado ao projeto de novos computadores que explora a característica dessas aplicações de terem tarefas que podem ser executadas em paralelo, resultando numa efetiva ampliação dessa limitação.

Por outro lado, a facilidade com que o cérebro humano resolve problemas de reconhecimento de padrões, como um atividade corriqueira, tem levado muitos pesquisadores a investigar como o cérebro trabalha na esperança que os princípios de seu funcionamento possam ser aplicados ao projeto de máquinas capazes de resolver problemas como esse. Tais máquinas, que modelam o trabalho do cérebro humano através de suas estruturas anatômicas, são conhecidas como "computadores neurais" ou "redes neurais" [Red88].

As redes neurais (rn) podem ser implementadas através de simuladores em computadores convencionais, onde a eficiência é limitada, ou diretamente sobre dispositivos VLSI[1] analógicos ou digitais. Aceleradores digitais em hardware podem modelar eficientemente os neurônios, dando respostas rápidas e precisas a um custo relativamente alto na área de silício ocupada pelas células. Dispositivos VLSI analógicos são igualmente rápidos, consomem pequenas áreas de silício porem suas respostas não são tão precisas.

Abu-Mostafa em [AMP87] descreve um computador neural ótico para reconhecimento de padrões de imagens. Essa tecnologia de computadores óticos é outra área de pesquisa interessante. Sua habilidade de estabelecer uma grande rede de comunicação entre os processadores elementares é a principal característica que a diferencia da tecnologia do silício. Dispositivos eletrônicos tais como um chip são limitados em número de pinos causando limitação na capacidade de comunicação. Num sistema ótico, milhões de raios de luz podem transmitir dados entre os processadores. Na verdade, conexões óticas estão sendo consideradas como um meio de solucionar as limitações de comunicação encontradas em chips VLSI.

1.1 Objetivos, Metas e Restrições

Este dissertação apresenta o estudo de algumas técnicas e arquiteturas utilizadas em reconhecimento de imagens, onde procurou-se identificar uma estrutura adequada à implementação em VLSI. A proposta de uma arquitetura para reconhecimento de padrões de imagens baseada no princípio das redes neurais digitais modeladas em pequenas memórias RAMs foi inspirada no Wisard, um dispositivo criado por Wilkie, Stonham e Aleksander, do grupo de computação neural do *Imperial College of Science and Technology*. Trata-se de uma implementação modelada sobre um sistema de propósito geral com algum *hardware* discreto dedicado. A vasta gama de aplicações (como a automação do controle de qualidade em indústrias), a ausência de referências na literatura a respeito de implementações em VLSI de arquiteturas semelhantes a esta, o interesse do grupo de arquitetura e ferramentas de EDA do DCC/UNICAMP e a disponibilidade de tais ferramentas motivaram a realização deste trabalho.

O objetivo principal é o projeto de um circuito integrado dedicado (ASIC[2]) usando tecnologia

[1] Very Large Scale Integration

[2] Application Specific Integrated Circuits

CMOS numa alta escala de integração (VLSI), capaz de classificar uma imagem desconhecida entre as diversas imagens previamente conhecidas. Para tanto foi desenvolvida a descrição de um sistema formado por uma matriz de tais ASICs, numa linguagem de alto nível para descrição de *hardware*.

A descrição funcional do sistema foi desenvolvida como suporte ao projeto do ASIC e sua implementação não constitui um objeto deste trabalho. A completa implementação do sistema envolve o desenvolvimento do *software* de controle, modulo de aquisição de imagens e confecção do cartão contendo a matriz de ASICs e interface com o computador hospedeiro, devendo ser considerada após a implementação do ASIC.

1.2 Organização

Esta dissertação está organizada em duas partes. A primeira parte, constituída dos capítulos 2 e 3, apresenta um estudo de técnicas e arquiteturas utilizadas para reconhecimento de padrões. A segunda parte, capítulos 4 ao 7, aborda o projeto do sistema de reconhecimento de imagens proposto.

O capítulo 2 aborda o reconhecimento de padrões de imagens sob três vertentes: métodos estatísticos executados em arquiteturas convencionais (mono-processadas), arquiteturas altamente paralelas e redes neurais. O capítulo enfoca o uso de paralelismo como um meio de alcançar a velocidade de processamento que a maioria das aplicações requerem. Como no exemplo onde a técnica de classificação pela mínima distância é executada em uma matriz de processadores denominada de *systolic array*, aumentando a velocidade de processamento em $3 \times n \div 2$ (com $n =$ número de *pixels* da imagem) em relação à execução em uma arquitetura mono-processada. A classe de arquiteturas baseada nos princípios de funcionamento de sistemas nervosos biológicos, conhecidas como redes neurais (rn), também utiliza paralelismo mas distingue-se das outras classes vistas (mono e multi processadas) por não ser dirigida a execução de um algoritmo. O assunto é introduzido no final do capítulo através de um exemplo de rn probabilística e aprofundado no capítulo 3. Por fim, uma análise comparativa do desempenho das abordagens é apresentada. Outros critérios, como imunidade a ruído, custo dos processadores e facilidade de implementação em VLSI, também são analisados.

O capítulo 3 detalha com maior profundidade os sistemas neurais incluindo aspectos de ordem prática como a implementação de tais sistemas em VLSI. O capítulo apresenta a constituição e funcionamento do neurônio, indicando as diferenças e semelhanças entre os sistemas neurais naturais e um computador. A organização das células nervosas é comentada através do exemplo da retina animal, onde são abordados os aspectos topológicos, funcionalidade e diversidade de tais células. Em seguida, são apresentados alguns modelos, implementados com circuitos eletrônicos, para o neurônio e os tipos de rn atualmente estudados. Do ponto de vista de implementação em VLSI, são apresentadas as alternativas analógica e digital, com uma análise das restrições e vantagens de cada implementação. O capítulo é finalizado com dois exemplos de implementação inspirados nos princípios dos sistemas neurais. O primeiro é uma implementação híbrida (analógica e digital) de uma retina em silício e o segundo, um sistema integrado de *software* e algumas placas dedicadas para reconhecimento de padrões, o qual inspirou o sistema proposto.

O capítulo 4 apresenta o sistema implementado, seu princípio de funcionamento e a evolução de sua concepção. Algumas importantes decisões de projeto, tais como a distribuição física do sistema, a forma de acesso ao computador hospedeiro, são apresentadas e justificadas através da análises de alternativas. O capítulo também apresenta uma descrição sumária da especificação em alto nível

desenvolvida em VHDL que foi usada na validação funcional do sistema.

A implementação do circuito integrado é abordada no capítulo 5, que enfatiza a necessidade de uma metodologia na execução de projetos VLSI. O capítulo descreve as principais fases desta metodologia, sugere algumas alternativas para implementação da *interface* entre o computador e os ASICs, aponta os principais problemas e falhas detectadas durante a validação de cada fase.

A validação das diversas fases do projeto, desde as fases de mais alto nível, como a validação do princípio de funcionamento do sistema, até o projeto físico, foi feita a partir da análise de resultados obtidos diretamente das ferramentas de simulação e é apresentado no capítulo 6.

O capítulo 7 apresenta um sumário da dissertação, extensões relativas ao teste do protótipo do ASIC, projeto do cartão do sistema, programa de controle e modulo de aquisição de imagens, concluindo com uma análise das características e limitações do sistema.

Parte I

Fundamentação Teórica

Capítulo 2

Hardware para Reconhecimento de Imagens

2.1 Introdução

Grandes esforços estão sendo atualmente realizados na elaboração de sistemas de analise de imagens estáticas e dinâmicas, em tempo real de execução. Muitos pesquisadores [MKD82] já enfatizavam a necessidade de se aprender os mecanismos de visão biológica visto que estes processam em tempo real, imagens dinâmicas e que poderiam indicar uma arquitetura apropriada para tais sistemas. Sistemas como redes neurais são exemplos da observação de sistemas biológicos os quais mostram se extremamente eficientes nas suas tarefas. A execução simultânea de tarefas em sistemas multiprocessados, sob forma de *arrays*, árvores ou pirâmides, eleva bastante a velocidade de processamento dos sistemas viabilizando muitas aplicações de tempo real que requerem operações simples como reconhecimento de um padrão de *pixels*.

No projeto de um sistema para reconhecimento de imagem devem ser levados em consideração:

- **Velocidade de processamento.** A maioria das aplicações requerem processamento em tempo real de execução. Em um sistema típico, a recepção de imagens de 1024×1024 *pixels* de 8 *bits* pode alcançar uma taxa de 30 quadros por segundo. Isso significa que o sistema deve ser capaz de processar um milhão de *pixels* em 33 ms.

- **Alto grau de paralelismo.** Para obter velocidades elevadas, como mostrado acima, o sistema deve efetuar o maior numero possível de operações em paralelo.

- **Linguagem de programação específica.** Uma linguagem de programação específica para máquinas massivamente paralelas deve facilitar a solução e programação de algoritmos paralelos.

- **Imunidade a ruídos.** Ruídos podem ser introduzidos nos dados a serem processados de várias maneiras. Vibração da câmara de TV, variação na iluminação da cena e captação de ruídos elétricos em partes dos circuitos, são fontes de ruídos que podem perturbar o funcionamento do sistema.

- **Arquitetura de baixo custo.** O alto grau de paralelismo implica em um numero de processadores elevado. Uma solução de compromisso entre velocidade e custo deve ser adotada conservando a eficiência do sistema.

- **Arquiteturas uniformes e interconexões regulares.** Esse requisito facilita a implementação dos sistemas em VLSI.

2.2 Técnicas Convencionais

De uma maneira geral, sistemas que usam computadores com um único processador, os quais executam suas instruções serialmente, são chamados de convencionais. Especificamente em processamento e reconhecimento de imagens, tais sistemas mostram-se bastante ineficientes devido ao grande número de operações envolvidas neste tipo de tarefa. Os sistemas convencionais tornam-se inviáveis quando algoritmos que levam em conta, e tentam compensar, fontes de degradação como ruídos, são usados e/ou quando execução em tempo real é exigida.

A seguir são apresentados dois métodos clássicos em reconhecimento de imagens que demonstram a grande quantidade de operações citadas acima.

2.2.1 Classificação pela Mínima Distância

MDC (*minimum-distance classification*) é uma técnica muito popular em reconhecimento estatístico de padrões. Ela consiste em classificar um padrão desconhecido em uma determinada classe se a distância entre este padrão e o padrão de referência (associado à classe) for mínima com relação às outras classes [Ree82].

Se X é um padrão desconhecido e $\{C_1, C_2, ..., C_l\}$ são classes com padrões de referências $\{R_1, R_2, ..., R_l\}$. Então X é tido como classificado na classe C_j se:

$$D(X, R_j) = Min\{D(X, R_i), i = 1, 2, ..., l\} \tag{2.1}$$

Onde $D(X, R_j)$ é a distância entre X e R_j, que é calculada através de 2.2, considerando $X = \{x_1, x_2, ..., x_k\}$ e $R_j = \{r_{j,1}, r_{j,2}, ..., r_{j,k}\}$ vetores que descrevem pontos de um espaço característico.

$$D_p(X, R_j) = (\sum_{i=1}^{k} |x_i - r_{j,i}|^p)^{1/p} \tag{2.2}$$

Para $p = 2$, a equação 2.2 é chamada de distância Euclideana e, o método, de classificação linear. Em [Ree82] é sugerida uma solução para *image registration*[1] que usa uma equação chamada de MSE (*mean squared error* ou erro médio quadrático) que, a menos de uma constante de normalização, é a mesma que calcula a distância Euclideana. O processamento da equação 2.2 envolve k subtrações, k exponenciações, $k - 1$ adições e 1 radiciação, resultando em $3k$ operações. O total de operações para computar as distâncias entre o padrão desconhecido e as l classes é $3 \times l \times k$.

[1] Quando uma imagem pode ser "casada" com uma outra através de deslocamentos no plano x, y.

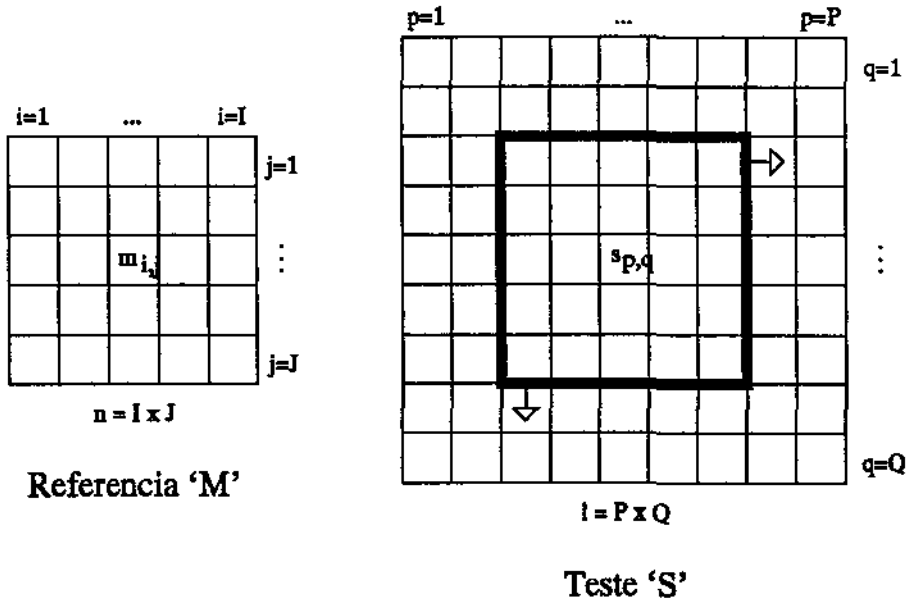


Figura 2.1: Modo de indexação das imagens de referência e de teste, usado na função de correlação.

2.2.2 Função de correlação

Outra medida de similaridade entre dois padrões pode ser obtida pela versão normalizada da função de correlação descrita por [Gem82], que é apresentada abaixo com uma conveniente mudança no modo de indexação dos elementos de imagens:

$$C(p,q) = \frac{\frac{\sum_{i,j} M(i,j)S(p+i,q+j)}{n} - \frac{\sum_{i,j} M(i,j) \sum_{i,j} S(p+i,q+j)}{n^2}}{(\frac{\sum_{i,j} M(i,j)^2}{n} - \frac{(\sum_{i,j} M(i,j))^2}{n^2})(\frac{\sum_{i,j} S(p+i,q+j)^2}{n} - \frac{(\sum_{i,j} S(p+i,q+j))^2}{n^2})} \tag{2.3}$$

Na equação 2.3 $M(i,j)$ é um elemento da imagem de referência, $n = I \times J$ é o número de elementos desta imagem, $S(i,j)$ é um elemento da imagem sob teste. A figura 2.1 mostra o modo de indexação dos elementos de ambas às imagens. A função é calculada para cada ponto (p,q) deslocando a imagem de referência sobre a de teste. Com este processo é possível, por exemplo, identificar a presença de objetos em uma determinada cena ou rastrear um objeto em movimento.

O resultado da função é limitado ao intervalo $[-1;+1]$, e um máximo de similaridade corresponde a um máximo da função $(+1)$. Seu resultado é também independente de transformações lineares de níveis de intensidades (causadas, por exemplo, pela mudança de iluminação do objeto). Observando que cinco entre os sete somatórios da função são distintos, a computação desta função para um ponto (p, q) requer $5 \times (i \times j - 1)$ (ou $5n-5$) adições, 3 subtrações, $I \times J + 2$ (ou $n+2$) multiplicações, 6 divisões e $2 \times J \times I + 5$ (ou $2n + 5$) exponenciações, resultando em um total de $8n + 11$ operações.

2.3 Técnicas Alternativas

A eficiência de sistemas de reconhecimento de imagem pode melhorar substancialmente com a execução da maioria das operações em paralelo. Estas operações, geralmente realizadas a nível de *pixels*, têm um grau de independência muito grande possibilitando a execução em paralelo. Isso, aliado ao desenvolvimento de técnicas e máquinas alternativas, possibilitaram o surgimento de sistemas multiprocessados dedicados ao processamento e reconhecimento de imagem com superior desempenho.

Diversas arquiteturas foram criadas com características distintas: máquinas com processadores trabalhado independentemente, outras com processadores executando as mesmas instruções, outras ainda trabalhando em *pipeline* onde, cada processador executa as mesmas instruções e passa os resultados para para o próximo, máquinas com processadores interligados em árvore, interligados com os vizinhos próximos, processadores organizados em forma de matriz, de anel etc.

Outra arquitetura atualmente em ascensão são as redes neurais. Elas são completamente distintas das arquiteturas mono ou multiprocessadas por não possuir um programa a ser executado e sim um conjunto de situações e soluções os quais são usados para o seu treinamento. Uma vez treinadas, entram em regime de operação onde são capazes de processar os dados e fornecer uma resposta satisfatória, com eficiência, mesmo quando estes dados são pouco condicionados (caso da imagem com ruído, por exemplo).

Nesta seção são apresentados sucintamente alguns sistemas com algumas das características acima mais que basicamente exploram paralelismo.

2.3.1 Matrizes de Processadores Binários

Esta técnica é baseada em uma matriz de elementos de processamento (PE's) idênticos, onde a imagem ou parte dela é distribuída ao longo da matriz e processada em paralelo. O esquema de comunicação geralmente adotado é o de interconexão com os vizinhos próximos (*near neighbor interconnections*) [Ree82].

A figura 2.2a mostra uma possível arquitetura para os elementos de processamento da matriz. O seletor de entrada atua como um filtro selecionando dados vindos de outros PE's, inclusive de suas memórias, ou da memória do próprio PE. O processamento desses dados é feito na unidade lógica aritmética que pode processar 2 ou mais operandos de 1 *bit*. Essas unidades também podem armazenar resultados intermediários em registradores internos de 1 *bit* e algumas arquiteturas incluem um registrador que mascara a atividade de processamento dos PE's.

Em geral os PE's podem se comunicar com os 4 PE's adjacentes mais próximos (fig. 2.2b). Variações tais como comunicação com os 8 ou 6 PE's adjacentes mais próximos são mostradas na figura 2.2c e 2.2d. Outras arquiteturas têm uma opção programável pelo usuário que permite selecionar um dos esquemas (comunicação com 4, 6 ou 8 PE's adjacentes mais próximos).

Sistemas com matrizes de processadores binários tais como CLIP4, DAP e MPP [Ree82] assemelham-se ao esquema mostrado na figura 2.3. Os dados são colocados e retirados serialmente da matriz via um *buffer* de memória de I/O, isto é, cada linha da matriz comporta-se como um registrador de deslocamento e a cada *clock* uma coluna é preenchida. As instruções são propagadas serialmente de um controlador micro-programado de alta velocidade, ao longo da matriz. A sincronização do sistema é controlada por um computador convencional que envia macro-instruções para o controlador, cuida do controle de armazenamento e recuperação em massa de dados e processa informações vindas de um extrator de informações globais.

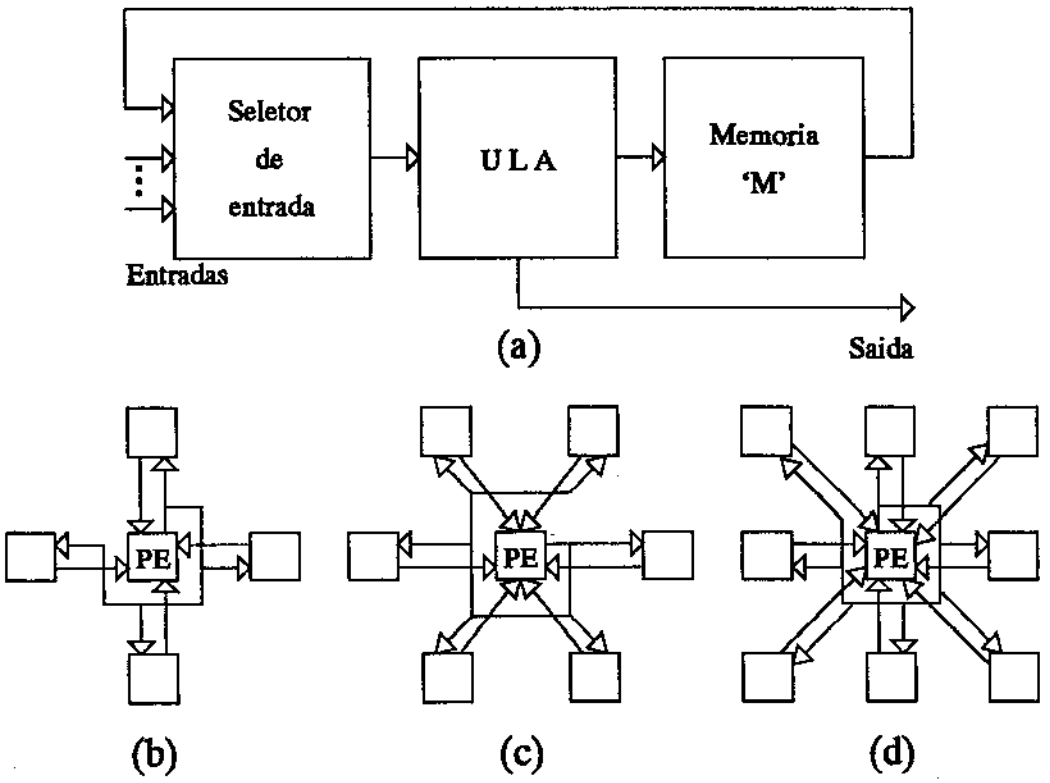


Figura 2.2: Matriz de processadores binários. Arquitetura de um PE (a) e esquemas de comunicação onde cada PE interconectam-se com os 4 (b), 6 (c) ou 8 (d) PEs adjacentes mais próximos.

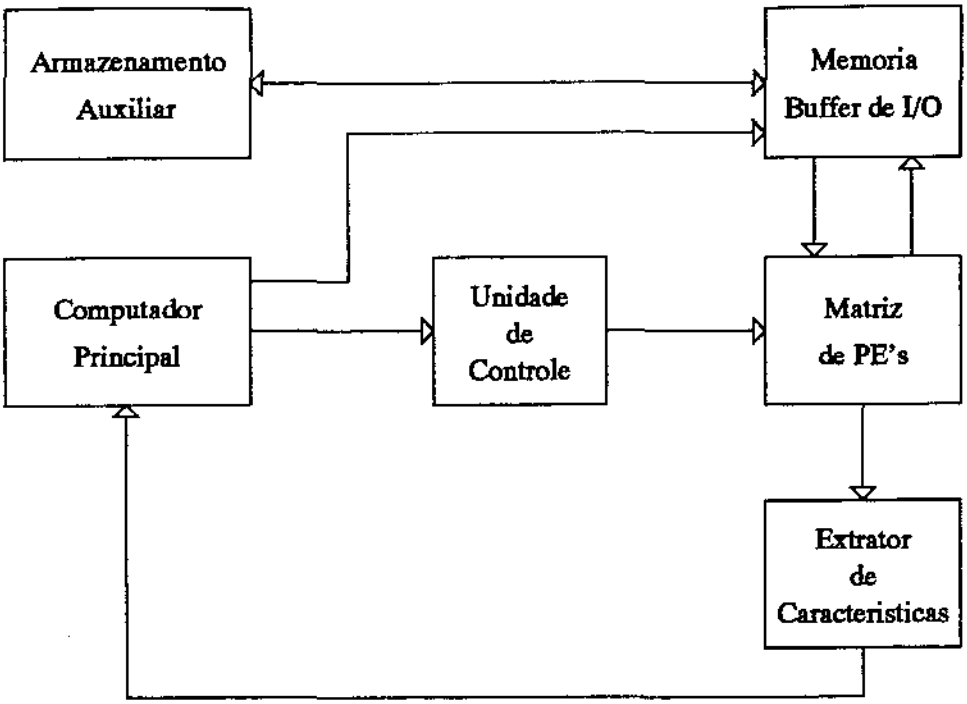


Figura 2.3: Organização de um sistema baseado em matriz de processadores binários

2.3.2 Systolic array

Systolic arrays são matrizes multidimensionais de idênticos processadores que fazem uso intensivo do mecanismo de *pipeline*. Tais estruturas são assim denominadas devido ao fato que o fluxo de dados entre os processadores, nas diferentes direções (ou dimensões), ocorre de maneira regular e rítmica. Os dados são processados na medida que são propagados através dos processadores. Cada processador contribui com pequenas operações e o processamento só é completado quando os dados saem dos processadores da periferia da matriz [Hay88]. *Systolic arrays* de duas dimensões mostram-se atraentes à implementações VLSI, podendo efetuar funções numéricas como multiplicação de matrizes e convolução em uma velocidade bastante elevada. Em geral, o aumento de velocidade é proporcional ao numero de estágios no(s) *pipeline(s)* da matriz.

Em [LF84] encontra-se uma aplicação típica para o uso de *Systolic array*. Trata-se da implementação do algoritmo de classificação pela mínima distância. A computação das distâncias entre os vetores X's de entrada (imagem desconhecida) e os padrões de referência R's é formulada como uma pseudo multiplicação de matrizes:

Da equação 2.2 tem-se:

$$Dp(X_j, R_i) = (|x_{j,1} - r_{i,1}|^p + |x_{j,2} - r_{i,2}|^p + ... + |x_{j,k} - r_{i,k}|^p)^{1/p}$$

Com $(i = 1, 2, ..., l)$ e $(j = 1, 2, ..., n)$. Onde n é o número de vetores a serem reconhecidos e l é o número de vetores de referências ou classes.

Como a extração da raiz p-ésima não muda o resultado da classificação, pode-se desprezar esta operação sem prejuízo para o resultado final:

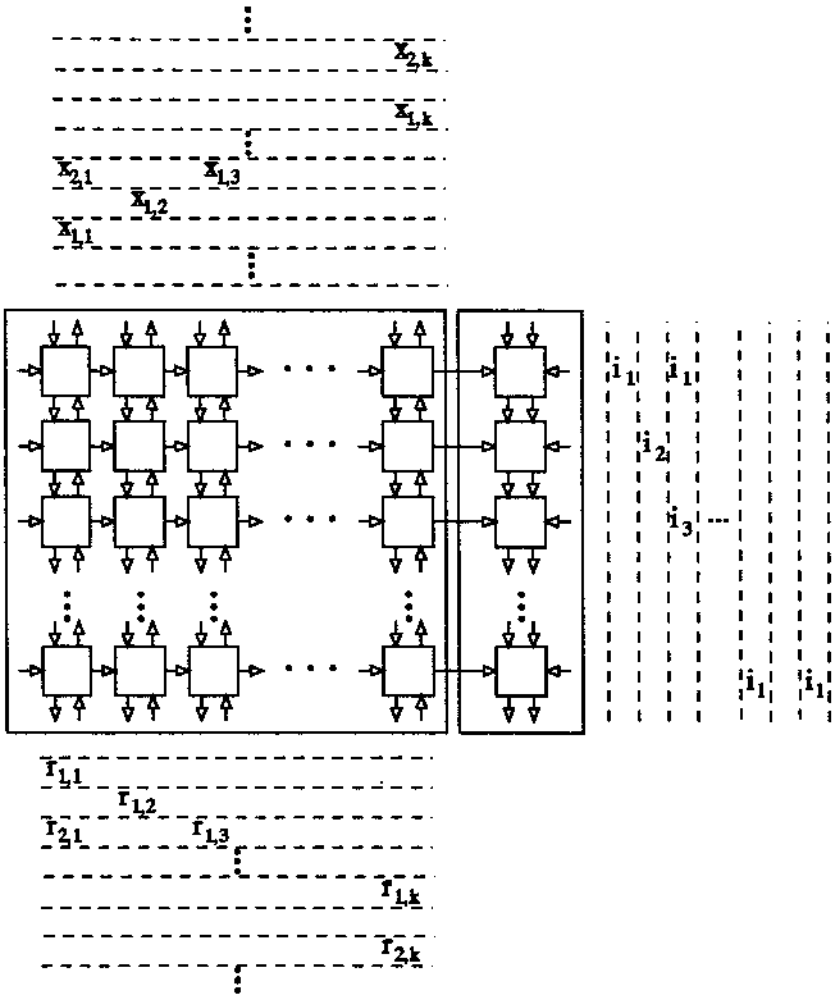


Figura 2.4: Organização do *systolic array* para o cálculo da distância mínima.

$$Dp(X_j, R_i) = |x_{j,1} - r_{i,1}|^p + |x_{j,2} - r_{i,2}|^p + ... + |x_{j,k} - r_{i,k}|^p$$

Considerando-se uma pseudo multiplicação de matrizes onde os elementos $Dp_{i,j}$ da matriz resultante são formados pela soma das diferenças entre os elementos da linha i da primeira matriz e os elementos da coluna j da segunda matriz, pode-se escrever todas as distâncias entre os vetores X's e R's como:

$$Dp_{k \times n} = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,k} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,k} \\ \vdots & \vdots & & \vdots \\ r_{l,1} & r_{l,2} & \cdots & r_{l,k} \end{bmatrix} \odot \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{n,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{n,2} \\ \vdots & \vdots & & \vdots \\ x_{1,k} & x_{2,k} & \cdots & x_{n,k} \end{bmatrix}$$

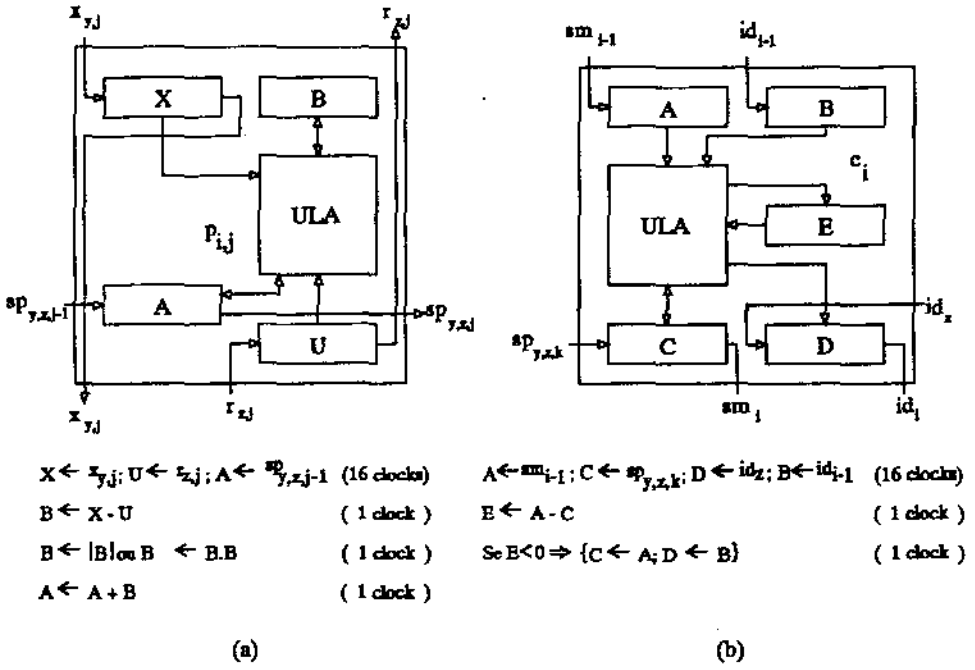


Figura 2.5: Arquitetura e micro-instruções dos processadores (a) e comparadores (b) do *systolic array*.

A figura 2.4 apresenta a estrutura do *systolic array* juntamente com a ordem de entrada dos dados (vetores de referências R's vetores desconhecidos X's e identificadores de classe I's). A matriz é formada de $l \times k$ processadores e l comparadores. Enquanto os processadores computam as distâncias, os comparadores escolhem as distâncias menores de modo a fornecer como resultado o identificador de classe conveniente. A figura 2.5 mostra a arquitetura dos processadores e comparadores. Os *processadores* consomem 16 *clocks* para propagar os componentes $x_{y,j}$, $r_{z,j}$, e a soma parcial $sp_{y,z,j-1}$ através dos registradores X, U e A respectivamente, 1 *clock* para efetuar a subtração $x_{y,j} - r_{z,j}$, armazenando o resultado no registrador intermediário 'B', 1 *clock* para o cálculo do modulo dessa subtração e mais 1 *clock* para adicionar este resultado à soma parcial recebida do processador da esquerda, propagando uma nova soma parcial para o processador da direita. Este conjunto de micro-instruções é usado no calculo de D_p com $p = 1$ mas este conjunto de micro-instruções pode ser facilmente alterado para o calculo de D_p com $p = 2$ (distância Euclideana), bastando para isso substituir o calculo do modulo, efetuado no registrador 'B', pelo calculo do produto de seu conteúdo com ele mesmo. A multiplicação pode ser feita em *hardware* paralelo, conservando o tempo de 1 clock, ou em multiplicador serial, onde neste caso seriam necessários 16 *clocks*. O comparador c_i recebe nos registradores 'A', 'B' 'C' e 'D' a soma mínima sm_{i-1} e respectivo identificador de classe id_{i-1}, a soma parcial $sp_{y,z,k}$ (que neste caso é a própria distância entre os vetores X_y e R_z) e o identificador de classe id_z respectivamente, em 16 *clocks*. Calcula a diferença entre a soma mínima e soma parcial, previamente recebidas, em 1 *clock* e usa esse resultado para decidir, em mais 1 *clock*, qual o novo par de soma mínima e identificador de classe deve ser propagado.

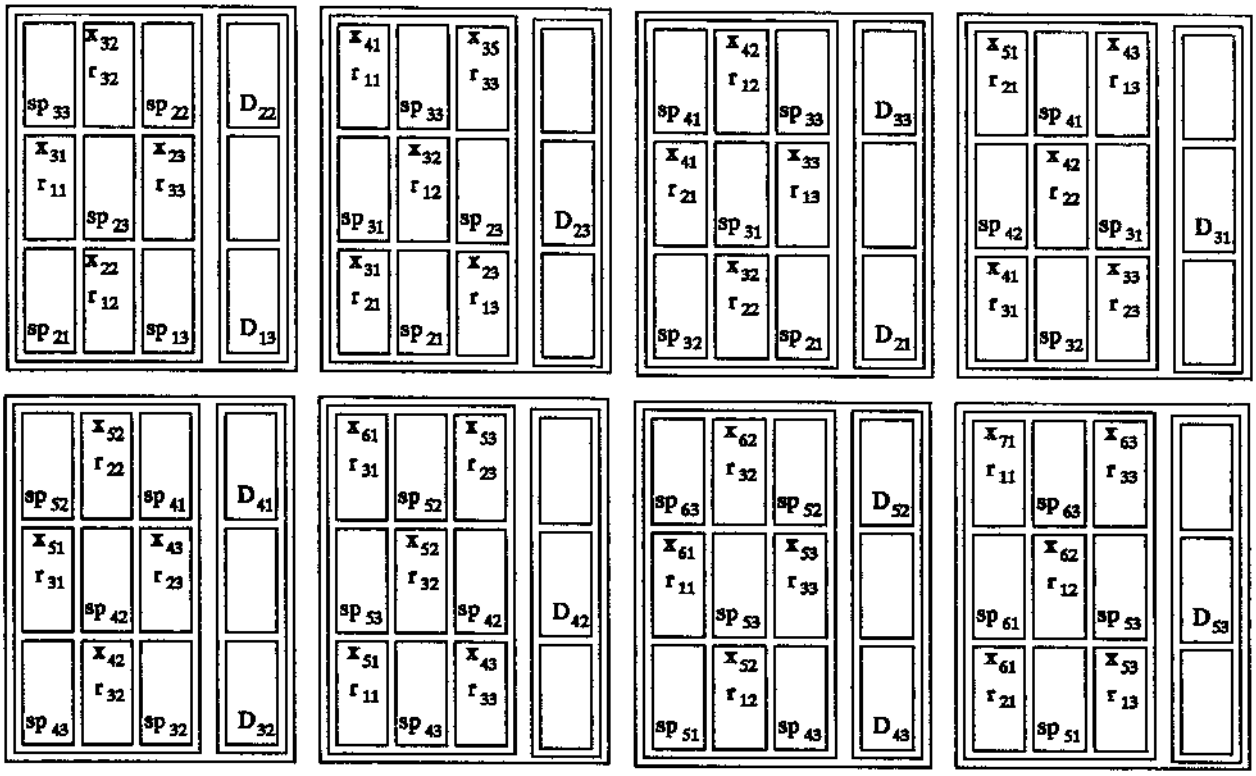


Figura 2.6: Estados de um *systolic array* de 3×3 processadores durante o processamento do vetor X_4.

A figura 2.6 apresenta uma seqüência de quadros com o conteúdo dos registradores internos dos processadores e comparadores do *systolic array* durante a computação das distâncias entre o vetor X_4 e os vetores R_1, R_2, R_3. Nota-se que a pseudo-multiplicação é calculada parcialmente por um processador que passa seu resultado parcial ($SP_{i,j}$) para o próximo processador formando um *pipeline*. O cálculo da distância mínima é finalizado quando uma soma parcial alcança a última coluna da matriz. Os resultados obtidos na última coluna de processadores são levados aos comparadores que também trabalham em *pipeline*, como evidencia a figura 2.6.

2.3.3 Pyramid machine

Pyramid machines [Tan82] são máquinas cujas arquiteturas estão estratificadas em camadas com funções bem definidas, ao contrário dos sistemas cuja arquitetura em uma única camada são projetados para propósitos gerais e usados na simulação de uma seqüência de camadas. O modelo nasceu da observação do processo de visão humana, o qual se dá através de níveis que começam na retina e estendem-se até o córtex visual.

A *Pyramid machine* é composta de uma série de camadas bidimensionais de dados intercaladas por transformadas que indicam como os dados são operados e movidos das camadas inferiores para às superiores. Tais transformadas podem estar operando simultaneamente entre seus pares de camadas. As camadas vão diminuindo gradativamente de tamanho na medida em que os seus

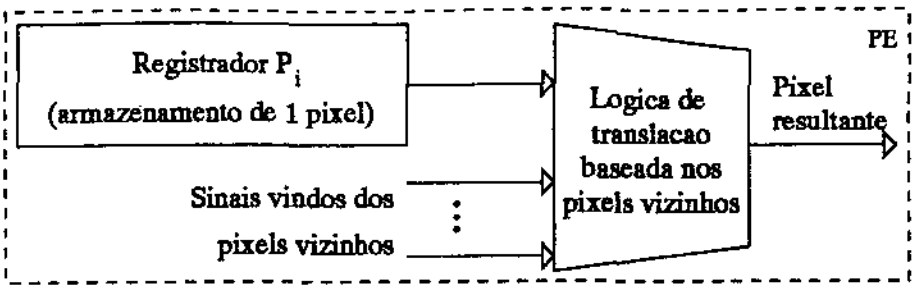


Figura 2.7: Estrutura de um PE de uma matriz de processadores paralelos usada em *neighborhood processors*.

níveis se elevam, convergindo para um ápice, de onde são retirados os resultados. Como exemplo de transformadas tem-se as operações lógicas *AND*, *OR* e *XOR* (para imagens binárias), operações aritméticas como média, mínimo e máximo, detenção de características locais com casamento de padrões, etc.

Máquinas com esta estrutura podem facilitar a confecção, e executar com maior eficiência, algoritmos que usam estratégias tais como:

- **Planning:** Uma versão reduzida da imagem original é criada e analisada. O resultado desta analise é usado para orientar posteriores análises na imagem original.
- **Dividir e conquistar:** O problema é dividido em pequenos problemas e cada subproblema é resolvido recursivamente e um algoritmo de combinação é usado para resolver o problema original.
- **Busca em árvore:** Pontos característicos particulares, como o canto superior esquerdo de um retângulo em uma imagem, podem ser localizados em um tempo de $O(\log N)$, onde 'N' é o número de linhas da imagem.
- **Análise local e/ou global:** Análises baseadas em operações de *neighborhood*, tal como em detenção de arestas, podem ser efetuadas com diferentes raios de ações, usando dados locais ou mais distantes de uma região.

2.3.4 Neighborhood processors

Neighborhood processors [Ste82] são dispositivos que processam uma matriz de dados (Imagem) produzindo outra matriz onde cada elemento é gerado em função do seu correspondente na matriz de entrada e seus vizinhos próximos.

Uma implementação possível são as matrizes de processadores paralelos, onde cada processador elementar (PE) corresponde a um *pixel* da imagem. As interconexões entre os PE's são semelhantes às vista nas figuras 2.2b, 2.2c e 2.2d. A arquitetura do PE é mostrada na figura 2.7. Existe um registrador destinado ao armazenamento do *pixel* e uma lógica de translação baseada nos vizinhos próximos ao *pixel*. Essa lógica pode ser fixa ou programável pelo usuário. O processamento é efetuado em um único *clock* com todos os *pixels* da matriz resultante sendo calculados simultaneamente.

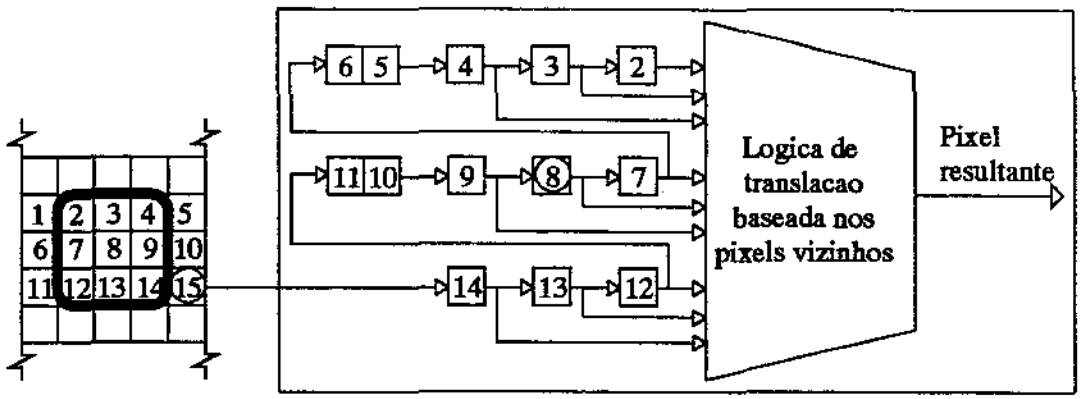


Figura 2.8: Estrutura de um PE de uma matriz de processadores seriais

Outra implementação possível são os vetores de processadores seriais, os quais são formados por uma cadeia de vários estágios de processadores seriais. A figura 2.8 apresenta uma estrutura possível para estes estágios. Os pixels são colocados serialmente numa fila de registradores que proverão a lógica de translação. A cada pulso de clock os dados da fila são deslocados para frente e um novo pixel é colocado no ultimo registrador da fila. O registrador mediano (localizado no meio da fila) fornece sempre o pixel a ser processado enquanto os outros registradores fornecem seus vizinhos próximos. Cada estágio processa um pixel a cada clock. No caso da figura 2.8, o *pixel* '8' e seus vizinhos '2', '3', '4', '7', '9', '12', '13' e '14' (todos contidos no quadrado em negrito) estão sendo processados pela lógica de translação, que pode ser previamente programada para efetuar uma operação específica. No próximo pulso de *clock* todos os pixels da fila são deslocados para frente e o *pixel* '15' é introduzido, possibilitando a lógica de translação processar o *pixel* '9' e seus vizinhos '3', '4', '5', '8', '10', '13', '14' e '15'.

Uma variação entre as duas implementações vistas é a matriz de processadores seriais particionada em paralelo. Neste caso dois ou mais processadores seriais dividem o processamento da matriz de dados de tal forma que cada processador fica responsável por n/p colunas da matriz, com 'n' sendo o número de colunas da matriz e 'p' o número de processadores. A figura 2.9 Mostra o caso para $n = 10$ e $p = 2$. Os processadores são semelhantes ao anterior (figura 2.8) com exceção da inclusão de alguns multiplexadores que fornecem ao bloco de translação, através de conexões entre os processadores, os vizinhos próximos ao pixel sendo processado mesmo quando estes se encontram na fronteira entre as duas metades da matriz. Esta versão processa 'p' pixels a cada pulso de clock.

2.3.5 Redes neurais

Uma rede neural (**rn**) é constituída de várias unidades de processamento simples conhecidas como 'neurônios', os quais se inter-conectam de alguma maneira entre si e com o mundo externo [Red88]. Os neurônios efetuam uma função sobre seus sinais de entrada para produzir um sinal de saída. Tais funções são dependentes do estado de uma memória local presente nos neurônios.

Genericamente **rns** são usadas como memória associativa (*associative recall*) [TH87], onde uma imagem previamente vista pode ser reconhecida mesmo quando uma versão parcial da imagem é apresentada, e para efetuar uma função através da generalização, isto é, para calcular a imagem de

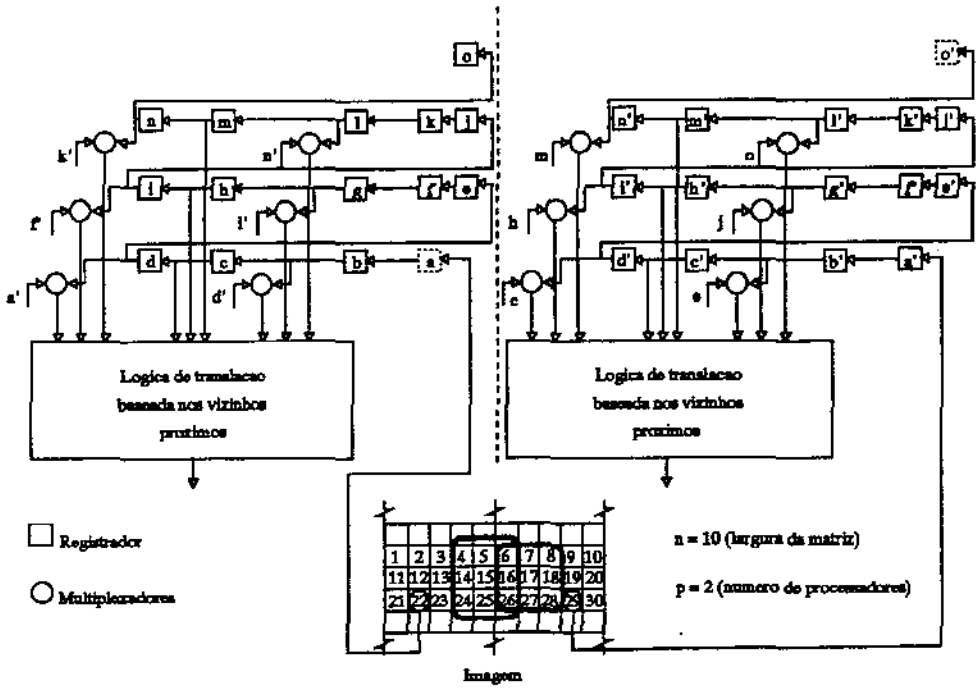


Figura 2.9: Estrutura dos PE's de uma matriz de processadores seriais particionada em paralelo

uma função em qualquer ponto de seu domínio através da generalização baseada em apenas alguns pontos apresentados na fase de treinamento. Em ambos os casos a rede deve ser treinada sobre um conjunto de exemplos e respostas requeridas, as quais serão fornecidas posteriormente (na fase de operação) pela rede.

Uma variante das **rns** são as redes lógicas probabilísticas (**PLN**). Elas usam modelos de neurônios baseados em uma espécie de RAM (*Random Access Memory*) com capacidade de armazenar um de três estados: '1', '0' e 'U' (para *Unknown* ou estado desconhecido). Tais RAMs são chamadas de GRAM (*Generic RAM*) e o estado 'U' produz em suas saídas '0' ou '1' com igual propabilidade. Em [AM91] é apresentada uma unidade neural com capacidade de generalização (GNU) que pode restaurar e/ou reconhecer imagens previamente treinadas. A figura 2.10 mostra a estrutura do GNU e um par de imagens (de entrada e resultante) como exemplo. O GNU é constituído de *'k'* GRAMs de $n = i + a$ entradas e uma saída, sendo *'i'* entradas ligadas aos *pixels* da imagem de entrada, de maneira aleatória porem fixa, e *'a'* entradas destinadas aos laços de realimentações.

Antes do treinamento todas as GRAMs iniciam em 'U'. Durante o treinamento algumas posições de memória endereçadas por um subconjunto da imagem são levadas a '1' e outras a '0'. Devido aos laços de realimentação, existe um processo de generalização onde algumas posições de memória que estão em 'U' são levadas ao mesmo estado que outras posições de memória com endereços semelhantes cujos estados são diferentes de 'U'. Posições de memória com endereços semelhantes que devido ao treinamento foram levadas a estados conflitantes são convertidas de volta para o estado 'U'. Após o treinamento o GNU pode entrar em operação. Uma imagem apresentada pode fazê-lo convergir para uma das imagens treinadas. Lucy apresenta em [Luc91] uma implementação

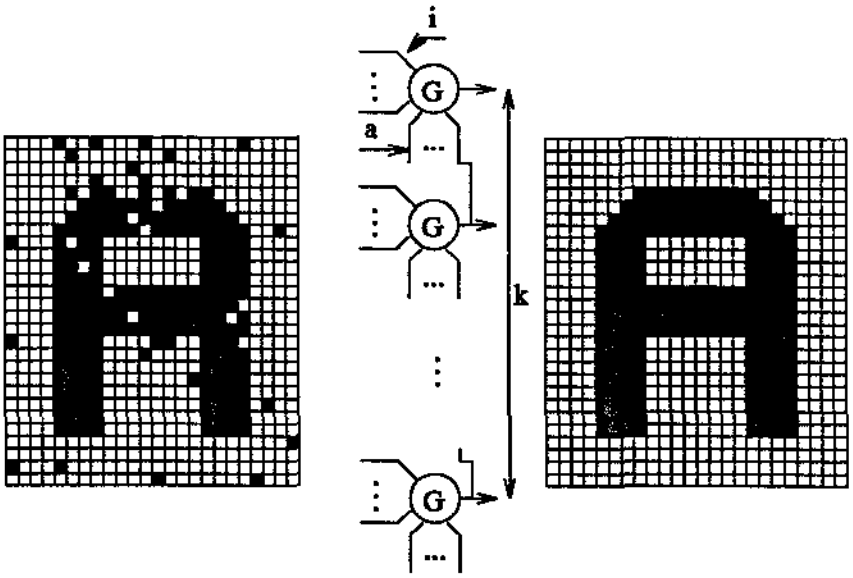


Figura 2.10: Unidade neural com capacidade de generalização

baseada nestes mesmos princípios mas ao contrario do GNU, onde todas as GRAMs são atualizadas de modo síncrono com um pulso de *clock*, a implementação de Lucy converge para uma resposta, atualizando uma GRAM a cada iteração.

A figura 2.11 apresenta uma série de imagens referentes a três padrões. Na primeira coluna (figura 2.11a) estão as imagens com as quais o GNU foi treinado. Na segunda coluna tem-se versões distorcidas por ruído, das imagens originais, que são apresentadas à rede. Nas colunas (c), (d) e (e) estão as respostas da rede para os casos $i = 1$, $i = 8$ e $i = 4$ respectivamente. No primeiro caso, quando o número de entradas de cada GRAM diretamente ligas aos *pixels* da imagem é $i = 1$, a rede é incapaz de recuperar corretamente os padrões apresentados. No segundo caso, $i = 8$, a rede recupera os padrões mas sua resposta apresenta um certo conteúdo de ruído. No caso $i = 4$, a rede recupera corretamente os padrões, produzindo respostas praticamente sem ruídos.

2.4 Análise das abordagens

O paralelismo encontrado na maioria das arquiteturas não convencionais confere a tais máquinas superioridade em desempenho, quando comparadas às máquinas convencionais. Toma-se como exemplo o método de classificação pela mínima distância: quando executado em uma arquitetura mono-processada, sob uma imagem de n *Pixels* e l classes, são necessárias $l \times n$ subtrações, $l \times (n-1)$ adições, $l \times n$ exponenciações e l radiciação totalizando um custo de $\bigcirc(n \times l)$. Num *systolic array*, considerando uma operação como a execução do conjunto de micro-instruções do processador ou comparador, são necessárias $l - 1$ operações para inicialmente posicionar os padrões de referências na matriz, n operações para o cálculo da distância com referência ao primeiro padrão de referência, 1 operação para cada um dos outros padrões de referências $(l-1)$, e uma comparação para classificar a imagem com respeito a última classe. Isto significa um custo de $\bigcirc(n)$. Mas quando um conjunto

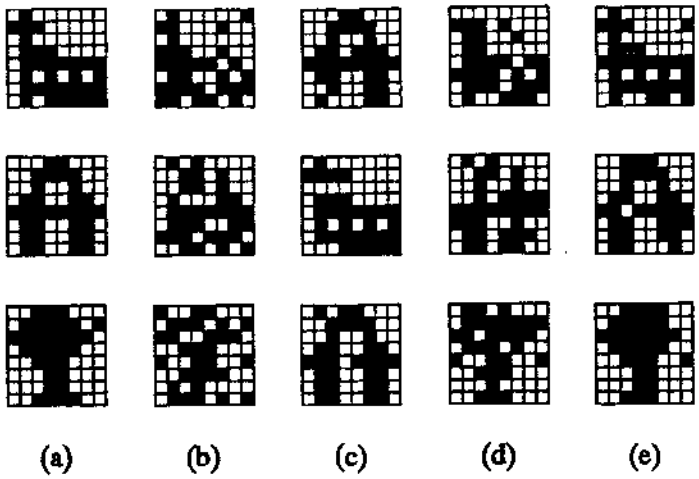


Figura 2.11: Reconhecimento de três padrões com $k = 64, n = 8$. (a) Imagem de treinamento, (b) Imagem de entrada com ruído, (c) Típico erro de recuperação com $i = 1$, (d) Recuperação com ruído para $i = 8$ e (e) recuperação ótima com $i = 4$

de k imagens tem que ser processado, as operações iniciais podem ser desprezadas e quando a matriz já tem seu *pipeline* preenchido, uma imagem é processada a cada 2 operações. Neste caso toda potencialidade da matriz é usada obtendo um acréscimo de velocidade de $l \times n/2$. Este impressionante aumento de velocidade é alcançado devido ao uso intensivo de paralelismo na matriz.

Método ou arquitetura	Custo ou desempenho
Classificação por distância mínima	$3ln$ ou $\bigcirc(ln)$
Função de correlação	$(8n + 11)l$ ou $\bigcirc(ln)$
Matriz de processadores binários	$(l + 1)\sqrt{n} + k$
Systolic array	$2l + d$
Pyramid machine	$l + \log_b n$
Neighborhood processor	$ln/p + d$

Tabela 2.1: Resumo apresentando o custo dos principais métodos e arquiteturas para um conjunto de l classes e uma imagem de n *pixels*(ver texto).

A tabela 2.4 apresenta o custo e desempenho dos principais métodos e arquiteturas citadas. Para os dois primeiros métodos, MDC e função de correlação, foi assumido que todas as operações têm um mesmo custo (custo médio). Na matriz de processador binário foi considerada uma matriz de $\sqrt{n} \times \sqrt{n}$ processadores e que o custo para propagar os vetores de referências e a imagem foi de $(l + 1)\sqrt{n}$. O termo k representa o custo para propagar as instruções na matriz, o qual não pôde ser estimado por falta de detalhamento, no artigo que a descreve, de como essas instruções são propagadas. O *systolic array* consome d operações para posicionar os primeiros componentes dos vetores e da imagem na matriz. Como a cada 2 operações ele processa uma classe, são necessárias

mais $2l$ operações para às l classes. A guisa de comparação, foi idealizada uma *pyramid machine* hipotética com n processadores em sua base e um custo de propagação entre camadas adjacentes de 1 operação. A altura da pirâmide, e por tanto o número de camadas, é dada por $h = \log_b n$, onde b é o grau de conectividade entre as camadas, por exemplo, $b = 2$ para uma pirâmide onde cada processador de uma camada está conectado com 2 processadores da camada imediatamente inferior. Neste caso o custo de propagação entre a base e o ápice é de $\log_b n$ operações. Como as camadas da pirâmide podem ser vistas como um *pipeline*, o custo total para as l classes é de $l + \log_b n$ operações. A versão serial particionada em paralelo para os *neighborhood processors* apresenta um custo p vezes menor que os computadores convencionais, desprezando-se o tempo inicial d para carregar a fila de registradores. O *systolic array* apresente o melhor desempenho, não só pelo grande número de processadores em sua matriz (nl), mas por processar dados de uma maneira que não necessita esperar por uma completa propagação dos dados. Os dados são processados à medida que são propagados.

Um outro parâmetro importante a ser analisado é o custo dos processadores nas arquiteturas. Uma máquina mono-processada por ter um único processador pode ter um processador complexo e de baixo custo. Com relação principalmente as matrizes de processadores, onde o número de processadores gira em torno de centenas de milhares, seus processadores elementares têm que ser o mais simples e ocupar a menor área possível de silício. Uma abordagem que visa minimizar estes custo foi adotada em [Ste82] com as matrizes de processadores seriais particionada em paralelo. De acordo com as necessidades de processamento exigidas pode-se usar um número adequado de estágios de processadores seriais trabalhando em paralelo, buscando-se uma solução de compromisso entre desempenho e custo.

Um problema decorrente do número e tamanho (ou complexidade) dos processadores é a impossibilidade de integrar todos os processadores em um único *chip*. Muitas vezes, a quantidade de interconexão entre os processadores inviabiliza a construção do sistema em um conjunto de *chips* devido à limitação de pinos dos mesmos. Uma solução adotada é a serialização dos sinais. Por exemplo, o *systolic array*, descrito acima, efetua transferências de dados entre seus processadores de modo serial, substituindo um barramento com 16 interconexões por uma única interconexão. Isto implica em um tempo de comunicação inter-processadores 16 vezes maior, mas devido ao alto grau de paralelismo conseguido no *pipeline* esse tempo não prejudica o desempenho da matriz.

Redes neurais são sistemas bastante distintos dos sistemas mono e multiprocessados. A computação ocorre através de rápidas mudanças nos estados dos neurônios enquanto que o algoritmo ou o "conhecimento" é encapsulado na maneira com que os neurônios interagem entre si. Sua principal vantagem sobre os outros sistemas é a capacidade de tratar dados pouco condicionados (como por exemplo nas imagens corrompidas por ruído). Como foi visto, uma imagem, mesmo que bastante distorcida, pode ser recuperada devido ao poder de generalização de uma **rn**.

