Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Depto. de Semicondutores, Instrumentos e Fotônica

Projeto e Implementação em VLSI de uma Rede Neural Auto-Organizável usando Síntese Automática de Alto Nível

Autor : **Marcelo Arturo Jara Pérez**

Orientador : **Prof. Dr. Furio Damiani**

Tese apresentade à Fac. de Eng. Elétrica da Univ. Est. de Campinas - UNICAMP, como parte dos requisitos exigidos para a obtenção do título de Doutor em Engenharia Elétrica.

Área de Eletrônica e Comunicações

Campinas, SP, Brasil, agosto de 1997

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

J28p

Jara Pérez, Marcelo Arturo
Projeto e implementação em VLSI de uma rede neural auto-organizável usando síntese automática de alto nível / Marcelo Arturo Jara Pérez.--Campinas, SP: [s.n.], 1997.

Orientador: Furio Damiani.
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes neurais (Computação) 2. Circuitos integrados digitais. 3. Computadores - Circuitos. 4. Arquitetura de computador. 5. Algoritmos paralelos. I. Damiani, Furio. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Agradecimentos

Dedico este trabalho aos meus pais Ruth e Aurelio, pela sua abnegação e carinho e à minha esposa Liana, pelo seu amor e pela sua paciência incondicional.

Agradeço ao professor Furio Damiani pela sua orientação acadêmica e pela sua contribuição relevante no meu crescimento pessoal e técnico-acadêmico. Ao professor Peter Tatsch, pelas conversas sempre interessantes, pelos constantes esclarecimentos técnicos e pela assistência nas fotos dos *chips*. A ambos agradeço também a companhia no *bistrô* local (o *laboratório* experimental onde são realizadas as degustações do produto dos cafezais da região).

Ao CNPq e à FAEP/Unicamp pela assistência financeira.

A todos os meus colegas e amigos do DSIF pelo companheirismo e amizade.

Agredeço especialmente ao meu amigo e colega de laboratório Wilfredo Machaca, cujo rigoroso trabalho no modelamento VHDL, no projeto e nos testes de avaliação permitiu que a parte experimental deste trabalho, o projeto e a construção do *chip* protótipo, fosse realizada com sucesso.

A Silvio Nogueira da CPqD/Telebrás pela sua labor essencial na correção dos defeitos do chip, utilizando com eficiência o feixe laser e o micromanipulador, eliminando o problema do curto-circuito indesejado.

A fabricação do protótipo foi realizada por meio do apóio financeiro da FAPESP. Agradeço às pessoas da comissão que avaliaram o meu projeto e autorizaram a construção do *chip* a través do programa CMP (*Circuits Multi-Projects*) da França na *foundry* da AMS (*Austria Mikro Systeme*).

Agradeço a *DU* pela gentileza e disposição na correção de todo o texto em portugûes.

Meus sinceros agradecimentos para Raquel, da secretaria do DSIF, pela valiosa cooperação.

Resumo

Projeto e Implementação em VLSI de uma Rede Neural Auto-Organizável usando Síntese Automática de Alto Nível

Neste trabalho realiza-se o estudo do algoritmo SOFM (*Self-Organizing Feature Map*) para a sua implementação em circuitos digitais ASIC VLSI. Foram projetados e construídos 2 *chips*: o primeiro implementa uma célula da rede neural e o segundo o bloco WTA (*Winner-takes-All*). O sistema foi inicialmente simulado com uma linguagem procedural (ANSI-C), construindo-se um programa com interface gráfica para plataforma UNIX. Posteriormente, foi realizada uma descrição em alto nível usando a linguagem VHDL (*Very high-speed circuits Hardware Description Language*). Em seguida, a descrição foi feita a nível RTL (*Register Transfer Level*) e o circuito foi sintetizado e otimizado seguindo uma metodologia *Top-Down*. Os circuitos foram implementados em tecnologia digital usando um processo CMOS de 1,2 *microns* para as células e de 0,8 *microns* para o bloco WTA. Esses circuitos foram objeto de testes e verificação funcional, para avaliação de seu desempenho.

Os resultados permitiram verificar a validade da metodologia *Top-Down* para o projeto de sistemas eletrônicos complexos. A frequência máxima de operação das células excede 20 MHz e a do bloco WTA excede 50 MHz. A dissipação de potência para 20 MHz foi de aproximadamente 50 mW para uma célula. Todos os circuitos foram implementados usando ferramentas de projetos (CAD-EDA) da Mentor-Graphics Co.™, e bibliotecas *std-cells* CMOS AMS. Observaram-se algumas diferenças entre os resultados das simulações e as medidas experimentais.

Palavras-chaves : Redes Neurais, Sistemas Auto-Organizáveis, Implementação em *hardware* de Redes Neurais, Circuitos Integrados VLSI. Síntese Automática de Circuitos. Modelamento e Descrição VHDL. Metodologia de projeto *Top-Down*. Algoritmos para Processamento Neural.

Abstract

Design and VLSI implementation of a Self-Organising Neural Network using High level Synthesis and Modelling

A Kohonen-based (SOFM - Self-Organizing Feature Map) artificial neural network was simulated, modelated and hardware implemented in a VLSI circuit. A Top-Down methodological approach was used by using ANSI-C and VHDL (Very High Speed Circuits, Hardware Description Language). The original SOFM algorithm was lightly modified for customizing to the hardware implementation requirements. After a high-level modelling and simulation, a fully-digital VLSI Neuroprocessor chip prototype was designed and manufactured in a CMOS 1.2 microns technology. Most of the circuits structures of Neuron were automatically generated from a VHDL RTL description using automatic synthesis, the others were obtained trough conventional schematics procedure. After functional verification, the resulting circuits were optimizated (drived by silicon area minimization) and mapped to the AMS technology, a 2-level metal process from Austria Mikro Systeme.

The Neuron cell has 6 bi-directional 3-bits capability connections, used for neighbours communication, allowing to implement a hexagonal type dynamic Nc(t) neighbourhood. Both Nc(t) radio and gain Alfa function may be programmed by using a set of registers, allowing high flexibility for studying different SOFM algorithm convergence conditions. A second chip was designed and manufactured using a AMS CMOS 0.8 microns technology for implementing a competitive on-chip learning. This circuit is part of a WTA (Winner-Takes-All) block used for determine a winner cell in each epoch of the self-organized training phase. Some differences were observed after comparing measures and simulation results.

Keywords : Artificial Neural Networks; Self-Organising Systems, VLSI (Very Large Scale Integration) systems/circuits design; Hardware (VLSI) implementation of Neural Networks; Automatic Synthesis of logic circuits and systems; High Level modelling and VHDL description; Top-Down design methodology; Neural processing and adaptive algorithms.

Introdução

As redes neurais artificiais apresentam desempenho interessante na solução de uma classe de problemas, onde principalmente é necessária uma grande quantidade de processamento paralelo. Porém, atualmente a maior parte dos estudos na área focalizam-se na análise teórica (propriedades de convergência, estabilidade, etc.) e nas simulações implementadas em *software*, devido obviamente à maior facilidade e ao seu relativo baixo custo de implementação. No entanto, a fim de aproveitar o potencial do paralelismo maciço que apresentam os modelos neurais, requer-se uso de arquiteturas cujo grau de paralelismo permita pelo menos a sua emulação em alta velocidade. A implementação em *hardware* de redes neurais permite uma maior aproximação ao paradigma de modelos inspirados biologicamente, especialmente do ponto de vista do seu desempenho. Como alternativas tecnológicas atuais existem os dispositivos ópticos, tal como implementações utilizando técnicas holográficas, e os circuitos digitais, analógicos e híbridos implementados em tecnologia CMOS, BiCMOS ou GaAs. Porém, as tecnologias ópticas são de pouca aplicação para interagir com sistemas convencionais operando no mundo real, e apesar das vantagens de velocidade da tecnologia GaAs, a maioria das implementações correntes têm sido realizadas utilizando processos CMOS convencionais. A principal causa desta escolha é a sua vantagem econômica, em comparação aos processos de fabricação em tecnologia BiCMOS ou GaAs e ao baixo consumo relativo de potência dos circuitos CMOS. Por outro lado, existe ainda a divisão causada pela forma de processamento dos sinais, assunto que está mais relacionado com o tipo de problemas que deseja-se resolver. Atualmente, tem havido um crescente aumento na produção de redes neurais implementadas em circuitos analógicos usando variadas tecnologias (modo corrente, seqüência de pulsos, etc.), aproveitando principalmente as vantagens de compactação que oferece tal tecnologia. O uso da tecnologia analógica revela-se muito mais interessante em aplicações relacionadas com processamento em *baixo nível*, como sistemas de percepção, sensores inteligentes, sistemas de visão computacional, retinas artificiais, etc. Em contrapartida às implementações analógicas, a alternativa da implementação digital apresenta-se atraente por causa de diversos aspectos, alguns deles mencionados a seguir.

Facilidade de realização da interface entre subsistemas. Na maioria das aplicações correntes, os sistemas eletrônicos digitais tornaram-se ubíquos. Considere-se o uso de tecnologia atual em áreas como por exemplo: comunicações digitais, armazenamento maciço de informação, processamento de sinais (compressão/descompressão de vídeo/imagem, voz), controle em tempo real, microsistemas e microcontroladores integrados, etc.

Precisão. Um aspecto essencial no desenvolvimento de sistemas que exigem processamento com alto grau de precisão, com vantagens para os circuitos digitais.

Programabilidade.O grau de programabilidade dos circuitos e sistemas define de alguma forma a sua flexibilidade, aspecto muito mais facilitado nas implementações digitais.

Uma vantagem que atualmente favorece particularmente os projetos de sistemas digitais, especialmente os sistemas eletrônicos complexos, é a utilização da modelagem em alto nível com linguagens de descrição de *hardware* (HDL - *Hardware Description Languages*). O uso cada vez mais freqüente de HDLs, tal como VHDL (*Very High Speed Circuits* **HDL**) ou *Verilog*, tem ocasionado um avanço significativo no projeto de sistemas e microsistemas eletrônicos integrados, embora este avanço seja normalmente quantificado mais em termos da produtividade ou do *time-to-market*, i.e., no tempo de desenvolvimento do projeto desde a sua concepção até a sua colocação no mercado. Associado ao uso de HDLs é a existência de ferramentas de automação de projetos (EDA - *Electronic Design Automation*) cada vez mais sofisticadas, que permitem a síntese de circuitos a partir de diversos níveis de modelagem. Após realizada a síntese, os circuitos são posteriormente mapeados e implementados numa tecnologia específica, com a assistência de completas bibliotecas e bases de dados (incluídas nos *kit designs* da tecnologia utilizada), onde incorporam-se as características mais relevantes dos dispositivos, tal como capacidade de *fan-in/fan-out*, capacitâncias, tempos de atraso das *standard cells*, etc. Existe hoje ainda a possibilidade de uso de bibliotecas mais sofisticadas como as HLLM (*High-Level Library Mapping*) e elementos IP (*Intellectual Property*), onde componentes mais complexos, como por exemplo unidades aritméticas (ALUs, DSPs, etc.), podem ser incorporados ao projeto e integrados no *chip*, utilizando e aplicando o conceito de reutilização de *hardware*. Além disso, há uma tendência cada vez mais intensa ao uso de ferramentas de verificação formal, para aumentar a confiabilidade no funcionamento dos circuitos, especialmente após a fase de síntese.

Em relação à implementação em *hardware* do modelo auto-organizável de Kohonen, já existem algumas propostas abrangendo diferentes tipos de máquinas, como sistemas sistólicos e arquiteturas baseadas em processamento estocástico, entre outras propostas. A diversividade de formas e tecnologias de implementação de modelos neurais tem revelado a necessidade de classificar as máquinas, de acordo com a sua natureza e propósito. Uma classificação proposta para *engenhos neurais* consiste na divisão entre sistemas de propósito geral e de aplicação específica. Os primeiros são normalmente constituídos por aceleradores de *hardware* que melhoram significativamente o desempenho de algoritmos para execução em máquinas convencionais. Esta proposta revela-se interessante do ponto de vista da flexibilidade, devido a que diversas variantes de um algoritmo e de topologias podem ser implementadas. Da mesma forma, facilita-se a implementação de diversos modelos, o que revela-se bastante viável desde uma perspectiva comercial. Em outro segmento, existem as implementações dedicadas, normalmente projetadas para execução em tempo real e para a aplicação de um algoritmo específico. As máquinas baseadas neste segundo paradigma evidentemente conseguem melhor desempenho que as da categoria anterior, com a desvantagem da restrição de seu campo de aplicação.

A arquitetura proposta neste trabalho, adequa-se à segunda categoria mencionada. Apresenta-se a seguir o estudo e o projeto de uma implementação em VLSI (*Very Large Scale Integration*) de um circuito digital dedicado à implementação em *hardware* do algoritmo neural SOFM (*Self-Organizing Feature Map*) proposto por T. Kohonen.

Índice

Cap. 5. Desenvolvimento de um protótipo de um Neuroprocessador para o SOFM.

Cap. 6. A arquitetura do Neuroprocessador *Neuron*.

Cap 7. Implementação física e Resultados Experimentais.

Capítulo 1

Auto-Organização - Retrospectiva

Escopo

Neste capítulo apresenta-se uma introdução ao estudo dos sistemas auto-organizáveis, começando com uma retrospectiva histórica.

Os sistemas auto-organizáveis têm sido objeto de estudo das tendências mais diversas da ciência como: Matemáticas (sistemas caóticos e não lineares), Física (sinergia), Biologia Celular, Neurociência, Cibernética, Teoria da Evolução e, nestes últimos anos, de grande parte de cientistas da área das Ciências Cognitivas. A evolução da tecnologia em computadores e da eletrônica, também atingiu *naturalmente* a área da ciência da computação e da engenharia, especialmente dos grupos interessados em Inteligência Artificial e em paradigmas de sistemas neurais artificiais. Considerando que a auto-organização forma parte de um dos aspectos mais importantes da natureza, têm se organizado diversos estudos multidisciplinares na área [1], [2]. Tais iniciativas se remontam aos finais da década dos anos 50 [3], numa época em que o desenvolvimento dos dispositivos semicondutores permitia a realização dos primeiros circuitos integrados, ainda muito distantes das atuais implementações em VLSI (*Very Large Scale Integration*) e ULSI (*Ultra Large Scale Integration*).

Neste capítulo far-se-á uma abordagem dos sistemas auto-organizáveis, partindo desde os conceitos iniciais até convergir aos sistemas inspirados biologicamente, origem dos trabalhos que tratam o estudo de modelos de redes neurais artificiais, como os sistemas neurais adaptativos, de eficiente aplicação na área de reconhecimento de padrões (imagem, voz, etc.). O objetivo deste trabalho concentra-se na implementação VLSI de algoritmos de uma rede neural do tipo auto-organizável, baseada no modelo proposto em 1982 por T. Kohonen [4], visando sua realização em sistemas eletrônicos da forma mais compacta possível, em tecnologia digital.

1.1 Introdução

Pode-se afirmar que os sistemas tecnológicos são organizados por **comandos** externos, normalmente fornecidos por humanos ou por outras estruturas ou máquinas. Por outro lado, os sistemas naturais (biológicos, físicos, etc.) normalmente são estruturados devido aos seus próprios processos internos, conduzindo a processos *auto-organizáveis*. O surgimento da ordem nestes sistemas é um fenômeno complexo, muitas vezes não bem definido.

O termo **auto-organização** pode ser abordado desde as mais diversas perspectivas. Uma tentativa de definição que leve a um consenso, deve incluir necessariamente uma visão multidisciplinar. Pode-se afirmar desde este ponto de vista que a auto-organização é considerada como um **processo**, que leva gradual e espontaneamente o sistema desde níveis inferiores a superiores de organização. No entanto, há atualmente uma carência de uma medida quantitativa ou uma escala de ordenação de **consenso** em relação à noção de organização.

Uma proposta feita há vários anos pelo matemático J. von Neumann tentou a busca de uma medida qualitativa, que pudesse representar fielmente o grau de organização de um sistema relativamente simples, tal como o definido por Schuster, num *reator de evolução* [5]. Numa outra abordagem proposta por H. Haken [6], a *Sinergia* é oferecida como a construção física básica para o entendimento de um processo de Auto-Organização. A proposta de Haken é fundamentada em uma abordagem matemática-física, que conduz a uma teoria que tenta explicar como alguns sistemas (como por exemplo, átomos e células) podem produzir estruturas e padrões através de processos auto-organizáveis. A *Sinergia* trata de sistemas que produzem estruturas espaciais macroscópicas, temporais ou funcionais e do estudo de sistemas abertos distantes do equilíbrio termodinâmico [7].

1.1.1. Algumas Definições Preliminares.

Devido ao fato de se dispor de informações provindos de diversos campos da ciência, tentar-se-á uma abordagem pluralista sem abandonar o objetivo da idéia principal deste trabalho, consistente na proposta de uma máquina que implemente um processo auto-organizável.

Desde o início da história da ciência da computação, tem se proposto uma variedade de máquinas destinadas a implementar operações diversas, abrangendo desde uma abordagem virtual e completamente não-implementável com a tecnologia disponível até estruturas completamente dedicadas a resolver uma única tarefa do mundo real. A nossa proposta é baseada num modelo bem conhecido proposto em 1982 por T. Kohonen [4], na época na Univ.Tec. de Helsinki. Este modelo tem se revelado muito adequado na solução de problemas relacionados com tarefas de classificação e reconhecimento de padrões, é um modelo inspirado biologicamente e sem dúvida fundamentado nas idéias pioneiras desvendadas por Willshaw et al. e publicadas num artigo de 1976 pela *Royal Society of London* [8].

Iniciar-se-á este capítulo enunciando uma série de postulados científicos básicos propostos na literatura desde a década dos anos 50 com uma abordagem desde o ponto de vista da ciência cognitiva até chegar aos postulados biológicos da auto-organização, percorrendo as

áreas da biofísica e outras relacionadas.

1.1.2. O Conceito básico de Auto-Organização (AO).

Numa recente publicação [2], vários autores dedicaram-se a desvendar muitos conceitos e idéias relativos à definição de auto-organização, desde um ponto de vista eclético e multidisciplinar. M. Debrun propõe inicialmente uma definição fundamentada numa perspectiva filosófica, definindo uma auto-organização primária e secundária, produtos da dinâmica do processo. Na essência, define a existência de AO cada vez que a "reestruturação ou o advento de uma forma é conseqüência do próprio processo de AO e das suas características intrínsecas", i.e., a influência de agentes externos (como por ex., o intercâmbio de informação, matéria, energia com o meio) no processo seria mínima.

1.2 Organização desde o ponto de vista do estudo macro-molecular.

O *reator de evolução* é um sistema aberto que permite a entrada de um fluxo de material de baixo peso molecular, trata-se de um *reator Vessel*, onde é possível controlar a sua temperatura e pressão. Sob tais condicões ideais é possível estudar o processo de competição entre diferentes unidades auto-replicantes, i.e., seqüências distintas de RNA ou DNA. As macromoléculas se auto-organizam e desenvolvem uma linguagem genética. A réplica é a propriedade de certas biomoléculas que permitem a sua cópia única, compondo um sistema em constante evolução, cuja dinâmica conduz a distintos graus de organização. Neste sistema idealizado, o grau de organização é definido por Schuster como uma função dependendo de 3 pontos característicos:

(1) *Dimensão N*: Número de macromoléculas ou espécies cooperativas, auto-replicantes. Schuster define o grau de cooperação como a ausência de competição.

(2) *Grau P* : Molecularidade, se relaciona especificamente ao número de macromoléculas que participam do processo de replicação.

(3) *Complexidade Dinâmica* do sistema, R. Relaciona-se com as leis (físicas) que governam a evolução no tempo do sistema.

Utilizando a combinação de parâmetros mencionados, Schuster define o grau de organização do sistema. Considere-se, por exemplo dois sistemas definidos pela tríade N_1,P_1,R_1 e N_k,P_k,R_k. Se $N_1>N_k$, $P_1>P_k$, $R_1>R_k$, o sistema **1** seria mais organizado que o **k**.

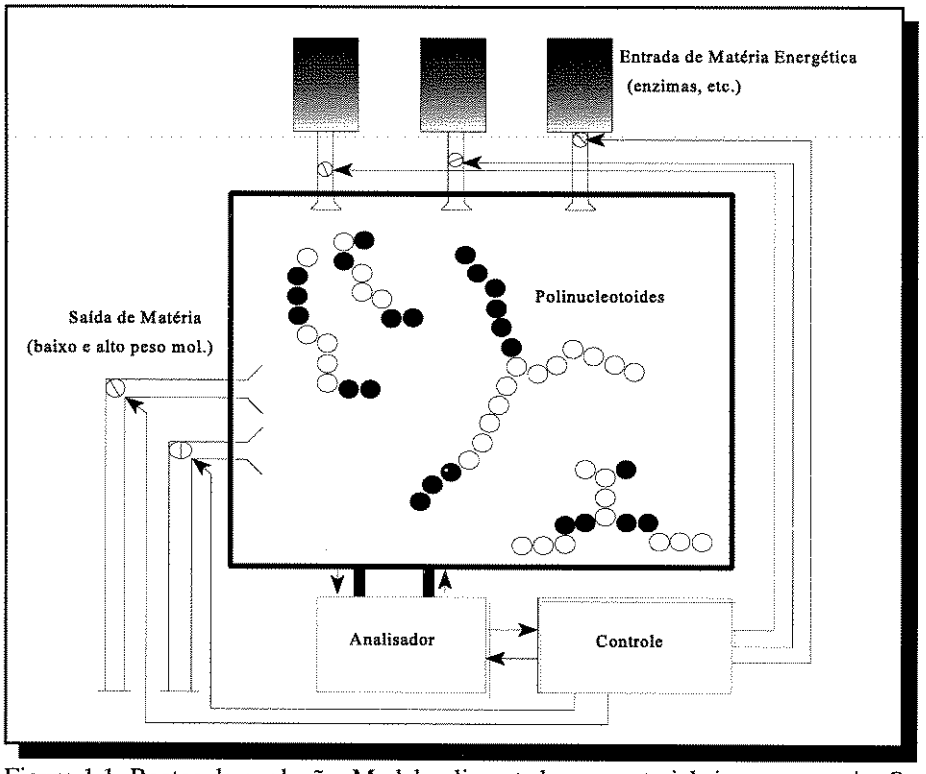


Figura 1.1. Reator de evolução. Modelo alimentado por material rico em energia. Os produtos degradados são constantemente removidos pelos dutos de saída de matéria de baixo e alto peso molecular.

Na Fig. 1.1, apresenta-se o diagrama simplificado do reator de evolução, conhecido também como *reator vessel*. A *auto-replicação* é considerada uns dos fenômenos fundamentais de auto-organização. Os polinucleotóides são as únicas biomoléculas conhecidas que têm a capacidade de se replicar, .i.e, são geradas cópias únicas a partir das células originais.

1.3. Organização desde o ponto de vista da evolução.

Freqüentemente definem-se novos conceitos no estudo da transição da matéria não viva à viva e do nosso sistema vivo (planetário, biológico, etc.), que pode ser entendido como o resultado de um longo processo de evolução. Os avanços nos estudos nesta área permitiram desenvolver uma chamada *Teoria do processo Auto-Organizável* [9]. Ebeling & Feister definem fisicamente no seu trabalho a evolução como um processo histórico irreversível, formado por uma série sem limite de passos auto-organizáveis. Eles acrescentam que cada passo é iniciado após uma chamada *flutuação crítica*, tal como uma mutação relevante, que levaria desde um estado original semi-estável a um outro de instabilidade. A chamada *flutuação* dispararia assim um processo de Auto-Organização que rearranjaria os elementos do sistema e o conduziria a um novo estado meta-estável. Na seqüência, o processo pode ser repetido levando o sistema a níveis

superiores de organização. Ebeling & Feister definem os estados do sistema em evolução como a distribuição de pontos localizados num espaço apropriado, como por exemplo, num **espaço de fase**. Desta forma, a evolução pode ser visualizada como um campo de ocupação dinâmico (em movimento) através do espaço de fase definido.

1.4. Auto-Organização desde o ponto de vista biológico.

Os organismos vivos constituem a forma mais excepcional de auto-organização. Todos eles devem ter a capacidade de classificar os estímulos apresentados pelo meio ambiente, a fim de permitir-lhes a sua própria sobrevivência. Os organismos mais simples possuem procedimentos de classificação predeterminados, enquanto outros mais sofisticados organizam e classificam os estímulos de entrada, se adaptando notavelmente às mudanças e ao aumento de complexidade das características do seu ambiente. Tal capacidade foi definida por B. Farley [10] como *Percepção Aprendida*. Farley considerou um modelo do processo de classificação como um sistema de processamento de dados, com a capacidade de computar medidas a partir das propriedades de algumas entradas, apresentadas durante um certo período de tempo. As propriedades consideradas por Farley foram características físicas, como: cor, temperatura, peso, etc. ou relações entre outras propriedades. Por exemplo, na percepção auditiva, são consideradas como propriedades mensuráveis a freqüência e a intensidade dos sinais acústicos.

Farley introduziu na sua proposta o conceito de **distribuição de probabilidade**, que será muito útil nos capítulos seguintes deste trabalho, ao ser apresentado o modelo neural auto-organizável de Kohonen. Como cada objeto pode ser definido em função de uma relação das propriedades associadas, se lhe atribue também o número de vezes que a propriedade é observada, i.e., a sua **distribuição de probabilidade** no espaço observado.

Esta característica permite a formação de classes para cada grupo de objetos com o conjunto de propriedades similares, com a vantagem que pode ser distinguido um vasto número de classes, através do uso de poucas propriedades. Falta agora só elucidar o procedimento que permitiria compilar ou organizar estas classes, a essência do processo auto-organizável.

A idéia proposta por Farley seria de criar conjuntos de regras de agrupamento baseadas na freqüência de ocorrência da propriedade e da sua continuidade no tempo, entre outros fatores. Espera-se assim que um sistema auto-organizável, quando exposto a um número razoável de entradas vindas do seu ambiente, seja capaz de gradual e automaticamente classificar adequadamante os padrões de entrada. Além do mais, o sistema deve poder eficientemente reconhecer um padrão desconhecido quando é submetido ao sistema e o mesmo mostra suficiente correlação com algum dos já armazenados e classificados.

Até este ponto não se tem definido qual será o procedimento que regulará este processo, a natureza exata das regras de agrupamento e classificação são desconhecidas e somente são evidentes a partir da observação das características biológicas e o comportamento de sistemas auto-organizáveis já existentes na natureza.

1.5. Um modelo abstrato.

Uma proposta de Gordon Pask [11], definiu um chamado *Modelo Abstrato de um Sistema Auto-Organizável*, baseado na suposição de uma série de condições que devem ser cumpridas para a construção do modelo :

(1) Um espaço de dimensões arbitrárias onde uma rede é definida através da conectividade efetiva entre pares de pontos. Supõe-se também que tal espaço é composto por um material inicialmente homogêneo e maleável.

(2) Uma grandeza, a qual poderia ser identificada com, por exemplo, a energia, que é conservada. As condições de conservação fazem-na mensurável e será garantida caso se defina uma taxa θ à qual a grandeza (energia) flue através do espaço.

(3) Um conjunto de servomecanismos. Pask identificou-os como amplificadores não-lineares ou osciladores, com energia local ou capacidade de armazenamento.

(4) Um conjunto de regras que determinam a mudança num sinal e conectividade, induzidas pela atividade no espaço. As regras devem ser expressadas pelas características de impedância de sinal do material maleável que compõe o espaço.

Supondo-se que o espaço da rede é indefinidamente extenso e que em lugar do fluxo da grandeza θ no espaço restringe-se o fluxo a θ por unidade de volume do espaço. Neste caso, há uma vantagem em termos da competição pela energia entre os elementos do servomecanismo, se estes elementos cooperam. Dito de outra forma, um conjunto de servomecanismos está em vantagem se as suas atividades extrapolam a região conectada no espaço da rede. Tal região estará limitada pelo ganho do servomecanismo e a energia disponível. Em sistemas implementáveis, uma região conectada e ativa movimenta-se ao longo do espaço da rede, capturando servomecanismos não envolvidos em atividade. Tal sistema descrito acima foi definido por Pask como um *Sistema Auto-Organizavel Abstrato.*

Implementação Física

Tal modelo foi fisicamente construído por Pask, utilizando elementos de computadores analógicos, como válvulas e resistores. Nesta empreitada, Pask teve que se enfrentar com muitos problemas de construção, devido principalmente ao comportamento não linear dos componentes. Uma válvula elétrica, por exemplo, recebe como entrada um sinal elétrico que é amplificado na sua saída. No entanto, tal elemento responde também às variações de parâmetros como temperatura e até vibração, gerando às vezes um comportamento imprevisível.

Não foram essas as únicas dificuldades encontradas por Pask. Ao tentar modelar o seu sistema Auto-Organizável com outros componentes, por exemplo, na implementação de um modelo mecânico, ele se deparou com os mesmo problemas, i.e, com componentes cujo

comportamento era extremamente ambíguo sob determinadas condições de operação.

A implementação final do modelo de Pask, estava constituído por servomecanismos elétricos, usando a energia elétrica como grandeza mensurável e impulsos elétricos como sinais. Tal máquina era, na verdade, uma máquina de aprendizado adaptativo (de aprendizagem finito).

1.6. O modelo *Perceptron* de Rosenblatt.

Na década de 50 e início dos anos 60, surgiu na comunidade científica conexionista uma grande controvérsia devido ao modelo proposto por Frank Rosenblatt [12]. A rigor, o sistema proposto originalmente por Rosenblatt, do *Cornell Aeronautical Laboratoty* (Buff., NY), era baseado nos neurônios formais de McCulloch e Pitts [13], cujo aprendizado era governado por algoritmos supervisionados. Na figura 1.2, representa-se uma versão simplificada da máquina proposta por Rosenblatt, chamada por ele de *cross-coupled perceptron.*

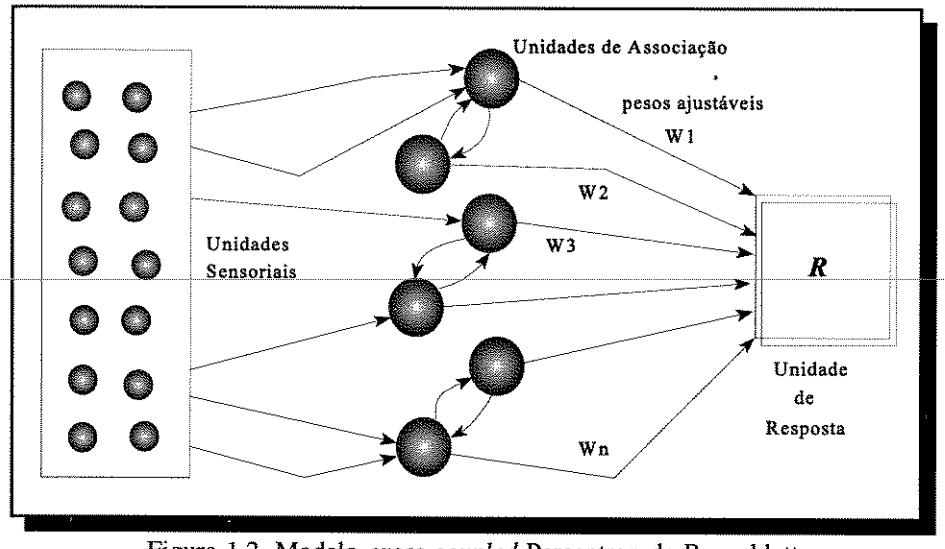


Figura 1.2. Modelo *cross-coupled* Perceptron de Rosenblatt

O modelo original construído por Rosenblatt estava formado por 8 unidades de saída. É formado por uma camada de unidades sensoriais, encarregadas de receber os estímulos do exterior, de uma outra camada de unidades de associação e de unidades de saída. A rigor, trata-se de uma rede de uma camada só com conexões do tipo *feedforward.* Todas as unidades de associação são conectadas à unidade de saída, através de pesos ajustáveis pelo algoritmo de aprendizado. Foram desenvolvidos vários algoritmos de treinamento para este modelo, um proposto pelo próprio Rosenblatt (1960) e outros como o LMS, proposto por Widrow e Hoff (1960)[14]. O *Teorema da Convergência do Perceptron* de Rosenblatt mostrou que o seu modelo era capaz de realizar tarefas de classificação, após uma série de ciclos de treinamento, envolvendo apresentação dos padrões, observação da saída das unidades e posterior ajuste dos pesos, em função do algoritmo utilizado.

O Teorema da convergência do Perceptron foi provado para uma versão simplificada,

onde não foram consideradas as unidades sensoriais de entrada. Mesmo assim, trata-se de um problema do tipo NP-completo, ou seja, exponencialmente intratável. Desta forma, embora o Perceptron possua um poderoso algoritmo de classificação, o treinamento acaba sendo empírico.

1.7. Os Mapas Organizados Topograficamente.

1.7.1. Introdução.

Os Mapas Organizados Topograficamente são bastante conhecidos na biologia, eles são normalmente encontrados em redes neurais biológicas de organismos animais superiores e permitem explicar como as conexões sinápticas são modificadas na fase de aprendizado. O trabalho pioneiro de D. Willshaw e C. von der Malsburg [8] demonstrou como, na exploração das propriedades de vizinhança de elementos no sistema nervoso, são estabelecidos os mencionados mapas topográficos.Os mapas mais conhecidos são de vertebrados superiores e na maioria dos casos são bi-dimensionais. Exemplos deste mapas são os da retina no córtex visual, e os mapas somatotópicos.Como exemplo de mapas uni-dimensionais pode-se mencionar o mapa tontópico, que preserva a organização do sistema auditivo da cóclea e do canal de audição.

A modelo proposto por Willshaw e Malsburg considera a conexão de células neuronais pertencentes a 2 camadas distintas, ele tentou desta forma representar os elementos pré e pós-sinápticos do sistema neuronal. A idéia é de que os elementos pré-sinápticos possuem axônios que permitem a conexão com os elementos pós-sinápticos, construindo-se um mapeamento entre uma camada sensorial e uma outra (a pós-sináptica) de processamento. Na camada de processamento as células ativadas por um estímulo externo geram padrões de atividade que podem ser representados como *Mapas* num plano bi-dimensional, onde as células pertencentes a uma vizinhança topológica são efetivamente ativadas e portanto geram uma resposta.

1.7.2. Modelagem.

O primeiro problema apresentado é a produção de correlações entre as células vizinhas, já que o modelo é baseado na suposição que a proximidade geométrica entre as células da camada pré-sináptica é codificada na forma de correlações da sua atividade elétrica. Tal atividade é medida em função de, por exemplo, o número de impulsos elétricos por unidade de tempo.A natureza exata desta atividade não é importante para efeitos de modelamento. Por outra parte a atividade entre células geometricamente próximas é reforçada mutuamente, de forma cooperativa, com conexões excitatórias, enquanto as células localizadas distantes do plano devem possuir conexões de tipo inibitórias. Como resultado desta disposição, pode-se deduzir que as células agrupadas em regiões próximas geram níveis de atividade fortes, enquanto as que se localizam em posições mais dispersas e distantes geram níveis de atividade comparativamente mais fracos. As duas camadas definidas anteriormente são interconectadas por sinapses modificáveis, ao longo do processo de aprendizado.

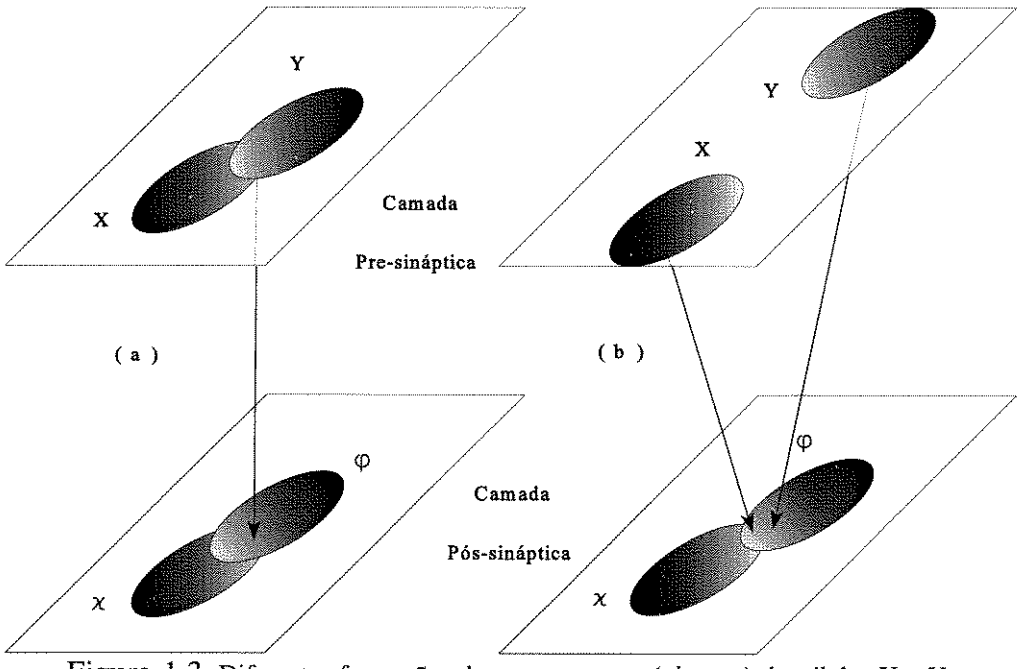


Figura 1.3. Diferentes formações de agrupamentos (*clusters*) de células X e Y.
(a) : configuração estável (b) : configuração instável. Para explicação, vide texto.

Na figura 1.3 se apresentam 2 formas de formação de agrupamentos o *clusters* pelo sistema definido anteriormente. Os *clusters* de células X e Y da camada pre-sináptica se conectam com os *clusters* χ e φ da camada pós-sináptica. O conjunto de células pertencentes à região sobreposta é definida como $X \cap Y$ e as sinapses entre X e χ por $(X \rightarrow \chi)$. As camadas pré e pós-sinápticas assumem 2 tipos de configuração: na configuração **(a)** X e Y compartilham *axons* e sinapses. Conseqüentemente, é altamente improvável que χ e φ ocupem regiões disjuntas na superfície pós-sináptica. Além disso, as sinapses pertencentes a $[(X \cap Y) \rightarrow (\chi \cap \varphi)]$ da configuração **(a)** são reforçadas pela ocorrência de X e pela ocorrência de Y.

Na configuração **(b)**, no caso de pequenas separações entre os clusters X e Y, apresenta-se uma condição de instabilidade. Devido à ação da inibição lateral, as células pertencentes aos *clusters* X e Y nunca estão ativas no mesmo instante de tempo. Desta forma, as sinapses pertencentes a $[X \rightarrow (\chi \cap \varphi)]$ e a $[Y \rightarrow (\chi \cap \varphi)]$, competem entre si, o que não acarreta vantagem para nenhum dois *clusters* de sinapses. No caso de grande separação entre X e Y, esta configuração é excluída do processo, devido às características do mapeamento.

Na modelagem de Willshaw e Malsburg, foi assumido que umas poucas células são inicialmente ativas num determinado instante de tempo, como resultado de algum processo aleatório.Tais células entram em alta atividade se estão agrupadas no mesmo *cluster* e a resposta gerada na camada pós-sináptica é alta, quando são ativadas células agrupadas proximamente. Quanto mais forte é a atividade de uma célula, maior será a modificação das suas sinapses, portanto a atividade evocada será, neste caso, reforçada.

1.7.3. Condições necessárias para geração de mapas auto-organizados.

O mecanismo descrito anteriormente atua somente num nível de atividade microscópico (local). As características de operação do modelo num nível macroscópico (global), devem ser determinadas por um conjunto de condições iniciais e por condições de contorno. Um dos aspectos críticos está relacionado com a possibilidade do processo de mapeamento ser captado por um mínimo local. Neste caso o mapeamento poderia produzir regiões de atividades desconexas e isoladas na sua superfície e nunca convergir a um mapa organizado globalmente. Para superar este problema devem ser destruídas parcialmente algumas regiões ou serem reorientadas, mas estas operações não podem ser realizadas por mecanismos microscópicos.

Em relação ao problema dos ótimos locais, Willshaw sugeriu restringir inicialmente o mapeamento a somente **uma** região de atividade, apartir da qual um mapeamento contínuo é distribuído às outras regiões.

Um outro problema é a determinação da orientação final do mapa. Não há informação suficiente para determinar *a priori* que uma orientação é melhor que outra, portanto é necessário introduzir certos mecanismos, sistematicamente, no padrão de conexões iniciais, para que o mapeamento final não seja resultado de um processo aleatório. Willshaw propôs incluir uma polarização suave e sistemática às conexões iniciais, que seria suficiente para determinar o mapeamento inicial entre as células das camadas pré e pós-sinápticas. Esta proposta é fundamentada pelo fato de que tal região de atividade impõe sua orientação sobre as vizinhas.

A condição de manter só uma região de atividade, na camada pré-sináptica, é essencial para a geração de projeções na camada pós-sináptica. A organização deve proceder pela ação estimulante das células pertencentes à região de atividade num procedimento cooperativo. Isto é, as células organizadas inicialmente devem catalisar a ordem nas células vizinhas através do uso da estimulação lateral. Este deveria ser o único princípio de operação do modelo. Pode-se assim deduzir que as atividades de células isoladas devem ser inibidas através de algum mecanismo, mesmo quando elas sejam sensibilizadas pela atividade pré-sináptica.

A solução proposta foi modificar as sinapses utilizando uma função de atividade pré e pós-sináptica, que desfavorecia os pequenos sinais que não conseguiam atingir um *limiar de modificação* pré-determinado.O valor do limiar foi determinado de forma tal que a atividade induzida pelas células pre-sinápticas não organizadas não fosse reforçada.

A formação dos mapas utilizando o procedimento mencionado garante que regiões de atividade pré-sinápticas desconexas não ativem as suas similares na camada pós-sináptica. Neste modelo, uma célula pós-sináptica **C**, em processo de organização, é ativada quase exclusivamente por estimulação lateral por células vizinhas já organizadas. Este procedimento é a essência do algoritmo proposto por Kohonen e que é o tópico de discussão deste trabalho.

Capítulo 2

O Mapa Auto-Organizável de Kohonen

Escopo

Neste capítulo apresenta-se uma descrição detalhada da rede neural não supervisionada proposta por Teuvo Kohonen: o *Mapa Auto-Organizável* (SOFM).

O modelo de Kohonen é conhecido na literatura como SOM ou SOFM (*Self Organizing Map* ou *Self-Organizing Feature Map*), foi proposto originalmente no ano 1982. O SOFM possui a propriedade de gerar representações internas de diversas características e propriedades dos sinais que lhe são apresentados. Tais representações internas são organizadas espacialmente em um chamado *Mapa Topológico*, que pode ser caracterizado, por exemplo sobre um plano bi-dimensional.

Os Mapas de características topológicas de formação auto-organizada têm sido utilizados com sucesso em diversas aplicações científicas e em áreas da engenharia, tal como no reconhecimento de padrões (imagens/voz), compressão e descompressão de imagens, na área de telecomunicações no processamento adaptativo de sinais, etc.

Neste capítulo, a arquitetura e o algoritmo que implementa a rede de Kohonen serão descritos principalmente desde uma perspectiva teórica, mas sempre visando a implementação em *hardware*. É de especial interesse a sua realização em sistemas e circuitos digitais VLSI (*Very Large Scale Integration*).

2.1 Fundamentos Teóricos

Uma rede neural auto-organizável caracteriza-se por uma arquitetura formada por um arranjo de elementos de processamento (células) conectados sob alguma estrutura topológica, onde células vizinhas reagem a uma sequência de estímulos de um espaço externo através de um processo adaptativo. Cada célula, ou um grupo de células pertencentes a uma **vizinhança topológica**, gera um nível de atividade associado à correlação (ou *sintonia*) entre elas e a entrada corrente distribuída a toda a rede.Os elementos de processamento estão conectados ao espaço externo por meio de *sinapses* e a atividade de cada célula é associada à variação das grandezas destas sinapses (modificação sináptica), em função do conjunto de sinais apresentados seqüencialmente na sua entrada. As modificacões sinápticas são determinadas por um conjunto de regras, conhecido geralmente como o **algoritmo de aprendizagem**, que define os valores iniciais e a forma como as forças sinápticas ou **pesos** (definidos a seguir) evoluem ao longo do tempo. Tal aprendizado é conhecido como **competitivo**.

Neste capítulo trata-se particularmente a descrição do modelo proposto por T. Kohonen [4], [15]-[19] e conhecido como Mapa Auto-Organizável (SOFM -*Self-Organizing Feature Map*), cujo trabalho é baseado tanto em estudos teóricos quanto em simulacões computacionais. A arquitetura do SOFM é composta de elementos físicos adaptativos que recebem sinais de um espaço externo de eventos observáveis, chamado também de espaço sensorial. As representacões dos sinais são automaticamente mapeadas num conjunto de respostas na saída do sistema, de forma que tais respostas adquirem a mesma ordem topológica que os eventos externos. Os SOFMs podem ser caracterizados por uma rede *n*-dimensional de elementos de processamento básicos (normalmente n=2), onde existe alguma forma de interação entre elementos vizinhos, normalmente as células são organizadas num arranjo com n=2. O sistema realiza então um mapeamento dos eventos observáveis sobre a rede bi-dimensional, portanto, existe uma correspondência entre a entrada e uma localização física na rede. O modelo pode ser representado pelo diagrama da figura 2.1.

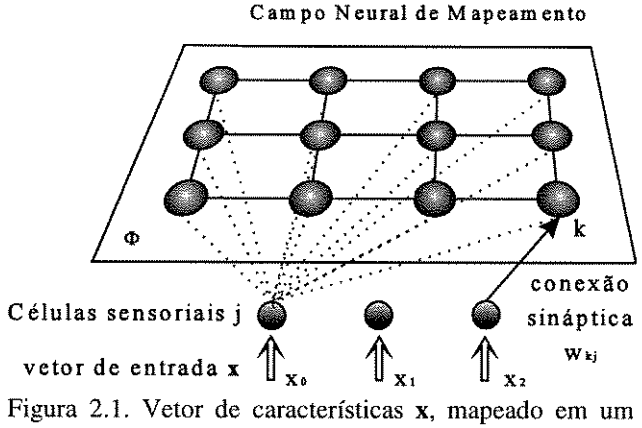


Figura 2.1. Vetor de características **x**, mapeado em um campo neural bi-dimensional Φ.

A figura 2.1 representa um conjunto de neurônios j pertencentes a um espaço sensorial, que recebe uma seqüência de vetores de características **x**, onde $\mathbf{x}=\{x_0,...,x_2\}$. Os sinais de entrada são mapeados em um *campo neural*, Φ, representado em um plano bi-dimensional.

A idéia de *campo neural* foi definida por Sun-Ichi Amari [20]. Na figura 2.1, o neurônio k em Φ é conectado com a célula sensorial j através do peso w_{kj}, a dimensão do espaço característico é determinado pelo número de células sensoriais. Na seqüência, descrever-se-á a forma de representação da informação numa rede neural, segundo a visão de Amari e a formação de mapas topológicos, segundo a proposta de T. Kohonen.

2.1.1.Mapeamento de funcões cerebrais.

Já há tempo que existe evidência da resposta localizada do cérebro de primatas a estímulos produzidos num espaço sensorial [21]. A localização ou distribuição espacial de certas funcões cerebrais, i.e., de uma organização topográfica do cerebro, produz-se particularmente na região do córtex cerebral. Nos cérebros de animais superiores parecem existir diversos tipos de mapas, de tal forma que a localização da resposta naquele mapa tem uma correspondência direta com a modalidade e a qualidade dos sinais sensoriais. Os mapas são então organizados em função das características intrínsecas aos sinais sensoriais, tal como os visuais, tonotópicos e somatotópicos [22]. Segundo Kohonen, existem mapas que representam qualidades abstratas de certas experiências sensoriais, como por exemplo, no processamento de palavras [23]. Pode-se dizer então que as representações internas de informação são de alguma forma organizadas espacialmente, ou de forma distribuída, segundo a perspectiva de Rumelhart et al [24]. Apesar da abordagem sistêmica do trabalho de Kohonen, os seus resultados indicam uma estreita relação com os processos biológicos [25]-[28]. Portanto, o mapeamento de funcões biológicas no cérebro e os Mapas Auto-Organizáveis, SOFMs, possuem características comuns, o que permite chamar este modelo de *inspirado biologicamente*.

2.1.2.Mapeamento da informação em neurocomputação.

Nesta secção serão descritas as características básicas do chamado *mapeamento neural*, segundo a proposta de Amari [20].

Considere-se um conjunto *I* de sinais de informação e uma rede neural com capacidade de formar automaticamente (i.e. sem supervisão) uma representação interna de *I*. Tem-se um *Mapeamento* dos sinais num *campo neural* Φ, quando cada sinal em *I* é representado por uma excitação localizada em uma posição específica de Φ. Entende-se como *campo neural* Φuma rede onde os elementos ou células de processamento são arranjados de alguma forma num espaço, por ex., bi-dimensional (vide fig.2.1), tal como no córtex cerebral. Este processo é chamado na área da neuro-ciência como *Mapa cortical* ou *Mapeamento Neural* da informação. Normalmente, o objeto de estudo destes mapas concentra-se na descoberta das suas propriedades básicas, tais como: estabilidade, resolução, capacidade de armazenamento, etc. [19], [29]-[35].

Considere-se um campo neural Φbidimensional com conexões recorrentes, i.e., onde são incluídas realimentacões desde a saída de algumas células. As posições espaciais dos neurônios em Φ são definidas por:

$$\xi=(\xi_1,\xi_2)$$

onde ξ_1 e ξ_2 definem as coordenadas de posicionamento em Φ(vide fig. 2.2).
O campo Φ recebe um conjunto de estímulos desde *I*, distribuído na rede e expresso por S(ξ).

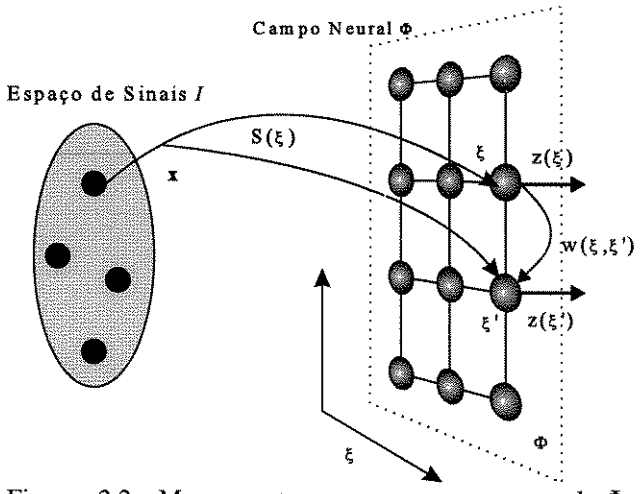


Figura 2.2. Mapeamento em um campo neural Φ, ocasionado pelo estímulo direto S(ξ) da entrada **x** e distribuído a toda a rede. A figura ilustra o efeito na célula localizada na posição em ξ e vizinha da localizada em ξ'.

A figura 2.2 representa um processo de mapeamento produzido pela entrada **x** no espaço *I*, através do estímulo direto distribuído S(ξ) em Φ. A célula em ξ gera um padrão de atividade expresso por z(ξ), que por sua vez, por meio da interação lateral (no caso do modelo de Kohonen) ou por conexão recorrente (no caso do modelamento de Amari) e expresso por w(ξ,ξ'), estimula a célula vizinha localizada em ξ'. A célula em ξ' recebe então o estímulo direto S(ξ) mais o estímulo lateral ou recorrente, que pode ser excitatório ou inibitório, dependendo de condições que serão tratadas posteriormente.

Amari generalizou a sua análise da dinâmica de redes neurais considerando ξ como a posição central de um grupo de neurônios, interagindo com outros localizados em ξ'. Se a expressão u(ξ,t) representa o *potencial médio* de atividade das células localizados ao redor da posição ξ no instante t, então pode-se definir a atividade dos neurônios como a razão ou *taxa de disparo de saída média*, expressa por:

$$z(\xi,t) = f[u(\xi,t)]$$

Definindo w(ξ-ξ') como o **peso** conectando os neurônios em ξ e ξ', Amari posteriormente define a atividade dinâmica da rede como um *padrão de atividade produzido* no campo Φ, por causa do estímulo distribuído S(ξ), onde a condição de equilíbrio deve satisfazer :

$$U(\xi) = w \circ f[U] + S(\xi) \qquad (2.1)$$

onde S(ξ) representa a soma ponderada total de estímulos diretos desde a fonte *I* aos neurônios em ξ. A expressão w ○ f[U] representa a soma total ponderada dos estímulos recorrentes desde os neurônios em outras posições do campo Φ, sendo definida por :

$$w \circ f[U] = \int w(\xi - \xi') \cdot f[u - (\xi',t)]\, d\xi' \qquad (2.2)$$

Desta forma a expressão (2.1) representa o padrão de atividade em equilíbrio provocado pelo estímulo distribuído S(ξ).

Um outro caso considera um sinal excitatório **x** originado em *I* e aplicado ao campo Φ, junto com um sinal inibitório x_0. O estímulo total recebido em ξ é então expresso por:

$$S(\xi;\mathbf{x}) = S(\xi)\cdot\mathbf{x} - s_0(\xi)\cdot x_0$$

A expressão U(ξ;**x**) representa a solução de equilíbrio da equação (2.1), quando o sinal de entrada aplicado a Φ é **x**. No caso das conexões recorrentes serem fortemente inibitórias, dado uma entrada **x**, o equilíbrio só é conseguido através de um padrão de excitação local expresso por U(ξ;**x**). Um padrão de excitação local existe quando ele é restrito a uma pequena região na vizinhança de ξ, ativando um número limitado de células.

Se ξ = m(**x**) representa a posição central do padrão de excitação local em Φ, originado por **x**, então U(ξ;**x**) será positivo somente dentro da vizinhança centrada em ξ. Amari não define como aquela vizinhança será delimitada nem como ela poderia variar dinamicamente durante uma operação de treinamento da rede, mas a sua análise permite verificar que m(**x**) representa o *mapeamento* do sinal de entrada **x** desde o espaço *I* ao campo Φ, i.e., m:*I*→Φ. O conceito pode ser visualizado na figura 2.3.

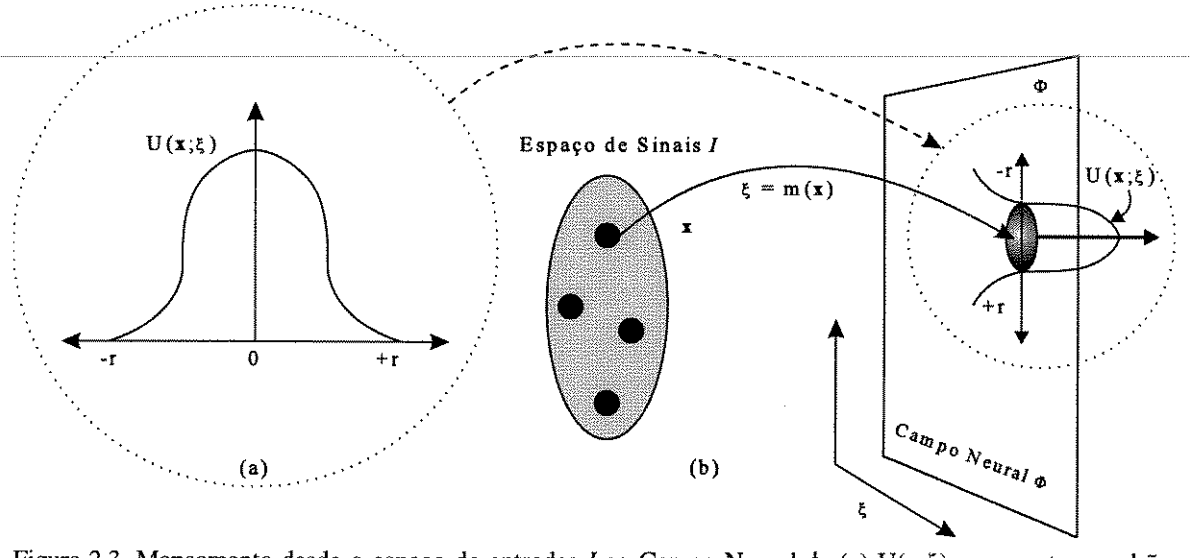


Figura 2.3. Mapeamento desde o espaço de entradas *I* ao Campo Neural ϕ. (a) U(**x**;ξ) representa o padrão de excitação local ou a atividade ao redor de uma vizinhança delimitada por -r e +r e cujo centro é localizado na posição ξ. O mapeamento é representado por ξ=m(**x**).

Na figura 2.3(a) representa-se a padrão de excitação local U(ξ;**x**) no campo neural Φ, originado pelo estímulo devido à entrada **x** presente em *I*, a atividade é restrita à região limitada pelo raio r (-r a +r), cuja posição central r=0 corresponde à posição $\xi = (\xi_1,\xi_2)$ em Φ. Em 2.3(b) representa-se o mapeamento topológico m:*I*→Φ.

As expressões descritas anteriormente formam parte de um conjunto de equações fundamentais derivadas por S.Amari, que governam a dinâmica do processo de auto-organização

de um campo neural Φ. Na sua perspectiva, a topologia bi-dimensional de Φ é representada pelas conexões recorrentes w e a topologia do espaço de sinais *I* é, por sua vez, representada por uma medida da correlação ou similaridade entre os seus sinais.

Tanto Amari quanto outros autores têm analisado o comportamento dinâmico de redes neurais auto-organizáveis. A metodologia varia desde uma análise matemática pura [36],[37] até abordagens algorítmicas e simulacões em computador. A arquitetura do modelo de T. Kohonen se apresenta interessante, já que a sua proposta é computacionalmente simplificada, permitindo intensificar os estudos baseados em simulações. Mesmo assim, a simulação de modelos complexos, compostos por muitas células de processamento e vetores de treinamento e pesos com grande número de componentes, é muitas vezes intratável em máquinas convencionais e o baixo desempenho da implementação em *software* far-se-á inviável para aplicações em tempo real, tal como no processamento de imagens (compressão/descompressão de vídeo digital, etc.). Estes são aspectos que são uma grande motivação para a implementação de modelos neurais auto-organizáveis em sistemas VLSI, tanto em tecnologia digital quanto analógica [38]-[41].

Na secção seguinte será descrito detalhadamente o mecanismo de operação do modelo simplificado de Kohonen, o algoritmo SOFM.

2.2.Mapas auto-organizáveis - o algoritmo SOFM.

Os mapas auto-organizáveis pertencem a uma categoria de redes neurais em que células vizinhas competem em suas atividades por meio de interações laterais mútuas. O aprendizado desta classe de redes é geralmente conhecido como *competitivo*, *não supervisionado* ou *auto-organizado*. Na seqüência, será descrito o mecanismo básico do aprendizado competitivo.

2.2.1. Aprendizado competitivo

No contexto das redes neurais, o aprendizado pode ser definido como um processo onde *os parâmetros livres de uma rede neural são adaptados ao longo de um processo contínuo de estimulação ocasionado pelo seu meio ambiente* [42]. Neste contexto, o aprendizado competitivo tem sido definido como um processo adaptativo, onde células de uma rede são *sintonizadas* às características específicas dos seus sinais de entrada [18]. A resposta da rede é então localizada, o que permite que o modelo seja adequado para detetar as características estatísticas mais relevantes associadas ao conjunto de sinais de entrada, tal como a *densidade de probabilidade*.

Operação

Primeiramente, supõe-se que existe uma seqüência de amostras de um evento observável que pode ser representado de forma vetorial e descrito por $\mathbf{x} = \mathbf{x}(t) \in \mathbb{R}^n$, onde t representa o tempo. Supõe-se um conjunto de vetores de pesos **w** (variáveis) de referência, tal que :

$$\{\mathbf{w}_i(t);\ w_i \in \mathbb{R}^n,\ i = 1, 2, ..., k\}$$

No instante t=0, os vetores $\mathbf{w}_i(0)$ são inicializados de alguma forma apropriada, o procedimento normalmente utilizado é a escolha aleatória.

No processo competitivo, o vetor de entrada $\mathbf{x}(t)$ é comparado simultaneamente com cada $\mathbf{w}_i(t)$, em cada instante sucessivo t, conhecido também como *época* (t=1,2,3,···). A cada t, o vetor $\mathbf{w}_i(t)$ com melhor *casamento* (*best-matching*) com a entrada corrente é modificado, de forma que ele movimenta-se na direção do vetor $\mathbf{x}(t)$. Se a comparação é baseada em alguma métrica de distância $d(\mathbf{x},\mathbf{w}_i)$, a modificação de $\mathbf{w}_i$ deve ser feita de tal forma que, se i = c indica o índice do vetor atualizado, $\mathbf{w}_c$, então $d(\mathbf{x},\mathbf{w}_c)$ deve ser diminuída, enquanto todos os outros vetores de referência $\mathbf{w}_i$, onde $i \neq c$ são inalterados. Desta forma, ao longo do treinamento, distintos vetores de referência tendem a sintonizar-se especificamente a distintos domínios da entrada $\mathbf{x}$.

O processo descrito anteriormente não considera as relações espaciais das células detectoras de características, tal como acontece na maioria dos modelos neurais. A rigor, as coordenadas espaciais (a localização) das respostas não têm relação com o **espaço** das entradas, devido principalmente ao fato de ter somente uma célula ativando-se a cada sinal de entrada $\mathbf{x}(t)$. O algoritmo SOFM original, proposto por Kohonen [16]-[19], considera a **ordem topológica** do espaço de entradas, refletida no mapeamento sobre a rede de células de processamento.

Segundo Tavan et al. [31], para um dado conjunto de dados de entrada caracterizados por uma densidade de probabilidade $p(\mathbf{x})$, o SOFM seria um classificador estatístico auto-organizável e uma memória auto-associativa para vetores característicos $\mathbf{x}$ de dimensão n. Na verdade, os mapas auto-organizáveis armazenam a informação sobre a densidade de probabilidade associada ao conjunto de entradas, $p(\mathbf{x})$. Para Kangas et al. [17] os vetores $\mathbf{w}_i$ tendem a serem localizados no espaço de entrada $\mathbb{R}^n$, de forma que eles aproximam-se de $p(\mathbf{x})$ no sentido de algum *erro residual mínimo*. Esta discussão não será aprofundada neste trabalho, considerando que os objetivos focalizam-se na implementação em *hardware* do modelo.

2.2.2.Mapeamentos organizados

Aspectos gerais.

O modelo de Kohonen tem evoluído a partir de proposta original [4], mas o princípio basicamente permanece inalterado [19], formando-se mapas topologicamente corretos a partir de distribuições estruturadas de sinais, em arranjos uni ou bidimensionais, que inicialmente não possuíam tal estrutura. O esquema da figura 2.4 representa o modelo simplificado de uma rede auto-organizável, cujas características básicas podem resumir-se em :

- Arranjo de células que recebem entradas coerentes desde um espaço de eventos, formando funções discriminantes simples dos sinais de entrada (ex. distância);
- Mecanismo de comparação das funções discriminantes que seleciona uma célula.
- Conexão entre células vizinhas, permitindo interação lateral;
- Processo adaptativo que permite a modificação dos parâmetros das células ativadas, relacionadas com a entrada corrente.

A informação do espaco externo manifesta-se por um conjunto de eventos A1, A2, A3,···. Tais eventos são captados por um conjunto de *unidades de processamento* pertencentes a uma *rede de pré-processamento*, que gera um conjunto de sinais sensoriais ou estímulos.

Os estímulos são distribuídos simultaneamente ao arranjo de células de processamento, não necessariamente idênticas. No entanto, supõe-se que os sinais de entrada a rede são coerentes, no sentido em que eles são determinados única e exclusivamente pelos mesmos eventos A_k(onde k=1,2,3, ... representa o número de eventos apresentados à rede).

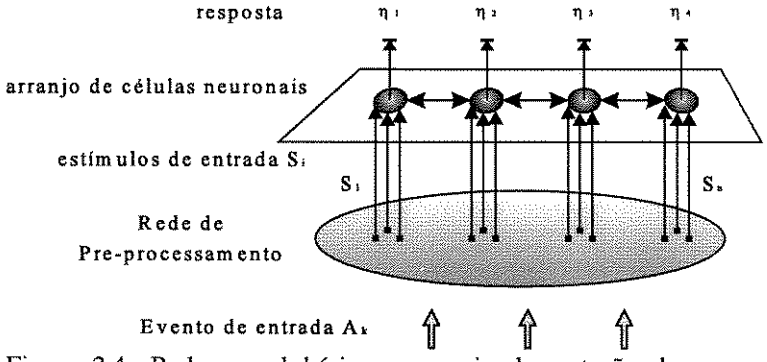


Figura 2.4- Rede neural básica para a implementação de mapas organizados

Considere-se que os eventos A_k possam ser ordenados de acordo com alguma métrica, de tal forma que exista uma relação do tipo $A_1 \Re A_2 \Re A_3 \cdots$, onde $\Re$ representa uma relação de ordenamento transitiva. Suponha-se que as respostas das unidades de processamento aos eventos externos A_k são representadas por valores escalares $\eta_i(A_1)$, $\eta_i(A_2)$, $\eta_i(A_3)\cdots$ (i=no. de neurônios)

Considerando-se um sistema como o ilustrado na figura 2.4. Kohonen definiu como *mapeamento ordenado* (caso uni-dimensional) se, para $i_1 > i_2 > i_3 > \cdots$ cumpre-se que:

$$\eta_{i1}(A_1) = \max_j\{\eta_j(A_1) \mid j=1,2,...,n\}$$

$$\eta_{i2}(A_2) = \max_j\{\eta_j(A_2) \mid j=1,2,...,n\}$$

$$\eta_{i3}(A_3) = \max_j\{\eta_j(A_3) \mid j=1,2,...,n\}$$

...

Kohonen assegura que a definição anterior pode ser facilmente generalizada para arranjos de células n-dimensionais, com $n \geq 2$. Faltando somente a definição de uma ordem topológica para os eventos A_k. Este é um aspecto bastante discutível, não há consenso generalizado para a métrica que definiria quão mais organizado seria um determinado mapa em relação a outro. Estudos determinam que as métricas utilizadas são no mínimo discutíveis [43].

Uma das formas mais comuns de definir a organização é considerar que a topologia do arranjo de células é definida pela relação entre as unidades vizinhas. Quando uma unidade em particular é considerada como a imagem de um evento em particular, diz-se então que o mapeamento é ordenado se as relações topológicas entre as imagens e os eventos são *similares*.

Na figura 2.4, a topologia do arranjo de células é definida pelas relações de vizinhança, podendo assumir diferentes formas, no caso de mapas projetados em planos n-dimensionais.

2.2.3. Algoritmo de ordenamento espacial.

2.2.3.1.Descrição geral

O procedimento descrito em 2.2.1 envolve uma tendência à auto-organização, mas a sua capacidade de ordenamento topológico é de efeito diminuído, a organização é restrita a pequenas regiões na superfície da rede. O algoritmo descrito na seqüência permite a geração de mapas organizados globalmente abrangendo toda a superfície de mapeamento, a arquitetura considerada é baseada no esquema da figura 2.1, onde os mapas são formados em um plano bi-dimensional.

2.2.3.2.Operação do algoritmo básico SOFM

Considerando a arquitetura representada pelas figuras 2.1 e 2.5, onde uma rede é formada por um arranjo bi-dimensional de células de processamento. A cada célula será atribuida um vetor de referência $\mathbf{w}_i$, conhecido também como **vetor de pesos** ou *codebook*. Na sua forma mais simples, um vetor de entrada **x** é transmitido simultaneamente a todas as células, dispostas de forma que implementam-se conexões laterais às vizinhas. Para efeitos algorítmicos, os detalhes de conexão são desprezíveis e somente são levados em conta os seus efeitos funcionais, derivados do algoritmo de aprendizagem dos vetores **w**. Como foi mencionado em 2.2.1, o aprendizado será implementado por um processo competitivo.

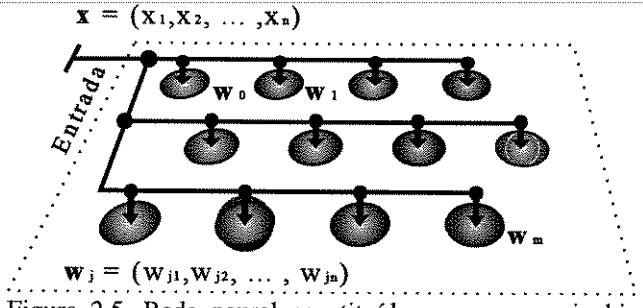


Figura 2.5. Rede neural constituída por um arranjo bi-dimensional de 12 células de processamento. Vetores de treinamento **x** e pesos **w** n-dimensionais.

O diagrama da figura 2.5 representa a arquitetura onde são produzidos os mapas. A entrada é distribuída a todos os elementos adaptativos da rede, cada célula j armazena o vetor-peso $\mathbf{w}_j$ da mesma dimensão do vetor de entrada **x**. A dimensão do vetor **x** depende da dimensão do espaço sensorial (vide secção 2.1). A operação do algoritmo de aprendizado competitivo (treinamento) é dividido em 2 fases principais:

I.	Determinação da célula ganhadora do processo competitivo.
II.	Atualização dos vetores pesos na região da vizinhança da célula ganhadora.

A fase I normalmente utiliza um processo conhecido como WTA (*winner-takes-all*), a fase II é realizada na seqüência, sobre um conjunto de células na vizinhança da célula ganhadora.

I. Determinação da célula ganhadora do processo competitivo.

A célula **c** com o melhor casamento (*best-matching*) entre o seu vetor de pesos e a entrada corrente **x**(t), é determinada de acordo com a equação (2.3), sendo definida como a **ganhadora** do processo competitivo no instante (época) t. O vetor **x**, representando o sinal de entrada, o conjunto de vetores de referência $\mathbf{w}_i \in \mathbb{R}$, e a célula ganhadora **c** relacionam-se por meio da expressão :

$$\|x(t) - w_c(t)\| = \min_i \{\|x(t) - w_i(t)\|\} \tag{2.3}$$

De forma geral, a similaridade entre **x** e **w** pode ser expressa em função de alguma métrica da distância vetorial d(**x**,$\mathbf{w}_j$), a célula ganhadora deve assim ser determinada, tal que:

$$d(x, w_c) = \min_j \{ d(x, w_j) \} \tag{2.4}$$

A métrica mais conveniente sugerida por Kohonen é a *Distância Euclidiana*, embora outras formas de determinação de distância vetorial também possam ser utilizadas. Uma métrica simplificada é, por exemplo, o produto interno $\mathbf{x}^T \cdot \mathbf{w}_j$, onde T é a transposta de **x** e j varia entre 1 e o número total de neurônios da rede. Diferentes métricas adequadas para implementação em *hardware* foram avaliadas neste trabalho, os resultados serão apresentados nos Capítulos 4 e 5.

II. Atualização de pesos da célula ganhadora e das células pertencentes à sua vizinhança.

A necessidade de implementar um processo em que os vetores de referência sejam organizados espacialmente, tem induzido a atualizar os vetores $\mathbf{w}_j$ em blocos definidos em torno da célula ganhadora no instante t, definindo assim uma **vizinhança topológica** $N_c(t)$, centralizada em **c**. No entanto, por causa da sobreposição dos blocos que definem $N_c(t)$ e por causa de correções impostas durante a fase de aprendizado, os valores dos pesos w_j, tendem a serem *suavizados (smoothed)*. Da mesma forma, eles tendem à auto-organização, este aspecto é particularmente notório nas regiões próximas à fronteira dos mapas, sendo o efeito sutilmente visualizado. Tratamentos matemáticos, que tentam explicar formalmente tal característica, assim como análise de estabilidade e convergência do modelo, podem ser consultados em [30]-[37].

A atualização dos pesos w_j é então restrita à vizinhança topológica $N_c(t)$. Esta vizinhança é normalmente variável e definida pela conectividade das células da rede.

A figura 2.6 representa um exemplo de vizinhanca topológica em diferentes instantes, para um caso hexagonal, onde cada célula da rede possui 6 vizinhas imediatas. O conjunto $N_c(t)$ representa o conjunto de células pertencentes à vizinhança de **c**, no instante t, cujo tamanho diminue monotonicamente em função do tempo : na fig. 2.6, $t_0 < t_1 < t_2$.

Considere-se que t varia de forma discreta, a atualização dos pesos é então definida por:

$$\begin{aligned} w_i(t+1) &= w_i(t) + \alpha(t)[x(t) - w_i(t)] \quad se\ i \in N_c(t) \\ w_i(t) &\quad se\ i \notin N_c(t) \end{aligned} \tag{2.5}$$

onde $\alpha(t)$ representa uma função linear ou não linear diminuindo monotonicamente durante o periodo do processo de treinamento, $0 < \alpha(t) < 1$.

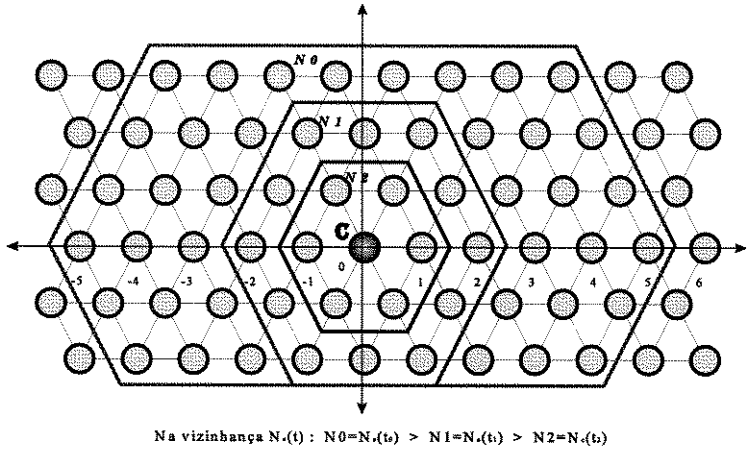


Figura 2.6. Vizinhança topológica $N_c(t)$ do tipo hexagonal para uma rede de 12x6 células. $N0=N_c(t_0)$ é a vizinhança definida no instante t_0. $t_0<t_1<t_2$.

A fim de conseguir um melhor desempenho no ordenamento global da rede, o processo de treinamento inicia-se em $t = t_0$ com um raio amplo para $N_c(t_0)$, geralmente com um valor equivalente a 75%-100% do tamanho total da rede. No fim do processo, para $t \gg t_0$, $N_c(t)$ deveria conter unicamente as células mais próximas da vizinhanca de **c**. Normalmente este último valor corresponde ao raio 1 ou 0, indicando que $N_c(t)$ contém somente as vizinhas imediatas a **c**, ou unicamente a célula ganhadora, respectivamente. O raio de $N_c(t)$ pode diminuir monotonicamente, de forma linear ou não linear, a sua taxa ou velocidade influi notoriamente na qualidade dos mapas finais [33].

As condições mencionadas anteriormente devem ser satisfeitas da melhor forma possível, para assim conseguir os desejados mapas organizados globalmente. Kohonen tem demonstrado o desempenho computacional dos seus algoritmos utilizando um grande número de simulações, cf.[15]-[19], [22]-[23].

2.2.3.3.Algumas Variantes do algoritmo original.

Uma notação alternativa utiliza um *kernel* ou núcleo para representar a influência da distância entre **c** e outra célula em $N_c(t)$. Esta é uma forma de introduzir o modelamento da função de interação lateral entre as células, que pode assumir tanto uma forma fixa quanto variável, dependendo da distância topológica entre células. A atualização dos $\mathbf{w}_i$, pode ser então expressa por :

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \Lambda_{ci}(t)[\boldsymbol{x}(t) - \boldsymbol{w}_i(t)] \qquad (2.6)$$

onde $\Lambda_{ci}(t) = \alpha(t)$ para $i \in N_c(t)$ e $\Lambda_{ci} = 0$, para $i \notin N_c(t)$.

A fim de introduzir a não-linearidade característica na interação lateral de redes neurais biológicas, a função $\Lambda_{ci}(t)$ pode ser representado por :

$$\Lambda_{ci} = h_0 \exp\left(\frac{-\|r_i - r_c\|^2}{\sigma^2}\right) \tag{2.7}$$

onde $h_0=h_0(t)$ e $\sigma=\sigma(t)$ são funções adequadas decrescentes no tempo.

No caso de utilizar-se vetores de entrada n-dimensionais, com "n" muito grande, muitas vezes sugere-se a sua normalização. Embora a normalização pode melhorar a precisão numérica, devido à padronização da sua faixa dinâmica, este não é um processo absolutamente necessário.

Um outro aspecto relaciona-se com a medida de distância : as regras de atualização dos vetores de referência devem ser compatíveis, respeito à métrica. Caso da utilização do produto interno para a determinação da célula **c** (2.9), então a regra de atualização dos vetores pesos(2.5) deve ser modificada, sendo definida pela expressão (2.8).

$$w_i(t+1) = \begin{cases} \dfrac{w_i(t) + \alpha_0(t)x(t)}{\|w_i(t) + \alpha_0(t)x(t)\|} & se\ i \in N_c(t) \\ w_i(t) & se\ i \notin N_c(t) \end{cases} \tag{2.8}$$

Devido à necessidade da normalização a cada passo de operação (época), a expressão (2.8) aumenta consideravelmente a complexidade computacional da atualização. Por outro lado, a implementação de divisão tem um alto custo em silício, não sendo conveniente para uma implementação em *hardware* digital. No entanto, a determinação de **c** pode ser simplificada:

$$x^T(t)w_c(t) = \max_i\{x^T(t)\ w_i(t)\} \tag{2.9}$$

Na expressão (2.8), $\alpha_0(t)$ é uma função decrescente onde $0<\alpha_0(t)<\infty$. Simulações realizadas por Kohonen et al.[16],[17] têm sugerido a utilização da expressão $\alpha_0(t)=\alpha_0/t$, com α_0 variando entre 10 e 100, o que teria produzido resultados adequados.

2.2.4.Vizinhanças definidas de forma dinâmica.

2.2.4.1. Fundamentos.

Um arranjo linear de células (uni-dimensional) é definido como uma topologia simples de ordem zero[17], onde cada célula possui duas vizinhas, com exceção das pertencentes aos extremos do arranjo. Um arranjo bi-dimensional de células define *topologias planares*. Tais topologias podem utilizar diversas estruturas regulares, a fim de definir uma vizinhança. Duas estruturas utilizadas freqüentemente neste caso são a retangular e a hexagonal.

Para o caso de estruturas retangulares é possível usar pelo menos dois tipos de topologias de vizinhança, definidas pela orientação do retângulo, embora a forma exata da vizinhança não seja muito relevante.

No caso de estruturas hexagonais (fig.2.6), cada uma das células é diretamente conectada a seis células vizinhas, a forma da vizinhança define o hexágono. Para todos os casos, sempre é possível definir a vizinhanca em função de um raio circular particular ao redor da célula ganhadora **c**.

Quando a distribuição das entradas p(**x**) não têm uma forma regular, com *clusters* concentrados em distintas regiões do espaço de entrada, os resultados das computações produzidas pelo algoritmo descrito em 2.2.3, usando a expressão (2.5) tendem a concentrar-se em uma fração das células do mapa. Os vetores **w** correspondentes a área de p(**x**)=0 podem ser influenciados pelos vetores de entrada **x** das áreas circundantes com p(**x**)≠0. As flutuações se interrompem assim que o tamanho de $N_c(t)$ diminui, mas o efeito é de que algumas células no mapa permanecem isoladas. Em contrapartida, quando a distribuição p(**x**) é mais uniforme, o conjunto de vetores de referência adapta-se de melhor forma às entradas [17], [27], [33].

Este problema motivou a procura de novos mecanismos que permitissem definir de forma adaptativa as relações de vizinhanca $N_c(t)$, durante o período de treinamento. Desta forma, foi proposto um algoritmo alternativo conhecido como MST (*Minimal Spanning Tree*) [17],[28].

2.2.4.2. O Algoritmo MST.

O algoritmo MST consiste na construção de grafos ou árvores, onde cada um dos nós é conectado por um único arco, minimizando a soma do comprimento total dos arcos. No MST, os comprimentos dos arcos são representados por *normas Euclidianas* não ponderadas das diferenças vetoriais dos vetores de referência **w**. Desta forma, a topologia de um MST define a vizinhanca de uma célula ao longo dos arcos que surgem daquela célula. A figura 2.7 representa um exemplo de vizinhança definida por MST.

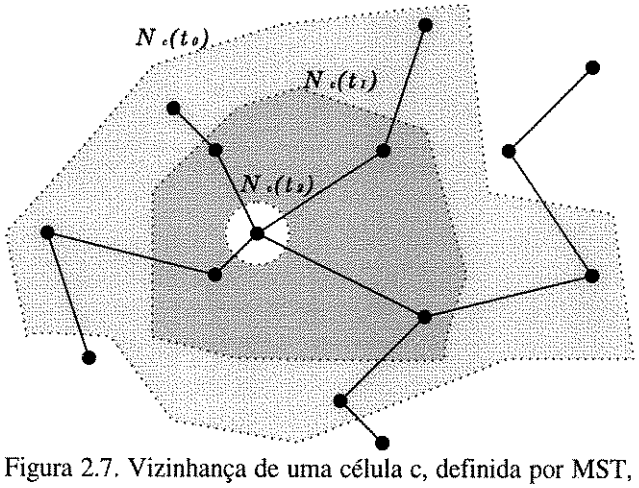


Figura 2.7. Vizinhança de uma célula c, definida por MST, $t_0<t_1<t_2$.

Da mesma forma que no algoritmo original SOFM, o processo é inicializado com uma vizinhança ampla, incluindo uma grande quantidade de células. No MST, isto significa incluir um grande número de arcos, percorrendo o grafo a partir da célula **c**, construindo assim a vizinhança $N_c(t)$. A medida que a vizinhança $N_c(t)$ diminui , a topologia do MST simplifica-se.

Kohonen assegura em [17] que tal processo auto-organizável não produz necessariamente mapeamentos globalmente organizados. Em contrapartida, a fase de aprendizagem pode ser significativamente acelerada, minimizando-se o número de passos de aprendizagem. É possível conferir os resultados de variadas simulações, utilizando MST, na ref. [17].

2.2.5. Conclusões parciais.

Foram descritos os algoritmos básicos propostos por Kohonen para a realização de Mapas Auto-Organizáveis mais a descrição parcial de algumas variantes. Um algoritmo modificado alternativo não considerado aqui é o chamado algoritmo LVQ de quantização vetorial (LVQ-*Learning Vector Quantization*). Os algoritmos LVQ foram propostos para a realização de tarefas de reconhecimento de padrões, sendo na verdade uma versão supervisionada do SOFM, a fim de realizar uma sintonia fina dos mapas resultantes. Tanto o MST quanto o LVQ parecem demonstrar melhor desempenho em tarefas mais dedicadas (reconhecimento de padrões) ou para casos com distribuição de probabilidades de entrada particulares (ex. *clusters* isolados). No entanto, a melhora é realizada ao custo de maior complexidade computacional, induzindo a implementação de estruturas de *hardware* bastante mais complexas. É ainda necessário conferir a efetividade dos algoritmos alternativos com simulações exaustivas e verificar a sua relação custo-benefício em relação à sua implementação em *hardware*, além disso é também necessária uma análise profunda das suas características de estabilidade e convergência. A simplicidade computacional do algoritmo SOFM original torna-o particularmente atraente para a implementação em sistemas e circuitos VLSI, permitindo a construção de protótipos utilizando tanto tecnologia digital quanto híbrida [38]-[40].

Capítulo 3

Análise Algorítmica do SOFM

Escopo

Neste capítulo é realizada uma análise do algoritmo básico apresentado no capítulo anterior. O estudo é conduzido visando primeiramente a sua implementação em *software*. Um simulador com interface gráfica foi desenvolvido antes de definir os aspectos relativos à implementação VLSI por uma rede de neuroprocessadores dedicados em tecnologia digital.

A análise tem como objetivo identificar os parâmetros mais críticos que influem no desempenho. Um primeiro aspecto é a identificação dos problemas que surgem ao realizar simulações numa máquina seqüencial e como deduzir certas características de convergência do modelo. Esta primeira fase experimental de implementação visa também descobrir possíveis aplicações em tempo real, além de definir as condições sob as quais o algoritmo poderia ser modificado para realizar a sua implementação em *hardware*, identificar as vantagens e detectar pontos críticos.

No final deste capítulo apresenta-se uma análise de complexidade do algoritmo, visando a sua implementação em um processador de arquitetura paralela.

3.1. Introdução

O algoritmo básico SOFM descrito no capítulo anterior não foi proposto pensando na sua implementação ou mapeamento em *hardware*, seja digital, analógico ou *mixed-mode* (híbrido). Neste capítulo será realizada uma análise teórica do algoritmo, identificando os aspectos mais relevantes relacionados com o seu desempenho, principalmente em termos da sua estabilidade e da sua taxa de convergência. A análise é motivada pela futura implementação em *hardware*. De forma geral, os algoritmos neurais não são desenvolvidos visando a sua implementação em algum tipo de *hardware*, seja óptico ou eletrônico, menos ainda numa determinada tecnologia. É necessário modificá-los e adequá-los às necessidades do *hardware*, de forma que os circuitos os implementem em condições de uso real, otimizando seu desempenho e/ou área de silício, dependendo dos requisitos, especificações e características de operação.

Um dispositivo de classificação automática deve ter a capacidade de construir categorias ou classes, a partir de uma coleção de padrões que lhe são apresentados. Um dispositivo de classificação em tempo real deve ter a capacidade de realizar esta tarefa à taxa ou à velocidade de chegada dos padrões. O modelo de Kohonen pode ser treinado *on-line* de forma a construir tais classes, mas o sistema deve atualizar o seu estado interno, processando e armazenando novas classes assim que são recebidas na sua entrada. Para conseguir este objetivo, algumas operações do algoritmo SOFM devem ser levemente modificadas afim de definir adequadamente os parâmetros que determinan o processo de aprendizagem. Antes de realizar tais modificações, deve-se realizar um cuidadoso estudo que permita identificar adequadamente quais são os parâmetros que necessitam e podem ser modificados sem alterar o comportamento do algoritmo, além de detectar os aspectos que podem modificar o seu desempenho.

Existem numerosos estudos teóricos em relação às propriedades e às características de convergência do algoritmo de Kohonen [30]-[37]. No entanto, até o momento não é fácil definir o que é que exatamente tem sido formalmente provado, devido fundamentalmente à diversividade de métodos matemáticos utilizados [36]. Devido à carência de uma estrutura matemática que prove formalmente as suas propriedades, análises rigorosas do modelo são normalmente fundamentadas em resultados de simulações. Até agora, uma análise completa para o caso multi-dimensional é inconclusa, somente pôde ser conseguida para o caso uni-dimensional, isto é, um espaço de entrada de dimensão 1, e sob a restrição da rede ser linear, i.e., todas as unidades estão dispostas num arranjo uni-dimensional [26]. A primeira prova completa de convergência foi obtida para o caso de distribuição do espaço de entrada uniforme e para uma vizinhança definida por uma função discreta, posteriormente apareceram publicações para casos mais gerais de distribuição de entrada. Os efeitos de diversos tipos de função de vizinhança foram estudados posteriormente. Ritter & Schulten [30] derivaram equações para o caso de uma e duas dimensões, mesmo assim as equações têm solução só para condições de contorno especiais. Erwin et al. [34], [35], detectaram condições para a existência de estados **estacionários meta-estáveis**, mas só para o caso uni-dimensional. No seu estudo pode-se verificar a influência da forma e da faixa da função de vizinhança, assim como a taxa de convergência do algoritmo, produto das mesmas funções. Eles concluíram que a existência (ou

não) de estados metaestáveis influenciam notoriamente no tempo de convergência, que é muito sensível a pequenas mudanças na função de vizinhança. Na figura 3.1. ilustra-se uma forma de representação para um espaço de estados tri-dimensional. Os vales representam pontos de mínima energia ou pontos de atração, semelhante a um *atractor* de uma *memória associativa* , ou *memória endereçada por conteúdo*. A esfera, representando o estado corrente do sistema, é atraída dinamicamente pelos *atractors*.

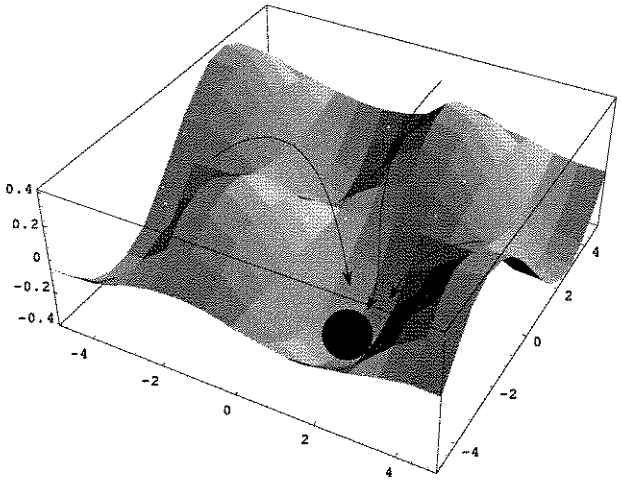


Figura 3.1. Superfície 3-D de estados meta-estáveis.

É importante destacar que até o momento não tem sido garantida a estabilidade do modelo de Kohonen, para o caso geral. Tanto Kohonen quanto Ritter e Schulten demonstraram que no *estado ordenado* e somente para o caso **uni-dimensional** o algoritmo SOFM é estacionário e estável. Uma análise formal da estabilidade está fora do escopo deste trabalho.

3.2. Definições.

Primeiramente, é necessário definir os conceitos mais relevantes relacionados com o estudo de redes neurais. Devido ao fato das redes neurais não serem programáveis, mas **treinadas**, devem ser definidas as regras de **aprendizagem** do modelo. No caso de SOFMs, devem ser principalmente definidas: a topologia da rede, a função de ganho $\alpha(t)$ e a função N(t) que define o *raio de vizinhança* ao redor da *célula ganhadora* no processo de treinamento (vide capítulo 2). Na sequência, descrever-se-á como as funções mencionadas afetam a aprendizagem da rede e a operação do algoritmo em termos do seu desempenho computacional.

3.2.1. Aprendizagem.

Num organismo biológico, tem se definido como aprendizagem a auto-modificação do organismo, como resposta a um ambiente complexo e variante. Em modelamento de redes neurais, define-se como *aprendizagem neuronal* o ajuste das conexões (sinapses) por meio de um procedimento sistemático, a fim de melhorar a sua capacidade de atuação num determinado

ambiente. Mais especificamente, os pesos das conexões sinápticas são modificados, a fim de satisfazer um conjunto de regras pré-definidas. No modelo SOFM, podemos traduzir isto como a capacidade contínua de adaptar-se aos padrões recebidos, sendo que cada padrão é classificado e mapeado na superfície ou espaço da rede, definido pelos componentes dos seus vetores pesos. Quando as regras de adaptação dos pesos são formalizadas, é criado um **algoritmo de adaptação**, ou de aprendizagem. É necessário mencionar que não é possível definir uma regra de aprendizagem geral para todos os modelos neurais existentes, logo, cada modelo define suas próprias regras, principalmente em função da sua topologia.

De modo geral, podem-se definir dois tipos de aprendizagem : supervisionada e não supervisionada.

3.2.1.1. Aprendizagem supervisionada.

Refere-se a modelos onde a aprendizagem é de alguma forma regulada pela presença de um supervisor, tal como uma função de minimização de erro ou custo. Este mecanismo permite condicionar a modificação dos pesos em função da reposta da rede, que é constantemente comparada à saída desejada durante a fase de aprendizado. Como exemplos clássicos de aprendizagem supervisionada, podem-se mencionar os modelos *Adaline* (*Adaptive Linear Element*) de Widrow e Hoff [14] (vide Fig. 3.2) e o *Perceptron* (vide Cap.1), de Rosenblatt [12].

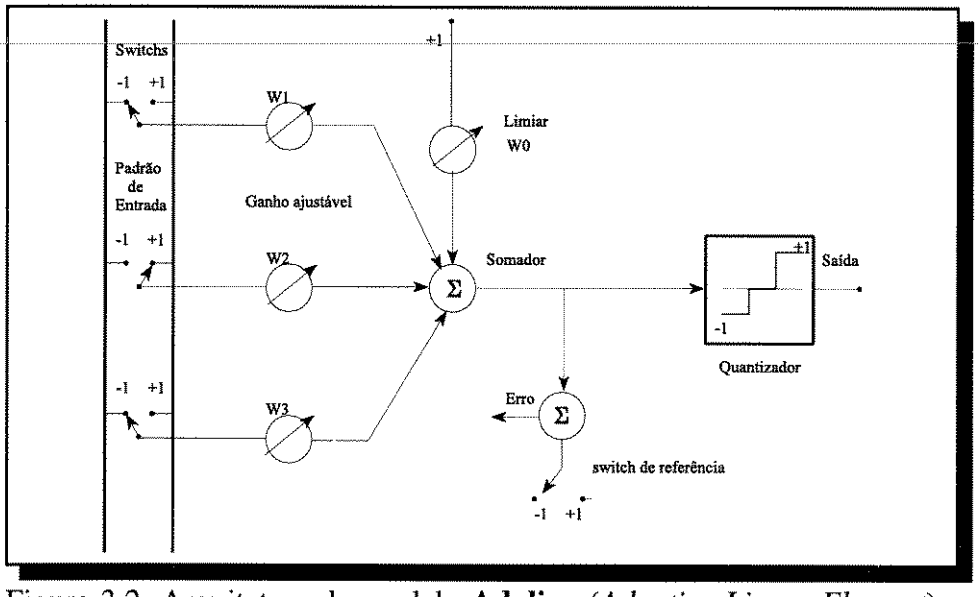


Figura 3.2. Arquitetura do modelo **Adaline** (*Adaptive Linear Element*)

No algoritmo proposto por Rosenblatt, o erro era minimizado em função da saída de cada unidade de resposta. Na proposta posterior de Widrow e Hoff, chamada de algoritmo LMS (*Least Mean Square*), o erro é minimizado em função da soma de todas as unidades de saída.

Logo foi demonstrado por Minsky e Papert em 1969 que os *Perceptrons* de uma camada estavam sujeitos a muitas restrições, eles demonstraram através de uma cuidadosa análise matemática que o modelo não podia representar uma simples função *or-exclusive* [44]. Apesar

da demonstração satisfatória por parte de Minsky e Papert, de que modelos de mais de uma camada com conexões *feed-forward* podiam superar muitas restrições, eles não apresentaram uma solução ao problema do ajuste dos pesos das unidades pertencentes às camadas escondidas, i.e., o aprendizado não foi rigorosamente definido.

Como metodologias de aprendizagem supervisionada são exemplos os chamados *métodos exatos*, como a *regra de projeção*, o *método do gradiente* (método de Newton), do *gradiente estocástica* e outras como as redes multi-camadas, tal como o conhecido modelo *Back-Propagation* [14], [24], [45].A regra de aprendizagem *back-propagation* foi proposta por Rumelhart, Hinton e Williams em 1986 e pode ser considerada como uma generalização da *regra Delta* para funções de ativação não-lineares e redes multi-camadas [24].

3.2.1.2. Aprendizagem não supervisionada.

Nos modelos discutidos na secção anterior, o treinamento é feito de forma de realizar um mapeamento $\mathcal{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, onde um conjunto de padrões de entrada $\mathbf{x}^p$ geram uma resposta da rede do tipo $\mathbf{o}^p = \mathcal{F}(\mathbf{x}^p)$, onde a saída desejada é condicionada por um erro que é realimentado na entrada $\mathbf{x}$. No entanto, há casos onde não estão disponíveis os dados de treinamento, na forma de pares de entrada e saída desejados, como $(\mathbf{x}^p,\mathbf{o}^p)$. Nestes casos a única fonte de informação disponível é o conjunto de padrões de entrada $\mathbf{x}^p$. Portanto, a informação relevante deve ser extraída dos mesmos padrões de treinamento. Alguns exemplos desta classe de problemas são:

- *Clustering* : Os dados de entrada podem ser agrupados em *clusters*. O sistema de processamento deve decidir como agrupar os dados, criando **classes** adequadas.

- *Quantização Vetorial* : Consiste na discretização de um espaço de entrada contínuo. Vetores n-dimensionais são apresentados na entrada do sistema e a saída deve ser uma representação otimizada discreta do espaço de entrada.

- *Redução da dimensionalidade* : Os dados de entrada devem ser agrupados num sub-espaço de menor dimensão. O sistema deve realizar um mapeamento ótimo, tal que a variança dos dados de entrada seja preservada no conjunto de saída.

- Extração de características de um conjunto de sinais de entrada. Este caso resume-se freqüentemente a um problema de redução da dimensionalidade.

No tipo de redes não supervisionadas, não há qualquer tipo de função ou mecanismo que monitore a resposta do sistema e o realimente à sua entrada a fim de *melhorar* o seu desempenho. A rede deve ser capaz de gerar respostas satisfatórias em função da sua própria topologia e do seu algoritmo de aprendizagem, sem ser auxiliada por qualquer mecanismo de correção de erro ou algo similar. Os exemplos mais claros de este tipo de modelos são os *Auto-organizáveis* e as *memórias associativas*, especialmente na sua variante **auto-associativa**.

O aprendizado competitivo é um procedimento onde um conjunto de padrões de entrada é dividido em *clusters* que são inerentes aos dados. O aprendizado é implementado unicamente pelos próprios vetores de entrada, tal como no algoritmo SOFM descrito no capítulo anterior.

O procedimento de aprendizado com *clusters* pode ser visualizado na fig. 3.3, para o caso de padrões de entrada e de pesos tri-dimensionais. Numa rede onde todas as unidades de saída **i** estão conectadas a todas as unidades de entrada **j** através de um peso $\mathbf{w}_{ij}$, existe **uma** célula ganhadora no processo competitivo. Uma vez que a célula ganhadora é determinada, ela atualiza o seu peso em função de alguma regra. A atualização do vetor de pesos $\mathbf{w}_i$ produz efetivamente a rotação do vetor na direção da entrada corrente **x**. Cada vez que é apresentada uma entrada **x**, o vetor de pesos selecionado na competição é rotacionado novamente na direção da entrada. Desta forma, os pesos são dirigidos na direção onde existe grande concentração de pontos,i.e, os *cluster*s na entrada.

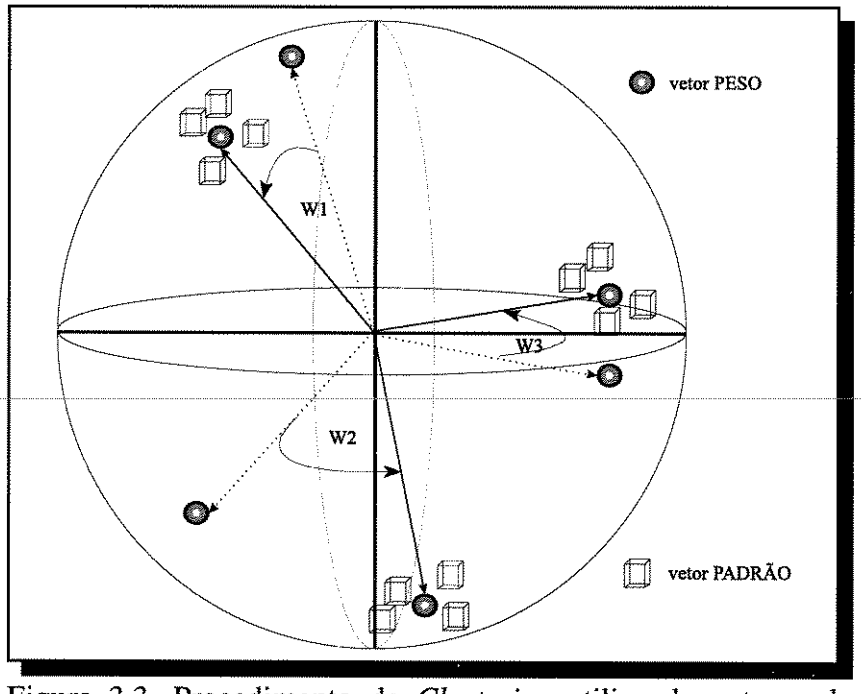


Figura 3.3. Procedimento de *Clustering* utilizando vetores de entrada e pesos normalizados numa esfera de raio unitário.

Na Fig. 3.3., assume-se que todos os vetores estão normalizados, de forma que a magnitude dos vetores restringe-se ao raio de uma esfera de raio unitário. Os algoritmos de atualização podem ocasionar falhas no caso de trabalhar com vetores não normalizados, devido a que a métrica para escolha dos vetores ganhadores durante a competição, pode determinar vetores que não são necessariamente corretos. Além da *distância Euclidiana*, definida no capítulo anterior, é possível utilizar, por exemplo, o produto escalar ou alguma outra métrica.

3.3. Avaliação do desempenho do SOFM.

3.3.1. Introdução.

Normalmente os mapas de características topológicas são implementados numa superfície composta por um arranjo de células ou unidades de processamento básico localizadas num plano bi-dimensional. Como foi mencionado no capítulo anterior, a resposta de uma célula a um padrão determinado **x**(t) é proporcional à similaridade entre o seu vetor de peso **w** e a entrada corrente. A resposta total do sistema é um padrão de atividade localizado, i.e., o vetor n-dimensional **x** ocasiona uma resposta num determinado setor da superfície do plano. Desta forma, **x** se sintoniza com **w** e com as unidades da sua vizinhança topológica.

Um aspecto crítico e influente na formação do mapa se refere à forma como a célula ganhadora interage com as outras pertencentes àquela vizinhança. As funções que determinam o raio de vizinhança, o fator de influência sobre as células vizinhas (inibição lateral) e o ganho $\alpha(t)$ são fundamentais no desempenho do algoritmo SOFM. Foram estudadas a influência de cada um destes fatores.

3.3.2.Função de inibição lateral contínua.

A atividade da célula ganhadora no processo competitivo, pode ser definida por :

$$\eta_i = \sigma(\Sigma_j w_{ij} x_j) \qquad (3.1)$$

Onde η_i representa a resposta da unidade **i**, localizada no mapa representado num plano bi-dimensional, **j** representa j-ésimo elemento do vetor e a função $\sigma(.)$ corresponde à função de ativação *sigmóide*, definida por :

$$\sigma(x) = \begin{cases} 0 & , se\ x \le \delta \\ (x-\delta)/(\beta-\delta) & , se\ \delta < x < \beta \\ 1 & , se\ x \ge \beta \end{cases} \qquad (3.2)$$

A função $\sigma(x)$ introduz uma não-linearidade na resposta, limitando a saída à faixa [0,1], sendo o **x** limitado a valores entre os limiares δ e σ.

Para compreender melhor o modelamento proposto por Kohonen [19], consideremos uma rede simples composta por células dispostas num arranjo linear (vide Fig. 3.4) e conectadas entre si lateralmente por meio de ligações excitatórias e/ou inibitórias.

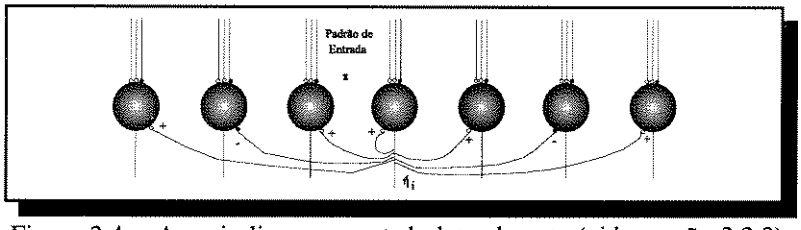


Figura 3.4. - Arranjo linear conectado lateralmente (vide secção 3.3.3).

3.3.3. O fator de realimentação lateral.

Estima-se que existem aproximadamente 10.000 conexões por cada neurônio de um sistema biológico, no neurocórtex. Um modelamento muito simplificado de parte de uma rede pode ser visualizado na figura 3.4. Neste diagrama, cada célula principal recebe uma série de conexões das células pertencentes à sua vizinhança. Existe evidência de que existe uma forte interação entre estas células vizinhas, definindo o que é chamado de *resposta por realimentação lateral*. Podem-se distinguir três regiões principais neste tipo de interação:

- *Interação lateral de curta distância* : Esta é uma região onde as conexões majoritárias são do tipo **excitatórias**. Se estende num raio de 50 a 100 μm nos cérebros de espécimes consideradas *primatas* (ex. animais mais *evoluídos*, como o macaco).

- *Área de inibição*: Consiste de uma área coberta por um raio entre 200 a 500 μm que circunda a área excitatória. Nesta região a ação é principalmente **inibitória**..

- Área excitatória de intensidade menor : Toda a área inibitória é por sua vez circundada por uma área excitatória de menor intensidade em relação à primeira. Segundo alguns estudos, esta área pode atingir vários centímetros.

O fator de realimentação lateral pode ser matematicamente modelado pela soma de duas funções de tipo *Gaussiana*, definindo uma chamada de **função do chapéu mexicano** (*mexican hat*).

$$\gamma_k = (1 - \gamma)\, e^{-\frac{1}{2}\left(\frac{\lambda}{\sigma_E}\right)^2} - \gamma\, e^{-\frac{1}{2}\left(\frac{\lambda}{\sigma_I}\right)^2} \qquad (3.3)$$

Onde γ_k representa o nível de interação lateral do neurônio k em função da distância λ aos vizinhos na superfície do mapa, neste caso definido num plano uni-dimensional (linear).

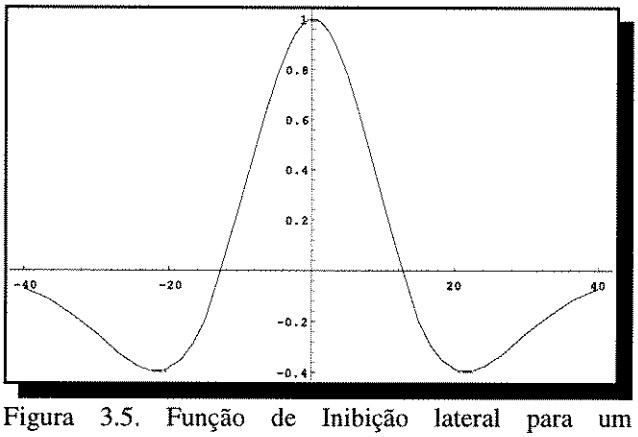


Figura 3.5. Função de Inibição lateral para um modelamento biológico, plano uni-dimensional.

O fator γ representa a relação de amplitude entre as duas funções gaussianas, σ_E a variância da gaussiana excitatória e σ_I a variância da gaussiana inibitória. A forma da função definida por (3.3) é apresentada na fig. 3.5, onde é possível visualizar parcialmente duas das 3 regiões de influência descritas anteriormente: um região excitatória de curta distância de máxima intensidade e uma região de ação inibitória.

Se considerarmos o caso real de neurônios distribuídos num plano, podemos redefinir a equação (3.3) tornando-a extensiva ao espaço tri-dimensional. Desta forma podemos visualizar como os neurônios interagem num plano bi-dimensional. Para este caso, a forma da curva de inibição lateral é representada pela superfície apresentada na Fig. 3.6.

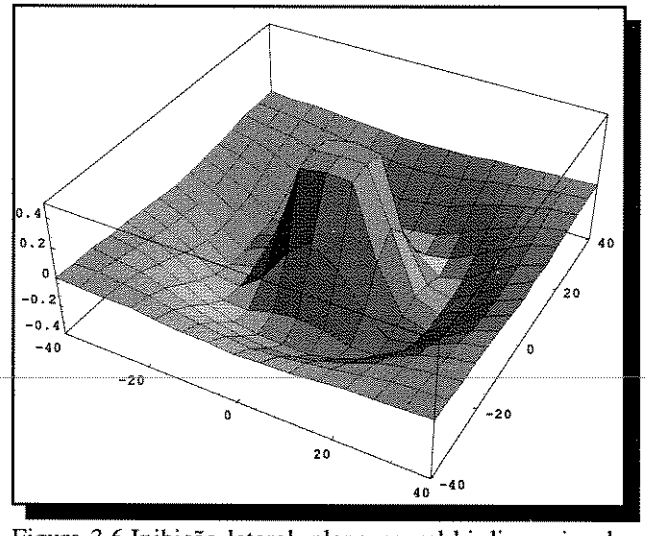


Figura 3.6.Inibição lateral, plano neural bi-dimensional.

A fig. 3.5 foi construída tomando a equação (3.3) e adicionando uma variável ao plano linear definido pela variável que representa a distância λ, definindo desta forma um plano neural bi-dimensional.