Ruídos, produzidos quando uma câmara de TV apresenta pequenas vibrações entre imagens adjacentes, pode exigir um pré-processamento dos dados. Em [Ree82] é apresentada uma solução para esse problema usando *image registration* em uma matriz de processadores binários. Dependendo do número de imagens a serem processadas, esse pode ser um problema sério para as máquinas multiprocessadas. Neste ponto é que as **rns** se destacam. Elas toleram bem esse tipo de problema mantendo sua eficiência a um custo, relativamente aos outros sistemas, baixo.

Mas as **rns** também possuem seus próprios problemas. No campo da implementação, diversas abordagens têm surgido tentando otimizar parâmetros como área de silício, velocidade de operação, precisão dos modelos etc. O próximo capítulo detalha melhor o assunto e discute esse tipo de

problema.

Capítulo 3

Os Sistemas Neurais

3.1 Introdução

Os sistemas nervosos biológicos são compostos por complexas redes de células especializadas chamadas de neurônios. Eles são a unidade básica de processamento do sistema. Embora estes sistemas sejam muito diferente de um computador (ver tabela 3.1) podemos identificar algumas semelhanças:

- Ambos processam informação.
- Sinais são representados por diferença de potenciais elétricos.
- Os sinais são conduzidos por "fios" formados por um tecido condutor de eletricidade rodeado por um bom isolante elétrico.
- Existem dispositivos ativos que respondem a uma variação de potencial elétrico em sua entrada, com uma variação de corrente na sua saída.
- Os dispositivos ativos apresentam um ganho de energia (amplificação) em relação aos seus sinais de entrada e saída, o qual é usado como uma função primaria no processamento de informações.
- Os dispositivos funcionam através de um mesmo princípio físico (o controle de uma finíssima barreira de energia.
- Ambos possuem elementos de memória que retêm a informação ou parte dela durante o processamento.

3.2 Sistemas neurais biológicos

3.2.1 O neurônio, Constituição e Funcionamento

O neurônio é a unidade básica de processamento dos sistemas nervosos. O esquema da figura 3.1 mostra suas partes componentes.

Os dendritos são ramificações filamentosas onde se encontram as sinapses, sua função é conectar, através das sinapses, um neurônio a outros. Um único neurônio do cérebro humano possui entre

Aspectos	*Sistemas nervosos biológicos*	*Computadores*
Constituição	São formados de células organizadas num espaço tridimensional	São construídos de material inorgânico rígido, numa organização bidimensional
Alimentação	São alimentados através de metabolismo bioquímico	São alimentados por uma fonte principal através de transformadores e retificadores
Sinais	Impulsos elétricos de aproximadamente 100 mV, durando 1 a 2 ms	Impulsos elétricos de 5 volts com períodos da ordem de nanosegundos
Robustez	A destruição de algumas células não irá causar notável degradação no desempenho	A perda de um simples transistor (a não ser em suas memórias) pode causar perda completa de funcionalidade
Consumo	Células nervosas típicas dissipam potência da ordem de 10^{-12} w	Portas lógicas típicas dissipam 10^7 vezes mais potência que células nervosas
Aplicação	Aceitam entradas pouco condicionadas, fazem uma computação que é mau definida e dão respostas aproximadas	São extremamente eficientes em produzir respostas precisas para questões bem definidas

Tabela 3.1: Principais aspectos que diferenciam os sistemas nervosos naturais de um computador (sumarizado de [Mea89]).

1000 a 10000 conexões sinápticas ([RR91], página 84). O corpo da célula possui um núcleo que provê, através de mecanismos bioquímicos, energia para o funcionamento do neurônio. O axônio é responsável pela digitalização e transmissão de sinais elétricos gerados na célula, para outros neurônios. O axônio é envolvido por um material isolante chamado mielina para evitar a degradação do sinal que ele transporta. A cada poucos milímetros existem buracos na mielina denominados de nós de Ranvier que agem como regeneradores do sinal. Um axônio assim descrito pode transportar sinais a uma distância de 1 metro ou mais [Mea89].

Os pulsos de corrente elétrica de outros neurônios, obtidos pelos dendritos são integrados através do armazenamento de cargas na capacitância da membrana da célula até que uma certa tensão de limiar é atingida, quando um pulso é gerado e propagado pelo axônio até uma estrutura ramificada de contatos sinápticos, para excitar os dendrites de outros neurônios.

A membrana plasmática e sua função. A membrana do neurônio é formada por moléculas de fosfolipídeos dispostas em 2 camadas interligadas por 2 cadeias hidrocarbônicas. A diferença entre a capacidade de polarização das cadeias hidrocarbônicas e do meio aquoso em que a membrana se encontra resulta numa barreira de energia que torna a membrana um perfeito isolante elétrico, impedindo a passagem de íons através da membrana. Para existir um fluxo de corrente (fluxo de íons) através da membrana é necessário um outro mecanismo. Esse mecanismo se dá através de substâncias neurotransmissoras lançadas na região sináptica pelo neurônio excitador. Os neurotransmissores abrem canais seletivos na membrana do neurônio excitado, no qual se tem um fluxo de um tipo específico de íons que irá mudar a carga na capacitância da membrana.

Fonte de Suprimento e Ganho de Energia Requerido ao Processamento. Através da manipulação dos neuro-transmissores os sistemas biológicos obtêm ganho de energia requerido ao processamento do sinal. Assim como nos circuitos eletrônicos, para se ter ganho de energia é

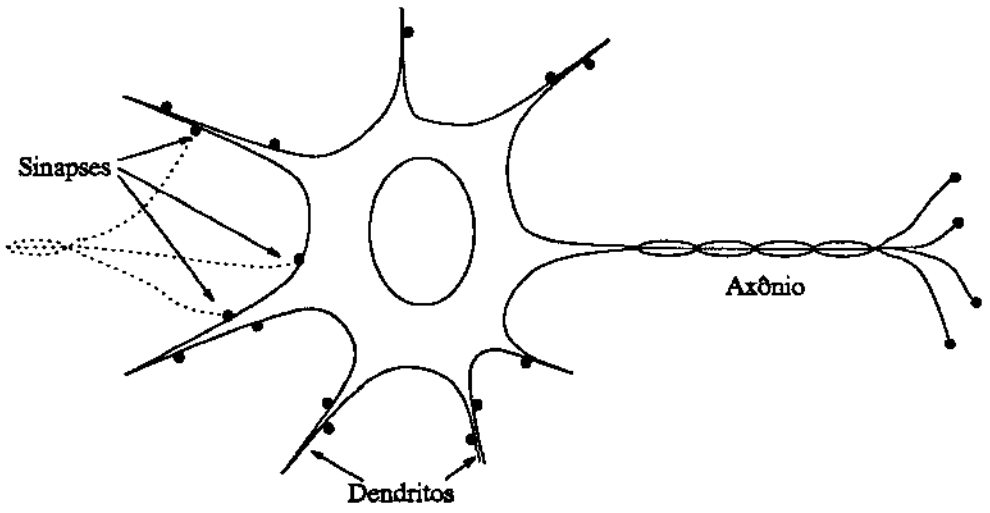


Figura 3.1: Esquema da constituição de um neurônio

necessária uma fonte de suprimento. Nas células, esta fonte é obtida através do mecanismo chamado de bomba de sódio, que expele íons de sódio para fora da célula enquanto absorve íons de potássio. A diferença de concentração destes íons, dentro e fora da célula, é usada como fonte de suprimento das atividades elétricas na membrana.

A figura 3.2 apresenta o potencial na membrana de um neurônio excitado por um pequeno pulso de corrente. Para pulsos de corrente que resultem em uma redução do potencial da membrana abaixo de aproximadamente 30 mV com relação ao seu potencial de repouso (-80 mV neste caso), o neurônio responde com um potencial que satura rapidamente, voltando ao potencial de repouso tão logo o pulso de corrente seja inibido. Para pulsos de corrente com maior amplitude, há um aumento exponecial do potencial, culminando na geração de um pulso. Se o pulso de corrente terminar antes que a membrana alcance aproximadamente -40 mV, a membrana recupera seu estado original e nenhum pulso é gerado. Mas se o pulso de corrente for suficiente para elevar o potencial para mais que aproximadamente -40 mV, um pulso é gerado mesmo que o pulso de corrente termine imediatamente.

3.2.2 Topologia das Redes Naturais

Os neurônios, nos sistemas nervosos biológicos, se organizam ou se interconectam de acordo com suas funções. O seu poder de processamento como um elemento isolado é pequeno. Mas quando milhões de neurônios se organizam em um sistema complexo, com laços de realimentações, e trabalhando numa forma altamente paralela, esse poder de processamento torna-se algo insuperável face aos mais potentes computadores já construídos. A maior dificuldade para construir uma máquina inspirada nos sistemas naturais e com considerável poder de processamento é a compreensão destes sistemas como um todo, com uma grande diversidade de maneiras com que os neurônios possam interagir.

Um exemplo, que tem levado alguns pesquisadores à implementações inspiradas nos sistemas biológicos, é o sistema de aquisição de imagens dos animais (veja "Exemplos de Implementações" no fim deste capítulo). Na retina da maioria dos animais encontra-se algumas variedades de células,

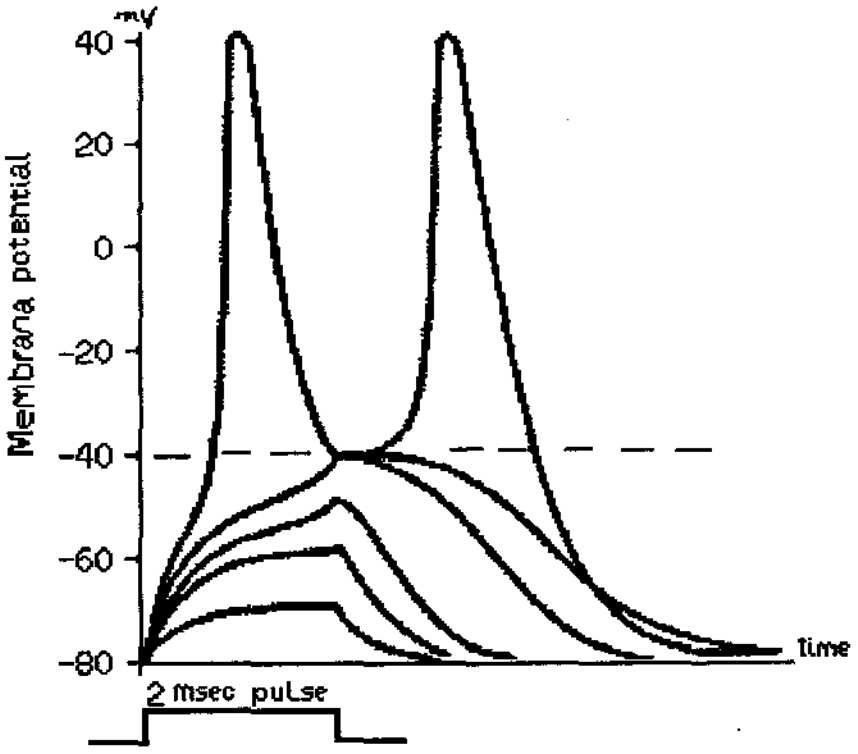


Figura 3.2: Resposta de um neurônio quando excitado por um pulso de corrente

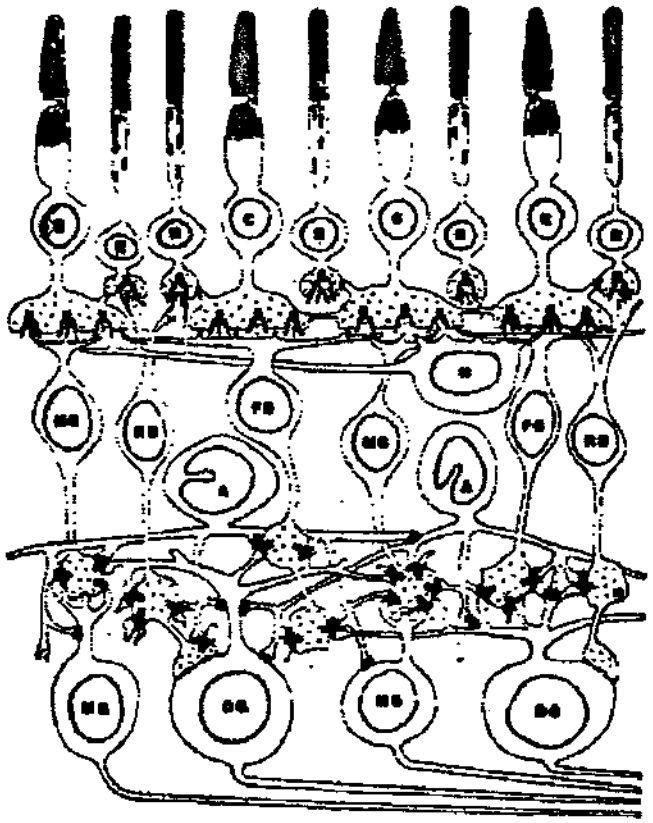


Figura 3.3: Esquema de um corte transversal da retina de um primata.

cada uma delas com funções específicas e formatos convenientes, replicadas em grande número. A descrição a seguir detalha em linhas gerais a estrutura de uma retina animal.

A Estrutura da Retina. Embora os detalhes na anatomia da retina [Ove76] [Mea89] sejam particulares em cada animal, sua estrutura, a grosso modo, tem se conservado entre os vertebrados. Ela forma um importante sistema que é um exemplo claro da potência dos sistemas neurais vivos. Esse sistema é responsável pela coleta e preprocessamento de toda informação visual que chega ao cérebro. A relativa independência destas informações com o nível de iluminação do ambiente e distância da cena são exemplos do preprocessamento realizado pela retina.

Primeiro a luz é focalizada, através da córnea e do cristalino, sobre a superfície curva do fundo do olho chamada de retina. O cristalino é uma lente natural cujo ponto focal é ajustado à medida que músculos ópticos controlam sua curvatura. Assim, objetos próximos ou distantes podem ser devidamente focalizados. A quantidade de luz que chega até a retina é controlada por um diafragma conhecido como íris. Seu diâmetro pode variar entre 1,5 a 8 mm, deixando uma pequena ou grande quantidade de luz alcançar a retina.

Depois, a luz é transformada em um sinal elétrico pelos foto-receptores (C e R) localizados na superfície da retina (figura 3.3). Existem dois tipos de células fotorreceptoras: os cones e os bastonetes. Os cones (C) são responsáveis pela percepção das cores e subdividem-se em células com

diferentes sensitividade espectral (de 2 à 5 tipos). Os bastonetes (R) são mais sensíveis e possuem apenas um tipo de resposta espectral (visão em preto e branco). Devido a sua alta sensibilidade, eles funcionam bem durante a noite, quando pouca luz incide sobre a retina. Os cones concentram-se na região central da retina, enquanto que os bastonetes, na periferia.

O fluxo principal desse sinal (vindo dos cones e bastonetes) passa pelas células bipolares (FB, MB e RB) através de sinapses triplas e chega até às células gânglios (MG e DG), as quais transmitem os sinais até os lobolos do córtex visual através de seus axônios. Cada tipo de célula da retina tem características e funções próprias. As células horizontais (H), junto com as fotorreceptoras e bipolares, representam sinais analógicos com variações lentas. As amacrinas estão envolvidas com eventos de movimentos de objetos na imagem. As gânglios possuem longos axônios que produzem pulsos *quasidigital* (sinal digital em amplitude mas não no tempo), transportando informações de modo mais seguro que por meio de sinais analógicos.

3.3 Modelagem de Sistemas Neurais

Os dispositivos que modelam os neurônios (chamados de *Neuromimes* em [Mac87]) podem ser classificados em:

- Simuladores em computadores digitais
- Dispositivos eletrônicos analógicos e digitais
- Modelos matemáticos
- Modelos estocásticos

A primeira classe, os simuladores, provavelmente são os mais práticos, flexíveis, adaptáveis e acessíveis pela maioria dos pesquisadores. Os dispositivos eletrônicos analógicos e digitais são atualmente restritos à pesquisadores da área de engenharia elétrica. Sua utilização requer laboratórios eletrônicos e conhecimentos específicos na área. Sua principal vantagem é uma alta velocidade, relativa aos simuladores, no processamento dos sinais de entrada gerando respostas com menor tempo de atraso. Acredita-se que o desenvolvimento de circuitos integrados VLSI dedicados pode levar à sistemas neurais significativamente maiores e velozes devido o alto grau de integração dessa tecnologia. A popularização de *chips* neurais comerciais (a exemplo do 80170NX da Intel [TWH93]) poderá ampliar a gama de usuários de tais dispositivos.

Modelos matemáticos são menos flexíveis e adaptáveis que simuladores digitais, e são restritos a pesquisas com enfoque matemático. Modelos estocásticos tentam descrever o comportamento dos neurônios através da teoria de sistemas estocásticos. O principal objetivo deste tipo de modelo é prever o intervalo de distribuição no espaçamento entre os estados ativos do modelo (*inter-spike*) em função de parâmetros de entrada e da dinâmica da membrana. Estes modelos usam teorias matemáticas acuradas que são complicadas e não tem levado ainda à uma solução geral clara.

A modelagem dos sistemas neurais pode ser dividida em duas partes. Na primeira parte é dado enfoque ao neurônio, onde formulam-se modelos capazes de imitar o comportamento das células nervosas biológicas. Na segunda parte o foco é dirigido à organização dos neurônios que formam uma rede, onde discute-se problemas referentes a sua topologia.

Genericamente, um modelo é composto por n entradas s_i e uma saída s_j, uma regra de propagação e uma função de ativação. A regra de propagação determina como os sinais de entrada

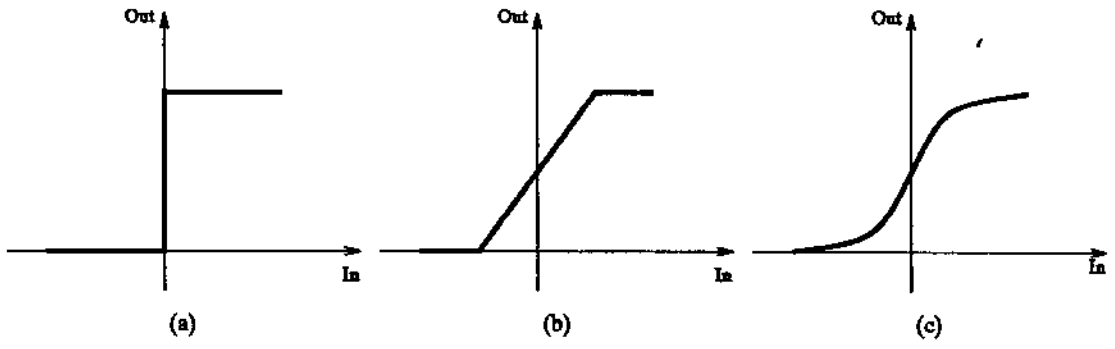


Figura 3.4: Exemplos de funções de ativação. Degrau (a), semi-linear (b) e *sigmoid* (c).

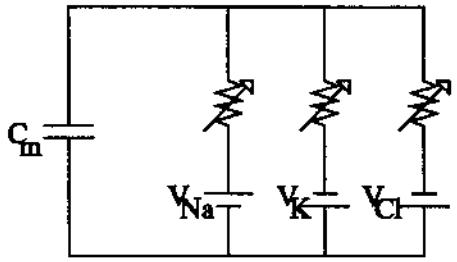


Figura 3.5: Modelo de Eccles de um neurônio.

serão computados, estabelecendo o grau de influência que cada sinal de entrada terá no estado de ativação do modelo. A função de ativação determina o novo estado do modelo baseando-se na regra de propagação. A regra de propagação mais comum é apresentada na equação 3.1, onde $w_{i,j}$ são chamados de pesos sinápticos e $\emptyset$ de *off set* ou *bias*. As funções de ativação mais usadas são: degrau, semi-linear e *sigmoid* (figura 3.4).

$$Pr_j = \emptyset + \sum_{i=1}^{n} s_i \times w_{i,j} \tag{3.1}$$

Os primeiro modelos eram implementados com dispositivos analógicos e eram inspirados nas atividades bioquímicas das células neurais vivas. Modelos subseqüentes, usando dispositivos analógicos e/ou digitais, surgiram dando mais importância à funcionalidade. As seções a seguir abordam alguns desses modelos. Ecles formulou seu modelo baseando-se nos fenômenos eletroquímicos da membrana de um neurônio, Nagumo usou um diodo túnel em seu modelo, McCulloch e Pitts formulou um modelo com interesse em operações lógicas e cognitivas e por ultimo um modelo simplificado de neurônio usando pequenas memórias RAMs é apresentado.

3.3.1 O modelo de Eccles

Uns dos mais antigos modelos do neurônio teve origem na tentativa de descrever as atividades elétricas na membrana e constituiu-se num marco entre as pesquisas da área. O modelo de John Eccles [Mac87] visto na figura 3.5 baseia-se nos fenômenos eletroquímicos da membrana. O

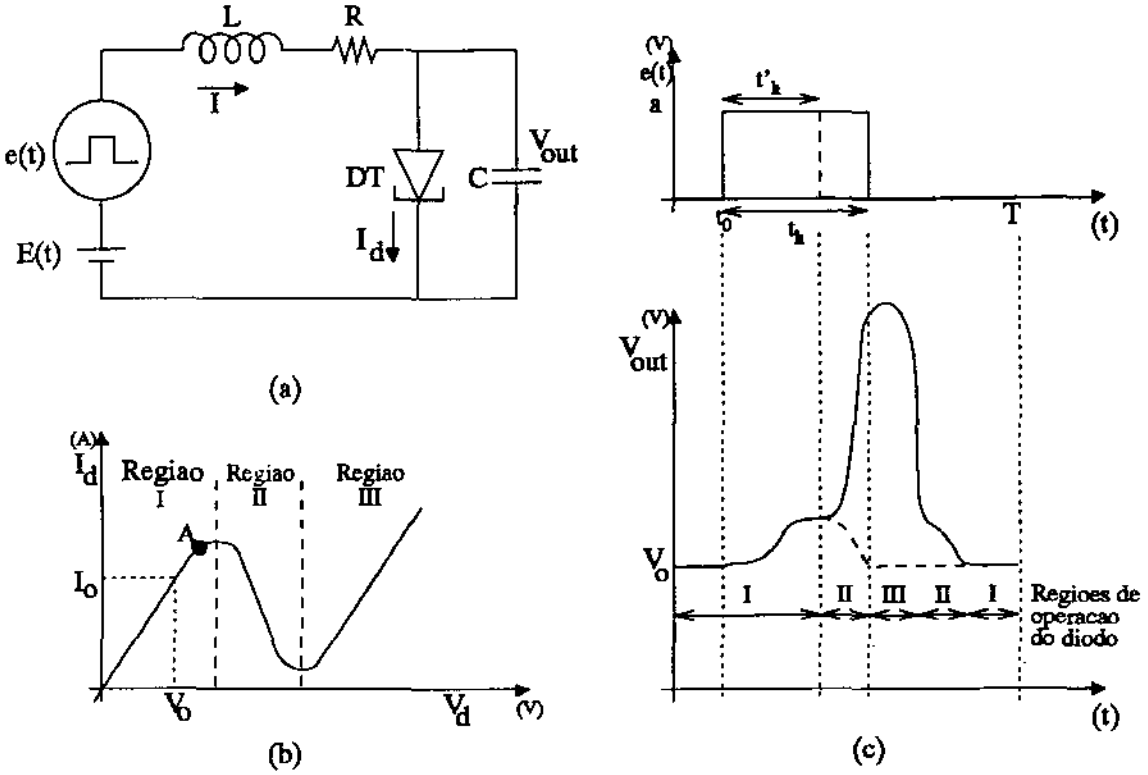


Figura 3.6: Modelo de Nagumo para um neurônio usando diodo túnel. Circuito (a), curva característica do diodo (b), sinal de entrada e resposta do modelo (c)

capacitor C_m representa a capacitância da membrana. As baterias V_{Na}, V_K e V_{Cl} representam a diferença de concentração dos íons, indicados nos respectivos índices, dentro e fora da célula. As condutâncias são moduladas através de ações sinápticas, que por conseqüência modulam o fluxo de íons (corrente elétrica) específicos em cada condutância, correspondendo a uma modulação do potencial elétrico na membrana. Os tipos de íons mais relevantes neste modelo são de potássio, sódio e cloro. Quando um neurônio é excitado sua condutância aos íons de sódio diminui e o fluxo deste tipo de íons para dentro da célula tende a polarizar a membrana aumentando seu potencial. Quando um neurônio é inibido sua condutância aos íons de potássio (ou cloro) diminui causando um fluxo de íons de potássio para fora da célula (ou um fluxo de íons de cloro para dentro) que despolariza a membrana diminuindo seu potencial.

3.3.2 O modelo de Nagumo

Um outro modelo analógico sugerido por Nagumo [YON82], como um modelo simplificado da equação de Hodgking-Huxley [Mac87], usando um diodo túnel é mostrado na figura 3.6a. A fonte e(t) representa o sinal de excitação na entrada do modelo. O resistor R polariza o diodo túnel DT no ponto (V_o, I_o) da curva característica do diodo (figura 3.6b) quando $e(t) = 0$.

No instante t_0 inicia-se um pulso em $e(t)$ com amplitude a provocando a carga do capacitor e um conseqüente movimento do ponto de operação do diodo em direção à região II. Se a área do

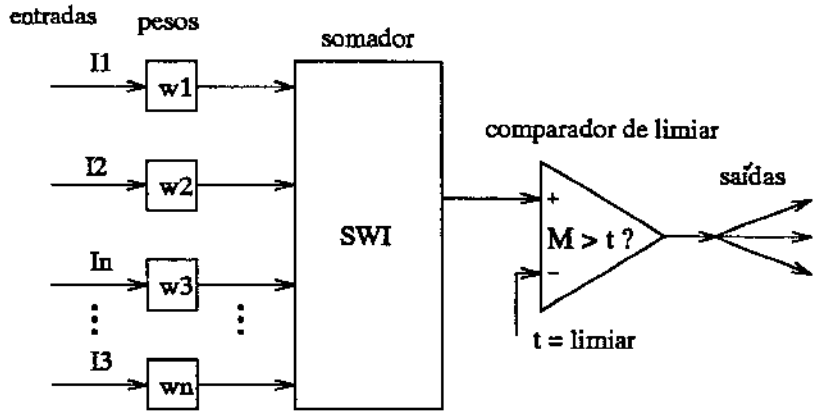


Figura 3.7: Modelo de McCulloch e Pitts de um neurônio

pulso for suficiente para deslocar o ponto de operação do diodo para a região II, a velocidade com que o capacitor se carrega aumenta rapidamente, visto que na região II a tendência da corrente no diodo é diminuir. Isso inicia um pulso na saída do modelo (neurônio ativo). Até que o diodo passe a operar na região III, onde a corrente do diodo tente a aumentar e capacitor começa a descarregar e retornar ao ponto normal de operação.

Os gráficos na figura 3.6c mostram o pulso e(t) e a tensão V_{Out} no capacitor. As curvas tracejadas correspondem ao caso em que a área do pulso não foi suficiente para ativar o modelo. Os sinais gerados por esse modelo assemelham-se aos sinais gerados no axônio de células nervosas vivas.

3.3.3 O modelo de McCulloch e Pitts

Em 1943, McCulloch e Pitts, mais interessados nas possíveis operações lógicas e cognitivas de seus circuitos que nas atividades elétricas do neurônio, apresentaram o modelo visto na figura 3.7. Ele possui 'n' entradas binárias, as quais podem vir de outros neurônios ou do mundo externo. Estes valores binários são multiplicados por seus respectivos pesos analógicos $(w_1, w_2, ..., w_n)$ e o resultado entra num somador. Pesos positivos representam entradas excitadoras, as quais tendem a provocar um estado de ativação do neurônio, enquanto que pesos negativos representam entradas inibidoras e pesos nulos correspondem a uma entrada sem nenhuma influência (conexão cortada).

A cada ciclo do sinal de *clock* a saída do somador é comparada com um limiar de ativação 't'. Se a soma for maior que o limiar então o neurônio responde com o nível binário '1' em sua saída (neurônio ativo) caso contrário o neurônio responde com '0'. Assim, o modelo responde apenas em níveis binários que são atualizados num intervalo de tempo discreto, que corresponde a um ciclo do sinal de *clock* chamado de tempo de condução do modelo.

O modelo possui ainda uma memória local e dependendo do estado da mesma os pesos dos sinais de entrada e o limiar de ativação podem variar tornando-os mais dinâmicos. Na prática os pesos e o limiar são grandezas discretas devido a limitação física da memória local.

3.3.4 O modelo baseado em RAM

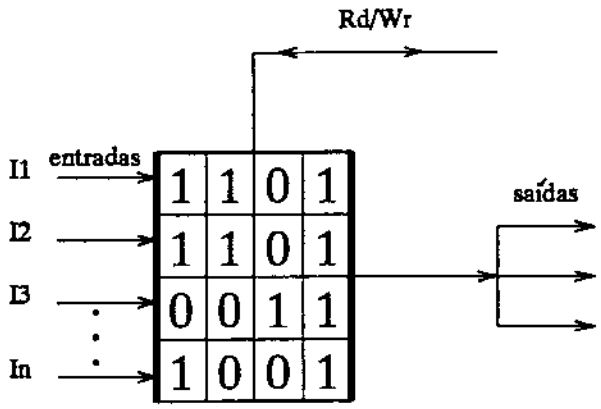


Figura 3.8: Modelo de um neurônio em RAM

Um modelo extremamente simplificado pode ser construído a partir de memórias RAMs (figura 3.8). As linhas de endereços recebem os sinais de excitação que não são mais tratados com pesos diferenciados, e a linha de dados propaga a resposta do neurônio. Como não existem mais pesos, as conexões entre os neurônios são fixas e definidas durante a fase de implementação da rede.

A função efetuada pelo modelo é definida pelos dados armazenados na RAM durante uma fase de treinamento. Um conjunto de estímulos na entrada do modelo forma um endereço x que é usado pela RAM para selecionar o bit (ou célula) endereçado, assim, o valor da função $f(x)$ corresponde ao bit selecionado. Existem 2^n diferentes resultados sobre as linhas de endereço que correspondem exatamente aos 2^n estados da RAM, a qual pode ser programada para efetuar qualquer função. No modelo em RAM (figura 3.8) tem-se uma RAM com 16 células, 4 entradas e 1 saída.

Um outro modelo de neurônios baseados em RAM, atualmente em estudo, é usado nas redes lógicas probabilísticas (PLN). O modelo se parece com o modelo em RAM descrito acima exceto que as posições de memória podem conter um de três valores: '1', '0' ou 'U' (*unknown* ou desconhecido). Se uma posição contendo 'U' é endereçada então o modelo responde com '1' ou '0' com igual probabilidade. No capítulo anterior encontra-se uma referência a um trabalho de Aleksander [AM91], onde ele investiga a capacidade de recuperação de padrões de uma PLN quando esta é alimentada com padrões distorcidos por ruídos.

3.4 Tipos de Redes neurais

As redes neurais artificiais classificam-se, quanto o número de camadas, em:

- Redes de uma camada (*single layer*)
- Redes com várias camadas (*multi layer*)

Quanto a presença de laços de realimentação, em:

- Redes sem realimentações (*feed-forward*)
- Redes com realimentações (*feed-backward*)
- Redes recorrentes

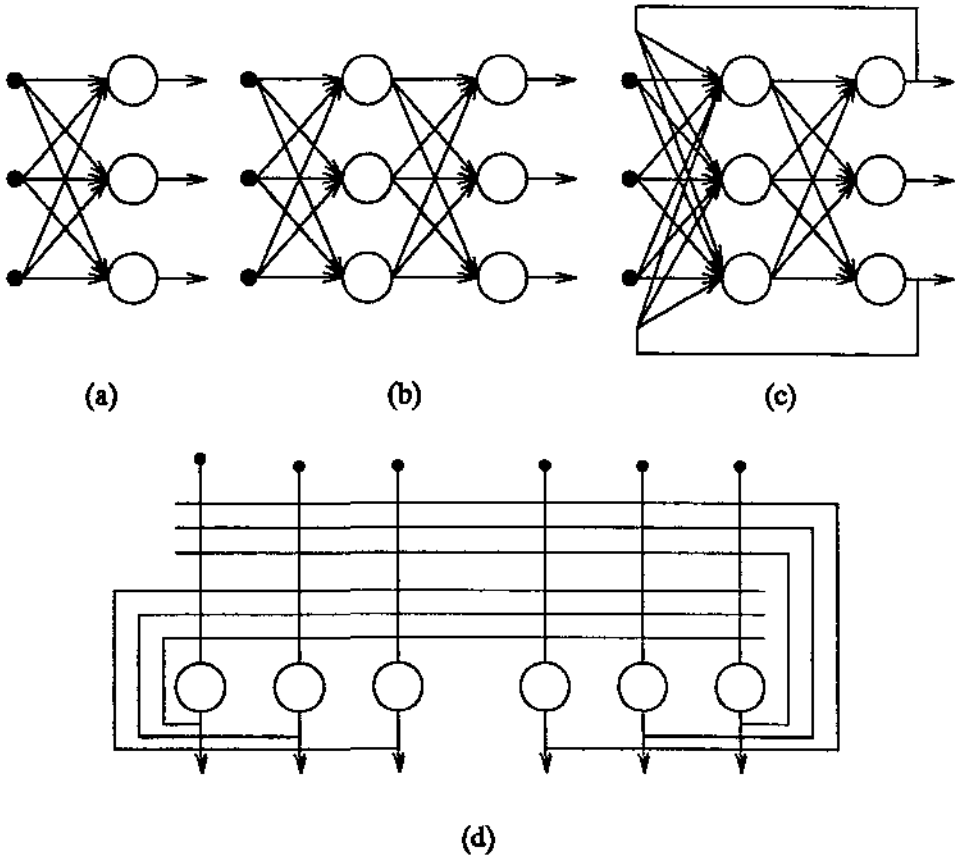


Figura 3.9: Topologias básicas para uma rede neural. (a) Uma única camada sem realimentação, (b) Múltiplas camadas sem realimentação, (c) Múltiplas camadas com realimentação e (d) recorrente.

As redes de uma só camada (*single layer*) geralmente não possuem laços de realimentação. As redes de múltiplas camadas podem ter ou não laços de realimentação mas não têm conexões entre neurônios pertencentes a uma mesma camada. Elas equivalem a uma rede de uma única camada quando suas funções de ativação são lineares. A redes recorrentes contém conexões entre neurônios da mesma camada e/ou de camadas anteriores. A figura 3.9 mostra os quatro tipos básicos de redes.

As redes também podem ser classificadas quanto ao algoritmo de aprendizado em:

- sem treinamento
- treinamento supervisionado
- treinamento não supervisionado

As redes sem capacidade de aprender não são treinadas e seus pesos são previamente estabelecidos para um dado conjunto de entrada e saída. As redes com treinamento supervisionado tentam minimizar o erro entre sua resposta e a resposta esperada, atualizando seus pesos através de uma regra de aprendizado, quando um par de entrada e saída é apresentado a rede. O algoritmo do *perceptron*, *delta rule* e *back propagation* são exemplos de algoritmos utilizados em redes com treinamento supervisionado. As redes com treinamento não supervisionado (*self-organization* não requerem vetores de saída esperados. *Competitive learning*, *self-organizing map*, o algoritmo de Grossberg e *counterpropagation* são exemplos de algoritmos de aprendizado usados neste tipo de rede. Wasserman apresenta em [Was89] uma boa introdução à redes neurais artificiais.

3.4.1 Perceptron

Umas das primeiras redes a surgir foi o *perceptron*. Ele consiste de uma rede de uma única camada sem realimentação e sua função de ativação é a função degrau. Os valores de entrada e saída são bipolares, isto é, '-1' e '1'. e tem um algoritmo de aprendizado supervisionado. Os pesos dos neurônios são atualizados toda vez que o seus estados são diferentes dos valores esperados, de tal forma que $\delta w_{i,j} = s_i t_j$, onde t_j é o estado esperado para o neurônio j. Este cálculo se repete até que os estados esperados sejam iguais aos estados reais. Os *perceptrons* apresentam uma série de limitações. Eles não conseguem, por exemplo efetuar uma função *exclusive or* (XOR). Este tipo de problema pode ser solucionado com o uso dos *perceptrons* com múltiplas camadas que são uma variação dos *perceptrons* obtida pela adição de uma ou mais camadas.

3.4.2 Pyramid Net

A *Pyramid Net* é um exemplo de uma topologia de múltiplas camadas sem realimentação(figura 3.10). As células são classificadas em níveis hierárquicos e cada célula de um nível recebe as respostas de um grupo de células do nível inferior levando seu resultado para o nível superior.

Um exemplo de aplicação desta topologia é o reconhecimento associativo de imagens. A rede responde à imagem que foi apresentada na sua entrada com outra imagem associada. Para cada ponto (*pixel*) da imagem de saída existe uma *Pyramid Net* cujas entradas recebem a imagem de entrada.

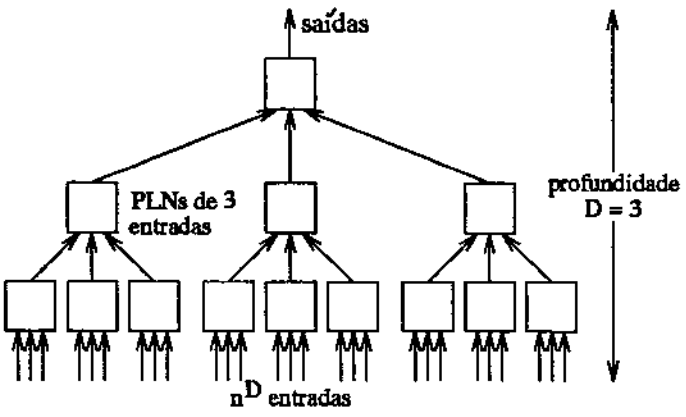


Figura 3.10: *Pyramid Net* em 3 níveis com fan-in de 3

3.4.3 Hopfield Net

A *Hopfield Net* é uma rede construída a partir do modelo de McCulloch e Pitts, na qual os pesos $w_{i,j} = w_{j,i}$ e $w_{i,i} = 0$, isto é, se o neurônio i esta conectado a j, então j esta conectado de volta para i com igual peso e nenhum neurônio esta conectado em si mesmo. Essa ultima restrição ($w_{i,i} = 0$) é tomada visto que os termos de auto-acoplamento $w_{i,i}$, quando positivos, tendem a estabilizar o estado atual da rede e tornam a dinâmica da rede mais lenta [Ges90].

A topologia da *Hopfield Net* é definida assim devido ao uso do algoritmo de aprendizado de Hebbian (Hebbian *learning* [Ges90]). O algoritmo consiste em atualizar os pesos com base nos estados (∇) dos neurônios como mostra a equação:

$$j_{i,j}(t+1) = j_{i,j}(t) + \lambda \times \nabla_i(t) \times \nabla_j(t) \tag{3.2}$$

Onde t e $t+1$ denotam o atual e próximo passo respectivamente, e λ é uma constante chamada de amplitude do aprendizado.

Este tipo de topologia é particularmente interessante na solução de problemas que visam minimizar grandezas [AHS85]. A rede quando treinada tende a convergir para um ponto de energia mínima. A equação 3.3 descreve a energia associada a um estado da rede:

$$E = -\frac{1}{2}\sum_i \sum_j j_{i,j}\nabla_i\nabla_j - \sum_j I_j\nabla_j + \sum_j \emptyset_j\nabla_j \tag{3.3}$$

A *Hopfield Net* apresenta algumas limitações. O numero de padrões que pode ser armazenado e corretamente recuperado é limitado a $0.15 \times n$, onde n é o numero de neurônios. A rede pode ficar instável, caso os padrões tenham vários bits em comum. A rede pode ficar presa em um ponto de energia mínima local e não global. O mapeamento do problema a ser resolvido na função energia nem sempre é trivial.

3.4.4 Memórias associativas bidirecionais

Este tipo de rede (BAM ou *Bidirectional Associative Memories*) é constituído de duas camadas, com conexões bidirecionais entre neurônios de diferentes camadas. Cada neurônio de uma camada

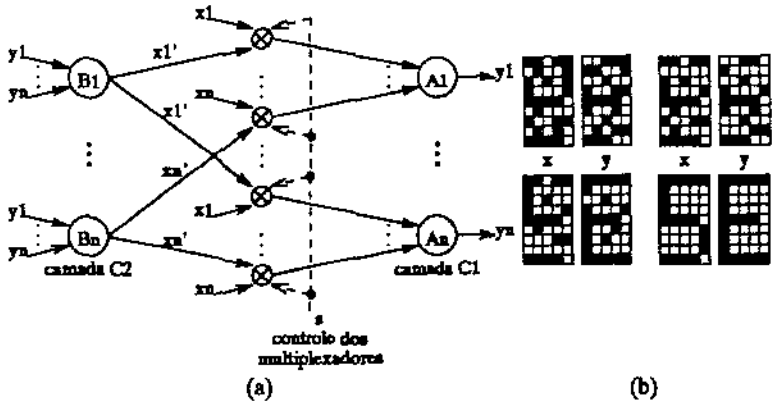


Figura 3.11: Memória associativa bidirecional com redes neurais (a) e exemplo de recuperação de um par de vetores (b)

esta conectado a todos os neurônios da outra camada e vice-versa (figura 3.11a). Elas são capazes de associar 2 vetores diferentes e também de generalizar, produzindo saídas corretas mesmo na presença de ruído.

Quando um vetor X é apresentado à camada C_1, esta produz um outro vetor de saída Y, que é apresentado à camada C_2, que por sua vez produz um vetor X', o qual é reapresentado à camada C_1. O estado de cada neurônio de uma camada é definido pelos estados dos neurônios da outra camada, pelos pesos sinápticos (regra de propagação) e pela função de ativação dos neurônios (equações 3.4). A medida que as camadas interagem entre si, o valor do vetor X' aproxima-se cada vez mais de X produzindo o vetor associado Y. O controle do multiplexador ('s' na figura 3.11) redireciona as entradas da camada C_1, permitindo que um estimulo externo X seja aplicado à camada. A figura 3.11b mostra um exemplo de recuperação de um vetor corrompido por ruídos e seu respectivo vetor associado.

$$\begin{aligned} y_j &= f(\sum_i (sx_i + (1-s)x'_i)w_{i,j}),\ com\ s = 0\ ou\ 1 \\ x'_j &= f(\sum_i y_i w_{i,j}) \\ f(x) &= \frac{1}{1+e^{-\lambda x}} \end{aligned} \tag{3.4}$$

Os pesos sinápticos são calculados em função do conjunto de vetores X e Y que serão associados. Em uma forma matricial os pesos são definidos por $W = \sum_i A_i^t B_i$. Se a matriz de pesos W for quadrada e simétrica a topologia da rede torna-se igual a topologia da *Hopfield net*. Sua capacidade de armazenamento é menor que $0.5n/log_2 n$ onde n é o numero de neurônios da rede. Existem outras variações, como BAM contínua (atualização dos estado dos neurônios de modo assíncrono), BAM adaptativa (auto ajuste dos pesos) e BAM competitiva (conexões inibidoras).

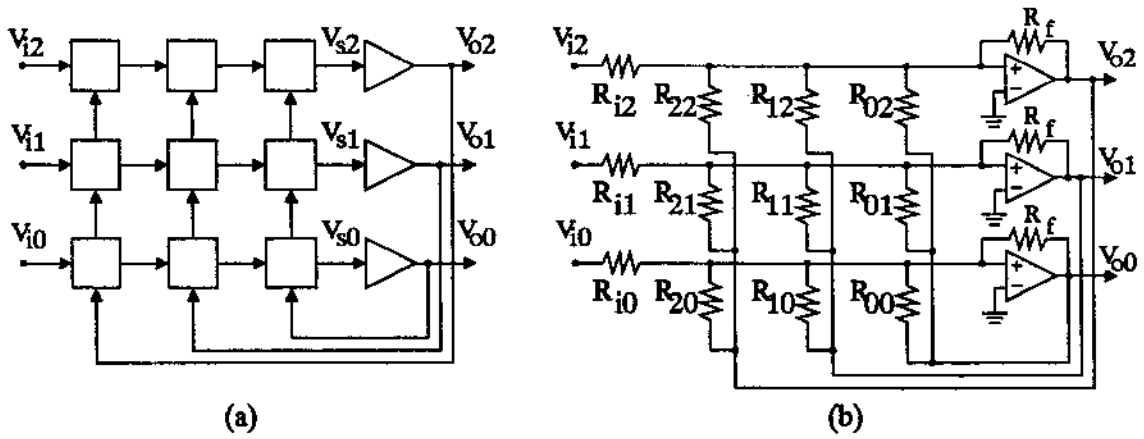


Figura 3.12: Rede neural genérica. a: versão digital; b: versão analógica

3.5 Implementação em VLSI

Redes neurais sintéticas podem ser implementadas por meio de circuitos integrados analógicos, digitais ou na forma híbrida (analógico-digitais) [Mur91]. Os exemplos a seguir motivam uma analise dos problemas e soluções no uso das abordagens citadas.

Uma rede genérica, onde a programação dos pesos sinápticos permite a implementação de qualquer tipo de topologia, é vista na figura 3.12a. A função sináptica, equação 3.5, é implementada num matriz de uma maneira sistólica, onde muitos produtos são avaliados em paralelo. Cada elemento da matriz possui um memória local onde é armazenado o peso $T_{i,j}$. A função de ativação dos neurônios, equação 3.6, é implementada pelos blocos de formato triangular localizados à direita da matriz.

$$V_s(k) = V_i(k) + \sum_{j=n}^{0} t(j,k) \times V_o(j) \tag{3.5}$$

$$V_o(k) = f(V_s(k)) \tag{3.6}$$

Uma versão analógica pode ser derivada desta, substituindo-se os neurônios, blocos triangulares na figura 3.12a, por amplificadores operacionais e os elementos da matriz por resistores $R_{j,k}$, formando circuitos somadores analógicos (figura 3.12b). Neste caso a função sináptica pode ser expressa em função dos resistores do circuito, como na equação 3.7, onde o resistor R_f é o resistor de realimentação em cada somador.

$$V_o(k) = \frac{1}{R_f}\left[R_i(k) \times V_i(k) + \sum_{j=n}^{0} R(j,k) \times V_o(j)\right] \tag{3.7}$$

3.5.1 Vantagens e desvantagens dos modelos analógicos e digitais

O exemplo de implementação digital, figura 3.12a, é totalmente flexível e seus circuitos podem computar rápida e precisamente as funções sinápticas e de ativação. Entretanto, existe uma res-

trição muito forte no tamanho da rede por *chip*, da ordem de poucas dezenas de neurônios, devido a quantidade, e as dimensões das células somadoras e multiplicadoras da matriz sináptica.

Uma solução natural é o uso de multiplexação no tempo de poucas células somadoras e multiplicadoras no calculo das funções sinápticas de cada neurônio. Esta solução implica num conseqüente aumento no tempo de resposta do sistema, que é função do numero de células disponíveis e do tamanho da rede, além de algum circuito adicional de controle da multiplexação das células.

Outra solução, apresentada por Murray [Mur91], foi a quantização do estado dos neurônios em múltiplos de 1/2. Assim, o produto $t(j,k)v_o(j)$ pode ser implementado por um simples deslocamento dos bits de $t(j,k)$ em um registrador de deslocamento, resultando em economia expressiva de área de silício. A redução na precisão provoca um efeito na função de ativação transformando sua variação suave em degraus. Tal redução de precisão é bem tolerada devido a própria natureza das redes neurais. Murray ainda observa que mesmo com a quantização dos estados em 5 níveis, o numero de sinapses que pode ser colocado num chip continua pequeno (< 100).

Implementações analógicas, tais como o exemplo da figura 3.12b, pode resultar em sistemas com maior numero de neurônios (pelo menos uma ordem de grandeza a mais que a versão digital). Cálculos podem não ser tão precisos mas isso não é crítico em sistemas como este. O principal problema é que os valores dos resistores são definidos uma única vez durante o processo de fabricação, não permitindo que a rede posa ser reconfigurada. Tais redes são programadas uma única vez durante o processo de fabricação. Seu uso restringe-se a aplicações cuja capacidade de aprender não é exigida, tal como em memórias associativas.

Uma solução possível seria a implementação de uma matriz de resistores, os quais seriam comutados eletronicamente para produzirem diferentes pesos, dependendo dos resistores escolhidos. Mas à custo do aumento do *hardware* e conseqüentemente de área. Uma implementação, citada em [Mur91], usa essa técnica para programar os pesos com um valor dentre três $(+1, 0, -1)$.

A representação do peso sináptico como uma carga Q, armazenada num capacitor de um circuito analógico, pode substituir a técnica com resistores e suas limitações. Semelhante ao método usado nas memórias RAMs dinâmicas, uma tensão analógica representando o peso é produzida quando uma certa carga elétrica é armazenada num capacitor. Este método apresenta uma das melhores soluções, com respeito a área de silício por sinapse, mas está sujeito a problemas com correntes de fugas intrínsecas ao processo MOS e corrupção de dados induzidos pelo acoplamento entre os nós do circuito. Para contornar esses problemas são usados circuitos de *refresh*.

Circuitos híbridos (análo-digital) são outra possibilidade de implementação. Na maioria dos casos os pesos são armazenados em pequenas memórias digitais e as funções de propagação e ativação são efetuadas na parte analógica. Embora resolva o problema apresentado pelos circuitos analógicos, esta abordagem ainda apresenta o inconveniente consumo de área que as memórias digitais demandam.

Um problema de implementação, tanto nos modelos analógicos como nos digitais, é a limitação de comunicação entre *chips* de um mesmo sistema, imposta pelo reduzido numero de pinos dos *chips*. Isto se reflete no sistema impossibilitando a implementação de redes com grande numero de neurônios fortemente interconectados. Uma sugestão de solução seria o uso de fibras óticas diretamente acopladas aos chips [AMP87]

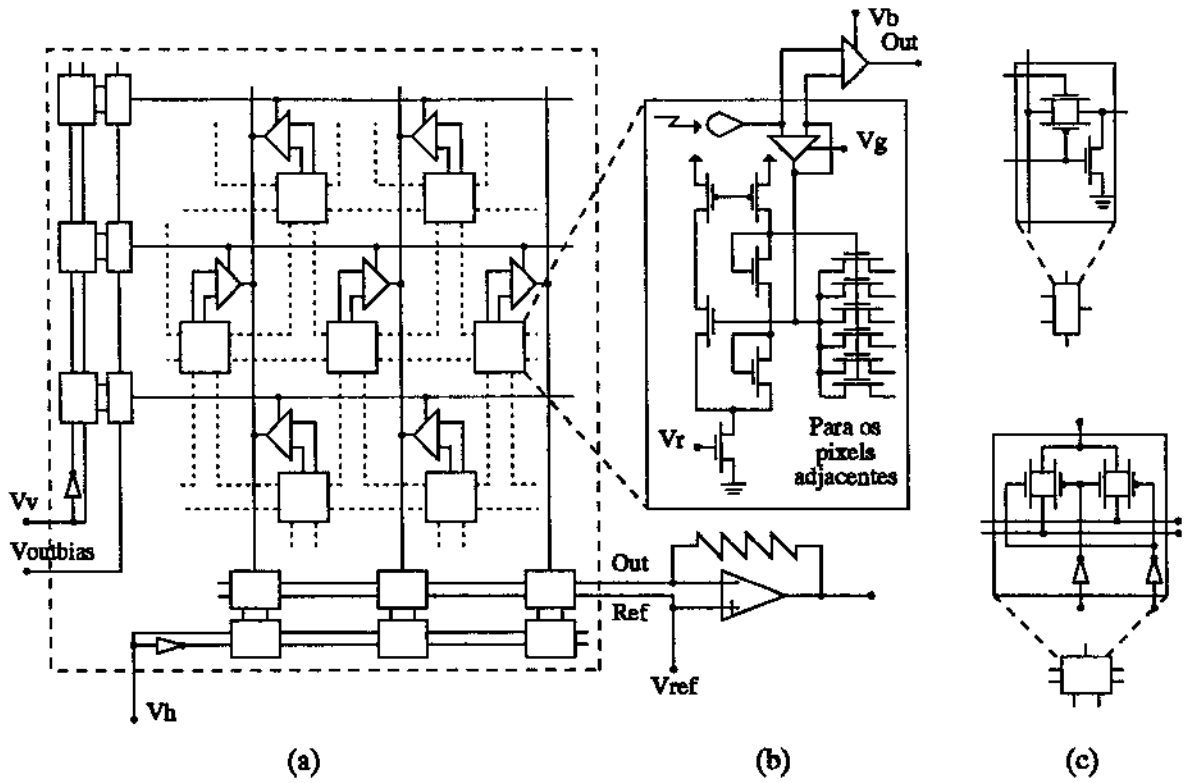


Figura 3.13: *Floorplan* de um chip que implementa uma retina artificial(a), detalhe de um *pixel* da matriz(b), circuito de ativação de linha e multiplexador (c).

3.6 Exemplos de Implementações

Implementações não convencionais, altamente inspiradas em redes neurais biológicas, surgiram em decorrência das inúmeras pesquisas na área e do avanço da tecnologia microeletrônica. Apresentam-se aqui dois exemplos. O primeiro, desenvolvido por Carver Mead no Instituto de Tecnologia da California, imita uma retina constituindo-se num interessante sistema de aquisição de imagens. O segundo exemplo foi desenvolvido por Aleksander no *Imperial College of Science and Technology* em Londres, tratando o problema de reconhecimento ou classificação de padrões de imagens usando uma rede neural digital formada por uma camada de memórias RAMs.

3.6.1 Um Modelo de Retina em Silício

Carver Mead em 1989 apresentou uma implementação híbrida (parte analógica e parte digital) em silício da retina [Mea89]. Sua arquitetura baseava-se em observações de uma retina animal. A figura 3.13a mostra o *floorplan* adotado. Os *pixels* são dispostos de modo a formar uma matriz hexagonal. Cada *pixel* no centro de um hexágono é conectado com os *pixels* de seus vértices, formando uma rede. O *chip* possui também um arranjo com registradores de deslocamento, em sua periferia, possibilitando a leitura de qualquer *pixel* processado.

Apenas um estágio do registrador de deslocamento vertical está ativado por vez. Assim, ape-

nas uma linha da matriz está ativa enquanto todas as outras linhas permanecem inativas, num determinado instante. Cada uma dessas linhas estão conectadas com todos os pinos V_b dos pixels pertencentes à linha. Apenas a linha que está ativa coloca o seu resultado nas colunas que estão conectadas aos multiplexadores do registrador de deslocamento horizontal. O sinal $V_{outbias}$ controla a intensidade com que estes sinais são colocados nas colunas.

Do mesmo modo que no vertical, apenas um estágio desse registrador horizontal está ativo por vez, fazendo com que um dos multiplexadores conecte o sinal correspondente a sua coluna com o sinal de saída *out*, enquanto que todas as outras colunas são conectadas ao sinal *ref*. O *chip* foi projetado para ser usado com um amplificador sensível a corrente que é conectado com estes sinais (*out* e *ref*). Este esquema mantém todas as colunas num mesmo potencial, eliminando a ocorrência de transientes quando uma coluna é selecionada.

O circuito de um pixel (figura 3.13b) recebe o sinal vindo de um foto-receptor cuja curva característica (resposta) é logarítimica. Isto, além de acomodar uma larga faixa de intensidade luminosa numa conveniente faixa elétrica de tensão, torna o sistema insensível a variação de luminosidade da cena observada. O circuito provê 6 conexões com *pixels* vizinhos e o sinal de saída *output* é proporcional a diferença entre o sinal vindo do foto-receptor e o sinal médio resultante dos sinais vindos dos *pixels* vizinhos. Assim, quando há uma variação de intensidade luminosa na cena, todos os pixel são afetados igualmente e como o sinal de saída está em função de uma diferença de dois logarítimos, o sinal de saída *output* não é afetado.

3.6.2 O Wisard

Wisard [Red87] é uma arquitetura para o reconhecimento de imagens proposta por Wilkie, Stoneham e Aleksander, no Grupo de Computação Neural do Departamento de Computação do *Imperial College of Science and Technology*. O Wisard permite o reconhecimento de imagens colocando-as em uma dentre diversas classes. Tais classes são formadas anteriormente na fase de treinamento, onde imagens típicas de cada classe são exibidas ao Wisard.

Wisard é uma *connectionist machine*. Suas partes componentes constituem-se de redes neurais, modeladas em RAMs, de uma única camada. A imagem do objeto, ao contrário da máquina de Von Neumann cujo processamento se dá apenas em um único processador, é processada por todo o conjunto de neurônios de uma dessas redes. Isso resulta no reconhecimento do objeto mesmo que este se apresente um pouco diferente (por exemplo: uma face sorrindo, franzindo as sobrancelhas ou meio encoberta por um lenço).

O Wisard é constituído de 8 discriminadores os quais são treinados um por classe de objetos. O discriminador (figura 3.14) é constituído de pequenos bancos (16×1 *bits*) de RAMs as quais têm suas linhas de endereço alimentadas com os pontos da imagem em uma ordem aleatória, sempre da mesma maneira pois esta ordem é definida por *hardware*. Na saída das RAMs existe um circuito que contabiliza o número de RAMs ativas (que respondem com o nível binário '1'). Essa arquitetura baseia-se na implementação de redes onde os neurônios são modelados através de RAMs. As linhas de dados (saída ou resposta) das RAMs (neurônio) excitadoras são diretamente ligadas às de endereço (entrada dos neurônios) das RAMs excitadas sem a interferência de nenhum peso, constituindo numa versão bastante simplificada em vista aos sistemas biológicos.