3.3.4. Modificação da função de interação lateral.

Tanto para efeitos de simulação em computador como para a realização em *hardware*, deve-se pensar na modificação das funções definidas por (3.3) e suas variantes. A implementação destas funções não-lineares é cara tanto em termos de velocidade e desempenho para implementação em *software* quanto em termos de área de silício, para implementações em *hardware* digital. Neste aspecto os circuitos analógicos revelam uma ampla vantagem, devido à exploração das suas características não lineares, o que permite a realização de circuitos bem mais compactos que os seus equivalentes em digital. Devido ao que este trabalho aborda unicamente a implementação digital do modelo, será estudado somente o aspecto relativo à implementação discreta desta função usando tal tecnologia. É também esta a oportunidade de mencionar como as implementações em modo *mixed* ou híbridas (analógica + digital) conseguem explorar as capacidades de ambos os tipos de tecnologia.

A maior parte das simulações de Kohonen [19] foram realizadas com a função de interação lateral na sua versão discreta, da forma como apresentada na fig. 3.7.

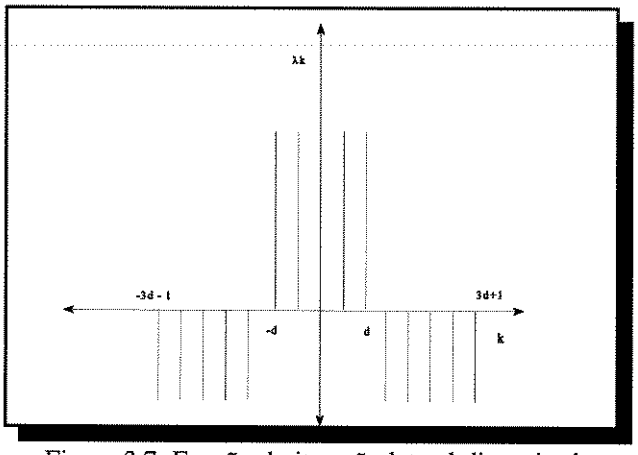


Figura 3.7. Função de iteração lateral discretizada

Dos resultados das suas simulações, detectaram-se claramente as regiões de atividade em torno da célula ganhadora. Ao redor dela forma-se uma chamada *bolha de atividade*, cujo tamanho é definida pelas características da função de interação lateral.

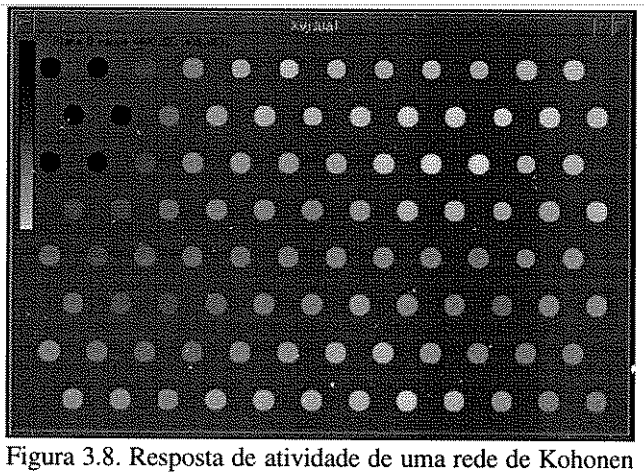


Figura 3.8. Resposta de atividade de uma rede de Kohonen de 12 x 8 células, num determinado instante *t*.

A figura 3.8 representa o resultado de uma simulação num determinado instante e ilustra a forma como a bolha de atividade está distribuída num plano bi-dimensional. A figura representa um arranjo de 12 x 8 células, definindo uma topologia de vizinhança hexagonal. A intensidade de luminosidade indica o grau de contribuição da atividade de cada célula, as células mais claras representan um índice de atividade maior. O nível de atividade é definido por:

$$\eta_i(t) = \sigma\left(\sum_j \mu_{i,j}\ \xi_j + \sum_k \gamma_{k,i}\ \eta_k(t - \Delta t)\right) \qquad (3.4)$$

Onde $\eta_i(t)$, representa a atividade da célula **i**, no instante de tempo t.$\gamma_{k,i}$ o grau de interação lateral desde a unidade **k** à unidade **i**, e $\eta_k(t - \Delta t)$ representa a atividade da unidade **k** durante o instante de tempo $(t - \Delta t)$, anterior ao corrente t. A expressão (3.4) representa a evolução da resposta da célula em função do tempo.

Segundo simulações realizadas por Miikkulainen [46], o efeito primário da inibição lateral é acentuar o contraste entre áreas de alta e baixa atividade. Se o raio de área de inibição lateral é equivalente à resposta inicial localizada da rede, a resposta é continuamente centrada ao redor da unidade com resposta máxima, após sucessivas iterações de (3.4). A medida que aumenta o grau de inibição lateral, em relação à excitação, a resposta estável final é mais focalizada, podendo este efeito ser visualizado nas figuras 3.9a e 3.9b.

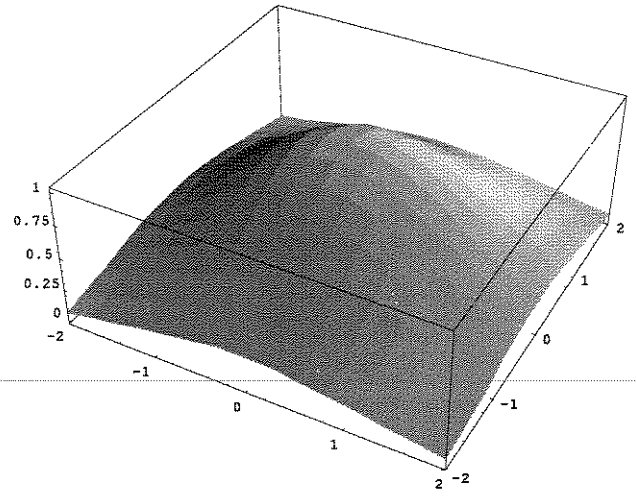


Figura 3.9a.- Resposta Inicial (t=0)

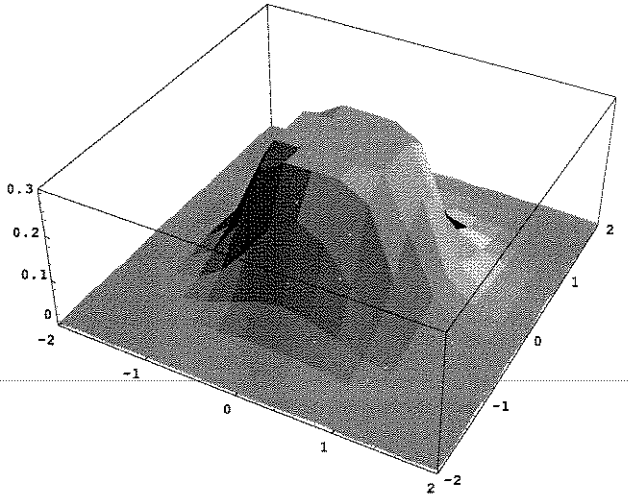


Figura 3.9b. - Resposta final (t >> 0)

Nas figuras acima, o eixo *z* representa o nível de atividade expressado por (3.4), os eixos *x-y* formam o plano onde estão localizadas as células neuronais, a unidade com a resposta máxima localiza-se na posição (0,0) deste plano.

A partir dos resultados das simulações de Kohonen e Miikkulainen, confirma-se a influência da interação lateral. No caso de forte excitação lateral, a bolha apresenta-se mais larga, enquanto que o aumento da inibição lateral em detrimento da influência excitatória, gera uma *bolha de atividade* de forma mais pronunciada.

3.3.5. Influência da Vizinhança.

Além do fator de interação lateral, outro fator que exerce influência relevante no desempenho do algoritmo SOFM é a forma como é determinado dinamicamente o conjunto de células vizinhas à ganhadora. No capítulo 2, este conjunto é definido por $N_c(t)$, uma função monotonicamente decrescente que determina quais são as células que modificam o seu vetor de pesos, para cada instante do aprendizado. Segundo os estudos de Lo e Bavarian [33], é determinante a escolha de uma função de vizinhança adequada, no lugar de um conjunto uniforme. Eles propuseram o uso de uma função de interação dependendo do tempo e da

distância lateral, de forma semelhante à função de interação lateral definida na secção anterior. Tal função assume a forma *gaussiana* seguinte :

$$\Lambda_i(i,t_k) = c \cdot d\ EXP(-\frac{h(i-I)^2}{2\sigma^2}),\quad para\ i \in N_c(t_k) \tag{3.5}$$

onde c,d e h são constantes e a variância σ é definida como o *conjunto de vizinhança* :

$$\sigma = N_c(t_k) \tag{3.6}$$

Os parâmetros da equação (3.5) devem ser determinados de forma que Λ_i esteja limitado a valores entre 0 e 1. A diferença fundamental com a função definida na secção 3.3.4. é a inclusão do tempo na avaliação, sendo que a variável t_k representa um instante no domínio discreto. Em ambos os casos, as funções são decrescentes, tanto no domínio espacial quanto no temporal, desta forma produz-se a propriedade de *focalização* do modelo. A regra de aprendizado, definida pela expressão (2.6), pode ser então generalizada a :

$$\begin{aligned} m_i(t_k+1) &= m_i(t_k) + \alpha(t_k)\ \Lambda_I(i,t_k)\ [x(t_k) - m_i(t_k)] \quad \forall\ i \in N_c(t_k) \\ m_i(t_k+1) &= m_i(t_k) \qquad\qquad\qquad\qquad\qquad\quad \forall\ i \notin N_c(t_k) \end{aligned} \tag{3.7}$$

A influência nas características de convergência do algoritmo SOFM por meio das expressões definidas por (3.5), $\alpha(t_k)$, $N_c(t_k)$ será estudada experimentalmente através de simulações. Até o momento, não existe um procedimento formal que permita prever o desempenho ou a estabilidade do algoritmo, que sirva como informação na escolha de parâmetros adequados.

Todas as simulações realizadas neste trabalho, em linguagem de alto nível (ANSI-C) e posteriormente em VHDL, no modelamento em diversos níveis de abstração da arquitetura proposta no capítulo 4, foram realizadas utilizando a expressão (3.7), no estudo do efeito da função de ganho $\alpha(t_k)$, a expressão (3.5) foi igualada a 1. Antes de descrever detalhadamente as simulações, serão analisados alguns aspectos relativos às entradas utilizadas para o treinamento do modelo.

3.3.6. Distribuição estatística dos sinais de entrada.

O algoritmo SOFM armazena informação a respeito da *densidade de probabilidade* [47] associada ao conjunto de dados de entrada. Do ponto de vista da classificação e do reconhecimento de padrões, pode-se descrever a operação principal do SOFM como uma tarefa de *quantização vetorial* dos vetores que lhe são apresentados como sinal de entrada. A *quantização* é um processo de aproximação de sinais de amplitude contínua, a sinais de amplitude discreta [48], a diferença entre os sinais é chamada de erro de quantização. A

quantização vetorial se refere ao mapeamento de um vetor x, de N-componentes de valor real e amplitude contínua, definido por $\mathbf{x}=[x_1\ x_2\ ...\ x_N]^T$, onde $\{x_k \leq 1 \leq k_N\}$ e T indica a transposta do vetor. O vetor **x** é mapeado em outro vetor **y**, cujos componentes são valores reais, mas de amplitude discreta. Neste caso se considera que **x** é quantizado como **y** e que **y** é o valor quantizado de **x**, definido por um operador de quantização q(.), obtendo-se a expressão :

$$y = q(x) \tag{3.8}$$

A quantização vetorial se define como um processo de remoção de redundância, utilizando, entre outras, a propriedade da forma da **função de densidade de probabilidade** (*pdf*). O algoritmo SOFM opera como um quantizador vetorial da forma explicada a seguir.

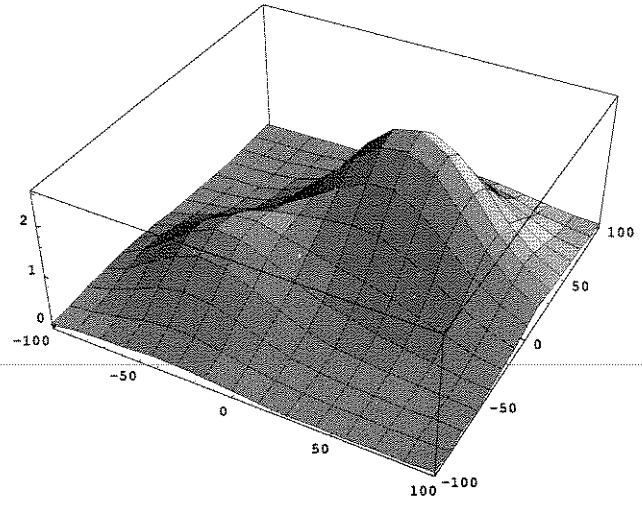


Figura 3.10. Densidade de probabilidade $P(x)$ associada a um conjunto de vetores de entrada **x**

Na figura 3.10 apresenta-se uma função de distribuição de densidade de probabilidade $P(x)$, de um vetor característico **x**. A função $P(x)$ da figura é formado por 2 funções bi-variáveis gaussianas de formas diferentes, que definem 2 regiões características associadas com 2 classes, com freqüência e variância diferentes. As classes correspondem aos picos das gaussianas localizados no plano bi-dimensional da figura e representam as regiões onde estão concentrados os vetores característicos que serão apresentados como entrada. Nas regiões com mínimo valor $P(x)$ (valores próximos a zero), haverá presença reduzida de sinais, os valores dos componentes dos vetores $\mathbf{x}=[x_1\ x_2]^T$ estão restritos ao retângulo definido pelo plano x-y.

Desta forma, a função que define a superfície da fig. 3.10, pode ser expressa por :

$$P(x_1,x_2) = k1\ \frac{1}{e^{\frac{(x1\ -\ x2)^2}{k2}}}\ \cdot\ k3\ \frac{1}{e^{\frac{(x1\ -\ x2)^2}{k4}}} \tag{3.9}$$

Podemos dizer que a função $P(x)$ é neste caso uma função de duas variáveis aleatórias x_1 e x_2, e que $P(x_1,x_2)$ define uma **pdf** não uniforme.

Na figura 3.11 ilustra-se a densidade associada à função **pdf** da fig.3.10. A função está definida num plano x-y de 200 x 200 pontos.

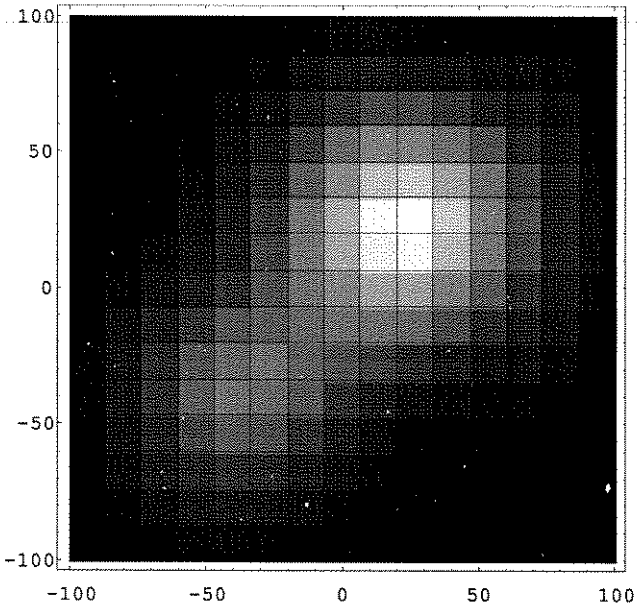


Figura 3.11. Densidade associada à função **pdf** de 3.10

Na figura 3.11., o nível de cinza é associado ao valor da função **pdf**, os valores menores são representados por níveis mais escuros, os máximos por níveis de cinza mais claros.

Os sinais de entrada não são necessariamente definidos por uma função **pdf** irregular, podendo estar distribuídos de forma mais regular no espaço de entrada. Na figura 3.12. se ilustra o exemplo de uma função de densidade de probabilidade uniforme, onde x_1 e x_2 são variáveis aleatórias independentes se a sua **pdf** é igual ao produto de suas densidades individuais (ou marginais),i.e., :

$$p(x_1,x_2) = p(x_1)\ p(x_2),\ \forall\ x_1,\ x_2\ \textit{independentes} \qquad (3.10)$$

A dependência referida nesta secção refere-se à dependência estatística, que pode ser classificada como *linear* e *não-linear*. A dependência linear é normalmente conhecida como *correlação*, duas variáveis correlatas são linearmente dependentes quando existe uma correlação entre elas. Caso as variáveis não sejam correlatas, não são mais linearmente dependentes, mas ainda podem ser estatísticamente dependentes sob um tipo chamado de dependência **não-linear**. Para duas variáveis aleatórias x_1 e x_2, elas não são correlatas se o valor esperado do seu produto é nulo, i.e. :

$$\mathcal{E}[x_1 x_2] = 0 \qquad (3.11)$$

Se x_1 e x_2 não são correlatas , mas dependentes, então a dependência é chamada de não linear.

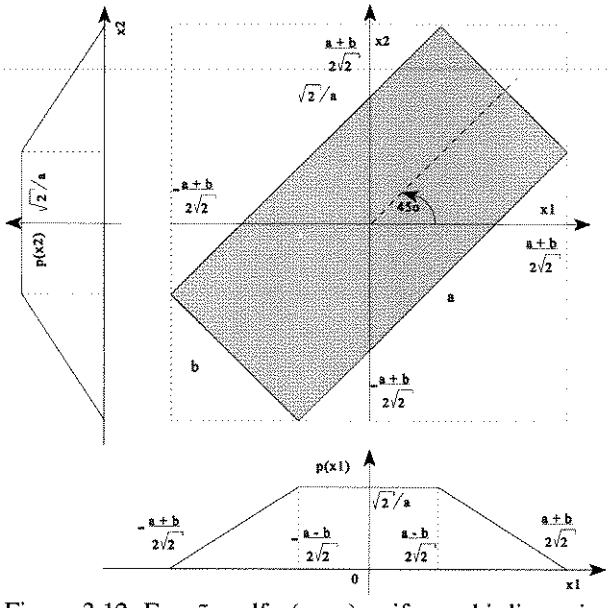


Figura 3.12. Função pdf $p(x_1,x_2)$ uniforme bi-dimensional

Na fig. 3.12, a densidade **pdf** é zero fora do retângulo de área *ab* e uniforme e distinta de zero dentro dele.As densidades $p(x_1)$ e $p(x_2)$ são mostradas na mesma figura e são conhecidas como *densidades marginais*, neste exemplo as variáveis x_1 e x_2 estão correlacionadas.

Na capítulo seguinte será ilustrado por meio de simulações como o algoritmo SOFM representa a distribuição **pdf** do conjunto de sinais de treinamento **x**(t) e como a quantização adequada de **x** é dependente da escolha dos parâmetros do algoritmo descritos em 3.3.

3.4. Complexidade do SOFM.

3.4.1. Introdução.

A complexidade algorítmica está relacionada com os aspectos quantitativos da solução de problemas computacionais e diz respeito às técnicas para otimizar o uso dos recursos de uma máquina.

Uma forma de comparar o desempenho de um algoritmo é quantificar a sua eficiência, para que esta medida seja útil isto deve ser feito independentemente da máquina. A questão da análise de um algoritmo refere-se ao processo de determinar que quão eficiente ele é, i.e. o tempo requerido para executar uma tarefa, o custo em termos de memória necessária para execução e como aproveita os recursos disponíveis do sistema.Tais medidas são normalmente expressas como uma função do tamanho do problema. Para o caso de algoritmos rodando em máquinas de arquitetura paralela, outras medidas devem ser também utilizadas, como: número de processadores utilizados, distribuição da carga de trabalho, *speed-up*, etc.

3.4.2. Uma Medida de complexidade.

Para o caso de computação em modelos de máquinas sequencias, como numa *máquina de Turing*, existe uma métrica bastante utilizada chamada de *aproximação assintótica*, que utiliza uma série de notações conhecidas como *Big Oh*, *Big Omega* e *Big Theta* [49], [50] e [51]. Nesta secção, limitar-nos-emos ao uso da *Big Oh*, ou simplesmente O(.).

A notação O(.), é utilizada para definir a função de complexidade no tempo do algoritmo. O tempo de execução é normalmente uma função do tamanho da sua entrada. O objetivo de medir a complexidade é quantificar o tempo que leva uma operação ou passo em particular e com que freqüência ocorre.

A seguir, a definição de Knuth :

Sejam duas funções f e $g : Z^+ \rightarrow R^+$. $f(n)$ é $O(g(n))$ ("f de n é O de $g(n)$"), se e somente se existe um $n_0 \in Z^+$ e um $c \in R^+$, de forma que se cumpra :

$$f(n) \leq c\, g(n), \quad \forall\, n \geq n_0 \tag{3.12}$$

Em termos gráficos $f(n)$ é $O(g(n))$ se e somente se existir um c tal que o gráfico de f não ultrapasse o de cg, após que um ponto dado, n_0, seja alcançado na abcissa. A notação O(.) ressalta formalmente duas idéias :

- A função exata, $g(.)$ não é criticamente importante, devido ao fato da função poder ser multiplicada por qualquer constante arbitrária.
- O comportamento relativo das duas funções, f e g, é comparado somente para grandes valores de n (assintoticamente) e não próximo da origem, onde os resultados podem ser desvirtuados.

3.4.3. Análise assintótica do SOFM.

Considere-se o caso de uma rede de Kohonen constituída por **n** unidades ou células neurais, onde os vetores de entrada **x** o dos pesos **m** compõem-se de **k** componentes.

A partir de (3.7), podem-se distinguir 3 fases principais na operação :

- *Cálculo da Distância* : Todas as células realizam esta operação de forma concorrente. A distância entre o vetor de entrada **x** é o vetor de pesos interno **m** deve ser avaliada, de acordo com alguma métrica adequada (distância Euclideana, Manhattan, etc.).

- *Determinação da célula ganhadora*: A célula cuja diferença |**x**-**m**| seja mínima deve ser determinada, esta é a ganhadora do processo competitivo.

- *Atualização dos pesos* : Após ter definido o tamanho da vizinhança $N_c(t)$ para o passo corrente do processo de aprendizado, a célula ganhadora e todas as pertencentes à vizinhança $N_c(t)$ modificam os seus pesos, de acordo com (3.7).

Para avaliar o custo do algoritmo, deve-se distingüir o modelo de máquina onde ele será implementado. Utilizar-se-á um modelo seqüencial e outro paralelo, cuja arquitetura será descrita na próxima secção. Para o cálculo da distância, considere-se a *Distância Euclidiana* e a *Distância Manhattan* [19], definidas a seguir :

Distância Euclidiana : $$d^2(\boldsymbol{m}_i,\boldsymbol{x}) = \sum_{k=1}^{n} (\boldsymbol{m}_i - \boldsymbol{x}_k)^2 \qquad (3.13)$$

Distância Manhattan : $$d(\boldsymbol{m}_i,\boldsymbol{x}) = \sum_{k=1}^{n} \|\boldsymbol{m}_i - \boldsymbol{x}_k\| \qquad (3.14)$$

3.4.3.1. Estudos de casos para processamento seqüencial.

Refere-se ao modelo seqüencial definido por uma máquina de Turing ou pela arquitetura de Von-Neumann, sendo o modelo onde serão realizadas as primeiras simulações.

Algoritmo 3.1.-Cálculo da Distância :

```
Passo 1.[Loop externo]      For i := 1 To n do
Passo 2.[Loop interno]        For j:= 1 To k do
Passo 3.[D. parcial+Atr]        D_ij ← (m_ij - x_ij)
Passo 4                       End j
Passo 5                     End i
```

Verifica-se que, sem considerar a métrica da distância, o tempo de computação é pelo menos da ordem O(*nk*), a escolha da métrica do cálculo da distância é então relevante no desempenho. O índice *nk* indica a forma como o tempo de computação, quantificada por exemplo pelo número de vezes que uma instrução é executada, cresce em função da entrada.

No algoritmo 3.1., o passo 1 é executado n+1 vezes; o passo 5, n vezes; o passo 2 é executado (k+1) vezes, cada vez que o *loop* externo é executado, dando um total de n(k+1) vezes; os passos 3 e 4 são executados nk vezes.

Pode-se deduzir assim que o tempo requerido é de t1(n+1) + t2n(k+1) + t3nk + t4nk + t5n, onde t1, ...t5 representam os tempos de cada passo do algoritmo. A expressão anterior assuma uma forma *nc1* + *nkc2*, onde c1 e c2 são constantes e *n* e *k* são os parâmetros associados com o tamanho do problema, prevalece então a ordem superior *nk*, indicada por O(nk).

Considere-se a seguir um arranjo DIST de *n* números (as *n* distâncias calculadas anteriormente). Na segunda fase, deve-se determinar a célula ganhadora, i.e., a célula cuja diferença (**m-x**) é mínima. O método mais simples é varrer o arranjo seqüencialmente :

Algoritmo 3.2.-Determinação do Mínimo :

```
Passo 1.[Atribuição]        i ← 1
Passo 2.[Atribuição]        min ← DIST[i]
Passo 3.[Loop]              While i<n do
Passo 4.[Incremento de i]     i ← i +1
Passo 5.[Teste]               If min > DIST[i] Then
Passo 6.[Atribuição]                min ← DIST[i]
Passo 7                       End If
Passo 8                     End Do
```

Duas operações essenciais destacam-se neste algoritmo. A comparação do valor mínimo com o valor do elemento corrente do arranjo DIST e a atribuição de um valor novo a min. O *loop* deve ser executado (n-1) vezes, definindo o limite inferior do tempo de execução. A atribuição é executada uma vez no início e não haveria mais atribuições se o primeiro elemento do arranjo for o mínimo. O outro caso extremo é descobrir um mínimo a cada iteração do *loop*, gerando-se *n* atribuições. Em média, haverá *n*! possibilidades de arranjos de números. Levando em conta que que a primeira atribuição é executada uma vez e assumindo que todas as permutações são igualmente prováveis, a probabilidade que o segundo elemento seja menor que o primeiro é 1/2. A probabilidade de que o terceiro seja ainda menor que os anteriores é 1/3 e assim sucessivamente, se obtem que a média de atribuções é :

$$1 + 1/2 + 1/3 + ... + 1/n ,$$

onde para grandes valores de *n* pode-se ter :

$$\mathrm{Log_e}(n) + 0.577.$$

Devido a que Log(n) cresce mais lentamente que o próprio *n*, a complexidade do algoritmo 3.2 é dominada pelo tempo de execução do *loop*, i.e., é proporcional a *n*, portanto, a ordem do algoritmo é O(n).

algoritmo 3.3.- Atualização dos pesos.

Para efeitos do estudo, considere-se que a função $\alpha(t_k)$ e a função $\Lambda_c(i,t_k)$ que representa a interação lateral entre a célula ganhadora *c* e a i-ésima célula dentro de $N_c(t_k)$, ambas definidas em (3.7), são substituídas por uma função *kernel* $h_{ci}(t_k)$ de forma que :

$h_{ci}(t_k) = \alpha(t_k)\Lambda_i(i,t_k)$ dentro da vizinhança $N_c(t_k)$
$h_{ci}(t_k) = 0$ fora da vizinhança $N_c(t_k)$,

onde *i* é a i-esima célula da rede de *n* neurônios e t_k representa o tempo discreto.

Passo 1.[Loop externo] **For** i := 1 **To** n **do**
Passo 2.[Loop interno] **For** j:= 1 **To** k **do**
Passo 3.[Atualização] $m_{ij}(t_k+1) \leftarrow m_{ij}(t_k) + h_{ci}(t_k)[x_j(t_k) - m_{ij}(t_k)]$
Passo 4 **End** j
Passo 5 **End** i

Análise :

O passo 1 é executado n+1 vezes; o passo 5, n vezes; o passo 2, n(k+1) vezes; os passos 3 e 4, nk vezes. Numa primeira aproximação, o algoritmo é semelhante ao descrito em 3.1.

Ao detalhar o passo 3, vemos que há 3 operações matemáticas : uma subtração, uma soma e uma multiplicação. Cada uma destas operações mais a atribuição (atualização do peso) é realizada nk vezes, a avaliação da função kernel $h_{ci}(t_k)$, será descrita a seguir.

A complexidade do algoritmo é dominada fundamentalmente pelo produto nk , portanto, é possível verificar que a sua ordem é O(nk).

No entanto, deve-se considerar o tempo que levam as atribuições e as operações matemáticas, a simples leitura da notação O(.), não permite visualizar a sua influência. Dependendo dos tipos de dados utilizados (ponto fixo, flutuante, *double precission*, etc.), o peso principalmente da operação de multiplicação será muito influente, demandando mais tempo de computação que o resto das operações. É impossível, neste momento, fazer uma estimativa quantitativa em tempo, já que é dependente da máquina onde será executado o algoritmo e do compilador determinado.

Um outro aspecto que a notação O não considera se refere ao uso adequado dos recursos do sistema utilizado, para termos uma idéia, a execução completa do algoritmo numa máquina seqüencial será a soma das 3 fases principais mencionadas no início desta secção.

3.4.3.2. A influência do *kernel.*

A função *kernel* $h_{ci}(t_k)$, permite modelar a interação lateral mencionada na secção 3.3.2. Normalmente esta é uma função do tipo *gaussiana*, como por exemplo do tipo [16] :

$$h_{ci} = h_0 \exp\left(-\frac{\|r_i - r_c\|^2}{\sigma^2}\right) \qquad (3.15)$$

onde $\mathbf{r}_i$ e $\mathbf{r}_c$ representam os vetores das coordenadas das células *i* e *c* respetivamente,
$h_0 = h_0(t)$ e $\sigma=\sigma(t)$ são adequadas funções decrescentes.

A implementação deste tipo de Kernel varia entre uma versão e outra de máquina e compilador. Mas, de qualquer forma, acrecescenta mais um nível de complexidade ao algoritmo, contribuindo a aumentar o tempo de execução total. Esta função não-linear será linearizada na implementação em *hardware* digital, por causa do seu alto custo de implementação em termos de área de silício.

3.4.3.3. Análise e conclusão.

A partir dos itens anteriores, e considerando-se os níveis de complexidade das 3 fases descritas no início da secção 3.4.3, pode-se deduzir que o algoritmo cresce a uma taxa O(nk), i.e., a sua complexidade no tempo é função do tamanho da rede e da dimensão dos vetores de treinamento e pesos. A influência da métrica do cálculo da distância, das funções de modelamento da interação lateral e da função de ganho são importantes no desempenho, mas não contribuem a aumentar a sua função de **complexidade**. A notação O(.) é mais que nada utilizada para avaliar a complexidade no tempo de execução e não diz nada a respeito do aproveitamento dos recursos da máquina, como por exemplo, armazenamento de vetores, matrizes, dados intermediários, etc .

3.4.4. Paralelização do SOFM.

Nesta secção não será feita uma análise rigorosa da implementação em paralelo do algoritmo para um caso geral. Uma análise completa implica necessariamente descrever os detalhes de implementação da uma máquina paralela em particular. No entanto, descrever-se-ão os aspectos gerais mais relevantes relativos à arquitetura paralela onde será implementado o SOFM.

A paralelização do algoritmo é feita naturalmente devido à natureza intrinsecamente paralela dos modelos de processamento neural. A máquina aqui proposta implementa o SOFM, reduzindo a sua complexidade à ordem O(k), a arquitetura pode ser classificada como do tipo *array processor*, um elemento é ilustrado na figura 3.13. Nela, podem se distingüir 4 blocos principais:

- *UA* : Unidade Aritmética, encarrega do cálculo das operações +/- e multiplicação. A operação e controlada pela unidade UC.
- *UC* : Unidade de controle de todas a operações.
- *α(t)* : unidade dedicada a geração da função $\alpha(t)$.
- **x**, **m**, ..., registradores dedicados ao armazenamento de vetores e variáveis.

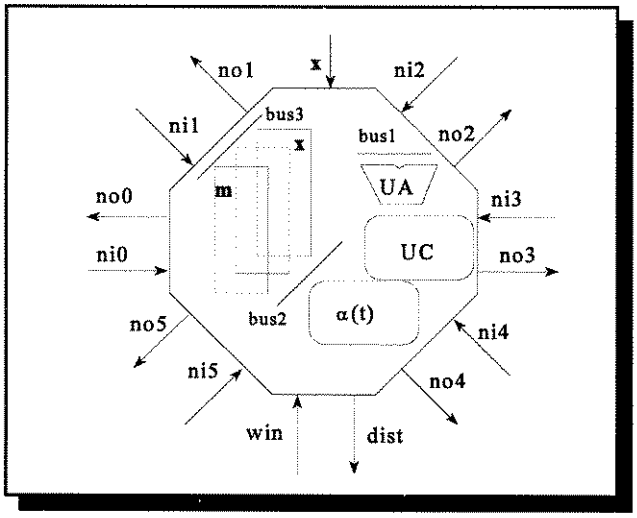


Figura 3.13.Proposta inicial para um elemento de um *Array Processor* dedicado à implementação paralela do algoritmo SOFM.

A figura 3.13 ilustra uma proposta inicial para um *Neuro-processador* dedicado à implementação paralela do SOFM, a figura representa **uma** célula da rede neural de Kohonen. A célula se comunica com as vizinhas pertencentes a $N_c(t_k)$, através das linhas no0,..., no5 e ni0,...ni5. O cálculo da distância é implementado concorrentemente por cada célula, que envia o valor *dist* a outra unidade encarregada de determinar a ganhadora. Caso a célula seja a ganhadora do processo competitivo, ela recebe um sinal externo para indicar este estado. O vetor **x**, é transmitido concorrentemente a todas as células, tal como no algoritmo original, o vetor **w** é armazenado antes do início da fase de aprendizado e o critério de parada é determinado internamente, assim como a função de interação lateral. Os módulos internos se comunicam através de barramentos dedicados.

Análise.

O algoritmo 3.1.é implementado individualmente por cada neuro-processador, eliminando os passos 1 e 5 (o *loop* externo), a ordem é reduzida para O(k).

O algoritmo 3.2 é implementada em *hardware* por uma unidade externa chamada de WTA (*Winner-Takes-All*) por meio de uma árvore de busca binária. Uma árvore binária completa de altura *h*, possui 2^{h+1} nós e 2^n folhas. O tempo requerido pelo algoritmo de busca binária é $K \cdot \log_2 n$, onde *n* é o tamanho da entrada e K é uma constante de proporcionalidade [52].

O caso do algoritmo 3.3. é semelhante ao 3.1, já que cada atualização é feita individualmente por cada célula, portanto, são eliminados os passos 1 e 5 correspondentes ao *loop* externo, a complexidade diminui para O(k).

Como já foi mencionado, todos os aspectos relativos à operação detalhada da arquitetura aqui apresentada, serão tratados no capítulo relacionado com o protótipo, descrito e modelado em nível RT (*Register Transfer*) na linguagem VHDL.

Capítulo 4

Primeira fase experimental : Implementação em *software* do SOFM

Escopo

Neste capítulo é descrita a implementação em *software* do algoritmo básico discutido nos capítulos 2 e 3. O estudo visa descobrir a influência dos parâmetros descritos no capítulo 3.

Nesta primeira fase experimental, todas as simulações foram implementadas na linguagem ANSI-C. Foi desenvolvido um simulador denominado KNN (*Kohonen Neural Network*), incluindo uma interface gráfica que permitisse a visualização dos mapas gerados pelo algoritmo. O objetivo desta fase é identificar os problemas que surgem ao realizar simulações numa máquina seqüencial e deduzir certas características de convergência do modelo. Esta implementação visa também descobrir possíveis aplicações em tempo real, além de definir as condições sob as quais o algoritmo poderia ser modificado para realizar a sua implementação em *hardware*, identificar as vantagens e detectar pontos críticos.

Ao final deste capítulo, será apresentada uma aplicação do modelo de Kohonen dedicada à classificação e posterior reconhecimento de um conjunto de padrões que representam o timbre acústico de diferentes instrumentos. Será comprovado experimentalmente que o algoritmo realmente consegue realizar esta tarefa com uma taxa de erro razoavelmente baixa.

4.1. Introdução

De forma geral, os algoritmos de redes neurais não são desenvolvidos visando a sua implementação em algum tipo específico de *hardware*, seja óptico ou eletrônico, menos ainda numa determinada tecnologia. É necessário modificá-los e adequá-los às necessidades do *hardware*, de forma que os circuitos os implementem em condições de uso real, otimizando seu desempenho e/ou área de silício. A maioria das vezes, o estudo de modelos de redes neurais depende fundamentalmente de simulações.

Neste capítulo, o algoritmo SOFM será estudado por meio de simulações exaustivas. O objetivo é verificar experimentalmente os aspectos mais relevantes relacionados com o seu desempenho, em termos da sua estabilidade e da taxa de convergência.

Na seqüência, será apresentada a estrutura básica do simulador KNN (*Kohonen Neural Network*) e dos componentes da sua interface gráfica. Nas secções seguintes, será apresentada uma série de resultados experimentais, produto da variação dos parâmetros do algoritmo e finalmente descrever-se-ão as conclusões deste estudo inicial.

4.1.1. Descrição geral do simulador KNN.

O simulador construído opera na base de um *micro-kernel* , que controla a execução de uma série de módulos, operando na base de um modelo *operador-operandos* (vide figura 4.1).

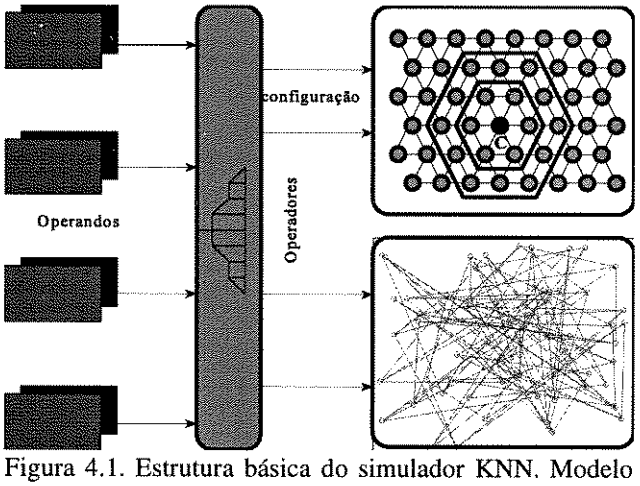


Figura 4.1. Estrutura básica do simulador KNN. Modelo operador-operandos proposto por Cheneval.

Os *operandos* acionam módulos que implementam uma série de funções (operadores), como geração de $\alpha(t_k)$, $N_c(t_k)$, distintas distribuições de probabilidade $P_i(\mathbf{x})$, etc. Os *operadores* exercem funções semelhantes às de um filtro: recebem um sinal de entrada, que é processado e posteriormente enviado para um outro módulo, que pode ser outro operando.Na fig. 4.1. ilustra-se a estrutura elementar do simulador composto por 3 blocos principais : operandos, operadores e interface gráfica. O modelo de simulador utilizado é semelhante ao proposto por Cheneval [53], onde a saída produzida pela ação dos *operadores* é classificada como outro conjunto de operandos, tal como o efeito de processamento de um filtro.

4.1.2. Interface Gráfica..

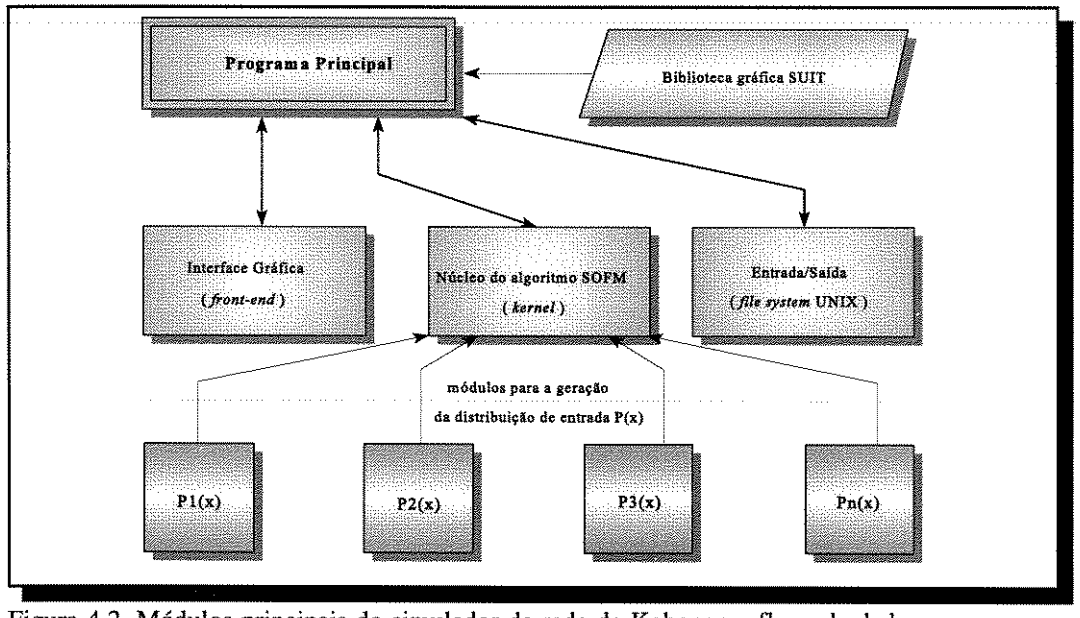


Figura 4.2. Módulos principais do simulador da rede de Kohonen e fluxo de dados.
Pi(x): Módulos para geração da Distribuição de Probabilidade da entrada **x**.

A implementação da interface gráfica foi baseado no uso da linguagem ANSI-C e de uma ferramenta de bibliotecas gráficas orientada a objetos (SUIT) [54]. As bibliotecas SUIT permitem basicamente a criação de uma série de objetos gráficos (*widgets*) que permitem interagir com o núcleo do programa de aplicação. O diagrama de blocos e o seu fluxo de dados é descrito na Figura 4.2 e compõe a sua estrutura básica. O algoritmo básico SOFM é implementado por um núcleo que se comunica continuamente com as rotinas da interface, cujos parâmetros mais influentes no desempenho foram descritos nos capítulos anteriores. Foram realizadas experiências com diversas funções de probabilidade , implementadas através de uma série de módulos (operadores).

Os estudos requerendo o uso de padrões de treinamento de dimensão maior que 2, foram realizados sem a interface gráfica, tal como no reconhecimento de padrões acústicos descrito na secção 4.6 deste capítulo.

4.2. Variação de Parâmetros.

As funções que influenciam a evolução do algoritmo são principalmente a função de ganho $\alpha(t_k)$ e a função de interação lateral (3.5), que define a vizinhança $N_c(t_k)$.

4.2.1. Função de ganho $\alpha(t_k)$.

Nesta primeira fase, utilizaram-se praticamente 2 funções de ganho, uma linear e outra não linear:

(i) *Linear* :

$$\alpha(t_k) = k1\ [1 - \frac{t_k}{1000}], \quad se\ t_k \leq 1000$$
$$k2\ [1 - \frac{t_k}{S_{max}}], \quad se\ t_k > 1000 \qquad (4.1)$$

(ii) *Não-Linear* :

$$\alpha(t_k) = k1\ \exp(-\ k3\, t_k) \qquad (4.2)$$

onde $S_{max,}$ representa o número máximos de passos de treinamento, t_k é o tempo discreto e k1, k2 são parâmetros constantes variando entre 0.1, ..., 1.0 e 0.005,.., 0.008 respectivamente.

4.2.2. Vizinhança $N_c(t_k)$.

A vizinhança foi variada linearmente, de acordo com a expressão seguinte :

$$N_c(t_k) = k3\, w\ [1 - \frac{t_k}{S_{max}}] \qquad (4.3)$$

onde, *w* representa o diâmetro da rede de *n* neurônios, cujo valor inicial $N_c(0)$ foi variado entre 100% e 50% do tamanho máximo da rede, sendo controlado por k3.

Nesta primeira fase, a função de interação lateral foi igualada a 1, já que inicialmente desejava-se estudar somente o efeito do ganho $\alpha(t_k)$ e de função $N_c(t_k)$, a fim de decidir a sua forma de implementação em *hardware*. A implementação de (3.5) será tratada no capítulo 5, relativa ao protótipo. A função (3.5), portanto, assume a forma seguinte :

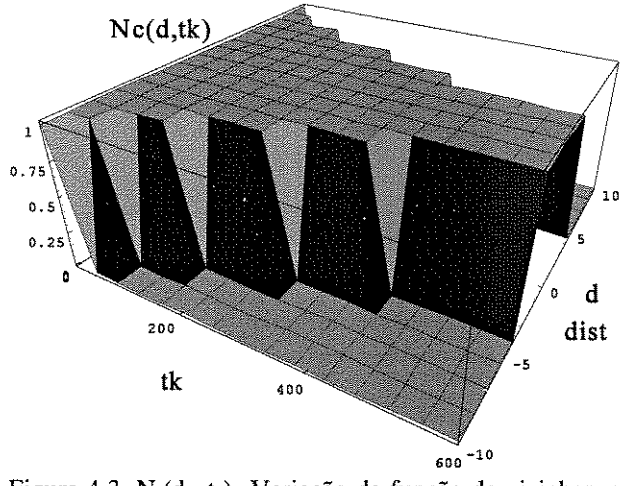


Figura 4.3. $N_c(d_c, t_k)$- Variação da função de vizinhança

Na figura 4.3, se ilustra a variação da vizinhança, em função do tempo. O raio de $N_c(t_k)$, decresce monotonicamente até alcançar um valor estável, normalmente 1, até o valor final de t_k.

4.3. Dados de Treinamento.

Utilizaram-se até 7 funções de distribuição de probabilidade uniforme (pdf) para o treinamento do SOFM (vide fig. 4.4) no simulador KNN. O número de vetores de treinamento foi variado entre 1.000 e 100.000, para verificar as diferenças nos mapas resultantes, sendo que a ordem de apresentação foi variada de forma aleatória.

Alguns exemplos de **pdf** utilizadas são :

- *Uniforme* : todos os dados de entrada estão distribuídos com a mesma **pdf** em todo o espaço, dentro de um intervalo [a,b]. A probabilidade da entrada não pertencer ao intervalo [a,b] é zero.

- *Circular* : vetores de entrada agrupados em um *cluster* de forma geométrica circular, probabilidade do vetor estar fora de círculo: 0, dentro do círculo : uniforme.

- *Cruz* : Os dados de entrada estão agrupados num *cluster* definido por um área de forma geométrica em cruz.. A probabilidade dos componentes do vetor estar dentro da área é uniforme, fora da área definida : 0.

- *Triangular* : as mesmas características do conjunto de sinais de entrada anterior. A forma geométrica definida pela **pdf** é um triângulo.

O conjunto de sinais de entrada é escolhido interativamente usando o simulador, alguns exemplos são ilustrados na figura 4.4.

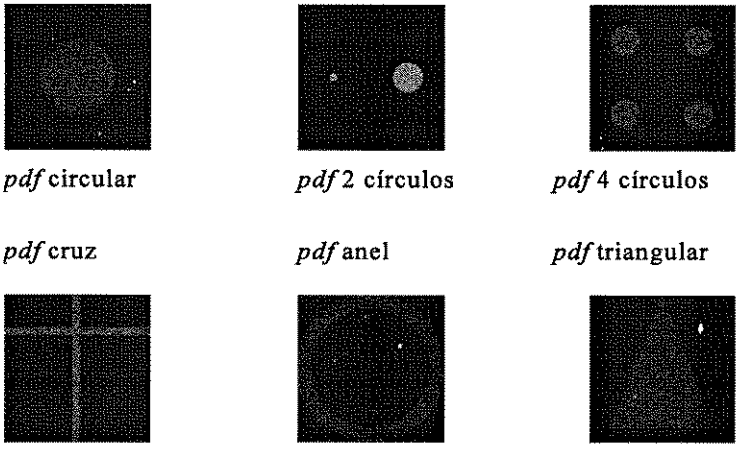


Figura 4.4.- Exemplos de distintas funções de distribuição **pdf** dos vetores de entrada $\mathbf{x}(t_k)$, usados para treinamento do SOFM

Na figura anterior todos os sinais de entrada são vetores bi-dimensionais, cujos componentes são valores reais pertencentes ao intervalo [0,1].

4.4. Experiências.

Simulação 4.1. Simulação com distribuição de vetores de entrada uniforme em todo o intervalo $[0,1] \in \mathbb{R}$. Foram utilizados 100.000 vetores de treinamento apresentados à rede aleatoriamente (vide fig.4.5.a). Os pesos foram inicializados em valores aleatórios (fig.4.5.b). O mapa resultante é ilustrado na figura 4.5.c.

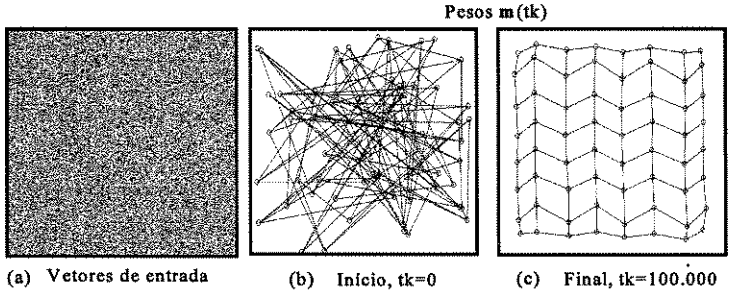


Figura 4.5. Simulação para 100.000 dados de treinamento

A figura 4.5 ilustra a forma convencional de representação de mapas adaptativos auto-organizados. Os pesos $\mathbf{m}_i(t_k)$ de uma rede de 8 x 8 células são graficados num espaço bi-dimensional, ligando os nós das células *i* e *j*,, representadas pelos pesos $\mathbf{m}_i$ e $\mathbf{m}_j$, quando as células *i* e *j* são vizinhas imediatas, i.e., quando separadas a uma distância=1. A vizinhança usada é do tipo *hexagonal*. As funções $\alpha(t_k)$ e $N_c(t_k)$ foram definidas pelas expressões (4.1) e (4.3).

É possível conferir no mapa final (fig.4.5.c) que os vetores de pesos espalham-se pelo espaço, de forma de cobrir o máximo possível a superfície definida pelos vetores de entrada. A evolução no tempo dos mapas será apresentada nas seguintes simulações.

Simulação 4.2. Na simulação ilustrada na figura 4.6, uma rede composta por 64 células

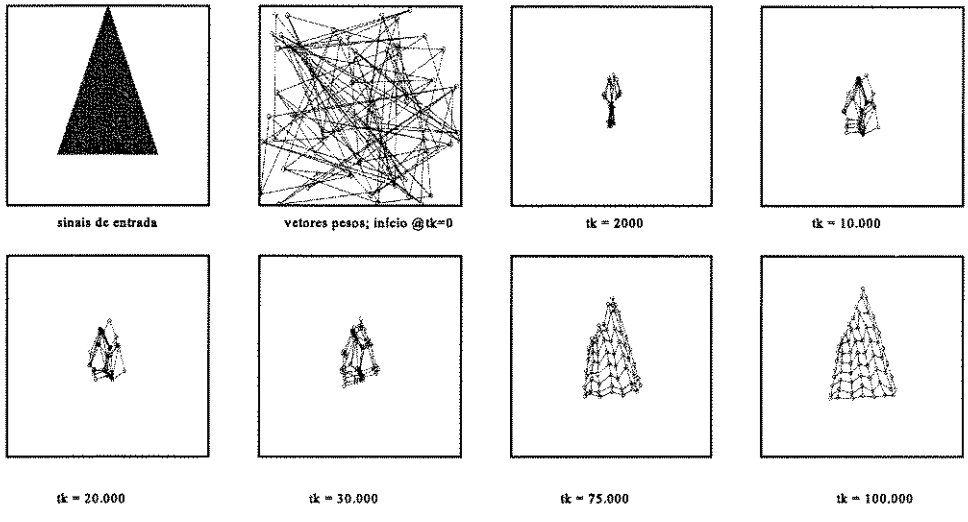


Figura 4.6. Simulação de uma rede de 64 células, treinada por um conjunto de vetores agrupados em um *cluster* de forma triangular com **pdf** uniforme

conectadas numa vizinhança de topologia hexagonal, foi treinada com 100.000 vetores de entrada cuja função **pdf** era uniforme, dentro de um *cluster* de forma geométrica triangular, os pesos foram inicializados com valores aleatórios. A probabilidade do vetor de entrada de não pertencer ao triângulo é 0.

As duas experiências descritas, simulações **4.1** e **4.2**, foram repetidas uma série de vezes modificando a sequência de apresentação dos vetores de entrada e modificando o próprio conjunto de vetores, mantendo sim o seu número e o tipo de função **pdf**. Não foi observada mudança nos mapas resultantes, comprovando que o algoritmo é bem comportado para conjuntos com **pdf** similares.

As experiências foram repetidas várias vezes utilizando funções **pdf** tal como as ilustradas na fig. 4.4. Foi observado o mesmo fenômeno anterior, i.e., os mapas resultantes convergem à mesma configuração, mesmo com sequências de entradas **x** distintas, quando é mantida a densidade de probabilidade **pdf**.

Tempo de simulação médio para 100.000 dados de entrada e n=64: entre 5-6 hrs. sparc2.

Influência da função de ganho $\alpha(t_k)$.

Na seqüência, apresenta-se uma série de simulações para verificar a influência da função definida por (4.1.) e (4.2), funções linear e não linear respectivamente.

Simulação 4.3. Considere-se o conjunto de 9000 vetores de entrada $\mathbf{x}(t_k) \in \mathbb{R}^2$, com uma distribuição **pdf** tal como a ilustrada na figura 4.7.a, e uma rede neural composta por 100 células, definida num arranjo com vizinhança hexagonal, cujos vetores pesos $\mathbf{m}(t_k) \in \mathbb{R}^2$ são inicializados (t_k=0) com valores aleatórios, mas restringidos a um intervalo [a,b], da forma como é ilustrado na figura 4.7.b.