Antes de chegar ao Wisard a imagem é pré-processada. O sinal elétrico analógico da imagem (obtida de uma câmara) é digitalizado. Esse sinal digitalizado é composto de 256 níveis de cinza os quais são convertidos para dois níveis (branco e preto). Pontos da imagem cujo o nível de cinza

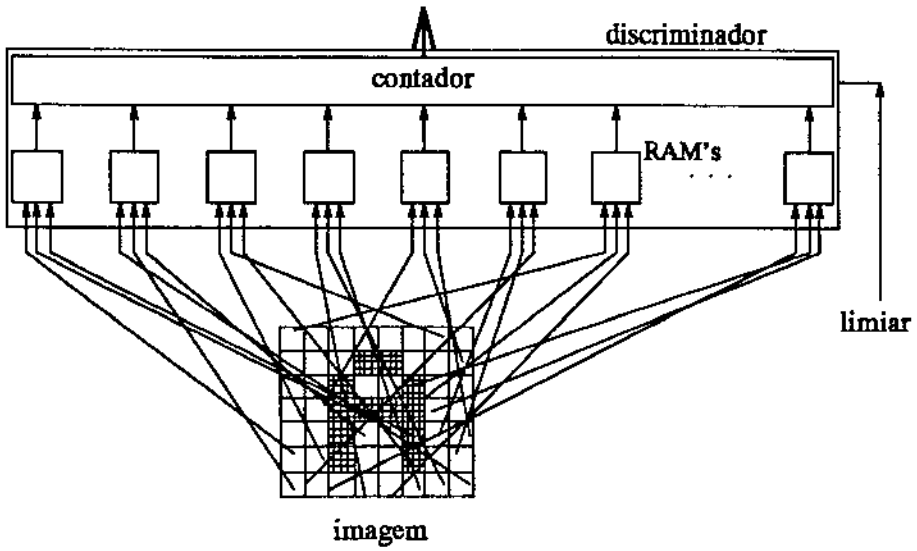


Figura 3.14: Estrutura de um discriminador

está acima de 127 são convertidos para preto, pontos com um nível de cinza menor que 128 são convertido para branco. A imagem resultante é constituída de uma matriz de 256 × 256 pontos (*pixels*) representados por 65536 *bits* (*bit stream*). Esta imagem deve está devidamente normalizada e enquadrada, visto que o Wisard, como todas as outras técnicas que trabalham diretamente sobre a representação *bit-mapped*, não reconhece imagens ampliadas, reduzidas ou com um desalinhamento muito grande. Pequenos desalinhamento (da ordem de ±10 *pixels*) são tolerados.

Em uma fase de treinamento, todos os *bits* de todas as RAMs são zerados e a imagem pré-processada é organizada aleatoriamente em 16384 (645536/4) grupos de 4 *bits* [AW85]. Cada grupo de 4 *bits* endereça uma RAM diferente (existem 16384 delas) e todos os *bits* endereçados são forçados para o nível '1'. Todas as RAMs tem um único *bit* em '1' os quais estão completamente relacionados com a imagem recebida. Este processo é repetido para cerca de 200 imagens de referência do objeto as quais apresentam pequenas diferenças entre si. Se as imagens forem bem diferentes as RAMs tendem a saturar (tendem a ter todos os seus *bits* em '1') o que não é desejável. Este processo corresponde ao treinamento de um discriminador. Os outros discriminadores podem continuar a ser treinados com outras imagens. O número de imagens que podem ser treinadas (número de classes) é igual ao número de discriminadores.

Após o treinamento, o Wisard pode entrar na fase de operação. O *bit stream* da imagem a ser reconhecida, após o pré-processamento, endereça as RAMs dos 8 discriminadores simultaneamente e as linhas de controle de escrita/leitura são colocadas para leitura. Assim cada RAM sobre cada discriminador responderá com '1' se o padrão de 4 *bits* coincidir com os padrões da fase de treinamento. O discriminador que responder com o maior número de '1s' em suas RAMs é o vencedor e a imagem é colocada em sua classe. Existe também um limiar mínimo de '1s' (o valor usado por Aleksander foi 70%) que o discriminador deve alcançar para considerar-se como tendo reconhecido a imagem. Se nenhum discriminador alcançar o limiar a imagem é dada como não reconhecida.

Parte II

Implementação de um Sistema de Reconhecimento de Imagens

Capítulo 4

O Sistema

4.1 Introdução

Este capítulo descreve algumas opções de projeto (seção 4.2) que surgiram durante a definição de uma arquitetura (seção 4.3) para o sistema de reconhecimento de padrões imagens proposto. O projeto foi orientado para uma estrutura simples, regular e escalável. A adoção de estruturas simples, sem comprometimento da funcionalidade, aumenta a margem de sucesso e reduz o tempo de execução do projeto. Regularidade é uma característica que facilita implementações em VLSI, permitindo elevada densidade de transistores, a qual é especialmente interessante quando a capacidade de processamento do circuito é linearmente dependente ao número de estruturas lógicas que ele contem. Arquiteturas escaláveis normalmente são exigidas em aplicações onde há uma crescente demanda de recursos como é o caso da maioria das aplicações que envolvem reconhecimento de padrões de imagens.

A idéia de especificar um sistema para reconhecimento de imagens, com a implementação de alguma de suas partes (um CI dedicado), surgiu da observação de que uma máquina, com um princípio de funcionamento como o do Wisard (seção 3.6.2), poderia ser implementada em VLSI possibilitando a exploração de uma vasta área de aplicação. Algumas versões mais rígidas para o sistema foram descartadas devido a essa diversidade de aplicações. A quantidade e o tamanho dos discriminadores deveriam ser redimencionadas pelo usuário, conforme as necessidades de sua aplicação. A especificação da capacidade máxima de processamento dependia de muitos fatores como capacidade das RAMs, o tamanho de suas células, área de silício disponível e método usado no projeto físico. Embora fosse difícil prever com precisão um número para tal capacidade máxima, 1 M*bits* foi adotado como um meta a ser atingida. Isto é, o sistema deveria ser capaz de processar uma imagem de 1 milhão de *pixels* em sua configuração máxima. Essa meta foi atingida com uma arquitetura escalável, baseada em uma matriz de *chips* dedicados.

O sistema é composto por um computador IBM PC/AT e um cartão. O computador servirá de plataforma oferecendo serviços de armazenamento em massa dos padrões de imagens, gerenciamento e controle do sistema. O cartão é composto pela matriz de circuitos integrados VLSI dedicados, os quais implementam os discriminadores, e um circuito de controle de requisição de barramento [Egg90], [IBM84] que dá acesso à matriz aos *pixels* da imagem armazenada na memória principal do PC. O cartão é visto pelo PC como um dispositivo de entrada e saída que pode ser programado para fazer acesso direto a memória (DMA) e pode fornecer os resultados através de uma simples leitura de *I/O*. O sistema foi descrito e validado a nível comportamental na linguagem de especificação de

hardware VHDL [Per91] [Arm89] [I3E87] através de compilador e simulador da Mentor Graphics [Cor90] (seção 4.4). O sistema pode ser aplicado em diversas áreas, tais como, automação de controle de qualidade, leitura artificial de texto, reconhecimento de padrões biomédicos, etc. A utilização de dispositivos como PLDs no controle e *interface* do cartão e as próprias características da arquitetura dos discriminadores facilitam a implementação do sistema.

4.1.1 Facilidade de Implementação em VLSI

A arquitetura proposta é bastante regular e possui uma pequena variedade de células. Isso permite a utilização da metodologia *full-custom*, otimizando o tamanho das células, sem o inconveniente longo tempo de projeto característico desta metodologia. A maior parte da área de silício é ocupada pelas células de memórias das RAMs e quase todas as conexões são bastante regulares permitindo que as mesma sejam efetuadas concomitantemente durante a fase de posicionamento das células (*placing*).

4.1.2 Motivação

A arquitetura do Wisard tem bom desempenho em aplicações onde as imagens são geradas mecanicamente ou tenham pouca variação de estilo. Por exemplo, o rosto de uma pessoa específica, pode ser reconhecido mesmo com pequenas mudanças de fisionomia (uma face sorrindo ou franzindo as sobrancelhas), mas a partir desse rosto pode não ser possível reconhecer outros rostos humanos devido à grande diversidade desse tipo de imagem. Outro exemplo de bom desempenho é na leitura artificial de textos produzidos por máquinas. Porém na leitura artificial de textos manuscritos há dificuldades devido à grande variação de estilos e, até mesmo num estilo de caligrafia.

Uma aplicação que satisfaz as condições de bom desempenho descritas acima é na inspeção visual em linha de montagem. O sistema pode reconhecer pequenos defeitos em peças com o mesmo padrão, numa velocidade bastante superior aos sistemas de inspeção visual manual. Tais sistemas permitem maior automação com conseqüente aumento de produção, o que não é viável com antigos sistemas de inspeção visual humana, já que eles são lentos e subjetivos.

Outra aplicação interessante é o reconhecimento e contagem de cédulas. A máquina pode aprender a reconhecer as cédulas monetárias de diversos valores e assim habilitar os caixas automáticos de agências bancárias a aceitar pagamentos e depósitos em dinheiro e até a voltar trocos.

4.2 Evolução da Concepção e Alternativas de Projeto

Durante a fase de especificação do sistema surgiram algumas opções relevantes ao projetos que devidamente analisadas contribuíram para o seu aprimoramento. Optou-se por uma arquitetura modular e escalável face a crescente demanda de recursos das aplicações. Outra característica levada em conta foi a simplicidade, procurou-se explorar o máximo os recursos da máquina hospedeira (PC), garantindo assim um tempo de projeto menor, um controle maior sobre eventuais falhas no sistema e um custo relativamente menor. Adotou-se, sempre que possível soluções criativas que tirassem proveito das limitações do sistema, a exemplo da metodologia de entrada dos *pixels* (nesta seção) e do esquema de contagem das RAMs ativas (próximo capítulo).

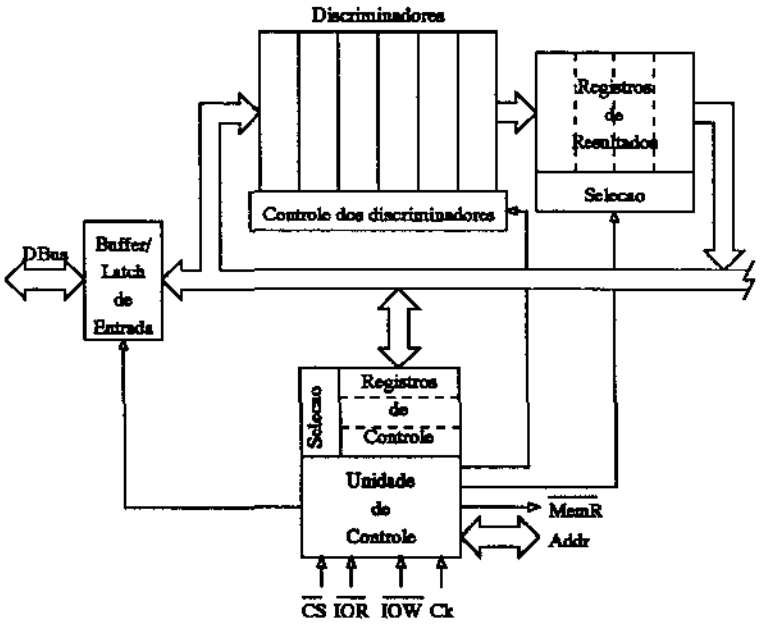


Figura 4.1: Primeiro esboço do sistema

4.2.1 Distribuição Física do Sistema

A análise da distribuição física dos blocos lógicos de alto nível de um sistema, durante a fase de especificação e descrição comportamental, contribui para a otimização de fatores como desempenho, custo, flexibilidade, etc. Em geral, há uma tendência em integrar o máximo de funcionalidade em um único ou poucos *chips*. Isto resulta em sistemas mais velozes, de baixo custo, simples e de fácil manutenção. Em outros casos, dependendo da aplicação, os sistemas devem oferecer um alto grau de flexibilidade, permitindo ao usuário aumentar o poder de processamento através da simples adição de módulos. As duas concepções apresentadas a baixo exemplificam como a distribuição física do sistema pode ser decisiva na implementação de um projeto.

Concepção em um único chip

A princípio, o sistema foi pensado como um único chip que conteria todos os discriminadores, como pode ser visto na figura 4.1. Esta apresentava uma série de limitações e problemas que forçaram a sua evolução.

A estratégia de manter os discriminadores dentro de um único *chip* se mostrou inviável fisicamente devido a grande quantidade de área de silício requerida e além disso tornaria o sistema muito rígido, isto é, os discriminadores teriam tamanhos fixos, impossibilitando uma arquitetura escalável.

Concepção escalável em vários chips

Para adequar o sistema aos requerimentos de recursos cada vez mais crescente das aplicações, sua organização foi repensada como um sistema composto de vários *chips* os quais poderiam ser configurados para trabalharem individualmente como discriminadores pequenos ou em grupos, for-

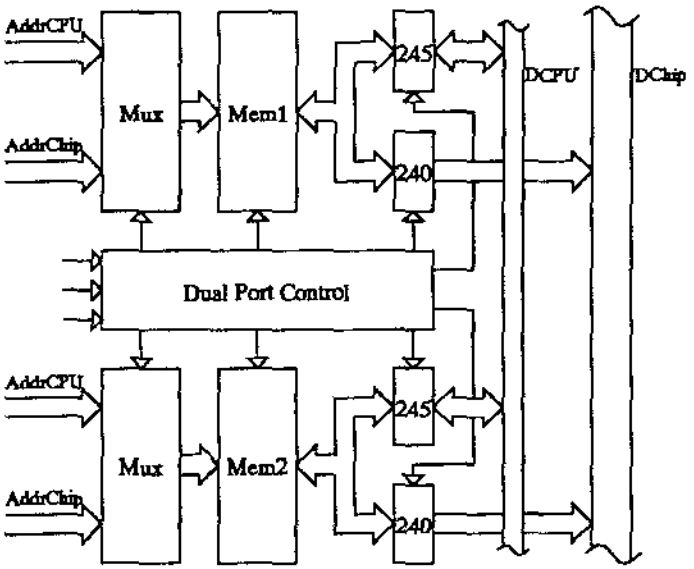


Figura 4.2: Opção de projeto com memória de duplo acesso *dual-port*

mando discriminadores maiores. Esta estratégia além dos benefícios já explicitados, resolveu o problema de área na medida que o sistema ficou subdividido entre vários *chips*.

Um ponto que mereceu atenção neste sistema foi o cuidado de ter-se o mínimo possível de circuitos de controle internos aos *chips*, que poderiam ser condensados em uma estrutura externa de controle geral, deixando uma maior área de silício para os discriminadores. Esse controle externo ficaria responsável pela *interface* do conjunto de *chips* com o PC.

4.2.2 Interface com o Computador Hospedeiro

As alternativas de *interfaces* com o PC foram analisadas segundo os critérios já citados de simplicidade e máxima utilização dos recursos do PC. Memória de duplo acesso possibilita tarefas concorrentes e pode contribuir para um aumento do desempenho do sistema porem, exige memória dedicada no cartão, tornando seu projeto mais complicado. Compartilhamento de barramento, embora não permita funcionamento simultâneo da CPU e do cartão, mostra-se mais atraente, por permitir escalabilidade e por ser simples.

Interface via memória de duplo acesso

A figura 4.2 apresenta um esquema de memória de duplo acesso com dois bancos. Os multiplexadores (Mux) funcionam com seletores escolhendo entre o barramento de endereços do PC (AddrCPU) e do sistema (AddrChip), possibilitando que um dos dois barramentos enderecem os bancos de memória. Os *buffers* 245 conectam, oportunamente, o barramento de dados do PC (DCPU) com os bancos de memória num modo bidirecional, permitindo ao PC ler ou escrever nos bancos. Os *buffers* 240 conectam os bancos ao barramento de dado do sistema em uma única direção, visto que o sistema apenas necessita fazer leitura nos bancos (para treinar seus discriminadores ou reconhecer/classificar as imagens). O bloco de controle (*Dual Port Control*) coordena

o fluxo dos sinais entre os barramentos e as memórias.

Esta alternativa apresenta as seguintes vantagens:

- Leitura simultânea de mais de um banco de memória. Diversos bancos de memória podem ser lidos simultaneamente pelo sistema tornando o barramento DChip mais largo e conseqüentemente aumentando a velocidade de leitura dos pixels pelo sistema.
- Acesso da CPU à outros blocos de memória. Enquanto o sistema está lendo determinados bancos de memórias, a CPU pode fazer acesso paralelo a outro blocos de memória, que não estão sendo acessado pelo sistema. Isso pode ser útil quando o processo de aquisição de imagens tem que trabalhar em paralelo com o sistema para obter tempo real de execução.

Em contra partida tem as seguintes desvantagens:

- Memória dedicada no cartão. Como as memórias do PC não são de duplo acesso, o sistema teria que prover tais bancos de memória no seu próprio cartão.
- Aumento de complexidade. A adição de memórias dedicadas ao cartão contribui para o aumento da complexidade do sistema.
- Limitação do espaço de endereçamento. O sistema só poderia acessar dados contidos nas memórias dedicadas, restringindo seu espaço de endereçamento.
- Controle de memória *dual-port* não trivial. Projeto de memória de duplo acesso são complicados exigindo uma estratégia coerente de controle de acesso dos *ports*.
- Difícil escalabilidade. O acesso dos bancos de memórias de uma placa pelos *chips* de outra seria mais complicado dificultando a escalabilidade do sistema.

Interface via compartilhamento do barramento

Nesta alternativa aproveita-se a própria estratégia de extensão de canais de DMA do PC [Egg90] (ver figura 4.3). O chip controlador de DMA obtém o controle do barramento do PC através de um par de sinais. Primeiro o controlador requisita o barramento à CPU ativando o sinal HRq (*Hold Request*) e fica aguardando uma resposta da CPU. Quando a CPU decide entrar no estado *hold* e entregar o barramento, ela ativa o sinal HldA (*Hold Aknolege*). Neste instante o controlador, que está monitorando a resposta da CPU (HldA), obtém o controle do barramento, executa a tarefa para qual foi programado e devolve o barramento à CPU desativando o sinal HRq, a CPU por sua vez ao percebe que o pedido de *hold* foi desativado, inibe sua resposta (HldA) e reassume o controle do barramento.

O controlador de DMA pode ser programado para operar em alguns modos e um destes modos permite que ele transforme um de seus canais em uma extensão para outros controladores. A idéia consistia em conectar os chips do sistemas em alguns desses canais como se eles fossem controladores de DMA e uma vez com a posse do barramento os chips poderiam acessar qualquer região do espaço de endereçamento do PC.

As vantagens desta alternativa são:

- Sistemas mais simples.

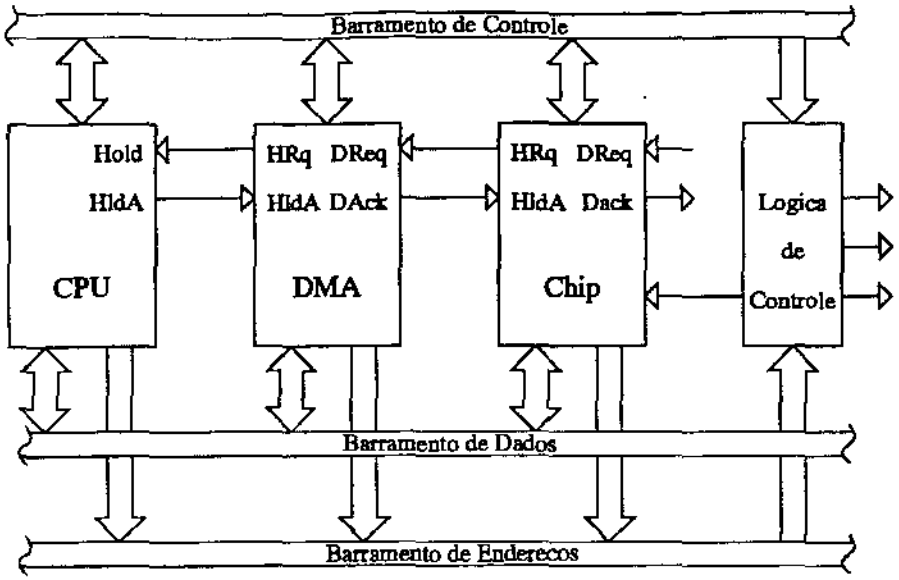


Figura 4.3: Opção de projeto com compartilhamento de barramento

- Acesso a todo o espaço de endereçamento.
- Facilita escalabilidade

Em relação a alternativa anterior perde as vantagens:

- Leitura simultânea de blocos de memória.
- Acesso da CPU a outros blocos de memórias.

A segunda alternativa foi adotada com alguma modificação. No lugar dos chips obterem diretamente o acesso do barramento, um circuito de controle externo, projetado com PLDs, o faz. E também foi usado um sinal (*master*) a mais, do próprio PC-AT, destinado ao processo de requisição de barramento por processadores externos.

4.2.3 Metodologia de Embaralhamento dos "Pixels"

Um requisito para o funcionamento de maquinas como o Wisard (que usam a técnica da *n-tuple* [AW85]) é a forma aleatória com que os grupos de pixels são formados. Se por exemplo, os grupos fossem formados por pixels adjacentes, obtidos através de uma leitura seqüencial da imagem, o grau de discriminação das RAMs diminuiria consideravelmente visto que pequenas diferenças entre o padrão e a imagem desconhecida só afetaria um número pequeno de RAMs. Como obter um embaralhamento satisfatório na formação dos grupos mantendo uma boa eficiência na leitura dos *pixels*. Foram investigadas algumas alternativas que são apresentadas a seguir:

Embaralhamento interno com leitura seqüencial: esta foi a primeira alternativa que ocorreu. Os pixels deveriam ser lidos seqüencialmente e embaralhados dentro do discriminador.

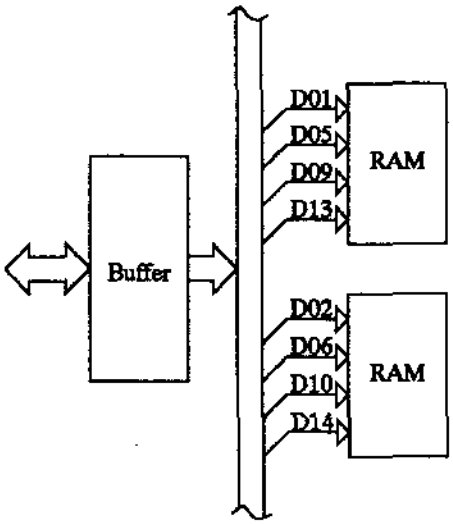


Figura 4.4: Opção de projeto usando embaralhamento interno com leitura aleatória

Tal alternativa mostrou-se inviável por ter uma entrada de dado ineficiente e roteamento complicado. Na entrada de dados, muitas palavras da imagem eram lidas mais de uma vez a medida que os pixels eram escolhidos. O roteamento dependia do *pixel* escolhido.

Embaralhamento interno com leitura aleatória (LFSR): nesta versão, as palavras de 16 *bits* deveriam ser lidas em uma ordem aleatória gerada por um circuito LFSR, e os bits de posições não adjacentes formariam os grupos que iriam endereçar as RAMs (figura 4.4).

As principais características desta alternativa são:

- Aproveitamento de leitura serial no embaralhamento. Devido a limitação de pinos, a leitura teria que ser feita de modo serial mas nada impedia que ela fosse feita em uma ordem aleatória.
- Simplicidade no roteamento. Neste caso os pixels que formariam os grupos já estavam previamente estabelecidos, facilitando o roteamento sem nenhum circuito adicional de seleção.
- Aleatoriedade limitada a uma palavra. Como os pixels de um grupo eram sempre formados de *bits* de uma mesma palavra o embaralhamento ficava restrito ao nível de palavras.
- Grande quantidade de pinos. O circuito gerador dos endereços aleatórios (LFSR) teria pelo menos 24 pinos para que qualquer porção de memória do PC pudesse ser endereçada pelos chips.
- Difícil escalabilidade dos discriminadores Como cada chip teria seu próprio LFSR, ficaria muito difícil manter a coerência entre o consumo da imagem por chips do mesmo discriminador. Alguns LFSRs só poderiam consumir palavras que ainda não tivesse sido consumida por nenhum dos outros.

Embaralhamento externo de palavras e interno de *bits*: esta alternativa surgiu como um

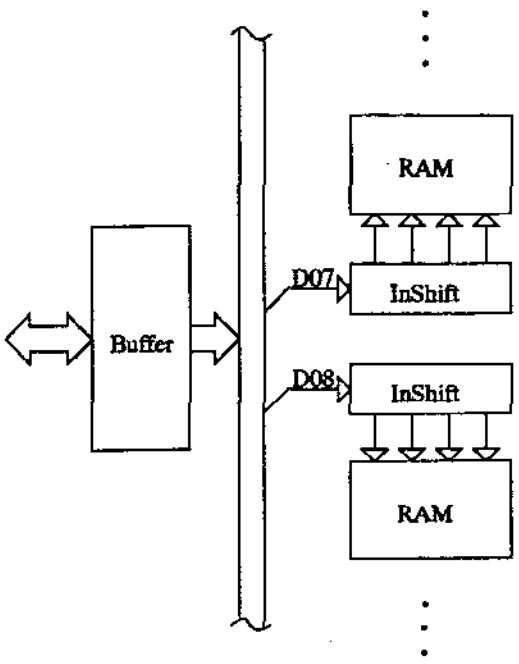


Figura 4.5: Opção de projeto usando embaralhamento externo de palavras e interno de *bits*

aprimoramento da anterior. Aqui um único LFSR fica externo aos *chips* endereçando as palavras, enquanto que internamente, (figura 4.5) cada *bit* é armazenado em um registrador de deslocamento (*InShift*). O registrador de deslocamento tem capacidade de armazenar 4 *bits* e depois da leitura de 4 palavras, cada um dos 16 registradores terão retido 4 *bits*, cada um de uma palavra diferente. Com isso garante-se:

- Aleatoriedade a nível de pixels.
- Menor quantidade de pinos (LFSR externo).
- fácil escalabilidade dos discriminadores.

4.3 A Arquitetura

A figura 4.6 mostra o diagrama esquemático do Sistema. O primeiro bloco (*PCBus*) representa o PC e seu barramento, o bloco *BUFFER* é constituído de dois *Buffers* 74LS245 para o barramento de dados, o bloco central (PLD) representa o bloco de controle do cartão e a direita temos uma matriz onde cada bloco representa um circuito integrado dedicado ao reconhecimento de imagens. O sistema pode ser dimensionado para tratar imagens variáveis em tamanho e número, através do uso de um número apropriado de circuitos integrados na matriz. Cada cartão suporta um número máximo de 64 circuito integrados, mas o sistema pode ser expandido para dois ou mais cartões.

O bloco de controle do cartão (PLD), cuja estrutura interna pode ser vista na figura 4.7, é composto dos seguintes blocos;

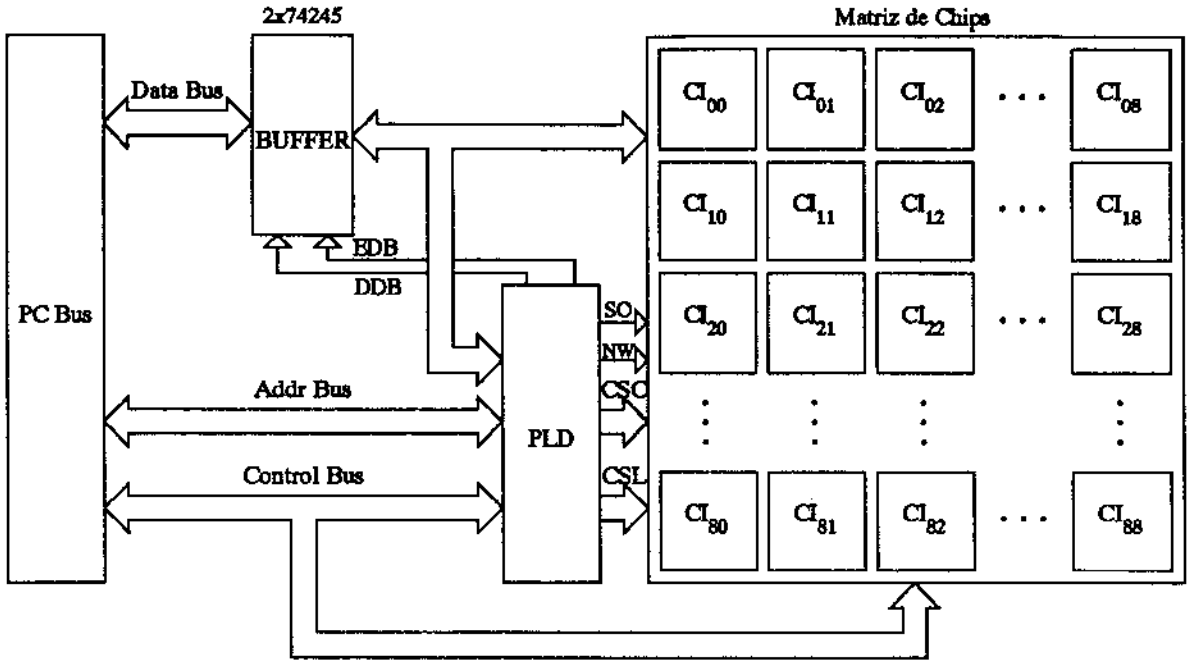


Figura 4.6: Arquitetura do sistema.

- Lógica de Controle de *I/O*
- Registradores de Programação
- LFSR
- Máquina de Estados

A lógica de controle de *I/O* efetua as funções de decodificação dos endereços de *I/O*, selecionando um dos registradores de programação ou um dos CIs da matriz. A seleção dos CIs se dá através dos sinais CSL e CSC (*Chip Select Line* e *Chip Select Column*). Esse tipo de seleção foi usado para minimizar a quantidade de sinais de seleção de um elemento da matriz.

Os registradores de programação armazenam informações de como o LFSR e a máquina de estados devem se comportar. Eles são:

- **BIR** (*Begin of Image Register*): Durante a programação, esse registrador e carregado com o endereço inicial da imagem. O endereço colocado no *AddrBus* pelo LFSR é composto pela soma do conteúdo deste registrador com um deslocamento que é gerado aleatoriamente no LFSR. Isso permite que a imagem possa ser carregada em qualquer posição do espaço de endereçamento do PC.
- **LLR** (*Length of LFSR Register*): Para assegurar que o deslocamento gerado pelo LFSR não passe do espaço de endereçamento da imagem, o LLR é programado de acordo com o tamanho dessa imagem.

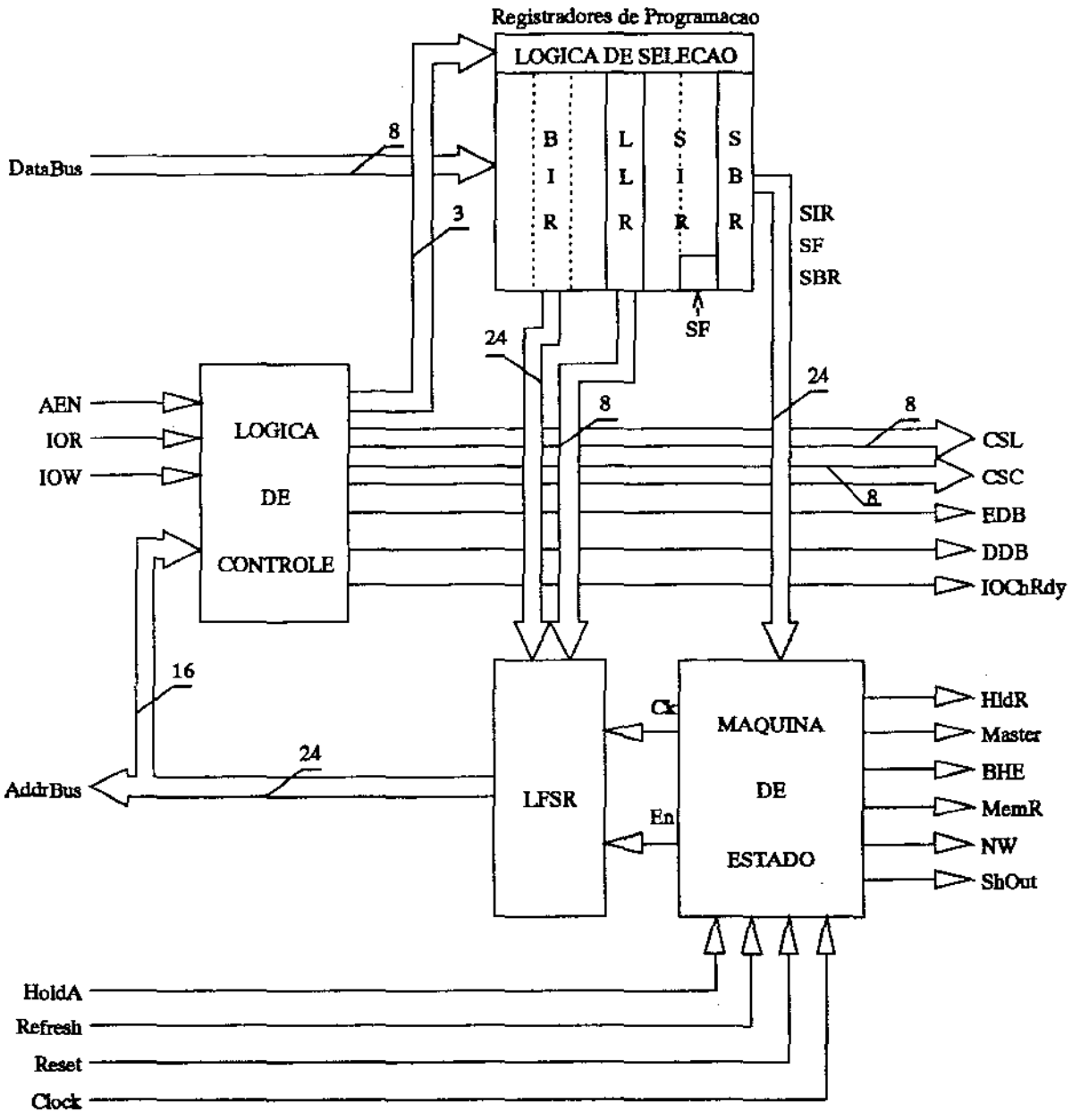


Figura 4.7: Arquitetura do bloco de controle do cartão (PLD)

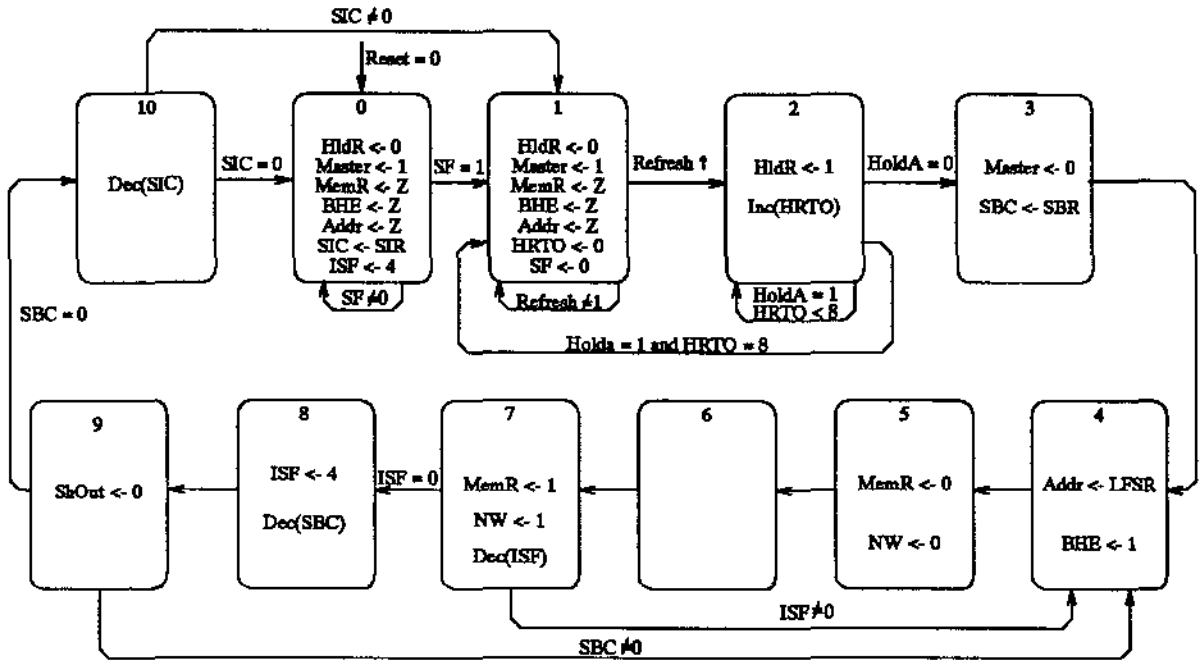


Figura 4.8: Diagrama da máquina de estados

- **SIR** (*Size of Image Register*): Define o tamanho da imagem e é carregado com a quantidade de blocos que compõem a imagem. Um bloco é dimensionado de tal forma que possa ser lido completamente num intervalo entre dois ciclos de *refresh* ($\simeq 15\mu s$).
- **SBR** (*Size of Block Register*): Define o tamanho de um bloco e é programado com o número de grupos de 4 palavras que constituem um bloco.
- **SF** (*Start Flag*): É programado junto com o segundo *byte* de SIR e é usado pela máquina de estados para iniciar seu ciclo.

O LFSR (*Linear Feedback Shift Register*) é um circuito que gera seqüências pseudo-aleatórias [Côr91]. Tem por função gerar a cada *clock* um endereço aleatório dentro do espaço de endereçamento da imagem, baseado nos registradores BIR e LLR.

A máquina de estados controla o acesso ao barramento, gera os sinais de leitura de memória (DMA) e escrita dos neurônios (na matriz) de modo sincronizado com o sinal de *refresh* do barramento. A imagem tem que ser lida, da memória, em blocos, porque a cada 15 μs o barramento deve ser devolvido ao PC para que os circuitos de *refresh* de memória dinâmica possam funcionar, justificando assim, a sincronização da máquina de estado com o sinal de *refresh*.

O diagrama de estados é mostrado na figura 4.8. A ativação do sinal *Reset* leva a máquina ao estado 0 e lá permanece até o SF ser ativado. Neste estado alguns sinais são convenientemente

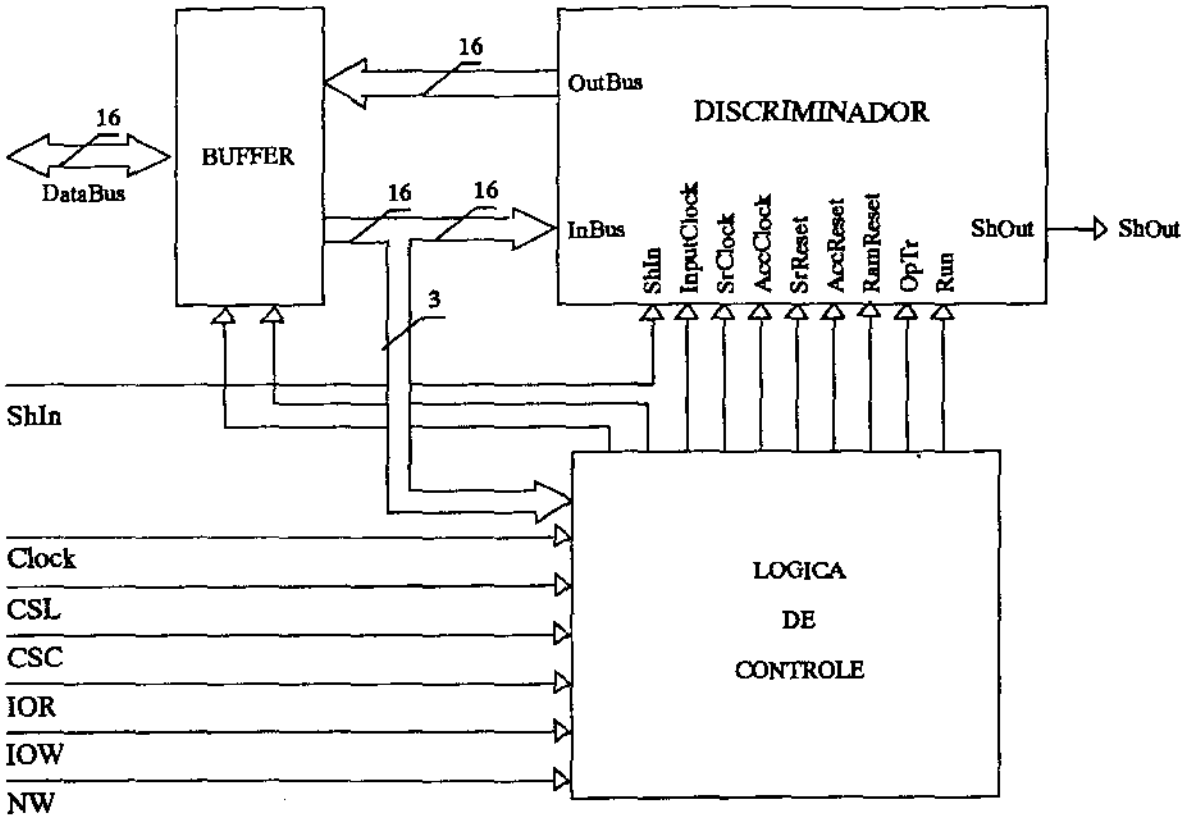


Figura 4.9: Arquitetura do Chip.

ajustados e os contadores SIC (*Size of Image Counter*) e ISF (*Input Shift Full*) são carregados. No estado 1 o contador HRTO (*Hold Request Time Out*) é zerado e o sinal *refresh* é monitorado. Quando o ciclo de *refresh* acaba a máquina passa para o estado 2, no qual requisita o barramento ativando HldR (*Hold Request*) e permanece neste estado monitorando a resposta da CPU (*HoldA*) por no máximo 8 *clocks* ($HRTO < 8$). caso a resposta seja positiva ($HoldA = 0$) a máquina passa para o estado 3, caso contrário passa para o estado 1 reiniciando o ciclo de requisição de barramento.

Uma vez no estado 3 o sinal *Master* é ativado indicando que o cartão já tem o controle do barramento. O contador SBC (*Size of Block Counter*) é carregado. O laço formado pelos estados 4 ao 7 corresponde a leitura de um grupo (4 palavras). Este laço é controlado pelo ISF. O laço mais externo (Estados de 4 a 9) corresponde a leitura de um bloco o qual é controlado pelo SBC. No estado 10 o contador SIC é decrementado e caso ainda reste algum bloco ($SIC \neq 0$) o controle é passado para o estado 1 para mais um ciclo de leitura de bloco, caso contrário, a máquina passa para o estado 0 e lá permanece até o SF ser ativado.

A estrutura interna do circuito integrado (ver figura 4.9) foi organizada em três blocos:

- *Buffer* de *I/O*
- Unidade de Controle
- Discriminador

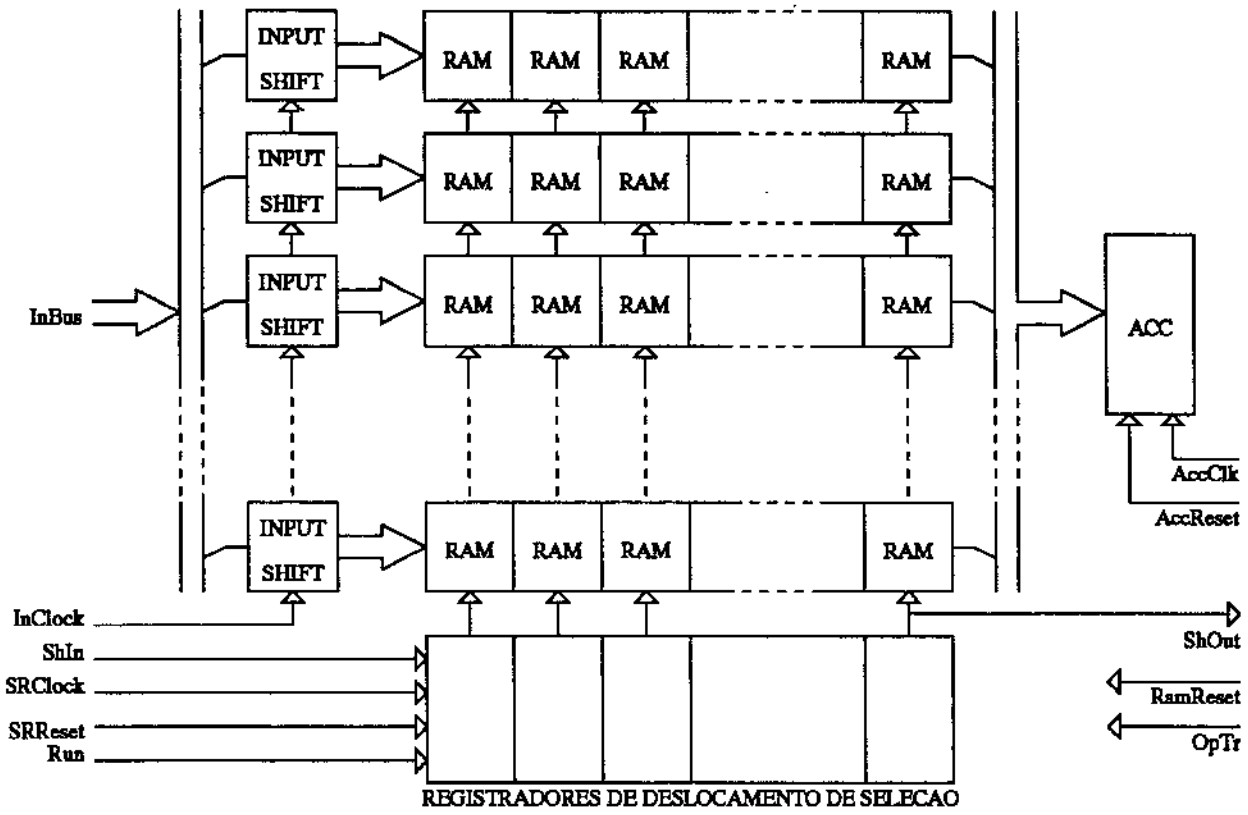


Figura 4.10: Arquitetura do Discriminador.

O primeiro bloco (*BUFFER*) controla o acesso aos barramentos internos do circuito integrado pelo o barramento do cartão. O segundo bloco (lógica de controle) contem um registro de controle de três *bits*, programável pela CPU que auxiliam no controle do modo de operação e *reset* dos componentes do discriminador. A unidade de controle também gera todos os sinais de controle do *Buffer* de *I/O* e de *clock* dos componentes internos do discriminador. O Discriminador é composto dos seguintes blocos (ver figura 4.10):

- Registrador de Deslocamento de Entrada (*INPUT SHIFT*)
- Matriz de RAMs (Neurônios)
- Registrador de Deslocamento de Seleção
- Acumulador (*ACC*)

A ordem aleatória dos *pixels*, a nível de palavra, é garantida através da bloco de controle do cartão, que com o auxilio de um LFSR gera endereços aleatórios de memória durante a leitura da imagem. Já a nível de *bit*, tal ordem é garantida através do registrador de deslocamento de entrada, que retém um *bit* de cada palavra lida. A cada grupo de quatro palavra os registradores de deslocamento retêm informações completas dos *bits* a serem treinados ou lidos (operação). O

registrador de deslocamento de seleção tem por função selecionar uma coluna na matriz de RAMs, enquanto os *pixels*, já previamente armazenados no Registrador de entrada, endereçam as células das RAMs que foram selecionadas.

O bloco Acc (Acumulador) tem por função contabilizar o número de RAMs ativas. Ele contem um registrador interno que armazena o resultado desta contagem, o qual pode ser lido pela CPU do PC para verificar se a imagem foi reconhecida ou não, em função do valor do limiar de reconhecimento, a ser definido pelo usuário.

4.4 Especificação do Sistema

O primeiro passo para especificação do sistema foi a validação da técnica usada pela Wisard. Mesmo tendo sido implementada numa versão discreta, precisou-se avaliar a funcionalidade e poder de discriminação desta técnica, em função do tamanho das RAMs, em diversos tipos de imagens.

4.4.1 Validação da Idéia

Foi elaborado um programa em Pascal para simulação da idéia aplicada no Wisard. O programa foi feito com o único objetivo de validar a técnica usada (*n-tuple*) e não tem nenhum envolvimento a nível de *hardware*. A principal vantagem deste tipo de validação é sua capacidade de manipular maior quantidade de dados (imagens maiores) num menor tempo de simulação, já que uma descrição comprometida com características de *hardware*, tal como temporização dos sinais, teria um tempo de simulação demasiadamente longo para imagens de tamanho pratico. O programa é totalmente parametrizável, permitindo que vários tipos de imagens possam ser usadas sob diversas condições. Um maior detalhamento e resultados de simulação deste programa encontra-se no capítulo 6.

Uma vez validada a técnica utilizada, o passo seguinte foi a especificação do sistema. Tal especificação teria que ter um certo nível de abstração de detalhes, concentrando mais na funcionalidade do sistema.

4.4.2 Validação Comportamental do Sistema

Todos os blocos funcionais, descritos, foram especificados em VHDL tomando como base modelos comportamentais que visam principalmente a validação da lógica e temporização dos sinais de dados e controle. Cada bloco é representado por uma *ENTITY* [1]. Numa modelagem em alto nível pode-se representar o sistema por uma única *ENTITY* mas optou-se por um nível intermediário que evidencia a organização do *hardware*.

O barramento do PC foi modelado de modo a cobrir os aspectos importantes do funcionamento do cartão. Apenas os ciclos de leitura e escrita em dispositivos de entrada e saída, leitura de memória, e requisição de barramento foram modelados, visto que: (1) os demais ciclos não ocorrem sobre o cartão, (2) uma emulação completa do PC-AT tornaria as simulações desnecessariamente lentas, e (3) não adiciona nenhuma informação importante para o projeto.

O modelo deste barramento pode ser resumido na figura 4.11. Ele inicia com atribuições concorrentes [2] de sinais de comportamento simples, tais como *Clock*, *Reset*, e *refresh*. Tais declarações

[1] Em VHDL, definem a *interface* de um circuito com o mundo externo

[2] Do ponto de vista de hardware, conectam dois sinais entre si.

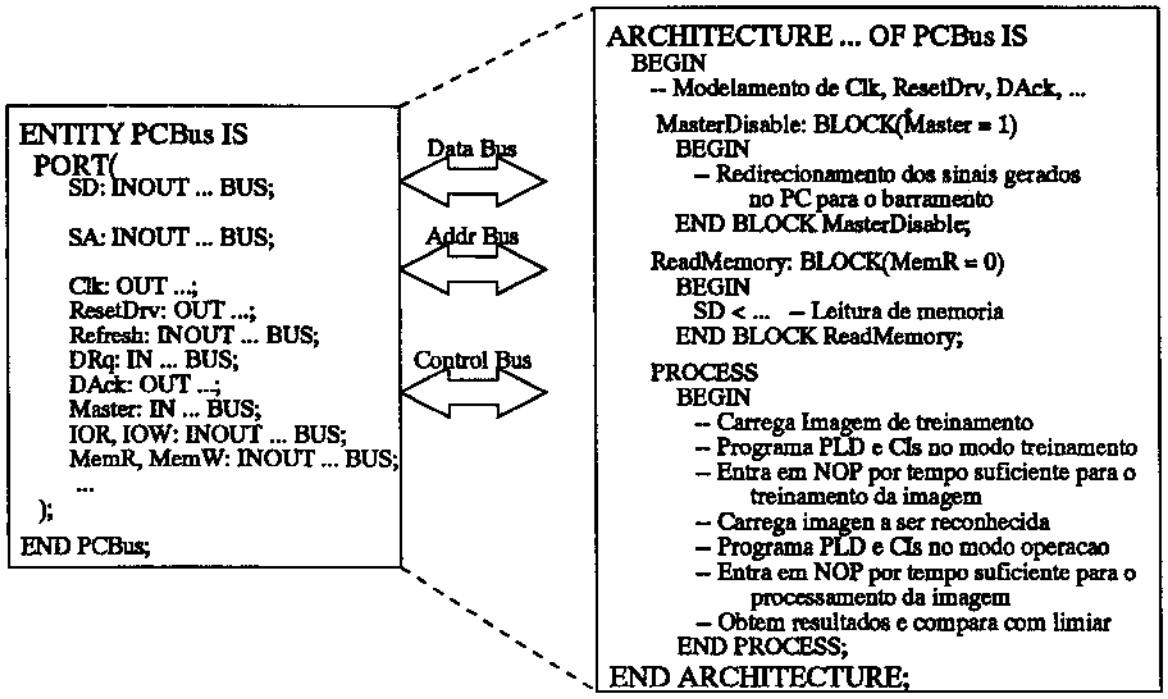


Figura 4.11: Modelagem do barramento do PC em VHDL.

definem o comportamento desses sinais. Em seguida dois *blocks* [3] controlam o barramento e a memória. O primeiro *block* isola o barramento dos sinais internos do PC, quando a placa tem o controle do barramento. O segundo *block* controla o acesso da memória ao barramento durante uma operação de leitura. Tem-se ainda um *process* [4] que comporta-se como um *software* de controle programando, gerenciando e obtendo os resultados processados pelo cartão. As ações do *process* que forçam a CPU a entrar em NOP, permitem que o cartão tenha tempo de tomar o barramento e treinar/reconhecer a imagem.

O bloco de controle do cartão (PLD) foi descrito através de dois *process* (figura 4.12). O primeiro executa a decodificação de endereços de *I/O* ativando os sinais CSL e CSC ou escrevendo os dados num dos registros internos de programação dependendo do endereço decodificado. Este *process* também gera o sinal IOChRdy após a decodificação. O segundo *process* executa as funções da máquina de estados cujo diagrama de transição de estados está apresentado na figura 4.8. O LFSR é modelado através de um função que executa a lógica do LFSR quando é chamada de dentro do *process* que modela a maquina de estados. Por conveniência esse detalhe não é mostrado na figura 4.12.

No bloco de controle do discriminador (figura 4.13) existem dois *process*. O primeiro verifica se o *chip* foi selecionado (CSL_i e CSS_j) e controla os ciclos de escritas no seu registrador de controle monitorando o sinal IOW. O registrador de controle tem 3 *bits*. O *bit* 0 controla a ativação dos sinais de *reset* de todos os componentes do discriminador (RAMs, acumulador e registradores de

[3] *Block* é uma construção do VHDL que só é executada quando uma condição (*guarded*) é satisfeita.
[4] Construção do VHDL que permite a modelagem de circuitos através de comandos seqüenciais.

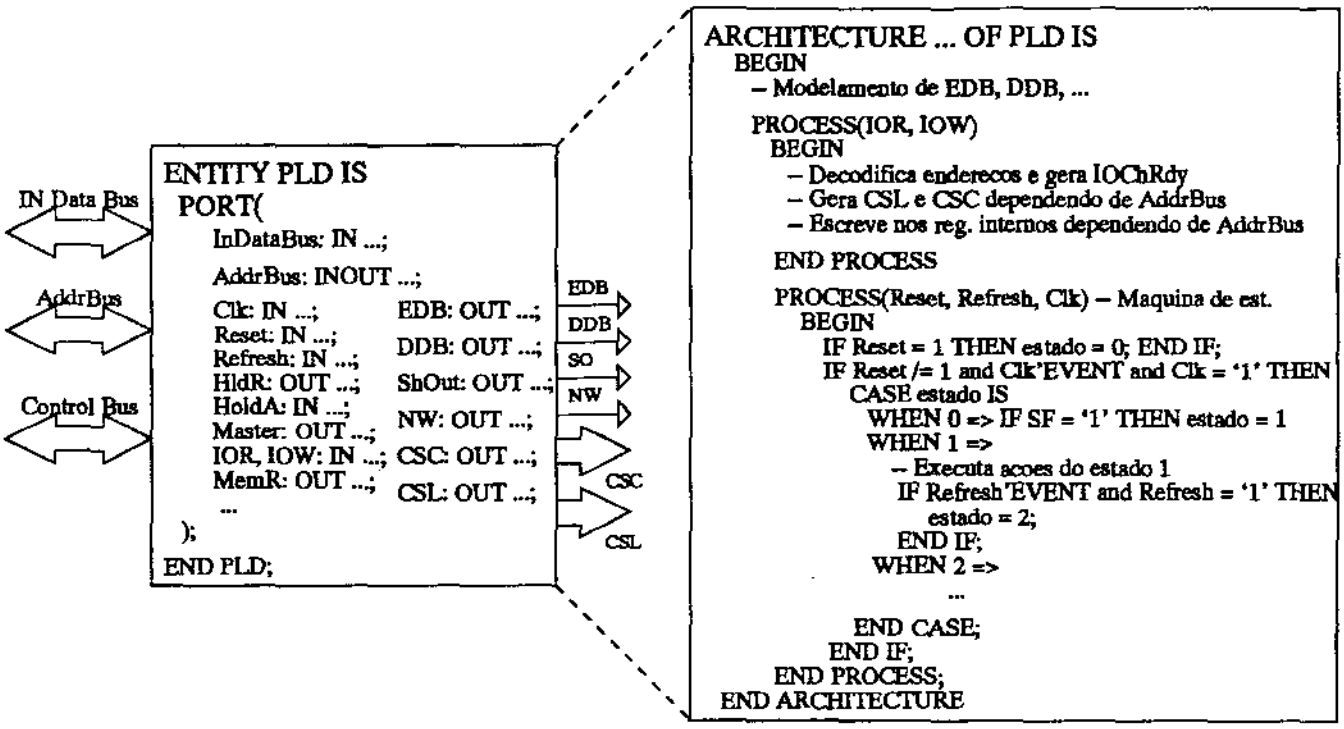


Figura 4.12: Modelagem da PLD de controle em VHDL.

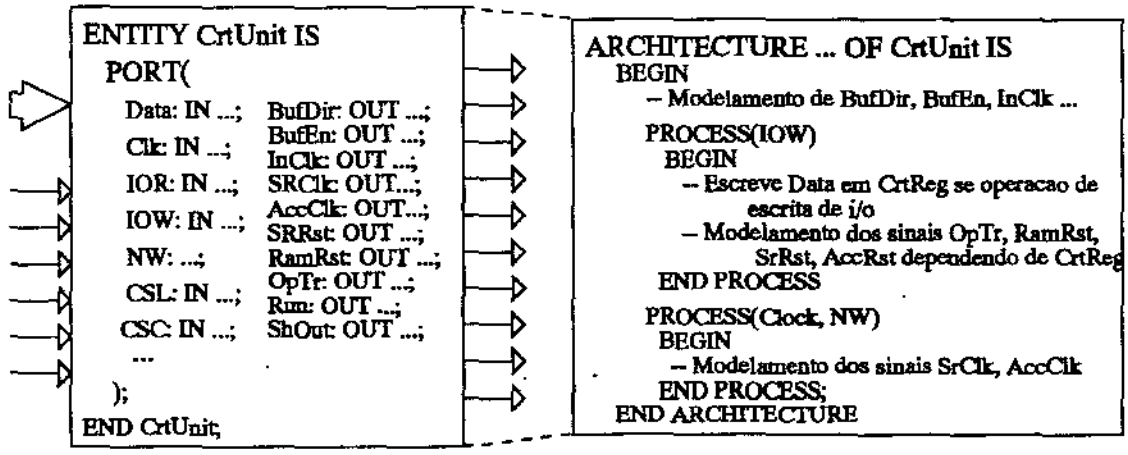


Figura 4.13: Modelagem do bloco de controle do discriminador em VHDL.