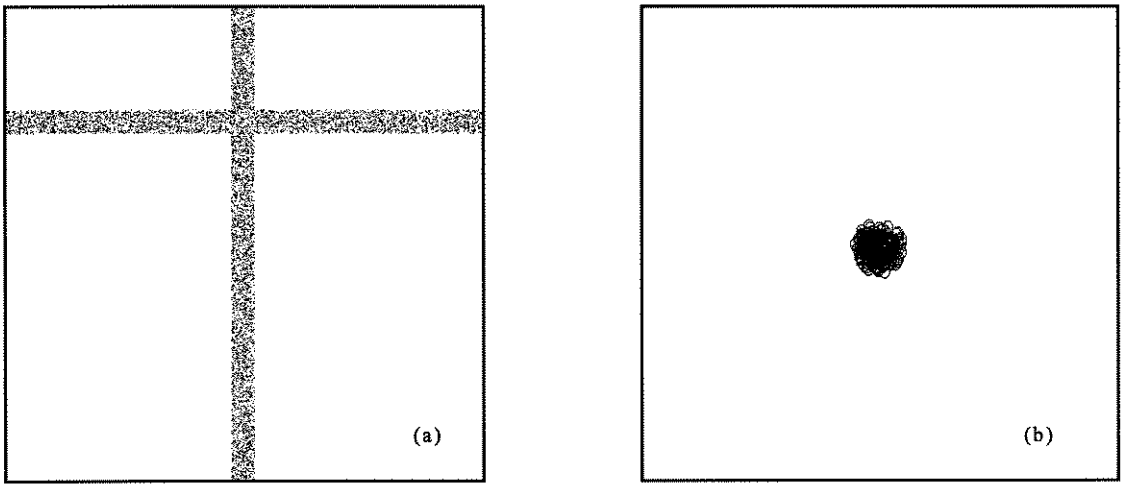


Figura 4.7. (a) Distribuição dos vetores de entrada **x**. (b) Inicialização vetores pesos **m**, t_k=0.

A rede foi treinada com o conjunto de vetores apresentados em 4.7 (a), o estado dos vetores pesos no instante t_k=0 é apresentado em 4.7(b). Os resultados são apresentados na fig.4.8.

Figura 4.8.a.

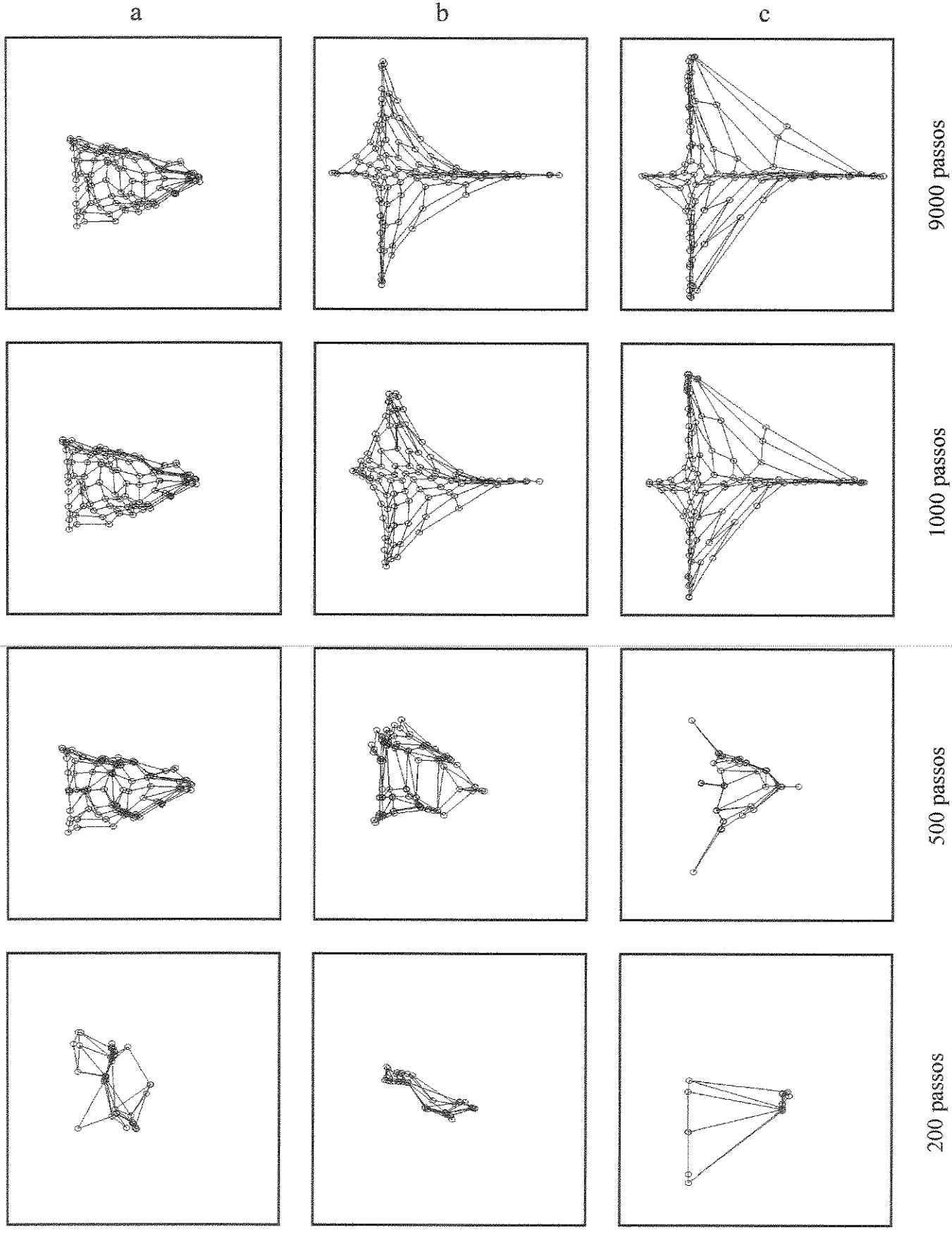
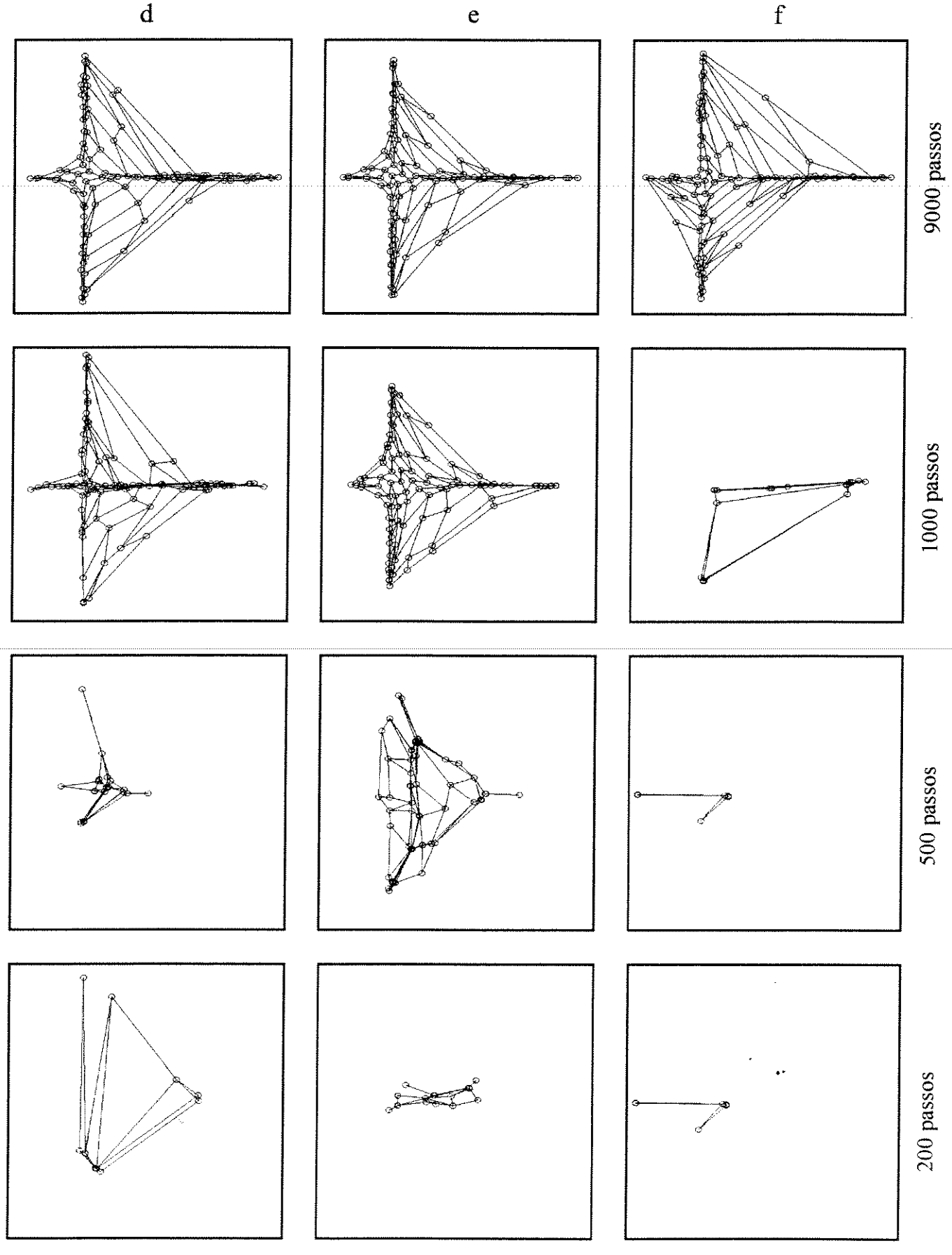
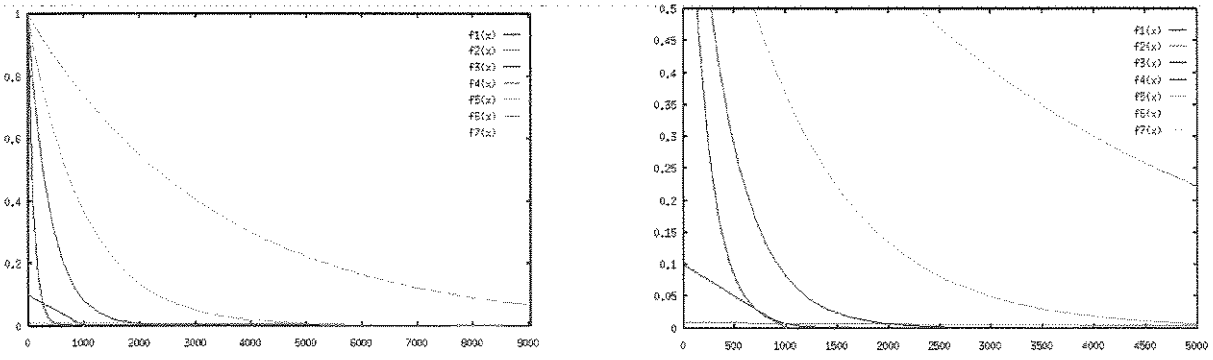


Figura 4.8.b



Nas figuras 4.8.a e 4.8.b se apresentam os resultados das simulações realizadas com o conjunto de dados descritos na fig. 4.7.a e 4.7.b. As curvas $\alpha(t_k)$ utilizadas na atualização dos pesos em cada instante são apresentadas na fig. 4.9(a)-(b).



(a) (b)

Figura 4.9.Curvas de ganho associadas as simulações das Fig.4.8.a e Fig.4.8.b. (b) *Zoom* de (a)

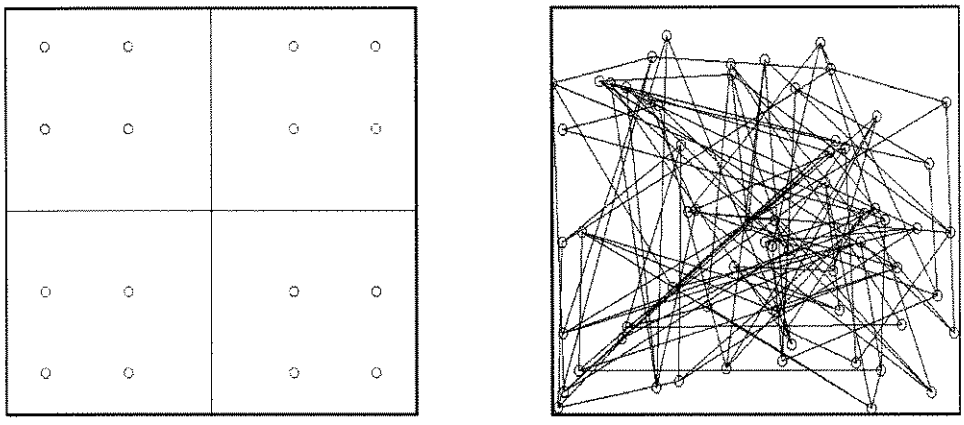
Os parâmetros de $\alpha(t_k)$ foram variados para verificar a influência nos mapas resultantes e na velocidade de convergência da rede. Foi comprovado que os mapas são altamente sensíveis à forma da curva de ganho, como se pode observar nos resultados gráficos. Ao utilizar uma função de ganho decrescendo muito rápido, a rede não consegue converger de forma de representar adequadamente a distribuição da entrada (vide a fig. 4.8.a e a sua curva de ganho relativa na fig.4.9). No outro extremo, uma curva decrescendo lentamente formou um bom mapa final, mas a sua convergência demonstra ser mais lenta, quando comparados os 500 primeiros passos de simulações, como se pode ver na fig. 4.8.d. Na evolução das simulações descritas nas fig.4.8.b e 4.8.c não foi possível verificar grandes diferenças: em 4.8b, se utilizou uma curva de ganho linear e em 4.8.c uma curva não linear, cujas formas aparecem em 4.9.a. Há uma pequena vantagem no mapa final de 4.8.c, devido ao fato de conseguir agrupar um pouco mais de pontos dentro da região coberta pelas entradas, prodizindo uma taxa de erro menor.
Tempo de simulação médio para 9.000 dados de entrada e n=100: aprox. 30 min (*Sparc2*).

Influência da ordem de apresentação de $\mathbf{x}(t_k)$.

Na seqüência, apresentam-se os resultados de uma série de simulações para verificar a influência da ordem de apresentação das entradas.

Simulação 4.4. Considere-se uma rede neural de 64 células distribuídas num arranjo de 8 x 8, com uma topologia de vizinhança hexagonal e um conjunto de 16 vetores de entrada $\mathbf{x}(t_k) \in \mathbb{R}^2$, com uma distribuição **pdf** regular, tal como a ilustrada na figura 4.10.a. A rede é treinada repetindo as 16 entradas 200 vezes, produzindo 3200 passos de treinamento. Os vetores pesos $\mathbf{m}(t_k) \in \mathbb{R}^2$,são inicializados ($t_k$=0) da forma ilustrada na figura 4.10.b.

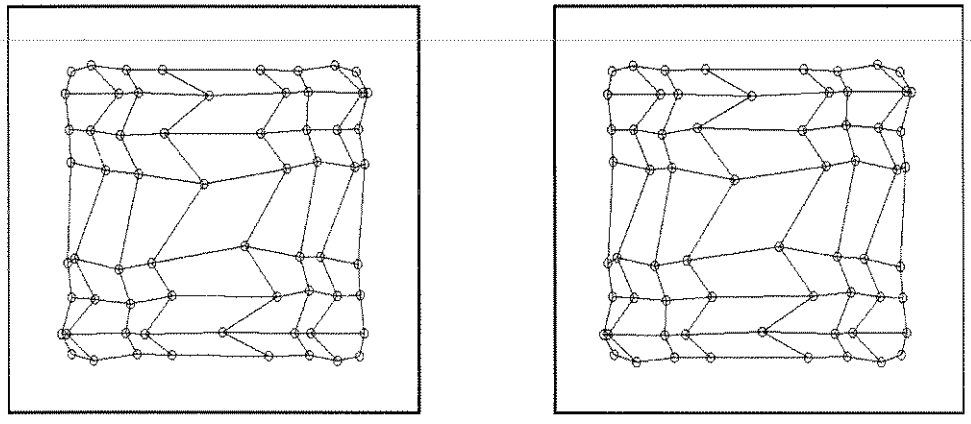
A ordem de apresentação dos vetores de entrada foi variada aleatoriamente, a fim de verificar experimentalmente a influência nos mapas resultantes. Os parâmetros das expressões (4.1) e (4.2), k1, e k3, foram modificados da mesma forma, os resultados para 3 casos de estudo são apresentados nas figuras 4.11 - 4.13.



(a) (b)

Figura 4.10.(a) Conjunto de entradas $\mathbf{x}(t_k)$ fixadas. (b) Pesos $\mathbf{m}(t_k)$, para t_k=0

Nas figuras 4.11(a)- 4.13(a), o conjunto de 16 vetores é apresentado seqüencialmente, sem variar a ordem de apresentação. Em (b) inclue-se a variação aleatória de apresentação de $\mathbf{x}(t_k)$. Três exemplos para distintos parâmetros de k1 e k3, são ilustrados.



(a) (b)

Figura 4.11. Mapas resultantes das simulações para k1 = 0.1 e k3 =0.002.

Pode-se verificar que o mapa resultante é praticamente idêntico em ambos os casos, o resultado também é similar quando os vetores pesos são inicializados com valores diferentes, inclusive quando todos os componentes de **m** são nulos em t_k=0. Os valores correspondentes aos parâmetros k1 e k3 das expressões (4.1) e (4.2) utilizados nestas simulações são :

k1	k3	Fig.
0.1	0.002	4.11.a, 4.11.b
0.9	0.002	4.12.a, 4.12.b
0.9	0.0005	4.12.a, 4.12.b

Em todos os casos se utilizaram as mesmas entradas, sendo repetidas até completar 3200 passos.

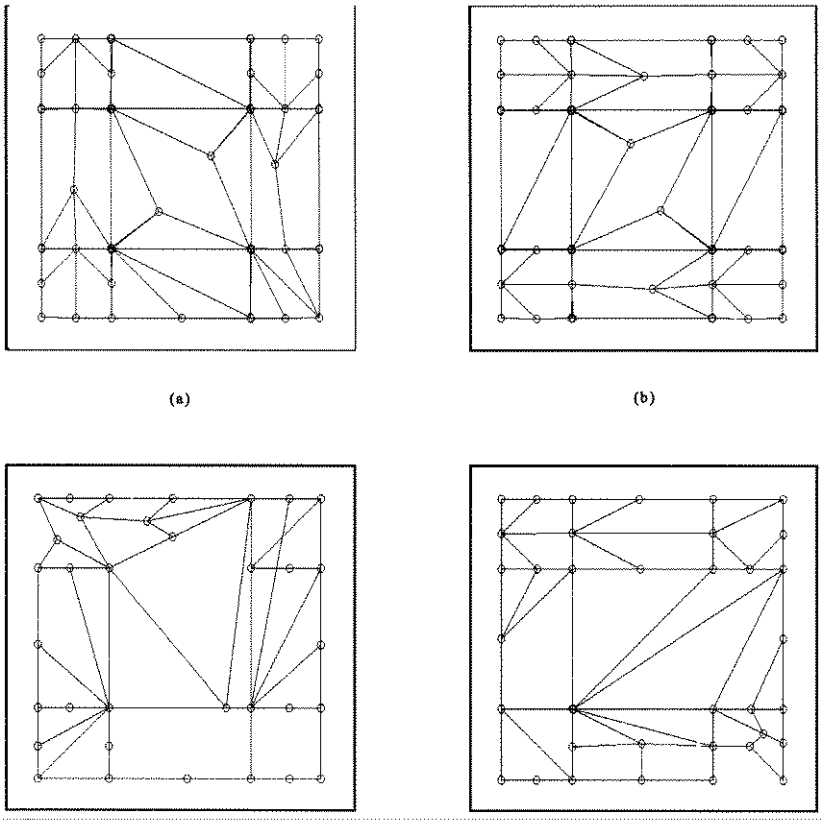


Figura 4.12. Mapas resultantes das simulações com ganho $\alpha(t_k)$ não linear.

Nas figuras 4.12 observam-se resultados semelhantes aos obtidos nas simulações ilustradas em 4.11. No entanto, no caso ilustrado nas figuras 4.12.c e 4.12.d, observam-se algumas pequenas diferenças quando compararmos os mapas resultantes, devidas provavelmente à curva de ganho $\alpha(t_k)$ não adequada. Os valores escolhidos para os parâmetros k1 e k3 influem negativamente no aprendizado da rede, a convergência é muito lenta e os mapas resultantes não são considerados de boa qualidade.

4.5. Análise conclusiva.

Todas as experiências anteriormente descritas foram repetidas exaustivamente, a fim de verificar as propriedades e o comportamento do algoritmo em diferentes condições de operação. As simulações descritas nos itens **sim4.1 - sim 4.3** foram repetidas variando as funções **pdf**, o número de vetores de entrada, a inicialização dos vetores pesos, etc. Em todos os casos as conclusões são semelhantes, no sentido da convergência e a estabilidade do algoritmo SOFM serem extremamente dependentes da forma da função de vizinhança e da velocidade com que elas decrescem monotonicamente, especialmente crítica é a escolha de parâmetros adequados.

As experiências realizadas com o algoritmo demonstraram a necessidade de contar com alta flexibilidade na futura implementação em *hardware* destas funções, de forma de poder controlar externamente a sua evolução e convergência.

4.6. Aplicações para uso em tempo real.

4.6.1. Processamento de Sinais.

4.6.1.1. Descrição.

Uma aplicação para uso em tempo real que se revela interessante é na área de comunicações. Diversos estudos tem sido realizados sobre o uso do SOFM para assistir sistemas de comunicações empregando modulação QAM (*Quadrature Amplitude Modulation*) [55], [56]. Os estudos de Raivio et al. tem visado o uso do SOFM como uma estrutura de auxílio a equalizadores do tipo DFE (*Decision-Feedback-Equaliser*). Neste caso, uma rede de 16 células representa uma constelação definida pelos valores (I,Q) de uma modulação 16-QAM. O conjunto de treinamento para esta aplicação assume uma forma tal como a ilustrada na figura 14.13a.

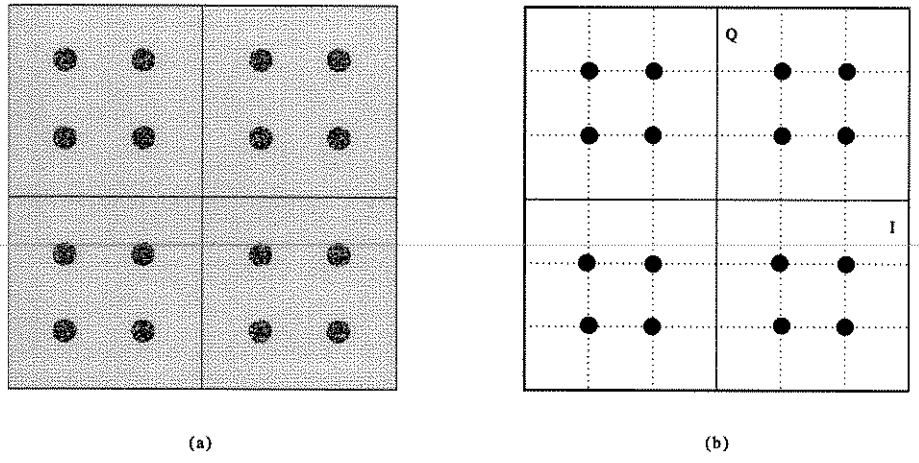


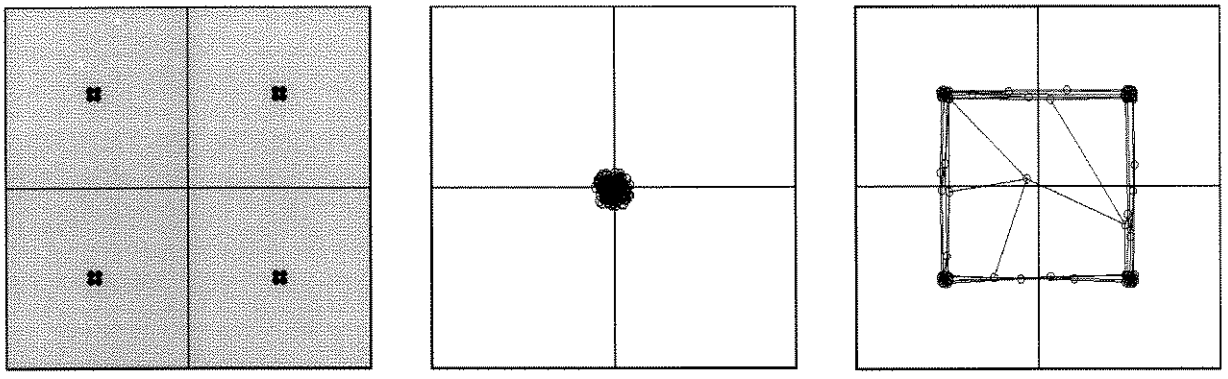
Figura 4.13.(a) Conjunto de entradas $\mathbf{x}(t_k)$. (b) Constelação 16QAM ideal.

A figura 14.13.a, representa uma constelação de sinais/símbolos transmitidos usando a modulação 16-QAM, os sinais normalmente estão associados a *centróides* ao redor dos quais produzem-se *clusters* de sinais [57]. Estes devem ser classificados adequadamente por um receptor que receba os sinais contaminados por ruído no canal de transmissão.A figura 4.13.b, representa uma constelação ideal, onde deveriam convergir os sinais transmitidos. O algoritmo básico SOFM foi modificado por Raivio para adequá-lo às necessidades da aplicação. A vizinhança variável foi eliminada assim como o ganho dependendo do tempo. A função $\alpha(t_k)$ foi substituída por parâmetros constantes, da mesma forma que a função $N_c(t_k)$.

4.6.1.2. Simulações.

Para demonstrar a forma que Raivio et al. tem utilizado os mapas SOFM na aplicação de estruturas de equalização adaptativa, serão apresentados os resultados de diversas simulações, comparando o algoritmo original com o modificado e proposto por Raivio. É necessário destacar

que esta é uma atividade de pesquisa em andamento e que ainda devem ser realizadas simulações exaustivas que permitam definir os parâmetros adequados para uma arquitetura dedicada a resolver eficientemente este tipo de problemas. O objetivo desta secção é demonstrar o potencial do algoritmo em aplicações para uso em tempo real.



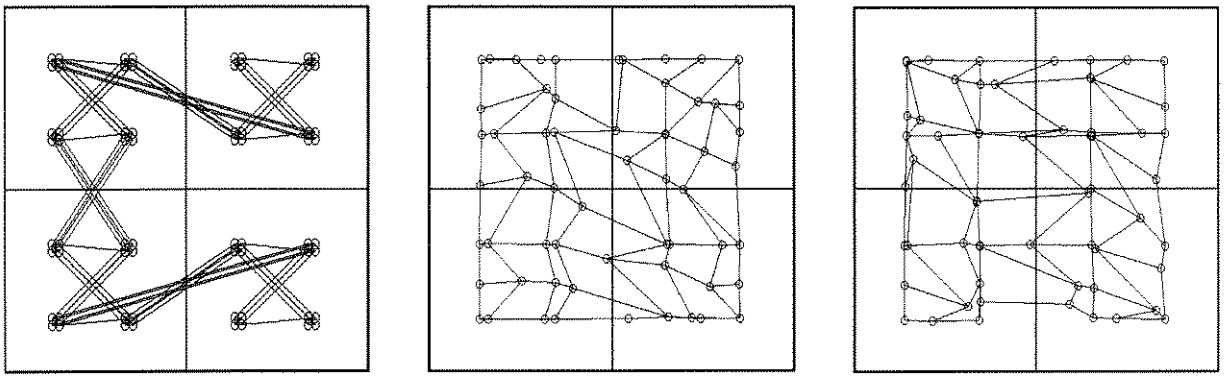
(a) (b) (c)

Figura 4.14. Simulação para aplicação do SOFM em Equalização Adaptativa. Algoritmo convencional.

Na figura 4.14, representa-se o caso do algoritmo SOFM convencional numa primeira abordagem para aplicação em equalização adaptativa em modulação 4-QAM. Os sinais de entrada de (a) representam *clusters* de sinais, (b) a inicialização centrada na origem e (c) o mapa resultante após 9000 iterações. A rede é composta por 100 células e os parâmetros da função de ganho $\alpha(t_k)$, expressada por (4.2) são k1=0.1 e k3=0.002.

Da figura, pode-se inferir, em termos gerais, que o algoritmo se comporta adequadamente ao representar a distribuição de entradas de (a). No entanto, para a aplicação de interesse, há vários vetores no mapa resultante ilustrado em (c) que não foram adequadamente classificados, no total são 20 células que ficaram fora dos *clusters* adequados.O resultado de 20% de erro não é interessante para uma aplicação real.

Observaram-se mapas resultantes similares ao inicializar a rede com vetores pesos aleatórios no intervalo [0,1], tal como é representado na figura 4.10.b. Por outro lado, a modulação 4-QAM, não é atualmente muito adequada para resolver problemas de comunicação real, uma estrutura mais utilizada seria a modulação 16-QAM.



(a) (b) (c)

Figura 4.15. Simulação para uma abordagem 16-QAM. Rede de 64 células, algoritmo SOFM convencional.

Na figura anterior, pode-se observar que os mapas resultantes não conseguem representar adequadamente a **pdf** de entrada representada pela figura 4.13(a), utilizada nesta simulação como entrada. As figuras 4.15(b) e (c) representam o resultado da inicialização dos vetores pesos com os sinais ilustrados em 4.15(a) e 4.14(b) respectivamente.

Ainda variando os parâmetros de (4.2) não é possível conseguir mapas resultantes adequados, utilizando o algoritmo SOFM convencional. Esta foi a razão que levou a Raivio et al. a modificá-lo, de forma de adequá-lo às suas necessidades [58]. As modificações mais relevantes ocorrem no uso de uma função de ganho e de vizinhança constantes no tempo.
Um outro aspecto é relativo à inicialização dos pesos da rede, sendo que para esta aplicação eles são inicializados com valores aproximados quando os neurônios são localizados em vizinhanças topológicas próximas, aspecto este ilustrado por meio da figura 4.16(a). Os resultados das simulações, considerando estas mudanças são apresentados na seqüência.

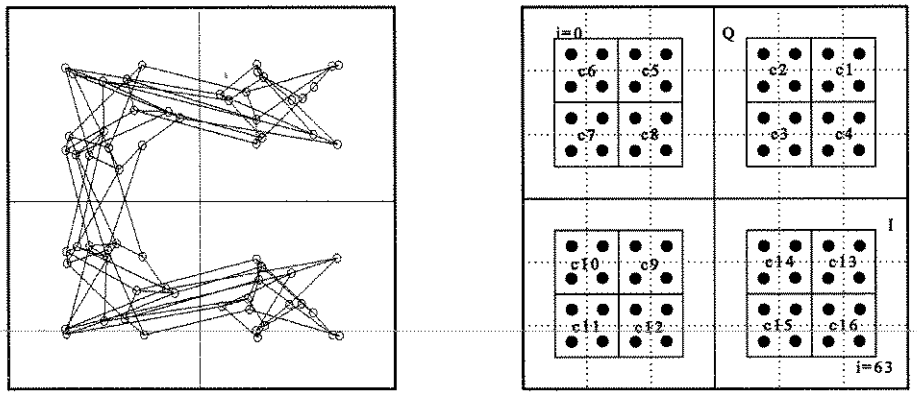


Figura 4.16.(a) Mapa resultante algoritmo SOFM modificado. $\alpha(t_k)$, $N_c(t_k)$=ctes.
(b) c1,....,c16 : *Clusters* agrupando células com pesos iniciais próximos p/(t_k=0)

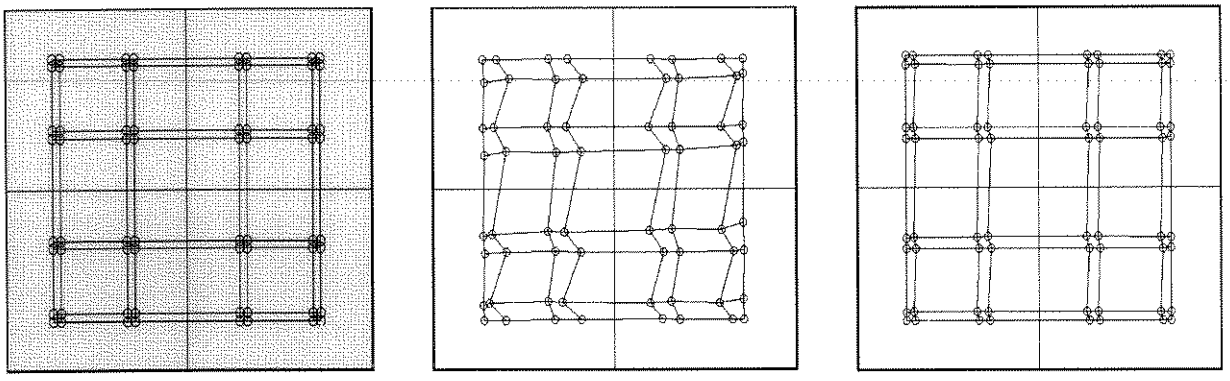
Na figura 4.16.(a), representa-se um mapa resultante do produto de uma simulação com valores iniciais semelhantes aos da figura 4.15.a e entrada tal como a da figura 4.13.a. Os pesos não foram inicializados como proposto por Raivio et al., podendo-se observar que o mapa resultante é distorcido em relação à distribuição da entrada usada como treinamento. Na figura 4.16(b) representa-se um conjunto de *clusters*, onde se agrupam as células com valores iniciais próximos. Foram definidos 16 clusters c1,...c16, onde cada um agrupava 4 células, gerando uma rede de 64 neurônios, num arranjo com topologia de vizinhança hexagonal.

Na figura 4.17 apresenta-se o resultado da simulação com as seguintes alterações:
(i) A função de ganho $\alpha(t_k)$ é constante, assim como a função de vizinhanca $N_c(t_k)$. Foi utilizado um raio fixo=1. A rede foi treinada com o conjunto de entradas ilustrada na fig. 4.13.

(ii) Outra modificação introduz 2 parâmetros que multiplicam a diferença $\{\mathbf{x}(t_k)-\mathbf{m}(t_k)\}$, na expressão do capítulo 3 (3.x): α_0 no caso da célula ganhadora e β_0 para as células dentro da região de vizinhança de raio fixo.

A fig. 4.17(a) representa os valores iniciais dos vetores peso (t_k=0), (b) e (c) representam os mapas resultantes com as seguintes variações para os parâmetros α_0 e β_0 :

α_0 = (a):0.003 (b):0.0005 β_0 = (a):0.001, (b): 0.0001



(a) (b) (c)

Figura 4.17. Simulação para aplicação em 16-QAM. Algoritmo SOFM modificado segundo proposta de K. Raivio et al. (a) Vetores pesos em t_k=0. (b) e (c) mapas resultantes após 9000 passos, segundo os parâmetros α_0 e β_0

Na figura anterior, uma rede de 64 células foi treinada com as mesmas entradas ilustradas na figura 4.13(a). Após 9000 passos de treinamento, os mapas resultantes são ilustrados em (b) e (c), onde os parâmetros α_0 e β_0 correspondem aos valores descritos acima.

A partir dos resultados observados, pôde-se verificar que a modificação proposta por Raivio et al. é relevante no desempenho do algoritmo, mesmo assim pode-se observar que, apesar do mapa de 4.17.(c) ser melhor definido que o da fig, 4.17(b), ainda persistem distorções no fim do treinamento. A modificação do algoritmo SOFM e a arquitetura dedicada para sua implementação em *hardware* é o eixo temático de uma pesquisa em andamento.

4.6.2. Uma aplicação no reconhecimento de sinais acústicos.

4.6.2.1. Descrição.

Nesta secção descrever-se-á a utilização do algoritmo SOFM no reconhecimento de sinais correspondentes ao timbre de diversos instrumentos acústicos armazenados em um *sampler*.

Foram realizadas simulações do modelo com o objetivo de utilizá-lo na discriminação das diferenças timbrísticas de tons musicais. O método consistiu em treinar a rede apresentando-lhe uma seqüência de 17 amostras diferentes de tons equivalentes a instrumentos orquestrais. No final da fase de treinamento, foi observada a formação de mapas auto-organizados num plano bi-dimensional, onde ocorrem agrupamentos por família instrumental. Numa fase posterior, apresentaram-se vetores correspondentes a vários tons musicais similares, com a finalidade de verificar a capacidade de reconhecimento do modelo. A capacidade de reconhecimento e a classificacão correta dos padrões, dentro de uma taxa de erro razoável, está diretamente relacionada com a geração de mapas topológicos cuja qualidade depende essencialmente das propriedades de convergência e da estabilidade do modelo.

Como conclusão desta experiência, verificou-se que o algoritmo é adequado para o reconhecimento de padrões timbrísticos, com baixa taxa de erro.

4.6.2.2. Metodologia e Procedimento Experimental.

Todas as experiências foram realizadas utilizando o algoritmo SOFM convencional, descrito pela expressão 3.7. Considerou-se uma rede com topologia de vizinhança hexagonal composta por um arranjo de 12 x 8 células, cujos vetores de pesos **m** e de treinamento **x** são definidos por: $\mathbf{m}_j = [m_1 \ldots m_d] \in \mathbb{R}^d$ e $\mathbf{x} = [x_1 \ldots x_d] \in \mathbb{R}^d$, onde $0 \leq j \leq 95$ e d=48.

O formato dos vetores de treinamento é apresentado na figura 4.17, representando a concatenação de 3 vetores correspondentes à captação do sinal em 3 instantes diversos, dentro de um intervalo de cerca de 500 ms de duração. Cada vetor representa o espectro discreto, calculado por meio da FFT correspondente ao instante de tempo t_k, onde $0 \leq k \leq 2$.

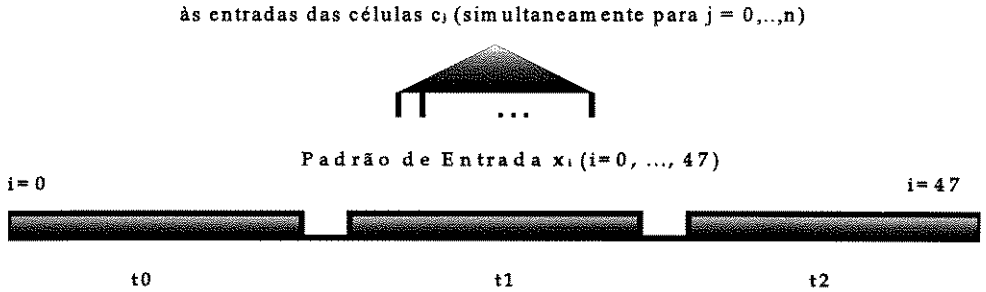


Figura 4.18. Formato dos vetores de treinamento $\mathbf{x}(t_k)$, n=96 neurônios.

Cada vetor de entrada t_k na figura 4.17, corresponde à estimativa do *Espectro de Potência* implementada pelo *método Welch* [59] e calculado pela ferramenta *MatLab*™. Tentou-se desta forma representar 3 características do sinal que são relevantes na percepção do timbre, segundo as evidências de estudos em acústica fisiológica [60], [61], [62]: o *ataque*, a *sustentação* e o *decaimento*.

O sinal acústico no instante t_k é representado por um vetor de 16 componentes. Foram captados sinais em 3 instantes e concatenados para formar um só vetor de treinamento de 48 componentes, de forma similar à proposta de J. Kangas [63]. O vetor t_0 corresponde ao instante do início do sinal onde o espectro de potência varia rapidamente, t_1 corresponde à parte estacionária do sinal e t_2 ao decaimento, onde a densidade de energia acústica normalmente diminui gradualmente. Os 17 timbres escolhidos para treinamento abrangem uma ampla gama de instrumentos musicais e pertencem à família ou classe das cordas, madeiras, metais e percussão. A classificação das famílias de instrumentos utilizadas obedeceu à seguinte convenção:

Classe A - Cordas :	violino, violoncelo, contrabaixo, pizzicato de violino.
Classe B - Madeiras :	oboe, flauta, clarinete, corne inglês.
Classe C - Metais :	tuba, trombone, trompete, trompa.
Classe D - Percussão :	queixada, bumbo, caixa, prato orquestral, apito.

A metodologia utilizada é fundamentada na proposta de J. Kangas, que a aplicou em redes do tipo TDNN (*Time-Delay Neural Network*) com a finalidade de acrescentar informação de contexto em tarefas de reconhecimento de fonemas. Nas nossas simulações, devido ao baixo número de dados experimentais, os vetores foram reapresentados primeiro seqüencialmente e depois em ordem aleatória , a fim de verificar a sua influência nos mapas resultantes.

4.6.2.3. Resultados.

O conjunto de 17 vetores de treinamento foi reapresentado à rede um número variável de vezes, produzindo entre 17 (1 ciclo) e 98600 passos (5800 ciclos) de iteração, na fase inicial de aprendizado, modificando-se inclusive a ordem de apresentação. Um par de exemplos de mapas resultantes são apresentados na seqüência.

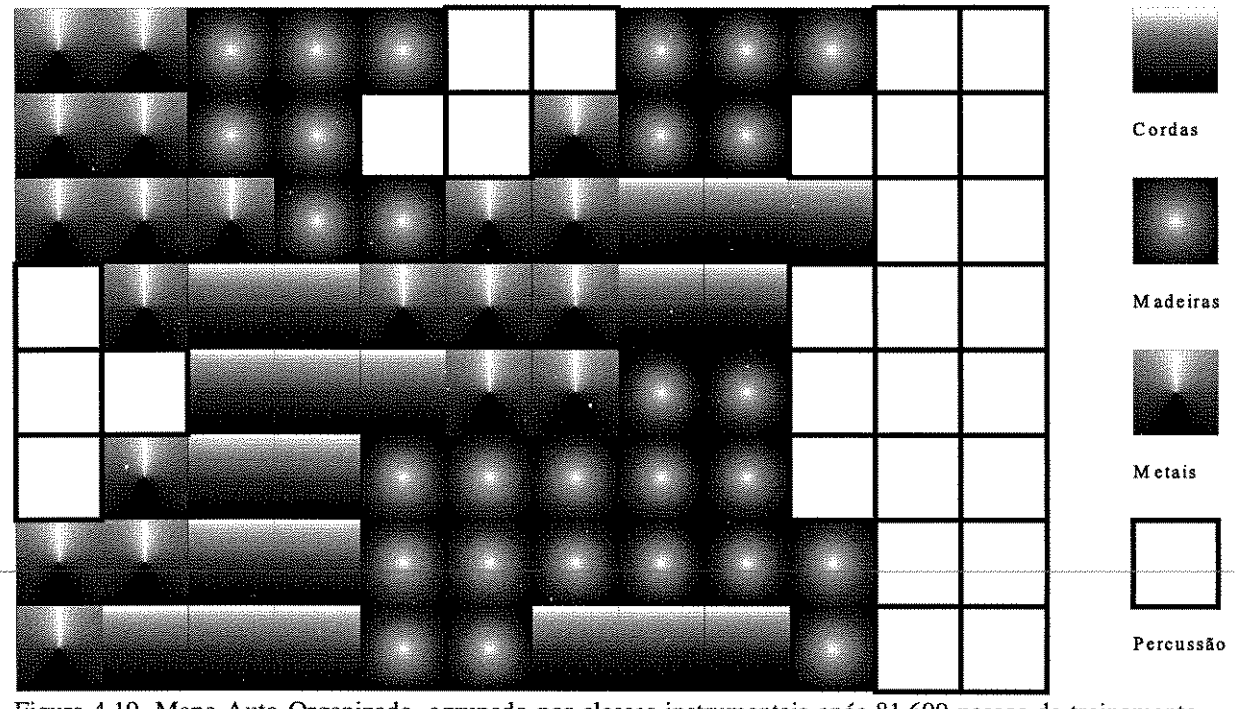


Figura 4.19. Mapa Auto-Organizado, agrupado por classes instrumentais após 81.600 passos de treinamento

O mapa apresentado na figura 4.19 foi obtido após 81.600 passos de treinamento, utilizando uma função de ganho $\alpha(t_k)$ não linear. Observam-se regiões que correspondem às diferentes classes ou famílias instrumentais. Cada unidade deste mapa representa uma célula do arranjo de 12×8 que ficou sintonizada num padrão definido. Após a fase de treinamento, apresentou-se à rede uma série de 68 vetores de entrada que representavam uma seqüência de vetores desconhecidos, definidos por pequenas variantes dos timbres usados para treinamento. A rede conseguiu reconhecer aqueles padrões, com uma taxa de erro mínima de 4.4%, chegando no pior caso a taxas da ordem de 35%, quando utilizadas curvas de ganho $\alpha(t_k)$ inadequadas. O mapa da figura 4.19 produz uma taxa de erro de 7.35 %, i.e., de 64 vetores desconhecidos, 5 foram classificados de forma errada. O resultado é considerado satisfatório.

Na figura 4.20 apresenta-se um outro mapa obtido após 64.600 passos de treinamento, em (a) se ilustra o mapa resultante, em (b) estão destacadas as células que estão melhor *sintonizadas* com cada um dos vetores de entrada, representando a cada instrumento.

Tal como nos casos anteriores descritos neste capítulo, nesta experiência se repete o fato da qualidade dos mapas resultantes ser fortemente dependente dos parâmetros do algoritmo que controlam a função de ganho $\alpha(t_k)$ e/ou a função de vizinhança $N_c(t_k)$. Os piores resultados foram obtidos com mapas finais de baixa qualidade, apresentando muitas regiões inconexas.

Mapa auto-organizado de uma rede de 12 x 8 células, taxa de erro de 4 %

I	I	A	A	A	A	L	L	L	N	N	N
I	G	G	D	D	D	L	L	N	N	N	O
F	G	G	G	D	K	K	C	H	H	O	O
F	G	G	D	K	K	K	H	H	H	O	O
Q	Q	E	I	K	K	K	C	H	H	O	O
Q	E	E	E	K	J	J	C	C	C	O	O
Q	Q	E	P	P	J	J	J	C	C	M	M
Q	E	P	P	P	J	J	B	B	B	M	M

Cordas

Madeiras

Metais

Percussão

Cordas:Q(violino), E(violoncello), C(contrabaixo), L(pizzicato)

Madeiras : K(oboe), I(flauta), F(clarinete), D(corne inglês)

Metais: P(tuba), O(trombone), N(trompete), J(trompa)

Percussão: M(queixada), H (caixa), G(prato orquestral), A(apito)

I				A			L				N
F		G		D							
					K			H			
											O
								C			
Q		E				J					M
			P					B			

Figura 4.20. Mapa resultante para uma rede de 12 x 8 células, após 64.600 passos de treinamento.

Capítulo 5

Desenvolvimento de um protótipo de um *Neuroprocessador* para o SOFM

Escopo

Neste capítulo é iniciada a discussão da implementação em *hardware* do algoritmo SOFM descrito nos capítulos anteriores. O estudo é conduzido à proposta de uma arquitetura e ao projeto de um *neuroprocessador* digital dedicado chamado de ***Neuron***. Uma rede composta por um arranjo de células foi modelado integralmente com a linguagem de descrição de *hardware* VHDL (*Very High Speed Circuits **Hardware Description Language***). O projeto da arquitetura de *Neuron* foi desenvolvido por meio do modelamento, descrição e síntese usando VHDL.

Na seqüência, descrever-se-á a proposta e o projeto VLSI de uma arquitetura especificada e modelada na linguagem VHDL em diferentes níveis de abstração. O projeto foi desenvolvido integralmente por meio do CAD *framework* de *Mentor Graphics Co*™. O algoritmo SOFM básico e as suas variantes foi levemente modificado para adequá-lo às necessidades e às especificações do projeto de *hardware*. O modelamento foi inicialmente uma tradução do algoritmo desde a linguagem ANSI-C para a linguagem VHDL, descrito de forma comportamental em alto nível de abstração (algorítmico). São apresentados os resultados de simulações para uma rede composta por um arranjo de 6×6 células, a fim de definir os aspectos mais relevantes da implementação do SOFM. Os resultados do modelamento em VHDL são comparados aos resultados das simulações implementadas em ANSI-C.

5.1. Introdução

5.1.1. Metodologia de projeto.

Neste capítulo será iniciada a descrição do projeto da arquitetura do *neuroprocessador* digital dedicado a implementar o algoritmo SOFM. Por meio do uso extensivo da linguagem VHDL [64], o sistema foi modelado e simulado em diferentes níveis de abstração. Empregou-se inicialmente a metodologia de projeto *top-down* (vide Figura 5.1.), iniciando-se o estudo com simulações implementadas em linguagem de alto nível (ANSI-C), descritas no capítulo anterior. O algoritmo foi posteriormente transformado a uma descrição comportamental em alto nível de abstração na linguagem VHDL. Neste primeiro nível de modelagem (*top-level*), utilizou-se a linguagem VHDL, a fim de transformar os algoritmos escritos originalmente em ANSI-C. A partir deste primeiro nível de modelagem, foi realizada uma série de simulações com a finalidade de verificar a funcionalidade do modelo e a influência dos parâmetros do algoritmo, como curva de ganho, função de interação lateral, métrica de distância para a diferença vetorial, etc .

A primeira fase de modelagem é considerada essencial no desenvolvimento do projeto, já que blocos funcionais importantes devem ser especificados neste nível de abstração, antes de começar a descrição RTL, que dará início ao processo de síntese automática e posterior *layout*.

Neste capítulo será descrita a modelagem do neuroprocessador e serão apresentados os resultados das simulações do mesmo. Na secção seguinte serão descritos os aspectos básicos relacionados com síntese de circuitos, a metodologia *top-down* e o modelamento de sistemas em alto nível de abstração visando a síntese lógica de circuitos integrados. Na seqüência, será apresentada uma introdução à síntese lógica de estruturas dedicadas à implementação de algoritmos para processamento neural. A seguir, será descrito o fluxo do projeto a a organização principal da arquitetura. Nas secções seguintes, será descrita a estrutura dos componentes ou blocos básicos do processsador e a sua relação com o algoritmo principal. Nas secções finais serão apresentados os resultados de simulações, considerando a influência dos blocos e parâmetros no desempenho da rede. Finalmente será descrito o projeto do elemento básico correspondente ao bloco WTA (*Winner-Takes-All*) que implementa a árvore de busca binária, fundamental na fase de aprendizagem competitiva do algoritmo.

5.1.2. Síntese Automática de circuitos.

5.1.2.1. Síntese de Alto Nível.

O projeto de sistemas VLSI tem evoluído enormemente nos últimos anos. Sistemas altamente complexos podem ser descritos funcionalmente por meio de linguagens dedicadas de descrição de *hardware*, como VHDL [65] ou Verilog ou ainda por linguagens que implementem operações e processos concorrentes, como Occam, utilizado efetivamente na especificação e no projeto de desenvolvimento dos processadores *Transputers* da INMOS™.

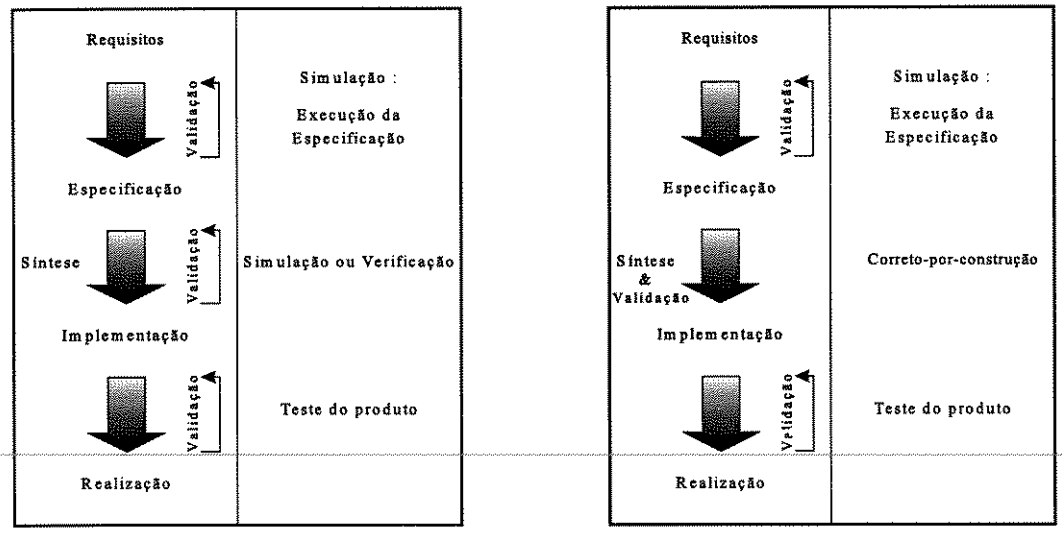


Figura 5.1. Metodologia *Top-Down*. (a) Sem verificação formal. (b) incluída a verificação formal.

Por meio do uso intensivo de linguagens de descrição de *hardware*, é possível descrever o comportamento de complexos sistemas eletrônicos concentrando-se especificamente nos aspectos funcionais do sistema, e portanto abstraindo-se dos detalhes relativos à implementação tecnológica. Este é talvez o aspecto mais relevante da metodologia *top-down*: a independência da implementação em determinada tecnologia, que permite especificar e desenvolver um sistema complexo de acordo com os objetivos operacionais e os requerimentos do projeto, como compactação de área de silício, custo, desempenho, etc.

Por meio do uso de técnicas avançadas de síntese de alto nível [66], [67], [68] é ainda possível a síntese de sistemas especificados em nível comportamental a uma descrição RTL ou a nível de portas lógicas, desta forma incrementando a complexidade do projeto. Algumas propostas incluem ainda a possibilidade do uso de técnicas de verificação formal [69], [70], para a obtenção de circuitos funcionalmente corretos, aspecto crucial na síntese de alto nível. Tal metodologia permite melhorar a eficiência no projeto, eliminando passos de verificação por simulação exaustiva, proibitivo no projetos de sistemas altamente complexos. A idéia é ilustrada na figura 5.1, em (a) se inclue uma operação reiterativa (*loop*) na fase de síntese sem verificação formal, o circuito sintetizado deve ser verificado por meio de simulações, ao contrário do fluxo

descrito em (b), onde a síntese de circuitos funcionalmente corretos é verificada formalmente.

A utilização de técnicas de síntese de alto nível acrescenta ainda outras vantagens: um algoritmo determinado pode ser mapeado em múltiplas arquiteturas ou em *hardware* específico, a partir de *transformações* realizadas durante a fase de síntese e otimização.As transformações se referem a passos específicos ordenados seqüencialmente durante a síntese do projeto, tal como é ilustrado na figura 5.2.