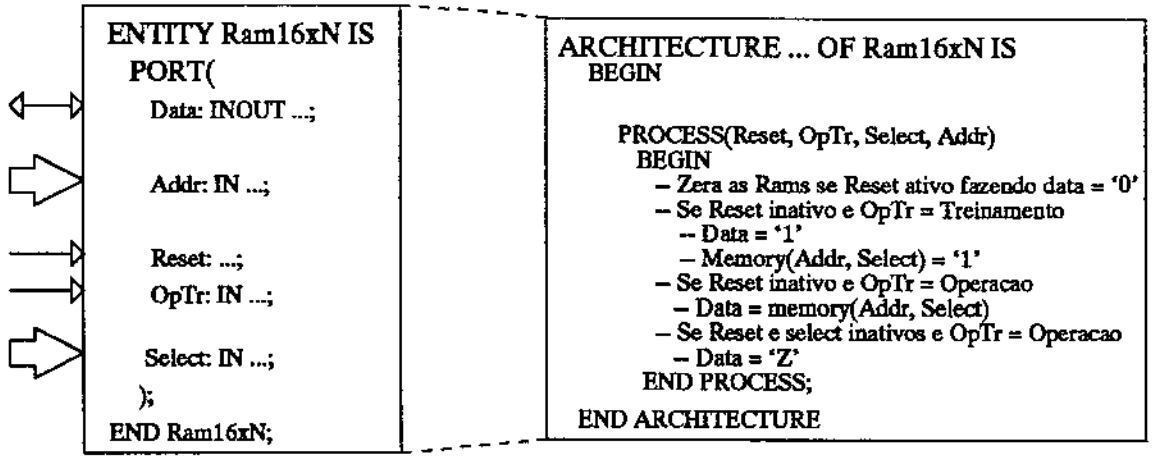


Figura 4.14: Modelagem das RAMs de uma linha do discriminador em VHDL.

seleção) e deve ser ativado quando o *chip* é programado para o modo de treinamento. O *bit* 1 define o modo de funcionamento (operação = '1' e treinamento = '0'). O *bit* 2 controla a ativação dos sinais de *reset* do acumulador e dos registradores de seleção quando o *chip* é programado para o modo de operação. O segundo *process* gera os sinais de *clock* dos registradores de seleção e do acumulador de modo síncrono com o *clock* externo. A carga dos dados nos registradores de entrada é feita pelo próprio sinal NW e a cada 4 bordas de subida desse sinal é gerado um pulso de *clock* para os registros de seleção e em seguida um pulso no acumulador.

As funções das RAMs do discriminador (figura 4.14) são modeladas através de um único *process* que monitora mudanças dos sinais *addr*, *sel*, *reset* e *optr* (barramento de endereço e seleção, *reset* e modo de funcionamento das Rams, respectivamente). Qualquer mudança nesses sinais ativa o *process* que verifica primeiro o sinal de *reset* e zera as RAMs se for apropriado senão verifica o modo de funcionamento e se for treinamento leva o nível lógico do barramento de dados a 1 e escreve este valor na posição de memória endereçada por *addr* na RAM selecionada por *sel*. Se o modo de funcionamento for operação o *process* atribui ao barramento de dados o valor da posição de memória endereçada por *addr* na RAM selecionada por *sel*. O *process* prevê o caso de nenhuma RAM está sendo selecionada pelo sinal *sel* e neste caso ele coloca o barramento de dados em alta impedância.

Capítulo 5

Implementação

5.1 Introdução

Este capítulo apresenta a implementação do sistema proposto anteriormente, detalhando seus projetos lógicos, elétrico e físico. A seção 5.2 situa essas três fase do projeto no contexto de uma metodologia, cujas etapas são sucintamente descritas, apresentando os objetivos e a natureza das atividades desenvolvidas em cada etapa. Os recursos de EDA[1], utilizados ao longo do projeto, são apresentados no final da seção, onde estão resumidos ao lado de um diagrama de fluxo de projeto (figura 5.1).

A seção 5.3 apresenta duas sugestões de implementação para o circuito de controle externo (também citado como PLD em função de sua implementação mais provável). A primeira alternativa sugere o uso de dispositivos de lógica programável de alta densidade (FPGA/CPLD), possibilitando uma implementação rápida, segura e de baixo custo. A segunda alternativa sugere o uso de um microcontrolador na emulação dos sinais gerados pelo circuito de controle externo, proporcionando maior flexibilidade, embora haja um custo no desempenho.

O projeto lógico do ASIC é apresentado na seção 5.4, através de diagramas de blocos funcionais, com a descrição das soluções adotadas para os problemas mais relevantes que foram detectados durante a validação desta fase. A seção 5.5 detalha a descrição dos blocos funcionais dedicados, a nível de transistores, justificando por exemplo o uso de transistores de passagem ou inversores de três estados. O detalhamento de blocos padrões já conhecidos, tais como registradores de deslocamento, portas lógicas etc, foram suprimidos, podendo ser facilmente encontrados na literatura técnica da área, a exemplo de [WE89]. É conveniente salientar que muitos dos sinais ativos em nível baixo, encontrados nos diagramas, foram nomeados segundo ambas as convenções: (1) uso de um prefixo 'n', (2) uso de uma barra horizontal acima do nome. Assim, os símbolos nSS e $\overline{SS}$ representam o mesmo sinal.

Por fim, o projeto físico é comentado na seção 5.6, apresentando o posicionamento das células da arquitetura e da unidade de controle (interna ao *chip*), o *layout* de algumas células e o *layout* geral do protótipo que poderá ser encaminhado ao projeto multiusuário PMU para fabricação.

[1] *electronic design automation*: recursos computacionais destinados a automação de projetos eletrônicos.

5.2 Metodologia e Recursos Utilizados

Projetos em VLSI tem um grau de complexidade elevado e o custo de um erro de projeto é muito alto. Assim visando minimizar tais custos, os projetos em VLSI devem ser executados em fases bem definidas entre as quais existem fases de simulações e de comprovação das especificações pretendidas. Conceitos como hierarquia e abstração tem sido usados com sucesso nos métodos que visam simplificar o projeto [WE89].

Existem alguns aspectos que devem ser observados, com diferente grau de importância de acordo com a aplicação, tais como:

- Desempenho (velocidade, potência, função);
- Tamanho do *die*[2];
- Temporização (*Timing*) do projeto;
- Facilidade de uso em aplicações;
- Facilidade de geração do teste e testabilidade.

Há um compromisso contínuo ao longo do projeto em obter-se os resultados mais adequados para todos estes parâmetros. Assim as ferramentas e metodologias usadas para um 'chip' particular, dependem destes aspectos. O aspecto principal (final) é a 'performance' mas outros parâmetros também são muito importantes. O tamanho do 'die', por exemplo, pode afetar o rendimento (*yield*) o qual economicamente é muito importante.

As fases de um projeto VLSI, de uma maneira geral, tem se conservado, apresentando pequenas variações entre os diversos estilos, particulares à cada projetista. As seções a seguir sumarizam as fases [Côr91] de um projeto VLSI.

5.2.1 Especificação e Validação Comportamental

Nesta fase, o objetivo do projeto (o problema) é estudado e equacionado. Os sinais necessários para interface com o mundo externo são analisados e definidos e as inter-dependências entre eles são estabelecidas através de equações booleanas, diagrama de estados etc, os quais são elaborados por meio de uma linguagem de descrição de *hardware*. Muitos aspectos estruturais são abstraídos permitindo ao projetista uma rápida validação das funções do sistema proposto. Por exemplo, um sinal abstrato pode ter valores, tais como 'NOP', 'ADD', 'SUB' etc, denotando as instruções de uma CPU. Em uma descrição estrutural posterior este sinal pode ser implementado como um barramento com valores binários alimentado por um decodificador de instruções. Esse tipo de recurso simplifica a descrição do sistema, tornando-a fácil de ser entendida, e agiliza a validação funcional do sistema.

Muitas características, tais como a capacidade de *drive*[3] e tempo de propagação dos sinais, podem ser descritas sob a forma de parâmetros. Valores estimados através de equações podem ser atribuídos a tais parâmetros para as simulações iniciais, ou podem ser extraídos de descrições físicas através do mecanismo de *back annotation* existente nas ferramentas de apoio, permitindo simulações mais apuradas (mais realistas) das funções do sistema.

[2]Porção de uma pastilha de silício onde o circuito integrado é difundido.

[3]Quantidade de circuitos que uma saída de um outro circuito pode alimentar

5.2.2 Projeto Lógico

Projeto dos circuitos a nível de portas lógicas, elemento de memória etc. Os sinais são um tipo de representação binária cujos valores além dos estados binários '0' e '1', prevêem os estados 'X' para um estado desconhecido e 'Z' para um sinal desconectado de sua fonte. A síntese do projeto lógico pode ser obtida por descrição manual do circuito através de diagramas esquemáticos ou através de uma descrição textual numa linguagem apropriada, ou ainda, através de ferramentas para síntese automática, que recebem uma descrição das especificações (produzidas na fase anterior) e produzem uma descrição do circuito sintetizado, geralmente sob a forma de um *netlist*.

5.2.3 Validação lógica

Consiste de simulações efetuadas sobre o modelo de descrição do circuito através de ferramentas chamadas de simuladores lógicos, visando verificar se o circuito, como projetado, atende as especificações. Dependendo dos resultados desta fase o projeto pode avançar para fase seguinte (se a validação foi positiva) ou voltar à anterior e até mesmo para a fase de especificação.

5.2.4 Projeto Elétrico

Nesta fase todos os elementos (portas lógicas, 'latchs' etc) são descritos a nível de transistores. Os transistores são dimensionados de modo a atender as especificações de atraso e de cargas dos circuitos. Geralmente existem bibliotecas de tais elementos, onde cada símbolo lógico é associado a um circuito elétrico previamente elaborado com características de atraso e de cargas padronizadas. O uso destas bibliotecas e das ferramentas de apoio convenientes pode facilitar muito o andamento do projeto elétrico.

5.2.5 Validação Elétrica

Similar a fase de validação lógica, com a diferença que as simulações nesta etapa são a nível de transistor. Muitos problemas que não podem ser detectados na validação lógica são detectados nesta fase visto que a simulação lógica abstrai estes tipos de problemas. Por exemplo, problemas relacionados com capacidade de *driving*, quando um elemento (porta), devido a seu projeto elétrico, não consegue alimentar as entradas dos elementos que estão conectadas ao mesmo. Problemas com o nível do sinal e ruídos também podem ser detectados nesta fase.

Os sinais envolvidos nesta etapa são do tipo analógicos e os simuladores analógicos têm como base a resolução do sistema de equações relacionados com as tensões correntes e resistências (ou condutâncias) do circuito. Este simuladores são usados para analisar detalhadamente pequenos circuitos, como células lógicas, devido ao elevado tempo de simulação. Para que o sistema seja simulado como um todo, uma caracterização dos elementos lógicos é efetuada de modo a obter seus principais parâmetros (velocidade, atrasos, cargas etc) e levados de volta a etapa de simulação lógica (*back annotation*).

5.2.6 Projeto Físico

Chama-se de projeto físico ao processo de concepção das diversas máscaras que vão ser usadas na fabricação do circuito integrado. Existem alguns métodos [HR91] bem conhecidos e que devem ser escolhidos de acordo com as características do projeto:

- **'Full Custom'**: Consiste na descrição de todas as máscaras (tipicamente 13, em CMOS) referentes as diversas camadas sem o uso de células pré-validadas. O projetista manipula as descrições das máscaras a nível de transistores, otimizando a utilização da área de silício e o desempenho do circuito. A alta densidade de transistores por *chip* é obtida pela construção e posicionamento manual das células. Por conseqüência, o custo e tempo de projeto são elevados, quando comparados com os outros métodos. Embora apresente um alto custo na fase de projeto, o custo de produção é pequeno, devido a melhor utilização da pastilha de silício e já que o método dispensa a utilização de biblioteca células que são geralmente fornecidas pelos fabricantes. Este método é recomendado para produções acima de 100 mil unidades, onde o custo por unidade é mais reduzido que em outros métodos, ou em casos onde alta densidade é essencial, ou mesmo quando estruturas especiais, não disponíveis em bibliotecas, são necessárias.

- **'Standard Cell'**: Consiste na utilização de circuitos com funções pré-validadas, com um nível LSI/VLSI de complexidade, que são agrupados e interconectados. Tais circuitos são padronizados e fornecidos pelos fabricantes de circuitos integrados através das chamadas bibliotecas de células padrões. A manipulação do projeto físico no nível de células, resulta em tempo e custo de projeto menores em relação ao método anterior. A densidade de transistores ou eficiência na utilização de área é menor devido a exigência de uma área extra para interconexão das células e sua padronização. *'Standard Cell'* é recomendado para um volume de produção entre 10 a 100 mil unidades.

- **'Gate array'**: Consiste na utilização de pastilhas de silício com transistores previamente difundidos que requerem apenas de uma a três máscaras de metal. Sua estrutura é formada por uma matriz de transistores, barramentos de alimentação e *pads* de I/O. As colunas da matriz de transistores são configuradas para formarem funções lógicas básicas escolhidas de uma biblioteca de células. O projeto físico é reduzido à escolha de blocos lógicos a serem configurados e descrição das máscaras de metal que conectam as células entre si e com os *pads* de I/O. Embora possa prover 200 mil portas lógicas por *chip*[HR91], apenas 70 a 90% dessas portas lógicas são usadas devido a dificuldade de criar canais de roteamento entre as células. O método apresenta um baixo custo e tempo de projeto e alto custo de produção relativamente aos outros método, possui uma baixa densidade de transistores utilizados e é recomendado para prototipagem rápida e para projetos com um volume de produção menor que 10 mil unidades.

É comum também o uso de métodos híbridos onde partes de um projeto são feitas em 'standard cell' e outras partes em 'full custom'. Por exemplo em uma arquitetura os projetistas podem optar por fazer a unidade de controle em 'full custom', por ela ser irregular e exigir alto desempenho, enquanto o resto do projeto em 'standard cell'. A tabela 5.2.6 resume as principais características dos métodos citadas acima.

O *netlist* é extraído da descrição das máscaras ao final do projeto físico de cada célula e uma nova simulação e caracterização assegura a funcionalidade e especificações das células. Ao final de todo o projeto físico uma verificação entre o *layout* e as descrições elétricas do circuito é efetuada com o auxílio do mecanismo chamado de LVS (*Layout Versus Schematic*) geralmente disponível nas ferramentas orientadas à edição de máscaras, onde podem ser detectados eventuais problemas de interconexões entre as células.

	Full custom	*Standard cell*	*Gate array*
Custo ou tempo de projeto	Alto	Medio	Baixo
Custo de produção	Baixo	Medio	Alto
Densidade de transistores	Alta	Media	Baixa
Flexibilidade	Alta	Media	Baixa
Número de máscaras	13	13	3
Volume adequado de produção	> 100.000	100.000 a 10.000	< 10.000

Tabela 5.1: Características dos métodos *full custom*, *standard cell* e *gate array*

5.2.7 Fabricação de protótipo, teste e caracterização

O protótipo é fabricado e testado para verificar se o chip atende as especificações e uma caracterização é efetuada levantando-se sua faixa de operação com relação a vários parâmetros, tais como freq
"uência e potência máxima de operação. A fabricação do protótipo em questão pode ser viabilizada através do 'Projeto Multiusuário' (PMU) brasileiro sob a coordenação do 'Centro Tecnológico para Informática' (CTI), envolvendo várias universidades e empresas nacionais.

5.2.8 Produção

Nesta fase os chips são produzidos em larga escala e posteriormente são submetidos ao teste de produção para poderem ser comercializados. São utilizados equipamentos de teste automático de alta velocidade.

Algumas destas fases podem ser executadas concorrentemente, (quando o projeto é desenvolvido por uma equipe) e outras podem ser repetidas várias vezes como em um processo de depuração. A figura 5.1 mostra as etapas e o fluxo do projeto. As etapas de validação/simulação visam detectar o mais cedo possível as falhas, minimizando o fluxo nas linhas pontilhadas (ver figura), isto é, evitando que as falhas detectadas numa etapa, desvie o fluxo do projeto para etapas antecessoras distantes.

A figura 5.1 também mostra, ao lado de cada etapa, as ferramentas utilizadas. O projeto deveria ser totalmente desenvolvido no ambiente de automação de projetos eletrônicos da Mentor Graphics [Men93d] mas devido à falta de recursos, executou-se o projeto físico com dois programas de domínio público produzidos em *Berkeley*: o simulador elétrico Spice [Dep90] e o editor de *layouts* Magic com DRC[4] embutido [Dep85]. O *Design Architect* [Men93a][Men93b] é uma ferramenta dedicada à projetos que suporta, entre outras coisas, edição e compilação de descrições em VHDL (padrão 1076 do IEEE) e de esquemáticos. O *QuickSim* [Men93f] é um simulador lógico que aceita ambas as formas descritivas de circuitos produzidas pelo *Design Architect*. O *IC Station* [Men93c] é a ferramenta especialista em *layout* do ambiente Mentor Graphics. O *Design Architect* e o *QuickSim* também integram este ambiente juntamente com outras ferramentas [Men93e].

[4] *Design Rule Checker*

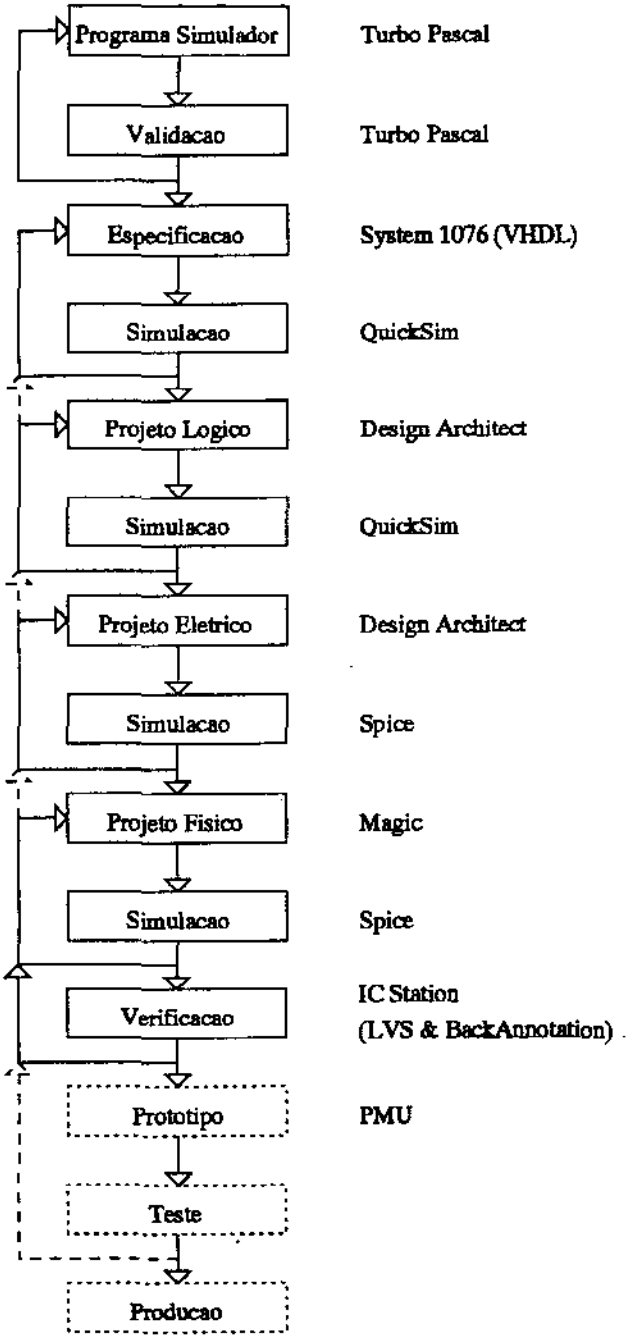


Figura 5.1: Etapas e fluxo do projeto

5.3 Implementação do Sistema

Do ponto de vista da implementação, os dois principais componentes do sistema são o *chip* dedicado e a PLD. A implementação do *chip* compreende as etapas de projeto lógico, elétrico e físico. As seções subseqüentes sumarizam as três etapas citadas, procurando mostrar não apenas os resultados finais mas a evolução de cada fase, resaltando os pontos relevantes. Informações adicionais encontram-se no apêndice B, que apresenta material relativo ao projeto do *chip* incluindo desde especificação comportamental em VHDL até detalhes da implementação física. Embora a implementação de todo o sistema não esteja no escopo deste trabalho, algumas sugestões para a implementação do controlador do cartão são apresentadas.

5.3.1 Implementação do Circuito de Controle Externo

O circuito externo de controle tem sido chamado genericamente de PLD em função de sua implementação mais provável. Uma grande variedade de PLDs são atualmente disponíveis no mercado mas a escolha do dispositivo adequado não é uma tarefa trivial. O projetista deve fazer sua opção levando em conta as necessidades do sistema e o custo dos dispositivos. O uso de PALs, por exemplo, forçaria a divisão do circuito de controle em vários *chips*, resultando numa maior complexidade do *layout* do cartão. As PLDs mais indicadas para esse tipo de circuito são as CPLDs e FPGAs [Lei93] [Wal93]. Elas possuem maior número de células, podendo implementar a quantidade de elementos de memória requerida pelo circuito.

O primeiro item a ser observado na escolha de uma FPGA/CPLD é o número de pinos de I/O, que deve ser igual ou maior ao número de sinais de I/O do circuito. No caso do circuito de controle externo apresentado na figura 4.7 existem 15 sinais de entrada, 33 de saída e 16 de I/O, totalizando 64 pinos. A quantidade de sinais de saída pode ser reduzida para 23 se os sinais CSL e CSC forem codificados em binário. Isso permite a escolha de uma FPGA/CPLD de menor custo, com um mínimo de 54 pinos de I/O. Os sinais codificados em binário podem ser decodificados com um CI da família 74. Outro ponto a ser analisado é o número de elementos de memória existentes no circuito. Para o circuito em questão, são necessários 56 elementos de memória para os registradores de programação, 24 para o LFSR e 32 para a máquina de estado (ver diagrama de estados, figura 4.8), totalizando em 112 elementos de memória. Na pratica, esse número de elementos de memória deve equivaler de 50 a 80% do número de células da FPGA/CPLD escolhida.

	ACT 1010	MAX 5128	LCA 3030	iFX 780
Pinos de I/O	57	60	???	62
Quantidade de células	295 módulos lógicos	128 macrocélulas 256 expanders	100 CLBs	80 macrocélulas
Quantidade máxima de elementos de memória	147	128	200	140
Percentagem de utilização	76%	87%	56%	140%

Tabela 5.2: Principais características de algumas FPGAs disponíveis no mercado

A tabela 5.3.1 [SJ93] apresenta algumas FPGA/CPLDs. A iFX780 da Intel está descartada por formar um máximo de 80 elementos de memória quando são necessários pelo menos 112. Das três alternativas restantes, a que apresenta maior quantidade de elementos de memória é a LCA 3030 podendo formar um máximo de 25 registradores de 8 *bits*. A ACT 1010 é uma outra alternativa possível, porém o número de elementos de memória necessários para o circuito representa 76% da sua capacidade máxima. Uma implementação com a MAX 5128 da Altera pode ser inviável uma vez que apresenta uma percentagem de utilização de 87%. Numa primeira análise a mais indicada seria a LCA 3030 seguida da ACT 1010. Isto é apenas uma ilustração de uma avaliação preliminar, quando na realidade existem inúmeras opções de PLDs disponíveis no mercado, que podem inclusive adequassem melhor a implementação do circuito em questão. Uma análise mais apurada [SJ93], ou até simulações através de *software* obtido dos fornecedores, podem auxiliar na escolha da FPGA adequada, com uma margem de segurança maior.

Uma outra implementação possível seria o uso de um microcontrolador. Os sinais gerados pelo circuito poderiam ser emulados com maior simplicidade e flexibilidade mas o sistema ficaria menos eficiente. Simplicidade pode ser obtida por exemplo com o uso do microcontrolador 8051 [Int91] que em um único *chip* incorpora CPU, memória de dados e programa, e dispositivos de entrada e saída. Flexibilidade é caracterizada quando um programa gravado na EPROM, interna ao microcontrolador, pode ser modificado para se obter o comportamento desejado. Sendo a funcionalidade descrita por um programa, perda de eficiência ocorre quando o microcontrolador busca suas instruções. Em se tratando do teste de um protótipo, flexibilidade é um fator importante devido a alta probabilidade de mudanças e ajustes no sistema, enquanto que eficiência poderia ser obtida após os teste do protótipo e posterior reprojeto.

5.4 Implementação do Chip: Projeto Lógico

Durante o projeto lógico foram descritos os blocos funcionais através de elementos lógicos primários, isto é, portas lógicas, *latchs*, *flip-flops* etc. O desenvolvimento dessa fase foi facilitado na medida que a descrição comportamental do sistema já apresentava detalhes da arquitetura e da *interface*. Na validação desta fase pôde-se detectar problemas de sincronismo entre a máquina de estados (PLD) e a unidade de controle do *chip*, e de atrasos devido a cargas nos barramentos. Na validação da unidade de controle (do *chip*) encontrou-se mais alguns problemas com relação ao funcionamento da máquina de estados da PLD, como também com a própria unidade de controle do *chip*.

5.4.1 O Discriminador

A primeira versão para o projeto lógico do discriminador desenvolveu-se como uma tradução de sua arquitetura (a arquitetura do discriminador foi descrita no capítulo 4). Versões posteriores, produzidas após algumas interações com os projetos elétrico e físico, trouxeram mudanças significativas. Algumas células novas foram introduzidas, outras modificadas, e ate mesmo blocos funcionais, como foi o caso do acumulador, totalmente reprojetados.

As células novas, que foram introduzidas para resolver problemas de cargas, atrasos e conflitos são:

- **DRV**. Os barramentos de endereços decodificados alimentados pelos blocos *'Decoder'* cortam transversalmente às RAMs do discriminador, fornecendo às mesmas os endereços que

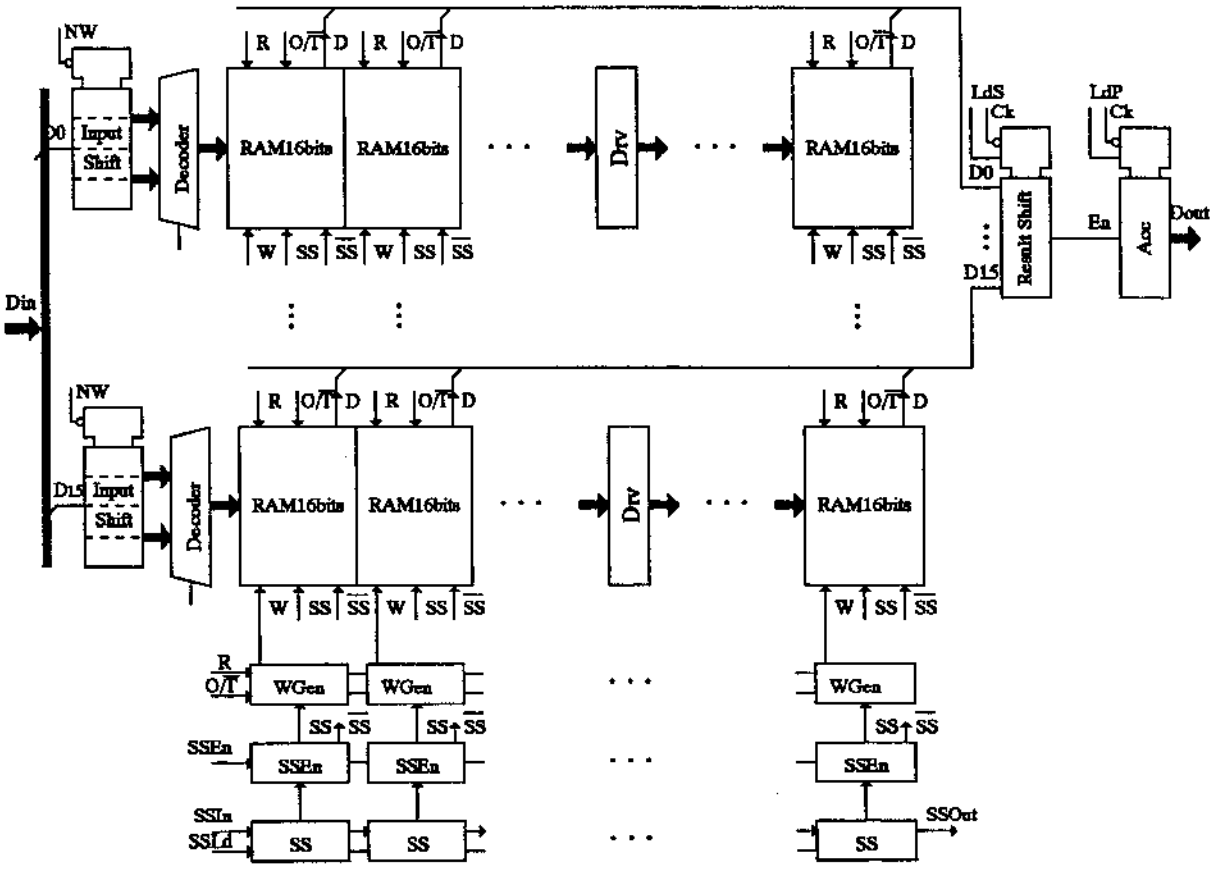


Figura 5.2: Versão final do projeto lógico do discriminador

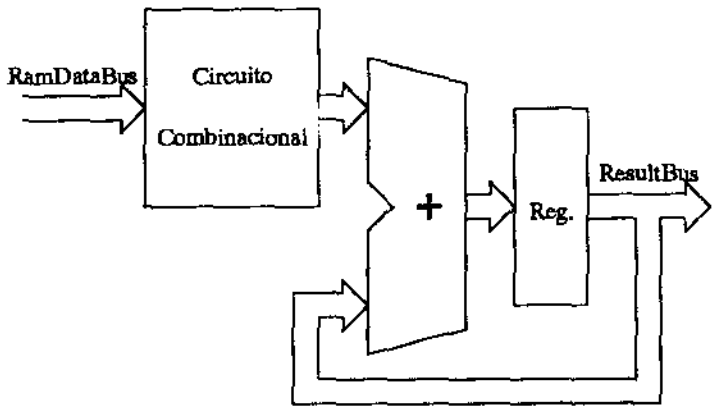


Figura 5.3: Primeira versão do acumulador: Circuito combinacional e acumulador seqüencial

selecionam uma célula de memória em cada RAM. Este barramento apresentava uma alta capacitância em suas linhas devido ao grande número de RAMs por linha do discriminador, provocando um atraso muito grande. O problema foi solucionado através do particionamento do barramento [WE89] intercalando *drivers* ('Drv') que isolam duas partições adjacentes, mantendo uma baixa capacitância em cada partição (figura 5.2).

- **SSEn.** Um problema análogo ao anterior ocorreu com os sinais de seleção que habilitam uma das colunas de RAMs por vez. Neste caso, foi suficiente a criação de uma única célula *driver*. Além de alimentar os sinais de seleção das RAMs (SS e $\overline{SS}$), as células 'SSEn' inibem estes sinais durante suas transições, evitando a seleção de mais de uma coluna, quando há uma superposição (*overlapping*) nos estados destes sinais, devido aos diferentes atrasos na propagação destes sinais no registrador de deslocamento de seleção 'SS' (figura 5.2).

- **Inv3St.** O 'Inv3St' alimenta o barramento de dados da matriz apenas quando sua coluna for selecionada. Esta célula também incorpora um circuito de auxílio dos modos de funcionamento treinamento, operação e *reset* (ver detalhes na seção 5.5).

- **WGen.** Uma célula nova WGen foi criada para auxiliar as operações de escrita e leitura nas RAMs. Ela gera o sinal W (*Write*) que abre os laços de realimentação nas células de memória durante uma escrita (treinamento) evitando conflitos (ver seção 5.5).

As células de RAMs foram modificadas várias vezes buscando-se otimizar sua área mantendo as funções lógicas desejadas. As seções 5.5 e 5.6, relativas aos projetos elétrico e físico, apresentam maiores detalhes a respeito das mudanças nas células existentes e da criação das células novas.

O acumulador, a princípio, foi projetado como um acumulador seqüencial do 'peso' do barramento de dados, como mostra a figura 5.3. O circuito combinacional gera o peso, ou melhor, um número binário correspondente ao número de *bits* do barramento de dado das RAMs que estão em nível lógico '1'. Um registrador armazena o resultado da soma dos diversos pesos computados ao longo do tempo. Verificou-se na implementação física que tal circuito consumia muita área e que era necessário uma diversidade de células muito grande para implementação do acumulador e cir-

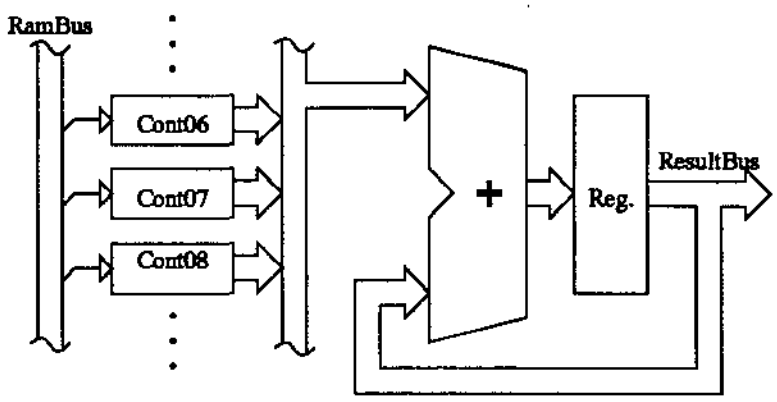


Figura 5.4: Segunda versão do acumulador: Contagem em dois níveis

cuito combinacional. Um elevado atraso no tempo de resposta do acumulador é caracterizado como conseqüência do encadeamento de células tanto no circuito combinacional quanto no acumulador.

Numa segunda versão, optou-se pela contagem individual de cada *bit* através de contadores que ao final do processamento tinham seus valores adicionados em um registrador através de um somador. A figura 5.4 apresenta esta versão. Apesar de apresentar boa regularidade, consumir menor área de silício e ter um menor número de tipos de células, nesta versão seriam necessários sinais de controle adicionais para computar o resultado final dos contadores.

O circuito atual do acumulador é uma evolução da versão anterior (figura 5.5). O funcionamento consiste em computar serialmente o valor de todos os *bits* em um único contador, enquanto novos dados estão entrando no *chip*. O circuito tem que computar 16 *bits* e para isso leva 16 períodos de *clock*. Como o tempo de entrada de novos dados nos registradores de deslocamento de entrada é de 18 períodos de *clock* as duas atividades podem ser realizadas de forma paralela, evitando que a matriz fique ociosa durante a contagem das rams ativas. A nova versão do acumulador é apresentada na figura 5.5. O contador é composto de 15 *flip-flops* tipo T em um arranjo onde cada *flip-flop* só é habilitado a mudar de estado quando o *flip-flop* anterior está habilitado e em nível lógico '1' ($T_n = T_{n-1}$ AND Q_{n-1}) formando um contador com *enable*. O sinal de *enable* é obtido de um registrador de deslocamento (RS_n na figura 5.5) que recebe os dados do barramento de saída das RAMs em paralelo, e transfere serialmente para o *enable* deste contador. Assim, enquanto os dados da ciclo atual estão entrando no chip, os resultados do ciclo anterior estão sendo contabilizados.

5.4.2 A unidade de controle

O projeto da unidade de controle foi conseqüência do detalhamento da descrição comportamental à nível lógico com algumas pequenas mudanças, ou melhor, adaptações que só puderam ser detectadas numa fase mais avançada. A principal mudança foi com respeito ao *clock*. Na descrição comportamental, a unidade de controle gerava diversos *clocks* para controlar as diversas partes da arquitetura. Durante o projeto lógico percebeu-se que isso poderia resultar em problemas de sincronismo entre as bordas dos diversos *clocks* devido aos diferentes atrasos que estes *clocks* estariam sujeitos. No projeto lógico foi adicionado a todos os elementos síncronos (elementos cujo sinal de saída é atualizado em um das bordas de um sinal de *clock*) um pino de *enable* e a unidade de

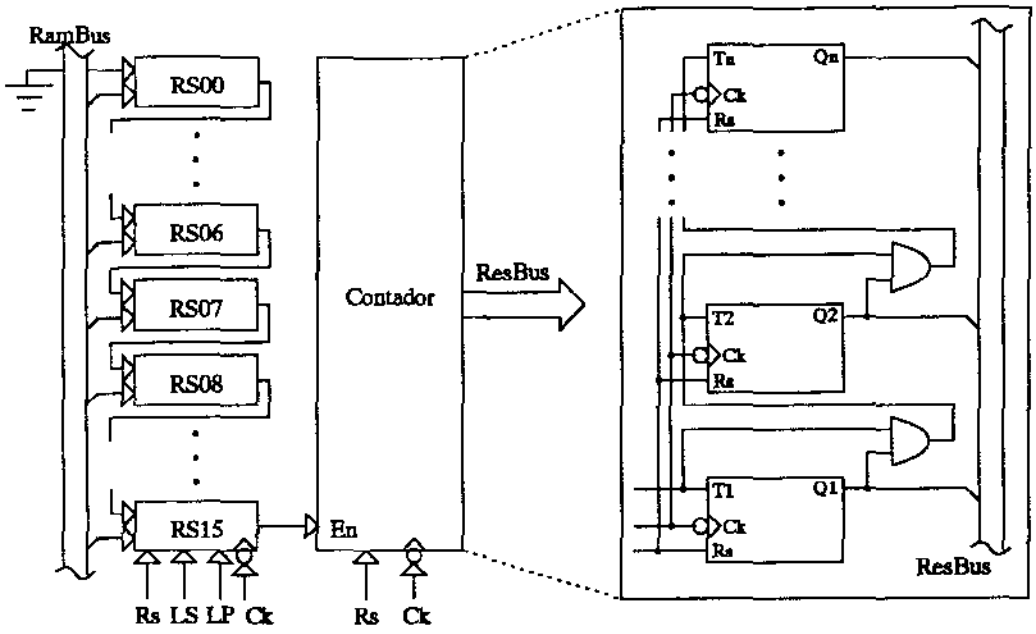


Figura 5.5: Versão atual do acumulador: Contagem serial dos *bits*

controle passou a prover estes *enables* enquanto um único *clock* passou a sincronizar tais elementos. Outra mudança diz respeito ao *reset* das estruturas do discriminador. Na descrição comportamental existiam três sinais de *reset* distintos, um para as RAMs, outro para o acumulador e outro para o registrador de seleção. Observando-se os modos de operações do chip percebeu-se que o acumulador e o registrador de seleção poderiam ter um único sinal de *reset*, simplificando o projeto lógico da unidade de controle. Outro detalhe foi a geração do complemento de alguns sinais de controle dentro da própria unidade de controle.

O diagrama lógico da unidade de controle é apresentado na figura 5.6. Ele está dividido em 5 blocos:

- O *I/O Buffer Controler* é um circuito combinacional que gera o sinal de controle de direção dos *buffers* de dados incorporados nos *pads* bidirecionais.
- Os *Internal Registers* são registradores que armazenam o modo de operação do *chip* e incluem lógica auxiliar para geração dos *resets*. Os registradores podem ser programados pela CPU e os *resets* são gerados por pequenas máquinas de estados que após a geração dos sinais de *reset* levam seus registradores correspondentes ao nível lógico '0'.
- O *ChipOn Generator* tem por função indicar quando o *chip* deve está ativo, já que vários chips podem estar ligados em série. Ele funciona através do monitoramento dos sinais de entrada e saída do registrador de seleção (SS_{In} e SS_{Out}), detectando quando o primeiro e ultimo *bit* deste registrador é ativado. O sinal *ChipOn* e ativado junto com o primeiro *bit* e desativado com o ultimo.
- O *Sync Generator* faz o sincronismo entre a entrada dos dados e a máquina de estado que gera

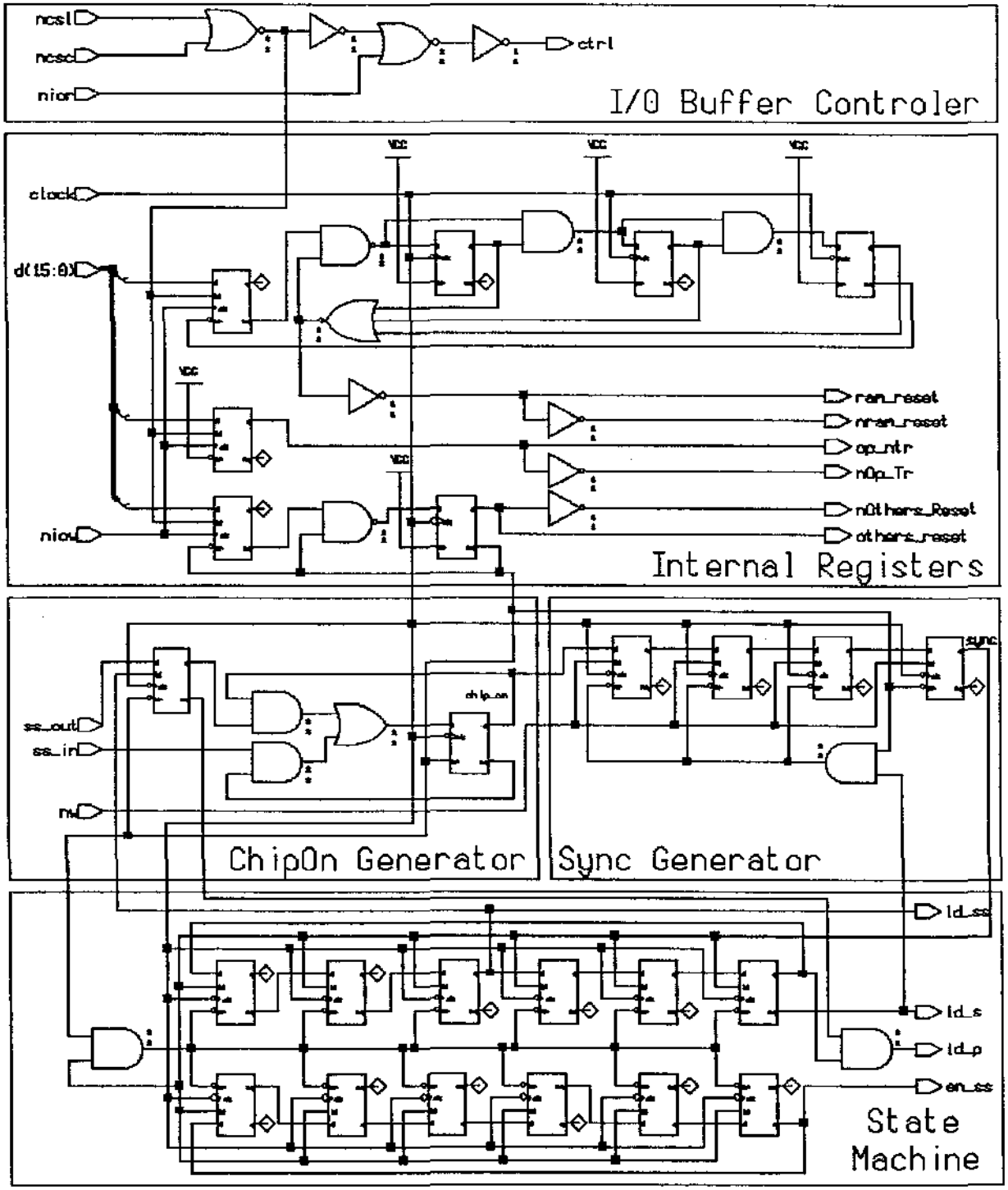


Figura 5.6: Projeto lógico da unidade de controle

os sinais de controle do acumulador e do registrador de seleção. O sinal *ChipOn* habilita seu funcionamento. Após o quarto ciclo de leitura de dados, quando os registradores de entradas estão cheios, este circuito habilita o funcionamento de tal máquina de estado. O sincronismos é efetuado através do sinal NW (Neurônio *Write*).

- A *State Machine* controla o funcionamento do registrador de seleção e acumulador, gerando uma seqüência de pulsos nos sinais LdSS, EnSS, LdP e LdS que respectivamente produzem: uma operação de *shift* no registrador de seleção; habilita a seleção da coluna de RAMs correspondente ao *bit* ativo do registrador de seleção; carrega, de modo paralelo, o barramento de dados da matriz de RAMs no registrador; que alimenta serialmente estes dados no *enable* do contador. Esta máquina de estado só entra em funcionamento quando é habilitada pelo circuito gerador de sincronismo.

As simulações deste circuito detectaram uma falha no funcionamento da máquina de estado da PLD de controle externo e outra falha na máquina de estado interna. Na máquina de estado externa verificou-se que o sinal SSOut era ativado no estado '0' e permanecia ativado todo o tempo. As simulações mostraram a necessidade de desativar este sinal após a leitura da primeira palavra de dados (estado 9) sob pena do mau funcionamento do circuito *ChipOn Generator*. A primeira versão da máquina de estados interna foi uma construção vertical, com 3 elementos de memórias e um decodificador de estados. Durante as simulações foram detectados *spikes* nos sinais gerados por essa versão, decorrente dos diferentes atrasos entre as bordas dos sinais dos elementos de memória, durante uma mudança de estado. Optou-se por uma montagem horizontal na segunda versão dispensando os circuitos decodificadores e eliminando os problemas com *spikes*. Esta versão facilita a implementação física pela reduzida diversidade de células (*flip-flops* tipo D e portas *and* de duas entradas).

5.5 Implementação do Chip: Projeto Elétrico

Nesta fase, cada componente do projeto lógico foi detalhado a nível de transistores. A maioria das células já possuíam um projeto elétrico bastante conhecido na literatura técnica. Algumas outras são mais especificas e foram cuidadosamente projetadas. Em ambos os casos, o dimensionamento dos transistores ocupou boa parte do tempo dedicado ao projeto elétrico. Após algumas interações com o projeto físico, teve-se noção das cargas que cada célula iria alimentar, possibilitando um dimensionamento mais seguro de cada célula. As dimensões dos transistores, em um primeiro passo, foram estimadas através de equações [WE89] com posteriores simulações. Como já foi dito, em alguns casos teve-se que introduzir células novas, principalmente nos circuitos de alimentação de barramentos.

A memória 'RAM16bits' é uma macro-célula dedicada com um sinal especial de *reset*. Ela é composta pelas células 'RAM1bit' e 'inv3st'. Seu diagrama elétrico é apresentado na figura 5.7. Os dados são armazenados ou lidos das 16 células 'RAM1bit' através dos laços de realimentações formados pelos inversores I1 e I2 nas respectivas células. O transistor T1 evita conflitos durante uma operação de escrita (treinamento). Os transistores T2, em cada célula de memória, funcionam como um seletor, conectando uma única célula ao barramento interno (Di), desde que apenas um dos sinais de endereço (Ao, A1, ..., A15) está ativo, exceto durante a operação de *reset*. A célula 'inv3st' tem dupla função. O inversor I1 conecta o barramento interno com o externo, quando os

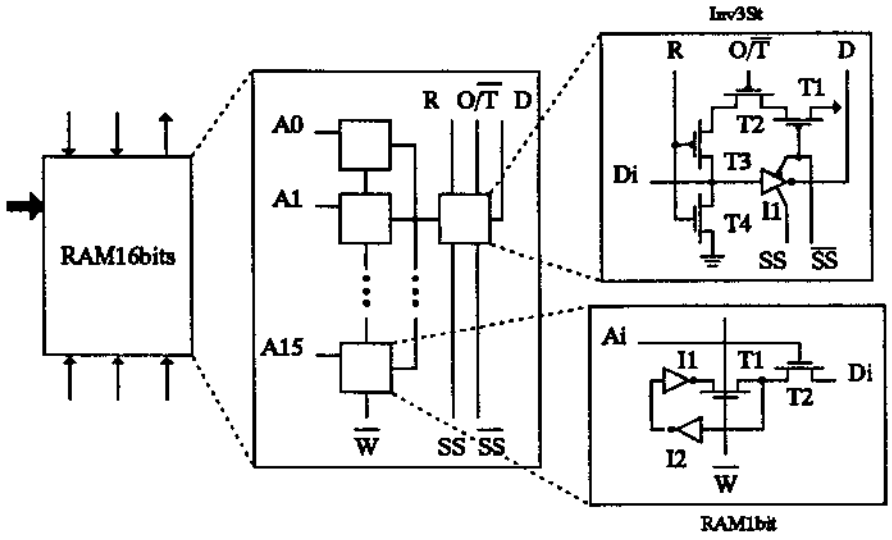


Figura 5.7: Diagrama elétrico da memória RAM usada no discriminador

sinais SS e $\overline{SS}$ estão ativos. Usou-se um inversor e não um transistor de passagem devido a alta capacitância do barramento externo. Os transistores T1 a T4 formam uma porta complexa que tem como função levar o barramento interno para '0' quando o 'reset' está ativo, ou para '1', quando o 'reset' não está ativo durante uma operação de treinamento. Esta porta complexa coloca sua saída em alta impedância quando nenhuma das condições ocorre, possibilitando a leitura da RAM através do inversor I1.

A célula decodificadora 'decoder' (figura 5.8) é composta de 16 portas *and* de 4 entradas, as quais são interligadas aos endereços formados no *input shift*, e aos seus complementos, convenientemente (como em um plano *and* de uma PLA) para produzirem as 16 linhas de endereços decodificados. Um circuito adicionado a saída das portas, força todas 16 linhas ao estado '1' quando o 'reset' está ativo, permitindo que todos os bits da RAM sejam levados ao estado '0'.

Outra porta complexa projetada foi a 'wgen' que gera o sinal de escrita (*write*) para as RAMs de uma coluna da matriz. A figura 5.9 apresenta seu diagrama elétrico juntamente com a tabela verdade que evidencia seu funcionamento durante as operações de *reset*, treinamento e operação. Os demais blocos lógicos são componentes já conhecidos e seus detalhes podem ser apreciados no apêndice B.

As simulações desta fase foram realizadas individualmente para cada célula e para as macro-células, validando o funcionamento das células dentro do seu próprio contexto. Após extração do *netlist* no Magic, as células eram simuladas no Spice. O principal problema detectado era com respeito ao dimensionamento dos transistores para manter os tempos de atraso nas células dentro de uma faixa aceitável. Os tempos de atraso obtidos nestas simulações foram levados de volta (*back-annotation*) para a descrição lógica do sistema onde pode-se efetuar uma simulação lógica mais realista.

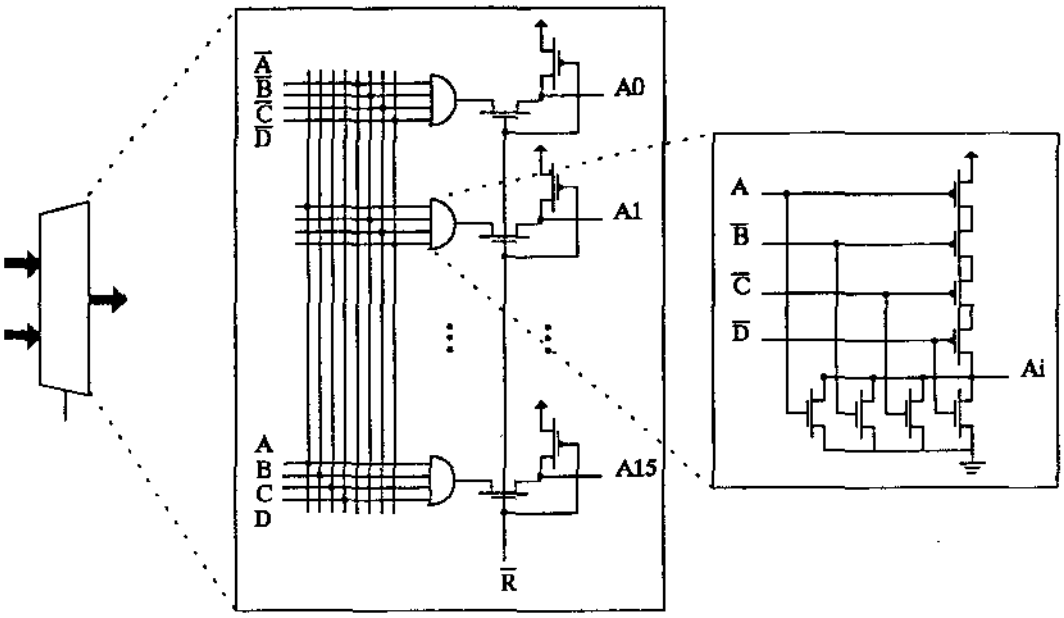
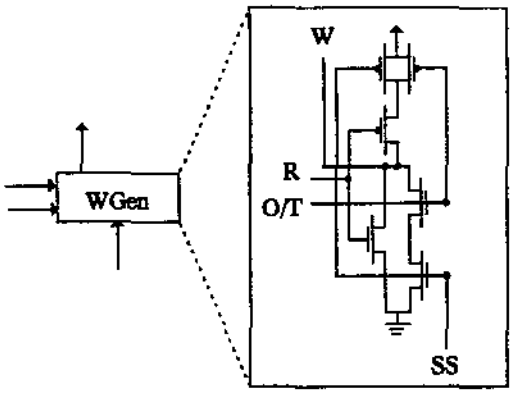


Figura 5.8: Diagrama elétrico do decodificador 'decoder'



R	O/T	SS	W	Obs:
0	0	0	1	—
0	0	1	0	Treinamento
0	1	X	1	Operacao
1	X	X	0	Reset

Figura 5.9: Diagrama elétrico do gerador do sinal de escrita

5.6 Implementação do Chip: Projeto Físico

O circuito integrado contém a matriz de RAMs os decodificadores, o acumulador e circuitos de controle. A maior parte da área do silício é ocupada pela matriz de RAMs.

A organização física (*floor planning*) do *chip*, exceto pelos *pads* e circuitos de controle, pode ser vista na figura 5.10. A Matriz é formada por linhas de RAMs às quais estão associadas as vias do barramento de entrada (uma linha por via). E cada coluna de uma linha é formada por uma RAM. Como as colunas da matriz são selecionadas uma por vez, otimizamos o projeto das RAMs usando apenas um decodificador de endereços para cada linha da matriz. Na linha associado ao *bit* sete, por exemplo, (fig. 5.10b) depois de preencher o registrador de entrada (*Input Shift*) com 4 *pixels* , estes *pixels* são decodificados e o resultado colocado em um barramento de 16 *bits* , que passam por todas as RAMs desta linha, e um desses *bits* seleciona uma célula (C_i) em cada RAM. Por outro lado apenas a RAM cuja coluna foi selecionada alimenta o *bit* do barramento de saída associado a ela em cada linha.

A figura 5.10b mostra também o posicionamento de outras células, tais como: dr (*driver*), W_gen (controle de escrita nas RAMs durante operação de *reset* e treinamento), SS_i (responsável pela seleção de colunas), Inv_3st (conecta cada RAM com o barramento de saída) e Result_Shift (guarda dado do barramento para posterior contagem). Devido às grandes capacitâncias nos barramentos foi necessário intercalar *drivers* no barramento de endereços das RAMs e usar inversores *3state* para conectar as RAMs ao barramento de saída.

A contagem de RAMs ativas (seção 5.4.1, antes idealizada por meio de um circuito somador, foi otimizada observando-se que a freqüência de entrada dos dados nos registradores de entrada (4 palavras a cada 18 *clock*) era suficiente para uma contagem serial. O circuito somador anterior exigia, além de uma área considerável, a criação de novas células. O registrador está distribuído, uma célula em cada linha do discriminador (*Result Shift* na fig. 5.10b), próxima ao barramento de saída das RAMs facilitando o roteamento dos sinais. Os *flip-flops* que compõem o contador também estão distribuídos ao longo do discriminador, cada um ao lado de uma célula do *Result Shift*.

A figura 5.11 mostra o *layout* de uma célula (*decoder00* dos decodificadores de endereços e 1 bit das RAMs. Antes destes *layout*, foram feitas algumas tentativas de otimização em área e houve pelo menos 3 ou 4 versões para as células, principalmente para a célula de 1 *bit* das RAMs, devido a grande quantidade de RAMs no *chip*. Interessante notar que as áreas efetivas das células são menores devido a superposição de suas bordas quando compondo uma macro-célula. Isso implica em ganho de área. Omite-se o *layout* das demais células, por conveniência (ver descrição detalhada do *chip* no apêndice B)

O circuito teve o seu projeto físico feito na metodologia *full-custom* devido à necessidade de otimizar a área de silício. Além disso, esta metodologia pôde ser aplicada aqui sem problemas em vista da boa regularidade do circuito e do pequeno número de células.

A figura 5.12 apresenta o *floor planning* da unidade de controle interna ao chip. À esquerda estão indicados os blocos funcionais de acordo com o circuito lógico já apresentado. O *layout* apresenta uma relação de aspecto semelhante a um retângulo de base bem estreita e altura bem maior. Isto permite um maior aproveitamento de área, já que a matriz também apresenta a mesma relação de aspecto. Seu *layout* foi desenvolvido através de uma variação do *Standard Cells* pela facilidade de implementação deste método e por essa unidade não consumir muito em termos de área. Nesta variação as células foram montadas verticalmente o que ofereceu uma maior facilidade de interconectar os sinais em relação a montagem tradicional do *Standard Cells*.