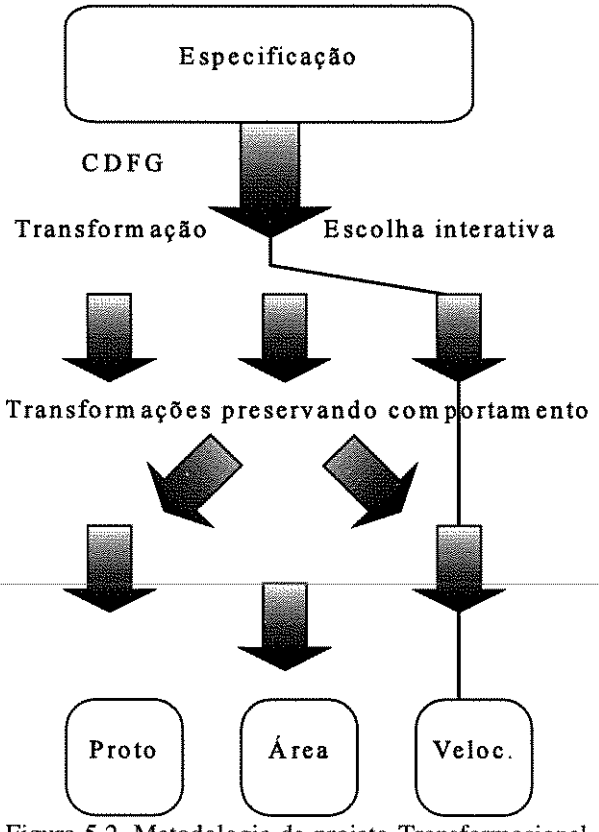


Figura 5.2. Metodologia de projeto Transformacional CDFG: *Control Data Flow Graph*. Proto: protótipo visando um compromisso entre área/desempenho.

Na figura anterior, ilustra-se uma metodologia de projeto *transformacional*, proposta por Middelhoek [70]. Transformações sucessivas verificadas formalmente são controladas de forma interativa pelo usuário da ferramenta de síntese, a fim de atingir o objetivo proposto. No exemplo, o objetivo é a obtenção de um circuito de alta velocidade, onde o compromisso com a compactação da área de silício é definida interativamente pelo projetista. Este é um método conhecido por **Correto-por-Construção** *(Correct-by-Construction)*.

No entanto, na maioria das atuais ferramentas de síntese, o fluxo é controlado pelo cálculo de estimativas de custo, algoritmos de otimização (ex. *simulated annealing*) e heurística e nem sempre é possível obter automaticamente circuitos eficientes. Devido ao uso intensivo de ferramentas de automação de projetos, a densidade de transistores por área de silício tem diminuido nos últimos anos, segundo Middelhoek, indicando que para projetos de circuitos de média complexidade, ainda algumas soluções manuais se apresentam mais eficientes, especificamente em termos de aproveitamento de área de silício.

Um outro problema relevante é o abordado por Dollas e Sterling [71] e refere-se à reutilização de *hardware*. Este aspecto trata da capacidade de reutilização de modelos sintetizáveis, interfaces padronizadas e da reutilização de subsistemas. Ele propõe o uso de uma versão de VHDL incrementado, que permita por exemplo reutilizar subsistemas não sintetizáveis em simulações, mas que permita a inferência de código sintetizável na fase da síntese.

5.1.2.2. Síntese Lógica.

Refere-se ao processo a partir do modelamento RTL (vide secção 5.1.2.3) até os níveis inferiores do *hardware*, compreendendo a alocação de recursos para a tecnologia específica e *netlits* otimizados segundo restrições de área e desempenho. A descrição RTL é traduzida a circuitos no nível de **portas**, compostos por exemplo por *standards cells*.

5.1.2.3. Níveis de Abstração e Representação de Sistemas eletrônicos.

Segundo verificado por autores como D.Gajski [72], as formas de representação de um sistema eletrônico mais utilizadas correntemente são :

(i) Comportamental
(ii) Estrutural
(iii) Física

(i) Comportamental: Especifica a funcionalidade de um sistema sem descrever explicitamente a sua implementação. O comportamento do sistema é especificado em função das suas entradas, a sua combinação e a sua evolução no tempo.

(ii) Estrutural: Descreve o sistema em função da sua composição estrutural, dos seus componentes e das suas conexões. O comportamento do sistema pode ser *inferido* sem que a sua funcionalidade seja explicitamente descrita.

(iii) Física: Neste nível, são especificadas as características físicas dos componentes descritos na especificação estrutural. A informação especificada refere-se a detalhes como tamanho dos componentes, relações, dissipação de calor, potência, posicionamento de pinos de entrada e saída, etc.

Níveis de Abstração.

Para cada um dos tipo de representação descritos, existe um conjunto de diferentes níveis, distingüíveis em função dos tipos de objetos utilizados. Normalmente tais objetos são restritos a transístores, portas, registradores e processadores.

Transístores: Os componentes descritos neste nível normalmente são transístores, resistores e capacitores. A funcionalidade é descrita em termos de equações diferenciais ou relações tensão-corrente. Uma *célula* pode representar um circuito incluindo os componentes mencionados e definida em termos dos *layouts* dos seus componentes.

Portas: Neste nível os componentes principais são portas lógicas (circuitos combinacionais) e *flip-flops* implementando operações *booleanas* e atuando como elementos de memória. Estes componentes podem ser agrupados para formar módulos aritméticos e de armazenamento, descritos por equações lógicas e máquinas de estados finito (FSM- *Finite State Machines*).

Nível de Transferência de registros (**RTL**) : É o nível de abstração onde os componentes principais são as unidades aritméticas e lógicas e as unidades de armazenamento. Entre eles há somadores, comparadores, multiplicadores, contadores, acumuladores, *buffers*, registradores, registradores de deslocamento, etc. Estes blocos compõem um sistema, descrito por diagramas de fluxo, conjuntos de instruções, máquinas de estados, etc. O sistema pode também ser descrito numa linguagem de descrição de *hardware*, sintetizável ou não.

Nível de processador/arquitetura: Constitui o nível mais alto de abstração. Os seus componentes são processadores, microcontroladores, memórias , etc., podem estar fixos em placas impressas (PCBs) e interligados por meio de conexões impressas. Um tipo particular de implementação é o *Multi Chip Module* (MCM), onde diversos *dies* (ou *microchips*) compartilham uma PCB. O sistema pode ser funcionalmente descrito por meio da linguagem natural, de linguagens de descrição de *hardware*, como VHDL ou Verilog ou algoritmicamente usando uma linguagem de alto nível.

A figura 5.3. representa a metodologia de projeto geral utilizada, incluídos os distintos níveis de abstração e especificação. O método é chamado por D. Gajski, de *Descreva e Sintetize* [72]. No projeto da célula *Neuron*, esta metodologia foi só parcialmente aplicada. Uma metodologia híbrida, conhecida como *Full Circle Synthesis* [73], parece adequar-se mais ao método utilizado, incluindo ambos os processos *top-down* e *bottom-up*.

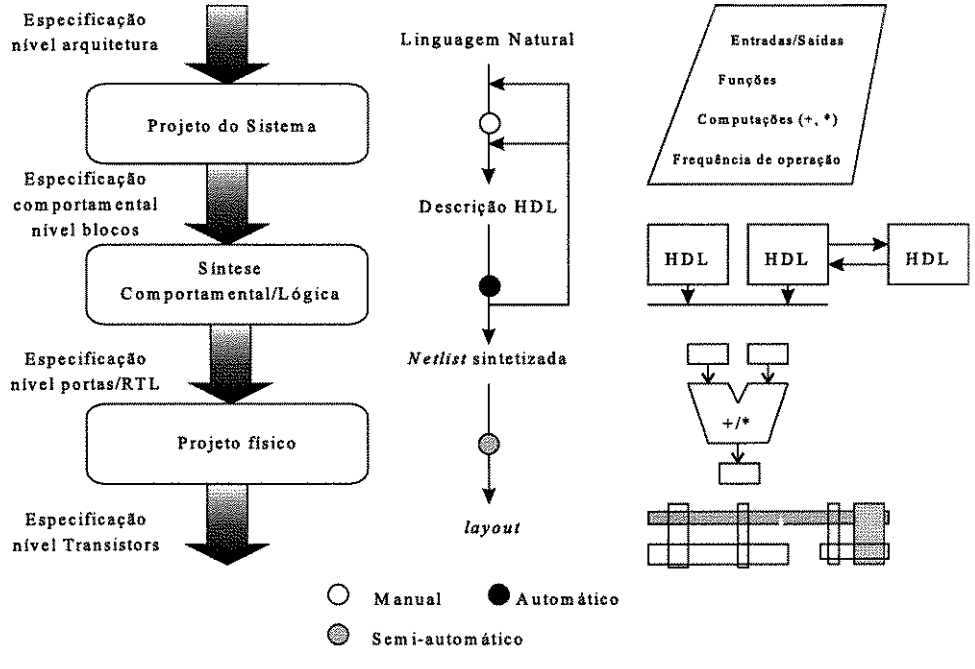


Figura 5.3. Diagrama para a metodologia *top-down* : Descrição e síntese

Na figura 5.3 ilustra-se o ciclo completo do projeto usando síntese automática. O ciclo (*loop*) na seqüência *Linguagem Natural → Descrição VHDL → Netlist* representa o processo iterativo após a síntese e simulação. O processo deve ser continuamente repetido enquanto não sejam atingidas as restrições impostas pelo projeto e enquanto não houver verificação funcional do circuito gerado pela síntese (vide figura 5.1).

5.1.3. Sobre a Síntese automática de circuitos neurais.

Devido às restrições impostas pelas ferramentas disponíveis e pela necessidade de eficiência em termos de espaço de silício, o processo de síntese foi realizado a partir da modelagem RTL, após a definição dos blocos principais da arquitetura de *Neuron*. O primeiro requisito do projeto é o aproveitamento máximo da área de silício disponível, portanto a otimização por área é prioritária. A síntese a partir do modelamento RTL é um processo consolidado pelas ferramentas utilizadas e a evolução do projeto pode ser melhor controlada pelo projetista, em cada fase de validação (vide figuras 5.1, 5.3). Toda a fase de síntese foi dirigida à obtenção de circuitos compactos, desta forma sacrificando a otimização dirigida à geração de circuitos de alto desempenho (velocidade). A justificativa é por causa do objetivo de tentar colocar o maior número possível de células neurais num só *chip*, um *die* de silício de 100 mm^2.

Para esclarecer um pouco mais o aspecto da síntese, será abordado o caso de um processo de síntese e de projeto automatizado dedicado a redes neurais artificiais (ANNs).

Uma proposta para a síntese automática de redes neurais foi feita por Achyuthan e Elmasry [74]. O método é descrito para a síntese de arquiteturas híbridas (*mixed*) compostas por circuitos analógicos e digitais. Naquela proposta, a rede é modelada por meio de grafos DFG (*Data Flow Graph*), sua saída é uma descrição em alto nível de blocos de circuitos analógicos e digitais, sendo que as não linearidades dos circuitos analógicos são tratadas por meio da modelagem comportamental, avaliando os seus efeitos na funcionalidade da rede. O uso de técnicas quantitativas para avaliação permite generalizar a metodologia para a maioria dos modelos de redes neurais; a técnica consiste na análise quantitativa de erros, gerados e propagados a partir de operadores e sinais compostos e não ideais. Um exemplo é baseado em aproximações de séries de Taylor e o seu efeito na implementação de um modelo *Back-Propagation*.

Os grafos DFG são utilizados exaustivamente na implementação de síntese de alto nível de circuitos digitais. Um exemplo simples de implementação para uso em síntese é descrito na figura 5.4., onde é utilizada uma descrição chamada de SIL (*Sprite Input Language*), uma linguagem intermediária entre linguagens de descrição de hardware (HDL) e compiladores de silício [70]. SIL é formado por grafos tipo CDFG (*Control Data Flow Graph*) que incluem nós condicionais, onde um mecanismo de controle permite a execução de nós. A figura representa a implementação de uma diferença com valor absoluto, cálculo usado para medir a distância entre a entrada **x** e o vetor de pesos **w**, no algoritmo SOFM, os nós que implementam a operação de subtração são executados dependendo da avaliação $x \geq y$. A descrição SIL é utilizada como uma linguagem *backbone* para síntese de alto nível usando transformações para projetos que incluam técnicas de verificação formal.

Por outro lado, utilizando a abordagem de Achyuthan, as operações essenciais do algoritmo SOFM podem ser representadas por DFGs da forma como é descrita na figura 5.5.

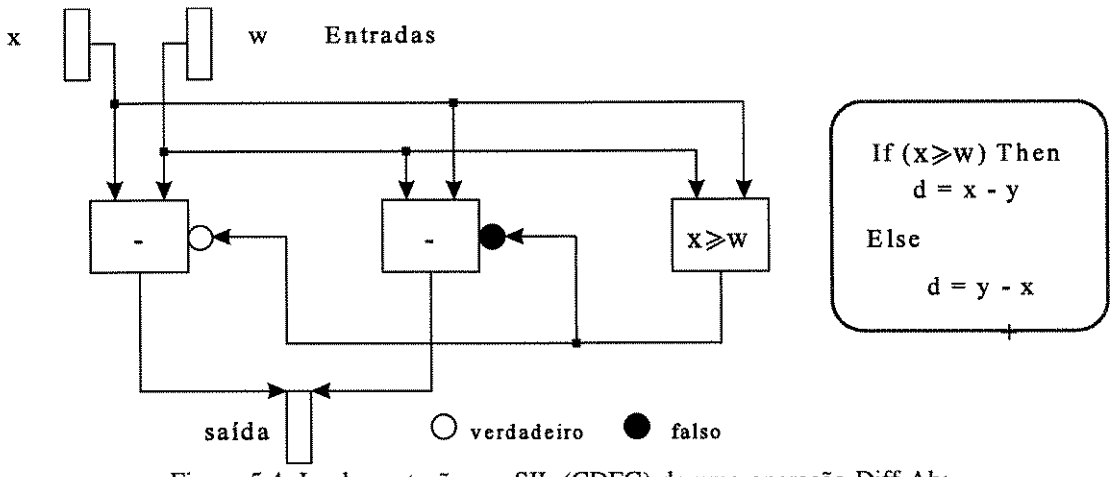


Figura 5.4. Implementação em SIL (CDFG) de uma operação Diff-Abs

Na figura 5.5. descreve-se a operação básica de atualização de pesos do algoritmo SOFM (vide expressão 3.7), implementada por um grafo DFG normal, sem o uso de nós condicionais.

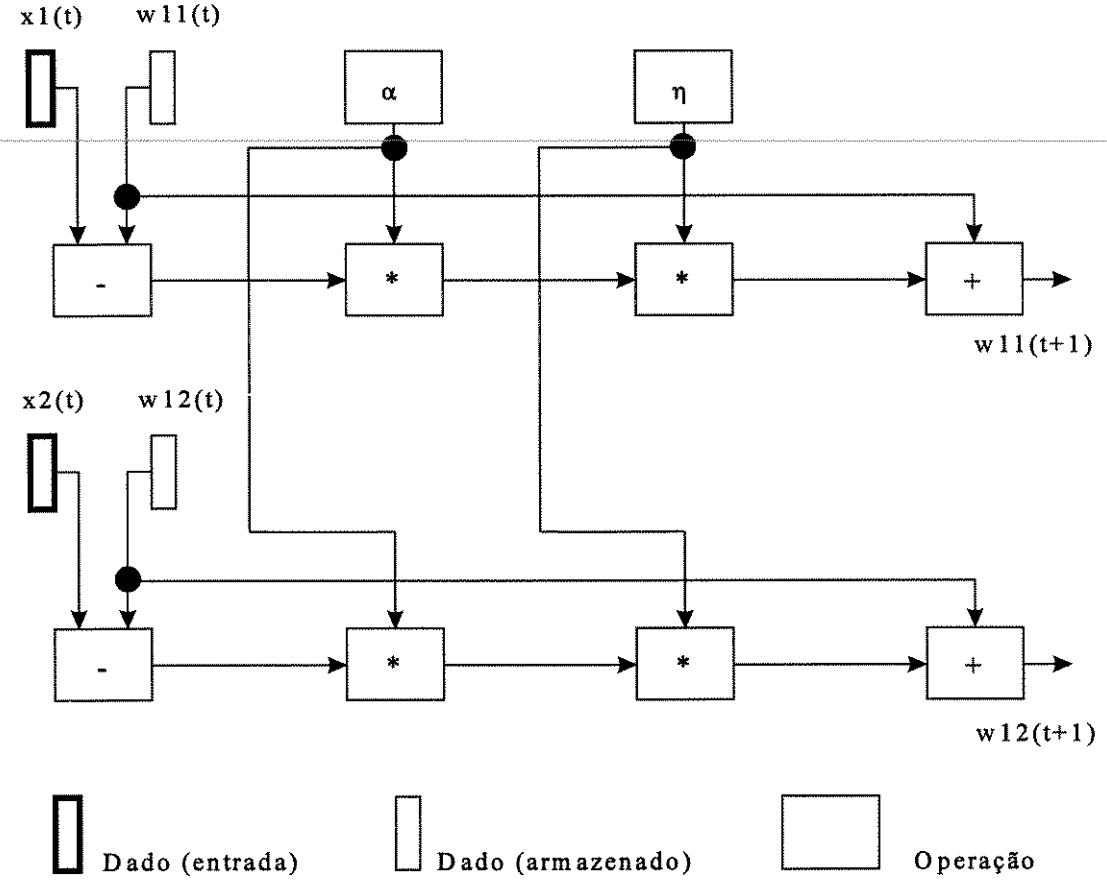


Figura 5.5. Implementação em DFG da operação de atualização de pesos do SOFM.
Os vetores de entrada **x** e os vetores de peso **w** da figura são bi-dimensionais, $\mathbf{x} = [x_1, x_2]$

Ambos os mecanismos são utilizados em síntese de alto nível de sistemas eletrônicos. A partir de descrições em grafos CDFG ou DFG, e utilizando transformações como uma forma de verificação formal, podem ser gerados circuitos funcionalmente corretos, o que é especialmente eficiente para o projeto de sistemas complexos, embora com topologia regular, como é o caso das redes neurais artificiais.

No projeto descrito a seguir, não tivemos à disposição ferramentas de verificação formal, sendo desenvolvido por meio da modelagem/descrição em VHDL, verificando a funcionalidade por meio de simulações exaustivas, tarefa que consome bastante tempo. A exploração de todas as possibilidades combinatórias dos sinais de entrada aos diversos blocos do sistema é proibitiva e as técnicas de verificação formal são mais eficientes para garantir que os circuitos gerados pelo processo de síntese sejam funcionalmente corretos.

Na secção seguinte será descrita a modelagem VHDL em alto nível da abstração da operação de uma rede neural e da célula de processamento elementar *Neuron*. Esta é a primeira fase na especificação da arquitetura definitiva, descrita no nível RTL por meio da linguagem VHDL e a partir da qual serão gerados os circuitos através de síntese automática.

5.2. Descrição do projeto da célula *Neuron*.

5.2.1. Requisitos e Especificações gerais.

O objetivo do projeto é a contrução de uma rede neural de Kohonen de alta flexibilidade, que permita implementar o algoritmo SOFM clássico, avaliado na secção 3.3. Entende-se como alta flexibilidade a capacidade de programar diferentes parâmetros, a fim de estudar a capacidade de convergência do algoritmo sob diferentes condições de operação, para diversas aplicações. Foi decidido que este modelo deveria implementar a máxima capacidade de processamento paralelo, da forma mais compacta possível. A prioridade pela compactação foi em detrimento de uma arquitetura com alto desempenho, em termos de velocidade de operação. O processo da síntese, mapeamento e otimização dos circuitos foi manejado com a finalidade de atingir os objetivos mencionados.

5.2.2. Fluxo do projeto.

O procedimento utilizado ao longo do projeto está ilustrado no diagrama de fluxo da figura 5.3. O processo foi iniciado por uma descrição em linguagem natural e pela tradução da implementação em linguagem de alto nível para o VHDL. A modelagem em VHDL comportamental (algorítmico) em alto nível de abstração precedeu à modelagem em nível RT, a partir de onde iniciou-se o processo de síntese. Alguns blocos foram implementados pelo método tradicional, usando captura esquemática, em função de algumas deficiências da versão VHDL sintetizável utilizada.

Os circuitos gerados por síntese automática foram exaustivamente simulados a fim de verificar a correta funcionalidade, tanto como blocos individuais quanto como sub-sistemas inseridos na arquitetura principal de Neuron. Os circuitos só foram validados após ter verificado a sua funcionalidade em ambos os níveis.

5.2.3. Modelagem comportamental algorítmica com VHDL.

5.2.3.1. A arquitetura da rede KNN (*Kohonen Neural Network*).

(a) Modelagem de uma Rede composta de 4x4 células.

O processo foi iniciado pela descrição de uma rede de 16 blocos funcionais representando as células neurais arranjadas numa topologia de vizinhança hexagonal, i.e., cada célula pode comunicar-se com 6 vizinhas por meio de conexões bidireccionais (vide figura 5.6).

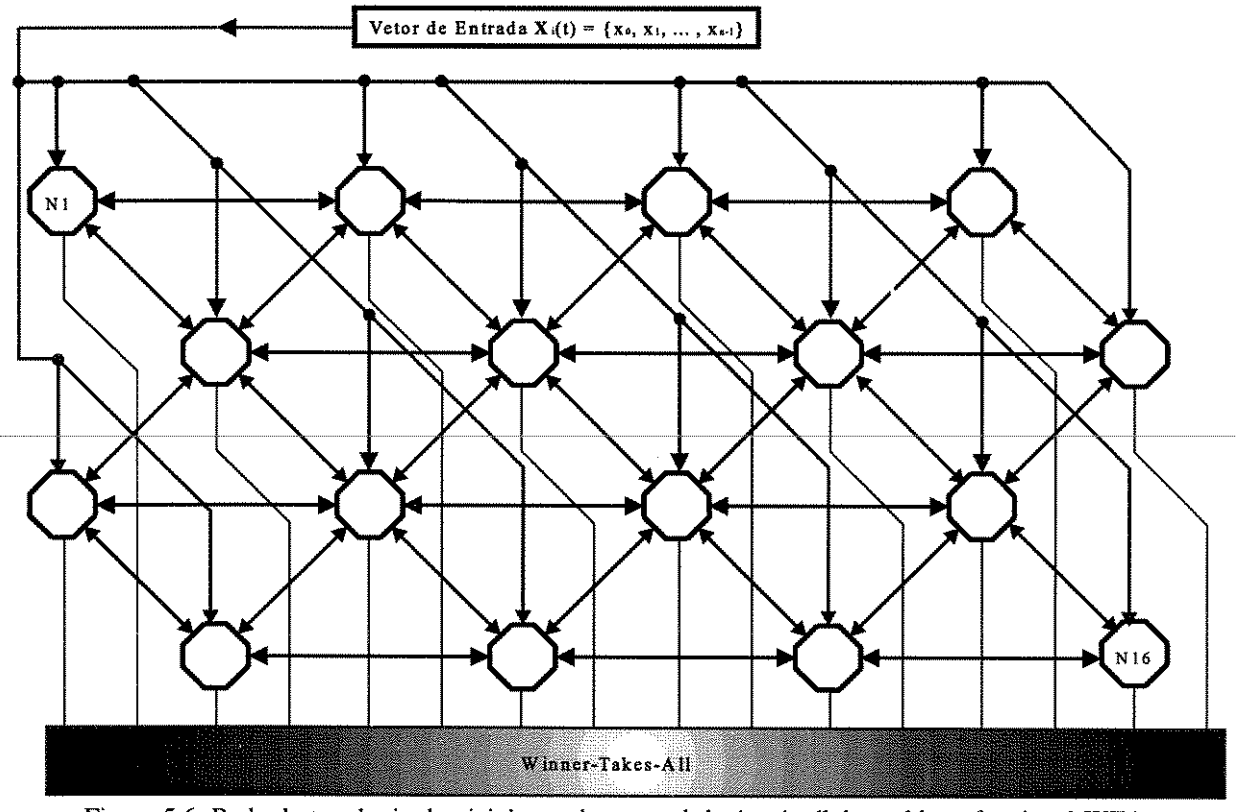


Figura 5.6. Rede de topologia de vizinhança hexagonal de 4 x 4 células + bloco funcional WTA.

Operação de aprendizado.

A figura 5.6 representa o modelo usado ao longo deste projeto. Cada célula tem a capacidade de conexão com 6 vizinhas e recebe o vetor de treinamento (entrada) $\mathbf{x}(t_k)$ de forma simultânea que o resto das unidades da rede. Cada célula i determina o cálculo de distância expresso por $|\mathbf{x}(t_k)\text{-}\mathbf{w}_i(t_k)|$, cujo valor é enviado ao bloco funcional WTA (*Winner-Takes-All*) que determina a célula ganhadora do processo competitivo no instante t_k. O bloco WTA retorna um sinal à célula ganhadora que inicia o processo de atualização dos seus pesos internos e que comunica às vizinhas imediatas que ela é a ganhadora nesse instante, atualizando assim os seus pesos em função do algoritmo de aprendizado descrito por (3.7). Este processo é repetido com um novo vetor de entrada $\mathbf{x}(t_k + 1)$, até o final da fase de treinamento.

(b) Processos Concorrentes Internos.

Nesta fase de modelagem a rede foi descrita estruturalmente, utilizando as células e o bloco funcional WTA. Tanto as células quanto o WTA foram descritas de forma combinada usando uma descrição estrutural e comportamental em termos das suas relações de entrada/saída e de uma série de processos concorrentes VHDL, comunicando-se por meio de sinais. As células foram decompostas numa descrição estrutural, onde cada processo interno torna-se um bloco funcional, que por sua vez pode ser decomposto de forma hierárquica. As unidades principais são:

(i) Bloco *Neuron* - descrição interna comportamental-estrutural
(ii) Bloco WTA - descrição comportamental.

5.2.3.2. O elemento de processamento *Neuron*.

O diagrama da figura 5.7. representa os processos internos à célula *Neuron*. Os processos se comunicam entre si e com o meio externo através de sinais, sendo ativados pela ocorrência de um evento em qualquer um dos seus canais de entrada.

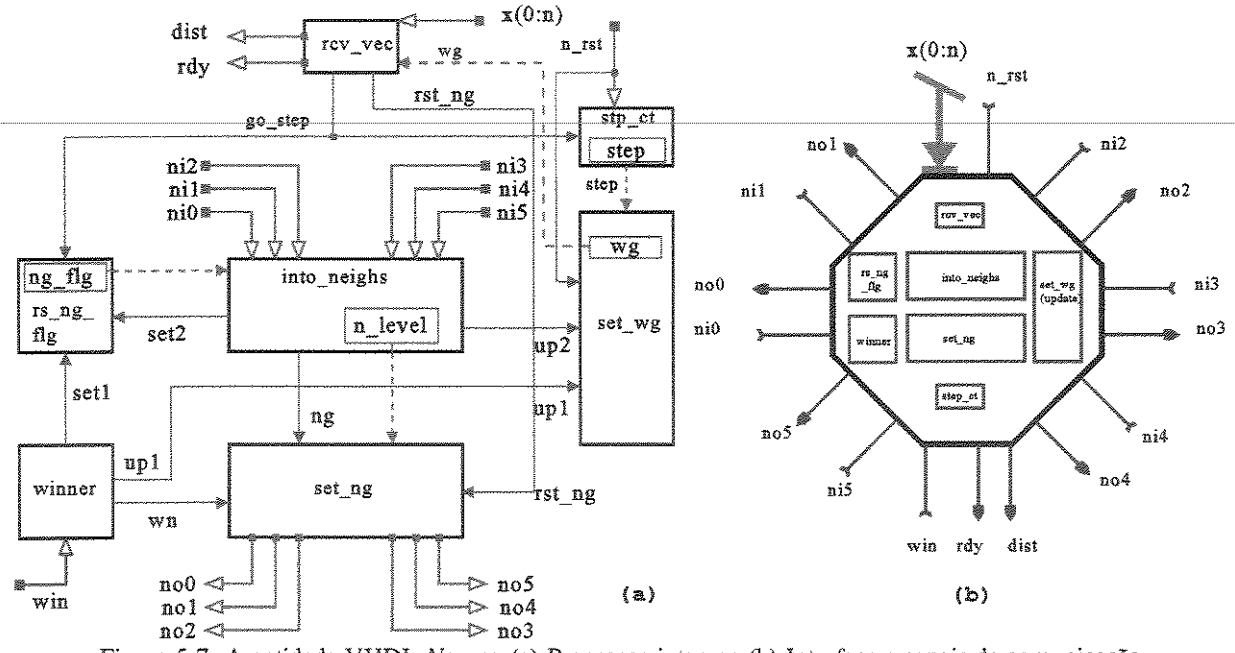


Figura 5.7. A entidade VHDL *Neuron* (a) Processos internos (b) Interface e canais de comunicação

A figura 5.7 representa o modelo da célula *Neuron*, elemento de processamento básico da KNN. A célula é modelada pela **entidade** VHDL *Neuron*, composta por um conjunto de processos concorrentes representados na figura 5.7.(a). A interface de *Neuron* com o meio externo, i.e., com a entrada à rede, as células vizinhas e o bloco WTA é representado em (b). A comunicação entre os processos é realizada por meio de canais externos e sinais internos VHDL. Uma **entidade** representa o módulo de processamento básico em VHDL, e o seu comportamento funcional é definido pela sua arquitetura composta pelo conjunto de processos concorrentes.

(i) Operação :

A célula *Neuron* recebe os sinais de treinamento pelo canal **x**(0:*n*), onde *n* representa a dimensão do vetor **x**. A distância é enviada ao bloco WTA a través do canal *dist*, junto com um sinal *rdy* (*ready*), indicando que o valor já está disponível na saída. O sinal *n rst* é ativado antes do início do treinamento para inicializar todos os processos internos adequadamente.

Ao receber um sinal externo desde o bloco WTA pelo canal *win* a célula atualiza os seus pesos e indica às vizinhas que ela é a ganhadora do processo competitivo nesse instante. Caso a célula não seja ganhadora, mas esteja dentro do raio de vizinhança definido pela função $N_c(t_k)$ (vide cap.2, 3), ela recebe um sinal por um dos canais de entrada conectados às vizinhas diretas (distância 1). Caso não ocorra nenhum dos casos anteriores, a célula não é ganhadora e não pertence ao raio de vizinhança corrente, o seu *status* é de espera por um novo vetor **x**.

(ii) Descrição dos Processos Internos.

Na linguagem VHDL, um processo possui uma *lista sensitiva* de sinais. O processo é **ativo** ao ocorrer um *evento* em qualquer um dos sinais dessa lista. Considera-se um evento à mudança de valor de um sinal.

Desta forma, o processo *rcv vec* da figura 5.7.(a) é ativado quando ocorre um evento no sinal de entrada **x**(0:*n*). A notação VHDL 0:*n* indica que 0 é o primeiro elemento do vetor **x** e *n* é o número total de elementos do vetor.

A seguir, apresentam-se os processos compondo a célula *Neuron*.

(a) **rcv vec** - *Recepção de Sinais Externos.*

Processo encarregado de receber os vetores de treinamento **x**(0:n), armazená-los e determinar a diferença vetorial $|\mathbf{x}(t_k)-\mathbf{w}_i(t_k)|$, cujo valor é enviado ao bloco WTA pelo canal *dist*. O processo tem acesso ao registrador *wg*, onde estão armazenados os valores do vetor peso no instante t_k, o passo corrente é armazenado no registrador *step* e é atualizado por *rcv vec*.

(b) **stp ct** - *Contador*

Processo encarregado de atualizar o passo corrente do algoritmo até o fim da fase de treinamento, utilizando o registrador *step*, que pode ser lido pelo processo *set wg*.

(c) **set wg** - *Atualização dos Pesos.*

O processo atualiza o valor do registrador *wg*, que armazena o valor do vetor de pesos da célula, ele é ativado por sinais vindo das células da vizinhança ou pelo processo detector do sinal de ganhadora enviado pelo bloco WTA, os pesos são inicializados pelo sinal *n rst*. Este processo executa a função $\alpha(t_k)$, descrita no capítulo 3.

(d) **into neighs** - *Cálculo da Função de Vizinhança* $N_c(i,t_k)$.

Este processo recebe o sinal de alguma célula vizinha quanda a célula não é ganhadora, mas se encontra dentro do raio de vizinhança corrente.Executa a função de interação lateral $N_c(i,t_k)$ (vide capítulo 3) e atualiza o valor de um registrador chamado de *n level*, que indica a distância entre a célula corrente e a ganhadora. Caso a célula corrente pertença à periferia da vizinhança (última linha dentro do raio corrente) o processo *set ng* não é disparado.

(e) **set_ng** - Envio de sinais às células vizinhas.

Processo ativado ao receber um sinal de *into_neighs*, após a ativação o processo transmite o valor atualizado do registrador *NLevel* para as 6 células vizinhas.

(f) **winner** - *Célula Ganhadora.*

Este processo é ativado por um sinal enviado pelo bloco WTA (canal *win*), indicando que a célula corrente é a ganhadora do processo competitivo no instante t_k. O processo deve ativar a atualização de pesos e a comunicação do *status* de ganhadora às 6 células vizinhas.

(g) **rs_ng_flg** - *Flag de ativação.*

Processo utilizado para inibir o efeito dos sinais enviados por outros processos vizinhos. Dada a possibilidade de receber sinais simultâneos por canais de entrada de distintas vizinhas, a célula corrente deve atualizar o peso só uma vez durante o passo t_k. A fim de evitar o efeito de múltiplas atualizações, um *flag*, armazenado em um bit em *ng_flg*, é ativado no instante da recepção do sinal de ganhadora, enviado pelo processo *winner*, ou quando recebido o primeiro sinal de algum canal de entrada conectado a uma vizinha imediata (n_i0, ..., n_i5).

(iii) Simulações.

Uma série de simulações foram realizadas utilizando os modelos descritos anteriormente. Os componentes foram compilados pelo utilitário *hdl* e simulados por *Quicksim*, ambas as ferramentas pertencem a *Mentor Graphics Corp.*

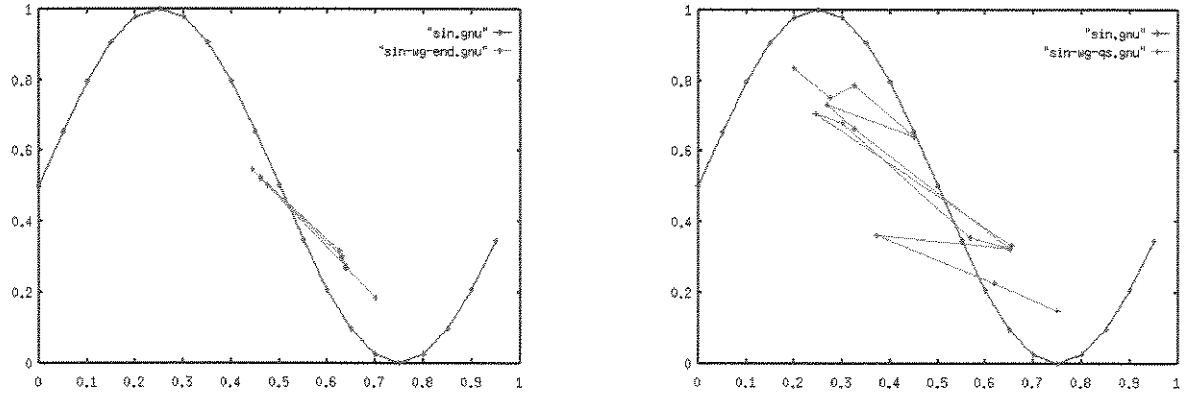


Figura.5.8. Resultados de simulações usando $N_c(t_k)$ fixa com raio 2.
(a) algoritmo implementado em ANSI-C. (b) simulação VHDL.

Os vetores de treinamento **x** foram gerados externamente e fornecidos seqüencialmente ao simulador. O número de iterações para a fase de aprendizado chegou até 4.000 em alguns casos. Os resultados foram comparados com simulações obtidas pela implementação do algoritmo SOFM na linguagem ANSI-C (vide figura 5.8). O resultado apresentado na figura 5.8 representa o desempenho de uma rede de 4x4 células para uma **pdf** de tipo onda senoidal.

Ao incorporar um número maior de células à rede, os resultados apresentados na figura 5.8. apresentam uma notável melhora. Os resultados apresentados a seguir foram obtidos usando os mesmos parâmetros utilizados nas simulações anteriores, mas modificando o tamanho do arranjo a 6x6 células. Na seqüência, diversos parâmetros foram alterados a fim de verificar a sua influência no desempenho do modelo, principalmente: métrica para medir a diferença entre vetores, função de ganho $\alpha(t_k)$ e função de interação lateral.

Todas as simulações foram realizadas numa rede de 6×6, com vizinhança topológica do tipo hexagonal.

Tabela 5.1. Dados das simulações 5.1 - 5.10.

No. Simulação	Fator $\Lambda(t_k,c)$ de Interação lateral	Variação Raio de vizinhança	Métrica de calc. de Distância	Função de ganho $\alpha(t_k)$
Sim 5.1	fixo	fixo = 2	dist. Euclidiana	Exponencial
Sim 5.2	fixo	fixo = 2	dist. Euclidiana	Linear (4)
Sim 5.3	fixo	fixo = 2	dist. Euc. Quad.	Linear (4)
Sim 5.4	fixo	var., max = 6	dist. Euc. Quad.	Linear (4)
Sim 5.5	fixo	var., max = 6	dist. Manhattan	Linear (4)
Sim 5.6	(3.5) discreto	max=6, min=1	dist. Manhattan	Linear (4)
Sim 5.7	fixo	fixo = 2	dist. Euclidiana	Discreta (9)
Sim 5.8	fixo	fixo = 2	dist. Manhattan	Discreta (9)
Sim 5.9	fixo	var. f.discreta	dist. Manhattan	Discreta (9)
Sim 5.10	(3.5) discreto	var. f.discreta	dist. Manhattan	Discreta (9)

Os dados da Tabela 5.1 são definidos a seguir, as 3 métricas usadas na determinação de distância entre o vetor de entrada $\mathbf{x}(t_k)$ e o vetor $\mathbf{w}(t_k)$ são (vide capítulo3, secção 3.4.3) :

Tabela 5.2. Definição das métricas para o cálculo da distância vetorial (d, dimensão dos vetores)

Tipo de Distância	Definição matemática , $i \in [1, ..., d]$
Distância Euclidiana	$d^2(\mathbf{w},\mathbf{x}) = \sum (w_i - x_i)^2$
Distância Euclidiana Quadrática	$d(\mathbf{w},\mathbf{x}) = \sum (w_i - x_i)^2$
Distância Manhattan	$d(\mathbf{x},\mathbf{w}) = \sum \|(w_i - x_i)\|$

As expressões para a função de ganho $\alpha(t_k)$ exponencial foram definidas no capítulo 4 (vide secção 4.2.1). A versão linear foi construída a partir da composição de 4 retas, para os casos das simulações 5.2 a 5.6 (vide figura 5.9). Uma versão definida por uma função discreta

composta por 9 níveis utilizou-se nos casos das simulações 5.7 a 5.10 (vide figura 5.16). A função expressa por (3.5), o fator de interação lateral $\Lambda(t_k,c)$, foi redefinida considerando a sua implementação en *hardware* por meio de operações de soma e deslocamento (*shift right-left*). A versão utilizada nas simulações 5.6 e 5.10 foi definida por :

$$\Lambda(x_c) = \frac{1}{2^{(x_c/2)}} \qquad (5.1)$$

onde x_c é a distância entre a célula corrente dentro de $N_c(t_k)$ e a célula ganhadora **c**. Uma expressão semelhante à (5.1) pode ser implementada por uma combinação se somas e de operações de deslocamento esquerda/direita (multiplicação/divisão por 2), a sua implementação em *hardware* é descrita no capítulo seguinte.

A função $\alpha(t_k)$ utilizada nas simulações 5.1 a 5.6 é ilustrada na figura 5.9.

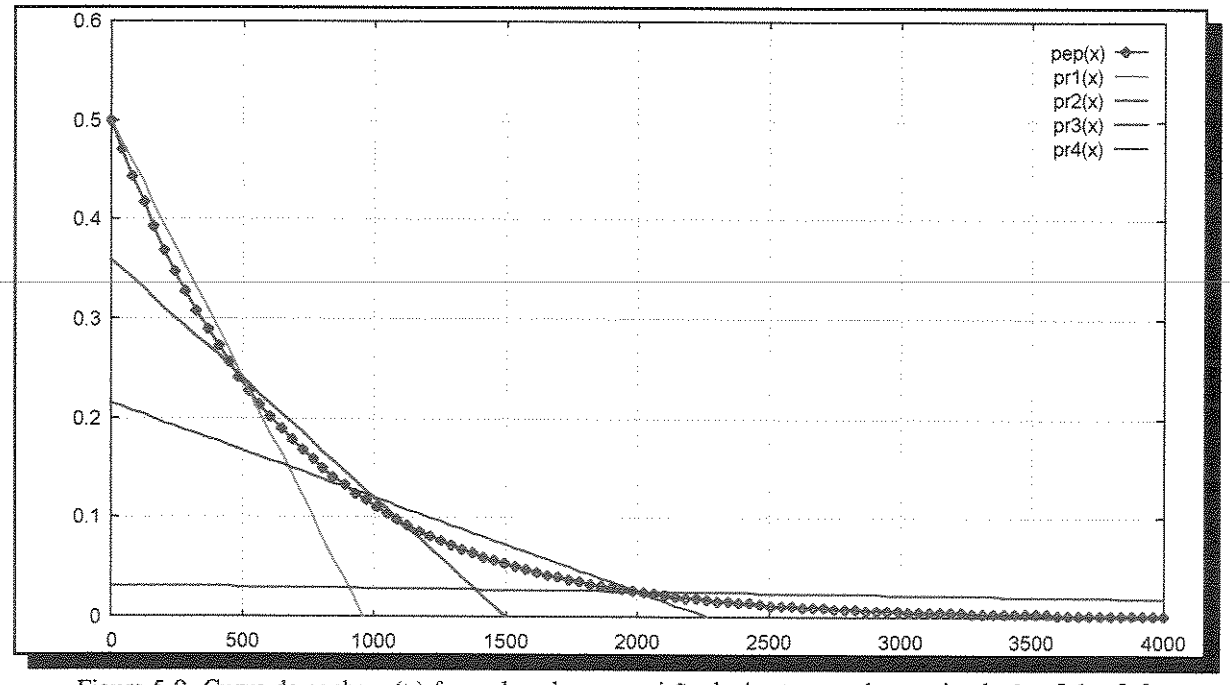


Figura 5.9. Curva de ganho $\alpha(t_k)$ formada pela composição de 4 retas, usada nas simulações 5.1 a 5.6.

A figura 5.9 representa a função $\alpha(t_k)$, implementada por 4 retas com inclinações diferentes (m_1, ... , m_4). A função de ganho é modificada assim que o passo de simulação armazenado no registrador *step_ct* é igual a um dos valores t_i (em *ns*), da figura (t1 = 500, t2 = 1.000, t3=2.000, t4 = 4.000). O número total de passos de simulação é de 4.000 ns.

Dados gerais para as simulações 5.1-5.10.

Vizinhança fixa, raio =2, Vizinhança variável : decrescente entre raio 6 e 1.
Ganho exponencial, linear e discreto. Exp.: $\alpha(t_k) = k1\ e^{(-k2tk)}$, (k1=0,05; k2=0,0015)
No. de vetores de treinamento : 20 No. total de iterações : 4.000 ns
Métrica de cálculo de distância : Dist. *Euclidiana*, Euc. Quadrática, Manhattan.

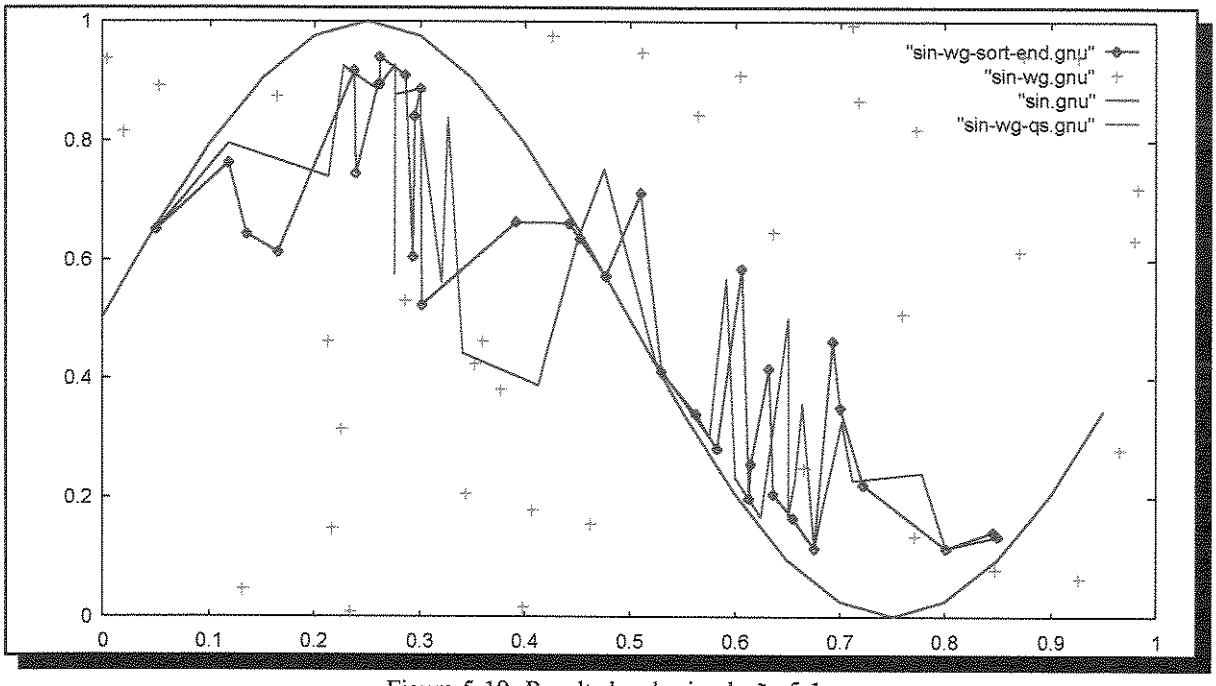


Figura 5.10. Resultados da simulação 5.1.

A figura 5.10 apresenta os resultados da simulação 5.1. A curva "sin-wg-sort-end" representa os pesos após 4.000 passos para a simulação em ANSI-C. Os pontos "sin-wg" representam os pesos no instante inicial (t_k=0), a curva "sin", a função de distribuição de entrada (pdf). A curva "sin-wg-qs" representa os pesos após 4.000 passos para a simulação da rede modelada em VHDL, executada por *Quicksim*. Os gráficos seguintes representam as mesmas funções e pontos definidas em 5.1, os dados correspondentes às tabelas 5.1 e 5.2.

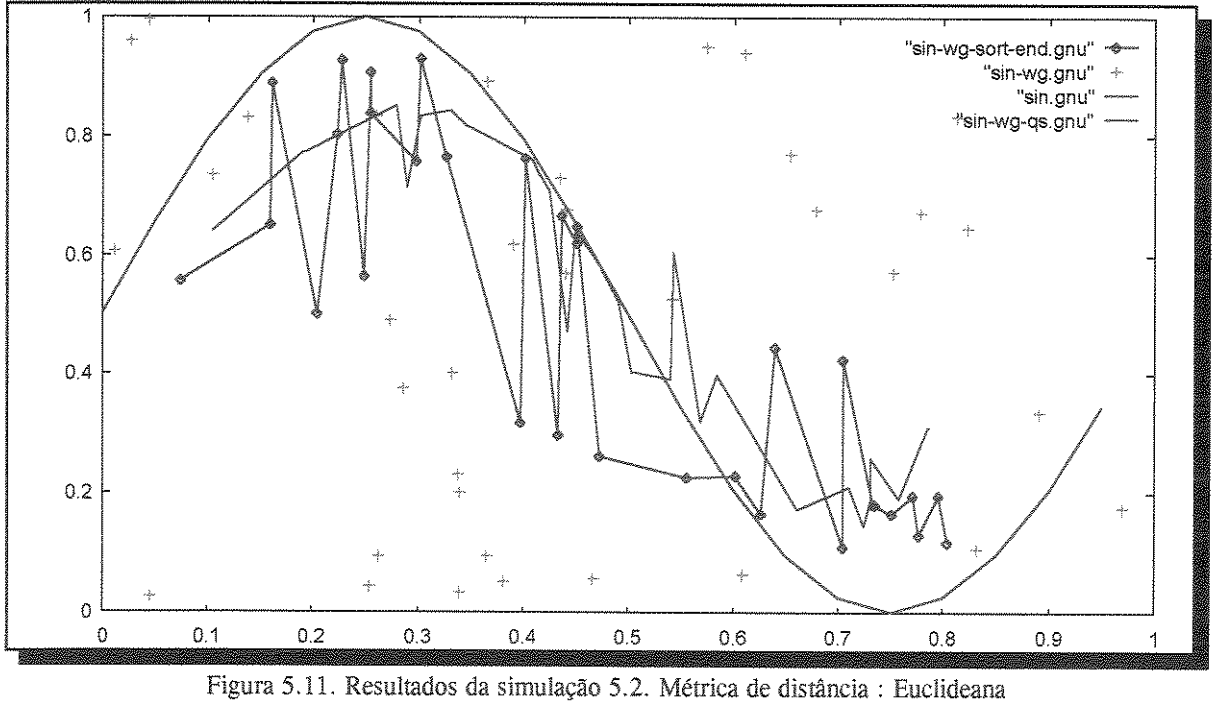


Figura 5.11. Resultados da simulação 5.2. Métrica de distância : Euclideana

A diferença entre as simulações anteriores é causada pela modificação da função $\alpha(t_k)$, a função de interação lateral é 1 em ambos os casos, foi usada a mesma métrica para a distância.

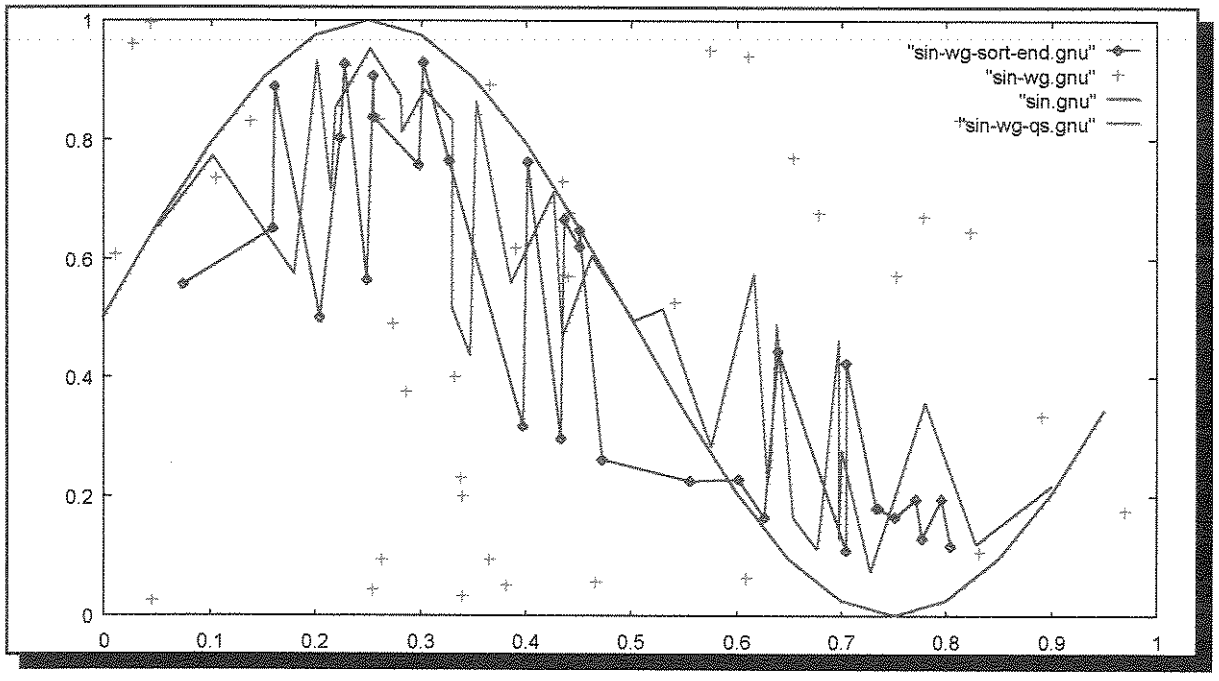


Figura 5.12. Simulação 5.3. Métrica de distância : Euclideana Quadrática, vizinhança fixa de raio 2.

A diferença entre as simulações 5.3 e 5.2 é causada pela métrica para o cálculo de distância, Euclidiana em 5.2 e Euc. Quadrática em 5.3 (vide tabela 5.2). Nos 3 casos descritos foi utilizada a vizinhança $N_c(t_k)$ fixa, com raio = 2.

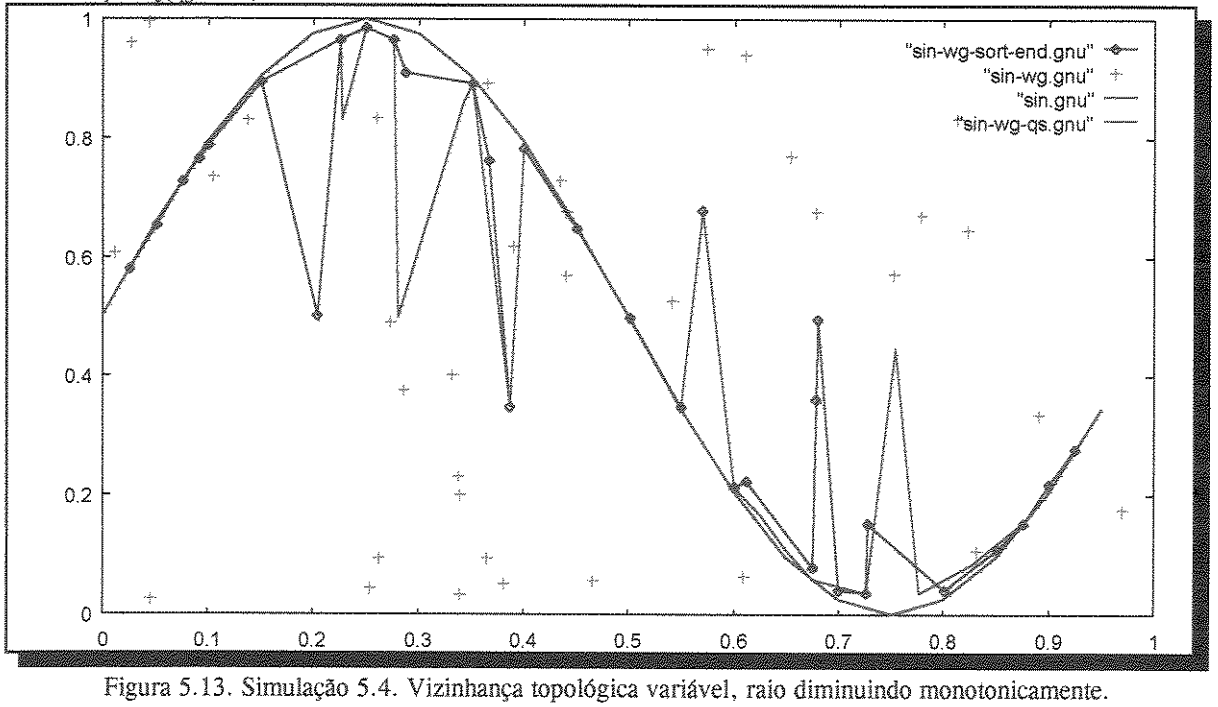


Figura 5.13. Simulação 5.4. Vizinhança topológica variável, raio diminuindo monotonicamente.