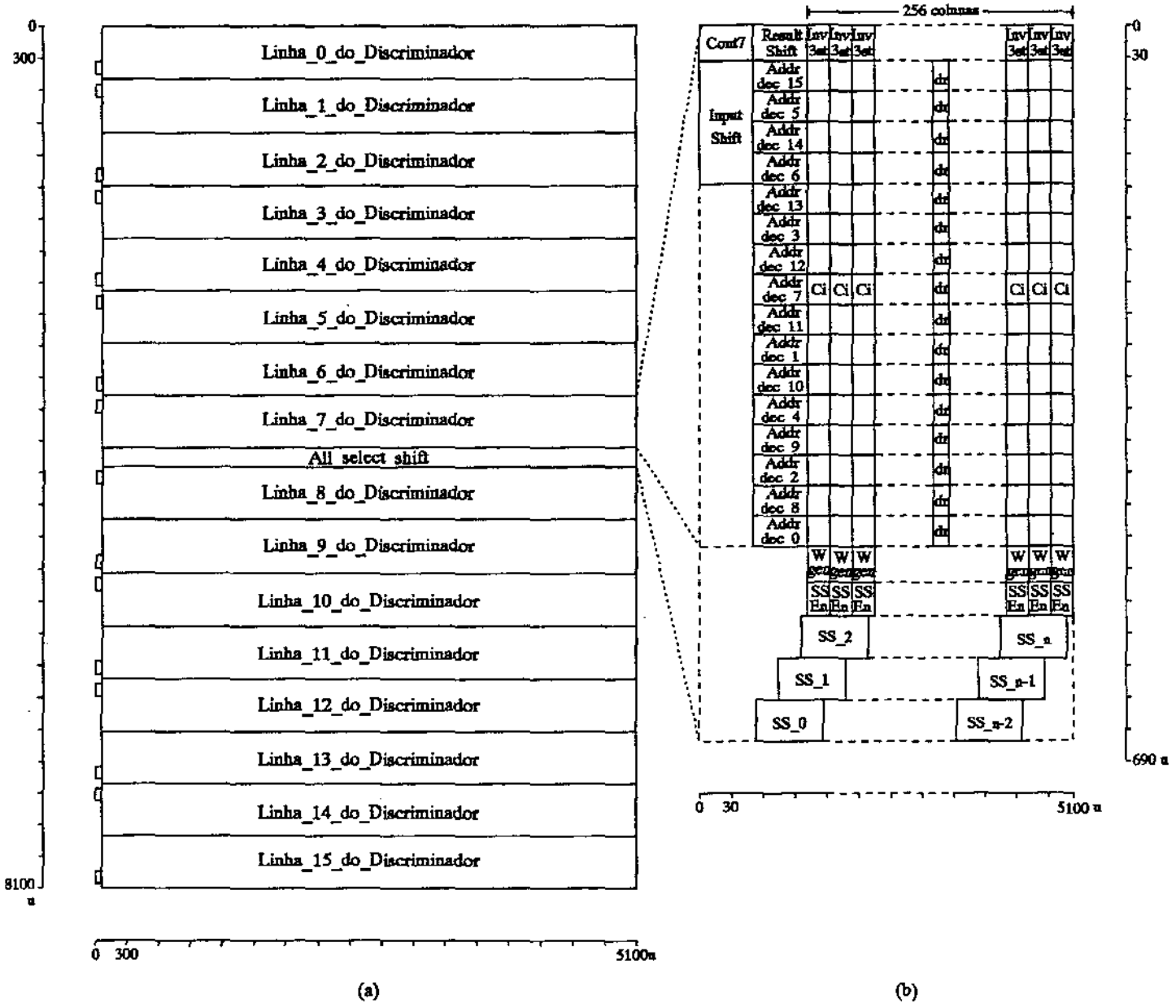


Figura 5.10: (a) *Floor Planning* da matriz de RAMs; (b) detalhe de uma linha da matriz.

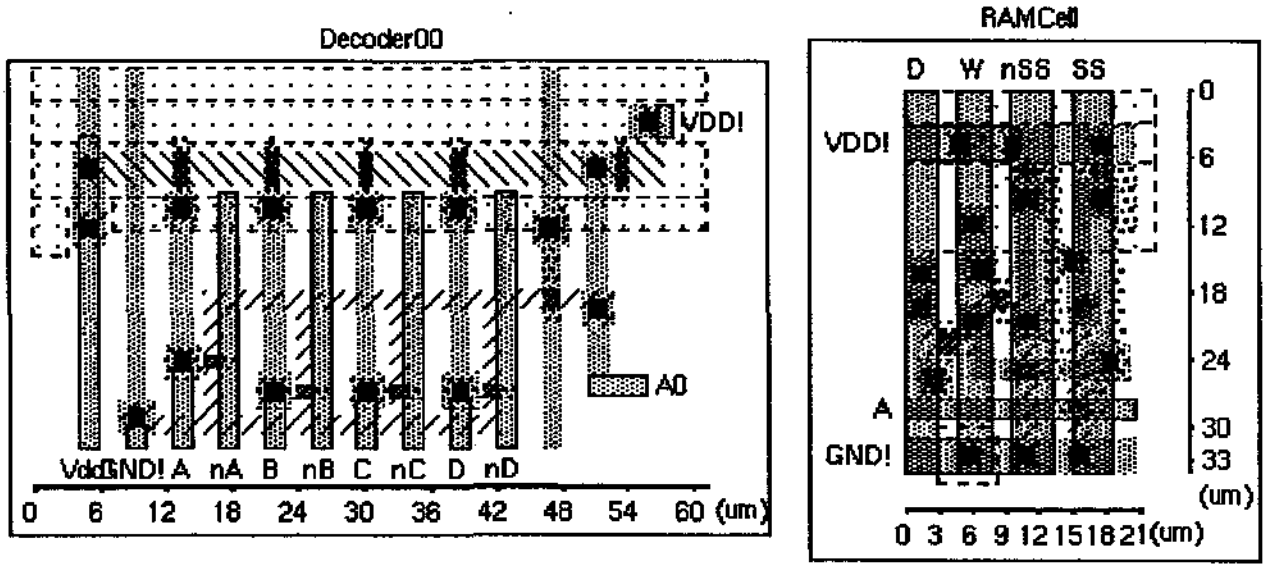


Figura 5.11: *Layout* de uma célula dos decodificadores de endereços e uma célula de 1 *bit* das RAMs.

O *layout* final é apresentado na figura 5.13. Os *pads* foram obtidos no CTI (Fundação Centro Tecnológico para Informática). A matriz, agora dividida em duas partes e de tamanho reduzido, tem uma melhor distribuição amenizando os efeitos de carga do barramento de seleção das RAMs. A unidade de controle está localizada no centro do *chip*. A matriz foi reduzida para adequar as dimensões do chip à realidade do projeto multiusuário (PMU) sob coordenação do CTI. A fabricação do protótipo pode ser realizada através do PMU CMOS11 com previsão para maio de 1994.

5.7 Sumário e Conclusões

Como foi visto, projeto de um circuito integrado pode ser descrito em vários níveis com diferentes graus de abstrações. Tais níveis estão contidos em três domínios: comportamental, estrutural e físico. A estratificação do projeto ao longo desses domínios possibilita ao projetistas, por meio de ferramentas de apoio, prever os problemas potenciais em níveis superiores, reduzindo o tempo e conseqüente custo do projeto.

Um exemplo típico desses problemas surgiu durante a validação lógica de uma máquina de estado com decodificador de estados (citado na seção 5.4) onde foi detectada a geração de pulsos espúrios (*spikes*). Se tal problema só fosse detectado durante a validação física, todo o tempo gasto com a descrição das máscaras relativas a esta máquina de estado teria sido perdido já que a mesma teria que ser reprojetada.

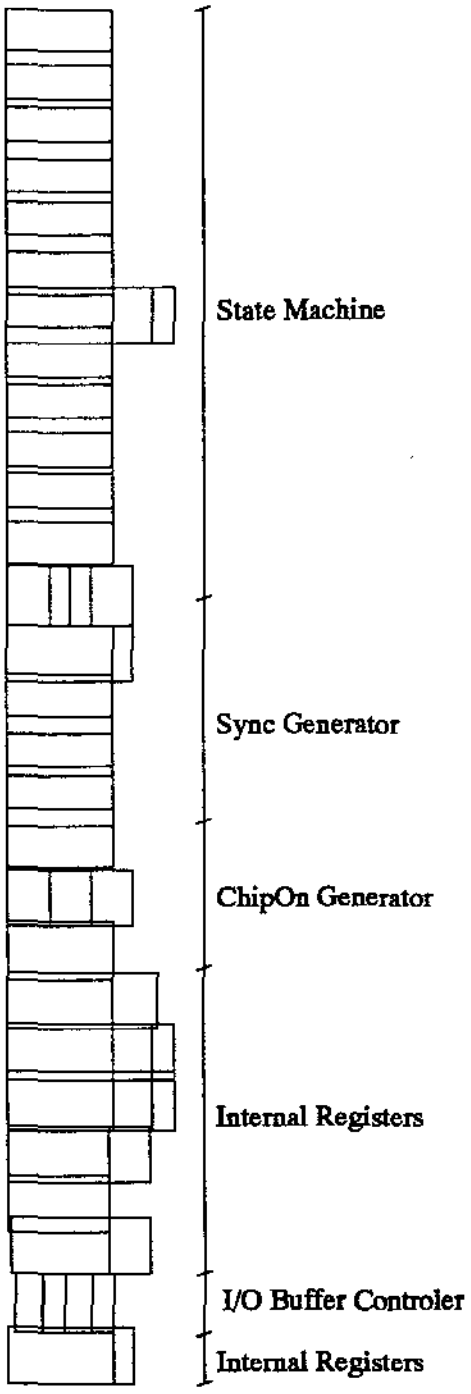


Figura 5.12: *Floor Planning* da unidade de controle.

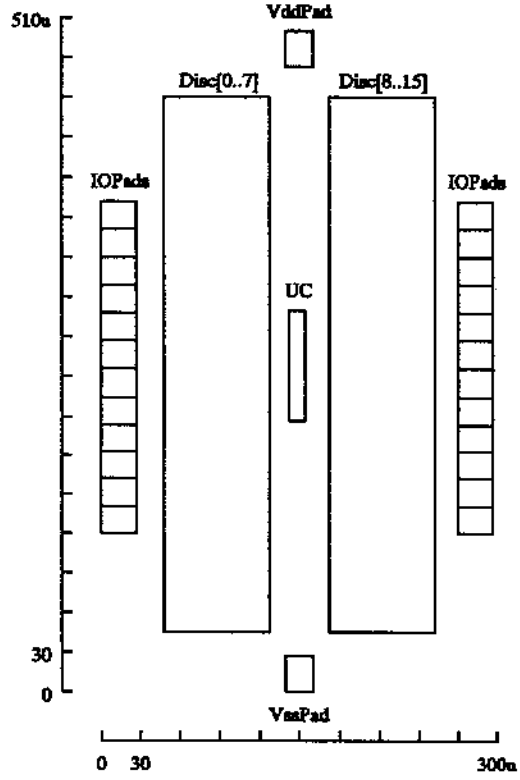


Figura 5.13: *Layout* final do *chip*

Capítulo 6

Análise das Simulações

6.1 Introdução

As etapas de validação, intercaladas entre as diversas fases de um projeto VLSI, fornecem dados que permitem avaliar quanto uma descrição do CI, em uma dada fase, está coerente com o funcionamento requerido. A principal ferramenta usada para coleta desses dados são os simuladores, dada a impossibilidade da montagem de um protótipo de um CI em um tempo relativamente curto e a um custo razoável. Mesmo em projetos de cartões, onde a montagem de um protótipo para teste é mais factível, o uso de simuladores diminuem consideravelmente o tempo e o custo de tais projetos. Os resultados de simulações permitem ao projetista ajustar melhor sua descrição para as reais condições de contorno do projeto que muitas vezes podem se perder devido a fatores tais como complexidade do sistema, falha humana, metodologia empregada etc. Decisões entre alternativas de projeto referentes ao dimensionamento e desempenho geral da arquitetura, podem ser tomadas com base nas simulações de alto nível (comportamental e lógica), enquanto que simulações estruturais podem auxiliarem o projetista na escolha dos circuitos mais apropriados para uma determinada função.

Este capítulo apresenta os resultados de algumas das simulações que foram feitas durante a validação das diversas fases do projeto do sistema de reconhecimento de imagens proposto. A seção 6.2 apresenta o programa de simulação desenvolvido em Pascal para a validação da técnica usada no *chip* e os resultados obtidos com imagens de algumas cédulas monetárias. Na validação do sistema, seção 6.3, são apresentados os ciclos de barramento obtidos nas simulações do modelo comportamental do sistema. Os resultados das simulações lógicas relativas ao ASIC são apresentados na seção 6.4 e um exemplo de validação elétrica é dada na seção 6.5, para a célula mais usada do ASIC, ou seja, a célula de memória RAM1Bit.

6.2 Validação da Técnica N-Tuple

Um problema das simulações de arquiteturas descrita em VHDL, mesmo no nível comportamental, é o elevado tempo de simulação quando se trabalha com grande quantidade de informações (imagens muito grandes, com por exemplo 256 x 256 *pixels*). Esse elevado tempo de simulação provém do fato do simulador e linguagem serem fortemente orientados ao *hardware*. A maioria do tempo de simulação é consumido no cálculo da temporização dos sinais previamente descritos, inviabilizando

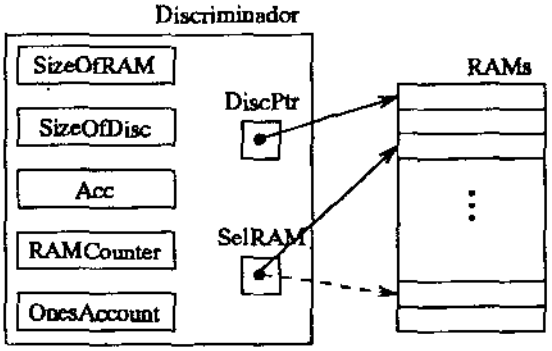


Figura 6.1: Estrutura de dados usado no programa simulador para validação da técnica *N-tuple*. Registro do discriminador e RAMs alocadas dinamicamente.

a validação do método em conjunto com a validação do sistema. Isso não implica em uma deficiência da ferramenta mas sim numa questão de objetivos (de simular *hardware*). Outro inconveniente de validar o método juntamente com o sistema é a falta de rotinas de tratamento de arquivos mais flexíveis e usáveis nas bibliotecas padrões que integram o ambiente de descrição/simulação. Por tais problemas, optou-se pela validação do método em separado, através de um programa simulador desenvolvido em uma linguagem de programação de fácil domínio que pudesse simular o método usando como dados imagens maiores em um tempo menor.

6.2.1 O Programa de Simulação

Embora a técnica usada aqui já tenha sido empregada com sucesso em dispositivos como o Wisard, fez-se necessário uma validação da mesma, onde foram levantados resultados em função de alguns parâmetros que possibilitaram o dimensionamento de estruturas tais como a capacidade das RAMs. O programa de simulação teria que ser capaz de aceitar o redimensionamento das RAMs e do barramento de endereços em tempo de execução e de desempenhar as principais funções do sistema (*reset* do discriminador, treinamento e reconhecimento). As estruturas de dados utilizadas no programa e seus principais procedimentos são descritos a seguir. Foram selecionados alguns resultados de simulação, os quais são apresentados e analisados no fim desta seção.

O programa foi desenvolvido no ambiente Turbo Pascal e encontra-se listado no apêndice A. Sua principal estrutura de dados, o discriminador, é um registro com o formato indicada na figura 6.1. O primeiro campo, 'DiscPtr', é um ponteiro que aponta para o inicio da região de memória reservada para as RAMs. Essa região de memória é alocada dinamicamente permitindo ao usuário dimensionar o discriminador em tempo de execução. O ponteiro 'SelRAM' seleciona seqüencialmente as RAMs nas quais efetuam-se as operações de treinamento e operação. Os campos 'RAMCounter' (contador de RAMs que já foram selecionadas) e OnesAccount (contador dos *bits* do banco de RAMs que estão em nível lógico '1') são usados no cálculo da densidade de '1s' no banco de RAMs. O campo 'Acc' é usado no modo 'operação' como contador de RAMs ativas. Os campos 'SizeOfRAM' e 'SizeOfDisc' indicam o número de *bits* da RAM (seu tamanho) e o número de RAMs do discriminador respectivamente.

A formação dos endereços das RAMs (ou das *N-tuples*) é apresentada na figura 6.2.

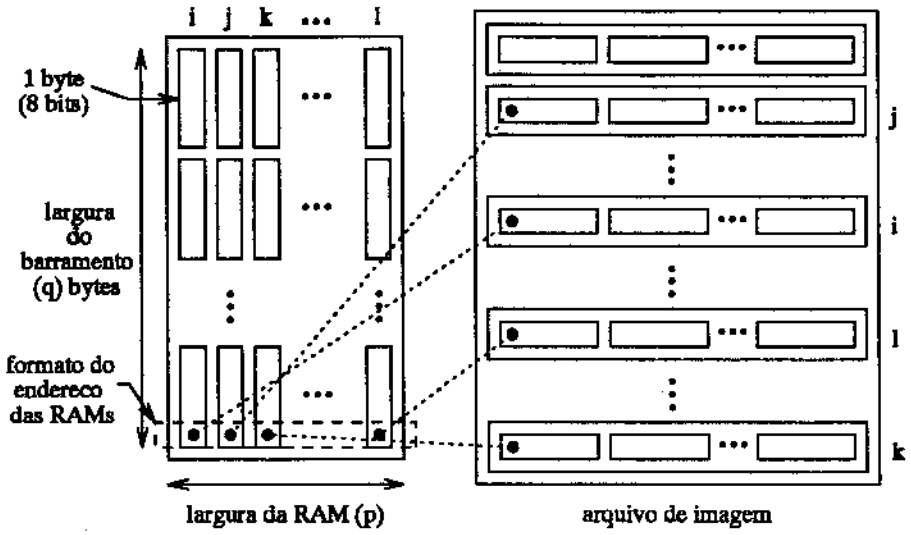


Figura 6.2: Formação dos endereços das RAMs.

Um *buffer* de dimensões $p \times q$ *bytes* é utilizado para este fim. Cada coluna do buffer recebe uma palavra de q *bytes* obtida de uma posição aleatória do arquivo de imagem (palavras 'i', 'j', 'k', ... 'l' na figura) e os endereços das RAMs são formados pelos *bits* contidos numa mesma linha deste *buffer*. Em outras palavras, as *n-tuples* $\{p_{0,i}, p_{0,j}, p_{0,k}, \cdots, p_{0,l}\}$, $\{p_{1,i}, p_{1,j}, p_{1,k}, \cdots, p_{1,l}\}$ ..., $\{p_{8\times q-1,i}, p_{8\times q-1,j}, p_{8\times q-1,k}, \cdots, p_{8\times q-1,l}\}$ são usadas para endereçar as 'q' primeiras RAMs. Esta operação se repete até que todos os *pixels* do arquivo de imagem tenham sido consumidos. No caso em que a imagem não seja um múltiplo de $p \times q$, os *bytes* restantes do *buffer* são preenchidos com '0'. Este esquema reflete exatamente o método de entrada dos *pixels* no *chip*. Os parâmetros p e q são configuráveis, o que possibilita validar não só a técnica *N-tuple* como também o método de embaralhamento dos *pixels*.

O programa aceita entrada de imagens no formato *x-bitmap*, o qual é um formato de arquivo texto e por isso não pode ser acessado aleatoriamente, devendo ser convertido para um formato interno batizado de 'str' (*stream*). Esta conversão é transparente ao usuário. O programa procura pela extensão 'str', se não a encontra, procura pela extensão 'xbm' e procede a conversão do arquivo para o formato 'str' antes de iniciar o processamento. O programa também adota, na fase de treinamento, um sistema de nomeação de arquivos reservando 6 caracteres para o nome do arquivo. Os outros 2 caracteres restantes são usados como um indexador que é automaticamente incrementado durante o treinamento de um conjunto de imagens, evitando que o usuário tenha que entrar com o nome de cada arquivo.

As principais rotinas do programa são:

- DiscReset: formada por um laço de $SizeOfDisc \times SizeOfRAM$ interações, que levam as posições de memória de todas as RAMs para '0'. Essa rotina também atribui o valor '0' ao campo 'OnesAccount'.

- LFSR: esta rotina simula o comportamento de um LFSR, retornando através de uma variável o próximo estado do LFSR toda vez que é executada. Ela possui um conjunto de polinômios

característicos [Côr91] que permite a geração de endereços aleatórios de até 17 *bits*.

- Make: é uma função que executa os processamentos de treinamento ou operação buscando os pixels do arquivo de imagem de entrada na ordem indicada pelo estado do LFSR e agrupando-os no *buffer* de modo a formar os endereços das RAMs.

- ReadWriteRAM: essa função é chamada por 'Make' e é responsável pelo treinamento (escrita) e operação (leitura) das RAMs. Ela também incrementa 'Onesaccount' sempre que um *bit* de qualquer uma das RAMs muda do estado '0' para '1' durante o treinamento. Já no modo de operação, ela incrementa 'Acc' sempre que uma RAM está ativa. Ela retorna *False* se houver uma tentativa de escrever ou ler em alguma RAM fora do limite do discriminador ($RAMCounter > SizeOfDisc \times SizeOfRAM$).

6.2.2 Resultados obtidos

As simulações foram efetuadas para os casos de (1) treinamento com apenas uma imagem e (2) treinamento com várias imagens do mesmo objeto. Em ambos os casos, as imagens foram obtidas com o auxílio de um *scanner*. Uma nota de um dólar foi usada durante as simulações do primeiro caso. A figura 6.3 apresenta três notas de um dólar. A primeira nota foi usada no treinamento do discriminador e as outras duas, as quais estão propositalmente rabiscadas, foram usadas durante a fase de operação.

A tabela 6.1 apresenta os resultados obtidos para o primeiro caso. Barramentos com 8, 16 e 32 *bits* foram considerados. Também foram considerados 4 discriminadores com RAMs, cujo tamanho varia entre 8 a 64 *bits*. Cada campo da tabela exibe dois índices. O primeiro índice corresponde à imagem do dollar01 e o segundo à imagem do dollar02. Nota-se que em todos os campos o primeiro índice varia entre 0,9 e 0,8 indicando que a imagem foi reconhecida como sendo à de uma nota de um dólar. Já o segundo índice ($< 0,6$) denota a pouca semelhança da imagem com relação a nota. Este resultado expressa exatamente as condições das duas notas. A imagem dollar01 foi rabiscada com traços finos o que possibilitou o seu reconhecimento. Entretanto a imagem dollar02 foi muito rabiscada com perda de boa parte de sua área. Isso justifica os baixos índices de semelhança obtidos para ela.

Com relação à variação do tamanho das RAMs, a tabela evidencia que todos os índices guardam uma relação inversa. Isto é, a medida que as RAMs aumentam de tamanho os índices diminuem. Este comportamento confirma que quanto maior forem as RAMs, menor o poder de generalização do discriminador, ou seja, menor a capacidade de reconhecer uma imagem fortemente corrompida, através da generalização de partes dessa imagem. Isto porque a probabilidade do *bit* em '1' (escrito durante a fase de treinamento) ser endereçado è menor em uma RAM com maior número de posições. Quanto à largura do barramento, não é notada uma variação muito sensível dos índices, demonstrando que os resultados obtidos com o método adotado para o embaralhamento dos *pixels* (seção 4.2.5) não é afetado pela largura do barramento, que deve ser definida em função da velocidade de entrada da imagem, largura do barramento do computador usado como *host* e o número de pinos disponíveis para o ASIC.

Para o segundo caso, foram obtidas imagens de várias cédulas de 5 e 10 mil cruzeiros antigos com as esfinges de Carlos Gomes e Vital Brazil, armazenadas em arquivos nomeados como CG_{nn} e VB_{nn}. A figura 6.4 apresenta as cédulas utilizadas na simulação do segundo caso. Foram obtidas um total de 22 imagens das cédulas de 5 mil e 2 imagens das cédulas de 10 mil. As 15 cédulas

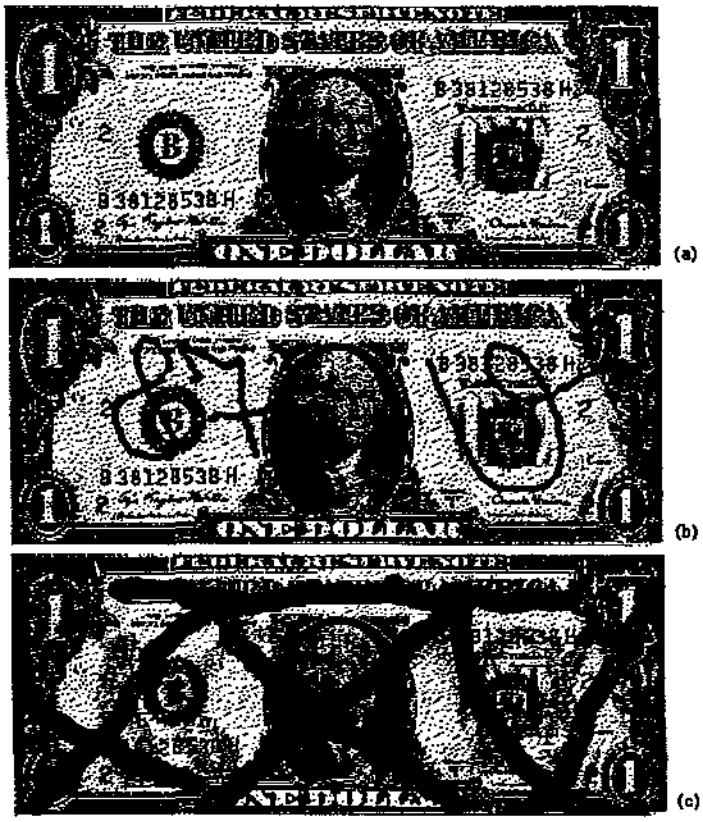


Figura 6.3: Imagens utilizadas durante a simulação da técnica *N-tuple*: (a) dollar00, (b) dollar01 e (c) dollar02.

capacidade das RAMs	largura do barramento			arquivo imagem
	8 *bits*	16 *bits*	32 *bits*	
8 *bits*	0,9035	0,9031	0,9024	dollar01
	0,5116	0,5101	0,5100	dollar02
16 *bits*	0,8753	0,8555	0,8757	dollar01
	0,4152	0,4091	0,4141	dollar02
32 *bits*	0,8486	0,8438	0,8446	dollar01
	0,3347	0,3297	0,3380	dollar02
64 *bits*	0,8186	0,8189	0,8175	dollar01
	0,2685	0,2671	0,2743	dollar02

Tabela 6.1: Índices de reconhecimento (ou semelhança) obtidos nas simulações para validação da técnica *N-tuple* com treinamento de um simples padrão (dollar00) e reconhecimento dos padrões dollar01 e dollar02 apresentados na figura 6.3.

mais semelhantes e menos manchadas (fig. 6.4a) foram selecionadas para o treinamento das RAMs. Duas cédulas de 5 mil, a primeira bem semelhante ao conjunto de treinamento e a segunda, uma cédula mal conservada com manchas, foram usadas no reconhecimento (fig. 6.4b e 6.4c). Como contra exemplo foram utilizadas as duas notas de 10 mil, uma razoavelmente conservada e outra bastante manchada (fig. 6.4d e 6.4e).

O resultado das simulações estão resumidos na tabela 6.2. Neste caso também foram considerados três barramentos e discriminadores com RAMs de 8 a 32 *bits*. As primeiras linhas de cada discriminador, apresentam a densidade (D) de *bits* das RAMs em nível lógico 1. A segunda e terceira linha de cada discriminador, apresentam os índices de reconhecimento para as notas CG15 e CG16. A quarta e quinta linha, apresentam os índices para as notas VB17 e VB18.

Um dado novo neste caso é a densidade de *bits* em nível 1 nas RAMs. No primeiro caso, como apenas uma imagem foi treinada, a densidade manteve-se constante e igual a 1, mas neste caso as pequenas diferenças entre as imagens do conjunto de treinamento forçaram mais que um posição de memória das RAMs para o nível 1. É importante que o conjunto de treinamento possua imagens semelhantes, mantendo a densidade baixa para que o discriminador não perca sua capacidade de reconhecimento. Por exemplo, na primeira tentativa de simulação para este caso, as imagens foram obtidas com uma resolução baixa e grande parte dos *pixels* de uma imagem não coincidiam com os das outras, provocando uma densidade elevada (8*bits*/RAM). Por conseqüência, durante a fase de operação todas as cédulas, inclusive as de 10 mil, foram reconhecidas como uma cédula de 5 mil. A tabela 6.2 indica que a densidade independe da largura do barramento e que ocorre um leve acréscimo da mesma quando é aumentada a capacidade das RAMs. Na verdade este aumento na densidade é verificado devido à redução do número de RAMs necessárias ao processamento, quando a capacidade delas aumenta, e pelo fato que a densidade está expressa por unidade de RAM. Se fosse adotada uma densidade de *bits* em nível 1 por total de *bits* do discriminador, seria verificado um decréscimo da densidade com relação ao aumento de capacidade das RAMs.

Os índices de reconhecimento apresentados na tabela 6.2, da mesma forma que no primeiro caso, independe da largura do barramento usado no embaralhamento dos *pixels*. Para um mesmo

Figura 6.4: Imagens utilizadas durante a simulação da técnica *N-tuple*: (a) conjunto de imagens de treinamento com 15 cédulas, (b) e (c) cédulas submetidas ao reconhecimento, (d) e (e) cédulas usadas como contra exemplo.

capacidade das RAMs	largura do barramento			arquivo imagem
	8 *bits*	16 *bits*	32 *bits*	
8 *bits*	2,5892	2,5897	2,5952	(densidade)
	0,9836	0,9835	0,9828	CG15
	0,9414	0,9409	0,9410	CG16
	0,7490	0,7517	0,7510	VB17
	0,6220	0,6219	0,6242	VB18
16 *bits*	3,2345	3,2325	3,2456	(densidade)
	0,9709	0,9706	0,9706	CG15
	0,9093	0,9084	0,9106	CG16
	0,6544	0,6610	0,6632	VB17
	0,5049	0,5088	0,5125	VB18
32 *bits*	3,9176	3,9126	3,9268	(densidade)
	0,9553	0,9540	0,9536	CG15
	0,8696	0,8707	0,8737	CG16
	0,5755	0,5797	0,5760	VB17
	0,4091	0,4112	0,4174	VB18

Tabela 6.2: Densidade de *bits* em nível '1' nas RAMs e índices de reconhecimento (ou semelhança) obtidos nas simulações para validação da técnica *N-tuple* com treinamento de um conjunto de 15 cédulas (fig. 6.4a) e reconhecimento dos padrões apresentados na figura 6.4b a 6.4e

discriminador e mesma imagem, os índices de reconhecimento praticamente não variam. Os índices para a imagem de 5 mil bastante semelhante as imagens de treinamento (fig 6.4b) mantiveram-se acima de 0,95 em todos os casos e até mesmo a cédula de 5 mil manchada foi reconhecida com um índice maior que 0,86. No caso do discriminador com RAMs de 8 *bits* é necessário um limiar de reconhecimento em torno de 0,80 a 0,85 para que as notas de 10 mil não sejam reconhecidas como uma de 5 mil. Esses elevados índices de reconhecimento para as cédulas de 10 mil, especialmente para a cédula mais conservada (VB17), é conseqüência do alto poder de generalização dos discriminadores com RAMs de pequena capacidade como já foi citado e também pela semelhança (em linhas gerais) entre as duas cédulas de 5 e 10 mil. Já para os discriminadores com RAMs de 16 e 32 *bits*, um limiar de 0,70 é suficiente.

O treinamento com mais de uma imagem aumenta o poder dos discriminadores em generalizar e discriminar, embora uma seleção cuidadosa das imagens seja necessária para evitar que as RAMs saturem. Nas aplicações reais nem sempre as imagens obtidas estão perfeitamente alinhadas. O treinamento com um conjunto razoável de imagens previamente escolhidas (tipicamente 100 a 200 imagens) pode fornecer informações suficientes ao discriminador a respeito destas imperfeições no sistema de aquisição de imagens.

A tabela 6.3 apresenta os índices para as mesmas imagens da tabela anterior com o treinamento sob apenas uma imagem a fim de comparar e comprovar a eficácia do treinamento com um conjunto de imagens. Por exemplo, a diferença entre os índices de reconhecimento (aqui denominada de distâncias) para as imagens CG16 e VB17, sob o barramento de 16 *bits*, da tabela 6.2 (treinamento

capacidade das RAMs	largura do barramento			arquivo imagem
	8 *bits*	16 *bits*	32 *bits*	
8 *bits*	0,6768	0,6747	0,6757	CG15
	0,5172	0,5181	0,5197	CG16
	0,4064	0,4074	0,4102	VB17
	0,3065	0,3067	0,3068	VB18
16 *bits*	0,5947	0,5933	0,5916	CG15
	0,4183	0,4147	0,4181	CG16
	0,3072	0,3051	0,3029	VB17
	0,2077	0,2092	0,2059	VB18
32 *bits*	0,5204	0,5176	0,5176	CG15
	0,3328	0,3336	0,3314	CG16
	0,2279	0,2281	0,2242	VB17
	0,1401	0,1452	0,1377	VB18

Tabela 6.3: índices de reconhecimento (ou semelhança) obtidos nas simulações para validação da técnica *N-tuple* com treinamento de apenas uma cédula de 5 mil cruzeiros antigos e reconhecimento dos padrões apresentados na figura 6.4b a 6.4e

com 15 imagens) é de 0,1892, 0,2474 e 0,2910 para os diversos tamanho de RAMs. Já para a tabela 6.3 (treinamento com apenas uma imagem), essas mesmas distâncias são 0,1107, 0,1096 e 0,1055. A comparação das distâncias entre essas duas imagens evidencia um aumento no poder de discriminação obtido nas simulações com treinamento de um conjunto de imagens. As distâncias para os três tipos de RAMs, neste caso, são 1,71, 2,25 e 2,76 respectivamente maiores que no caso de treinamento com uma única imagem. Analogamente, as distâncias para as duas notas de 5 mil (CG15 e CG16) para o caso de treinamento com um conjunto de imagens são 3,68, 2,87 e 2,20 vezes menor que no caso de treinamento simples, indicando uma melhora na generalização do reconhecimento das notas de 5 mil.

Os resultados desta fase, além de validar o princípio de funcionamento do sistema, possibilitou o dimensionamento das RAMs dos discriminadores, de uma forma mais embasada. Como foi visto, o discriminador formado por RAMs de 32 *bits* apresentou os melhores resultados com relação à discriminação das imagens e os piores, com relação à generalização. Uma situação inversa foi verificada para o discriminador formado por RAMs de 8 *bits*. O discriminador formado por RAMs de 16 *bits* apresentou resultados intermediários para ambas as características e sua implementação possui um número razoável de células. Por estas razões, a capacidade das RAMs foi dimensionada em 16 *bits*.

6.3 Validação do Sistema

O sistema foi validado através de simulação da descrição comportamental. O simulador usado foi o *QuickSim II*, que é o simulador lógico do ambiente de apoio a projetos eletrônicos da *Mentor Graphics*. Com ele foi possível obter a representação gráfica dos sinais do sistema para as diversas

operações, tais como leitura e escrita de I/O, leitura de memória, etc. A descrição do sistema foi baseada nos ciclos de barramento do IBM PC-AT [IBM84][Egg90]. Foram necessárias algumas interações entre as fases de descrição e simulação até que as formas de ondas dos principais ciclos do PC-AT fossem obtidas como especificadas. Outros ciclos, como escrita de memória, não foram modelados por não ocorrem diretamente ligados com o funcionamento do sistema.

6.3.1 Principais ciclos de barramento

Os sinais sob o barramento do PC e PLD de controle foram modelados segundo condições previamente especificadas (ver seção 4.3.2 e/ou apêndice A), de modo a produzirem periodicamente os ciclos de barramento requeridos ao funcionamento do sistema em uma ordem apropriada, como indicado na tabela 6.4.

Tarefas	Ciclos
CPU programa PLD e matriz de *chips*	Ciclo de escrita de I/O)
PLD requisita o barramento	Ciclo de requisição de barramento
PLD auxilia o acesso da matriz de *chips* aos *pixels* da imagem	Ciclo de leitura de memória
CPU lê resultado de cada *chip*	Ciclo de leitura de I/O

Tabela 6.4: Os ciclos de barramento do sistema

A figura 6.5 apresenta os ciclos de leitura e escrita de I/O. À esquerda, o ciclo de escrita é iniciado quando a CPU coloca o endereço do dispositivo de I/O no barramento de endereço. O sinal PCBHE é mantido pelo PC e indica o tipo de transferência de dados (16 *bits*, 8 MSB e 8 LSB). Este sinal em conjunto com o sinal A0 pode ser usado para decodificar o tipo de transferência de dados de um ciclo, como mostra a tabela 6.5. No caso da figura 6.5 trata-se de uma escrita com os 8 *bits* menos significativos do barramento de dados (D7 à D0). Em seguida o sinal PCIOW é ativado. Essa situação é mantida até que o dispositivo de I/O ative o sinal PCIOChRdy indicando que está pronto para capturar os dados, bastando para tanto que a CPU desative o sinal PCIOW. A CPU põe o dado no barramento de dados e desativa o sinal de escrita PCIOW. Os outros sinais são desativados em seguida finalizando o ciclo de escrita. Este ciclo ocorre durante a programação dos registros internos da PLD e dos registros da unidade de controle de cada *chip* da matriz.

PCBHE	A0	Função
1	0	Transferência de 16 *bits*
1	1	Transferência dos 8 *bits* mais significativos (8 MSB)
0	0	Transferência dos 8 *bits* menos significativos (8 LSB)

Tabela 6.5: Tipos de transferência de dados

O ciclo de leitura de I/O é semelhante ao de escrita. As formas de ondas da direita da figura 6.5 apresenta um ciclo de leitura de 16 *bits* que ocorre durante a leitura do resultado dos discriminadores

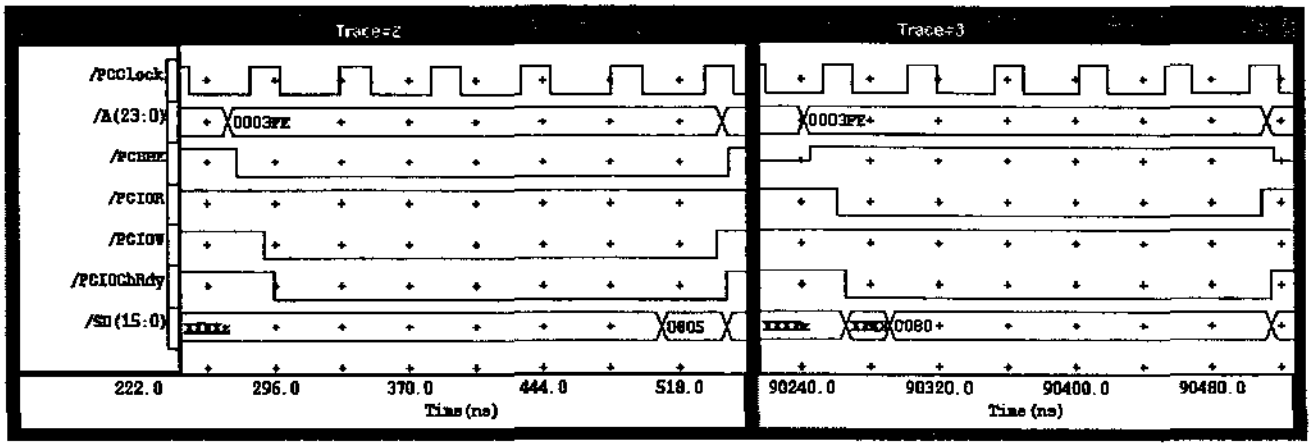


Figura 6.5: Ciclo de leitura e escrita em dispositivos de entrada e saída.

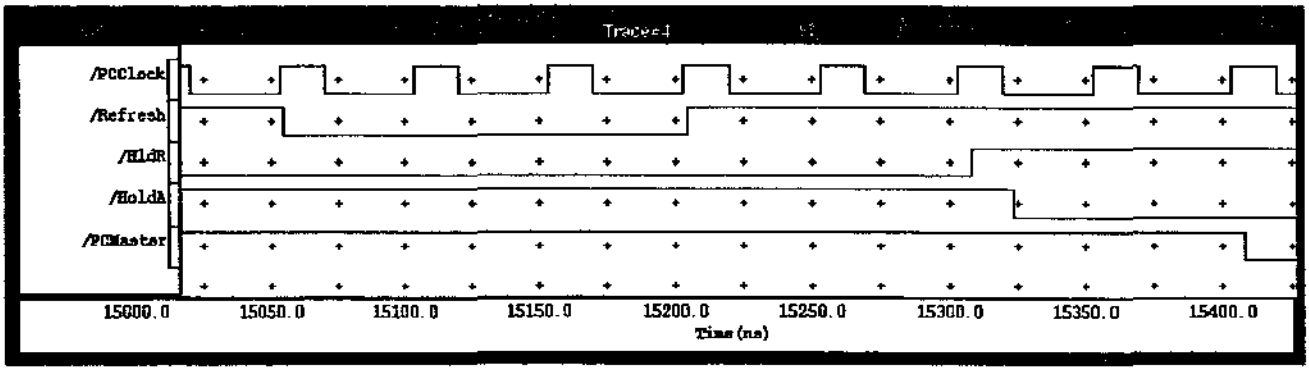


Figura 6.6: Ciclo de requisição de barramento

nos acumuladores internos de cada *chip* da matriz. O ciclo de requisição de barramento (figura 6.6) foi descrito por meio de uma máquina de estados (seção 4.2.6). Este ciclo é sincronizado com o pulso de *refresh* de modo a não interferir nas funções de *refresh* das memórias dinâmicas. O ciclo de leitura de memória é apresentado na figura 6.7 e ocorre durante a entrada dos pixel (da imagem armazenada na memória do PC) nos discriminadores. Neste caso o controle do barramento já está com a PLD de controle que gera os sinais A(23:0), PCBHE, PCMemR e NW através da máquina de estado já citada.

Os demais ciclos envolvendo os sinais relativos ao ASIC foram validados, ao nível comportamental, com o mesmo cuidados que os outros, sendo omitidos aqui evitando repetições. A validação lógica do *chip* apresenta as formas de ondas dos principais sinais da arquitetura do ASIC.

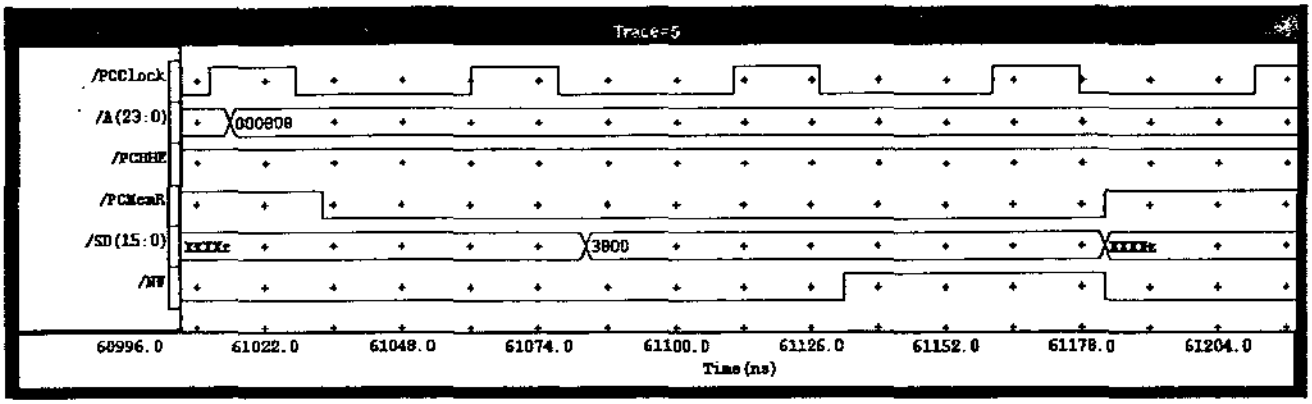


Figura 6.7: Ciclo de leitura de memória

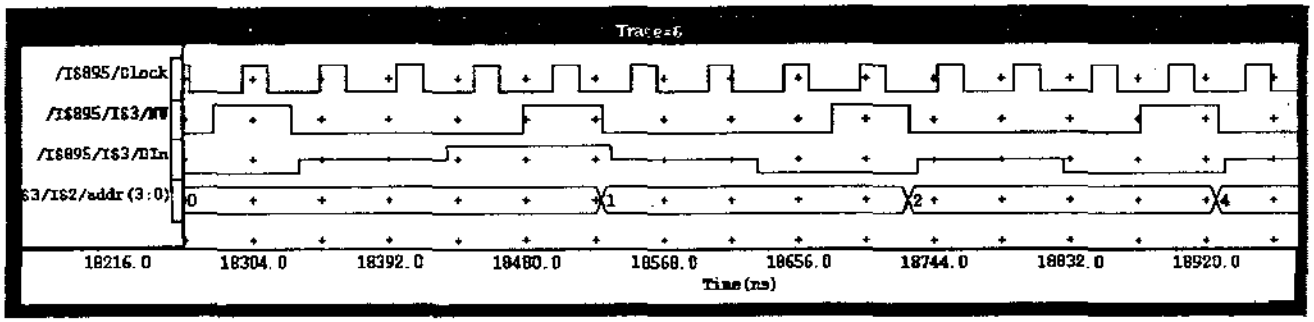


Figura 6.8: Ciclo de entrada de dados

6.4 Validação Lógica do Chip

A validação lógica do *chip* foi realizada através de simulações da descrição comportamental do sistema após a substituição da descrição comportamental do *chip* pela sua descrição lógica. Os resultados apresentados nesta seção foram obtidos da simulação final, onde todos os parâmetros de atraso de propagação foram atualizados com valores obtidos diretamente de simulações elétricas das descrições físicas das células. São apresentados os sinais dos principais ciclos internos à arquitetura e unidade de controle do *chip*.

As primeiras formas de ondas obtidas com a simulação da descrição lógica do *chip* dizem respeito à entrada dos dados e são apresentadas na figura 6.8. O sinal NW (Neurônio *Write*), gerado pela PLD de controle, habilita a entrada dos dados e sincroniza as funções da unidade de controle interna do *chip*. O dado vindo da memória do PC, o qual tem um dos *bits* representado por DIn na figura, deve estar disponível e estável durante o tempo em que NW estiver ativo. O nível intermediário entre '0' e '1' apresentado por DIn, no início do ciclo, indica que DIn está em alta impedância ('Z'). O *chip* captura o dado na borda de decida do sinal Clock quando NW está ativo.

A figura 6.9 apresenta os sinais dos principais pontos do discriminador quando o *chip* está

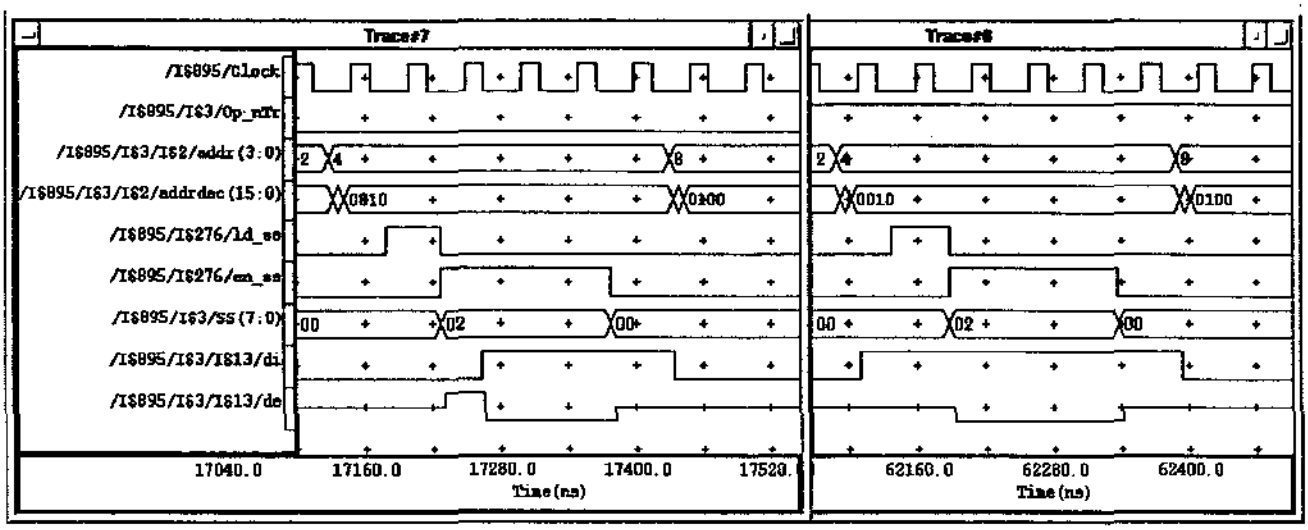


Figura 6.9: Ciclo de treinamento e operação

funcionando nos modos de treinamento (esquerda) e operação (direita). O projeto lógico do discriminador (figura 5.4) pode auxiliar no entendimento destas formas de ondas. O sinal addr(3:0) é gerado pelo *Input Shift* de acordo com o ciclo de entrada de dados discutido a pouco. O sinal addrdec(15:0) é gerado pelo *Decoder* que fornece o endereço decodificado para as RAMs de uma mesma linha do discriminador. O sinal ld_ss (SSLd na figura 5.4) habilita o funcionamento do SS (*Select Shift*) e em conjunto com o sinal en_ss geram o sinal SS(7:0) e seu complemento. O sinal 'di' da RAM selecionada (barramento de dados internos detalhado na figura 5.9) é levada a '1' forçando a célula de memória ativada por addrdec(15:0) ao nível '1'. O sinal 'de' reflete o estado do inversor de três estados que alimenta o barramento externo. O ciclo de operação é semelhante ao de treinamento, exceto pelo sinal 'di' que é alimentado pela célula de memória endereçada por addrdec(15:0) da RAM selecionada.

Um problema observado nesta validação foi os diferentes atrasos de propagação nas células do decodificador (figura 5.10) os quais mudam os estados dos sinais addrdec(15:0) não simultaneamente. Isto é representado nas formas de ondas da figura 6.9 pelos pequenos losangos intercalados em addrdec(15:0) durante uma mudança de estados. Em conseqüência, duas linhas podem ficar ativas e selecionarem células de memórias com conteúdos opostos ocasionando a perda dos dados armazenados. As simulações elétricas, cuidadosamente direcionadas nesse sentido mostraram que o intervalo de tempo requerido para estabilização dos sinais addrdec(15:0) é muito pequeno sendo insuficiente para que duas células endereçadas simultaneamente destruam seus dados.

Os ciclos de contagem das RAMs ativas ocorrem paralelamente aos ciclos de operação, computando sempre as RAMs ativas do ciclo de operação anterior. A figura 6.10 apresenta as formas de ondas relativas a este ciclo e a estrutura do acumulador pode ser vista na figura 5.7. O sinal DIn, vindo do barramento externo das RAMs, alimenta o registrador de deslocamento RS na borda de descida do Clock quando o sinal LdP está ativo. Logo em seguida este sinal é desativado ao tempo em que o sinal LdS é ativado dando inicio a serialização dos dados em AccEn. É interessante notar que há uma inversão nos níveis lógicos durante a serialização visando anular a inversão ocorrida

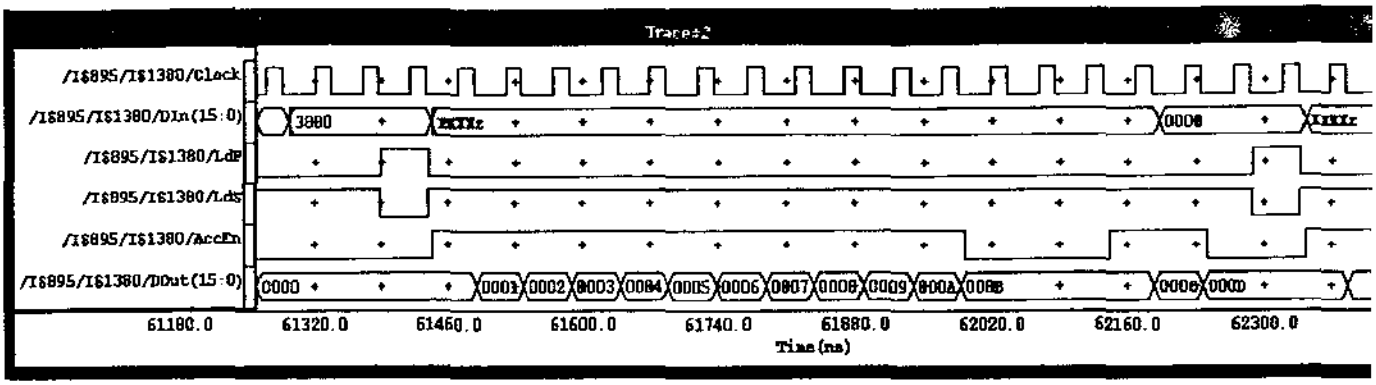


Figura 6.10: Contagem das RAMs ativas

no inversor de três estados que alimenta o barramento externo das RAMs. Assim no exemplo da figura 6.10 o dado 3800 (0011 1000 0000 0000 em binário) é convertido para C7FF (1100 0111 1111 1111 em binário) e apresentado sob forma de onda como FF7C em AccEn devido ao sentido da serialização. Os resultados parciais acumulados pelo contador pode ser observado em DOut(15:0). Para C700, por exemplo, o resultado é 000D que corresponde em decimal a 13, ou seja o número de *bits* em '1' em C700.

Uma preocupação nas simulações do contador foi com respeito ao tempo de propagação de cada célula que tinha de ser mantido abaixo de 1/15 do período de clock já que o tempo de propagação do sinal AccEn através de todas as 15 células do contador teria de ser menor que o período de clock, isto é, este sinal devia se propagar até a décima quinta célula antes da próxima borda de descida do clock.

Os resultados da simulação lógica referentes à unidade de controle foram divididos em cinco formas de ondas. Operações de escrita e leitura (figura 6.11), programação dos modos treinamento e operação, enfatizando os estados dos sinais de *reset* (figura 6.12), e funcionamento da máquina de estado, incluindo alguns sinais internos (figura 6.13). Os sinais descritos por estas formas de ondas podem ser localizados e acompanhados no diagrama lógico da unidade de controle, apresentado na figura 5.8.

As duas primeiras formas de ondas, apresentadas na figura 6.11, validam o funcionamento do circuito combinacional de controle de direção dos *buffers* bidirecionais incorporados nos *pads* dos pinos de dados durante as operações de escrita e leitura (programação dos registros internos da unidade de controle e leitura do contador de RAMs ativas, respectivamente). O sinal ctrl, que controla tais *buffers*, é mantido em nível '1' forçando os *pads* funcionarem como *pads* de entrada, exceto quando há um ciclo de leitura. Esta condição é percebida pelo circuito quando o *chip* é selecionado (ncsl e ncsc ativos em nível lógico baixo) e quando o sinal de leitura de I/O nior é ativado. Nestas condições ctrl muda para o nível '0' forçando os *pads* funcionarem como *pads* de saída.

O modo de funcionamento do *chip*, o *reset* das RAMs e o *reset* de outros elementos (*bits* 1, 2 e 0 da palavra de controle) são definidos quando os registradores internos da unidade de controle são programados. Na esquerda da figura 6.12, estes registradores são programados com 5 (101 em binário) na borda de subida de niow. Isso define o modo de treinamento com *reset* das RAMs e dos

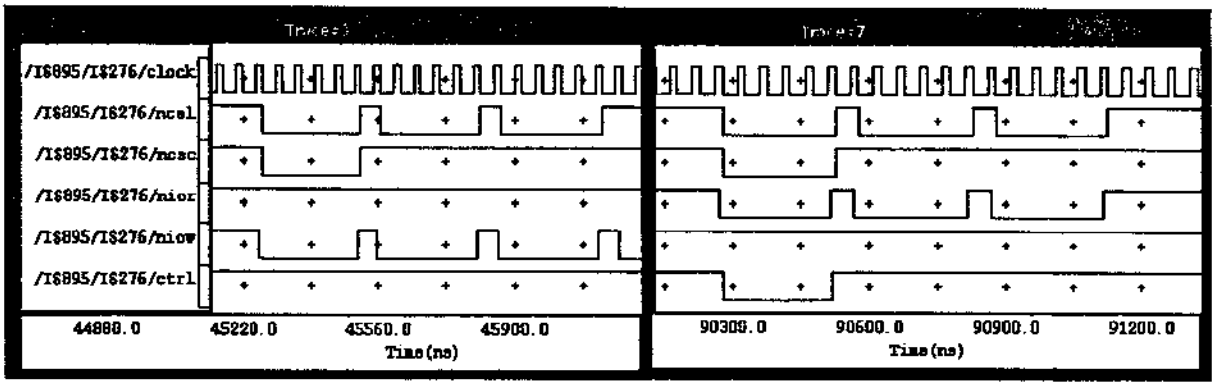


Figura 6.11: Sinais gerados na unidade de controle. Escrita e leitura de I/O

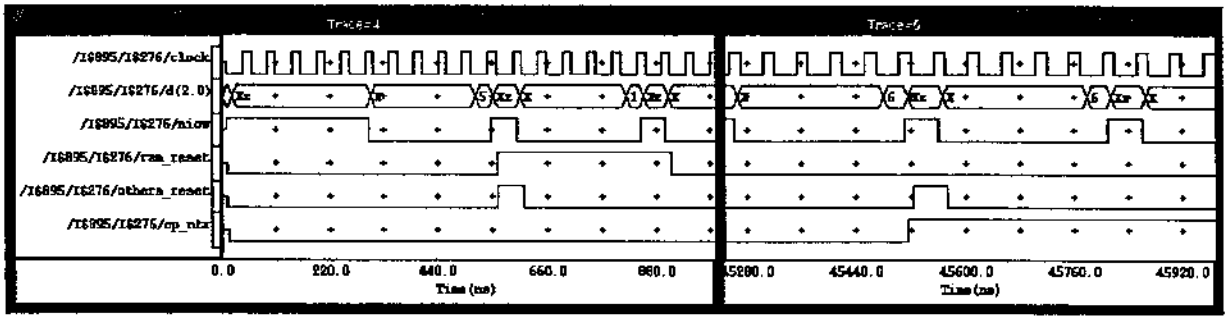


Figura 6.12: Sinais gerados na unidade de controle. *Reset* e modo de funcionamento

outros elementos. Dois pequenos contadores controlam a largura dos sinais de *reset*, desativando estes sinais ao fim do primeiro pulso de *clock* para o *others_reset* e do sétimo, para o *ram_reset*. Na direita da mesma figura, está a programação do modo de operação, onde apenas o *reset* dos outros elementos é ativado.