Na simulação 5.4, a vizinhança em torno da célula ganhadora decresce em forma monotônica entre um valor máximo de 6 até 1. A métrica de distância para 5.3 e 5.4 é a Euc. Quadrática.

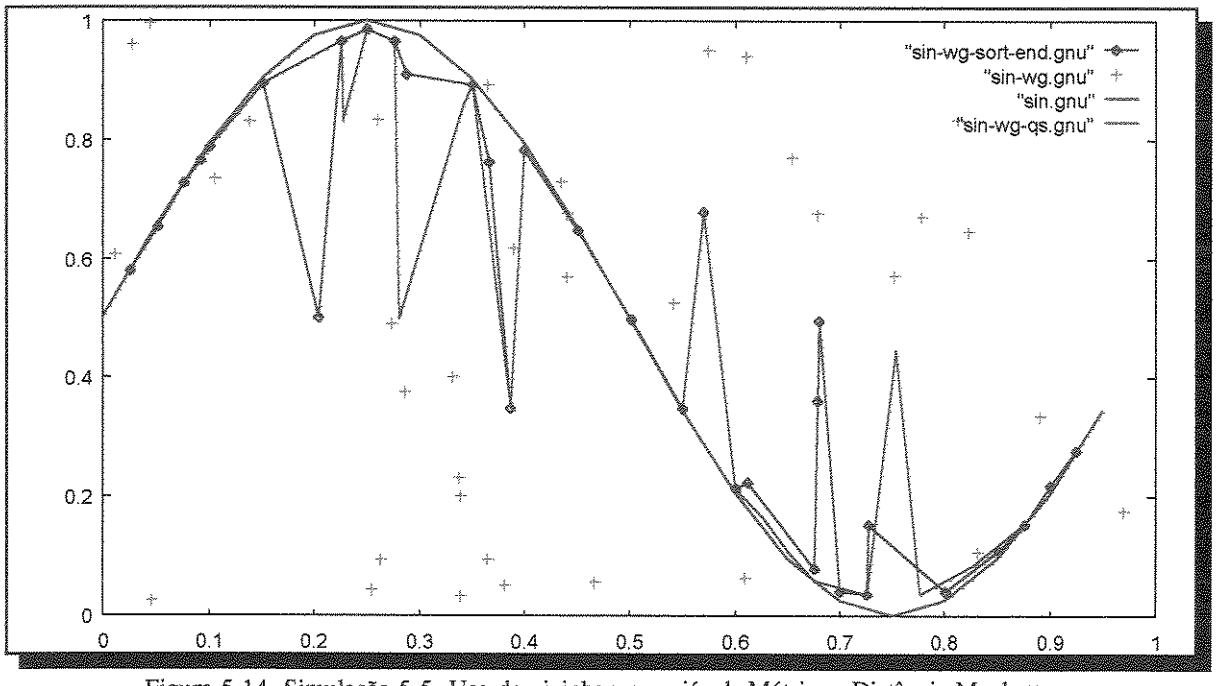


Figura 5.14. Simulação 5.5. Uso de vizinhança variável. Métrica : Distância Manhattan.

Na simulação 5.5. foi variada a métrica para a distância vetorial entre **x** e **w**, utilizando-se a *distância Manhattan* (vide Tabela 5.2).A vizinhança é variável entre 6 (máx.) e 1 (min.).

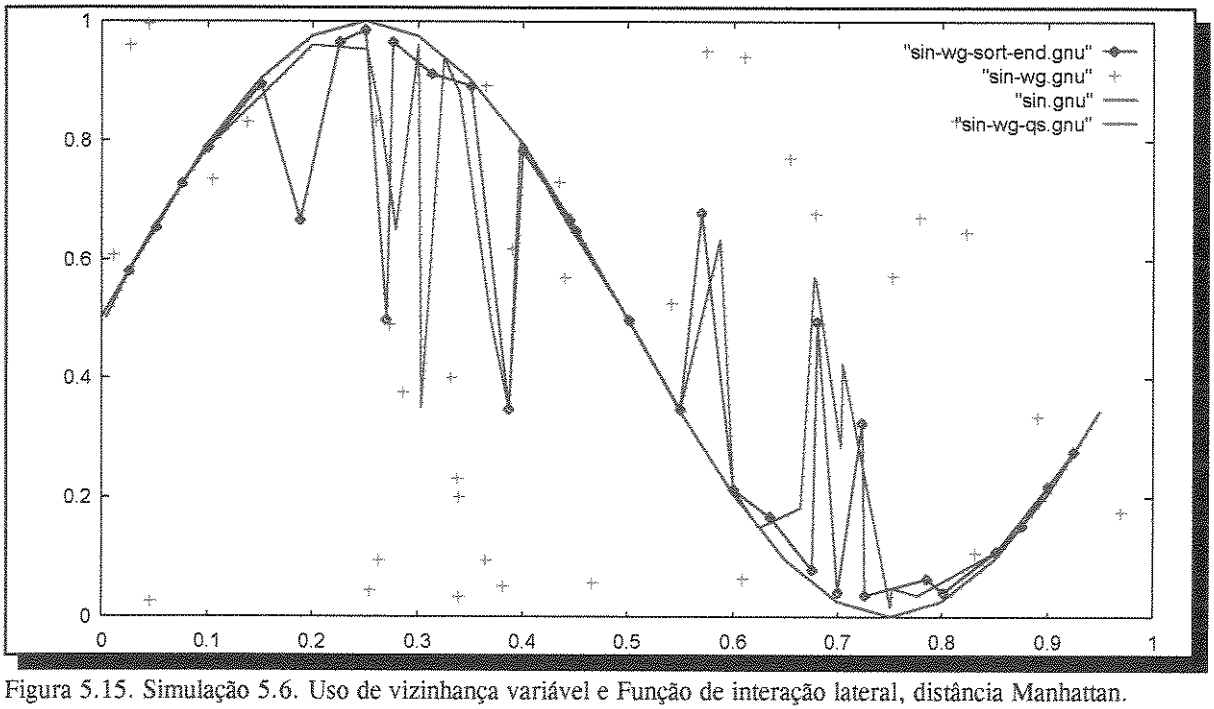


Figura 5.15. Simulação 5.6. Uso de vizinhança variável e Função de interação lateral, distância Manhattan.

Na figura 5.15, são apresentados os resultados da simulação 5.6, acrescentando-se o uso da função de interação lateral, expressa por (5.1).

As simulações 5.1 a 5.6 foram realizadas utilizando a função de ganho $\alpha(t_k)$ ilustrada na figura 5.9, implementada por 4 retas com inclinações diferentes.

Nas simulações apresentadas a seguir (5.7 a 5.10), $\alpha(t_k)$ foi implementada por uma versão discreta da função $\alpha(t_k) = k1\ e^{(-k2tk)}$, ambas as funções são apresentadas na figura 5.16, onde k1= 0.5 e k2 = 0.0015.

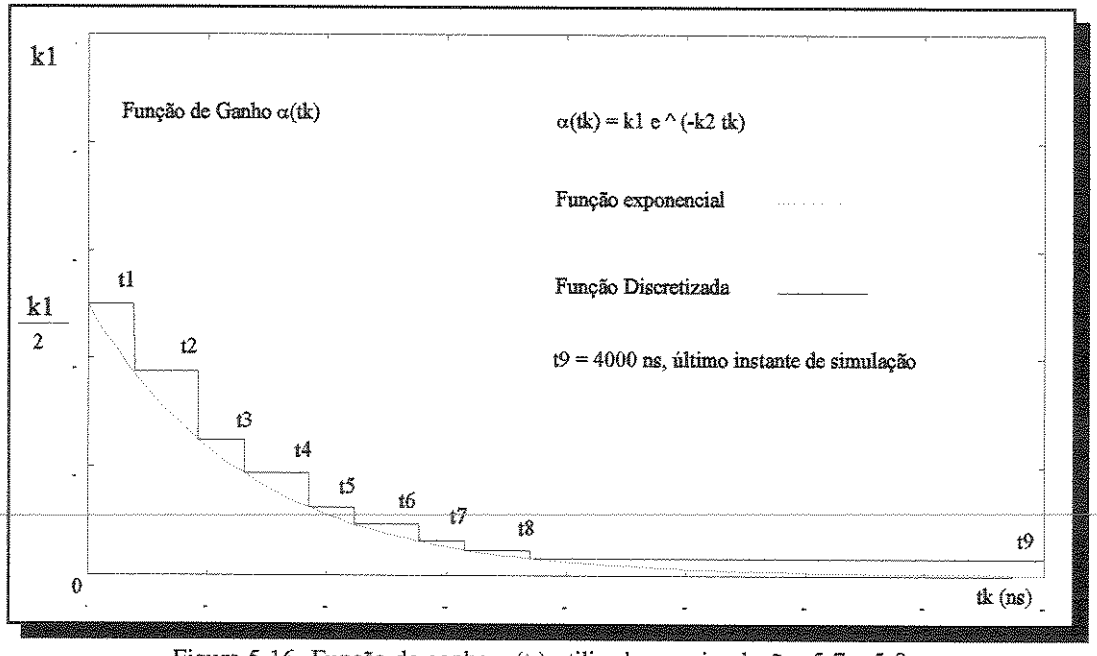


Figura 5.16. Função de ganho $\alpha(t_k)$ utilizada nas simulações 5.7 a 5.9..

Na figura 5.16, apresenta-se a função de ganho discreta utilizada definitivamente na arquitetura de *Neuron*. Devido fundamentalmente à necessidade de maximizar o uso da área de silício disponível e considerando o alto custo de implementação em *hardware* digital (em termos de área) de funções exponenciais e de uma unidade de divisão (para implementar dinamicamente o cálculo de retas com diferentes inclinações), foi decidido o uso de uma função tal como a apresentada na figura anterior. A função de ganho $\alpha(t_k)$ é então implementada em *hardware* por uma combinação de operações de soma e deslocamento (*shift*). Nove registradores são utilizados para armazenar instantes pré-definidos de tempo t_1, ..., t_9, de forma de definir o perfil da função. No momento em que o registrador *step_ct* é igual a um dos valores t_i armazenados, $\alpha(t_k)$ é atualizada.

Os resultados das simulações , incorporando a função mencionada, são descritas a seguir nas figuras 5.17-5.20. A distância *Euclidiana* foi utilizada na simulação 5.7, para medir d(**w**,**x**). Nas simulações 5.8 a 5.10 foi utilizada a distância *Manhattan*, escolhida como solução final, devido a sua prática implementação em *hardware* digital.

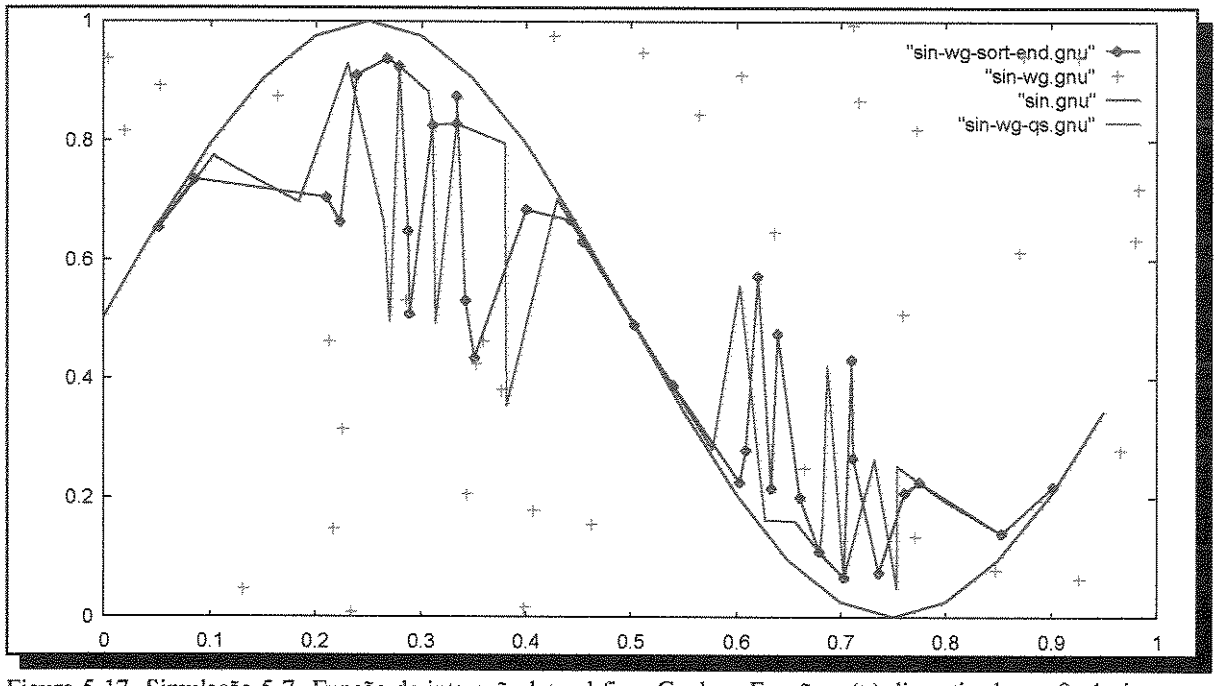


Figura 5.17. Simulação 5.7. Função de interação lateral fixa. Ganho : Função $\alpha(t_k)$ discretizada em 9 níveis.

Na simulação 5.7. a interação lateral é fixa, assim como o raio da vizinhança $N_c(t_k)$. A métrica é a *distância Euclideana*. A simulação difere da 5.2 no uso de distintas funções $\alpha(t_k)$.

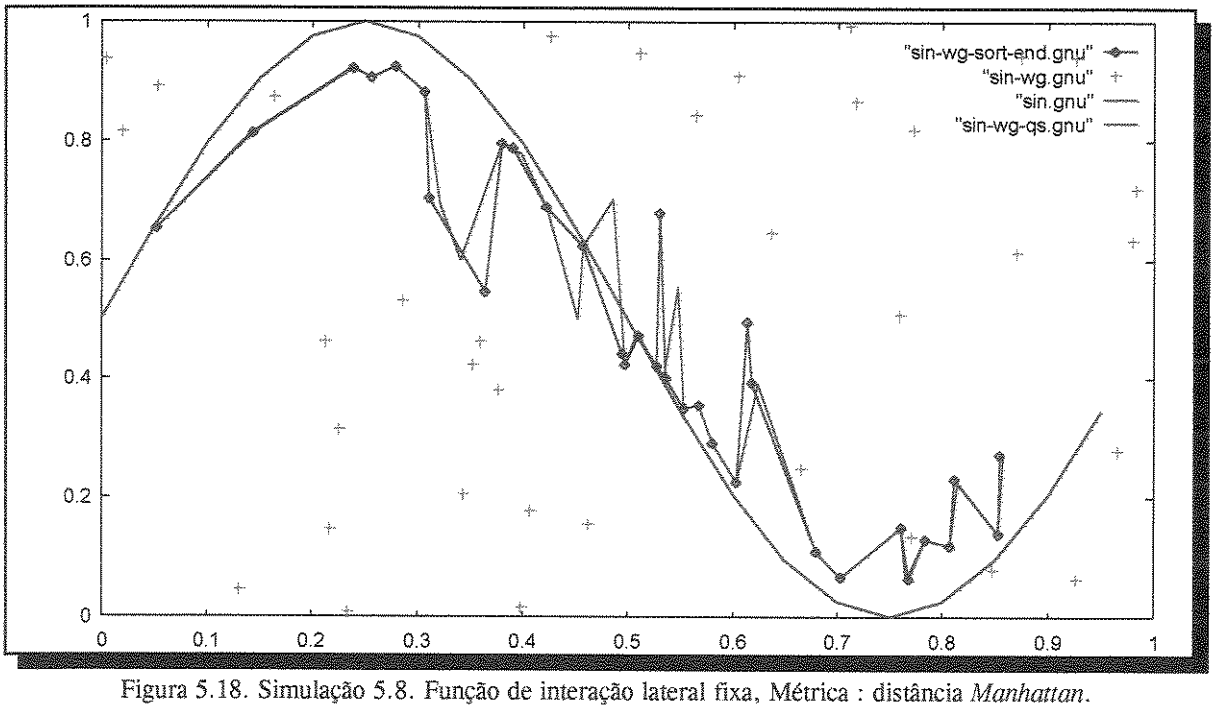


Figura 5.18. Simulação 5.8. Função de interação lateral fixa, Métrica : distância *Manhattan*.

Na simulação 5.8, foi substituído o cálculo da dist. *Euclideana* pela dist. *Manhattan*.

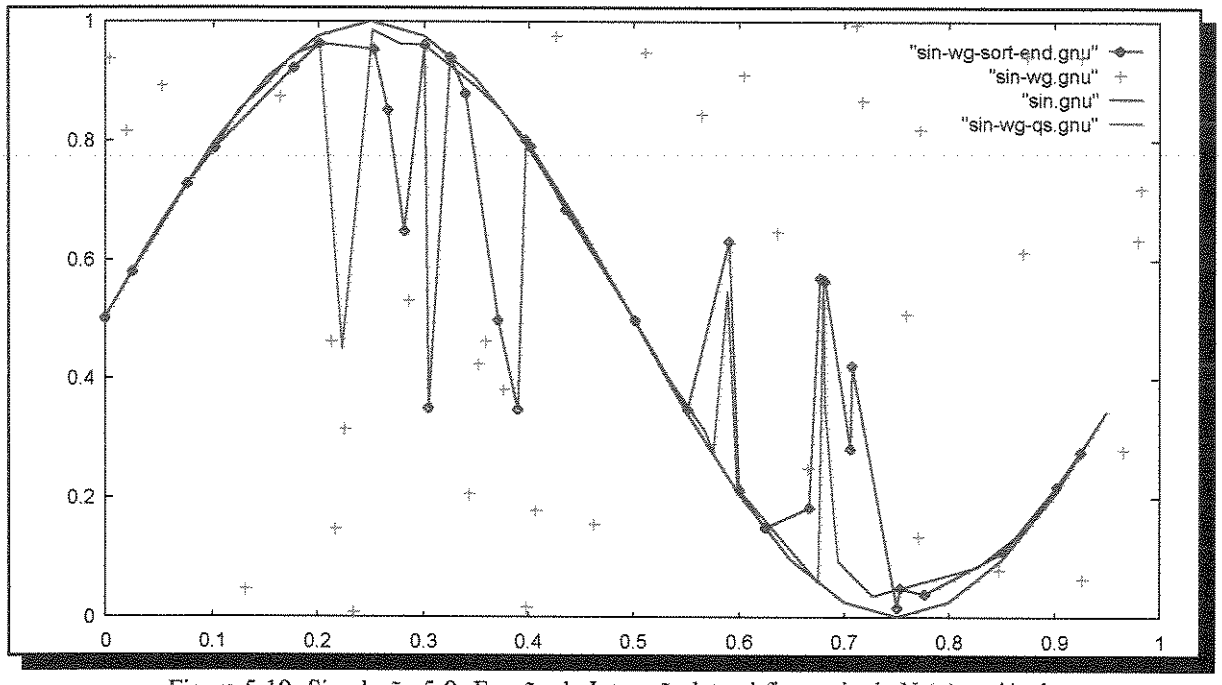


Figura 5.19. Simulação 5.9. Função de Interação lateral fixa, raio de $N_c(t_k)$ variável.

Na simulação 5.9, cujo resultado é ilustrado acima, o raio da vizinhança $N_c(t_k)$ foi implementado como uma função decrescente e discreta (vide figura 5.21.), da mesma forma que para a simulação 5.10, ilustrada na figura 5.20.

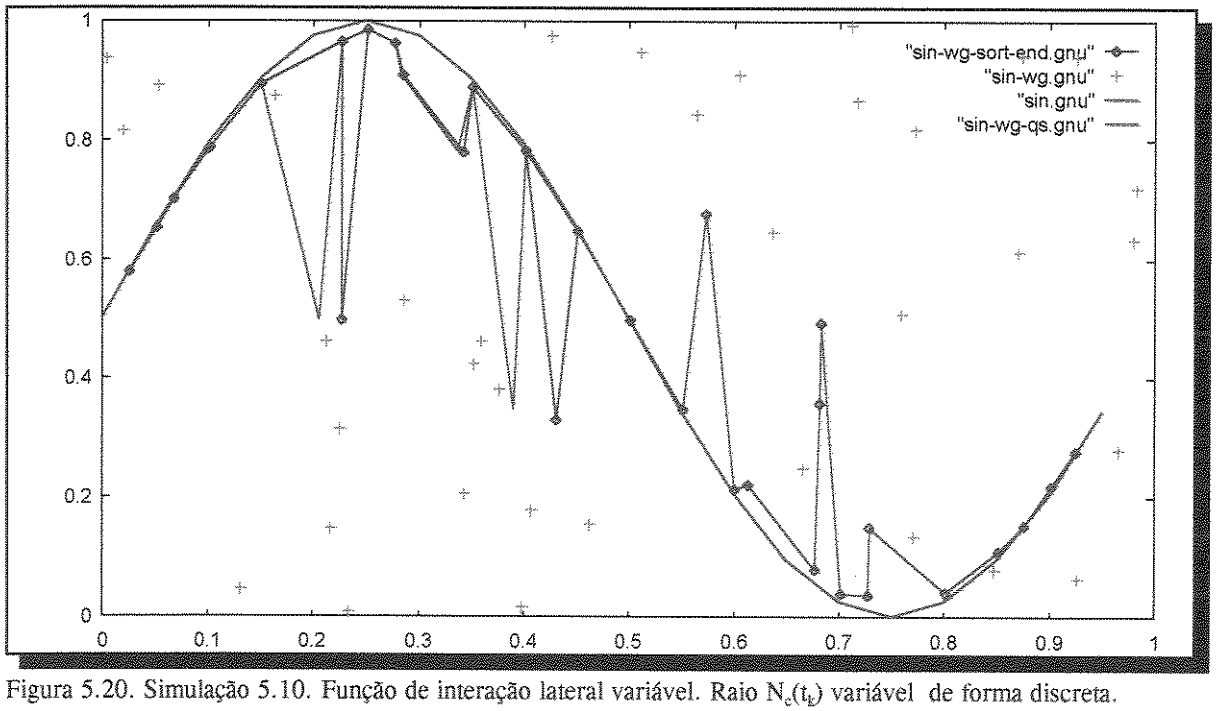


Figura 5.20. Simulação 5.10. Função de interação lateral variável. Raio $N_c(t_k)$ variável de forma discreta.

O raio da vizinhança $N_c(t_k)$ correspondente as simulações 5.9 e 5.10 é representado na figura 5.21. Um registrador de deslocamento (*shift-left*) com um valor pré-definido é inicializado em t_k=0. O seu valor é deslocado à esquerda (i.e., x2) caso a comparação com o registrador do instante corrente de simulação, armazenado em *step_ct*, resultar em igualdade. O processo continua até o instante final da simulação. Cada vez que a comparação entre o valor de *step_ct* seja igual ao do registrador de deslocamento, o raio de $N_c(t_k)$ é diminuído em uma unidade.

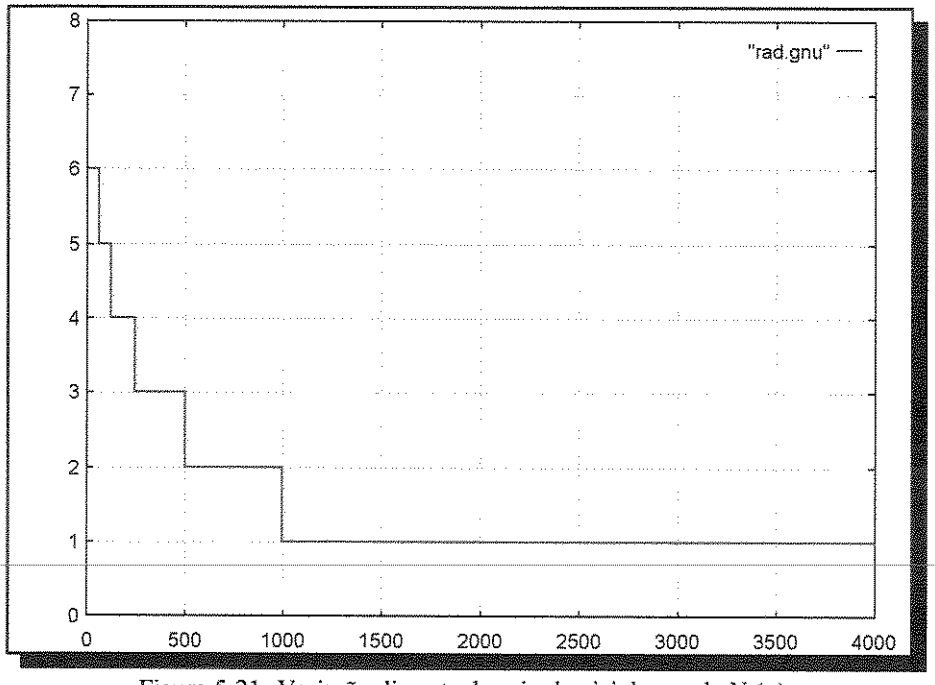


Figura 5.21. Variação discreta do raio de vizinhança de $N_c(t_k)$.

5.2.4.Conclusões Iniciais.

Em função da análise dos resultados das simulações apresentadas anteriormente e de outra série de simulações não apresentadas aqui, foram definidas as características essenciais da arquitetura do neuroprocessador *Neuron*. Os aspectos mais relevantes deduzidos nesta fase são :

- *Função de ganho* $\alpha(t_k)$: É importante ter a opção de implementar uma função linear e outra exponencial, de forma a estudar a sua influência no desempenho do algoritmo. Devido ao custo de implementação em *hardware* digital de funções não lineares, foi decidido substituir a função original por uma discreta, implementada por simples operações de soma e deslocamento esquerda/direita, i.e., multiplicação/divisão por 2.

- *Função de Interação lateral* $\Lambda(t_k,c)$: Não foi verificada completamente a influência do desempenho desta função para o caso de redes de pequeno porte. No entanto, a interação lateral é importante como forma da modelagem de redes inspiradas biologicamente. Outro fator que decide a favor da implementação é a sua influência no desempenho em redes neurais de tamanho médio e grande porte (acima de 100 neurônios).

- *Raio da vizinhança* $N_c(t_k)$: Sem dúvida o desempenho do algoritmo SOFM é melhorado quando é utilizada a vizinhança variável. Há aplicações onde tem se utilizado vizinhanças de raio fixo. No entanto, é essencial que a arquitetura considere esta opção. A influência deste parâmetro pode ser verificada a partir da comparação dos resultados das simulações descritas anteriormente.

- *Métrica para a distância.* Este é um aspecto especialmente crítico, pois foram descobertas pequenas diferenças no desempenho do algoritmo quando a métrica é modificada. Não há possibilidade de definir que uma métrica é sempre mais adequada que outra. As simulações demostram que este parâmetro afeta de forma diferente o algoritmo dependendo das características da rede, como forma da vizinhança topológica, taxa de decaimento do raio dinâmico de $N_c(t_k)$, forma da função de ganho $\alpha(t_k)$, etc. No entanto, quando se pensa em uma implementação em *hardware*, o custo em termos de área de silício é relevante. Foi decidido finalmente utilizar a métrica da distância *Manhattan*.

No tipo de rede até aqui estudada há diversos aspectos que ainda estão em aberto desde o ponto de vista analítico, produto de uma análise formal incompleta do modelo para o caso geral, i.e., para dimensões de vetores pesos **w** e de entradas **x** de dimensão d, ($d \in \mathbb{R}^n$). Não há qualquer estudo conclusivo suficientemente rigoroso que permita definir sob quais condições o algoritmo entra em regime estável ou não, ou sob quais condições a sua convergência é garantida. Estes aspectos levam a pensar em definir uma máquina que permita realizar simulações exaustivas do modelo em alta velocidade, sob diferentes condições de operação. É o que é chamado aqui de uma arquitetura de *alta flexibilidade*, i.e., uma máquina que permita programar externamente os parâmetros do algoritmo e desta forma dirigir a sua evolução.

No capítulo seguinte, será descrita integralmente a arquitetura de *Neuron*, a proposta deste trabalho para a implementação de uma rede altamente flexível. Os componentes de *Neuron* serão simulados separadamente e posteriormente integrados num sistema. Os circuitos gerados por síntese automática serão simulados incorporando a informação da tecnologia objeto, para verificar a funcionalidade do sistema com *timing* preciso.

5.3. Descrição do bloco *WTA*.

5.3.1. Requisitos e Especificações gerais.

O propósito do bloco é a detecção da célula ganhadora no processo competitivo. O bloco recebe como entrada os valores correspondentes às distâncias entre os vetores de entrada e os pesos internos de cada célula. A distância é calculada concorrentemente e enviada diretamente ao bloco (vide figura 5.6). A detecção da célula ganhadora foi implementada por meio de uma árvore de busca binária, onde cada nó da árvore recebe 2 sinais como entrada, determina o valor menor, que é colocado na sua saída e transmitido ao nó do nível inferior da árvore (vide figura 5.22). Cada nó deve saber reconhecer qual canal de entrada lhe enviou o valor menor, para transmitir-lhe um sinal de volta aos nós dos níveis superiores e assim definir a única célula

ganhadora na fase de aprendizado.

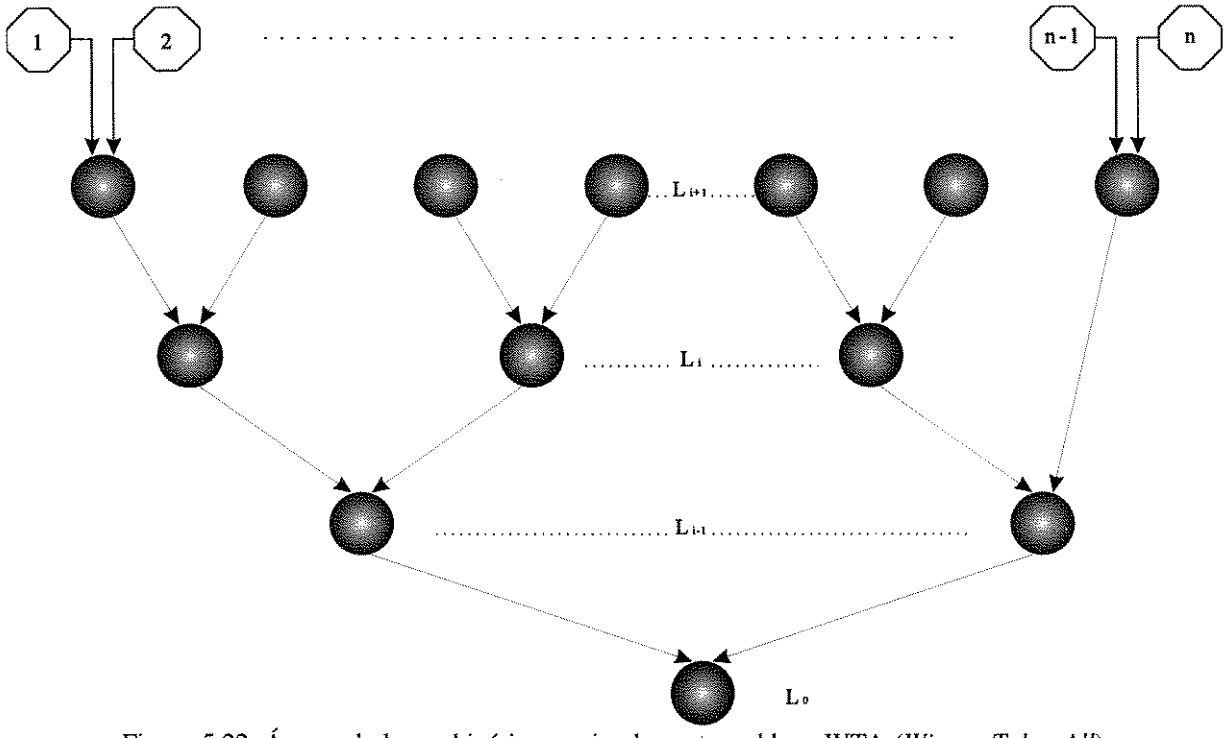


Figura 5.22. Árvore de busca binária para implementar o bloco WTA (*Winner-Takes-All*).

5.3.2. Operação.

Na figura 5.22. representa-se a árvore de busca binária que implementa a operação de detecção da célula ganhadora. O nó do nível L_i determina a distância menor entre as duas entradas recebidas do nível imediatamente superior L_{i+1}.e o transmite ao nó no nível L_{i-1}. Quando o valor chega ao nó do nível inferior L_0, um sinal (não ilustrado na figura 5.22) é enviado de volta ao nó no nível imediatamente superior L_{i+1}, que enviou o menor valor. O sinal é assim transmitido sucessivamente até chegar ao nível topo, onde a célula *i* com a menor distância recebe-o, indicando-lhe que é a ganhadora do processo. A arquitetura interna dos nós será descrita no seguinte capítulo.

5.3.3. Conclusão parcial.

Da experiência obtida até este ponto do projeto, pode-se comprovar a facilidade da linguagem VHDL para modelar e simular sistemas de média a grande complexidade. É possível descrever sistemas contendo diversos níveis de hierarquia, tal como usado em projetos de complexos filtros digitais 2-D [75]. O próximo passo é a modelagem em VHDL,visando a síntese e o mapeamento em uma tecnologia específica.

Capítulo 6

A arquitetura do neuroprocessador *Neuron*

Escopo

Neste capítulo é descrita a arquitetura e a implementação em *hardware* do neuroprocessador *Neuron*. A maioria dos componentes foram gerados por meio de síntese automática, a partir de descrições escritas em VHDL no nível RTL. Alguns blocos foram gerados por métodos tradicionais de captura esquemática e pelo uso de macrofunções disponíveis no CAD utilizado.

Na seqüência, descrever-se-ão individualmente os componentes mais relevantes de *Neuron*. O projeto foi desenvolvido integralmente por meio do CAD *framework* de *Mentor Graphics Co.*, tanto na fase de modelagem e descrição VHDL, quanto na geração dos blocos por captura esquemática. O algoritmo SOFM original foi levemente modificado para adequá-lo às necessidades e às especificações do projeto de *hardware*, alguns dos blocos foram sucintamente apresentados no capítulo anterior. Os blocos mais relevantes da arquitetura foram modelados no nível RTL (*Register Transfer Level*), em linguagem VHDL sintetizável, sendo posteriormente integrados compondo a arquitetura definitiva de *Neuron*. A partir deste nível de modelagem, a maioria dos blocos foi sintetizada pela ferramenta *Autologic*, mapeada e otimizada para uma tecnologia alvo. O processador foi implementado em *standard-cells* na tecnologia AMS CMOS 1.2 *micras*. Após a fase de otimização, os circuitos foram exaustivamente simulados, incorporando a informação de *timing* das bibliotecas da tecnologia objeto.

A descrição das estruturas é realizada por meio de uma abordagem funcional : os blocos são descritos em secções divididas por sua funcionalidade, incluindo-se os diagramas esquemáticos e os seus algoritmos associados. O resultado das simulações da célula será apresentado no capítulo 7.

6.1. Introdução

6.1.1. Considerações e Restrições gerais.

O objetivo principal perseguido nesta fase é o projeto de um elemento de processamento básico que implemente da forma mais eficiente possível o algoritmo SOFM, descrito nos capítulos anteriores. Por eficiente considera-se um sistema onde o ganho de desempenho sobre a implementação em um computador seqüencial convencional seja relevante. As possibilidades de implementação do SOFM são variadas, mas as restrições impostas pela tecnologia disponível são um fator limitante. Neste projeto foram considerados importantes os seguintes aspectos relacionados com o desenvolvimento da estrutura do *hardware*:

(i) Processamento maciçamente paralelo.

Como ponto inicial no desenvolvimento de um protótipo, decidiu-se aproveitar as características de paralelismo maciço que a rede neural possue. Esta primeira restrição envolve uma grande capacidade de processamento concorrente **da rede**, i.e., é necessário o maior número possível de células de processamento operando em paralelo. Esta primeira restrição levou a decidir por uma célula compacta, os circuitos gerados por meio da síntese foram durante todo o processo otimizados por área, em detrimento de circuitos de alta velocidade. Os circuitos de alto desempenho geralmente acarretam a ocupação de maior área de silício, devido ao uso intensivo de *pipelines*, largos barramentos e estruturas de *hardware* do tipo soma-de-produtos altamente paralelas, etc.

(ii) Flexibilidade.

Foi decidido que esta primeira versão do neuroprocessador, deveria permitir emular uma variedade razoável de diferentes curvas de aprendizado. Entende-se como curva de aprendizado a função de ganho $\alpha(t_k)$, que exerce enorme influência no desempenho do algoritmo, como foi inferido das simulações realizadas em diferentes níveis e descritas em capítulos anteriores. Desta forma, a fim de implementar $\alpha(t_k)$, foi escolhida a função discreta apresentada no capítulo anterior, devido ao balanço adequado do compromisso custo/desempenho. A função é então implementada por um módulo composto por 9 registradores de 12 bits, mais um par de registradores e uma pequena estrutura de controle, o seu funcionamento é descrito na secção 6.3.

(iii) Processamento aritmético.

Um outro aspecto muito influente no tamanho do elemento de processamento é relativo ao mecanismo de processamento aritmético. Uma decisão deve ser tomada para escolher entre as principais formas de processamento, a que se acomode melhor ao algoritmo, sem sacrificar o seu desempenho, alterar significativamente as suas propriedades de convergência nem introduzir distorções devido à quantização. A alternativa escolhida foi o uso de processamento em ponto fixo com sinal, usando unidades aritméticas de 12 bits. A forma de implementação e o sistema numérico utilizado é descrito na secção 6.4.

6.1.2. Metodologia de projeto.

No capítulo anterior, já foi descrito de forma geral o método de projeto utilizado. Nesta secção será detalhada um pouco mais tal metodologia. Um diagrama de fluxo é apresentado a seguir, na figura 6.1.

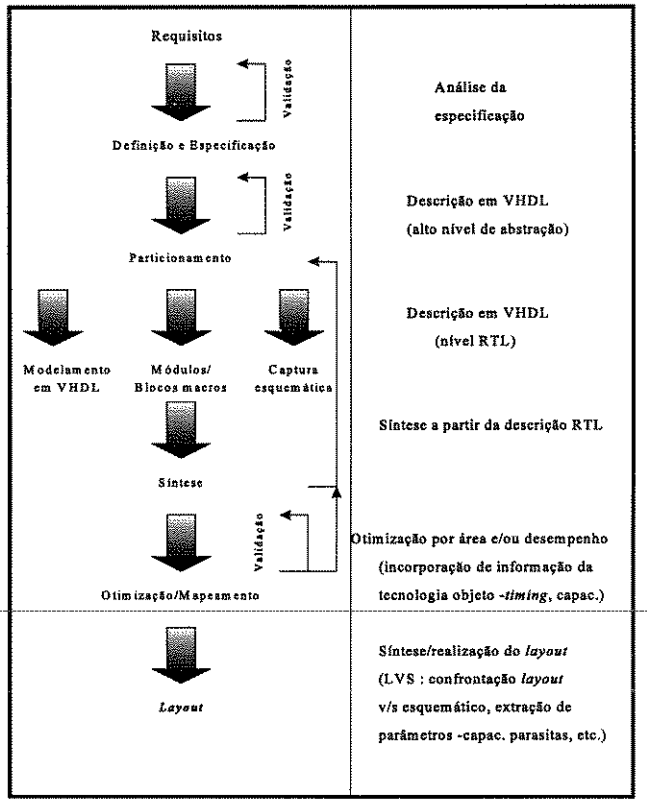


Figura 6.1. Diagrama do fluxo do projeto de *Neuron*.

A figura anterior representa o fluxo seguido ao longo do projeto. A metodologia é conhecida como *top-down* [76], embora na fase da divisão e descrição dos blocos por captura esquemática, foram empregados aspectos da metodologia *bottom-up*. O fluxo é iniciado pela especificação, normalmente feita em linguagem natural. Métodos automatizados podem ser empregados pelo uso de uma especificação conhecida como SDL (*Specification and Description Language*) [77], onde a partir de uma especificação gráfica e a linguagem SDL, podem ser geradas especificações em VHDL sintetizável. O método não é ainda muito empregado. Apesar das vantagens devido à automatização, normalmente há algumas deficiências no desempenho das estruturas geradas. O método manual de descrição permite ajustar com melhor precisão os detalhes referentes à síntese. Os circuitos gerados por síntese são sensivelmente dependentes da forma de codificação em VHDL e métodos de geração automática de código não sempre produzem resultados satisfatórios.

6.2. Estrutura Básica e Interface.

Os blocos descritos a seguir compõem a estrutura principal do neuroprocessador. Será feita uma divisão por blocos funcionais. A estrutura simplificada da arquitetura é ilustrada na figura 6.2.

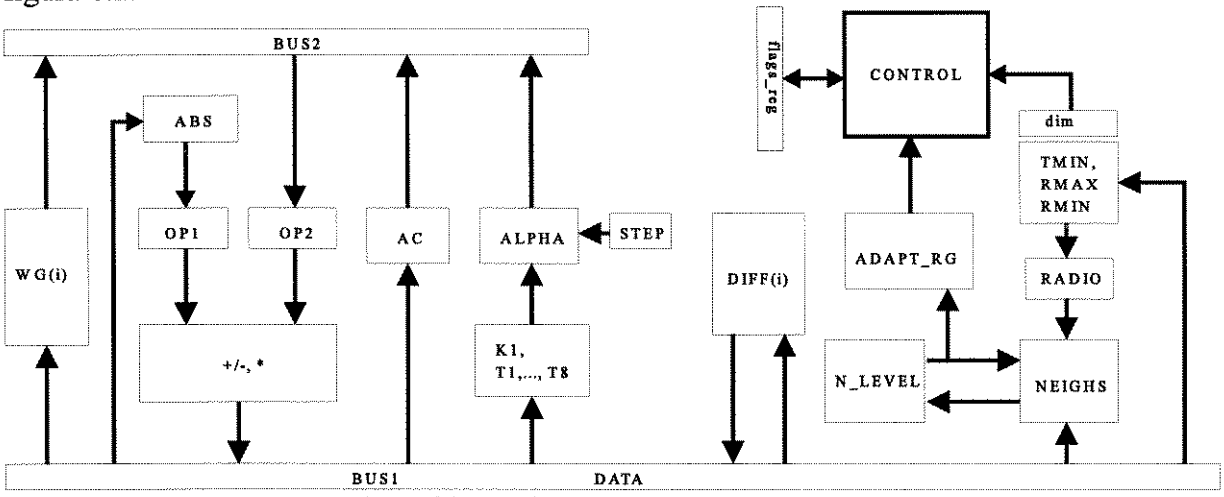


Figura 6.2. Arquitetura básica da célula *Neuron*.

A figura representa a organização básica da arquitetura da célula *Neuron*. Pode-se dividir a arquitetura em módulos funcionais, na forma convencional : unidades aritméticas, registradores e unidade de controle. Dois barramentos principais, BUS1 e BUS2, permitem a comunicação entre os distintos blocos funcionais e com o exterior.

(a) Unidades Aritméticas : Foram implementadas um somador/substrator de 12 bits, com sinal, operando em complemento a 2. A outra unidade é um multiplicador de 12 bits, operando com entradas de 12 bits, a saída foi truncada para 12 bits. O bloco ABS produz o valor absoluto da sua entrada, e envia a saída para o *latch* OP1. Quando o segundo operando já está disponível e estável no *latch* OP2, o controle libera ambos valores à unidade aritmética para continuar o processamento. Ambas unidades foram implementadas a partir de macrofunções de biblioteca.

(b) Registradores : Um conjunto de distintos registradores permitem armazenar os parâmetros que controlam a forma da função de ganho $\alpha(t_k)$ e a função de interação lateral, usando os blocos ALPHA e NEIGHS da figura 6.2, junto aos registradores K1,T1,...,T8, RADIO. A função de todos os registradores será explicada na seguinte secção.

(c) Unidade de Controle : O controle da operação de todos os módulos é realizado por uma **máquina de estados finitos** (FSM-*Finite State Machine*), que opera com palavras de controle de 23 bits, acionada por um relógio comum, externo à célula. A máquina foi modelada em VHDL e posteriormente sintetizada e otimizada à tecnologia alvo. Foi implementada uma versão apartir da linguagem *kiss* [78], para efeitos de comparação com a versão em VHDL.

O acesso aos barramentos BUS1 e BUS2 e controlado pela FSM por meio de sinais enviados a um conjunto de dispositivos *tri-state*, não ilustrados na figura 6.2.

(d) Interface : A comunicação de *Neuron* com as outras células, a máquina hospedeira (*host*) e o bloco WTA (*Winner-Takes-All*) é ilustrada na figura 6.3. São utilizados um total de 97 sinais para a comunicação de *Neuron* com o exterior.

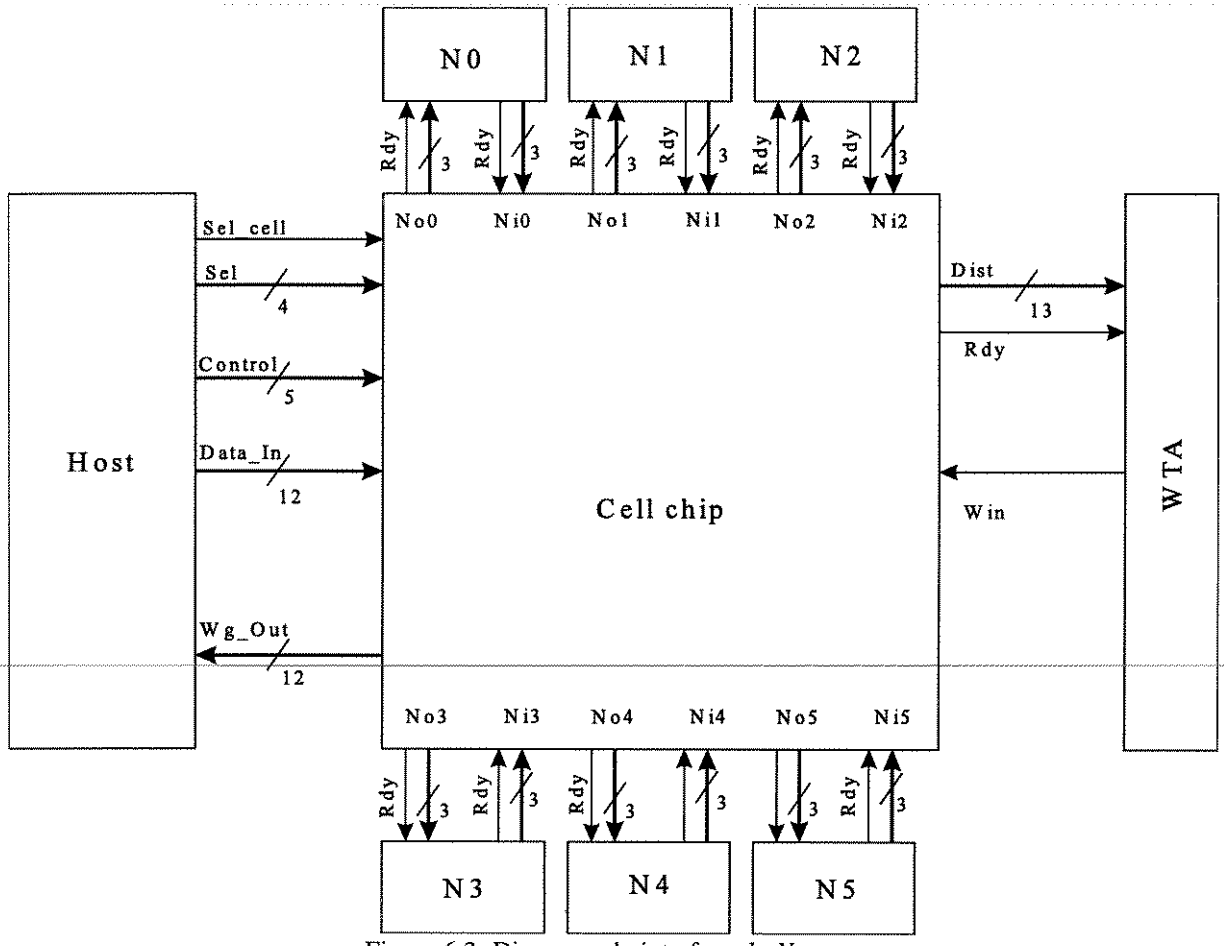


Figura 6.3. Diagrama da interface de *Neuron*.

O barramento BUS1 (também chamado de *Data In*) é responsável pela comunicação com o *host*. Na fase de aprendizado, os vetores de treinamento são ingressados por BUS1 . A saída do barramento BUS2 (ou *Wg Out*) permite ler o conteúdo dos registradores que armazenam os pesos Wg_i durante o processo de atualização. Este mecanismo é necessário para o monitoramento constante da evolução dos vetores pesos durante a fase de aprendizado. Nesta implementação o protótipo permite operar com vetores de até 4 componentes, o que foi considerado suficiente para propósitos de teste e implementação em *hardware* do algoritmo.

6.3. Blocos funcionais principais.

Neste secção, a arquitetura de *Neuron* será descrita detalhadamente, de acordo com a sua estrutura funcional.

6.3.1. Máquina de Estados (FSM).

Cada uma das operações da célula é controlada por um circuito seqüencial que implementa uma *máquina de estados de Moore* [79], onde a saída da máquina é função unicamente do estado corrente (independente da entrada) e as transições entre estados são ativas pela borda de subida (*rising-edge*) de um *clock* externo, distribuído a todas as células da rede.

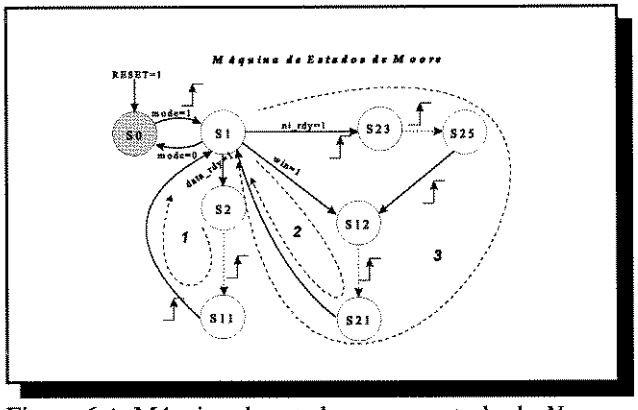


Figura 6.4. Máquina de estados para controle de *Neuron*

O diagrama da figura 6.4 representa de forma simplificada a máquina de Moore utilizada para implementar a operação da célula *Neuron*. Na figura, mostram-se só os estados mais relevantes às transições para cada um dos 3 ciclos principais (1)-(3), descritos na tabela 6.1.

Ciclo	Operação	Estados
(0)	Inicialização - programação de parâmetros	S_0
(1)	Cálculo da *distância Manhattan* entre **x** e **w**	S_1-S_{11}
(2)	Atualização do vetor de pesos **w**, caso da célula ganhadora	S_{12}-S_{21}
(3)	Atualização do vetor de pesos **w**, caso da célula pertencente à vizinhança $N_c(t_k)$	S_{23}-S_{26} + S_{12}-S_{21}

Tabela 6.1. Distribuição simplificada de operações na máquina de estados de Moore.

A Máquina de Moore foi implementada por meio de uma descrição VHDL, logo sintetizada e otimizada para a tecnologia alvo. Foram utilizados um total de 27 estados para o funcionamento completo da célula. Em cada estado é gerada uma saída composta por uma palavra de 23 bits que controla os blocos funcionais necessários às operações de cada um dos ciclos descritos na tabela 6.1. A estrutura de controle permite o acesso aos barramentos BUS1, BUS2 (Data In eWg Out na fig. 6.3) , carga (*store*) e habilitação de saída dos registradores para pesos entradas, parâmetros do algoritmo, habilitação da saída da unidade aritmética, habilitação da unidade de cálculo de complemento a 2, etc.

Funcionamento:

Ciclo (0)

Um pulso assíncrono na entrada *Reset* (*Control*(0), na figura 6.3) coloca a célula no estado S_0 (fig. 6.4), onde ela pode ser programada armazenando os parâmetros do algoritmo SOFM. No estado S_0 é possível escrever os dados nos registradores internos correspondentes à lista de parâmetros descritos na tabela 6.2.

Parâm.	T1	T2	T3	T4	T5	T6	T7	T8
Addr.	0	1	2	3	4	5	6	7
Parâm.	K1	Tmin	Rmax	Dim	Wg0	Wg1	Wg2	Wg3
Addr.	8	9	A	B	C	D	E	F

Tabela 6.2. Parâmetros programados externamente no estado inicial S_0.

Os parâmetros descritos na tabela 6.2 permitem definir as características da rede e a forma de operação da curva de ganho para cada célula individualmente. Os valores T1,...,T8 definem a forma de $\alpha(t_k)$, junto com K1; Tmin define a forma como $N_c(t_k)$ varia dinamicamente durante a operação do algoritmo, diminuindo a partir do seu raio máximo Rmax; Dim define o número de componentes usado para os vetores peso **w** e as entradas **x**, até um máximo de 4. Na tabela 6.2, *Addr.* corresponde ao endereço correspondente ao registrador de 12 bits da célula.

Ciclo(1)

Após a fase de programação, um pulso na entrada *Mode* (*Control*(4)), coloca a célula no estado S_1, onde pode-se passar aos estados S_2, S_{12}, S_{23} da FSM, ou voltar a S_0. Ao receber um sinal na entrada *Data RDY* (*Control*(3)) inicia-se o processo de cálculo de distância entre **x** e **w**, para **todas** as células. No final deste ciclo, cada célula envia um sinal pela saída RDY, indicando ao bloco WTA que pode iniciar o processo competitivo e determinar a ganhadora corrente. A operação da rede pode ser melhor compreendida por meio do diagrama da figura 6.5.

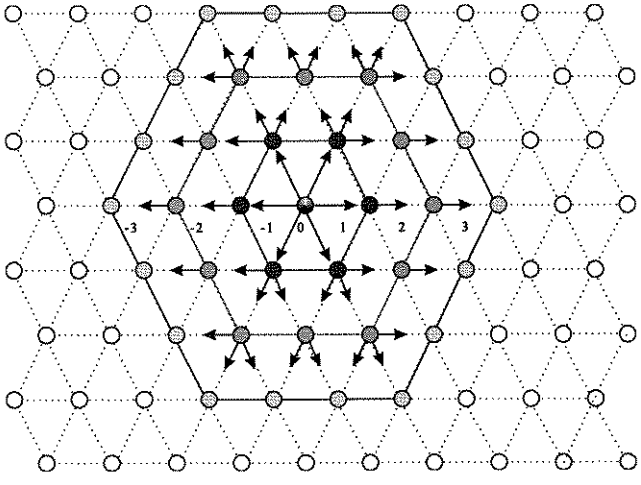


Figura 6.5. Região de atividade numa rede de 10x8 células, para um raio de vizinhança corrente $N_c(t_k)$=3.

Ciclos(2) e (3).

Na figura 6.5, uma rede com vizinhança topológica hexagonal está em operação no instante em que o raio dinâmico de $N_c(t_k)$=3. A célula ganhadora do processo competitivo para o instante t_k, é indicada com o *label* "0". Após receber um sinal do bloco WTA (vide cap.5) pela entrada *Win* (vide fig. 6.3), ela passa para o estado S_{12} (figura 6.4), onde inicia-se o processo de atualização de pesos controlado no ciclo(2) pela FSM. As células imediatamente vizinhas à ganhadora (raio=1) recebem dela um sinal de ativação, colocando-as no estado S_{23}. O sinal ingressa por alguma das entradas Ni RDY, junto com o valor da distância topológica entre elas (1, para o primeiro nível e assim sucessivamente). Desta forma, as células vizinhas localizadas no raio=1 passam para o estado S_{23}, iniciando-se o processo de atualização de forma concorrente, através do ciclo(3) controlado pela FSM.

O processo repete-se sucessivamente para as células localizadas no raio 2, que recebem os sinais de ativação desde o nível 1, até àquelas localizadas no limite do raio corrente de $N_c(t_k)$. Cada célula envia sinais de ativação para as 6 vizinhas topológicas no estado S_{13} da FSM, sempre e quando esteja localizada dentro dos limites da vizinhança topológica $N_c(t_k)$.

A versão completa utilizada para a máquina de estados é representada no diagrama da figura 6.6 e a descrição detalhada das saídas respectivas descreve-se nas tabela 6.3 e 6.4.

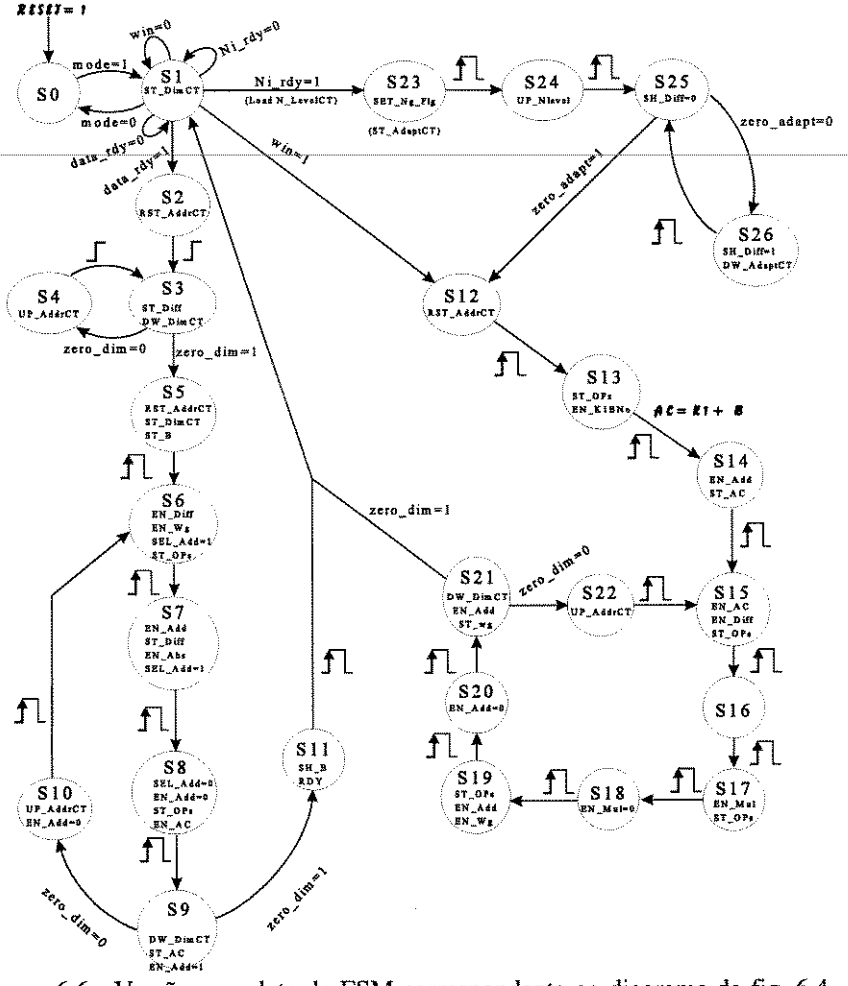


Figura 6.6.- Versão completa da FSM correspondente ao diagrama da fig. 6.4.

O diagrama da figura 6.6 representa a máquina de Moore completa utilizada para o controle de *Neuron*. Cada arco do grafo representa uma transição disparada pela borda de subida do *clock* geral ou por um determinado valor presente na entrada da máquina. Os nós representam o estado corrente S(t) e a ação realizada em cada transição. A saída pode ser representada por:

$$\eta(t) = \lambda\{S(t)\}$$

onde λ representa a função de saída, dependente unicamente do estado corrente S(t). As entradas à máquina foram definidas por um vetor ξ, cujos componentes são :

$\xi(0)$ = Zero Adapt (sinal interno a *Neuron*)
$\xi(1)$ = Zero Dim (sinal interno a *Neuron*)
$\xi(2)$ = Ni RDY (sinal externo de célula vizinha)
$\xi(3)$ = Win (sinal externo do bloco WTA)
$\xi(4)$ = Data RDY (sinal de controle externo - *Control*(3))
$\xi(5)$ = Mode (sinal de controle externo - *Control*(4))

O modelo utilizado pode ser representado pelo diagrama de blocos descrito na figura 6.7.

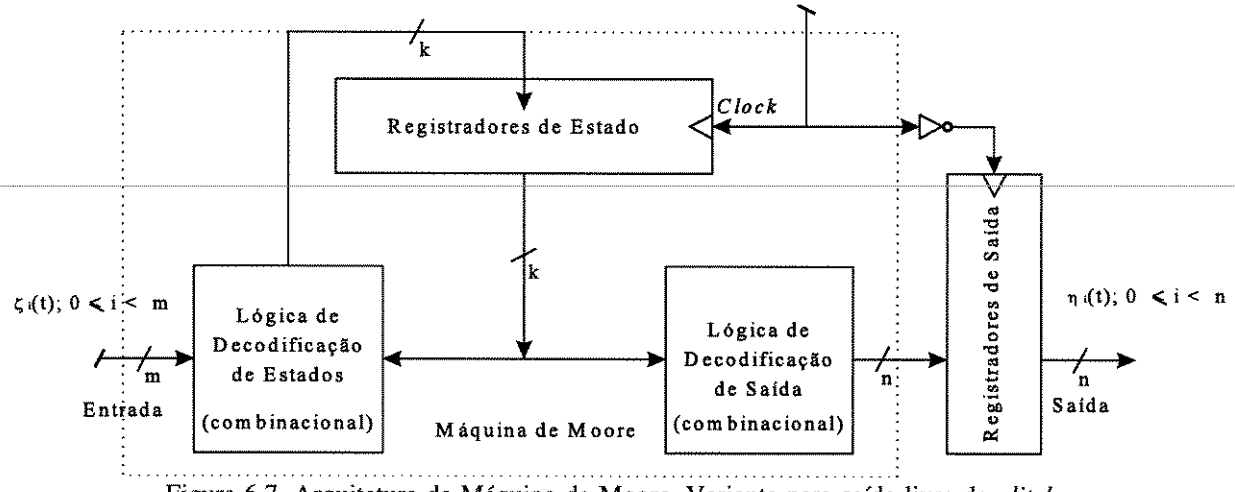


Figura 6.7. Arquitetura da Máquina de Moore. Variante para saída livre de *glitchs*.

A arquitetura sintetizada da figura 6.7 refere-se a uma variante da máquina de Moore original, cuja saída é sincronizada com a borda de subida do *clock*. A variante utilizada é livre de *glitchs*, potencialmente ocasionados na saída do circuito combinacional de decodificação. Esta versão contém 2 conjuntos de registradores: um permite armazenar o estado corrente S(t), o outro armazena a saída $\eta(t)$, na borda oposta do *clock* que é utilizada para armazenar o estado S(t). Portanto, a máquina deve ser otimizada com a restrição de que o atraso intrínseco à lógica de saída deve ser no máximo a largura do pulso do *clock* utilizado, menos o tempo de *set-up* dos *latchs* de saída. Na FSM otimizada para o controle de *Neuron*, m=6, n=23 e k= 5, os 27 estados foram codificados em 5 bits, utilizando código Gray [76], [79].

A combinação de entradas/saídas da máquina para os estados correspondente ao ciclo(1) é apresentada na tabela 6.3. As entradas são representadas em binário, as saídas em Hexadecimal.

Entrada $\xi_i(t)$; i $\in$[5,...,0]	Transição de Estados	Saída $\eta(t)$ (HEX- 23 Bits)	Estado S(t)
1-----	S_0-S_1	000010	S_1
11----	S_1-S_2	400000	S_2
1-----	S_2-S_3	020008	S_3
1---0-	S_3-S_4	200000	S_4
1---1-	S_3-S_5	408010	S_5
1-----	S_4-S_3	020008	S_3
1-----	S_5-S_6	140A00	S_6
1-----	S_6-S_7	020680	S_7
1-----	S_7-S_8	0008C0	S_8
1-----	S_8-S_9	000428	S_9
1---0-	S_9-S_{10}	200000	S_{10}
1-----	S_{10}-S_6	140A00	S_6
1---1-	S_9-S_{11}	004001	S_{11}
1-----	S_{11}-S_1	000010	S_1

Tabela 6.3. - Combinação de Entradas/Saídas para o ciclo(1) da FSM.

Para o caso do ciclo(2), correspondente à atualização de pesos, a combinação de entradas/saídas é apresentada na tabela 6.4. Os valores representam-se em binário e haxedecimal.

Entrada $\xi_i(t)$; i $\in$[5,...,0]	Transição de Estados	Saída $\eta(t)$ (HEX- 23 Bits)	Estado S(t)
1-1---	S_1-S_{12}	400000	S_{12}
1-----	S_{12}-S_{13}	002800	S_{13}
1-----	S_{13}-S_{14}	000420	S_{14}
1-----	S_{14}-S_{15}	040840	S_{15}
1-----	S_{15}-S_{16}	000000	S_{16}
1-----	S_{16}-S_{17}	000900	S_{17}

1-----	S_{17}-S_{18}	000000	S_{18}
1-----	S_{18}-S_{19}	100C00	S_{19}
1-----	S_{19}-S_{20}	000000	S_{20}
1-----	S_{20}-S_{21}	080408	S_{21}
1---0-	S_{21}-S_{22}	200000	S_{22}
1---1-	S_{21}-S_1	000010	S_1

Tabela 6.4. - Combinação de Entradas/Saídas para o ciclo(2) da FSM.

O 3o. ciclo, correspondente à atualização de pesos para as células não ganhadoras pertencentes a $N_c(t_k)$, representa-se parcialmente, na tabela 6.5.

Entrada $\xi_i(t)$; $i \in [5,...,0]$	Transição de Estados	Saída $\eta(t)$ (HEX- 23 Bits)	Estado S(t)
1--1--	S_1-S_{23}	000002	S_{23}
1-----	S_{23}-S_{24}	000004	S_{24}
1-----	S_{24}-S_{25}	000000	S_{25}
1----0	S_{25}-S_{26}	011000	S_{26}
1----1	S_{25}-S_{12}	400000	S_{12}

Tabela 6.5. - Combinação de Entradas/Saídas para o ciclo(3) da FSM.

O ciclo(3), completa-se com o ciclo(2) descrito na tabela 6.4 e no diagrama de estados da fig. 6.6, correspondendo aos estados S_{12}-S_{22}.

Nas tabelas descritas anteriormente,a entrada $\xi(t)$ é representada em binário, onde 1 corresponde ao nível lógico "1" (*high*), 0 ao nível lógico "0"(*low*) e "-" representa "*don't care*". A saída da máquina de estados, $\eta(t)$, corresponde a uma palavra de 23 bits, representada em valor hexadecimal. Em cada um dos estados, a saída $\eta(t)$ controla diretamente o funcionamento dos blocos da arquitetura de *Neuron*, a sua operação será detalhada nas próximas secções.

Para a implementação do algoritmo SOFM, é imprescindível contar com blocos independentes que gerem adequadamante a função $\alpha(t_k)$ e definam dinamicamente a vizinhança $N_c(t_k)$, cujas estruturas serão descritas a seguir.

Critérios que permitiram definir a estrutura de *Neuron*.

Aspectos que influenciam particularmente no custo em silício do projeto do circuito são:

1. Implementação da métrica e da estrutura para cálculo de distância
2. Estrutura para implementar a função de ganho $\alpha(t_k)$.
3. Dimensionamento ótimo dos vetores de peso e de treinamento.
4. Precisão aritmética : largura das palavras, processamento ponto fixo/flutuante.

Muitas das decisões a respeito da arquitetura foram tomadas em função das informações obtidas de simulações, parte já descrita no capítulo anterior (vide sim. 5.7-5.10 e figura 5.16). Desta forma, foi decidido implementar uma função de ganho $\alpha(t_k)$ que pudesse emular tanto uma função linear quanto uma não-linear. A função não-linear pode ser razoavelmente emulada por uma função discretizada em diversos níveis.

A questão do tamanho adequado dos vetores de pesos e de treinamento está estritamente relacionada com o tipo de problemas que se pretende resolver com uma arquitetura particular. Por se tratar de um circuito protótipo, considerou-se que a implementação de vetores de até 4 componentes (4-dimensional) permitiria estudar um número suficiente de aplicações do algoritmo. Para o caso de resolução de problemas específicos, tal como o reconhecimento de sinais acústicos tratado em 4.6.2, a arquitetura deve ser modificada para implementar vetores com maior número de componentes num circuito dedicado.

6.3.2. A Estrutura do bloco gerador da função $\alpha(t_k)$.

A estrutura deste bloco é composta por 8 registradores principais, sendo que sua arquitetura permite gerar uma função discretizada em 9 níveis (vide figura 5.16), considerado como solução razoável para uma implementação digital de baixo custo em silício. A curva permite emular de forma aproximada uma função não-linear do tipo exponencial decrescente. O mecanismo composto por um banco de registradores e um *flip-flop JK* controla a sua operação (vide figuras 6.8 e 6.9).

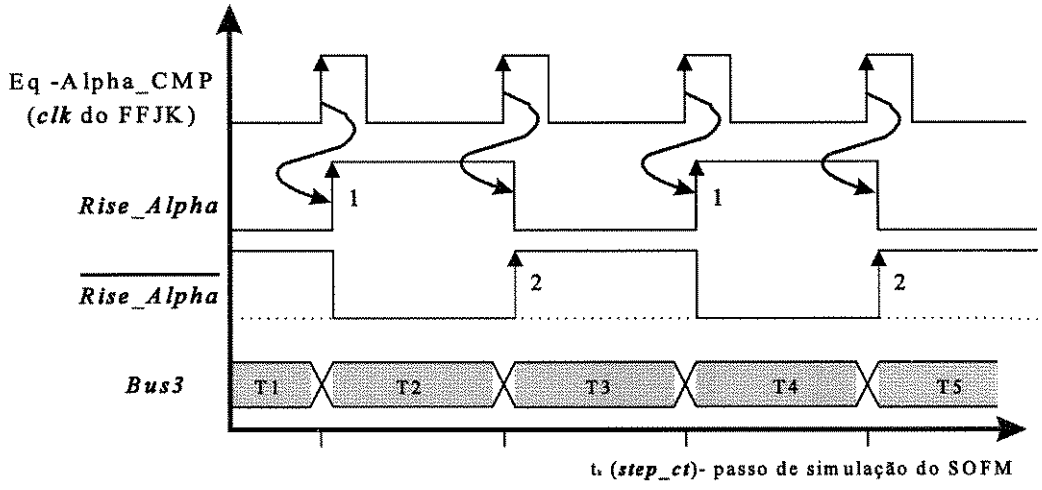


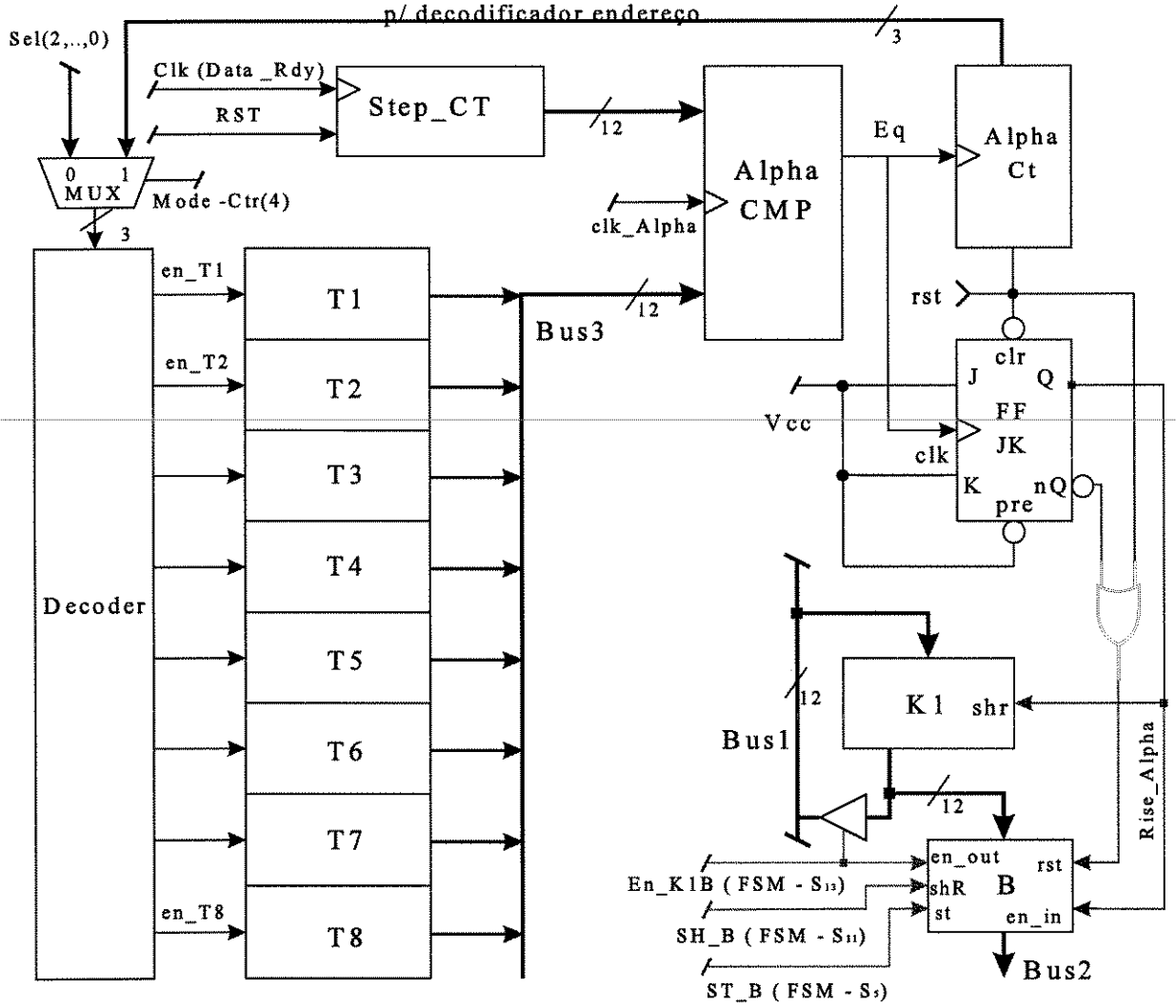
Figura 6.8. *Timing* de operação da geração da função $\alpha(t_k)$, para $0 \leq t_k \leq T5$.

Um registrador chamado de *step ct* armazena o passo corrente de simulação t_k, que é incrementado a cada chegada de um novo conjunto de vetores **x** (sinal *Data Rdy*). Quando o valor de *step ct* é igual a T_i ($1 \leq i \leq 8$), um sinal *Eq*, na saída de um comparador *Alpha CMP*, ocasiona a mudança de estado de um *flip-flop JK*, cujas entradas *J* e *K* estão em "1" lógico. Na fig. 6.8, as saídas do *flip-flop JK* (Q= *Rise Alpha)* controlam 2 registradores K1 e B (fig. 6.9). Na transição "0"→"1" de *Rise Alpha* (*label* "1"da fig. 6.8), é realizada uma operação de *ShiftR* no registrador *K*, ao mesmo tempo habilita-se a entrada *en in* de *B*, permitindo a sua carga. Na

transição "1"→"0" de *Rise Alpha* (*label* "2" em *NOT Rise Alpha* da figura), B é *zerado*.

Na figura 6.9, um registrador T_i é habilitado por um decodificador, por meio do sinal *en* T_i, colocando no barramento BUS3 o valor de T_i corrente utilizado pelo comparador Alpha CMP, a comparação é realizada na transição "0"→"1" do sinal *clk Alpha*.

Inicialmente avalia-se a função $\alpha(t_k)$ entre o instante 0 e T_1. No instante em que o valor corrente de t_k é igual ao valor de T_1, $\alpha(t_k)$ é atualizada e o decodificador agora aponta para o registrador T_2. O processo repete-se sucesivamente para cada valor de T_i (i.e., durante 8 vezes), até o fim da operação do SOFM. Desta forma são gerados 9 níveis discretos que formam o perfil da função de ganho $\alpha(t_k)$ (vide cap.5, fig. 5.16).



$$\alpha(t_k) = K1\ (Bus1) + B\ (Bus2)\ ,\ (\text{transiçao } S_{13} > S_{14})$$

Figura 6.9. Estrutura do bloco funcional para geração de $\alpha(t_k)$.

Computação de $\alpha(t_k)$:

O algoritmo de operação é descrito em forma resumida na tabela 6.6 e cada passo é

explicado mais detalhadamente na seqüência.

A operação é completada aproveitando distintos estados da FSM descrita em 6.3.1. A tabela representa o estado onde é realizada a operação corrente e o mecanismo de controle associado, executado pela saída $\eta(i)$ da FSM ($0 \le i \le 23$) ou pelas saídas do *flip-flop JK*.

Estado	Operação	Controle
S_0	K1 ← cte.; B←0	(*st*+*en in*) K1; *rst B* (RESET ext.)
S_1	IF(*rise Alpha* = "1") ⇒ K1 ←ShiftR	*Rise Alpha* = "1": Q/FFJK
S_5	IF(*en in* B = "1") ⇒ B← K1	St B(FSM); en in B, Q/FFJK
S_{11}	B ← ShiftR (÷ 2)	sh B (FSM)
S_{13}-S_{14}	AC ← K1 + B	En K1BNo, St AC (FSM)

Tabela 6.6. Operação de cálculo de $\alpha(t_k)$, resultado armazenado em AC.

Passo1 - S_0. Neste instante o registrador *K1* é inicializado em um valor constante determinado pelo usuário e o registrador *B* é inicializado em 0 (zero). O sinal externo de *reset* coloca o *flip-flop JK* no estado Q=0, ambas entradas J e K são colocadas em "1" lógico.

Passo2 - S_1. Caso exista uma transição "0"→"1" no sinal *Rise Alpha*, é realizada uma operação de deslocamento à direita no registrador K1 (÷ 2).

Passo3 -S_5. A FSM envia um sinal de *store* ao registrador B, $\eta(15)$-*St B*. O valor presente na entrada vindo diretamente da saída de K1 é armazenado caso o sinal *en in* de B esteja em "1" lógico (saída Q-*Rise Alpha* do FFJK). Caso contrário a operação de *store* é ignorada.

Passo4 -S_{11}. A FSM envía um sinal de deslocamento à direita para o registrador B, $\eta(14)$-*Sh B*. A operação é realizada unicamente quando a sua entrada *en in* é habilitada (*Rise Alpha*="1").

Passo5- S_{13}-S_{14}. Os valores armazenados nos registradores K1 e B são transmitidos à unidade aritmética, habilitados no estado S_{13} pela saída $\eta(13)$ da FSM, *En K1BNo*. Realiza-se de imediato a soma de ambos. O resultado é armazenado num registrador de 12 bits chamado de AC, no estado S_{14}, a través da saída $\eta(5)$, *St AC*, da FSM.

Passo6. - No estado S_1 da FSM, a célula reinicia o processo repetindo o *Passo2*.

O algoritmo descrito acima realiza sempre a operação AC ← K1 + B, armazenando-se o valor estável da soma no estado S_{14} da FSM descrita em 6.3.1, onde B=0 ou B = K1÷2, dependendo do estado do *flip-flop* JK (vide fig.6.8). O registrador K1 é dividido por 2 sempre que haja uma transição de "0"→"1" no sinal *Rise Alpha* (Q/FFJK), i.e.:

AC ← K1 + 0 ⇐ Q/FFJK = "0"
AC ← K/2 + B/2 = K/2 + K/4 ⇐ Q/FFJK = "1"

6.3.3. Determinação da Vizinhança $N_c(t_k)$.

6.3.3.1. Raio dinâmico de $N_c(t_k)$.

A vizinhança dinâmica é definida pelo raio variável durante a evolução do SOFM. O raio máximo inicial é definido como o parâmetro *Rmax* e programado pelo usuário no estado S_0 da FSM (vide secção 6.3.1, tabela 6.2). A operação é iniciada com o raio de $N_c(t_k)$ definido no valor máximo, diminuindo monotonicamente até chegar no mínimo de 1, definindo uma vizinhança final de 6 células ao redor da ganhadora. O perfil da curva pode ser visto na figura 5.21, correspondente às simulações 5.9 e 5.10, descritas no capítulo 5.

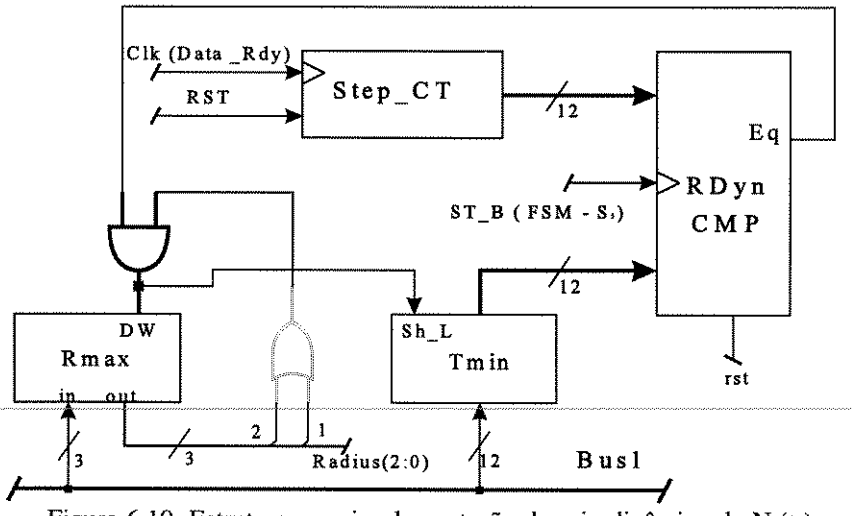


Figura 6.10. Estrutura para implementação do raio dinâmico de $N_c(t_k)$.

O esquemático da figura 6.10 representa a estrutura básica que permite implementar em cada célula o raio variável de $N_c(t_k)$. O registrador *Tmin* é carregado com um valor determinado pelo usuário, junto com o raio máximo. Foi considerada uma vizinhança com um raio máximo de 8 para este protótipo, portanto, o raio corrente da vizinhança Rmax é representado em 3 bits. Quando t_k (o passo corrente do SOFM) é igual a Tmin, o comparador *Rdyn CMP* envia um sinal ao registrador com decrementador Rmax na sua entrada *DW*, diminuindo o seu valor corrente em 1 (Rmax ← Rmax-1), sempre que o valor armazenado anteriormente seja maior que 1 (condicionado pela operação OR dos 2 *msb* de *Rmax*). O valor de Tmin é atualizado pelo mesmo sinal *DW*, ocasionando uma operação de *shift Left* no seu valor corrente (Tmin← 2x Tmin). O efeito desta estrutura pode ser visualizado na fig. 5.21.

6.3.3.2. Determinação de membro da vizinhança $N_c(t_k)$.

Um outro mecanismo associado deve determinar se cada célula pertence ou não a $N_c(t_k)$. Quando uma célula é identificada como ganhadora pelo WTA, ela envia sinais às vizinhas imediatas pelos 6 canais de saída No_j RDY ($0 \le j \le 5$; vide figs. 6.3 e 6.5). Este sinal ativa as vizinhas que encontram-se no estado S_1, após realizado o cálculo da distância no ciclo(1) da FSM. As vizinhas recebem o sinal por uma das entradas Ni_j RDY ($0 \le j \le 5$), colocando-as no estado S_{23} e iniciando-se o processo de atualização de pesos através do ciclo(3) da FSM.

Junto com o sinal de ativação Ni RDY as células recebem o valor da distância topológica entre elas e a ganhadora no instante t_k (1, para as vizinhas imediatas). Caso este valor seja menor que o raio corrente de $N_c(t_k)$, são ativadas as células localizadas à mesma distância topológica no nível imediatamente superior (vide fig. 6.5). A estrutura utilizada é representada pelo diagrama esquemático da figura 6.11.

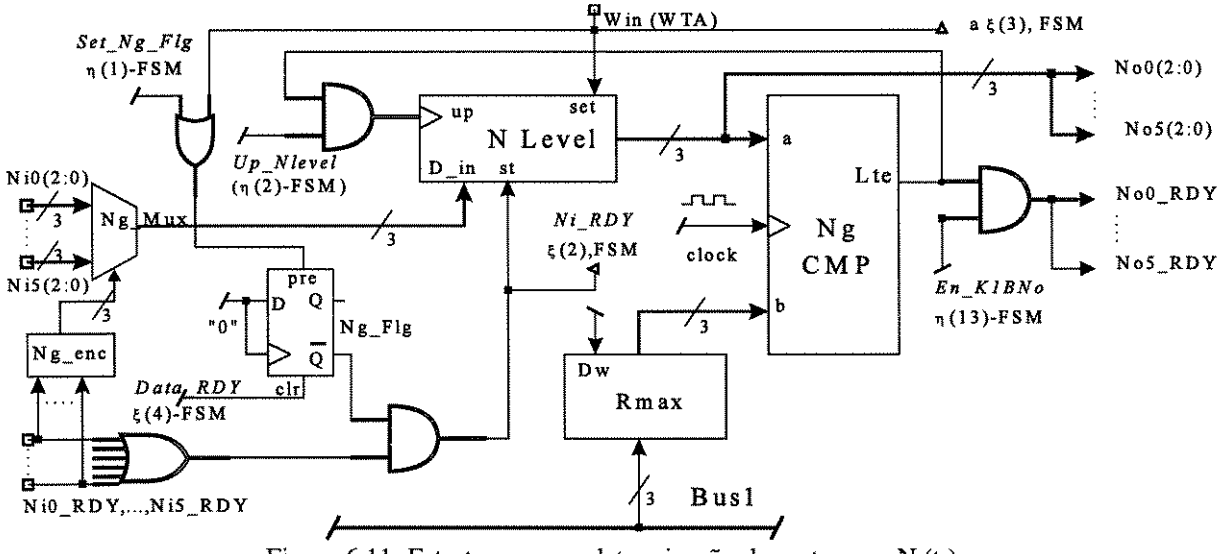


Figura 6.11. Estrutura para a determinação de pertença a $N_c(t_k)$.

Na figura anterior um registrador de 3 bits chamado de *NLevel* armazena o valor da distância topológica transmitido entre a célula corrente e as vizinhas imediatas. O bloco *Ng CMP* realiza a comparação entre *Nlevel* e o raio corrente de $N_c(t_k)$ armazenado em *Rmax*, a saída é "1" lógico se *NLevel* ≤ *Rmax*. O *Flip-Flop-D*, chamado de *Ng Flg*, coloca um "1" na saída Q ao receber um sinal de *Win* ou da saída η(1) da FSM, indicando que a célula corrente não pode ser ativada por outra vizinha. O processo inicia-se com a recepção de um sinal na entrada WIN, vinda do bloco WTA, indicando que a célula corrente é a ganhadora. Os outros blocos, junto com a descrição completa da operação são descritos na seqüência.

6.3.3.2.1. Caso da célula ganhadora.

Passo	Operaçao	Controle
1a	NLevel ← 1 , registrador inicializado	set ← "1"; bloco WTA
1b	Ng Flg ← "1"; saída Q/FFD="1"	pre/FFD ←"1" ent. Win
2a	NLevel ≤ Rmax ? (comparação sinc.)	*@clock* "0"→"1"
2b	No0(2:0), ... ,No5(2:0) ← 1	1 ← NLevel (2:0)
3	"1" ← LTE/Ng CMP	*@clock* "0"→"1"
4	(No0 RDY,...,No5 RDY) ← "1"	@S_{13} FSM-*En K1BNo*

Tabela 6.7. Operação para determinar pertença a $N_c(t_k)$, caso célula ganhadora.

Algoritmo de Operação (Tabela 6.7).
Passos 1a, 1b. Ambos os passos são executados de forma concorrente. Ao haver uma transição "0"→"1" na entrada Win, conectada à saída do bloco WTA, a célula passa ao estado S_{12} da FSM, o sinal inicializa o registrador *Nlevel* em 1 (representado em 3 bits), ao mesmo tempo que um *flag* é levantado por meio do *flip-flop*-D (*Ng Flg*), indicando que a célula não pode ser mais ativada por uma vizinha, no passo corrente de operação do SOFM, indicado por t_k.

Passos 2a,2b. Passos executados concorrentemente. O valor corrente de *NLevel* (1) é comparado com o valor de *Rmax*, o raio corrente de $N_c(t_k)$, sendo a comparação sincronizada com a borda de subida do *clock* global distribuído. De forma assíncrona, após o atraso de *set-up* do registrador *NLevel*, o valor armazenado é enviado às 6 células vizinhas pelas saídas {No0,...,No5}(2:0).

Passo 3. Após o tempo de atraso do comparador Ng CMP, é colocado um "1" lógico na sua saída LTE, sempre que a condição (Nlevel ≤Rmax) seja verdadeira.

Passo 4. Ao passar a célula ao estado S_{13} (um ciclo de *clock* após o passo 1), a saída η(13) da FSM, gera uma transição "0"→"1", desta forma indicando às células vizinhas (cujo *Ng flg* esteja abilitado) pelos canais de saída No0 RDY, ... ,No5 RDY que o valor corrente de *NLevel*, transmitido no Passo 2b, já esta disponível e estável.

A simulação correspondente a esta estrutura é ilustrada no Capítulo7, *secção 7.4-Simulações*, realizadas por meio do simulador *QuickSim*™ [80].

6.3.3.2.2. Caso da célula não ganhadora pertencente a $N_c(t_k)$.

Passo	Operaçao	Controle
1	Ni0(2:0) | , ..., | Ni5(2:0)← y	No_j(2:0) célula vizinha
2	Ni0 RDY |, ..., | Ni5RDY← "1"	No_j RDY célula vizinha
3a	IF(NOT Q/Ng Flg= "1") ⇒ Nlevel ← Ng Mux	st/NLevel ← "1" saída Ng enc
3b	S_1 → S_{23} FSM	ξ(2) FSM ← "1"
4a	NOT Q/Ng Flg ← "0"	@S_{23} FSM -η(1)
4b	No0(2:0), ... , No5(2:0) ← y	y ← NLevel (2:0)
5	IF ("1" ← LTE/Ng CMP) ⇒ Nlevel ← Nlevel + 1	@*clock* "0"→"1" @S_{24} FSM -η(2)
6	No0(2:0), ... , No5(2:0) ← y + 1	y ← NLevel (2:0)
7(*)	(No0 RDY,...,No5 RDY) ← "1"	@S_{13} FSM-*En K1BNo*

Tabela 6.8. Operação para determinar pertença a $N_c(t_k)$, caso célula não ganhadora em $N_c(t_k)$.

A tabela 6.8, descreve a operação para o caso da célula não ganhadora, mas pertencente à vizinhança $N_c(t_k)$ de raio corrente Rmax. (Fig. 6.11). O passo 7, é explicado na seqüência.

Passo 1. A célula recebe um valor *y* por alguma das entradas $Ni_j(2:0)$ $(0 \leq j \leq 5)$. O sinal é transmitido por alguma das células vizinhas imediatas. Caso a vizinha imediata for a ganhadora é recebido o valor *y*=1 (vide passo 2b, na Tabela 6.7). O valor de *y* varia entre 1 e o raio corrente de $N_c(t_k)$, e representa a distância topológica existente entre a célula corrente e a ganhadora no instante indicado por t_k.

Passo 2. É recebido um pulso de largura controlada pela FSM por alguma das entradas Ni_j RDY $(0 \leq j \leq 5)$. O sinal é enviado por alguma das células vizinhas imediatas, após ter o valor corrente de NLevel atualizado e estável na saída (vide o caso da Tabela 6.7, passo 4).

Passo 3a, 3b. Passos executados concorrentemente. Caso a célula corrente não esteja ativa (saída Q=0, no *flip-flop*-D *Ng Flg*) um "1" lógico é transmitido à entrada *st* do registrador NLevel (vide Fig. 6.11) e à entrada $\xi(2)$ da FSM.

Em (a), NLevel armazena o valor corrente presente na saída do multiplexer *Ng Mux* e condicionada pela saída do decodificador *Ng enc*. O sinal Ni_j RDY deve estar estável o tempo suficiente para ser codificado por *Ng enc*, que seleciona como saída uma das suas entradas Ni0(2:0),...,Ni5(2:0). Em (b), o mesmo sinal que permite armazenar *NLevel*, ativa a célula que, encontrando-se no estado S_1, passa para o estado S_{23}, iniciando-se o ciclo (3) da FSM.

Passo 4. (a) A célula recebe um sinal *Set Ng Flg*, desde a sua própria FSM no estado S_{23}. Após o atraso de *set-up* da porta OR (fig. 6.11), a entrada *pre* (*preset*) do *flip-flop*-D recebe um "1" que coloca *Ng Flg* em Q="1"/ (NOT Q)="0", colocando a célula em estado de *reserva*, impedindo-a de ser ativada por qualquer sinal de outra vizinha.

(b) Como conseqüência do passo 3(a), após o atraso de *set-up* do registrador NLevel, o valor armazenado *y* é transmitido às 6 saídas $No_j(2:0)$ $(0 \leq j \leq 5)$.

Passo 5. Após o tempo de atraso do comparador Ng CMP, é colocado um "1" lógico na sua saída LTE, sempre que a condição (NLevel $\leq$Rmax) seja verdadeira. Logo, no estado S_{24} da FSM, é realizada uma operação AND entre as saída h(2) da FSM e LTE/Ng CMP, que incrementa em 1 o valor corrente de *y*, i.e., $y \leftarrow y + 1$.

Passo 6. O valor atualizado de NLevel é colocado nas saídas $No_j(2:0)$ $(0 \leq j \leq 5)$, após o tempo de *set-up* do registrador.

Passo 7. No estado S_{13}, a saída $\eta(13)$ da FSM, gera uma transição "0"→"1", indicando às células vizinhas (cujo *Ng flg* esteja habilitado), pelos canais de saída No_j RDY, $(0 \leq j \leq 5)$ que o valor corrente de NLevel, transmitido no Passo 6, já esta disponível e estável. Caso a célula esteja localizada no limite da vizinhança $N_c(t_k)$, em S_{13} o valor LTE/Ng CMP="0", portanto este passo **não é executado** (vide Cap7- Resultados das simulações lógicas, realizadas por *QuickSim* [80]).

6.3.4. A aritmética de *Neuron*.

Antes de descrever as estruturas encarregadas das operações aritméticas de *Neuron*, precisa-se conhecer a forma de representação numérica dos vetores (pesos e treinamento) e dos parâmetros do algoritmo. Nesta secção é explicado o sistema numérico utilizado e a forma como a aritmética é implementada na arquitetura.

6.3.4.1. Sistema numérico.

Foi decidido empregar um sistema numérico baseado em operações em ponto fixo. A decisão fundamenta-se principalmente em função do custo em silício menor em comparação à representação em ponto flutuante. A implementação de um protótipo trabalhando em ponto fixo (*fixed-point* -FXP) com representação numérica em 12 bits com sinal (vide figura 6.12) , foi considerado de precisão suficiente (vide 6.3.4.2 e [82]).

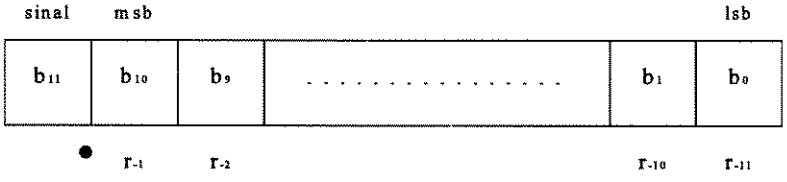


Figura 6.12. Representação numérica FXP de 12 bits com sinal.

Se **x** é um número $\in \Re^n$, tal que, $0 \leq \mathbf{x} \leq 1$, representado em algum sistema numérico, de forma geral o seu valor pode ser expresso por :

$$x = \sum_{i,j}^{N} (b_i \times r^{-j}) \qquad (6.1)$$

Em um sistema numérico digital, a expressão (6.1) pode representar o número de 12 bits com sinal da figura 6.12, onde $10 \geq i \geq 0$ e $-1 \geq j \geq -11$. Cada dígito $b_i \in \{0,1\}$, *r* representa a *base* do sistema numérico (r=2, para a representação digital) e b_{11} assume o valor 0 para un número positivo e 1 para um valor negativo. O dígito b_{10} representa o bit mais significativo (*msb*) de **x**. Os números negativos são representados em *complemento de 2* [81].

Segundo mencionado nos capítulos 2 e 3, o algoritmo SOFM deve utilizar vetores normalizados, especialmente na fase de determinação da distância mínima entre **x** e **w** (vide Tabela 5.2), o que determina que os componentes dos vetores mencionados assumam valores entre 0 e 1 ($\mathbf{x},\mathbf{w} \in \Re^n$). Foram modelados e simulados blocos aritméticos utilizando distinta precisão numérica, a fim de verificar a sua influência no resultado final das operações. Os resultados deste modelamento em VHDL em alto nível de abstração (não sintetizável), estão resumidos a seguir, na Tabela 6.9. As operações matemáticas de interesse são representadas no diagrama da figura 6.14. Na figura, visualiza-se após cada operação um erro gerado produto da representação numérica e do truncamento realizado após as operações de multiplicação. Uma operação de multiplicação de 2 números de *m* e *n* bits, produz um resultado de (*m*+*n*) bits. No protótipo projetado, foram truncados 12 bits após a operação de multiplicação, para manter o

formato da representação descrito na figura 6.12.

6.3.4.2. Determinação do número mínimo de bits.

Para a implementação tanto digital quanto analógica do SOFM, as entradas $\mathbf{x}=(x_1, x_2, ...,x_n)$,onde *n* é um número finito, estarão sempre limitadas de tal forma que $(X_{min} \leq \|\mathbf{x}\| \leq X_{max})$. Da mesma forma, os pesos limitam-se a $(W_{min} \leq \|\mathbf{w}\| \leq W_{max})$. Podemos considerar que os valores de x_i e w_i são de alguma forma *quantizados* por uma função Q(·), definindo-se um *passo de quantização*, q, expresso por:

$$q = \frac{X_{max} - X_{min}}{2^b - 1} \tag{6.2}$$

Na expressão (6.2), b representa o número de bits da representação binária de **x** e **w**. Os efeitos da sua quantização na implementação em *hardware* do SOFM foram investigados por Thiran et al.[82]. Para o caso bi-dimensional de **x** e **w**, a função de quantização é ilustrada na figura 6.13.

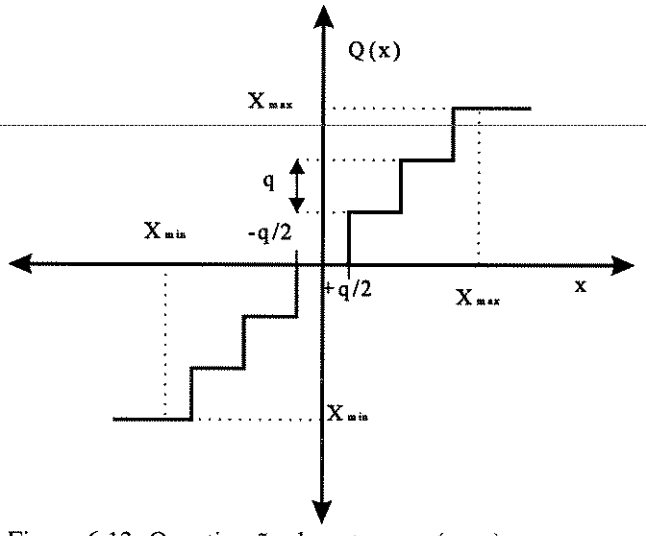


Figura 6.13. Quantização do vetor $\mathbf{x} = (x_1,x_2)$

A função Q(x) representada na figura 6.13 representa um *mapeamento* dos componentes de **x** em múltiplos do passo de quantização q.

Segundo Thiran et al., há 3 fatores importantes que influem no comportamento da rede após a quantização: o número mínimo de bits, a função $\alpha(t_k)$ e a função de interação lateral Λ. Para uma rede de M×M células de processamento, supondo uma distribuição uniforme sobre um espaço quadrado de a × a (onde $X_{max}=1$ e $X_{min}=0$), obtém-se que o número mínimo de bits necessários para quantizar os pesos é expresso por :

$$b > \log_2\left(1 \cdot \frac{2M}{a}\right) \tag{6.3}$$

A restrição é então dependente da distribuição de entrada p(**x**) e do número de neurônios da rede. Considerando as expressões (6.2) e (6.3), determina-se que com b=11 bits, poderiam implementar-se redes com até 1023×1023 células de processamento. No entanto, os resultados das simulações realizadas por Thiran et al. demonstram que esta única restrição é insuficiente. Os parâmetros que definem a forma de $\alpha(t_k)$ e da função de vizinhança (interação lateral) $\Lambda(r,t_k)$ influem notoriamente na convergência adequada do SOFM, para um determinado número de bits. A determinação do número mínimo de bits para uma implementação dedicada a uma aplicação particular, tal como a descrita em 4.6, necessita de um completo estudo da distribuição das entradas p(**x**), que permita definir a resolução e a forma do espaço de entradas, o mínimo número de neurônios e as coordenadas ideais para $\mathbf{w} \in \mathbb{R}^n$.

6.3.4.3. Propagação de erro para precisão finita em SOFM.

Numa máquina de cálculo aritmético de precisão finita, um número exato $y \in \mathbb{R}$ é representado internamente pela aproximação $\hat{y}= y(1 + \varepsilon)$, onde $\varepsilon = (\hat{y} - y)/$ é conhecido como o *erro de aproximação relativa* [83].

Na computação da distância $\hat{y}=d(\mathbf{x},\mathbf{w})$, o resultado final em precisão finita é produto de uma seqüência de operações com erros propagados através das unidades aritméticas e cuja fonte inicial é o própio erro de aproximação relativa por causa da quantização de **x** e **y**.

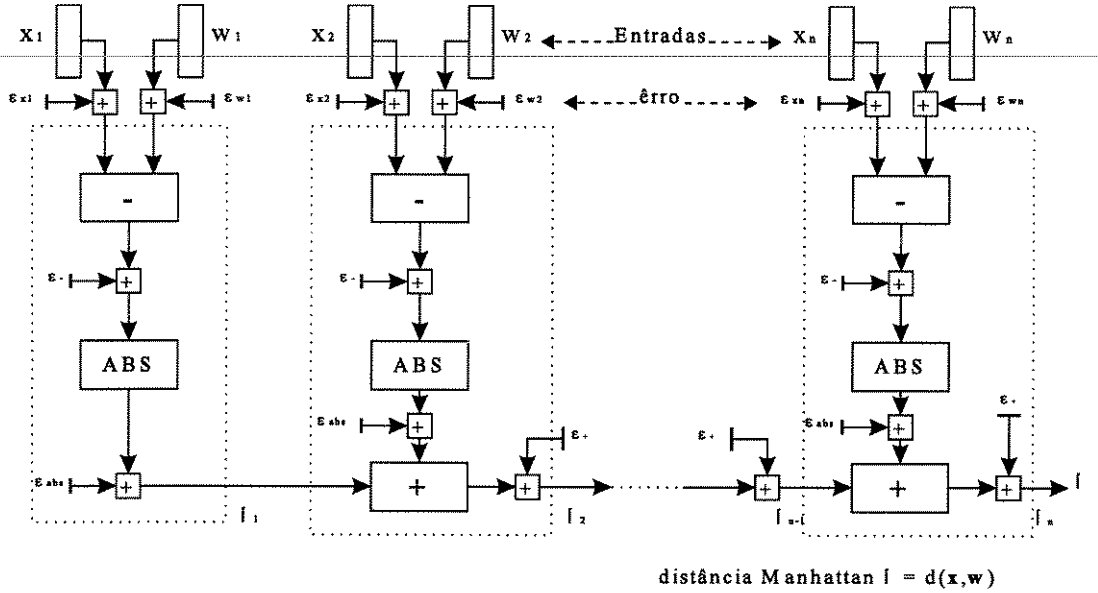


Figura 6.14.Geração e propagação de êrro para o cálculo de distância Manhattan.

A figura 6.14 representa o grafo do cálculo de distância d(**x**,**w**), utilizando a distância Manhattan e a geração e propagação de êrro associado à operação. O erro originado por causa da precisão finita de **x** e **w** é propagado através do grafo de cálculo, representado por ϵ_x e ϵ_w, respectivamente. Outros erros são originados pela computação em precisão finita, associados às operações de substração x_i-w_i e o valor absoluto $\|x_i$-$w_i\|$, onde $1 \le i \le n$, os erros são representados respectivamente por ϵ_- e ϵ_{abs}.

O resultado final da computação de $y = \phi(x,w)$, onde ϕ representa um operador, em precisão finita, pode ser expresso por $\hat{y}$ através de uma aplicação sucessiva de operadores ϕ_1, ϕ_2,...,ϕ_n, tal que (cf.[84]):

$$\hat{y} = \phi_n(\cdots (\phi_2(\phi_1(x + \epsilon_x, w + \epsilon_w) + \epsilon_{\phi 1}) + \epsilon_{\phi 2}) \cdots) + \epsilon_{\phi n},$$

onde $\epsilon_{\phi i}$ representa o erro ocasionado pela operação em precisão finita ϕi. O diagrama da figura 6.14 pode ser então expresso por (6.4)-(6.6):

$$\begin{aligned} \hat{y}_1 &= \phi_1((x_1 \cdot \epsilon_{x1}) - (w_1 \cdot \epsilon_{w1}) \cdot \epsilon_-) \cdot \epsilon_{\phi 1} \\ &= \phi_1(x_1 - w_1 \cdot \epsilon_{x1} - \epsilon_{w1} \cdot \epsilon_-) \cdot \epsilon_{\phi 1} \\ &= \|x_1 - w_1 \cdot \epsilon_{x1} - \epsilon_{w1} \cdot \epsilon_-\| \cdot \epsilon_{abs} \end{aligned} \tag{6.4}$$

para o segundo componente, o o resultado parcial $\hat{y}_2 = \phi_2(\hat{y}_1, \phi^*_2(x_2, w_2) + \epsilon_{\phi^* 2}) + \epsilon_{\phi 2}$

$$\begin{aligned} \hat{y}_2 &= \phi_2(\hat{y}_1, \phi'_2((x_2 \cdot \epsilon_{x2}) - (w_2 \cdot \epsilon_{w2}) \cdot \epsilon_-) \cdot \epsilon_{\phi 2'}) \cdot \epsilon_{\phi 2} \\ &= \phi_2(\hat{y}_1, \phi'_2(x_2 - w_2 \cdot \epsilon_{x2} - \epsilon_{w2} \cdot \epsilon_-) \cdot \epsilon_{\phi 2'}) \cdot \epsilon_{\phi 2} \\ &= \phi_2(\hat{y}_1, \|x_2 - w_2 \cdot \epsilon_{x2} - \epsilon_{w2} \cdot \epsilon_-\| \cdot \epsilon_{abs2}) \cdot \epsilon_{\phi 2} \\ &= \hat{y}_1 \cdot \|x_2 - w_2 \cdot \epsilon_{x2} - \epsilon_{w2} \cdot \epsilon_-\| \cdot \epsilon_{abs2}) \cdot \epsilon_- \end{aligned} \tag{6.5}$$

a saída do resultado final $y=d(\mathbf{x},\mathbf{w})$ em precisão finita é então expresso por $\hat{y}=\hat{y}_n$:

$$\begin{aligned} \hat{y}_n &= \phi_2(\hat{y}_{n-1}, \phi'_n((x_n \cdot \epsilon_{xn}) - (w_n \cdot \epsilon_{wn}) \cdot \epsilon_-) \cdot \epsilon_{\phi n'}) \cdot \epsilon_{\phi n} \\ &= \phi_n(\hat{y}_{n-1}, \phi'_n(x_n - w_n \cdot \epsilon_{xn} - \epsilon_{wn} \cdot \epsilon_-) \cdot \epsilon_{\phi n'}) \cdot \epsilon_{\phi n} \\ &= \phi_n(\hat{y}_{n-1}, \|x_n - w_n \cdot \epsilon_{xn} - \epsilon_{wn} \cdot \epsilon_-\| \cdot \epsilon_{absn}) \cdot \epsilon_{\phi n} \\ &= \hat{y}_{n-1} \cdot \|x_n - w_n \cdot \epsilon_{xn} - \epsilon_{wn} \cdot \epsilon_-\| \cdot \epsilon_{absn}) \cdot \epsilon_- \end{aligned} \tag{6.6}$$

Nas expresões (6.4)-(6.6), ϵ_-,ϵ_+ e ϵ_{abs} representam os erros ocasionados pelas operações em precisão finita de subtração, soma e valor absoluto, respectivamente.