Como já citado, o controle do registrador de deslocamento de seleção e do acumulador foram gerados através de uma máquina de estado que é ativada quando o quarto *pixel* de cada linha do discriminador é carregado no registrador de deslocamento de entrada. A figura 6.13 apresenta os resultados de simulação para essa parte da unidade de controle. O sinal sync é ativado no quarto pulso de nw e desativado no primeiro (do grupo de quatro subseqüente), sempre síncrono com a borda de descida do *clock*. Este sinal ativa a máquina de estado que por sua vez gera os sinais de controle para o discriminador. O sinal chip_on é ativado na primeira borda de descida do *clock* quando ss_in está em nível alto e desativado da mesma forma quando ss_out está em nível alto, habilitando ou travando o funcionamento do circuito de sincronismo da máquina de estado. Este detalhe ocorre no inicio e fim do processamento dos *pixels* e não é apresentado na figura, que enfatiza apenas o funcionamento da máquina de estado.

O conjunto de formas de ondas apresentadas nesta seção asseguram a validação do projeto lógico do circuito integrado dedicado que foi descrito no capítulo anterior. As formas de ondas

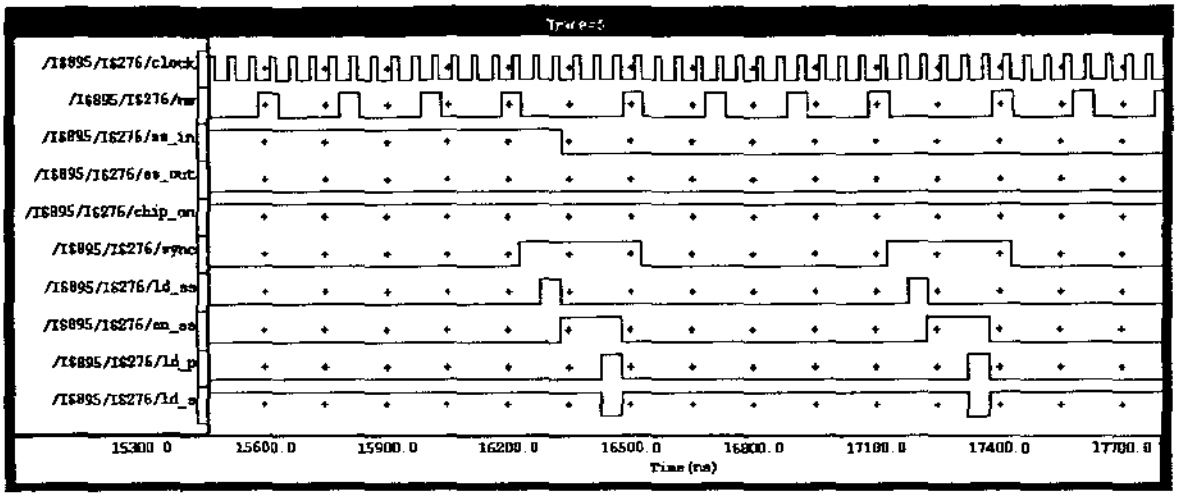


Figura 6.13: Sinais gerados na unidade de controle. Sinais internos e máquina de estado

apresentadas foram obtidas de simulações cuja descrição lógica foi realimentada (*back annotation*) com parâmetros extraídos das simulações dos projetos elétricos e físicos. Estes por sua vez foram validados à nível de células, à exemplo da validação da célula de RAM descrita na próxima seção. Uma descrição sumária das características de outras células encontra-se no apêndice B.

6.5 Exemplo de Validação Elétrica e Física para uma Célula

Na validação da célula RAM1Bit foi utilizado o circuito da figura 6.14. Todas as células diretamente ligadas a ela foram incluídas para que a simulação representasse o mais fielmente possível o comportamento da célula sob teste, no contexto da arquitetura. A célula SSEn também foi incluída desde que o atraso na propagação dos sinais SS e $\overline{SS}$ constitui um ponto crítico no funcionamento da RAM. Em alguns barramentos foram colocadas capacitâncias para representar a carga de outras células que não foram replicadas, a exemplo dos sinais $\overline{W}$, DI, SS e $\overline{SS}$. As formas de ondas aqui apresentadas foram produzidas com o simulador elétrico Spice, simulando o circuito de teste, com a descrição das células extraídas diretamente do projeto elétrico com o auxilio do editor de *layout* Magic.

A figura 6.15 apresenta as formas de ondas dos sinais envolvidos na operação de *reset* do elemento de memória. Os sinais R e nR (*Reset*) atuam nas células Decoder, WGen e Inv3St ativando os sinais Addr e $\overline{W}$ e levando DI ao nível lógico zero. Neste instante o ponto DL (ver circuito de teste na fig. 6.14) é desconectado do laço de realimentação e conectado a DI que faz com que seu potencial vá também para zero. No final da operação de *reset* o sinal addr volta ao seu estado anterior, DI fica flutuando, $\overline{W}$ volta ao nível lógico 1 e DL permanece em zero até que uma operação de escrita (treinamento) leve ele para o nível 1. DI fica flutuando desconectado de uma fonte visto que o circuito de teste possui apenas uma célula de memória. Na realidade, ele irá assumir o valor lógico da célula endereçada por uma das 16 linhas Addr que estiver ativa no momento imediatamente posterior à operação de *reset*. Como a operação de *reset* ocorre em todas as células, DI assume o

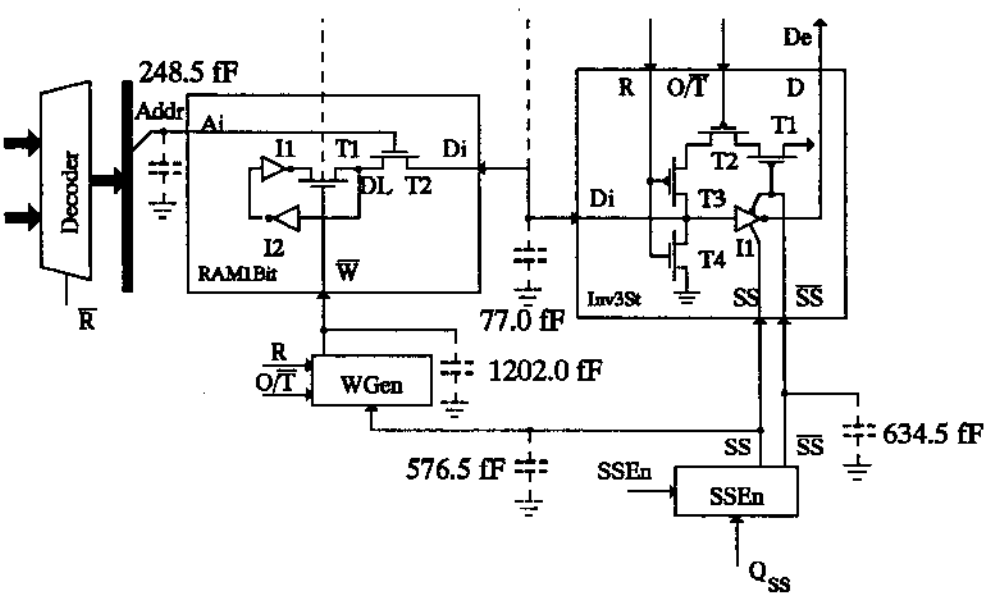


Figura 6.14: Circuito utilizado para validação da célula de um *bit* da RAM.

valor lógico zero após essa operação. A mudança do estado de $\overline{W}$ de zero para um é bastante lenta, devido a grande capacitância imposta pelas células de memória a este barramento, mas não afeta o funcionamento dos elementos de memória desde que as capacitâncias parasitas presentes no ponto DL são suficientes para manter o nível lógico deste ponto até que $\overline{W}$ atinja o nível 1, fechando o laço de realimentação. Isso permite o uso de transistores pmos (mos de canal P) pequenos e conseqüente redução de área da célula WGen.

O ciclo de escrita é apresentado na figura 6.16. O decodificador leva o sinal Addr para o nível lógico 1, quando um determinado endereço é formado pelo acúmulo de quatro *pixels* no registrador de deslocamento de entrada. O barramento interno DI é conectado ao ponto DL através do transistor de passagem controlado pelo sinal Addr e como DI está flutuando, seu estado é fixado agora no mesmo estado de DL, ou seja, zero. Quando os sinais SS e nSS são ativados, as células Inv3St e WGen forçam DI para 1 e $\overline{W}$ para 0, respectivamente. Como os pontos DI e DL ainda estão conectados, DL assume o nível lógico 1. Por fim SS e nSS são desativados, $\overline{W}$ volta ao nível 1, restabelecendo o laço de realimentação da célula de memória e Addr é desativado, desconectando a célula de memória (ponto DL) do barramento interno DI. O alto tempo de subida de $\overline{W}$ também ocorre neste ciclo e não oferece nenhum perigo de perda de informação, como já explicado.

O ciclo de leitura é semelhante ao de escrita e está representado graficamente na figura 6.17. Neste caso $\overline{W}$ permanece em nível alto e o barramento interno DI é alimentado pela célula de memória endereçada. O sinal é obtido no barramento externo DE enquanto SS e nSS estiverem ativos.

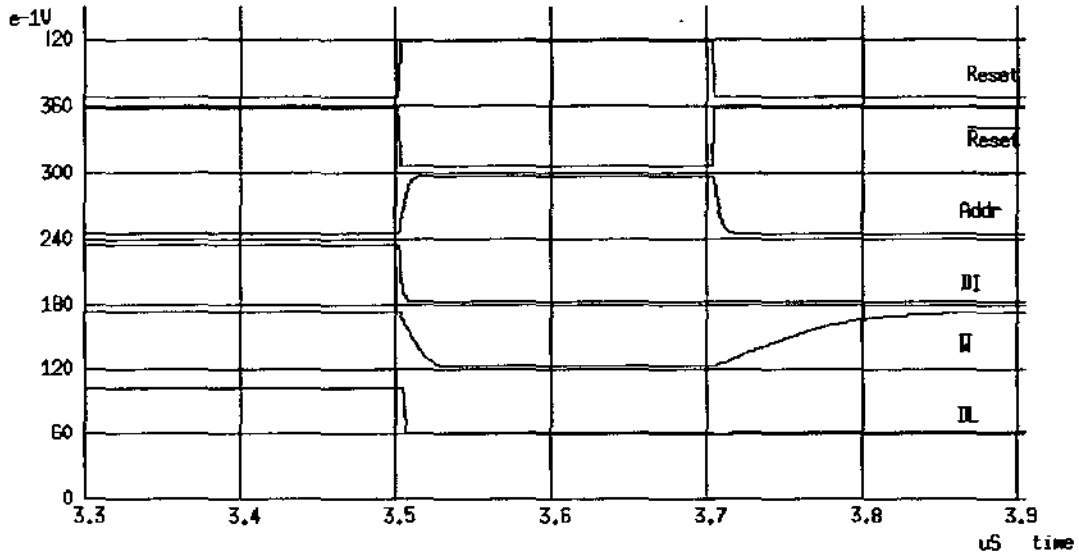


Figura 6.15: Validação analógica da operação de *reset* de uma célula de memória.

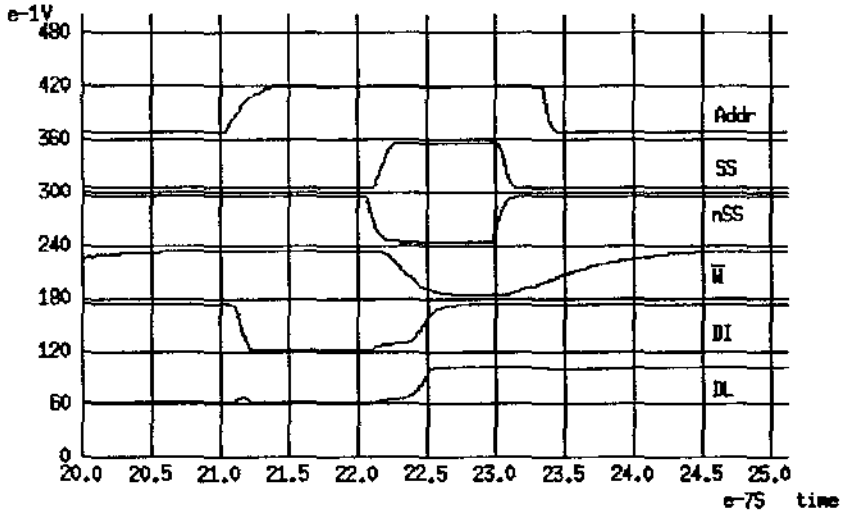


Figura 6.16: Validação analógica de uma célula de memória durante a operação de escrita.

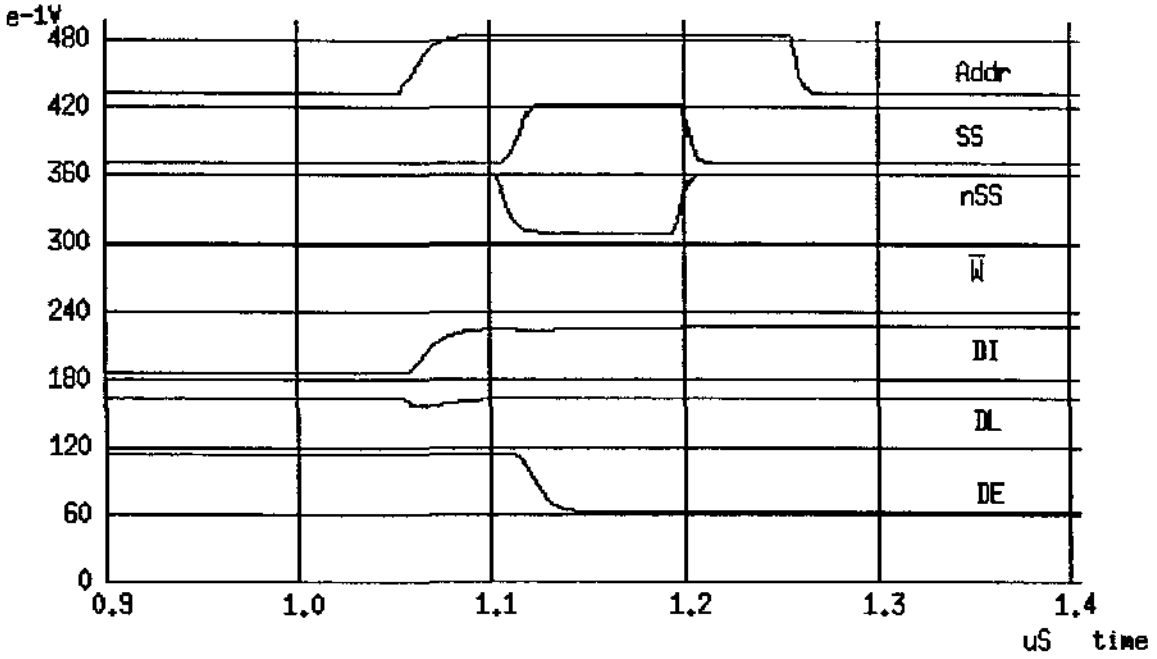


Figura 6.17: Validação analógica da operação de leitura de uma célula de memória.

6.6 Resumo e Conclusões

Foram apresentados os resultados de simulações nas diversas fase do projeto do ASIC proposto. As simulações de alto nível, como as efetuadas durante a validação da técnica *n-tuple*, possibilitaram o correto dimensionamento das estruturas do *chip*. Muitos detalhes estruturais puderam ser abstraídos, simplificando e agilizando tal dimensionamento. O relacionamento entre o desempenho, a eficiência, a área de silício etc e parâmetros como largura do barramento e capacidade das RAMs puderam ser previamente definidos nestes níveis de validações agilizando o projeto. O projeto lógico pôde ser facilmente implementado em função das validações dos níveis anteriores. Em uma primeira análise, as simulações sobre a descrição lógica garante a funcionalidade dos blocos descritos nos níveis anteriores, quando detalhados em termos de primitivas lógicas tais como, portas, *flip-flops* e *latch*. Sobre uma segunda análise, tais simulações permitem ao projetista consolidar a validação do sistema através do mecanismo de *back annotation*, onde os tempos de propagação das células lógicas são atualizados com os valores obtidos de simulações elétricas dos circuitos extraídos diretamente do *layout* das células. Este capítulo descreveu e analisou alguns resultados de simulações que além de comprovar o sucesso de cada fase do projeto, reiteraram a importância do uso de uma metodologia de projeto para o desenvolvimento de sistemas com este grau de complexidade.

Capítulo 7

Conclusão

Esta dissertação apresentou uma arquitetura para reconhecimento de padrões, resultante do desenvolvimento das fases de pesquisa de bibliografias correlatas, treinamento em ferramentas de EDA, estudo de alternativas, especificação e projeto do sistema. Durante a fase de pesquisa procurou-se identificar as características de cada *hardware* analisado, confrontando-as com os requisitos de uma aplicação típica de reconhecimento de padrões, fim de suportar o estudo de alternativas e especificação do sistema.

Por fim, é apresentada uma revisão sucinta dos temas abordados, seguida de sugestões e comentários a cerca dos trabalhos de extensão, concluindo com a apresentação de características (como flexibilidade e desempenho) e limitações do sistema.

7.1 Resumo

Foram apresentadas algumas arquiteturas para reconhecimento de imagens, especialmente as baseadas em redes neurais, bem como o projeto de um sistema para este fim, o qual inclui a implementação de um *chip* dedicado (ASIC) que utiliza a arquitetura de rede neural baseada em memória RAM. As arquiteturas apresentadas foram analisadas segundo os critérios de desempenho, paralelismo, custo, imunidade a ruídos e facilidade de implementação em VLSI. Foram também apresentados os conceitos básicos dos sistemas nervosos biológicos e seus modelos eletro-eletrônicos, com exemplos de diversos modelos e análise de implementações em VLSI. Um dos exemplos citados, o wisard, inspirou o sistema proposto, com seu princípio de funcionamento baseado em memórias RAMs. A arquitetura é adequada para aplicações de reconhecimento de classes de imagens onde o número de classes seja pequeno e a variação das imagens dos indivíduos de cada classe sejam de determinada natureza, por exemplo na inspeção visual de linha de montagem de placas de circuito impresso ou contagem de cédulas de dinheiro.

O sistema foi apresentado gradualmente de modo a transmitir para o leitor a evolução de sua concepção e as alternativas mais relevantes para o projeto. A descrição comportamental do sistema foi apresentada através de diagramas resumidos dos principais blocos do sistema, dando uma noção geral de como foi descrita a funcionalidade de cada bloco. Na implementação, as fases de projeto do ASIC foram apresentadas, enfocando a importância do uso de uma metodologia adequada que acelere o tempo de projeto de sistemas complexos através de mecanismos como uso de hierarquia e diferentes níveis de abstração. Uma regra geral e muito importante no sucesso da aplicação de tal metodologia é a validação completa de uma fase antes do inicio de fases subseqüentes. Este

critério proporciona uma otimização no tempo destinado a cada fase, antecipando a detecção de erros de projeto e por conseqüência, evitando a reavaliação de fases anteriores distantes. Com respeito a tais validações, elas foram abordadas com especial atenção às simulações de alto nível e finais. O princípio de funcionamento do ASIC foi validado num nível que abstrai detalhes da própria arquitetura, auxiliando inclusive no dimensionamento de barramentos e blocos funcionais. Os diversos ciclos de barramento do sistema e do ASIC foram apresentados através de resultados obtidos diretamente das ferramentas de análise utilizadas, durante a simulação final de cada fase.

7.2 Extensões Futuras

As extensões ao projeto apresentado consistem no (1) teste do protótipo do ASIC, (2) projeto e montagem do cartão, (3) desenvolvimento do software de controle e, (4) especificação e desenvolvimento do subsistema de aquisição de Imagens. Para o teste do protótipo é sugerida a confecção de uma jiga de teste utilizando o método clássico da aplicação de um padrão de teste previamente estabelecido, observando as respostas e comparando com aquelas esperadas. A especificação do módulo de aquisição de imagens deve ser orientada para a aplicação, onde equipamentos como câmaras de vídeo, *scanners* e placas gráficas devem ser escolhidas segundo os requisitos da aplicação (tal como execução em tempo real). Outras recomendações e sugestões relativas ao projeto e montagem do cartão, bem como ao programa de gerenciamento e controle do sistema são apresentadas em seguida.

7.2.1 Teste do Protótipo

Nos grande centros de desenvolvimento de projetos de circuitos integrados existe um equipamento denominado ATE (*Automated Test Equipment*) que é usado para aplicar um padrão de testes ao protótipo, afim de validar sua funcionalidade. Ele trabalha numa velocidade de operação elevada (tipicamente 100 Mhz) e pode controlar e acumular observações a cerca do protótipo em teste, mediante a programação de um padrão de testes. Embora tais equipamentos facilitem e acelerem a aplicação dos vetores de teste e validação do protótipo, eles são extremamente dispendiosos, sendo inacessíveis aos pequenos núcleos de desenvolvimento de CI. Uma sugestão para uma jiga simples e alguns padrões e procedimentos de testes são apresentados em seguida, tendo em vista a dificuldade de uso de tais equipamentos.

A idéia consiste no uso de portas paralelas bidirecionais no controle e/ou observação de cada pino do CI. Essa jiga pode ser montada em um cartão para PC com um simples CI 8255 e um soquete para o CI sob teste. O 8255 incorpora três portas (A, B e C) bidirecionais programáveis de 8 *bits*, das quais uma (C) pode ser programada *bit* a *bit*. O barramento de dados (16 *bits* do ASIC pode ser conectado às duas portas A e B e os demais pinos (*Clock*, SSIn, SSOut, CSL, CSC, IOR, IOW e NW) podem ser conectado à porta C. Um programa de teste pode ser desenvolvido para programação do 8255, aplicação do padrão de teste e coleta dos resultados.

A versão do ASIC que será fabricado através do PMU possui 528 RAMs que devem ser testadas sob padrões de 2.112 *pixels* ou 132 palavras de 16 *bits*. As sugestões apresentadas a seguir usaram um conjunto de 16 padrões ($P_0, P_1, ..., P_{15}$) que são formados pela repetição por 33 vezes dos subpadrões enumerados na tabela 7.1. Esta configuração de padrões permite que a cada vetor, seja endereçada a mesma posição de memória em todas as RAMs. Por exemplo, quando o padrão P_6 é usado, as RAMs terão sempre as células RAM1Bit_{06} endereçadas. Para assegurar esse comportamento, o

Padrão	*Subpadrão utilizado*							
P_0	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
P_1	0x00	0x00	0x00	0x00	0x00	0x00	0xFF	0xFF
P_2	0x00	0x00	0x00	0x00	0xFF	0xFF	0x00	0x00
P_3	0x00	0x00	0x00	0x00	0xFF	0xFF	0xFF	0xFF
P_4	0x00	0x00	0xFF	0xFF	0x00	0x00	0x00	0x00
...	...							
P_{14}	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0x00	0x00
P_{15}	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF

Tabela 7.1: Subpadrões (em hexadecimal) usados na composição dos padrões de teste $P_0, P_1, ..., P_{15}$. O padrão é formado pela repetição do subpadrão correspondente 33 vezes.

programa de teste deve substituir a ordem aleatória de busca dos *pixels* pela ordem seqüencial (0, 1, 2, ...) durante a emulação dos sinais de controle do ASIC.

- **Teste do acumulador de RAMs ativas**

  1. Programar o ASIC no modo treinamento com *reset* das RAMs e outras estruturas (palavra de controle = 5).
  2. Proceder com o treinamento selecionando um dos padrões.
  3. Programar o ASIC no modo operação com *reset* de outras estruturas (palavra de controle = 3).
  4. Proceder com a operação de reconhecimento do mesmo padrão.
  5. Verificar o conteúdo do acumulador de RAMs ativas que deve ser 100%, isto é, 528.

  Obs: Ao fim deste procedimento, é possível testar, além do funcionamento do contador, os registradores de seleção de RAMs, verificando o estado do sinal SSOut que deve transitar do nível 'zero' para 'um'.

- **Teste do *reset* das RAMs**

  1. Programar o ASIC para efetuar o *reset* das RAMs (palavra de controle = 4, 5, 6, ou 7).
  2. Programar o ASIC no modo operação com *reset* de outras estruturas (palavra de controle = 3).
  3. Proceder com o reconhecimento do padrão P_0.
  4. Verificar o conteúdo do acumulador que deve ser 0 (zero).
  5. Repetir os passos 2 a 4 para os demais padrões.

  Obs: O procedimento deste teste com todos os 16 padrões de teste permite verificar se todas as posições de memória das RAMs foram levadas para o nível lógico 0 (zero) após a operação de *reset*.

- **teste de escrita nas RAMs**

  1. Programar o ASIC no modo treinamento (palavra de controle = 5) e proceder com o treinamento do padrão P_0.
  2. Programar o ASIC no modo operação, procedendo com o reconhecimento dos 16 padrões.
  3. Verificar o conteúdo do acumulador que deve ser 0 (zero) para todos os casos, exceto para P_0 que deve ser 100% ou 528.
  4. Repetir o procedimento para os demais padrões, substituindo P_0 por cada um dos outros padrões.

  Obs: O reconhecimento dos 16 padrões para cada um dos padrões treinados assegura que a escrita do *bit* '1' ocorre apenas nas células de memória endereçadas. Em muitos projetos de memória é comum o efeito onde a mudança de estado de um elemento de memória perturba ou corrompe o conteúdo de células vizinhas. Este efeito em geral tende a mascarar a detecção das falhas em função da ordem com que os padrões de teste são aplicados[Côr91] (*pattern sensitive faults*). Tais tipos de falhas se manifestam em memórias com amplificadores analógicos (*sense amplifier*), sensíveis a pequenas variações nos níveis de tensão armazenados nas células de memórias, usados para diminuir o tempo de acesso de tais memórias. Considerou-se desnecessário um cuidado especial com cobertura de falhas deste tipo no protótipo em questão pelo fato das memórias terem sido implementadas sem uso de tais amplificadores, em uma versão inteiramente digital.

7.2.2 Projeto e Montagem do Cartão

O projeto do cartão (PCB) que acomodará a matriz de ASICs, o controlador externo e alguns *buffers*, deve respeitar as dimensões de um cartão padrão para PCs IBM ou compatíveis. A escolha do dispositivo que ira implementar o controlador externo deve estar previamente definido. A esse respeito foram sugeridas algumas opções no capítulo 5. O projeto deve ser desenvolvido num ambiente CAD, a exemplo do módulo de apoio a projetos de PCBs da Mentor Graphics (*Board Station*), onde as ferramentas usadas podem acelerar o projeto e detectar algum provável erro antes de sua execução.

7.2.3 Especificação do Modulo de Aquisição e Pré-processamento de Imagens

Em linhas gerais, o módulo de aquisição e pré-processamento de imagens pode ser composto de uma câmara de vídeo, cartão com acelerador gráfico e *software* para o pré-processamento das imagens. Esta configuração pode variar de acordo com a aplicação. Por exemplo, no reconhecimento de cédulas a câmara de vídeo pode ser substituída por um *scanner*. O cartão acelerador gráfico pode ser dispensável mas se a aplicação exige processamento em tempo real de execução, sua aquisição pode ser uma boa alternativa. O *software* requerido deve ser capaz de executar funções tais como, conversão para padrão preto e branco, normalização e alinhamento das imagens. Seu desenvolvimento deve ser orientado ao uso dos recursos do acelerador gráfico citado, agilizando o pré-processamento das imagens nas aplicações com altas taxas de entrada de imagens, como na inspeção de peças de uma linha de montagem.

7.2.4 Desenvolvimento do Programa de controle

O programa de controle deve ser capaz de programar a matriz de ASICs e PLD, coordenando as atividades de treinamento e operação. Ele deve "conversar" com o módulo pré-processador de modo a obter parâmetros tais como, tamanho e endereço do topo da imagem, para programar os registradores internos da PLD. Uma alternativa interessante seria o desenvolvimento das funções de controle e pré-processamento em um único programa, eliminando os custo de comunicações entre dois processos. Por fim, o programa teria uma interface onde o usuário poderia configurar o sistema e acompanhar os resultados. Tais resultados poderiam, além de ser exibidos via interface, estar disponíveis em uma porta de I/O, alimentando um sistema de automação, tal como um braço mecânico.

7.3 Conclusões

O sistema para reconhecimento de imagens proposto permite a manipulação de imagens com algum conteúdo de ruído, de maneira flexível, com baixo custo e bom desempenho. A técnica utilizada, como demonstraram as simulações de alto nível, permite ao sistema tratar imagens que apresentam certo nível de ruído. Sua arquitetura escalável permite ao usuário dimensionar o sistema, de acordo com as necessidades da aplicação, pela simples adição de CIs dedicados em um cartão para PC-AT. A utilização de um computador como o PC-AT e de seus recursos resultou em um projeto mais simples e de menor custo. A disposição física dos componentes do sistema permite também a utilização de outros computadores mais potentes, bastando para isso, reprojetar a PLD de controle. Quanto ao desempenho, são feitas algumas estimativas e comparações abaixo.

Analisando o diagrama da máquina de estados (figura 4.8) pode-se estimar que o sistema proposto, a um *clock* de 33Mhz, será capaz de processar um bloco de 104 palavras em $15\mu s$. Supondo-se que o cartão consiga o controle do barramento toda vez que ele for requisitado, uma imagem de 128×128 *pixel* (1024 palavras) pode ser processada em $150\mu s$ ($1024 \div 104 \times 15\mu s$). Estimativas prevêm que um PC-386 33Mhz executando um algoritmo de reconhecimento de imagem baseado na equação que mede o erro quadrático médio entre duas imagens [Ree82] com as mesmas dimensões da anterior (128×128 *pixels*), leva 1,34 s neste processamento. Reeves [Ree82] estima que o tempo de execução em uma matriz de processadores binários neste caso é de $618\mu s$. É interessante notar que dependendo do tamanho das imagens o cartão pode reconhecer, operando em paralelo, uma imagem entre até oito imagens, treinadas uma em cada linha da matriz de CIs. No caso do PC-386 e da matriz de processadores o tempo de processamento aumentaria em pelo menos oito vezes.

Devido à constituição do *hardware*, o tamanho máximo da imagem que o sistema pode processar é linearmente proporcional ao tamanho de seus discriminadores (RAMs). De acordo com os cálculos do projeto físico cada chip (versão anterior à versão para o PMU) pode processar uma imagem com um tamanho máximo de 16384 (128×128) *pixels*, já que os discriminadores possuem 16 linhas e 256 colunas e cada RAM é endereçada com 4 *pixels* ($16 \times 256 \times 4 = 16384$). Isto significa que uma linha da matriz de *chips* com 8 *chips* pode processar 131072 ($\simeq 362 \times 362$) *pixels* (8×16384). Com a devida interligação entre as 8 linhas da matriz o cartão pode processar uma imagem de ($\simeq 1024 \times 1024$) *pixels*, o que é mais que suficiente para as aplicações típicas. Estes cálculos estão de acordo com as dimensões da versão anterior ao do protótipo destinado ao PMU (figura 5.10). Com a redução para 33 colunas de RAMs, a capacidade de processamento do protótipo ficou reduzida para imagens de 2112 *pixels* ($\simeq 46 \times 46$).

Apêndice A

Especificação do sistema

A.1 Introdução

Este apêndice fornece informações complementares de um sistema para reconhecimento de imagens que utiliza um chip especialmente dedicado a esse fim (que é tratado no apêndice B). A arquitetura e sua funcionalidade é brevemente apresentada (seção A.2), uma vez que foi exaustivamente discutida no corpo da dissertação (capítulo 4). A programação dos registradores interno da PLD de controle é detalhada juntamente com o *timing* dos sinais envolvidos nos principais ciclos de barramento do sistema. A seção A.2 ainda apresenta os valores máximos, mínimos e típicos para os atrasos de propagação desses sinais.

A seção A.3 apresenta a listagem do programa de simulação usado para a validação da técnica *n-tuple*, desenvolvido em Pascal. O programa e sua estrutura de dados foram explicados no capítulo 6 (seção 6.2.1). As descrições comportamentais do PC-AT e PLD de controle, desenvolvidas em VHDL, também são apresentadas nesta seção. Tais descrições foram abordadas no capítulo 4 seção 4.4.2). Estas listagens complementam as informações a respeito do projeto, facilitando a execução de trabalhos futuros ou mesmo fornecendo material mais específico aos leitores interessados na área.

A.2 Arquitetura

A arquitetura do sistema é apresentada na figura A.1. Este consiste de um cartão IBM PC-AT com uma matriz de *chips* dedicado (ver apêndice B) e uma PLD de controle. A PLD de controle tem a função de alimentar a imagem a ser processada na matriz. Outras funções, como decodificação de endereços de IO e lógica para requisição de barramento também foram incorporadas à PLD.

A organização interna da PLD é apresentada na figura A.2. A lógica de controle efetua a decodificação dos endereços de IO, selecionando um *chip* na matriz ou um de seus registradores internos dependendo do endereço gerado pela CPU. A máquina de estados controla os ciclos sobre o barramento e é a principal responsável pela requisição do barramento e transferência da imagem para a matriz. O LFSR é responsável pela geração de endereços aleatórios dentro do espaço de endereçamento ocupado pela imagem. Os registros de programação armazenam parâmetros que vão alimentar o LFSR e a máquina de estados para produzirem os efeitos desejados.

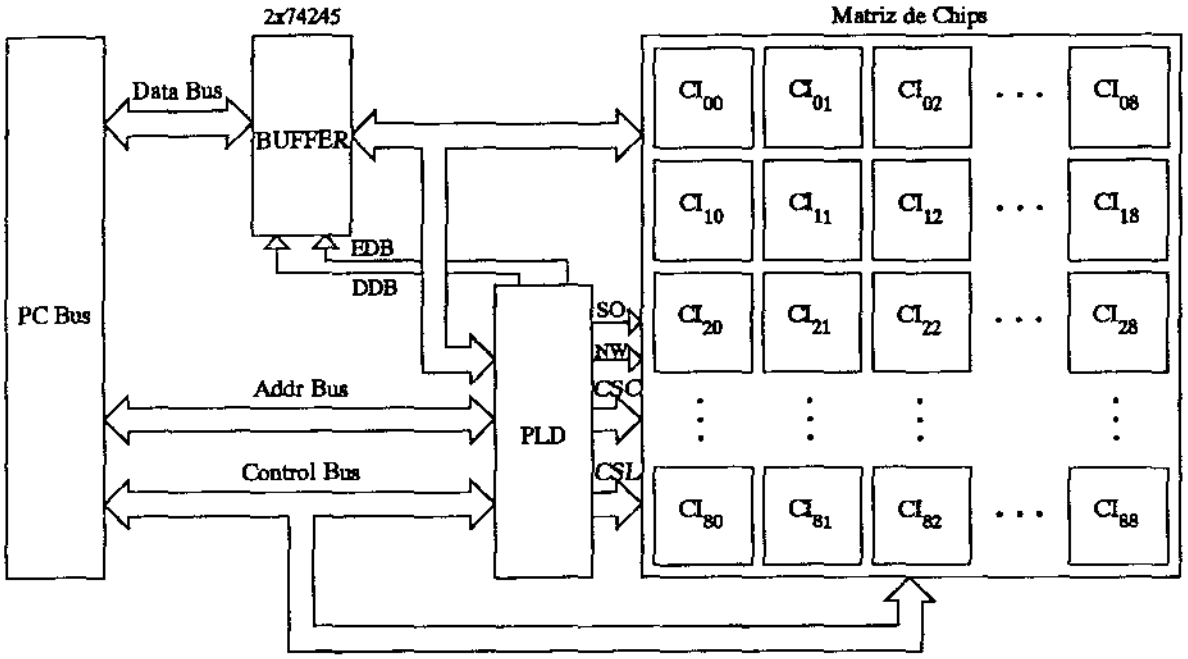


Figura A.1: Arquitetura do sistema.

A.2.1 Programação dos Registros de Controle

A PLD de controle foi concebida de modo a efetuar a transferência da imagem, para a matriz, em blocos intercalados entre os ciclos de *refresh* de memória dinâmica do PC, devolvendo o controle do barramento à CPU sempre que um ciclo de *refresh* é necessário. Para permitir que imagens com diferentes tamanhos sejam transferidas deste modo para a matriz, existem dois registros de controle denominados SIR e SBR (ver figura A.3). O registrador SIR determina o tamanho da imagem em números de blocos necessários à transferência. O registrador SBR determina o tamanho dos blocos em números de grupos de 4 palavras (G1, G2, ..., Gn). O SBR deve ser programado de modo que o tempo necessário para transferência de um bloco não seja maior que o intervalo de tempo máximo entre dois ciclos de *refresh*. O efeito destes registradores sobre a transferência da imagem também é apresentado na figura A.3.

O registrador BIR define o topo da região de memória onde encontra-se a imagem a ser processada. O LLR define a largura do LFSR (em *bits*) de modo que os endereços sejam gerados sempre dentro da região de memória reservada para a imagem. O SF dispara o funcionamento da máquina de estados dando início a transferência e é programado juntamente com o *byte* mais significativo do registrador SIR. Os registradores são programados pela CPU através de operação de escrita em dispositivos de IO de 8 *bits*. Os endereços e respectivos registradores, os quais foram usados na descrição comportamental listada na seção A.3.3, são apresentados na figura A.3. O ultimo *byte* a ser programado deve ser o referente ao SF (endereço 13 FF) uma vez que este dispara o processo de transferência da imagem.

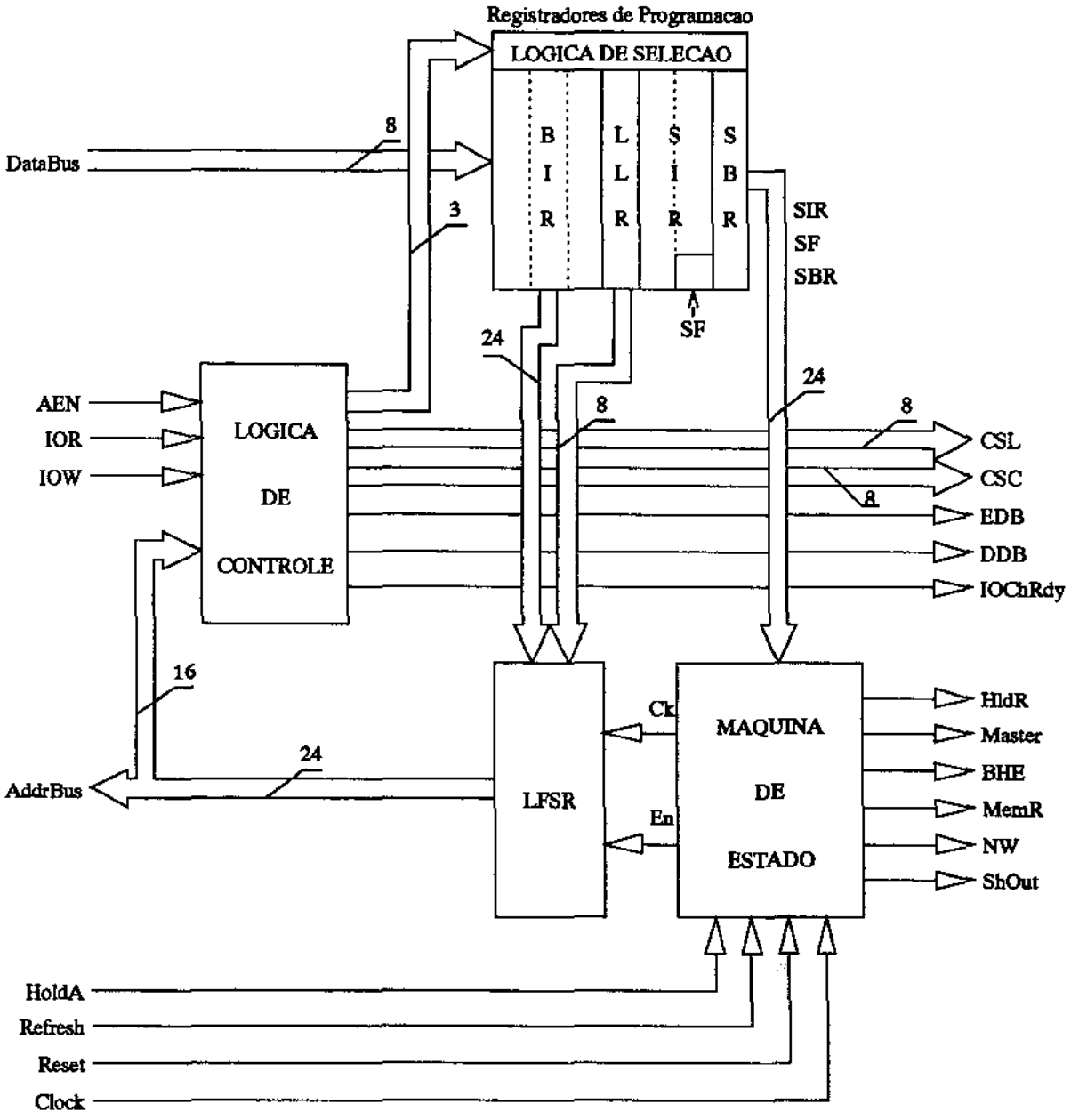


Figura A.2: Arquitetura do bloco de controle do cartão (PLD)

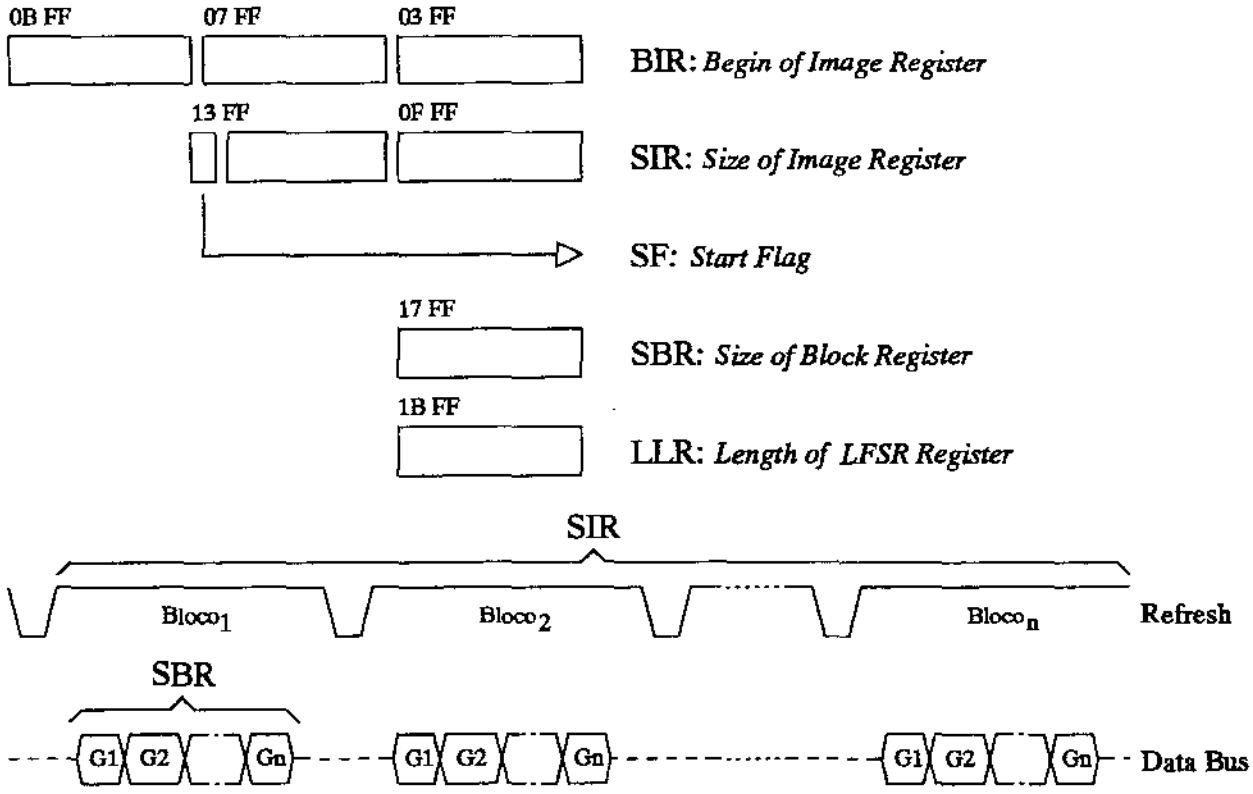


Figura A.3: Registros de controle da PLD e o efeito da programação dos registros SIR e SBR na transferência da imagem para os discriminadores

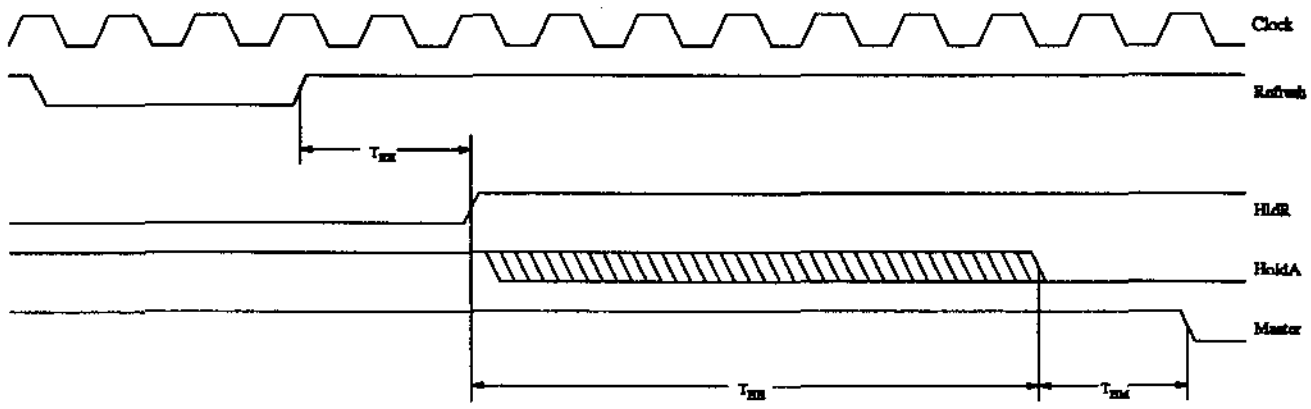


Figura A.4: Ciclo de requisição de barramento

A.2.2 Temporização dos Sinais

As formas de ondas das figuras A.4 à A.7 e a tabela A.1, apresentam a temporização de sinais para os principais ciclos sobre o sistema. Os dados apresentados foram compilados de informações obtidas dos manuais do PC-AT e CPU 80386 [IBM84][Int86] e de simulações com o modelo lógico final do sistema.

A figura A.4 apresenta o ciclo de requisição de barramento. O sinal HldR sinaliza para o PC que uma CPU externa (PLD) está requerendo o barramento. Este sinal é gerado pela PLD e é síncrono com o sinal de Refresh. O sinal HoldA, gerado pelo PC, confirma o pedido de barramento para a PLD. Tal confirmação deve ocorrer no máximo 8 ciclos depois fim do ciclo de *refresh* garantido que o a transferência de um bloco seja inteiramente feita entre dois ciclos consecutivos de *refresh*.

A figura A.5 apresenta o ciclo de escrita de IO. O dado deve ser escrito no dispositivo de IO (PLD ou *chips*) durante a subida do sinal IOW quando o barramento de dados esta definido e estável. O sinal IOChRdy pode ser usado para adicionar wait states, compatibilizando a velocidade entre o dispositivo de IO e o PC. O ciclo de leitura de IO (usado para ler o resultado dos discriminadores) é apresentado na figura A.6. Análogo ao ciclo de escrita, o *chip* deve definir, de maneira estável, o barramento de dados um pouco antes (T_{DS}) da subida do sinal de IOW, quando o PC assume que o dado já está disponível. Os sinais CSC e CSL são gerados pela PLD em função da decodificação do endereço do dispositivo, selecionando o *chip* conveniente.

A figura A.7 apresenta o ciclo de leitura de memória. Neste caso, a PLD coloca o endereço gerado pelo LFSR no barramento de endereços e ativa o sinal MemR. As memórias do PC devem carregar o barramento de dados antes da subida de MemR, quando o dado é transferido para a matriz de *chips*.

A.3 Descrição Comportamental do Sistema

Esta seção apresenta a listagem do programa de simulação para a validação da técnica n-tuple, descrita no capítulo 6, e as listagens em VHDL para o barramento do PC e da PLD de controle, as quais são abordadas no capítulo 4. As demais listagens, referentes a matriz de *chips*, são apresentadas no apêndice B.

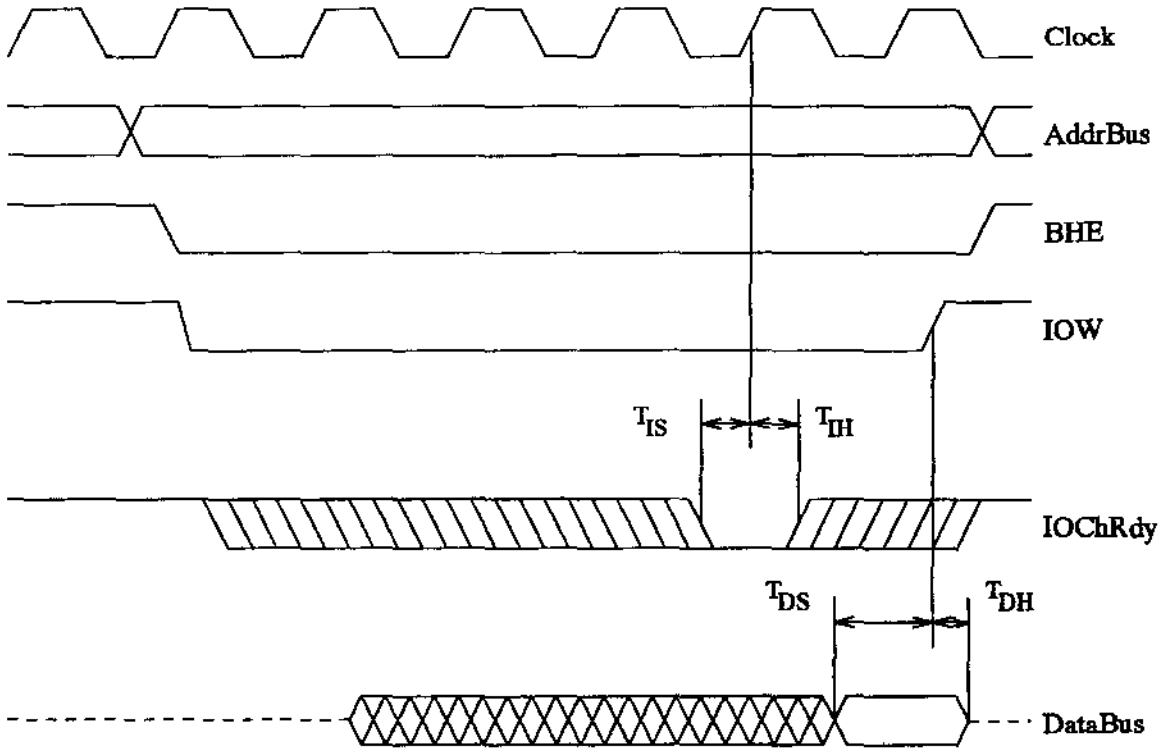


Figura A.5: ciclo de escrita de IO

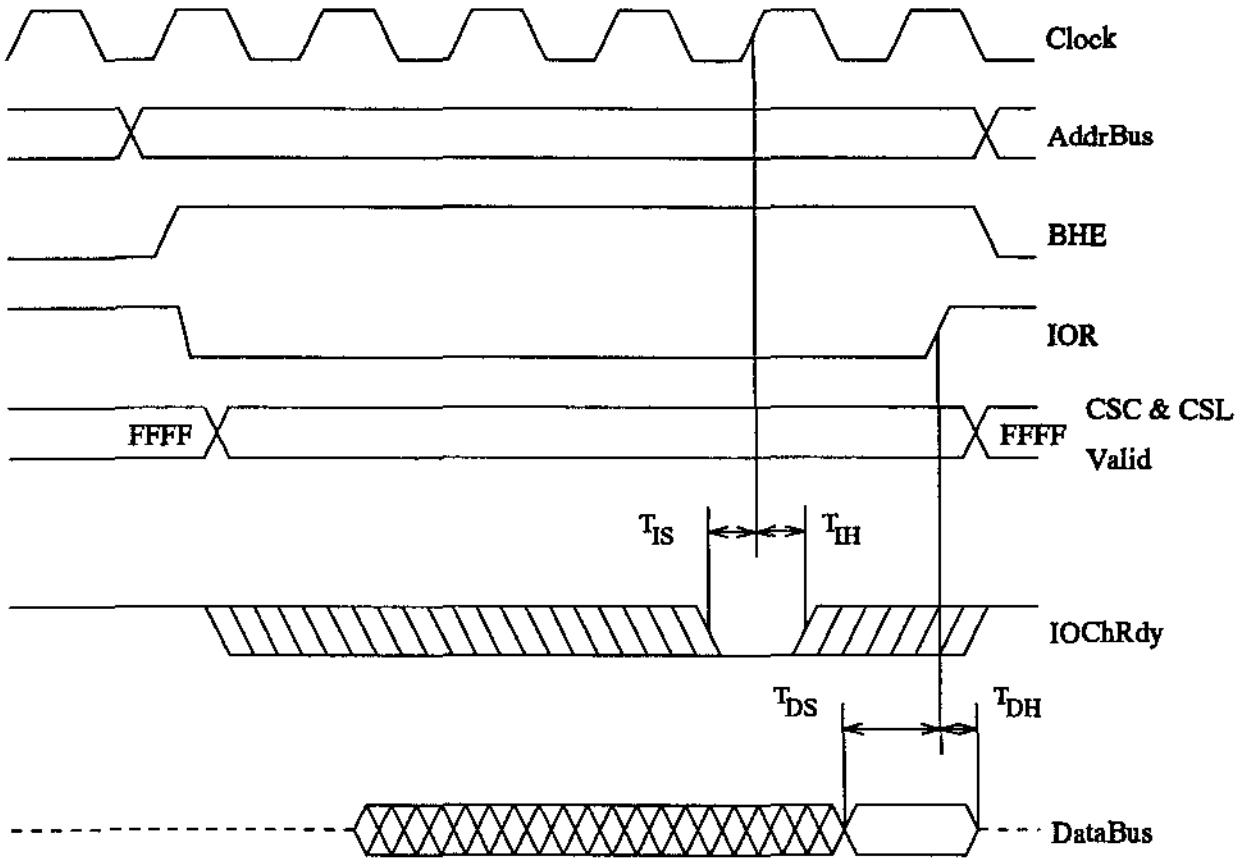


Figura A.6: Ciclo de leitura de IO

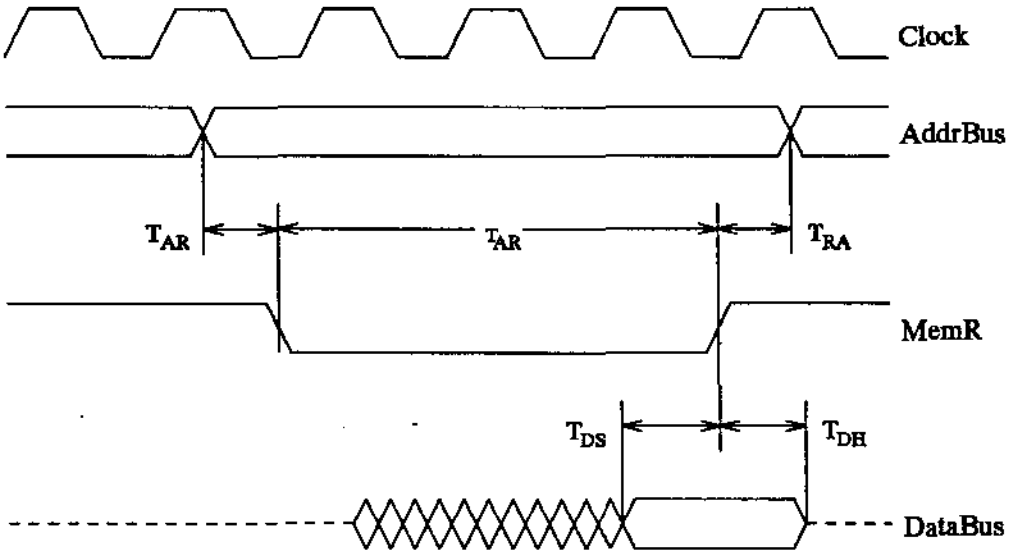


Figura A.7: Ciclo de leitura de memória

Símbolo	Parâmetro	Min	Max	Típico	Unidade	Ciclo
T_RH	Propagação entre HldR e Refresh		100	68	ns	reqbus
T_HH	Propagação entre HoldA e HldR		215	16	ns	reqbus
T_HM	Propagação entre Master e HoldA	35	65	50	ns	reqbus
T_IS	IOChRdy *setup timing*	> 22		37	ns	iordy
T_IH	IOChRdy *hold timing*	< 5		35	ns	iordy
T_DS	DataBus *setup timing*	15		22	ns	ior/w
T_DH	DataBus *hold timing*	5		5	ns	ior/w
T_AR	AddrBus *setup timing*			19	ns	memr
T_RA	AddrBus *hold timing*			33	ns	memr
T_DS	DataBus *setup timing*	15		100	ns	memr
T_DH	DataBus *hold timing*	0			ns	memr

Tabela A.1:

A.3.1 Listagem do Programa de Validação da Técnica *n-Tuple*

```
Program WS8;
{-------------------------------------------------------------------------}
{  Programa para simulação da técnica n-tuple implementada com          }
{  memórias RAMs para reconhecimento de padrões.                        }
{  José M. Lanna: Versão com variáveis estáticas (Jan/93)               }
{  Josemir C. Alexandrino: Variáveis dinâmicas e interface (Out/93)     }
{-------------------------------------------------------------------------}

Uses Crt, Graph, Common;

  Type
    RAMPtr = ↑Byte;
    DiscType = Record
      DiscPtr,                  { Ponteiro das RAMs do discriminador          }
      SelRAM:     RAMPtr;       { Ponteiro da RAM selecionada                 }
      RAMCounter: LongInt;      { Contador de RAMs que já foram selecionadas  }
      SizeOfRAM:  Byte;         { Tamanho da RAM = (8 * SizeOfRAM) Bits       }
      SizeOfDisc,               { Numero de RAMs do discriminador             }
      Acc,                      { Contador de RAMs ativas (no modo operação)  }
      OnesAccount: LongInt;     { Numero de bits (do discriminador) em '1'    }
    End;
    StreamFile = File Of Byte;

  Const
    NULL =      #0;
    BEEP:       Char = ' ';
    ESC =       #27;
```

```
    CR =          #13;
    SINGLE =      FALSE;
    DUBBLE =      TRUE;
    LEARNE =      TRUE;
    RECOGNITION = FALSE;
    WITHSTOP =    -1;
    MENUMSG:      String = 'Entre com a primeira Letra de uma ' +
                           'das opcoes do menu ou Esc para sair';
    Base:         String = 'COLLOR';
    Inicio:       Byte = 0;
    Fim:          Byte = 0;
    InFileName:   String = 'COLLOR00';
    Disc:         DiscType = (DiscPtr:     Nil;
                              SelRAM:      Nil;
                              RAMCounter:  0;
                              SizeOfRAM:   2;
                              SizeOfDisc:  3168;
                              Acc:         0;
                              OnesAccount: 0);
    SizeOfBus:    Byte = 2;
    SizeOfRAMBus: Byte = 4;

Procedure MakeBox(Borda: Boolean; X1, Y1, X2, Y2: Byte; Title: String);
{------------------------------------------------------------------------}
{ Define uma janela com bordas simple (desativada) ou dupla (ativada) com }
{ canto superior esquerdo (X1, Y1) e inferior direito (X2, Y2) e Title no }
{ topo.                                                                   }
{------------------------------------------------------------------------}
  Var
    v, h, Luc, Ruc, Ldc, Rdc: Char;
    i, l: Byte;
  Begin
    If (Borda)
      Then Begin
        v   := #186; h   := #205;
        Luc := #201; Ruc := #187;
        Ldc := #200; Rdc := #188;
      End
      Else Begin
             v   := #179; h   := #196;
        Luc := #218; Ruc := #191;
        Ldc := #192; Rdc := #217;
      End;
    Window(1, 1, 80, 25);
    Gotoxy(X1, Y1); Write(LUC);
```

```
    l := (X2 - X1 - 3 - Length(Title)) Div 2;
    For i := 1 To l Do Write(H);
    Write(' ', Title, ' ');
    For i := 1 To l Do Write(H);
    If (Wherex ≠ X2) Then Write(H);
    Write(RUC);
    For i := Y1 + 1 To Y2 - 1 Do Begin
      Gotoxy(X1, i); Write(V);
      Gotoxy(X2, i); Write(V);
    End;
    Gotoxy(X1, Y2); Write(LDC);
    For i := X1 + 1 To X2 -1 Do Write(H);
    Write(RDC);
  End;

Procedure MakeMenuScr(Borda: Boolean);
{------------------------------------------------------------------------------}
{ Define janela de menu com as operações disponíveis.                          }
{------------------------------------------------------------------------------}

  Begin
    MakeBox(Borda, 2, 2, 79, 4, 'MENU');
    Window(3, 3, 78, 3); ClrScr;
    Gotoxy( 1,  1); Write('[T]reinamento');
    Gotoxy(20,  1); Write('[O]peracao');
    Gotoxy(36,  1); Write('[R]eset');
    Gotoxy(49,  1); Write('[V]isualizacao');
    Gotoxy(69,  1); Write('[Esc]');
  End;

Procedure MakeTreScr(Borda: Boolean);
{------------------------------------------------------------------------------}
{ Define janela de dialogo para a função Treinamento.                          }
{------------------------------------------------------------------------------}

  Begin
    MakeBox(Borda, 2, 5, 40, 20, 'TREINAMENTO');
    Window(3, 6, 39, 19); If Borda Then ClrScr;
    Gotoxy(2,  2); Write('Base:');
    Gotoxy(2,  3); Write('Inicio:');
    Gotoxy(2,  4); Write('Fim:');
    Gotoxy(2,  8); Write('Bus:');
    Gotoxy(2,  9); Write('RAMs:');
    Gotoxy(2, 10); Write('N', #167);
    MakeBox(SINGLE, 3, 17, 21, 19, 'Densidade');
    MakeBox(SINGLE, 3, 12, 21, 16, 'Discriminador');
    MakeBox(SINGLE, 3,  6, 21, 11, 'Input');
```

```
    MakeBox(SINGLE, 22, 6, 39, 19, 'Imagem(s)');
    Window(3, 6, 39, 19);
  End;

Procedure MakeOpeScr(Borda: Boolean);
{-------------------------------------------------------------------------}
{ Define janela de dialogo para a função Operação.                        }
{-------------------------------------------------------------------------}

  Begin
    Makebox(Borda, 41, 5, 79, 8, 'OPERACAO');
    Window(42, 6, 78, 7); If Borda Then ClrScr;
    Gotoxy(1, 1); Write('Arquivo:');
    Gotoxy(1, 2); Write('Resultado:');
  End;

Procedure MakeResetScr(Borda: Boolean);
{-------------------------------------------------------------------------}
{ Define janela de dialogo para a função Reset.                           }
{-------------------------------------------------------------------------}

  Begin
    MakeBox(Borda, 41, 9, 79, 11, 'RESET');
    Window(42, 10, 78, 10); If Borda Then ClrScr;
  End;

Procedure MakeShowScr(Borda: Boolean);
{-------------------------------------------------------------------------}
{ Define janela de dialogo para a função Visualização.                    }
{-------------------------------------------------------------------------}

  Begin
    MakeBox(Borda, 41, 12, 79, 14, 'VISUALIZACAO');
    Window(42, 13, 78, 13); If Borda Then ClrScr;
    Gotoxy(1, 1); Write('Arquivo:');
  End;

Procedure MakeMenssageScr(Borda: Boolean);
{-------------------------------------------------------------------------}
{ Define janela de menssagens.                                            }
{-------------------------------------------------------------------------}

  Begin
    MakeBox(Borda, 2, 21, 79, 23, 'MENSAGENS');
    Window(3, 22, 78, 22); If Borda Then ClrScr;
  End;

Procedure Menssage(MenStr: String);
{-------------------------------------------------------------------------}
{ Imprime uma mensagem na janela de menssagens.                           }
```

```
{-------------------------------------------------------------------------}
  Var X1, Y1, X2, Y2, Cx, Cy: Byte;
  Begin
    X1 := Lo(WindMin)+1; Y1 := Hi(WindMin)+1;
    X2 := Lo(WindMax)+1; Y2 := Hi(WindMax)+1;
    Cx := Wherex;        Cy := Wherey;
    Window(3, 22, 78, 22);
    Write(MenStr, ' ':74 - Length(MenStr), BEEP);
    Window(X1, Y1, X2, Y2);
    Gotoxy(Cx, Cy);
  End;

Function min(Arg1, Arg2: Integer): Integer;
{-------------------------------------------------------------------------}
{Retorna o menor dos argumentos Arq1 ou Arg2.                             }
{-------------------------------------------------------------------------}

Begin
   If Arg1 < Arg2 Then min := Arg1 Else min := Arg2;
End;

Function OpenStreamFile(FileName: String;
                                  Var InFile: StreamFile): boolean;
{-------------------------------------------------------------------------}
{ Abre um arquivo tipo Stream e retorna TRUE se a operação for bem        }
{ sucedida ou FALSE caso contrario.                                       }
{-------------------------------------------------------------------------}

  Begin
    Assign(InFile, FileName + '.STR');
    {$I-} Reset(InFile) {$I+};
    OpenStreamFile := (IOResult = 0);
  End;

Procedure Show(InFileName: String; SDelay: Integer);
{-------------------------------------------------------------------------}
{ Exibe a imagem armazenada no arquivo InFileName no modo gráfico por um  }
{ um tempo SDelay ou até alguma tecla ser pressionada quando SDelay = 0.  }
{-------------------------------------------------------------------------}

  Const
    SizeOfVideoBuf: Integer = 4000;
  Var
    InFile:      StreamFile;
    TC:          Char;
    VideoBufBak: Array[0..3999] Of Byte;
    VideoBuf:    Char Absolute $B800:$0000;
    Buf:         Array[1..2] Of Byte;
```

```
    XY1, XY2:   Integer;
    Cx, Cy:     Byte;
    GrDriver:   Integer;
    GrMode,     ErrorCode,
    WidthImg,   HeightImg,
    Width,      Height,
    LeftMarg,   UpMarg,
    MaxX,       MaxY,
    PosX,       PosY,
    i, j, k:    Integer;
Begin
    MakeMenuScr(SINGLE);
    MakeShowScr(DUBBLE);
    Repeat
        Menssage('Entre com o nome da imagem a ser Mostrada');
        InputStr(InFileName, 8, 10, 1, ['0'..'9', 'a'..'z', 'A'..'Z'], [CR], TC);
    Until OpenStreamFile(InFileName, InFile);
    Move(VideoBuf, VideoBufBak, SizeOfVideoBuf);
    XY1 := WindMin;
           XY2 := WindMax;
           Cx := Wherex;        Cy := Wherey;
    GrDriver := Detect;
    InitGraph(grDriver,grMode,'');
    ErrorCode := GraphResult;
    If ErrorCode ≠ GROK
              Then WriteLn('Graphics error: ', GraphErrorMsg(ErrorCode))
       Else Begin
          Read(InFile, Buf[1]); Read(Infile, Buf[2]);
                          Width := 256 * Buf[1] + Buf[2];
          Read(InFile, Buf[1]); Read(Infile, Buf[2]);
                          Height := 256 * Buf[1] + Buf[2];
          WidthImg := Min(Width, GetMaxX);
          HeightImg := Min(height, GetMaxY);
          LeftMarg := (GetMaxX - WidthImg) Div 2;
          UpMarg := (GetMaxY - HeightImg) Div 2;
          MaxX := LeftMarg + WidthImg;
          MaxY := UpMarg + HeightImg;
          PosY := UpMarg;
          For i := 1 To Height Do Begin
             PosX := LeftMarg;
             For j := 1 To Trunc((width + 7) / 8) Do Begin
                Read(InFile, Buf[1]);
                For k := 0 To 7 Do Begin
                   If (PosX ≤ MaxX) And (PosY ≤ MaxY) Then
                      PutPixel(PosX, PosY, (Buf[1] ShR k) And 1);
```

```
                    Inc(Posx);
                End;                    { For k }
            End;                        { For j }
            Inc(PosY);
        End;                            { For i }
    End;                                { Else }
        If SDelay > 0
            Then Delay(SDelay)
    Else Begin Repeat Until KeyPressed; TC := ReadKey; End;
  CloseGraph;
  Move(VideoBufBak, VideoBuf, SizeOfVideoBuf);
        Window(Lo(XY1)+1, Hi(XY1)+1, Lo(XY2)+1, Hi(XY2)+1);
  GotoXY(Cx, Cy);
  MakeShowScr(SINGLE);
  Menssage(MENUMSG);
  MakeMenuScr(DUBBLE);
End;

Procedure DiscReset(Var Disc: DiscType);
{-------------------------------------------------------------------------}
{ Leva todas as posições de memória das RAMs e o campo OnesAccount do     }
{ discriminador para 0 (zero).                                            }
{-------------------------------------------------------------------------}
  Var
    i:   LongInt;
    Resp: String;
    CT:  Char;
  Begin
    MakeMenuScr(SINGLE);
    MakeResetScr(DUBBLE);
    Write('Reset (S/N)? '); Resp := 'N';
    Menssage('ATENCAO: Confirme Operacao de reset');
    InputStr(Resp, 1, 14, 1, ['S', 'N'], [CR], CT);
    If (Resp = 'S') Then Begin
      Menssage('Reset ...');
      Disc.SelRAM := Disc.DiscPtr;
      For i := 1 To Disc.SizeOfDisc * Disc.SizeOfRAM Do Begin
        Disc.SelRAM↑ := 0;
        Inc(Disc.SelRAM);
      End;
      Disc.OnesAccount := 0;
      Gotoxy(1, 1); Write('Discriminador resetado');
    End;
    MakeResetScr(SINGLE);
    Menssage(MENUMSG);
```

```
      MakeMenuScr(DUBBLE);
   End;

Function ReadWriteRAM(Mode:    Boolean;
                      Var Disc: DiscType;
                      AddrByte,
                      AddrBit: Byte;
                      Var Ppt: Byte): Boolean;
{-------------------------------------------------------------------------}
{ Executa o Treinamento ou Operação sobre a RAM atualmente selecionada    }
{ e retorna TRUE para operações bem sucedidas.                            }
{-------------------------------------------------------------------------}

Const PptType: Array[0..3] Of Char = ('-','\','—','/');
Var i: Byte;
Begin
   If Ppt Mod 80 = 0 Then Begin
      Write(PptType[Ppt div 80]);
      GotoXY(WhereX-1, WhereY);
      If Ppt = 240 Then Ppt := 255;
   End;
   Inc(ppt);
   With Disc Do Begin
      If RAMCounter < SizeOfDisc * SizeOfRAM
                Then Begin
           For i := 1 To AddrByte Do Inc(SelRAM);
           If Mode
                          Then Begin
                If ((SelRAM↑ ShR AddrBit) And 1 = 0) Then Begin
                   Inc(OnesAccount);
                   SelRAM↑ := SelRAM↑ Or (1 ShL AddrBit);
                End;
              End
              Else
                 Acc := Acc + ((SelRAM↑ ShR AddrBit) And 1);
           For i := AddrByte + 1 To SizeOfRAM Do Inc(SelRAM);
           Inc(RAMCounter);
           ReadWriteRAM := TRUE;
         End
         Else ReadWriteRAM := FALSE;
   End;
End;

Procedure LFSR(Var State: LongInt; MaxState: LongInt; Len: Byte);
{-------------------------------------------------------------------------}
{ Retorna em State um numero pseudo-aleatório que é usado na busca de     }
{ uma palavra do arquivo de imagem.                                       }
```

```
{_____________________________________________________________}
Const
   POLY: Array [7..17] Of LongInt = (131, 285, 529, 1033,
      2053, 4179, 8219, 16441, 32771, 65593, 131081);
Var
   Aux1, Aux2: LongInt;
   i, fb, zd: Byte;
Begin
   Repeat
      zd := 0; fb := 0;
      Aux1 := State And POLY [Len];
      Aux2 := State;
      For i := 0 To Len-1 Do Begin
         fb := fb Xor (Aux1 And 1);
         zd := zd  Or (Aux2 And 1);
         Aux1 := Aux1 ShR 1;
         Aux2 := Aux2 ShR 1;
      End;
      State := (State ShR 1) + ((fb Xor ((Not zd) And 1)) ShL (Len-1));
   Until (State ≤ MaxState)
End;

Function Make(Mode:          Boolean;
                  Var Disc:       DiscType;
                  Var TrainingFile: StreamFile): Boolean;
{_____________________________________________________________}
{ Gerencia as funções de treinamento e operação, enche o Buffer com }
{ palavras aleatoriamente escolhidas do arquivo de imagem e define os }
{ argumentos de endereço das RAMs para a função ReadWriteRAM.       }
{_____________________________________________________________}
   Const
      STARTSTATE = 0;
   Var
      i:          LongInt;
      j, k, l:    Byte;
      Prompt:     Byte;
      State,
      MaxState,
      NumOpFillBuf: LongInt;
      AddrBit,
      AddrByte,
      SizeOfAddrBit,
      LenLFSR:    Byte;
      Width,
```

```
    Heigth:     Integer;
    Buffer:     Array[1..9, 1..7] Of Byte;
Begin
  Prompt := 0;
  State := STARTSTATE;
  Read(TrainingFile, j, k); Width := 256 * j + k;
  Read(TrainingFile, j, k); Heigth := 256 * j + k;
  NumOpFillBuf := Trunc((Trunc((Width +7) / 8) * Heigth +
    SizeOfBus * SizeOfRAMBus -1) / (SizeOfBus * SizeOfRAMBus));
  MaxState := NumOpFillBuf * SizeOfRAMBus -1;
  If (MaxState > 127)
    Then Begin
      i := MaxState ShR 7;
      For j := 7 To 31 Do Begin
        If Odd(i) Then LenLFSR := j;
        i := i ShR 1;
      End;
      Inc(LenLFSR);
    End                                  { LenLFSR = Teto(Log2 MaxState)    }
    Else LenLFSR := 7;
  If (SizeOfRAMBus > 3)
    Then SizeOfAddrBit := 3
         Else SizeOfAddrBit := SizeOfRAMBus;
  For i := 1 To NumOpFillBuf Do Begin
    For j := 1 To SizeOfRAMBus Do Begin   { Enche Buffer.                   }
      For k := 1 To SizeOfBus Do Begin
        Seek(TrainingFile, SizeOfBus * State + k + 3);
        If EOF(TrainingFile)
                        Then Buffer[k, j] := 0
          Else Read(TrainingFile, Buffer[k, j]);
      End;
      LFSR(State, MaxState, LenLFSR);
    End;
    For j := 1 to SizeOfBus Do Begin      { Esvazia o Buffer.               }
      For k := 1 To 8 Do Begin
        AddrBit := 0;                     { Define AddrBit.                 }
        For l := 1 To SizeOfAddrBit Do Begin
          AddrBit := (AddrBit ShL 1) + (Buffer[j, l] And 1);
          Buffer[j, l] := Buffer[j, l] ShR 1;
        End;
        AddrByte := 0;                    { Define AddrByte                 }
        For l := SizeOfAddrBit + 1 To SizeOfRAMBus Do Begin
          AddrByte := (AddrByte ShL 1) + (Buffer[j, l] And 1);
          Buffer[j, l] := Buffer[j, l] ShR 1;
        End;
```

```
            Make := ReadWriteRAM(Mode, Disc, AddrByte, AddrBit, Prompt);
          End;
        End;
    End;
    Write(' '); GotoXY(WhereX-1, WhereY);
End;

Function ReadHex(Var InFile: Text): Byte;
{-------------------------------------------------------------------------}
{ Ler um byte do arquivo InFile em notação hexadecimal, convertendo       }
{ para notação decimal.                                                   }
{-------------------------------------------------------------------------}

Const Map: Array[48 .. 70] Of Byte = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 0,
                                      0, 0, 0, 0, 0, 10, 11, 12, 13, 14, 15);
Var
    Ch: Char;
    Aux: Byte;
Begin
    Repeat
    Read(InFile, Ch);
    Until (UpCase(ch) = 'X');
    Read(InFile, Ch);
    Aux := Map[Ord(UpCase(Ch))];
    Read(InFile, Ch);
    Aux := (Aux ShL 4) + Map[Ord(UpCase(Ch))];
    ReadHex := Aux;
End;

Procedure Find(Var InFile: Text; Patern: String);
{-------------------------------------------------------------------------}
{ Procura pelo padrão Patern no arquivo InFile, posicionando o ponteiro   }
{ do arquivo no carácter imediatamente posterior a ocorencia.             }
{-------------------------------------------------------------------------}

Var
    i:      Byte;
    Ch:     Char;
    Finded: Boolean;
Begin
    Ch := NULL;
    Repeat
        Finded := TRUE;
        While (UpCase(Ch) ≠ Patern[1]) Do
            Read(InFile, Ch);
            i := 2;
                While i ≤ Length(Patern) do Begin
```

```
                Read(InFile, Ch);
                If (UpCase(Ch) ≠ Patern[i]) Then Begin
                   i := Length(Patern);     { Força saída do While }
                   Finded := FALSE;
                End;
                Inc(i);
             End;
      Until (Finded)
End;

Procedure XBitMapToStream(FileName: String);
{-------------------------------------------------------------------------}
{ Cria um arquivo equivalente a FileName no formato interno Stream        }
{-------------------------------------------------------------------------}
   Var
      InFile: Text;
      OutFile: StreamFile;
      Field: String[4];
      i,
      Width,
      Height: Integer;
      Aux:    byte;
Begin
   Assign(InFile, FileName + '.XBM');
   {$I-} Reset(InFile) {$I+};
   IF (IOResult ≠ 0) Then Exit;
   Assign(OutFile, FileName + '.STR');
   {$I-} Rewrite(OutFile) {$I+};
   IF (IOResult ≠ 0) Then Exit;
   Find(InFile, 'WIDTH');
   Read(InFile, Field);
   Val(Field, Width, i);
   Aux := Hi(Width); Write(OutFile, Aux);
   Aux := Lo(width); Write(OutFile, Aux);
   Find(InFile, 'HEIGHT');
   Read(InFile, Field);
   Val(Field, Height, i);
   Aux := Hi(Height); Write(OutFile, Aux);
   Aux := Lo(Height); Write(OutFile, Aux);
   Find(InFile, '{');
   For i := 1 To  Trunc((Width + 7)/8) * Height Do Begin
      Aux := ReadHex(InFile); Write(OutFile, Aux);
   End;
   Close(InFile);
   Close(OutFile);
```

```
End;

Function Stop(Mode: Boolean; Msg: String): Boolean;
{-------------------------------------------------------------------------}
{ Força uma parada até que alguma tecla seja pressionada.                 }
{-------------------------------------------------------------------------}

Begin
   Menssage(Msg + ' (Digite qq tecla ou Esc para sair)');
   Repeat Until(keyPressed);
   If (ReadKey = ESC)
      Then Begin
         If Mode
            Then MakeTreScr(SINGLE)
            Else MakeOpeScr(SINGLE);
         Menssage(MENUMSG);
         MakeMenuScr(DUBBLE);
         Stop := TRUE;
      End
      Else Stop := FALSE;
End;

Procedure Operation(Var Disc:DiscType);
{-------------------------------------------------------------------------}
{ Entrada e saída de dados relativos a função de operação.                }
{-------------------------------------------------------------------------}

   Var
      TC, Ch:    Char;
      StateFile: Boolean;
      InFile:    StreamFile;
   Begin
      MakeMenuScr(SINGLE);
      MakeOpeScr(DUBBLE);
      Repeat
         Menssage('Entre com o nome da imagem a ser reconhecida sem extens~ao');
         InputStr(InFileName, 8, 10, 1, ['0'..'9', 'a'..'z', 'A'..'Z'], [CR], TC);
         StateFile := OpenStreamFile(InFileName, InFile);
         If Not(StateFile) Then Begin
            XBitMapToStream(InFileName);
            StateFile := OpenStreamFile(InFileName, InFile);
            If Not(StateFile) Then
               If Stop(RECOGNITION, 'Erro de I/O em <' + InFileName + '>') Then Exit;
         End;
      Until StateFile;
      Menssage('Operacao ...');
      Disc.SelRAM := Disc.DiscPtr;
```

```
      Disc.RAMCounter := 0;
      Disc.Acc := 0;
      If Not(Make(RECOGNITION, Disc, InFile)) Then Begin
         Menssage('Imagem maior que o discriminador.');
         Delay(1000);
      End;
      Close(InFile);
      Gotoxy(12, 2); Write(Disc.Acc/Disc.RAMCounter:1:4);
      MakeOpeScr(SINGLE);
      Menssage(MENUMSG);
      MakeMenuScr(DUBBLE);
   End;

Procedure InputLongInt(Var LongVar: LongInt; l, x, y: Integer;
                       LS, TS: CharSet; Var TC: Char);
{-------------------------------------------------------------------------}
{ Lê um inteiro longo do console e retorna em LongVar.                    }
{-------------------------------------------------------------------------}
   Var StrVar: String; St: Integer;
   Begin
      Str(LongVar, StrVar);
      InputStr(StrVar, l, x, y, LS, TS, TC);
      If (StrVar ≠ '') Then Val(StrVar, LongVar, St);
   End;

Procedure InputByte(Var ByteVar: Byte; l, x, y: Integer;
                    LS, TS: CharSet; Var TC: Char);
{-------------------------------------------------------------------------}
{ Lê um inteiro tipo byte do console e retorna em LongVar.                }
{-------------------------------------------------------------------------}
   Var StrVar: String; St: Integer;
   Begin
      Str(ByteVar, StrVar);
      InputStr(StrVar, l, x, y, LS, TS, TC);
      If (StrVar ≠ '') Then Val(StrVar, ByteVar, St);
   End;

Procedure Training(Var Disc: DiscType);
{-------------------------------------------------------------------------}
{ Entrada e saída relativa a função de treinamento.                       }
{-------------------------------------------------------------------------}
   Procedure EndTraining;
      Begin
         MakeTreScr(SINGLE);
         Menssage(MENUMSG);
         MakeMenuScr(DUBBLE);
```

```
    End;
  Var
    i:              Integer;
    TC:             Char;
    FileName:       String;
    InMemory:       Boolean;
    SizeOfRAMNew:   Byte;
    SizeOfDiscNew:  Longint;
    SizeOfRAMBusNew: Byte;
    TrainingFile:   StreamFile;

  Begin
    MakeMenuScr(SINGLE);
    Window(42, 10, 78, 10); ClrScr;        { Apaga mensagem de RESET                }
    MakeTreScr(DUBBLE);
    Menssage('Entre com o nome base da(s) imagem(s) de treinamento');
    InputStr(Base, 6, 12, 2, ['a'..'z', 'A'..'Z'], [CR], TC);
    Menssage('Entre com o indice inicial do(s) arquivo(s) de treinamento');
    InputByte(Inicio, 2, 12, 3, ['0'..'9'], [CR], TC);
    Menssage('Entre com o indice final do(s) arquivo(s) de treinamento');
    InputByte(Fim, 2, 12, 4, ['0'..'9'], [CR], TC);
    InMemory := TRUE;
    SizeOfRAMNew  := Disc.SizeOfRam;
    SizeOfDiscNew := Disc.SizeOfDisc;
    SizeOfRAMBusNew := SizeOfRAMBus;
    Repeat
      Gotoxy(10, 8); Write(' ':8);
      Menssage('Entre com o tamanho do barramento (em bytes)');
      InputByte(SizeOfBus, 1, 10, 8, ['0'..'9'], [CR], TC);
      Gotoxy(10, 8); Write((SizeOfBus * 8):3, ' Bits');
      Gotoxy(10, 9); Write(' ':8);
      Menssage('Entre com o tamanho do barramento das RAMs (Max. 7)');
      InputByte(SizeOfRAMBusNew, 1, 10, 9, ['1'..'7'], [CR], TC);
      If (SizeOfRAMBusNew < 4)
        Then SizeOfRAMNew := 1
        Else SizeOfRAMNew := 1 ShL SizeOfRAMBusNew ShR 3;  { X := 2^X/8          }
      Gotoxy(10, 9); Write((1 ShL SizeOfRAMBusNew):3, ' Bits');
      Gotoxy(6, 10); Write(' ':8);
      Menssage('Entre com o n' + #167 + ' de RAMs do Discriminador');
      InputLongInt(SizeOfDiscNew, 7, 6, 10, ['0'..'9'], [CR], TC);
      Gotoxy(6, 10); Write(SizeOfDiscNew:7, ' RAMs');
      IF (Disc.SizeOfDisc * Disc.SizeOfRAM <>                { Verifica se precisa alocar }
          SizeOfDiscNew * SizeOfRAMNew ) Then Begin          { ou desalocar memória       }
        FreeMem(Disc.DiscPtr, Disc.SizeOfDisc * Disc.SizeOfRAM);
        If MaxAvail - 4 < SizeOfDiscNew * SizeOfRAMNew
```

```
            Then Begin
               InMemory := False;
               GetMem(Disc.DiscPtr, Disc.SizeOfDisc * Disc.SizeOfRAM);
               Menssage('Nao ha memoria suficiente ... (Digite qq tecla ou Esc para sair)');
               Repeat Until (KeyPressed);
               If (ReadKey = ESC) Then Begin
                  Gotoxy(10, 9); Write((1 ShL SizeOfRAMBus):3, ' Bits');
                  Gotoxy(6, 10); Write(Disc.SizeOfDisc:7, ' RAMs');
                  EndTraining;
                  Exit;
               End;
            End
            Else Begin
               InMemory := TRUE;
               Disc.SizeOfDisc := SizeOfDiscNew;
               Disc.SizeOfRAM := SizeOfRAMNew;
               GetMem(Disc.DiscPtr, Disc.SizeOfDisc * Disc.SizeOfRAM);
               SizeOfRAMBus := SizeOfRAMBusNew;
               DiscReset(Disc);
               Window(42, 10, 78, 10); ClrScr;          { Apaga mensagem de RESET    }
            End;
         End;                                           { Fim da Verificação         }
      Until (InMemory);
      Menssage('Treinamento ...');
      Window(25, 8, 38, 18);
      For i:= Inicio To Fim Do Begin
         Str(i, FileName); If (i < 10) Then FileName := '0' + FileName;
         FileName := Base + FileName;
         If OpenStreamFile(FileName, TrainingFile)
            Then Close(TrainingFile)
            Else XBitMapToStream(FileName);
         If OpenStreamFile(FileName, TrainingFile)
            Then Begin
               Disc.SelRAM := Disc.DiscPtr;
               Disc.RAMCounter := 0;
               If Make(LEARNE, Disc, TrainingFile)
                  Then Begin
                     Writeln(FileName);
                     Close(TrainingFile);
                  End
                  Else Begin
                     Close(TrainingFile);
                     If Stop(LEARNE, 'Imagem maior que o discriminador.') Then Exit;
                  End;
            End
```

```
        Else Begin
          If Stop(LEARNE, 'Erro de I/O em <' + FileName + '>') Then Exit;
        End;
    End;
    Window(3, 6, 39, 19);
    Gotoxy(2, 13); Write(Disc.OnesAccount/Disc.RAMCounter:1:4);
    EndTraining;
  End;

Procedure MakeSW;
{------------------------------------------------------------------------}
{ Inicializa tela, desenha logotipo DCC, etc.                            }
{------------------------------------------------------------------------}

  Begin;
    Clrscr;
    MakeBox(SINGLE, 1,  1, 80, 24, 'SW - SIMULADOR PARA O WISARD');
    MakeMenssageScr(SINGLE);
    MakeBox(DUBBLE, 41, 15, 54, 20, '');          { Inicio do desenho do ábaco }
    MakeBox(DUBBLE, 41, 15, 49, 20, '');
    Window(41, 15, 79, 20);
    GotoXY( 4, 1); Write(#205, #205);
    GotoXY( 9, 1); Write(#203);
    GotoXY( 9, 6); Write(#202);
    GotoXY( 2, 2); Write(#233,#233,#233,#233,#196,#196,#233);
    GotoXY( 2, 3); Write(#233,#196,#196,#233,#233,#233,#233);
    GotoXY( 2, 4); Write(#233,#233,#233,#233,#196,#196,#233);
    GotoXY( 2, 5); Write(#233,#196,#196,#233,#233,#233,#233);
    GotoXY(10, 2); Write(#196,#196,#233,#233);
    GotoXY(10, 3); Write(#233,#196,#196,#233);
    GotoXY(10, 4); Write(#233,#196,#196,#233);
    GotoXY(10, 5); Write(#233,#196,#196,#233);
    GotoXY(17, 3); Write('DCC - IMECC - UNICAMP');
    GotoXY(23, 5); Write('SEMISH 93');            { Fim do desenho do ábaco }
    MakeShowScr(SINGLE);
    MakeResetScr(SINGLE);
    MakeOpeScr(SINGLE);
    MakeTreScr(SINGLE);
    MakeMenuScr(DUBBLE);
    Menssage(MENUMSG);
  End;

  Var
    Key: Char;

Begin                                  { Program }
  If (ParamCount > 0) Then
```

```
    If (ParamStr(1) = '+b') Then BEEP := #7;
  GetMem(Disc.DiscPtr, Disc.SizeOfDisc * Disc.SizeOfRAM);
  Disc.SelRAM := Disc.DiscPtr;
  MakeSW;
  Repeat
    Repeat Until Keypressed;
    Key := UpCase(ReadKey);
    Case Key OF
      'T': Training(Disc);
      'O': Operation(Disc);
      'R': DiscReset(Disc);
      'V': Show(InFileName, WITHSTOP);
    End;
  Until (Key = ESC);
  Window(1, 1, 80, 25);
  ClrScr;
End.
```

A.3.2 Listagem da Descrição em VHDL do Barramento do PC-AT

```
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;
USE std.textio.ALL;
USE mgc_portable.qsim_relations.ALL;

ENTITY pcbus IS
-  ------------------------------------------------------------------  -
-  Modelo Comportamental do barramento de um PC-AT para as condições  -
-  especificas de teste do sistema para reconhecimento de padreoões de Imagens  -
-  que utiliza redes neurais modeladas em memórias RAMs  -
-  Josemir C. Alexandrino  -
-  ------------------------------------------------------------------  -

  PORT (smemw:   out   qsim_state;
        la:      inout qsim_state_resolved_x_vector(23 DOWNTO 17) BUS;
        master:  in    qsim_state_resolved_x BUS;
        sbhe:    inout qsim_state_resolved_x BUS;
        clk:     out   qsim_state;
        refresh: inout qsim_state_resolved_x BUS;
        osc:     out   qsim_state;
        ws0:     in    qsim_state_resolved_x BUS;
        drq0300: in    qsim_state_resolved_x_vector(3 DOWNTO 0) BUS;
        irq1210: in    qsim_state_resolved_x_vector(12 DOWNTO 10) BUS;
        iocs16:  in    qsim_state_resolved_x BUS;
        irq0703: in    qsim_state_resolved_x_vector(7 DOWNTO 3) BUS;
        memr:    inout qsim_state_resolved_x BUS;
```

```
        bale:    out  qsim_state;
        smemr:   out  qsim_state;
        sa:      inout qsim_state_resolved_x_vector(19 DOWNTO 0) BUS;
        aen:     out  qsim_state;
        iow:     inout qsim_state_resolved_x BUS;
        iochrdy: in   qsim_state_resolved_x BUS;
        ior:     inout qsim_state_resolved_and BUS;
        dack0300: out  qsim_state_vector(3 DOWNTO 0);
        memcs16: in   qsim_state_resolved_x BUS;
        dack0705: out  qsim_state_vector(7 DOWNTO 5);
        sd:      inout qsim_state_resolved_x_vector(15 DOWNTO 0) BUS;
        drq0705: in   qsim_state_resolved_x_vector(7 DOWNTO 5) BUS;
        irq1514: in   qsim_state_resolved_x_vector(15 DOWNTO 14) BUS;
        resetdrv: out  qsim_state;
        tc:      out  qsim_state;
        iochck:  in   qsim_state_resolved_x BUS;
        memw:    inout qsim_state_resolved_x BUS);
END pcbus;

ARCHITECTURE esboco OF pcbus IS
  CONSTANT numdev:    INTEGER := 10;
  CONSTANT numchip:   INTEGER := 3;
  CONSTANT sizeofimage: INTEGER := 128; - em words

  SUBTYPE pcaddrt  IS qsim_state_resolved_x_vector(23 DOWNTO 0);
  SUBTYPE pcdatat  IS qsim_state_resolved_x_vector(15 DOWNTO 0);
  SUBTYPE word     IS bit_vector(15 DOWNTO 0);

  TYPE  memotype IS ARRAY(sizeofimage DOWNTO 0) OF word;
  TYPE  tabtype  IS ARRAY(1 TO numdev) OF INTEGER;

  CONSTANT iodevicesaddr: tabtype := (
    01022, - Chip(0,0) IO Address
    09214, - Chip(0,1) IO Address
    58366, - Chip(0,7) IO Address
    07167, - Lenght of LFSR Register Adrress
    06143, - Size of Bloco Register Address
    01023, - Begin of Image Register 0 Address
    02047, - Begin of Image Register 1 Address
    03071, - Begin of Image Register 2 Address
    04095, - Size of Image Register 0 Address
    05119); - Size of Image Register 1 and Start Bit Address
  CONSTANT iodevicesdata: tabtype := ( - Treinamento
    001, - Chip(0,0) Control Byte (reset_ss_acc = 0, op_ntr = 0, reset_all = 1)
    001, - Chip(0,1) Conttrol Byte
    001, - Chip(0,7) Control Byte
    007, - Lenght of LFSR
```

```
    015,  – Size of Bloco
    000,  – Begin of Image Byte 0
    000,  – Begin of Image Byte 1
    000,  – Begin of Image Byte 2
    002,  – Size of Image Byte 0
    128); – Size of Image Byte 1 and Start Bit
  CONSTANT iodevicesdata2: tabtype := ( – Operação
    006,  – Chip(0,0) Control Byte
    006,  – Chip(0,1) Conttrol Byte
    006,  – Chip(0,7) Control Byte
    007,  – Lenght of LFSR
    015,  – Size of Bloco
    000,  – Begin of Image Byte 0
    000,  – Begin of Image Byte 1
    000,  – Begin of Image Byte 2
    002,  – Size of Image Byte 0
    128); – Size of Image Byte 1 and Start Bit
  CONSTANT ClkHTime:   TIME := 16.67 ns; – Clock High Time
  CONSTANT ClkLTime:   TIME := 33.33 ns; – Clock Low Time
  CONSTANT resethtime: TIME :=  200 ns; – Reset High Time
  CONSTANT resetltime: TIME :=  102 us;
  CONSTANT rshltime:   TIME :=  150 ns;
  CONSTANT rshhtime:   TIME := 14.85 us;
  CONSTANT rshprop:    TIME :=    4 ns;
  CONSTANT dackprop:   TIME :=    4 ns;
  CONSTANT avprop:     TIME :=   22 ns;
  CONSTANT azprop:     TIME :=   43 ns;
  CONSTANT dvprop:     TIME :=   10 ns;
  CONSTANT dzprop:     TIME :=   45 ns;
  CONSTANT bheprop:    TIME :=    4 ns;
  CONSTANT iow0prop:   TIME :=   41 ns;
  CONSTANT iow1prop:   TIME :=   40 ns;
  CONSTANT ior0prop:   TIME :=   41 ns;
  CONSTANT ior1prop:   TIME :=   40 ns;
  CONSTANT memoprop:   TIME :=   50 ns;
  CONSTANT memohold:   TIME :=   15 ns;
  CONSTANT iochrdprop: TIME :=   55 ns;
  CONSTANT activedge:  qsim_state := '0';

  SIGNAL ClkI:      qsim_state;
  SIGNAL dack0705i: qsim_state_vector(7 DOWNTO 5);
  SIGNAL AI:        PCAddrT;
  SIGNAL DI,acc:    PCDataT;
  SIGNAL refreshi:  qsim_state;
  SIGNAL resetdrvi: qsim_state;
  SIGNAL bhei:      qsim_state;
```

```
SIGNAL iowi:    qsim_state;
SIGNAL iori:    qsim_state;
SIGNAL memri:   qsim_state;
SIGNAL cpunoop: BOOLEAN;
SIGNAL cpuw:    BOOLEAN;
SIGNAL memo:    memotype;
SIGNAL sdt:     qsim_state_vector(15 DOWNTO 0);

FUNCTION getmemo(la, sa: qsim_state_resolved_x_vector; memo: memotype)
RETURN qsim_state_resolved_x_vector IS
```

Retorna conteudo de memória endereçado por 'la' e 'sa'

```
  VARIABLE i: INTEGER;
  VARIABLE d: pcdatat;
  BEGIN
    i := to_integer('0' & qsim_state_vector(la(23 DOWNTO 20)) &
              qsim_state_vector(sa(19 DOWNTO 1)) & '0', 0);
    IF i ≤ sizeofimage
      THEN
        d := qsim_state_resolved_x_vector(to_qsim_state(memo(i)));
      ELSE
        d := (OTHERS =>'Z');
    END IF;
    RETURN d;
  END;
BEGIN - ACHITECTURE
  ClkI  <='0' AFTER ClkHTime WHEN (ClkI = '1') ELSE
          '1' AFTER ClkLTime WHEN (ClkI = '0') ELSE
          '0';
  Clk   <=ClkI;
  resetdrvi  <=  '0' AFTER resethtime WHEN (resetdrvi = '1') ELSE
                 '1' AFTER resetltime WHEN (resetdrvi = '0') ELSE
                 '1';
  resetdrv  <= resetdrvi;
  refreshi  <= '0' AFTER rshhtime WHEN (refreshi = '1') AND (resetdrvi = '0') ELSE
               '1' AFTER rshltime WHEN (refreshi = '0') AND (resetdrvi = '0') ELSE
               '1';
  aen  <=    '0';
  dack0705i(7)  <=  NOT(drq0705(7)) AFTER dackprop
                      WHEN cpunoop AND (clki = activedge) AND clki'EVENT ELSE
                    '1' AFTER dackprop WHEN (resetdrvi = '1') ELSE
                    dack0705i(7);
  dack0705i(6)  <=  NOT(drq0705(6)) AFTER dackprop
```

```
                    WHEN cpunoop AND (clki = activedge) AND clki'EVENT AND
                      (drq0705(7) = '0') ELSE
                  '1' AFTER dackprop
                    WHEN (resetdrvi = '1') ELSE
                  dack0705i(6);
  dack0705i(5) <= NOT(drq0705(5)) AFTER dackprop
                    WHEN cpunoop AND (clki = activedge) AND clki'EVENT AND
                      (drq0705(7 DOWNTO 6) = "00") ELSE
                  '1' AFTER dackprop
                    WHEN (resetdrvi = '1') ELSE
                  dack0705i(5);
  dack0705 <=dack0705i;
  bhei <= ai(0) AFTER bheprop
            WHEN cpuw AND (master ='1') ELSE
          NOT(ai(0)) AFTER bheprop
            WHEN NOT(cpuw) AND (master ='1') ELSE
          'Z';

  master_disable: BLOCK (master = '1')
  BEGIN
    refresh <= GUARDED refreshi AFTER rshprop;
    sd <= GUARDED di;
    ior <= GUARDED iori;
    ior <= GUARDED '1';
    iow <= GUARDED iowi;
    memr <=GUARDED '1';
    sa <= GUARDED ai(19 DOWNTO 0);
    la <= GUARDED ai(23 DOWNTO 17);
    sbhe <= GUARDED bhei;
  END BLOCK master_disable;

  read_memory: BLOCK (memr = '0')
  BEGIN
    sd <= GUARDED getmemo(la, sa, memo) AFTER memoprop;
  END BLOCK read_memory;

PROCESS

FUNCTION hexa_to_bit(c: CHARACTER) return bit_vector IS
--  ------------------------------------------------------------------  --
--  Converte um caracter hexadecimal (0 .. 9, a .. f) em bit_vector     --
--  ------------------------------------------------------------------  --

BEGIN
  IF c ≥ '0' AND c ≤ '9' THEN
    RETURN to_bit(CHARACTER'POS(c) - CHARACTER'POS('0'), 4);
  ELSIF c ≥ 'a' AND c ≤ 'f' THEN
    RETURN to_bit(CHARACTER'POS(c) - CHARACTER'POS('a') + 10, 4);
```

```
   ELSE
      RETURN "0000";
   END IF;
END;

PROCEDURE initmemo(SIGNAL mem: OUT memotype) IS
-  ------------------------------------------------------------  -
-        Inicializa "mem" com os pixel da imagem em arquivo      -
-  ------------------------------------------------------------  -

CONSTANT width_of_line: INTEGER := 74;
FILE     input_file: TEXT IS IN "/proj/cadmc/msc/josemir/images/input_image.xbm";
VARIABLE input_line: LINE;
VARIABLE c:          CHARACTER;
VARIABLE c2:         STRING( 1 TO 2);
VARIABLE p:          INTEGER RANGE 0 TO 80 := 1;

BEGIN
   FOR i IN 1 TO 4 LOOP                    - Elimina header
      READLINE(input_file, input_line);
   END LOOP;
   READ(input_line, c);
   FOR i IN mem'range LOOP

      FOR j IN 1 DOWNTO 0 LOOP
         WHILE c /= 'x' AND c /= '}' LOOP
            IF p ≥ width_of_line THEN
               READLINE(input_file, input_line);
               p := 0;
            END IF;
            p := p + 1;
            READ(input_line, c);
         END LOOP;
         IF c = 'x' THEN
            READ(input_line, c);
            mem(i)(j*8+7 DOWNTO j*8+4)  <=  hexa_to_bit(c);
            READ(input_line, c);
            mem(i)(j*8+3 DOWNTO J*8)  <=  hexa_to_bit(c);
            p := p + 2;
         ELSE EXIT;
         END IF;
      END LOOP; - FOR j
   END LOOP;  - FOR i
END;

   BEGIN - PROCESS
      cpunoop  <=FALSE;
      di  <=  TRANSPORT (OTHERS => 'Z');
```

```
ai <= TRANSPORT (OTHERS => 'Z');
iori <= TRANSPORT '1';
iowi <= TRANSPORT '1';
WAIT ON resetdrvi UNTIL (resetdrvi = '0');
initmemo(memo);
- Escrita de IO Devices (treinamento):
cpuw <= TRUE;
FOR i in iodevicesaddr'RANGE LOOP
   WAIT ON ClkI UNTIL (ClkI = activedge);
   ai <= TRANSPORT qsim_state_resolved_x_vector(
            to_qsim_state(iodevicesaddr(i), 24)) AFTER avprop;
   iowi <= TRANSPORT '0' AFTER iow0prop;
   FOR j IN 1 TO 4 LOOP
      WAIT ON clki UNTIL (clki = activedge);
   END LOOP;
   WAIT ON clki UNTIL (clki /= activedge);
   WHILE (iochrdy /= '0') LOOP
      WAIT ON clki UNTIL (clki /= activedge);
   END LOOP;
   WAIT ON ClkI UNTIL (ClkI = activedge);
   di <= TRANSPORT qsim_state_resolved_x_vector(
            to_qsim_state(iodevicesdata(i), 16)) AFTER dvprop;
   iowi <= TRANSPORT '1' AFTER iow1prop;
   di <= TRANSPORT (OTHERS=>'Z') AFTER dzprop;
   ai <= TRANSPORT (OTHERS=>'Z') AFTER azprop;
END LOOP;
cpunoop <=TRUE;
FOR i in 0 TO iodevicesdata(numdev-1) LOOP
   WAIT ON refreshi UNTIL (refreshi = '1') AND (refreshi'LAST_VALUE = '0');
END LOOP;
cpunoop <=FALSE;
initmemo(memo);
- Escrita de IO Devices (operação):
FOR i in iodevicesaddr'RANGE LOOP
   WAIT ON ClkI UNTIL (ClkI = activedge);
   ai <= TRANSPORT qsim_state_resolved_x_vector(
            to_qsim_state(iodevicesaddr(i), 24)) AFTER avprop;
   iowi <= TRANSPORT '0' AFTER iow0prop;
   FOR j IN 1 TO 4 LOOP
      WAIT ON clki UNTIL (clki = activedge);
   END LOOP;
   WAIT ON clki UNTIL (clki /= activedge);
   WHILE (iochrdy /= '0') LOOP
      WAIT ON clki UNTIL (clki /= activedge);
```

```
        END LOOP;
        WAIT ON ClkI UNTIL (ClkI = activedge);
        di <= TRANSPORT qsim_state_resolved_x_vector(
                  to_qsim_state(iodevicesdata2(i), 16)) AFTER dvprop;
        iowi <= TRANSPORT '1' AFTER iow1prop;
        di <= TRANSPORT (OTHERS=>'Z') AFTER dzprop;
        ai <= TRANSPORT (OTHERS=>'Z') AFTER azprop;
      END LOOP;
      cpunoop <=TRUE;
      FOR i in 0 TO iodevicesdata(numdev-1) LOOP
        WAIT ON refreshi UNTIL (refreshi = '1') AND (refreshi'LAST_VALUE = '0');
      END LOOP; - Leitura dos registros acumuladores dos chips:
      acc <= TRANSPORT (OTHERS=>'0');
      cpuw <= FALSE;
      FOR i in iodevicesaddr'LOW TO numchip LOOP
        WAIT ON ClkI UNTIL (ClkI = activedge);
        ai <= TRANSPORT qsim_state_resolved_x_vector(
                to_qsim_state(iodevicesaddr(i), 24)) AFTER avprop;
        iori <= TRANSPORT '0' AFTER ior0prop;
        FOR j IN 1 TO 4 LOOP
          WAIT ON clki UNTIL (clki = activedge);
        END LOOP;
        WAIT ON clki UNTIL (clki /= activedge);
        WHILE (iochrdy /= '0') LOOP
          WAIT ON clki UNTIL (clki /= activedge);
        END LOOP;
        WAIT ON ClkI UNTIL (ClkI = activedge);
        iori <= TRANSPORT '1' AFTER ior1prop;
        di <= TRANSPORT (OTHERS=>'Z') AFTER dzprop;
        ai <= TRANSPORT (OTHERS=>'Z') AFTER azprop;
        WAIT ON iori UNTIL (iori = '1');
        acc <= TRANSPORT qsim_state_resolved_x_vector(to_qsim_state(
                  to_integer('0' & qsim_state_vector(acc), 0) +
                  to_integer('0' & qsim_state_vector(sd), 0), 16));
      END LOOP;
    END PROCESS;
END esboco;
```

A.3.3 Listagem da Descrição em VHDL da PLD de Controle

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

ENTITY cb IS
```

```
PORT(Reset, IOW, IOR, AEn, Refresh, Clock,
     HoldA: IN control_type;
     D:     IN qsim_state_vector(7 DOWNTO 0);      - Data Bus -
     HldR,                                         - Hold Request -
     Master,
     BHE,                                          - Bus High Enable -
     EDB,                                          - Enable Data Bus -
     DDB,                                          - Dir. Data Bus -
     IOChRdy,
     MemR,                                         - Memory Read -
     NW,                                           - Neuron Write -
     ShOut: OUT control_type;                      - Shift Out -
     A:     INOUT qsim_state_vector(23 DOWNTO 0);  - Address Bus -
     CSL,                                          - Chip Select Line -
     CSC:   OUT qsim_state_vector(7 DOWNTO 0));    - Chip Select Col -
END cb;

ARCHITECTURE esboco OF cb IS

CONSTANT hldr_prop:     TIME := 5 ns;
CONSTANT master_prop:   TIME := 5 ns;
CONSTANT MemRProp:      TIME := 5 ns;
CONSTANT bhe_prop:      TIME := 5 ns;
CONSTANT rle_prop:      TIME := 5 ns;
CONSTANT ade_prop:      TIME := 5 ns;
CONSTANT ddb_prop:      TIME := 5 ns;
CONSTANT edb_prop:      TIME := 5 ns;
CONSTANT IOChRdyProp:   TIME := 5 ns;
CONSTANT a_prop:        TIME := 4 ns;
CONSTANT so_prop:       TIME := 5 ns;
CONSTANT nw_prop:       TIME := 5 ns;
CONSTANT CSProp:        TIME := 10 ns;
CONSTANT hold_req_time: TIME := 200 ns;
CONSTANT MaxI:          INTEGER := 55;   - Maximo indice para os Registros Internos -

SUBTYPE BIRI IS INTEGER RANGE 23 DOWNTO 0;   - Begin of Image Range Index -
SUBTYPE SIRI IS INTEGER RANGE 38 DOWNTO 24;  - Size of Image Range Index -

CONSTANT SFRI: INTEGER := 39;                - Start flag Index -

SUBTYPE SBRI IS INTEGER RANGE 47 DOWNTO 40;  - Size of Block Range Index -
SUBTYPE LLRI IS INTEGER RANGE 55 DOWNTO 48;  - Length of LFSR Range Ind. -

CONSTANT BIOA1: INTEGER := 1023;             - Board Input Output Address 1 -
                                             - (registros internos da PLD) -
CONSTANT BIOA2: INTEGER := 1022;             - Board Input Output Address 2 (chips) -

SIGNAL MemRI:   control_type;                - MemR Interno -
SIGNAL IOChRdyI: control_type;               - IOChRdy Interno -
```

```
    - Descrição dos registros internos:                                    -
        - RI(BIRI): Begin of Image Address Register (24 bits)              -
        - RI(SIRI): Size of Image Register (15 bits), em blocos            -
        - RI(SFRI): Start Flag                                             -
        - RI(SBRI): Size of Block Register (8 bits), em conjuntos de 4 words -
        - RI(LLRI): Length of LFSR Register (8 bits) Deve ter sempre um valor <= 23 -
SIGNAL RI:      qsim_state_vector(MaxI DOWNTO 0);
SIGNAL RI1, RI2: qsim_state_vector(MaxI + 1 DOWNTO 0);


BEGIN
  MemR   <= MemRI AFTER MemRProp;
  IOChRdy <= IOChRdyI AFTER IOChRdyProp;
  edb    <= '0' AFTER edb_prop WHEN (MemRI = '0') ELSE
            IOR AND IOW AFTER edb_prop;
  ddb    <= IOR AFTER ddb_prop;
  RI     <= RI1(maxI DOWNTO 0) WHEN NOT(RI1'QUIET) ELSE
            RI2(maxI DOWNTO 0) WHEN NOT(RI2'QUIET) ELSE
            RI;
  PROCESS(IOR, IOW)        - IOChRdy, Chip Select e programacao dos registros internos
-
  BEGIN
    IF RI1(MaxI + 1) = 'X' THEN RI1(MaxI + 1) <= '0'; END IF;
    RI1(MaxI DOWNTO 0) <= RI;
    IF ((to_integer('0' & A(9 DOWNTO 0), 0) = BIOA1) OR
        (to_integer('0' & A(9 DOWNTO 0), 0) = BIOA2)) AND
        (AEn = '0') AND (IOR='0' OR IOW='0')
          THEN IOChRdyI <= '0';
          ELSE IOChRdyI <= 'Z';
    END IF;
    IF (AEn = '0') AND (IOR='0' OR IOW='0') AND
       (to_integer('0' & A(9 DOWNTO 0), 0) = BIOA2)
          THEN
            CSL(to_integer('0' & A(12 DOWNTO 10), 0)) <= '0' AFTER CSProp;
            CSC(to_integer('0' & A(15 DOWNTO 13), 0)) <= '0' AFTER CSProp;
          ELSE
            CSL(i) <= (OTHERS => '1') AFTER CSProp;
            CSC(i) <= (OTHERS => '1') AFTER CSProp;
    END IF;
    IF (IOW'EVENT) AND (IOW = '1') AND
       (to_integer('0' & A( 9 DOWNTO  0), 0)  = BIOA1) AND
       (to_integer('0' & A(15 DOWNTO 10), 0) * 8 + 7  <= MaxI)
          THEN
```

```
            RI1(to_integer('0' & A(15 DOWNTO 10), 0) * 8 + 7 DOWNTO
               to_integer('0' & A(15 DOWNTO 10), 0) * 8) <= D;
            RI1(MaxI + 1) <= NOT(RI1(MaxI + 1));
   END IF;
END PROCESS;

PROCESS(Reset, Refresh, Clock)                          - Maquina de estado      -
VARIABLE State:     INTEGER RANGE 0 TO 10 := 0;
VARIABLE LFSR_State: qsim_state_vector(A'RANGE) := (OTHERS => '0');

VARIABLE HRTO:      INTEGER RANGE 0 TO    8 := 0;  - Hold Request Time Out -
VARIABLE ISF:       INTEGER RANGE 0 TO    4 := 4;  - Input Shift Full      -
VARIABLE SIC:       INTEGER RANGE 0 TO 32768;      - Size of Image Counter -
VARIABLE SBC:       INTEGER RANGE 0 TO   63;       - Size of Block Counter -
BEGIN
   IF RI2(MaxI + 1) = 'X'
      THEN RI2(MaxI + 1) <= '0';
   END IF;
   IF (Reset = '1') OR (Clock'EVENT AND (Clock = '1') AND (state = 0))

      THEN
         State := 0;
         LFSR_State := (OTHERS => '0');
         SIC := to_integer('0' & RI(SIRI), 0);
         HldR  <= '0' AFTER hldr_prop;
         master <= '1' AFTER master_prop;
         nw <= '1' AFTER nw_prop;
         MemRI  <= 'Z';
         bhe   <= 'Z' AFTER bhe_prop;
         A <= (OTHERS =>'Z') AFTER a_prop;
         ShOut  <= '1' AFTER so_prop;
   END IF;
   IF (Reset = '0') AND Clock'EVENT AND (Clock = '1') THEN

      CASE state IS
         WHEN 0 =>
            IF (RI(SFRI) = '1')
               THEN state := 1;
            END IF;
         WHEN 1 =>
            hldr <= '0' AFTER hldr_prop;
            master <= '1' AFTER master_prop;
            MemRI  <= 'Z' AFTER MemRProp;
            bhe  <= 'Z' AFTER bhe_prop;
            A <= (OTHERS => 'Z') AFTER a_prop;
            hrto := 0;                                  - Hold Request Time Out  -
            IF (refresh'LAST_EVENT <= hold_req_time) AND
```

```
        (refresh'LAST_VALUE = '0') THEN state := 2;
    END IF;
WHEN 2 =>
    hldr <= '1' AFTER hldr_prop;
    hrto := hrto + 1;
    IF (holda = '0') THEN
        state := 3;
    ELSIF (hrto = 8) THEN
        state := 1;
    END IF;
WHEN 3 =>
    master <= '0' AFTER master_prop;
    SBC := to_integer('0' & RI(SBRI), 0);
    State := 4;
WHEN 4 =>
    a(0) <= '0' AFTER a_prop;
    lfsr_state := lfsr(poly, lfsr_state);
    A(23 DOWNTO 1) <= lfsr_state(to_integer(RI(LLRI),0) DOWNTO 0) +
                                    RI(BIRI)(23 DOWNTO 1) AFTER a_prop;
    bhe <= '1' AFTER bhe_prop;
    state := 5;
WHEN 5 =>
    MemRI <= '0' AFTER MemRProp;
    nw <= '0' AFTER nw_prop;
    state := 6;
WHEN 6 =>
    state := 7;
WHEN 7 =>
    MemRI <= '1' AFTER MemRProp;
    nw <= '1' AFTER nw_prop;
    isf := isf - 1;
    IF (isf = 0)
        THEN state:= 8;
        ELSE state:= 4;
    END IF;
WHEN 8 =>
    isf := 4;
    sbc := sbc - 1;
    state := 9;
WHEN 9 =>
    ShOut <= '0' AFTER so_prop;
    IF (SBC /= 0)
        THEN state := 4;
        ELSE state := 10;
 END IF;
```

```
        WHEN 10 =>
          SIC := SIC - 1;
          IF (SIC = 0)
            THEN
              state := 0;
              RI2(BIRI) <= RI(BIRI);
              RI2(SIRI) <= RI(SIRI);
              RI2(SFRI) <= '0';
              RI2(SBRI) <= RI(SBRI);
              RI2(LLRI) <= RI(LLRI);
              RI2(MaxI + 1) <= NOT(RI2(MaxI + 1));
            ELSE
              state := 1;
          END IF;
        END CASE;
      END IF;
    END PROCESS;
END;
```

Apêndice B

Especificação do ASIC

B.1 Introdução

O ASIC, especialmente projetado para o reconhecimento de padrões de imagens, é o componente básico do sistema apresentado na segunda parte dessa dissertação. Quando "treinado" sobre uma imagem, ele pode informar através de um índice de semelhança, se uma imagem qualquer foi reconhecida ou o quanto ela é semelhante ao padrão treinado. Sua concepção modular e escalável permite a aplicação em reconhecimento de imagens de diversos tamanhos à um custo proporcional a sua capacidade de processamento. A velocidade de processamento é praticamente determinada pela velocidade de transferência das imagens, possibilitando o uso em aplicações que exigem tempo real de execução. A arquitetura do ASIC, apresentada nos capítulos 4, 5 e resumida na seção B.2, foi concebida de modo independente de protocolos de comunicação com o barramento do sistema, oferecendo maior flexibilidade no projeto do sistema e suas aplicações.

Após uma breve apresentação da arquitetura do ASIC (seção B.2), são listadas as descrições comportamentais em VHDL dos blocos funcionais desta arquitetura (seção B.3) e em seguida são apresentadas as células lógicas utilizadas neste projeto, juntamente com seus esquemáticos e *layouts* (seção B.4). O *layout* do protótipo do ASIC é por fim apresentado na seção B.5.

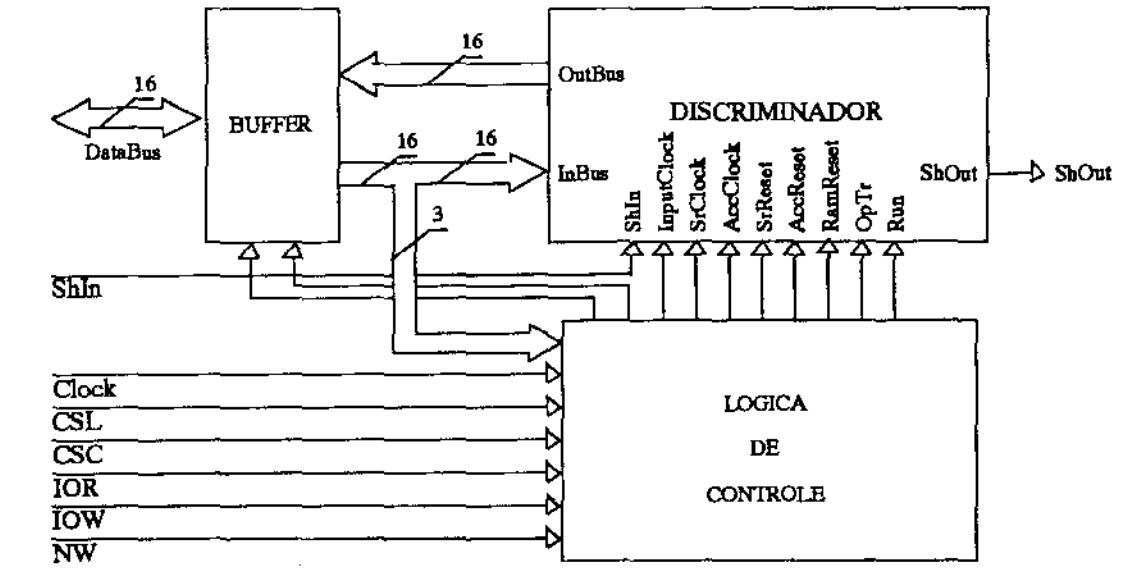
B.2 Arquitetura do ASIC

A arquitetura do ASIC foi apresentada no capítulo 4 e seu funcionamento é baseado no principio de funcionamento do Wisard (seção 3.6.2). A presente descrição enfoca as especificações iniciais para o ASIC e outros detalhes que foram oportunamente omitidos nos capítulos anteriores.

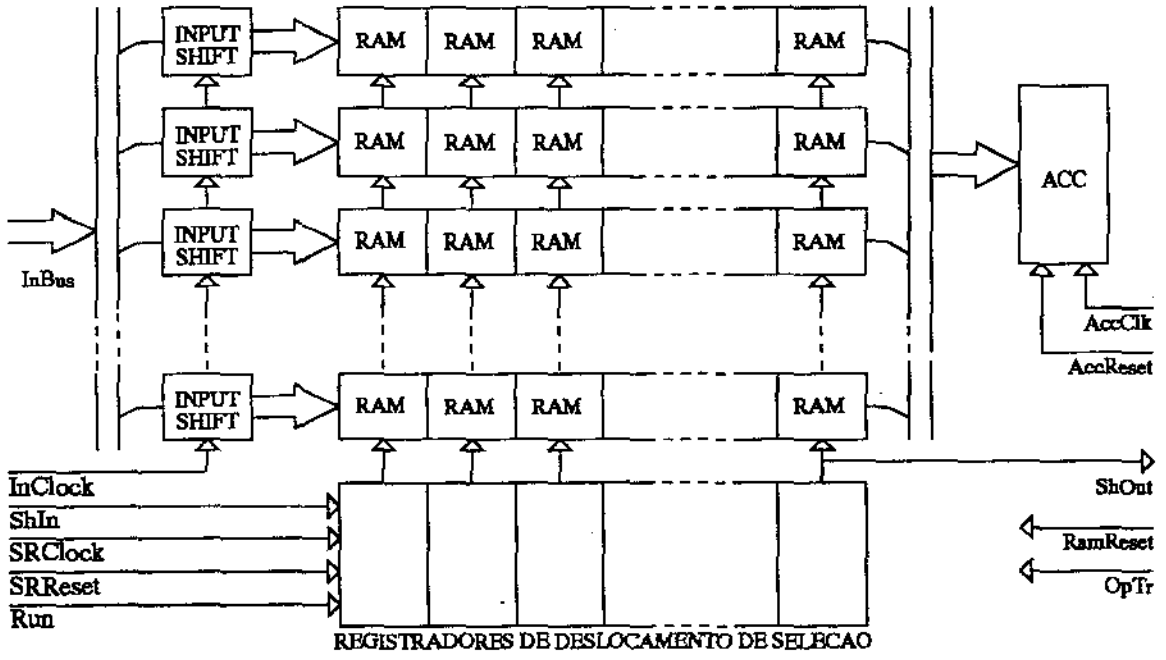
A arquitetura do chip é apresentada na figura B.1a. Ela é composta pelo discriminador, a unidade de controle e o *buffer* de dados. A organização interna do discriminador é vista na figura B.1b.

B.2.1 O Discriminador e a Entrada de Dados

Como pode ser visto na figura B.1b, os endereços das pequenas RAMS do discriminador são formados pelo acumulo de 4 palavras da imagem no registrador de entrada (*Input Shift*) enquanto que a seleção das RAMs de uma dada coluna da matriz é realizada através de um endereçamento implícito seqüencial com o auxílio do registrador de seleção. Este esquema de seleção facilita a interconexão



(a)



(b)

Figura B.1: Arquitetura do ASIC (a) e do discriminador (b).

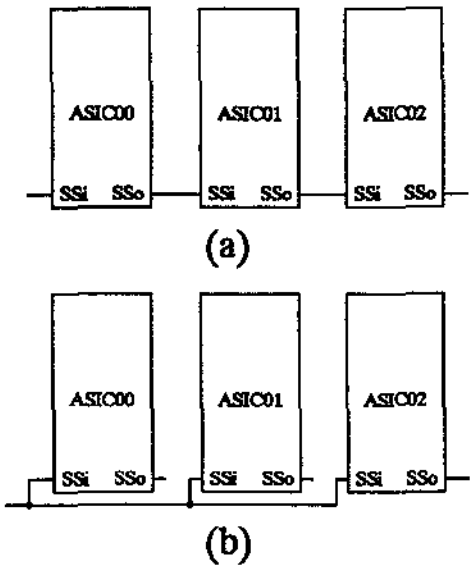


Figura B.2: Esquemas de interconexões entre os ASICS: (a) em série e (b) em paralelo.

de dois ou mais ASICs, ampliando a capacidade de processamento do sistema. A figura B.2 apresenta dois possíveis modos de interconexão entre alguns ASICs. Em (a), eles estão ligados em série, formando um único discriminador com uma capacidade de processamento maior e em (b), eles são ligados em paralelo, formando vários discriminadores com capacidade de processamento igual a de um ASICs cada um. Os ASICs ainda podem ser ligados em uma combinação série/paralelo, podendo compor um sistema com um número variável de discriminadores com diferentes capacidade de processamento.

O ciclo de transferência da imagem (uma palavra) para o discriminador é apresentado na figura B.3. O dado é lido da memória do computador de modo assíncrono (seu sincronismos é com relação a MemR e não ao Clock). O dado é capturado pelos registradores de entrada (*Input Shift*) na borda de descida do último pulso de *clock* quando o sinal NW (*Neuron Write*) deve estar ativo. O dado deve ser mantido estável durante $T_{DS} + T_{DH}$ para assegurar o sucesso da operação. Estes parâmetros são especificados no apêndice A.

B.2.2 O Acumulador e a Saída de Dados

O resultado acumulado no contador (ACC) pode ser lido através de uma operação de leitura de IO. A figura B.4 apresenta o ciclo de leitura de IO para o ASIC. O *chip* coloca o conteúdo do acumulador no barramento de dados quando os sinais IOR, CSL_i, CSC_i forem ativados. Os sinais CSL_i e CSC_i selecionam o ASIC e devem ser gerado da decodificação dos endereços de IO. O dado permanece estável no barramento de dados até que o sinal IOR seja desativado, encerrando o ciclo de leitura de IO.

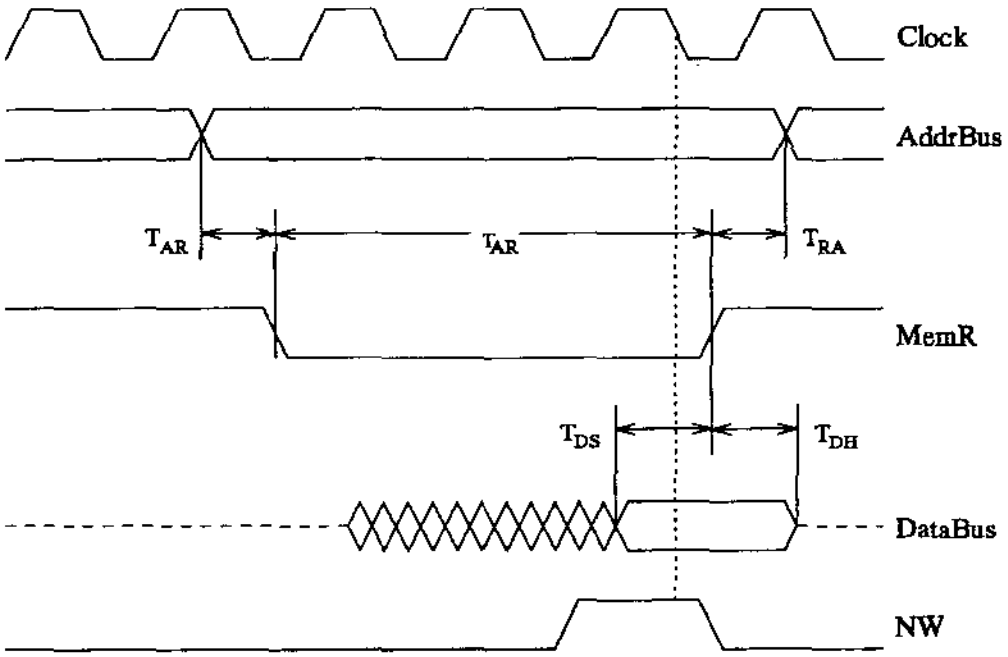


Figura B.3: Ciclo de entrada de dados.

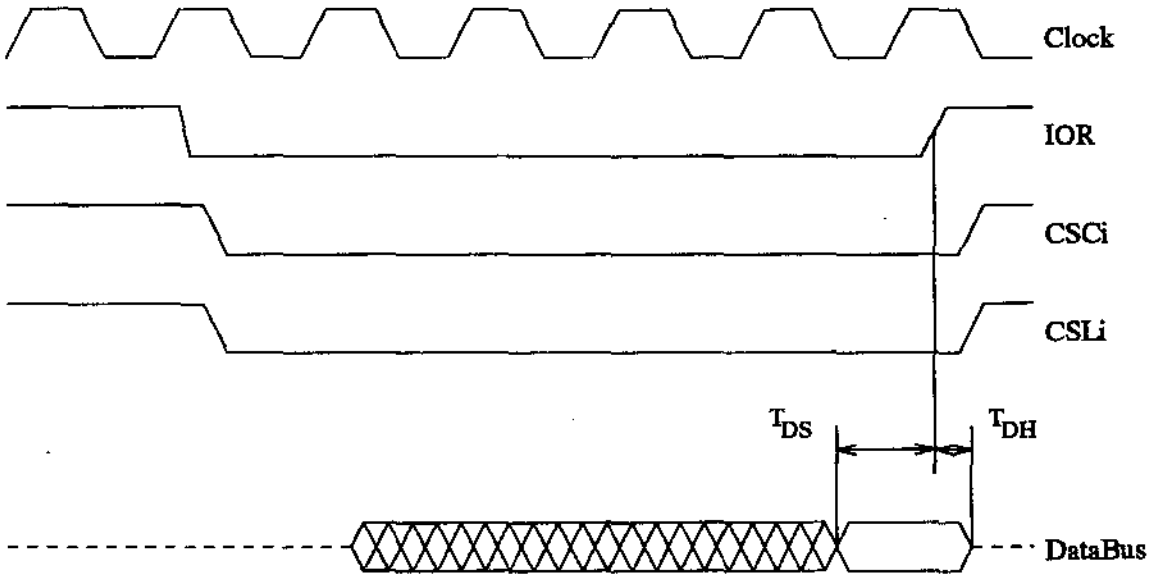


Figura B.4: Ciclo de leitura de IO.

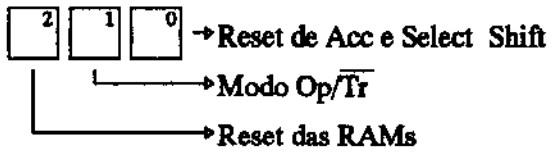


Figura B.5: Registro interno da unidade de controle.

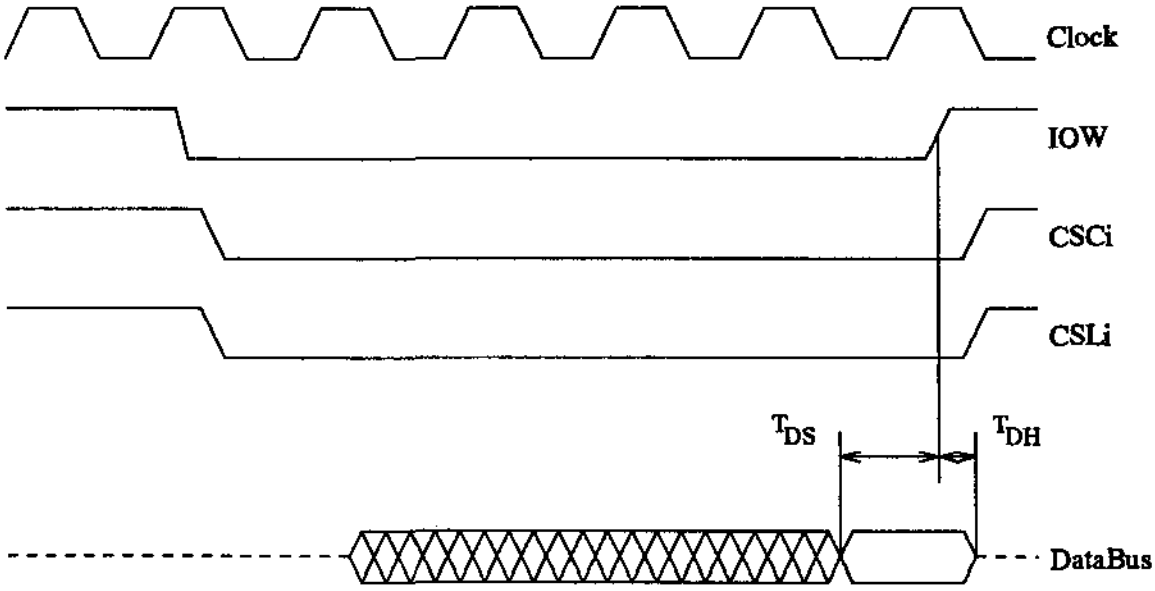


Figura B.6: Ciclo de escrita de IO.

B.2.3 O Registro Interno da Unidade de Controle

Para auxiliar a unidade de controle a inicializar as estruturas do discriminador, existe um registrador interno com 3 *bits* (figura B.5) que pode ser programado através de uma operação de escrita de IO. O *bit* 0 é levado para o nível '1' para indicar que o conteúdo do acumulador e registrador de seleção de RAMs devem ser inicializados com 0 (zero). O *bit* 1, quando ativo, indica a unidade de controle que o *chip* deve funcionar no modo 'operação', caso contrario (*bit* 1 = '0'), no modo 'treinamento'. O *bit* 2 é ativado para indicar que todas as células das RAMs do discriminador devem ser inicializadas com 0.

A figura B.6 apresenta o ciclo de escrita de IO que permite a programação deste registrador. Este ciclo é análogo ao ciclo de leitura, sendo o dado escrito na transição positiva do sinal IOW.

B.3 Descrição Comportamental em VHDL do ASIC

A descrição comportamental do sistema (incluindo o ASIC) possibilitou o dimensionamento de blocos funcionais e auxiliou a definir o relacionamento entre estes blocos. A quantidade de registradores internos na PLD e a capacidade das RAMs do discriminador do ASIC são exemplos

do dimensionamento de tais blocos. Usando os conceitos de abstração e hierarquia, as simulações sobre o modelo comportamental permitiram tal dimensionamento sem o inconveniente nível de detalhamento que, por exemplo, é encontrado em um *netlist* baseados em transistores ou mesmo em portas lógicas. A descrição dos blocos lógicos pertinentes ao ASIC são listadas a seguir. Elas foram referenciadas no capítulo 4, O qual explica estas descrições em linhas gerais.

B.3.1 Listagem da Descrição do Input Shift

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

ENTITY input_shift IS
   PORT(data: IN dio_type;
        clock: IN control_type;
        addr: OUT addr_type);
BEGIN
   clock_timing_check: PROCESS(clock)
      BEGIN
         ASSERT (clock = not(mode_triger)) OR data'STABLE(data_setup)
            REPORT "Setup Time Failure on data"
            SEVERITY WARNING;
         ASSERT (clock = '1') OR clock'DELAYED'STABLE(is_clock_width)
            REPORT "Minimum Pulse Width Failure on clock"
            SEVERITY WARNING;
      END PROCESS clock_timing_check;

   data_time_check: PROCESS(data)
      BEGIN
         ASSERT clock'DELAYED'STABLE(data_hold) OR
                (clock = '0')
            REPORT "Hold Timing Failure on data"
            SEVERITY WARNING;
      END PROCESS data_time_check;
END input_shift;

ARCHITECTURE esboco OF input_shift IS
BEGIN
   PROCESS(clock)
      VARIABLE addr_cp: addr_type;        - Copia de addr p/ uso interno.
   BEGIN
      IF clock = mode_triger THEN
         addr_cp(1) := addr_cp(2);
         addr_cp(2) := addr_cp(3);
         addr_cp(3) := addr_cp(4);
         addr_cp(4) := data;
         addr <= addr_cp AFTER input_shift_delay;
```

```
      END IF;
   END PROCESS;
END esboco;
```

B.3.2 Listagem da Descrição do Select Shift

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

ENTITY select_shift IS
   PORT(clock: IN control_type;
        reset: IN control_type;
        sh_in: IN control_type;
        sel:  OUT select_type);
   BEGIN
      reset_check_timing: PROCESS(reset)
         BEGIN
            ASSERT (reset = '1') OR reset'DELAYED'STABLE(op_reset_width)
               REPORT "Minimum Pulse Width Failure on reset"
               SEVERITY WARNING;
         END PROCESS reset_check_timing;
      clock_check_timing: PROCESS(clock)
         BEGIN
            ASSERT (clock = '1') OR clock'DELAYED'STABLE(sr_clock_width)
               REPORT "Minimum Pulse Width Failure on clock"
               SEVERITY WARNING;
            ASSERT (clock ='0') OR sh_in'STABLE(sh_in_setup)
            REPORT "Setup Time Failure on sh_in"
            SEVERITY WARNING;
         END PROCESS clock_check_timing;
      sh_in_check_timing: PROCESS(sh_in)
         BEGIN
            ASSERT clock'DELAYED'STABLE(sh_in_hold) OR (clock = '0')
               REPORT "Hold Timing Failure on sh_in"
               SEVERITY WARNING;
         END PROCESS sh_in_check_timing;
   END select_shift;

ARCHITECTURE esboco OF select_shift IS
BEGIN
   PROCESS(clock, reset)
      VARIABLE wheres:  integer RANGE 0 TO sel'RIGHT := sel'RIGHT;
      VARIABLE sh_in_cp: control_type;      - Copia de shin de uso interno -
   BEGIN
      IF NOT(reset'STABLE) AND (reset = '1') THEN
```

```
        sel <= (OTHERS => sel_off) AFTER reset_sel_prop;
        wheres := sel'RIGHT;
      END IF;
      IF (reset = '0') AND NOT(clock'STABLE) THEN
        IF clock = '1' THEN
          IF wheres = sel'RIGHT THEN
            sh_in_cp := sh_in;
          END IF;
          IF (wheres ≥ sel'LEFT) AND (wheres <= sel'RIGHT) THEN
            sel(wheres) <= sh_in_cp AFTER clock_rise_prop;
            wheres := wheres - 1;
          END IF;
        END IF;
        IF (clock = '0') AND (wheres ≥ sel'LEFT - 1) AND (wheres <= sel'RIGHT - 1)
THEN
          sel(wheres + 1) <= sel_off AFTER clock_fail_prop;
        END IF;
      END IF;
    END PROCESS;
END esboco;
```

B.3.3 Listagem da Descrição das RAMs

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

ENTITY ram_mxn IS
  PORT(addr:  IN   addr_type;        - Address bus (0 .. log2n - 1)        -
       sel:   IN   select_type;      - Ram select (0 .. m-1)               -
       dio:   INOUT dio_type;        - Data bus                            -
       reset: IN   control_type;
       op_ntr: IN  control_type);    - Operacao/treinamento                -
  BEGIN
    addr_timing_check: PROCESS(addr)
      BEGIN
        ASSERT sel'DELAYED'STABLE(sel_setup) OR
            (bcd_to_int(sel) /= unselect)
          REPORT "Setup Time Failure on sel"
          SEVERITY WARNING;
        ASSERT (bcd_to_int(sel) = unselect)
          REPORT "addr not stable on actived select"
          SEVERITY WARNING;
      END PROCESS addr_timing_check;
    sel_timing_check: PROCESS(sel)
```

```
        BEGIN
          ASSERT (bcd_to_int(sel) /= unselect) OR sel'DELAYED'STABLE(sel_width)
            REPORT "Minimum Pulse Width Failure on sel"
            SEVERITY WARNING;
          ASSERT (bcd_to_int(sel) = unselect) OR addr'STABLE(addr_setup)
            REPORT "Setup Time Failure on addr"
            SEVERITY WARNING;
        END PROCESS sel_timing_check;
      reset_timing_check: PROCESS(reset)
        BEGIN
          ASSERT (reset = '1') OR reset'DELAYED'STABLE(ram_reset_width)
            REPORT "Minimum Pulse Width Failure on reset"
            SEVERITY WARNING;
        END PROCESS reset_timing_check;
END ram_mxn;

ARCHITECTURE esboco OF ram_mxn IS
BEGIN
   PROCESS(addr, sel, reset, op_ntr)
      VARIABLE memory: mem_type;
      VARIABLE s: integer RANGE unselect TO sel'RIGHT;

   BEGIN
      IF NOT(reset'STABLE) AND (reset = '1') THEN
         dio <= '0' AFTER reset_dio_prop;
         FOR i IN memory'RANGE LOOP
            memory(i) := '0';
         END LOOP;
      END IF;
      IF reset = '0' THEN
         IF (op_ntr = '0') THEN
            dio <= '1' AFTER trein_dio_prop;
         END IF;
         s := bcd_to_int(sel);
         IF (s /= unselect) THEN
            IF op_ntr = '0' THEN
               memory(to_int(addr) + 16 * s) := '1';
            ELSIF op_ntr = '1' THEN
               dio <= to_qsim_state(memory(to_int(addr) + 16 * s))
                        AFTER read_delay;
            END IF;
         END IF;
         IF (s = unselect) AND (op_ntr = '1') THEN
            dio <= 'Z';
         END IF;
```

```
    END IF;
  END PROCESS;
END esboco;
```

B.3.4 Listagem da Descrição do Acumulador

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

ENTITY accumulator IS
  PORT(data_bus:  IN  qsim_state_vector(num_bank-1 DOWNTO 0);
       clock:     IN  control_type;
       reset:     IN  control_type;
       result_bus: OUT qsim_state_vector(num_bank-1 DOWNTO 0));
  BEGIN
    data_bus_timing_check: PROCESS(data_bus)
      BEGIN
        ASSERT clock'DELAYED'STABLE(data_bus_hold) OR (clock = '0')
          REPORT "Hold Timing Failure on data_bus"
          SEVERITY WARNING;
      END PROCESS data_bus_timing_check;
    clock_timing_check: PROCESS(clock)
      BEGIN
        ASSERT (clock = not(acc_mode_triger)) OR data_bus'STABLE(data_bus_setup)
          REPORT "Setup Time Failure on data_bus"
          SEVERITY WARNING;
        ASSERT (clock = '1') OR clock'DELAYED'STABLE(acc_clock_width)
          REPORT "Minimum Pulse Width Failure on clock"
          SEVERITY WARNING;
      END PROCESS clock_timing_check;
    reset_timing_check: PROCESS(reset)
      BEGIN
        ASSERT (reset = '1') OR reset'DELAYED'STABLE(acc_reset_width)
          REPORT "Minimum Pulse Width Failure on reset"
          SEVERITY WARNING;
      END PROCESS clock_timing_check;
  END accumulator;

ARCHITECTURE esboco OF accumulator IS
BEGIN
  PROCESS(clock, reset)
    VARIABLE result_int: qsim_state_vector(num_bank-1 DOWNTO 0);
  BEGIN
    IF NOT(reset'STABLE) AND (reset = '1') THEN
```

```
        result_int := (OTHERS => '0');
        result_bus <= result_int AFTER reset_result_prop;
      END IF;
      IF (reset = '0') AND (clock = acc_mode_triger) AND NOT(clock'STABLE) THEN
        result_int := result_int + to_qsim_state(cont_1(data_bus), num_bank);
        result_bus <= result_int AFTER clock_result_prop;
      END IF;
    END PROCESS;
END esboco;
```

B.3.5 Listagem da Descrição da Unidade de controle

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

ENTITY control_unit IS
   PORT(data:    IN qsim_state_vector(num_bank-1 DOWNTO 0);
        nnw:     IN control_type;        - Neuronio Write                    -
        ncs:     IN control_type;        - Chip Select                       -
        nior:    IN control_type;        - I/O Read                          -
        niow:    IN control_type;        - I/O Write                         -
        sh_out:  IN control_type;
        sh_in:   IN control_type;
        clock:   IN control_type;
        buf_dir: OUT control_type;       - Buffer Direction ('0': input; '1': output)  -
        buf_ena: OUT control_type;       - Buffer Enable ('0': 3-State; '1': Active)   -
        in_clk:  OUT control_type;       - Input clock                       -
        sr_clk:  OUT control_type;       - SR Clock                          -
        acc_clk: OUT control_type;       - Acc Clock                         -
        ram_rst: OUT control_type;       - Ram reset                         -
        sr_rst:  OUT control_type;       - Sr Reset                          -
        acc_rst: OUT control_type;       - Acc Reset                         -
        op_ntr:  OUT control_type);      - Operacao/Treinamento              -
BEGIN
END control_unit;

ARCHITECTURE esboco OF control_unit IS
SIGNAL int_op_ntr: control_type          - Uso interno do op_ntr             -
SIGNAL sr_rst_in:  control_type;
SIGNAL ff_ss:      control_type;
CONSTANT ff_prop: TIME := 5 ns;
BEGIN

   op_ntr <= int_op_ntr;
   sr_rst <= sr_rst_in;
   in_clk <= nnw AFTER in_clk_prop;
```

```
  buf_dir <= NOT(nior OR ncs) AFTER dir_prop;
  buf_ena <= nnw AND (ncs OR nior) AND (ncs OR niow) AFTER ena_prop;
  ff_ss <=    '0' AFTER ff_prop
                  WHEN (sh_out = '0') AND sh_out'EVENT
                    AND (sh_out'LAST_VALUE = '1')ELSE

              '1' AFTER ff_prop
                  WHEN sh_in  = '1' AND sh_in'EVENT  ELSE
              ff_ss;

  PROCESS(niow)                          - Programação reg. de controle:      -
    VARIABLE control_reg:                - Bit 0: RAM, shift_sel. e acc. Reset -
      qsim_state_vector(2 DOWNTO 0);     - Bit 1: op_ntr mod                  -
  BEGIN                                  - Bit 2: shift_sel. e acc Reset      -
    IF (ncs = '0') AND (niow = '1') THEN
      control_reg := data(control_reg'RANGE);

      int_op_ntr ≤ control_reg(1) AFTER op_ntr_prop;
      IF control_reg(0) = '1' THEN
        ram_rst ≤ TRANSPORT '1' AFTER ram_rst_prop;
        sr_rst_in ≤  TRANSPORT '1' AFTER sr_rst_prop;
        acc_rst ≤ TRANSPORT '1' AFTER acc_rst_prop;
        ram_rst ≤ TRANSPORT '0' AFTER ram_rst_prop + ram_reset_width;
        sr_rst_in ≤  TRANSPORT '0' AFTER sr_rst_prop  + op_reset_width;
        acc_rst ≤ TRANSPORT '0' AFTER acc_rst_prop + op_reset_width;
      ELSIF (control_reg(2) = '1') THEN
        sr_rst_in ≤  TRANSPORT '1' AFTER sr_rst_prop;
        acc_rst ≤ TRANSPORT '1' AFTER acc_rst_prop;
        sr_rst_in ≤  TRANSPORT '0' AFTER sr_rst_prop  + op_reset_width;
        acc_rst ≤ TRANSPORT '0' AFTER acc_rst_prop + op_reset_width;
      END IF;
    END IF;
  END PROCESS;

PROCESS(nnw, clock)
VARIABLE ibf: INTEGER RANGE 0 TO 5 := 0;
BEGIN
  IF nnw'EVENT THEN
    IF (nnw = '1') AND (nnw'LAST_VALUE = '0') THEN

      IF ibf = 5
        THEN ibf := 1;
        ELSE ibf := ibf + 1;
      END IF;
    ELSIF nnw = '0' AND (ibf = 4) THEN
      ibf := 5;
      acc_clk ≤ ('1' AND ff_ss) AFTER sr_clk_prop;
    END IF;
```

```
    END IF;
    IF (clock = '1') AND clock'EVENT THEN
      CASE ibf IS
        WHEN 2 => acc_clk <= ('0' AND ff_ss) AFTER acc_clk_prop;
        WHEN 4 => sr_clk <= '1' AFTER sr_clk_prop;
        WHEN 5 => sr_clk <= '0' AFTER sr_clk_prop;
        WHEN OTHERS => NULL;
      END CASE;
    END IF;
END PROCESS;

END esboco;
```

B.4 Características Elétricas e Físicas das Células

Como complemento dos capítulos 5 e 6 são apresentadas a seguir as fichas técnicas contendo informações a cerca do projeto elétrico e físico das células lógicas usadas no projeto do ASIC.

Projeto Elétrico/Físico da Célula Decoder00

Descrição: Decodificador de endereços com reset

Autor: Josemir/Lanna Data: Dez/93

Dimensões: 34 X 61 um^2

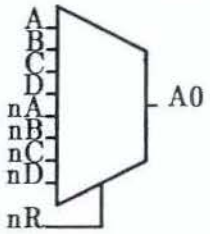
Descrição funcional

$A0 = \overline{A.B.C.D} + \overline{nR}$

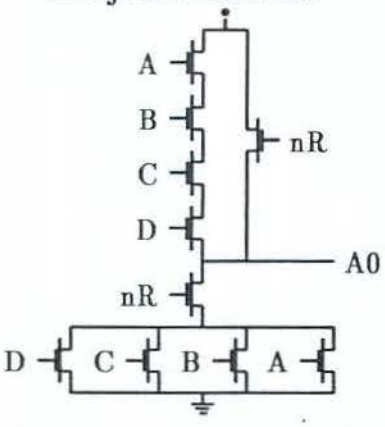
Capacitância de entrada

$C_{nR} = 16$ fF
$C_A = 14$ fF
$C_B = 14$ fF
$C_C = 14$ fF
$C_D = 14$ fF

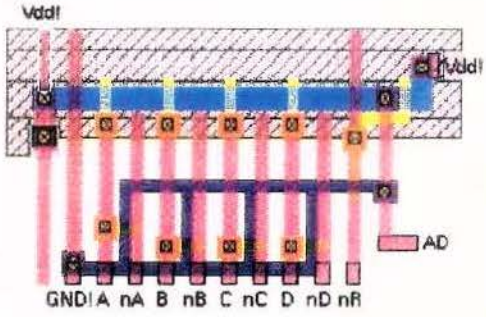
Símbolo lógico



Projeto Elétrico



Projeto Físico



Características

Tempo de propagação quando nR é ativado = 1.33 ns
Tempo de propagação quando nR é desativado = 1.65 ns
Tempo de propagação na decodificação (A0 = 1) = 4.28 ns
Tempo de propagação na decodificação (A0 = 0) = 1.48 ns

OBS: Carga na saída A0 = 100 fF

Projeto Elétrico/Físico da Célula Drive

Descrição: Drive para *buffering* de barramento

Autor: Josemir/Lanna Data: Dez/93

Dimensões: 34 X 20 um^2

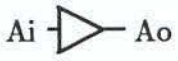
Descrição funcional

$Ao = \overline{Ai}$

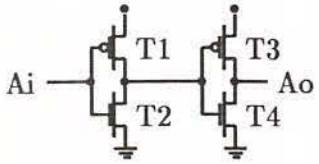
Capacitância de entrada

$C_{Ai} = 13fF$

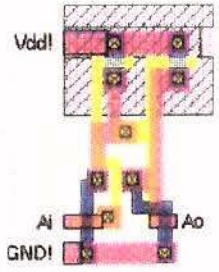
Símbolo lógico



Projeto Elétrico



Projeto Físico



Características

Tempo de propagação (subida de Y) = 1.71 ns
Tempo de propagação (descida de Y) = 1.60 ns
Tempo de subida = 2.65 ns
Tempo de descida = 1.65 ns

Obs: carga na saída Ao = 100 fF

Projeto Elétrico/Físico da Célula FFD

Descrição: Flip-Flop tipo D sensível a borda de descida do clock com load

Autor: Josemir/Lanna Data: Dez/93

Dimensões: 34 X 60 um^2

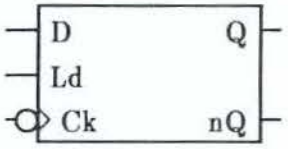
Descrição funcional

Ld	Ck	Q	nQ
1	↓	D	nD
0	X	Q(t-1)	nQ(t-1)

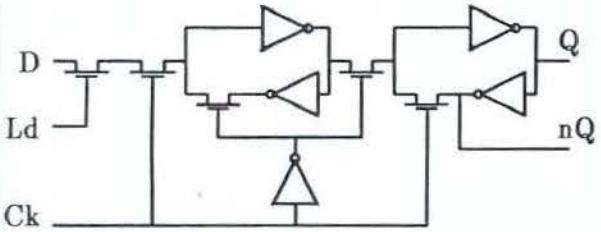
Capacitância de entrada

$C_{Ck} = 26fF$
$C_{Ld} = 0fF$
$C_d = 0fF$

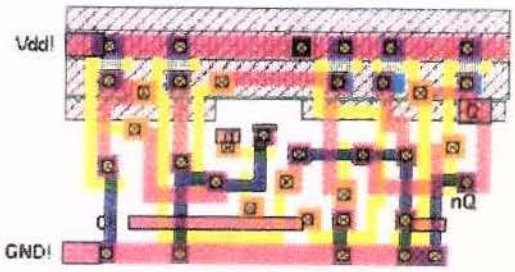
Símbolo lógico



Projeto Elétrico



Projeto Físico



Características

Tempo de propagação com relação a Ck (D = 0) = 2.91 ns
Tempo de propagação com relação a Ck (D = 1) = 2.28 ns
Idem para nQ (D = 0) = 5.5 ns
Idem para nQ (D = 1) = 7.06 ns

OBS: Carga nas saídas Q e nQ = 100 fF

Projeto Elétrico/Físico da Célula Inv

Descrição: Inversor comum

Autor: Josemir/Lanna Data: Dez/93

Dimensões: 48 X 29 um^2

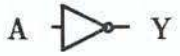
Descrição funcional

$Y = \overline{A}$

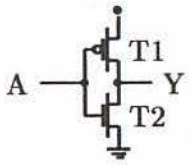
Capacitância de entrada

$C_A = 18$ fF

Símbolo lógico



Projeto Elétrico



Projeto Físico

Características

Tempo de propagação (subida de Y) = 1.04 ns
Tempo de propagação (descida de Y) = 0.78 ns
Tempo de subida = 1.56 ns
Tempo de descida = 1.26 ns

Projeto Elétrico/Físico da Célula Inv3St

Descrição: Inversor 3-state

Autor: Josemir/Lanna Data: Dez/93

Dimensões: 46 x 29 um^2

Descrição funcional

Op	nSS	Rs	Di
0	0	0	1
X	X	1	0
0	X	1	Z
X	0	1	Z

nSS	SS	De
0	1	nDi
1	0	Z

Capacitância de entrada

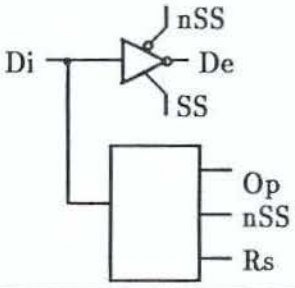
$C_D = 28$ fF

$C_{SS} = 16$ fF

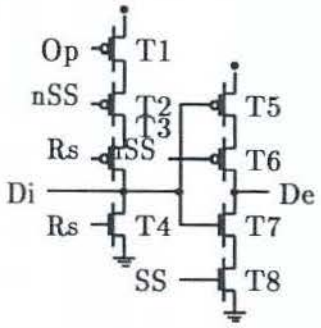
$C_{nSS} = 24$ fF

$C_{Rs} = 13$ fF

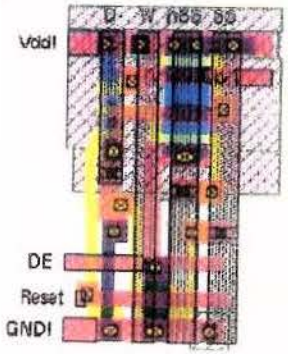
Símbolo lógico



Projeto Elétrico



Projeto Físico



Características

Tempo de propagação quando Rs é ativado = 1.37 ns ($C_{Di} = 111$ fF)

Tempo de propagação quando Rs é desativado = 8.68 ns

Tempo de propagação entre Di e De com relação a nSS = 3.53 ns ($C_{De} = 212$ fF)

Tempo de propagação entre Di e De com relação a SS = 4.48 ns

Projeto Elétrico/Físico da Célula nAnd

Descrição: Porta lógica nAnd comum

Autor: Josemir/Lanna Data: Dez/93

Dimensões: 35 X 25 um^2

Descrição funcional

$Y = \overline{A \cdot B}$

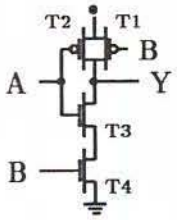
Capacitância de entrada

$C_{Qss} = 20$ fF

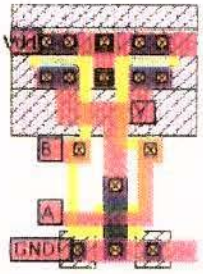
$C_{EnSS} = 21$ fF

Símbolo lógico

Projeto Elétrico



Projeto Físico



Características

Tempo de propagação em relação a A (subida de Y) = 1.07 ns
Tempo de propagação em relação a A (descida de Y) = 0.90 ns
Tempo de propagação em relação a B (subida de Y) = 1.07 ns
Tempo de propagação em relação a B (descida de Y) = 0.81 ns

OBS: Carga em Y = 100 fF

Projeto Elétrico/Físico da Célula nAndInv

Descrição: Gera sinais de seleção SS e nSS

Autor: Josemir/Lanna Data: Dez/93

Dimensões: 37.5 X 25.5 um^2

Descrição funcional

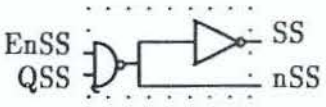
EnSS	SS	nSS
0	0	1
1	Qss	$\overline{Qss}$

Capacitância de entrada

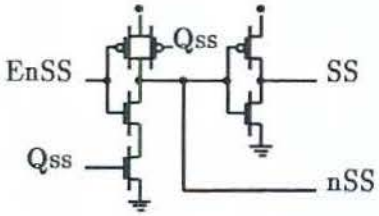
$C_{Qss} = 20$ fF

$C_{EnSS} = 21$ fF

Símbolo lógico



Projeto Elétrico



Projeto Físico

Características

Tempo de propagação entre EnSS e nSS na descida de nSS = 1.23 ns

Tempo de propagação entre EnSS e nSS na subida de nSS = 11.25 ns

Tempo de propagação entre EnSS e SS na descida de SS = 2.04 ns

Tempo de propagação entre EnSS e SS na subida de SS = 2.48 ns

Obs: carga nas saídas SS e nSS = 100 fF

Projeto Elétrico/Físico da Célula Nor

Descrição: Porta lógica nor de 2 entradas

Autor: Josemir/Lanna Data: Dez/93

Dimensões: 35 X 17 um^2

Descrição funcional

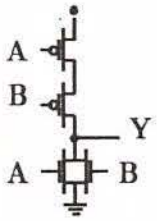
$Y = \overline{A + B}$

Capacitância de entrada

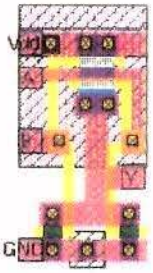
$C_A = 21fF$
$C_B = 19fF$

Símbolo lógico

Projeto Elétrico



Projeto Físico



Características

Tempo de propagação em relação a A (subida de Y) = 1.59 ns
Tempo de propagação com relação a A (descida de Y) = 0.77 ns
Tempo de propagação com relação a B (subida de Y) = 0.1.50 ns
Tempo de propagação com relação a B (descida de Y) = 0.77 ns

OBS: Carga na saída Y = 100 fF

Projeto Elétrico/Físico da Célula Nor3

Descrição: Porta lógica nor de 3 entradas

Autor: Josemir/Lanna Data: Dez/93

Dimensões: 35 X 17 um^2

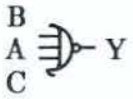
Descrição funcional

$$Y = \overline{A + B + C}$$

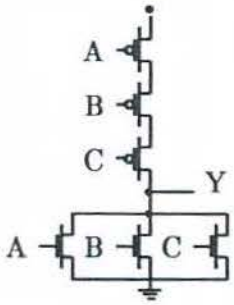
Capacitância de entrada

$C_A = 24fF$
$C_B = 12fF$
$C_C = 20fF$

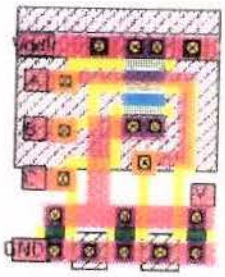
Símbolo lógico



Projeto Elétrico



Projeto Físico



Características

Tempo de propagação em relação a A (subida de Y) = 1.88 ns
Tempo de propagação com relação a A (descida de Y) = 0.81 ns
Tempo de propagação com relação a B (subida de Y) = 1.96 ns
Tempo de propagação com relação a B (descida de Y) = 0.82 ns
Tempo de propagação com relação a C (subida de Y) = 0.1.94 ns
Tempo de propagação com relação a C (descida de Y) = 0.78 ns
OBS: Carga na saída Y = 100 fF

Projeto Elétrico/Físico da Célula RAM1bit

Descrição: Célula de memória est'atica

Autor: Josemir/Lanna Data: Dez/93

Dimensões: 35 x 23 um^2

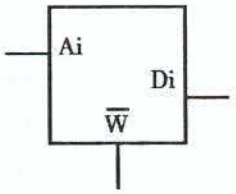
Descrição funcional

Ai	W	Di	Dl
1	1	Dl	X
1	⊔	X	Di
0	1	X	X

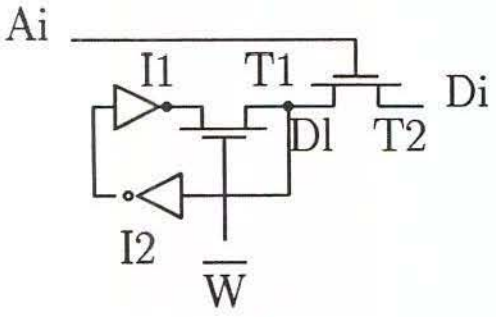
Capacitância de entrada

C_W = 10 fF
C_A = 8 fF

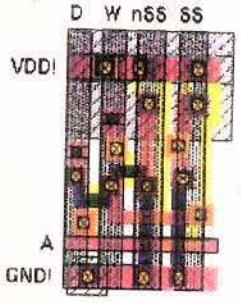
Símbolo lógico



Projeto Elétrico



Projeto Físico



Características

Tempo de leitura com relação a Ai (bit 0) = 1.90 ns
Tempo de leitura com relação a Ai (bit 1) = 9.01 ns
Tempo de escrita com relação a Ai (bit 0) = 0.23 ns
Tempo de escrita com relação a Ai (bit 1) = 0.22 ns

Projeto Elétrico/Físico da Célula WGen

Descrição: Célula geradora do sinal nW (nWrite)

Autor: Josemir/Lanna Data: Dez/93

Dimensões: 39 X 21 um^2

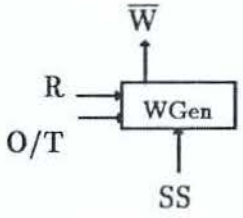
Descrição funcional

R	O/T	SS	$\overline{W}$	Obs:
0	0	0	1	—
0	0	1	0	Treinamento
0	1	X	1	Operação
1	X	X	0	Reset

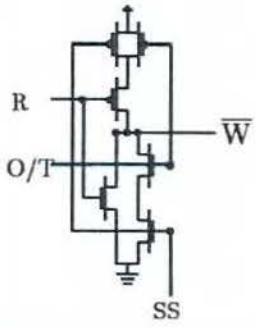
Capacitância de entrada

$C_R = 15fF$
$C_{SS} = 12fF$
$C_{O/T} = 17fF$

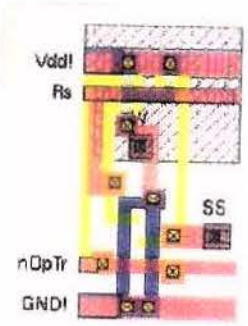
Símbolo lógico



Projeto Elétrico



Projeto Físico

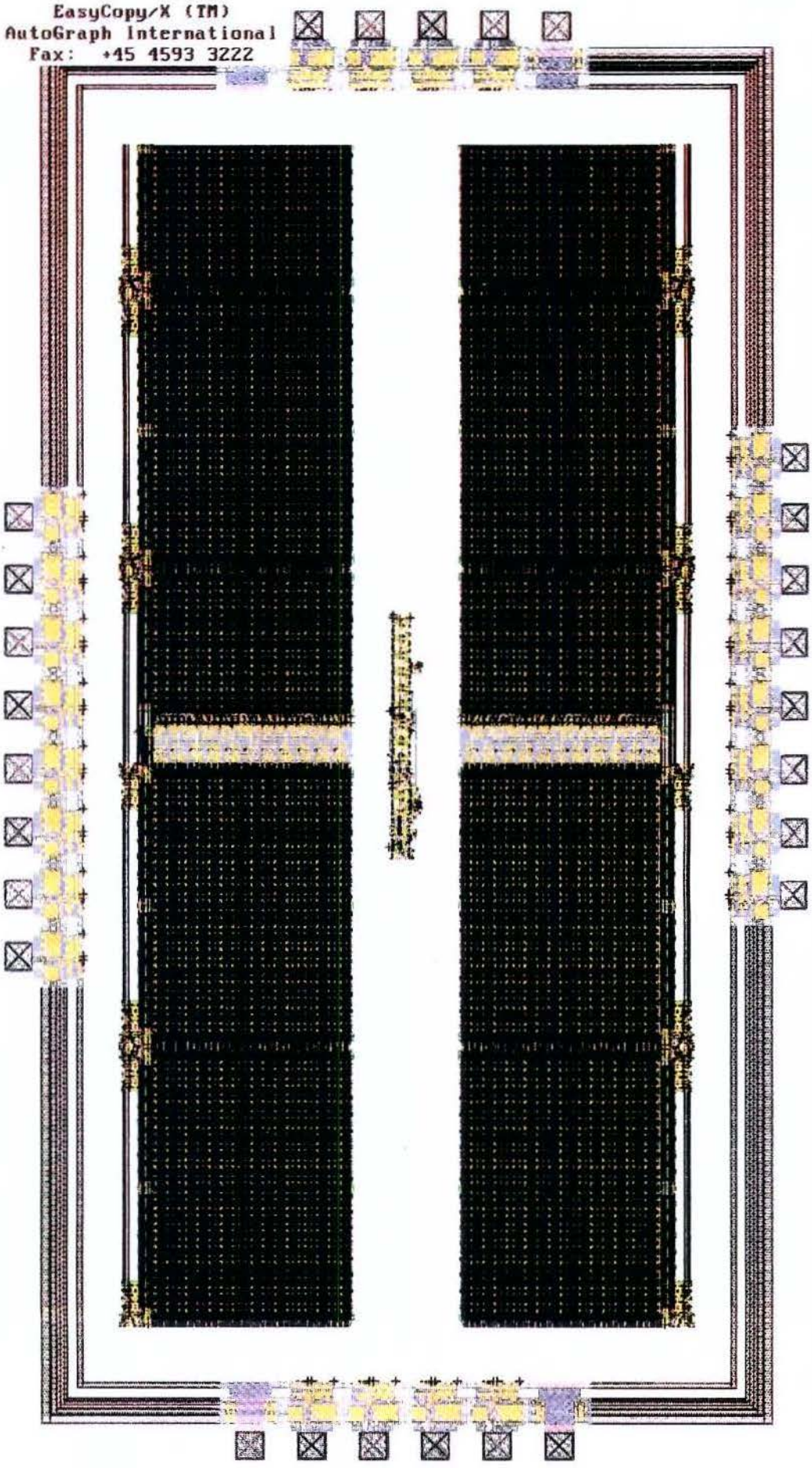


Características

Tempo de propagação quando R é ativado 10.89 ns ($C_W = 1500$ fF)
Tempo de propagação quando R é desativado = 27.25 ns
Tempo de propagação quando SS é ativado = 17.26 ns
Tempo de propagação quando SS é desativado = 43.83 ns

B.5 Layout do Protótipo do ASIC

O *layout* do protótipo do ASIC é apresentado a seguir. Ele ocupa uma área de aproximadamente 14 mm^2 e contem cerca de 61 mil transistores, resultando numa densidade de 230 um^2 por transistores para o ASIC como um todo e 53 um^2 por transistores para o discriminador. Outras considerações a respeito do projeto físico do ASIC podem ser encontradas na seção 5.6.



Bibliografia

[AHS85] D. H. Ackley, G. E. Hinton, and T. J. Sejownowski. A learning algorithm for boltzmann machines. *Cog. Sci. 9*, pages 147 – 169, 1985.

[AM91] I. Aleksander and H. B. Morton. General neural unit: Retrieval peformance. *Eletronics Letters*, pages 1776 – 1778, September 1991.

[AMP87] Yaser S. Abu-Mostafa and Demetri Psaltis. Optical neural computers. *Scientific American*, pages 66 – 73, March 1987.

[Arm89] James R. Armstrong. *Chip-level modeling with VHDL.* Printice-Hall, Inc., 1989.

[AW85] I. Aleksander and M. J. Dobree Wilson. Adaptive windows for image processing. *Proc. IEEE, Computers and Digital Techniques*, pages 233–245, September 1985.

[Cor90] Mentor Graphics Corporation. *V8.0 An Introduction to Modeling in VHDL - Student Workbook.* 1990.

[Côr91] M. L. Côrtes. *Introdução ao Teste de Circuitos Digitais.* 1991. V EBAI.

[Dep85] Department of Electrical Eng. and Comp. Sciences, University of California, Berkeley. *Magic Tutorials*, 1985.

[Dep90] Department of Electrical Eng. and Comp. Sciences, University of California, Berkeley. *Spice 3 Version 3d2 User's Manual*, 1990.

[Egg90] Lewis C. Eggebrecht. *Interfacing to the IBM Personal Computer.* SAMS, 1990.

[Gem82] Peter Gemmar. Image corelation: Processing requirements and implementation structures on a flexible image processing system (flip). In *Multicomputers and Image Processing Algorithms and Programs*, pages 87 – 98. Academic Press, Inc., 1982.

[Ges90] Tama's Geszti. *Phisical Models of Neural Networks.* World Scientific, 1990.

[Hay88] John P. Hayes. *Computer Architecture and Organization.* McGraw-Hill, 1988.

[HR91] John P. Huber and Mark W. Rosnec. *Successfull ASIC Design, the First Time Through.* Van Nostrand Reinhold, 1991.

[I3E87] *IEEE Standard 1076 VHDL Language Reference Manual*, 1987.

[IBM84] *Technical Reference Manual For PC-AT*, 1984.

[Int86] *Intel 80386 High Performance 32-bit Microprocessor with Integrated Memory Manage-mant*, 1986.

[Int91] *Intel 8-bit Embedded Controllers*, 1991.

[Lei93] Dertrand Leigh. Complex pld & fpga architectures. *ASIC & EDA, Technologies for System Design*, pages 44 – 50, February 1993.

[LF84] Hsi-Ho Liu and King-Sun Fu. VLSI arrays for minimum-distance classifications. In *VLSI for Patern Recognition and Image Processing*, pages 45 – 63. Springer-Verlag, 1984.

[Luc91] J. Lucy. Perfect auto-associators using RAM-type nodes. *Eletronics Letters*, pages 799 – 800, May 1991.

[Mac87] Ronald J. MacGregor. *Neural and Brain Modeling.* Academic Press, Inc., 1987.

[Mea89] Carver Mead. *Analog VLSI and Neural Systems.* Addison-Wesley Pub. Company, 1989.

[Men93a] Mentor Graphics Corporation. *Design Architect Reference Manual*, 1993.

[Men93b] Mentor Graphics Corporation. *Design Architect Users's Manual*, 1993.

[Men93c] Mentor Graphics Corporation. *Getting Started with IC Station Training Workbook*, 1993.

[Men93d] Mentor Graphics Corporation. *A Guide to Design Process and Database Concepts*, 1993.

[Men93e] Mentor Graphics Corporation. *Mentor Graphics Technical Publications Overview*, 1993.

[Men93f] Mentor Graphics Corporation. *QuickSim II Users's Manual*, 1993.

[MKD82] Bruce H. McCormick, Ernest Kent, and Charles R. Dyer. A visual analizer for real-time interpretation of time-varying imagery. In *Multicomputers and Image Processing Algorithms and Programs*, pages 453 – 464. Academic Press, Inc., 1982.

[Mur91] A. F. Murray. Silicon inplementations of neural networks. *IEE Proceedings - F*, pages 3 – 12, February 1991.

[Ove76] Lan Overingtom. *Vision and Acquisition.* Pentech Press, 1976.

[Per91] Douglas L. Perry. *VHDL(Computer Hardware Description Language).* McGraw-Hill, Inc., 1991.

[Red87] A. Redgers. Adrian's guide to understanding the wisard. Technical report, Imperial College of Science and technology, 1987.

[Red88] A. Redgers. An orverview of neural nets. Technical report, Imperial College of Science and Technology, 1988.

[Ree82] Anthony P. Reeves. Parallel algorithms for real-time image processing. In *Multicomputers and Image Processing Algorithms and Programs*, pages 7 – 18. Academic Press, Inc., 1982.

[RR91] U. Ramacher and U. Rückert. *VLSI Design of Neural Networks.* Kluwer Academic Publisher, 1991.

[SJ93] Jay Sturges and Jesse Jenkins. Benchmarks, metrics, and brass tacks. *ASIC & EDA, Technologies for System Design*, pages 26 – 42, June 1993.

[Ste82] Stanley R. Sternberg. Pipeline architectures for image processing. In *Multicomputers and Image Processing Algorithms and Programs*, pages 291 – 305. Academic Press, Inc., 1982.

[Tan82] Steven L. Tanimoto. Programming techniques for hierarchical parallel image processors. In *Multicomputers and Image Processing Algorithms and Programs*, pages 421 – 429. Academic Press, Inc., 1982.

[TH87] D. W. Tank and J. J. Hopfield. Collective computation in neuronlike circuits. *Sci. Am.*, pages 62 – 70, December 1987.

[TWH93] Mark Thorson, Forest Warthman, and Mark Holler. A neural network audio synthesizer. *Dr. Dobb's*, pages 50 – 64, February 1993.

[Wal93] Larry Waller. High-density programmables lead the pack. *ASIC & EDA*, pages 38–49, July 1993.

[Was89] Philip D. Wasserman. *Neural Computing: Theory and Practice.* Van Nostrand Reinhold, 1989.

[WE89] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design: a System Perspective.* Addison-wesley Publishing Co., 1989.

[YON82] Shuji Yoshizawa, H. Osada, and Jin-Ichi Nagumo. Pulse sequences generated by a degenerate analog neural model. *Biological Cybernetics*, pages 23 – 33, 1982.