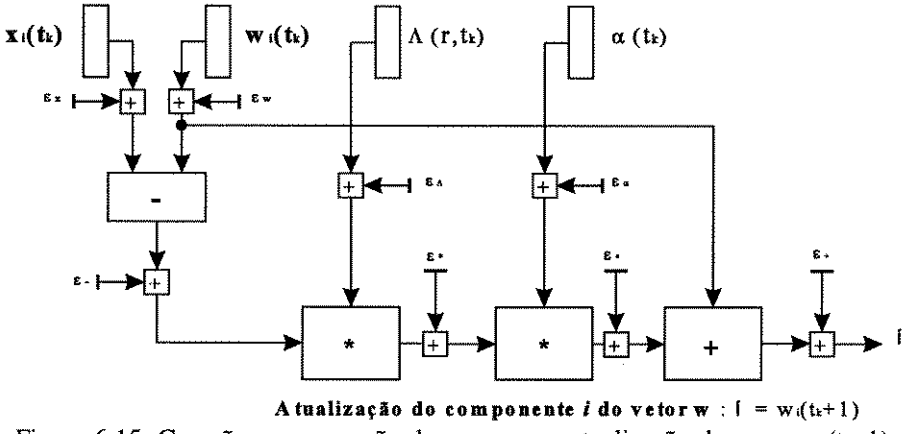


Figura 6.15. Geração e propagação de erro para a atualização de pesos, $\mathbf{w}(t_k+1)$.

Na figura 6.15 representa-se o grafo de cálculo incluindo a geração e propagação de erros ocasionados pela precisão finita, na atualização de pesos do componente i do vetor **w**. Os erros ϵ_x e ϵ_w são ocasionados pela quantização de x_i e w_i, respectivamente. As operações em ponto-fixo de subtração, soma e multiplicação geram os respectivos erros ϵ_-, ϵ_+, ϵ_*. As funções de ganho $\alpha(t_k)$, de vizinhança variável $\Lambda(r,t_k)$ introduzem os erros ϵ_αe ϵ_Λ.

Da mesma forma que para o cálculo da distância Manhattan, a atualização de **w** pode ser expressa pela aplicação sucessiva dos operadores (vide fig. 6.15), definidos a seguir:

$$\begin{aligned} \hat{y}_1 &= \phi_1(x \cdot \epsilon_x, w \cdot \epsilon_w) \cdot \epsilon_{\phi 1} \\ \hat{y}_2 &= \phi_2(\hat{y}_1, \alpha \cdot \epsilon_\alpha) \cdot \epsilon_{\phi 2} \\ \hat{y}_3 &= \phi_3(\hat{y}_2, \Lambda \cdot \epsilon_\Lambda) \cdot \epsilon_{\phi 3} \\ \hat{y}_4 &= \phi_4(\hat{y}_3, w \cdot \epsilon_w) \cdot \epsilon_{\phi 4} \end{aligned} \qquad (6.7)$$

expandindo $\hat{y}_3$, $\hat{y}_2$ e $\hat{y}_1$ em $\hat{y}_4$, obtém-se a expressão completa para o resultado final $\hat{y} = \hat{y}_4$. Os operadores $\phi 1$, $\phi 2$=$\phi 3$ e $\phi 4$, correspondem à subtração, multiplicação e soma, respectivamente. Os erros associados são ϵ_1, ϵ_2, ϵ_3 e ϵ_4. A expressão resultante para $\hat{y}$ é :

$$\hat{y} = \phi_4(\phi_3(\phi_2(\phi_1(x \cdot \epsilon_x, w \cdot \epsilon_w) \cdot \epsilon_{\phi 1}, \alpha \cdot \epsilon_\alpha) \cdot \epsilon_{\phi 2}, \Lambda \cdot \epsilon_\Lambda) \cdot \epsilon_{\phi 3}, w \cdot \epsilon_w) \cdot \epsilon_{\phi 4} \qquad (6.8)$$

A expressão (6.8) corresponde à operação da atualização para cada componente dos vetores **w** , em precisão finita. Em *Neuron*, a representação em ponto fixo em 11 bits com sinal, permite uma precisão de 0.0005 ($1/2^{11}$). Nas próximas secções serão descritas as 2 operações principais, considerando a arquitetura e a estrutura de controle descrita em 6.3.1.

6.3.5. Cálculo da Distância $|x(t_k)-w_i(t_k)|$.

A implementação da *distância Manhattan*, $d_i(\mathbf{x},\mathbf{y})$, é realizada no ciclo(1), entre os estados S_2 e S_{11} da FSM descrita em 6.3.1. (vide figuras 6.4 e 6.6). Seu grafo de cálculo é representado pela figura 6.14. A operação completa, incluindo os sinais de controle, descreve-se na tabela 6.9., representando-se o estado onde é realizada a operação corrente e o mecanismo de controle associado, executado pela saída $\eta(j)$ da FSM ($0 \le j \le 23$).

Estado	Operação	Controle por $\eta(j)$ da FSM
S_2	Addr CT← 0 (I=0)	*Rst AddrCT* = '1' - $\eta(22)$
S_3	$Diff_i$ ← BUS1(11:0) - Data In Dim CT← Dim CT-1	*ST Diff* = '1' - $\eta(17)$ *DW DimCT* = '1' - $\eta(3)$
S_4	Addr CT ← Addr CT + 1	*UP AddrCT* = '1' - $\eta(21)$
S_5	Addr CT← 0 (i=0) DIM CT ← DIM(1:0) B ← data out(11:0)/K	*Rst AddrCT* = '1' - $\eta(22)$ *ST DimCT* = '1' - $\eta(4)$ *St B* = '1' - $\eta(15)$

S_6	BUS1(11:0) ← $Diff_l$ BUS2(11:0) ← Wg_l Operação = Resta Op1 - ABS(11:0), Op2 ← BUS2(11:0)	*En Diff* = '1' - η(18) *En Wg* = '1' - η(20) *Sel Add* = '1' - η(9) *St Ops* = '1' - η(11) (armz. nível)
S_7	BUS1(11:0) ← L3 (*Latch com tri state*) $Diff_l$ ← BUS1(11:0) Ativação do Bloco ABS (compl. a 2)	*En Add* = '1' - η(10) *St Diff* = '1' - η(17) *En ABS* = '1' - η(7)
S_8	BUS2(11:0)← AC (@S_2, AC ← 0) BUS1(11:0) ← "Z" (*Latch L3 em Hi Z*) Operação = Soma Op1 - ABS(11:0), Op2 ← BUS2(11:0)	*En AC* = '1' - η(6) *En Add* = '0' - η(10) *Sel Add* = '0' - η(9) *St Ops* = '1' - η(11) (armz. nível)
S_9	BUS1(11:0) ← L3 (*Latch com tri state*) AC ← BUS1(11:0) Dim CT ← Dim CT-1	*En Add* = '1' - η(10) *St AC* = '1' - η(5) *DW DimCT* = '1' - η(3)
S_{10}	Addr CT ← Addr CT + 1 BUS1(11:0) ← "Z" (*Latch L3 em Hi Z*)	*UP AddrCT* = '1' - η(21) *En Add* = '0' - η(10)
S_{11}	"1" ← RDY B ← B÷2 (*Shift Right* B)	η(10) *Sh B* = '1' - η(14)

Tabela 6.9. Controle e Operação de determinação da distância Manhattan, enviada ao WTA.

Na seqüência, descreve-se a operação da determinação de $d_i(\mathbf{x},\mathbf{w})$, realizada de forma concorrente por cada uma das células i pertencentes à rede. A estrutura que realiza a operação descrita a seguir, representa-se na figura 6.17. Os passos 1-3 permitem armazenar $\mathbf{x}=(x_0,...,x_3)$ nos registradores Diff0,...,Diff3 respectivamente. Nos passos 5-10 determina-se $\|\mathbf{x} - \mathbf{w}_i\|$.

Passo 1 - S_2. Ao receber um vetor de entrada **x**, no barramento *Data In*(11:0)/BUS1(11:0), a FSM passa do estado S_1 para S_2. A transição é ativada pelo sinal de controle do barramento de entrada *Control*(3) -Data RDY, que passa de "0"→"1", ao mesmo tempo coloca-se AC em "0" lógico (*reset*). O registrador *Addr CT* ="0" (*reset*) aponta para o primeiro componente (l=0) do conjunto de registradores $Diff_l/Wg_l$ ($0 \le l \le 3$).

Passo 2 - S_3. No estado S_3 da FSM, armazena-se o valor presente em BUS1(11:0) no registrador *Diff*$_l$, ao mesmo tempo diminui-se em 1 o valor corrente armazenado no registrador *Dim CT*, cujo valor inicial foi armazenado em S_1. Se a saída de *Dim CT* ≠ "1" (indicando que não há mais componentes em **x**), passa-se ao estado S_5, caso contrário, ao estado S_4.

Passo 3 - S_4. Incremento do valor corrente armazenado em Addr CT, apontando-se ao seguinte componente (l = l + 1) do conjunto de registradores $Diff_l/Wg_l$ ($0 \le l \le 3$). Ao *Passo2*, estado S_3..

Passo 4 - S_5. Realiza-se o *reset* de Addr CT para apontar novamente a $Diff_0/Wg_0$. Armazena-se

no registrador *Dim CT* a dimensão de **x**/**w**. Esse estado é aproveitado para armazenar no registrador *B* o valor corrente de K1, utilizado no cálculo da função $\alpha(t)$, no ciclo(2) da FSM.

Passo 5 - S_6. Colocam-se nos barramentos BUS1(11:0) e BUS2(11:0), os valores armazenados em $Diff_l$ e Wg_l respectivamente. Seleciona-se a operação de subtração para a unidade *Add Sub*. Inicia-se o armazenamento (controlado por nível) nos *latches* OP1 e OP2.

Passo 6- S_7. O resultado da substração OP1-OP2 (x_l-w_{li}) já estabilizado, é armazenado no *latch* L3, que controla nesse momento o acesso (*driving*) ao BUS1(11:0). A diferença armazena-se em *Diff*$_l$, cujo controle é realizado por nível. Ativa-se o bloco de valor absoluto, que realiza o complemento a 2 do valor presente na sua entrada, caso tal número for negativo, i.e., se o *msb*, BUS1(11) = "1".

Passo 7- S_8. Seleciona-se a operação de soma para a unidade *Add Sub*. O *latch* L3 controla o acesso ao BUS1(11:0) e o acumulador AC ao BUS2(11:0). O valor inicial de AC=0, é ocasionado pelo *reset* do sinal Data RDY (vide fig. 6.16), na transição S1→ S2. Inicia-se o armazenamento (controlado por nível) nos *latches* OP1 e OP2.

Passo 8 -S9. O resultado da soma de OP1+OP2, $|x_l$-$w_{li}|$+AC já estabilizado, é armazenado no *latch* L3, que controla nesse momento o acesso (*driving*) ao BUS1(11:0). O resultado armazena-se em *AC*, cujo controle é realizado por nível.

Diminui-se em "1" o valor corrente de *Dim CT*. Caso houver mais componentes em **x** (saída de DIM CT, *zero dim*="0"), passe-se ao estado S_{10}. Caso contrário (*zero dim*="1"), passa-se ao estado S_{11}, *Passo10*.

Passo 9 -S_{10}. O barramento BUS1(11:0) é colocado em estado de *alta impedância*. Incrementa-se em uma unidade o valor de *Addr CT*, para apontar ao registrador $Diff_{l+1}$. Para o estado S_6.

Passo 10 -S_{11}. Um sinal RDY é enviado ao bloco WTA, avisando-o que o valor final de $d_i(\mathbf{x},\mathbf{w})$ já está disponível e estável na saída DIST(12:0) da célula i (vide fig. 6.3).

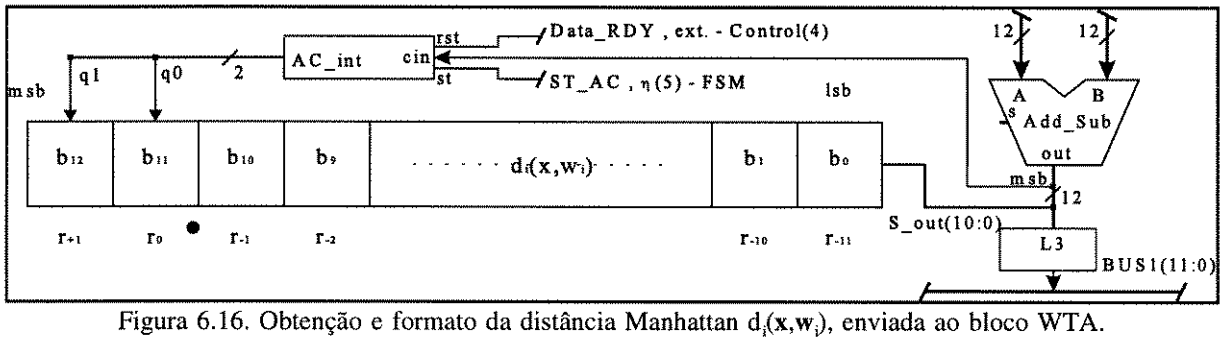


Figura 6.16. Obtenção e formato da distância Manhattan $d_i(\mathbf{x},\mathbf{w}_i)$, enviada ao bloco WTA.

O valor final transmitido por DIST(12:0) é montado tomando-se diretamente os 11 bits menos significativos desde a saída da unidade aritmética *Add Sub* (que opera em 12 bits). O *msb* (dígito mais significativo) é armazenado em um acumulador de 2 bits (*AC Int* , fig. 6.16), onde monta-se a parte inteira de $d_i(\mathbf{x},\mathbf{w})$ em b_{12} e b_{11}. A célula *Neuron* trabalha com **x** e **w** máximos

de 4 componentes, onde cada componente $0 \leq x_l, w_l \leq 1$. Portanto, a parte inteira acumulada de $d_i(\mathbf{x},\mathbf{w}_i)$ e armazenada em *AC Int* não excederá de $(4)_{10}$, representado de forma binária por 2 bits. O formato completo da representação é ilustrado na figura 6.16.

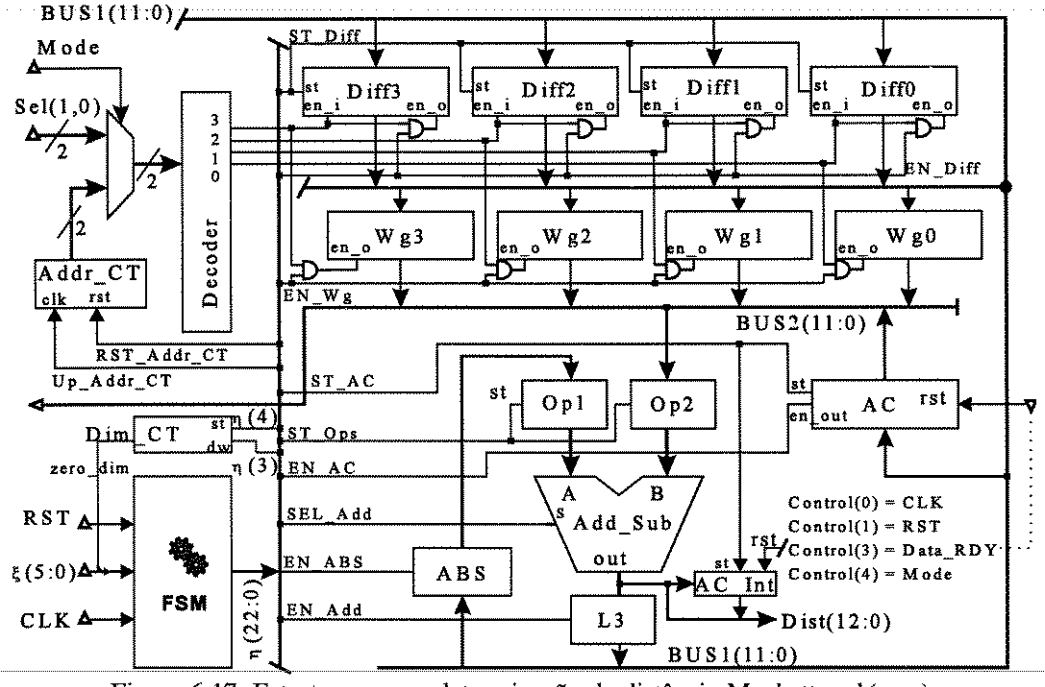


Figura 6.17. Estrutura para a determinação da distância Manhattan $d_i(\mathbf{x},\mathbf{w}_i)$.

Na figura anterior representa-se a estrutura utilizada por *Neuron* para a determinação da distância Manhattan. Cada célula possui esta estrutura para realizar a operação de forma concorrente, a máquina de estados (FSM) controla toda a operação até o envio do sinal RDY, no *Passo10*, para avisar ao bloco WTA do término desta fase do algoritmo. Na seqüência, descrever-se-á a operação de atualização do vetor de pesos **w**.

6.3.6. Atualização do vetor de pesos $w(t_k)$.

Os casos da célula ganhadora do processo competitivo e o da pertencente à vizinhança $N_c(t_k)$, serão descritos separadamente. A diferença na operação firma-se unicamente na determinação do fator de inibição lateral, realizado nos estados S_{23}-S_{26} da FSM. Após realizada a fase descrita em 6.3.5, o bloco WTA determina qual é a célula ganhadora *c* do processo competitivo no instante t_k, o raio corrente de $N_c(t_k)$ determina quais são as células cujos pesos serão atualizados junto com os de *c*.

6.3.6.1. Caso da célula ganhadora.

A atualização do vetor **w**, para o caso da célula ganhadora, é realizada no ciclo(2), entre os estados S_{12} e S_{22} da FSM descrita em 6.3.1. (vide figuras 6.4 e 6.6). Seu grafo de cálculo é representado pela figura 6.15. A operação completa descreve-se na tabela 6.10., representando-se

o estado onde é realizada a operação corrente e o seu mecanismo de controle, executado pela saída $\eta(j)$ da FSM ($0 \leq j \leq 23$).

Estado	Operação	Controle por $\eta(j)$ da FSM
S_{12}	Addr CT← 0	*Rst AddrCT* = '1' - $\eta(22)$
S_{13}	BUS1(11:0) ← K1, BUS2(11:0)←B Op1←BUS1(11:0), Op2 ← BUS2(11:0)	*En K1BNo* = '1' - $\eta(13)$ *St Ops* = '1' - $\eta(11)$ (armz. nível)
S_{14}	BUS1(11:0) ← L3 (*Latch com tri state*) AC ← BUS1(11:0)	*En Add* = '1' - $\eta(10)$ *St AC* = '1' - $\eta(5)$
S_{15}	BUS2(11:0)← AC - $\alpha(t_k)$ BUS1(11:0) ← $Diff_i$ Op1←BUS1(11:0), Op2 ← BUS2(11:0)	*En AC* = '1' - $\eta(6)$ *EN Diff* = '1' - $\eta(18)$ *St Ops* = '1' - $\eta(11)$ (armz. nível)
S_{16}	multiplicação de OP1 x OP2	*estado de transição*
S_{17}	BUS1(11:0)←L1; BUS2(11:0)←L2 Op1←BUS1(11:0), Op2 ← BUS2(11:0)	*En Mul* = '1' - $\eta(8)$ *St Ops* = '1' - $\eta(11)$ (armz. nível)
S_{18}	BUS1(11:0) ,BUS2(11:0) ← "Z" (L1 e L2 em estado *Hi Z*)	*En Mul* = '0' - $\eta(8)$
S_{19}	BUS1(11:0) ← L3 (*Latch com tri state*) BUS2(11:0) ← Wg_i Op1←BUS1(11:0), Op2 ← BUS2(11:0)	*En Add* = '1' - $\eta(10)$ *En Wg* = '1' - $\eta(20)$ *St Ops* = '1' - $\eta(11)$ (armz. nível)
S_{20}	BUS1(11:0) ← "Z" (*Latch L3 em Hi Z*)	*En Add* = '0' - $\eta(10)$
S_{21}	BUS1(11:0) ← L3 (L3 controla acesso) Wg_i ← BUS1(11:0) Dim CT ← Dim CT-1	*En Add* = '1' - $\eta(10)$ *St Wg* = '1' - $\eta(19)$ *Dw DimCT* = '1' - $\eta(3)$
S_{22}	Addr CT ← Addr CT + 1	*UP AddrCT* = '1' - $\eta(21)$

Tabela 6.10. Controle e Operação de atualização do vetor de pesos **w**, para o caso da célula ganhadora.

Na seqüência, descreve-se a atualização do vetor de pesos **w**. A estrutura que realiza a operação representa-se na figura 6.19. Nos estados S_{12}-S_{14} é gerado o valor corrente de $\alpha(t_k)$, armazenando-se no registrador AC. O vetor $\mathbf{w}=(w_0,...,w_3)$ armazena-se nos registradores Wg0,...,Wg3 respectivamente. A multiplicação de OP1×OP2 é determinada em 2 fases, entre os estados S_{15}-S_{18}:

i. Geram-se os resultados parcias SUM(23:0) e COUT(23:0) na saída da unidade aritmética *Mul*, ambas saídas são truncadas e

ii. somadas pela unidade *Add Sub*, obtendo-se $\alpha(t_k)\times(Diff_i)$, onde $Diff_i=x_i-w_i$.

Posteriormente realiza-se a operação $w(t) + [\alpha(t_k)\times(Diff_i)]$, nos estados S_{19}-S_{21}.

O procedimento anterior acrescenta 3 ciclos de máquina adicionais à operação, no entanto, o multiplicador parcial (sem soma final) permite uma diminuição de área de silício de aproximadamente 15% em relação a um multiplicador com soma final.

Passo 1 - S_{12}. Ao receber um sinal pela entrada *Win*(vide figura 6.3), a FSM passa do estado S_1 para S_{12}. A transição é ativada pelo sinal enviada pelo bloco WTA, neste estado coloca-se o registrador *Addr CT* ="0" (*reset*) apontando para o primeiro componente (l=0) do conjunto de registradores $Diff_l/Wg_l$ ($0 \leq l \leq 3$).

Passo 2 -S_{13}. Habilitam-se as saídas dos registradores K1 e B. Após estabilizados, os valores são armazenados em OP1 e OP2 respectivamente, sendo o armazenamento controlado por nível.

Passo 3 -S_{14}. A saída da unidade aritmética Add Sub, operando em modo soma, é colocada em BUS1(11:0). Após estabilizado, o dado é armazenado no registrador AC, na transição "1"→"0" do sinal de controle *ST AC*. Desta forma AC armazena o valor corrente de $\alpha(t_k)$ = K1+B.

Passo 4 -S_{15}. Início da operação de multiplicação de $[\alpha(t_k) \times Diff_l]$. O controle da FSM habilita as saídas dos registradores $Diff_l$ e AC, que ingressam em OP1 e OP2, respectivamente. Após estabilizados, os dados são armazenados nos *latchs* correspondentes.

Passo 5 -S_{16}. Estado de transição utilizado para colocar em "0" o sinal *ST OPs*, permitindo o armazenamento em OP1 e OP2 (realizado por nível).

Passo 6 -S_{17}. Os *latchs* OP1 e OP2 alimentam simultaneamente as unidades aritméticas *Add Sub e Mul*. Neste estado habilitam-se as saídas do multiplicador parcial SUM(23:0) e COUT(23:0), por meio do acesso dos *latchs* L1 e L2 a BUS1(11:0) e BUS2(11:0) respectivamente. É realizado um truncamento em ambas as saídas de *Mul*, tomando-se os bits b_{22}-b_{11} de cada uma, para continuar o processamento em 12 bits. Os 2 bits mais significativos de SUM e COUT são ignorados, devido ao formato da representação (vide fig. 6.12) e ao procedimento usado na multiplicação de 2 números representados em complemento a 2, que utiliza o algoritmo de *Booth* e o procedimento conhecido como *Wallace-Tree* (cf. refs.[81], [83], [85]-[88]).

Passo 7 -S_{18}. Os *latchs* L1 e L2 são colocados em estado de alta impedância.

Passo 8 -S_{19}. Habilitam-se as saídas do *latch L3*, que neste estado já deve ter realizado a soma de [SUM(22:11) + COUT(22:11)] = $[\alpha(t_k) \times Diff_l]$, junto com a de Wg_l. Após estabilizados, ambos valores são armazenados em OP1 e OP2 respectivamente, no fim do período de S_{19} (armazenamento controlado por nível).

Passo 9 -S_{20}. O *latch* L3 é colocado em estado de alta impedância.

Passo 10 -S_{21}. A unidade aritmética combinacional *AddSub* continua realizando a soma dos valores armazenados em OP1=[SUM(22:11) + COUT(22:11)] = $[\alpha(t_k) \times Diff_l]$ e OP2=Wg_l. Ao habilitar-se o seu *latch* de saída L3, o resultado da operação $[\alpha(t_k) \times Diff_l] + Wg_l$ é colocado em BUS1(11:0), armazenando-se no fim do período de S_{21} (arm. por nível) em Wg_i, desta forma

atualizando-se. O valor de Dim CT é decrementado em uma unidade.
Caso o sinal *zero dim*="1" (saída de *Dim CT*), a máquina volta ao estado S_1, indicando que não há mais componentes em $\mathbf{w}=(w_0,...,w_3)$ e que o processo de atualização está finalizado.
Caso contrário, zero dim="0", a máquina passa para o estado S_{22}, continuando o processo.

Passo 11 - S_{22}. Incrementa-se o valor corrente do contador Addr CT, desta forma apontando para o conjunto de registradores x/w_{l+1}. O processo continua no *Passo4*, estado S_{15}.

Para o caso da célula ganhadora, o processo descrito permite realizar a atualização de um componente de **w** em 7 ciclos de máquina : S_{15}-S_{21}. O processo completo é realizado adicionando os 3 ciclos-estados, S_{12}-S_{14}, correspondentes à obtenção de $\alpha(t_k)$, mais 7 ciclos de máquina por cada componente de $\mathbf{w}=(w_0,...,w_3)$.

6.3.6.2. Caso da célula pertencente à vizinhança $N_c(t_k)$.

Neste caso é preciso determinar o fator de inibição lateral, tratado no capítulo 3 (vide Cap.3, secção 3.3.3. e figs. 3.5-3.6). A célula pertencente a $N_c(t_k)$ determina $\Lambda(t_k,r)$ (vide expressões (2.6) e (2.7)) em função da distância topológica *r* da célula ganhadora do processo competitivo. A versão em *hardware* da expressão equivalente a (3.7) utiliza como base um conjunto de registradores de deslocamento (*shift-registers*), conseguindo-se implementar o efeito da expressão (5.1), descrita no capítulo 5. A curva resultante ilustra-se na figura 6.18.

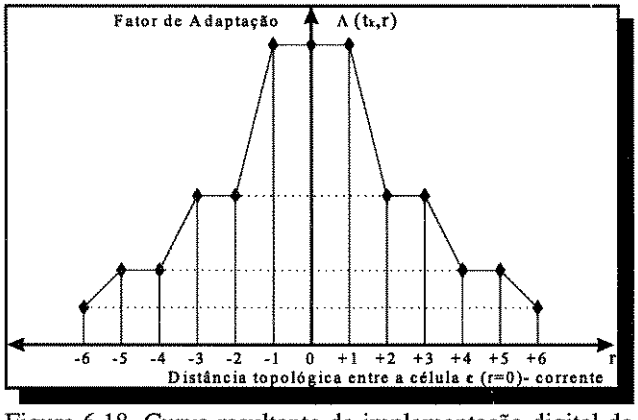


Figura 6.18. Curva resultante da implementação digital do fator de interação lateral $\Lambda(t_k,r)$.

A curva na fig.6.18 representa o fator de adaptação (interação ou inibição lateral) implementado por uma parte da estrutura de *hardware* representada na figura 6.19, onde r=0 representa a posição da célula **c**, ganhadora do proceso competivo e **r** representa a distância topológica entre a célula corrente dentro da vizinhança $N_c(t_k)$ e a ganhadora, medida no plano bidimensional. Na figura, considera-se uma vizinhança com raio dinâmico máximo de 12.

A operação é executada entre os estados S_{23}-S_{26} e ativos pela saída $\eta(j)$ da FSM ($0 \le j \le 23$).

Estado	Operação	Controle por η(j) da FSM
S_{23}	Ng FLG ← 1; Adapt CT ← No(2:0) (Dout/NLevel)	*Set Ng Flg* = '1' - η(1) (*store* Adapt CT)
S_{24}	NLevel ← NLevel + 1	*Up NLevel* = '1' - η(2)
S_{25}	estado de transição	*Sh Diff* = '0' - η(16)
S_{26}	$Diff_i$ ← Diffl ÷ 2 (*shift right*) BUS1(11:0) ← $Diff_i$	*Sh Diff* = '0' - η(16) *Dw AdaptCT* = '1' - η(12)

Tabela 6.11. Controle e Operação de atualização do vetor de pesos **w**, para o caso da célula em $N_c(t_k)$.

Na seqüência, descreve-se o procedimento ilustrado na tabela 6.11.

Passo 1 -S_{23}. Um *flag* implementado por um flip-flopD *Ng Flg* é colocado em "1", impedindo a continuação do progresso da máquina, caso outra vizinha tente ativar a célula corrente. O procedimento é necessário para controlar a ativação da célula em $N_c(t_k)$ por uma única vizinha. O mesmo sinal η(1) controla o armazenamento do bloco *Adapt CT*, que armazena a distância topológica corrente entre ela e a ganhadora, enviada pela célula vizinha ativadora.

Passo 2 -S_{24}. O valor corrente armazenado no registrador *NLevel* é incrementado em 1. Este registrador armazena a distância topológica entre a célula corrente e a ganhadora. O valor corrente é incrementado e transmitido a **todas** as células vizinhas, um sinal de controle é enviado às vizinhas sempre que elas pertençam a $N_c(t_k)$, i.e., se NLevel ≤ Raio corrente de $N_c(t_k)$.

Passo 3 -S_{25}. Estado de transição utilizado para retorno de S_{26} e voltar o sinal de controle *Sh Diff* a "0". Se o sinal de entrada à FSM, "zero adapt="0", o processo de *shift* dos registradores $Diff_i$ continua. Caso contrário, o processo de atualização de **w** continua no estado S_{12}, tal como no caso da atualização para a célula ganhadora.

Passo 4 -S_{26}. Realização da operação de deslocamento à direita (*shift right*), simultaneamente em todos os registradores *Diff*. Ao mesmo tempo, decrementa-se em 1 o valor corrente armazenado no registrador *Adapt CT*.

O registrador com decrementador *Adapt CT* é inicializado (no estado S_{23}), com o valor armazenado em *NLevel*, equivalente à distância topológica enviada pela célula vizinha. Tal valor inicial é decrementado em cada passagem pelo estado S_{26}. Desta forma, na medida que a célula corrente está mais distante da ganhadora -mas dentro de $N_c(t_k)$- os valores armazenados nos registradores $Diff_i$ são divididos por 2 de forma proporcional. A estrutura do mecanismo representa-se na figura 6.20.

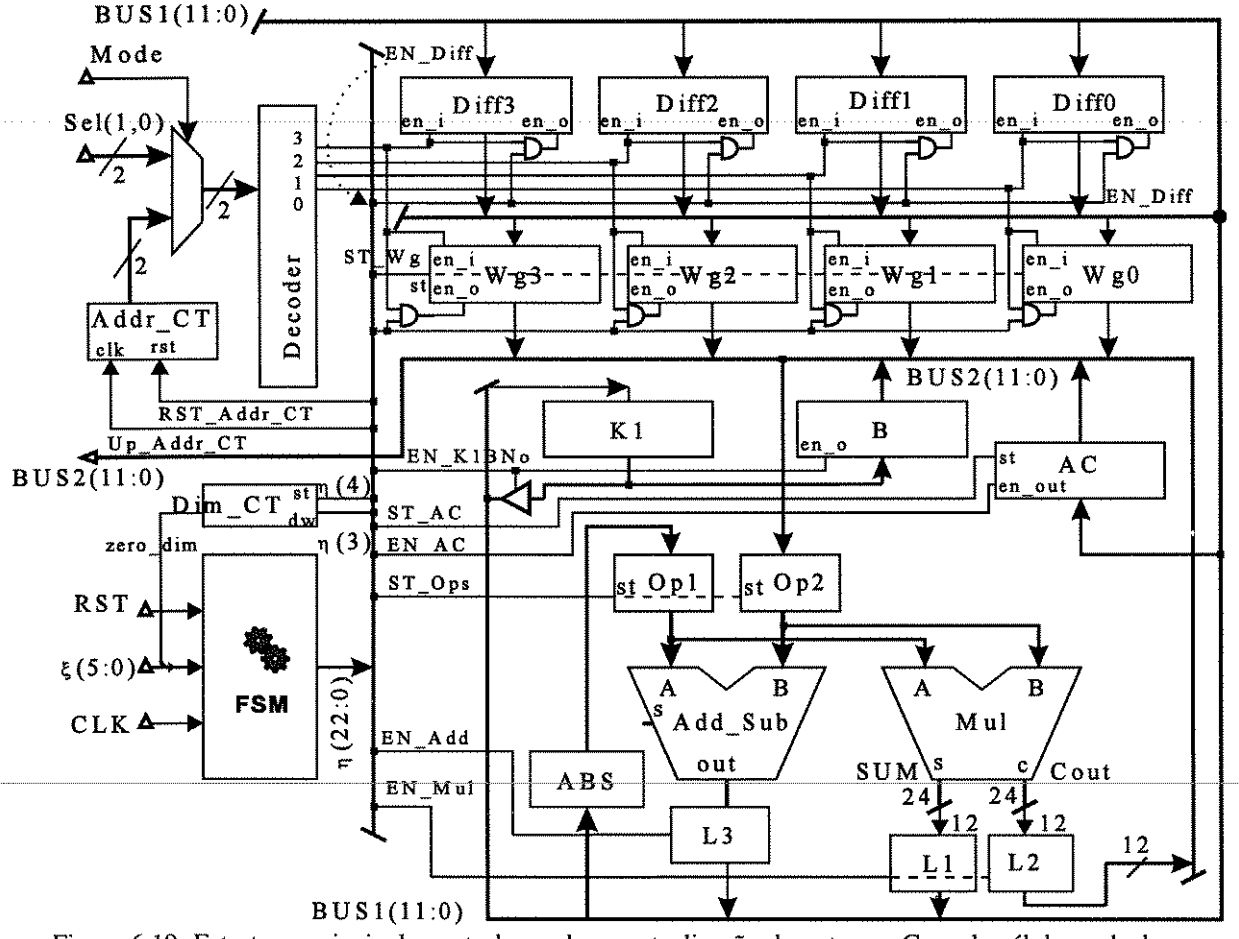


Figura 6.19. Estrutura e sinais de controle usados na atualização do vetor **w**. Caso da célula ganhadora.

Os sinais de controle da FSM detalhados na figura 6.19 relacionam-se unicamente com a operação de atualização de **w** para o caso da célula ganhadora. A estrutura adicional utilizada no caso da não ganhadora pertencente à vizinhança $N_c(t_k)$, representa-se na figura 6.20.

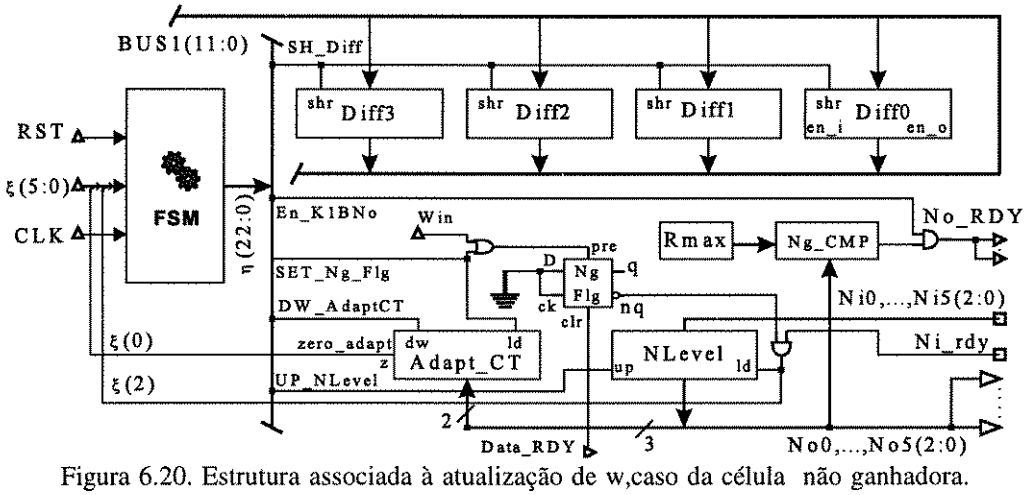


Figura 6.20. Estrutura associada à atualização de w,caso da célula não ganhadora.

6.4. Conclusão.

Neste capítulo foram descritas as estruturas principais utilizadas na operação de *Neuron*, junto com os algoritmos associados ao *hardware*. A estrutura completa de *Neuron* no seu nível topo é representado pelo diagrama esquemático da figura 6.21, incluindo-se os componentes que formam o *glue-logic* do sistema, i.e., a lógica adicional necessária para a integração dos blocos. O diagrama foi obtido diretamente da ferramenta CAD-EDA (*Electronic Design Automation*) de captura esquemática, *Design Architecture* ™ (*Mentor Graphics Falcon Framework*).

A maior parte dos blocos foi implementada mediante o uso de ferramentas de síntese a partir da modelagem em VHDL. Posteriormente, os circuitos foram otimizados por área e mapeados na tecnologia objeto disponível. No entanto, não foi possível descrever todos os blocos utilizando a metodologia *top-down* clássica descrita na secção 6.1.2. Após a fase de modelagem, síntese e otimização usando componentes pertencentes à biblioteca *Autologic*™ de *macrofunções* [76],[78], foram detectados problemas nas simulações de alguns deles, tal como no contador *Step CT* (12 bits) que armazena o passo corrente de operação do SOFM. Isto motivou a sua implementação pelo método tradicional de captura esquemática, o problema foi repetido em outros componentes, onde a versão VHDL sintetizável utilizada não conseguia inferir as estruturas de *hardware* desejadas e/ou funcionalmente corretas.

Alguns blocos foram realizados em diversas versões, utilizando até a linguagem Kiss [78] para a realização de experiências, como foi feito no caso da máquina de estados. A versão descrita em 6.3.1 foi obtida após diversas implementações, onde foi variada a forma de codificação dos estados (*Gray*, *Spectral*, *Mustang*, etc.), o tipo da máquina (Moore ou Mealey, livre ou não de *glitchs*), etc. Esta fase experimental demandou um considerável esforço, especificamente pelo nível de cuidado e detalhe com que devem ser realizadas as simulações e a verificação funcional dos circuitos.

A célula básica *Neuron* foi simulada exaustivamente em todos os níveis hierárquicos descritos neste capítulo e nos anteriores, incluindo a modelagem VHDL em alto nível do sistema composto por um arranjo de 4×4 células [89],[90]. Os resultados das simulações mais relevantes, serão descritas no capítulo 7..

No capítulo seguinte serão tratados os aspectos relacionados com a implementação física (*layout*) dos circuitos e a estrutura hierárquica dos blocos principais da arquitetura.

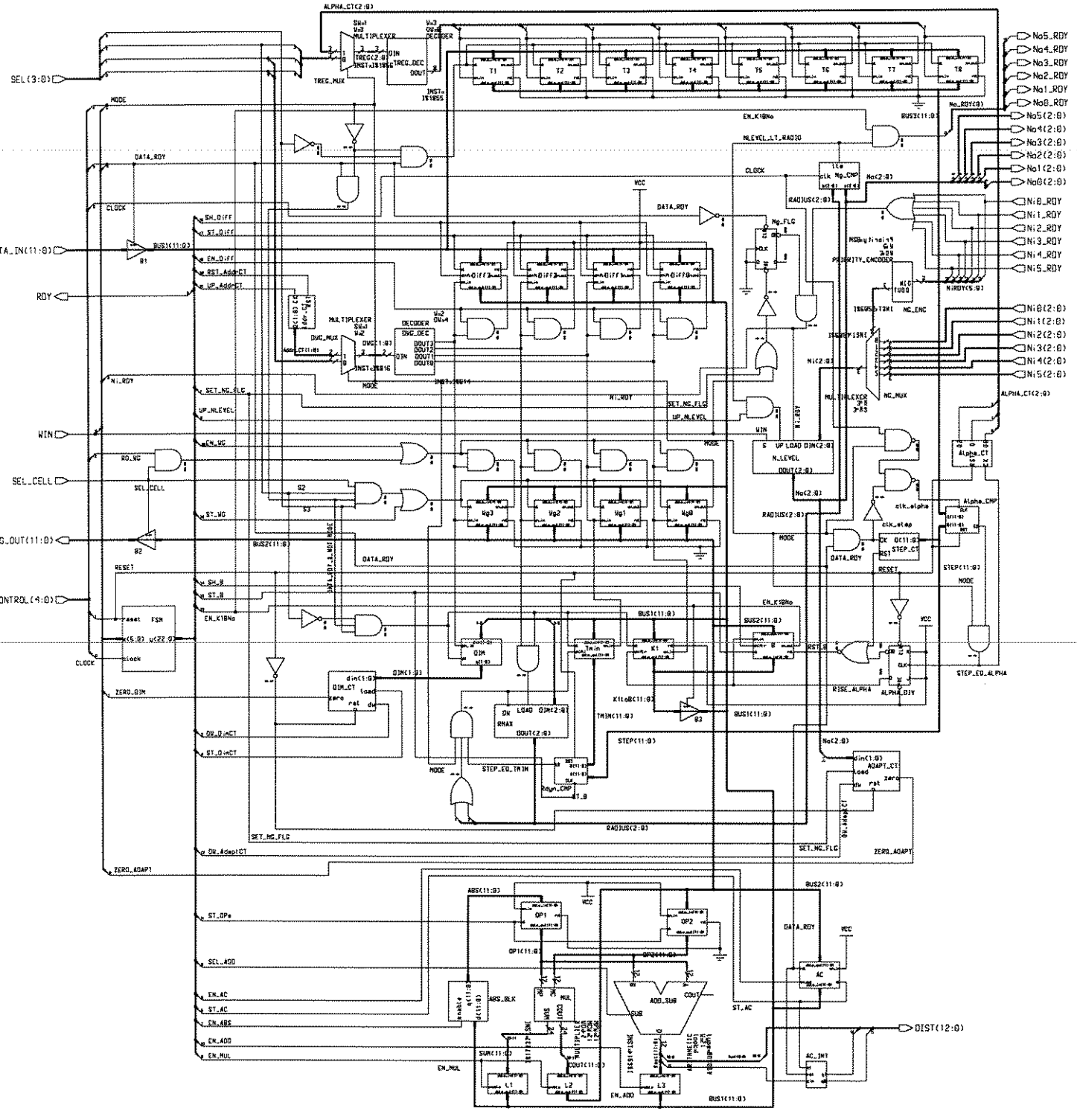


Figura 6.21. Diagrama esquemático representando a arquitetura de *Neuron*.

Capítulo 7

Implementação Física de *Neuron* e Resultados Experimentais

Escopo

Neste capítulo é descrita a implementação física do neuro-processador *Neuron*. Realizou-se inicialmente o *layout* de cada um dos componentes, para finalmente posicionar e rotear as conexões entre eles usando as funções de roteamento automático disponíveis no CAD utilizado.

Em uma primeira tentativa, foi mantida a estrutura hierárquica original, presente na fonte lógica e transmitida ao *layout*. Os blocos foram desta forma posicionados manualmente no *chip* para posteriormente conectá-los usando funções para roteamento automático (*AutoRouting*). Apesar da realização de diversas versões com esta abordagem não foram conseguidos resultados satisfatórios, com aproveitamento ineficiente da área disponível. Uma segunda abordagem foi eliminar a composição hierárquica no *layout*, mantendo-se o da fonte lógica. O resultado foi a obtenção de um *chip*, cujo *layout* apresentou muito melhor aproveitamento da área de silício. No entanto, o seu custo é a perda da visão organizada dos blocos funcionais que apresenta o *layout* onde a hierarquia é preservada. O *neuroprocessador* foi implementado na tecnologia AMS CMOS 1.2 *microns*.

Após realizada a extração de parâmetros dos circuitos, foram realizadas novas simulações e confrontadas com as medidas obtidas na fase de caracterização dos circuitos.

7.1 Introdução

Atualmente a implementação física de circuitos VLSI e sistemas eletrônicos integrados é baseada principalmente na utilização de ferramentas do tipo CAD-EDA (*Computer Aided Design-Electronic Design Automation*). Um dos tipos mais completos de ferramentas CAD/EDA são os chamados *frameworks* [71], [91], compostos por um conjunto extensivo de módulos especializados, para o projeto de circuitos e sistemas eletrônicos complexos. No caso da implementação da célula neural descrita no capítulo 6, o seu projeto foi realizado utilizando o *Falcon Framework*™ de Mentor Graphics Corp [92]. Os módulos contidos neste *framework* executam inúmeras operações para a realização das tarefas necessárias ao projeto do sistema: captura esquemática, compilação de código fonte em VHDL para o modelagem em alto nível, síntese apoiada por uma série de algoritmos de otimização (*simulated annealing*, etc.) para a realização da síntese lógica dos circuitos, inferência de estruturas de *hardware* e uso e reutilização de componentes inseridos em uma biblioteca, especialmente úteis no desenvolvimento de sistemas com uso extensivo de processamento aritmético [93], como DSPs. Na implementação física de circuitos e/ou sistemas eletrônicos complexos em silício (ou outro substrato), a realização do *layout* ,na tecnologia escolhida, é uma tarefa particularmente delicada e complexa de se realizar sem o apóio de ferramentas de automação adequadas.

Neste capítulo será descrita a implementação física das estruturas descritas no capítulo 6, incluindo o *layout* final da célula neural. Após definida a estrutura ilustrada pelo diagrama esquemático da figura 6.21, todos os circuitos foram otimizados por área e mapeados na tecnologia objeto. A sua implementação foi realizada sobre células básicas do tipo *standard-cells* [94] inclusas no *Kit-Design* da AMS [95] e construídas em tecnologia CMOS de 1.2 *microns*.

7.2. Composição Hierárquica de *Neuron* e seus Blocos Principais.

Nesta secção descreve-se a estrutura hierárquica que compõe a célula neural. Esta estrutura foi mantida durante todo o projeto da célula *Neuron*. A decomposição da célula em componentes organizados hierarquicamente permite manejar de forma mais eficiente as tarefas do projeto: modelagem em VHDL e captura esquemática→ simulação →síntese e ótimização → simulação→ *layout* e extração de parâmetros → simulação. De forma geral, o modelo de decomposição assume uma forma tal como a ilustrada na figura 7.1(a).

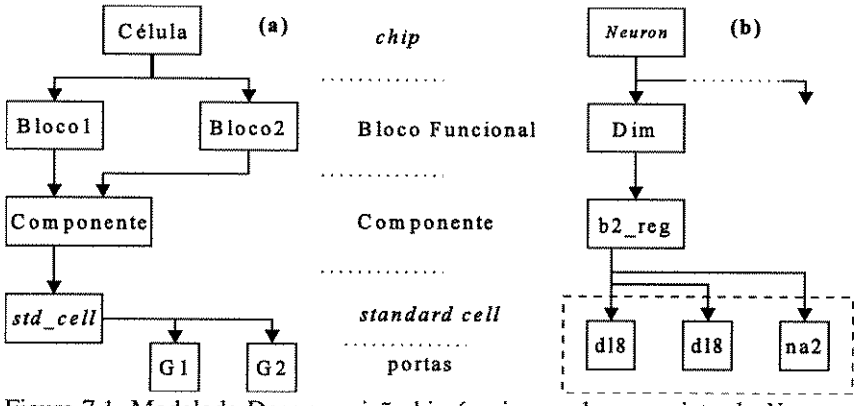


Figura 7.1. Modelode Decomposição hierárquica usado no projeto de *Neuron*.

Na figura 7.1, o nível topo representa o *chip*, logo os blocos funcionais, componentes e *std cells* (compostos por portas). Um exemplo simples desta decomposição é representado pelo registrador *Dim* (vide fig. 6.21), que armazena a dimensão d do vetor **x** e **w** ($0 \leq d \leq 3$), expandido segundo o modelo em 7.1(b).

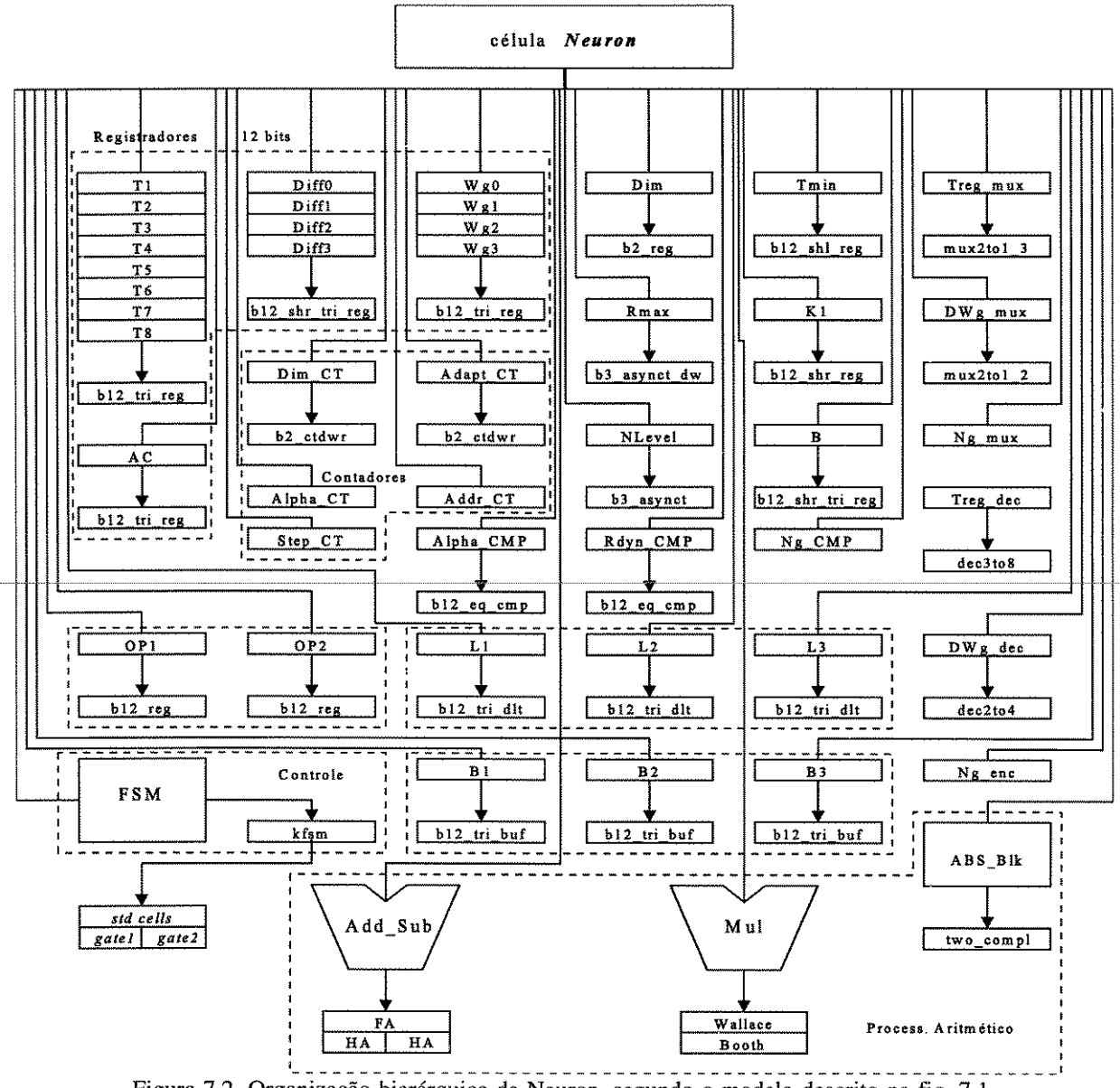


Figura 7.2. Organização hierárquica de Neuron, segundo o modelo descrito na fig. 7.1.

A composição hierárquica para os blocos funcionais mais relevantes, representa-se na figura 7.2. Para a realização física de *Neuron*, mantendo a organização hierárquica ilustrada na fig. 7.2, foi inicialmente abordada uma estratégia de implementação do *layout*, onde tentou-se manter em áreas fisicamente próximas blocos funcionalmente semelhantes, a fim de minimizar o comprimentos das trilhas de metal, para diminuir os efeitos das capacitâncias parasitas e o atraso resultante.

7.3. *Floorplanning* e posicionamento dos blocos funcionais.

7.3.1. Manutenção da composição hierárquica no *Layout.*

A primeira versão da implementação de *Neuron* foi obtida posicionando-se manualmente os blocos na área disponível do *chip*, visando minimizar a área e as linhas de comunicação e realizando a interconexão entre blocos por meio do roteamento automático a través da função *AutoRoute* da ferramenta *IC Station* [96]. Esta versão produziu um *layout* que ocupou aproximadamente 6.715 μm × 3.700 μm, i.e., uma área aproximada de 25 mm^2, sem considerar os *pads* de conexão.

Ao adicionar os *pads*, a área ocupada praticamente duplica-se, devido ao número necessário de *pads* (103, incluindo sinais de entrada/saída e alimentação) mais a área adicional ocupada para roteamento entre o núcleo da célula e seus *pads*. A figura 7.3 representa a imagem (sem *pads*) do *layout* obtido desde a ferramenta CAD, para esta primeira versão.

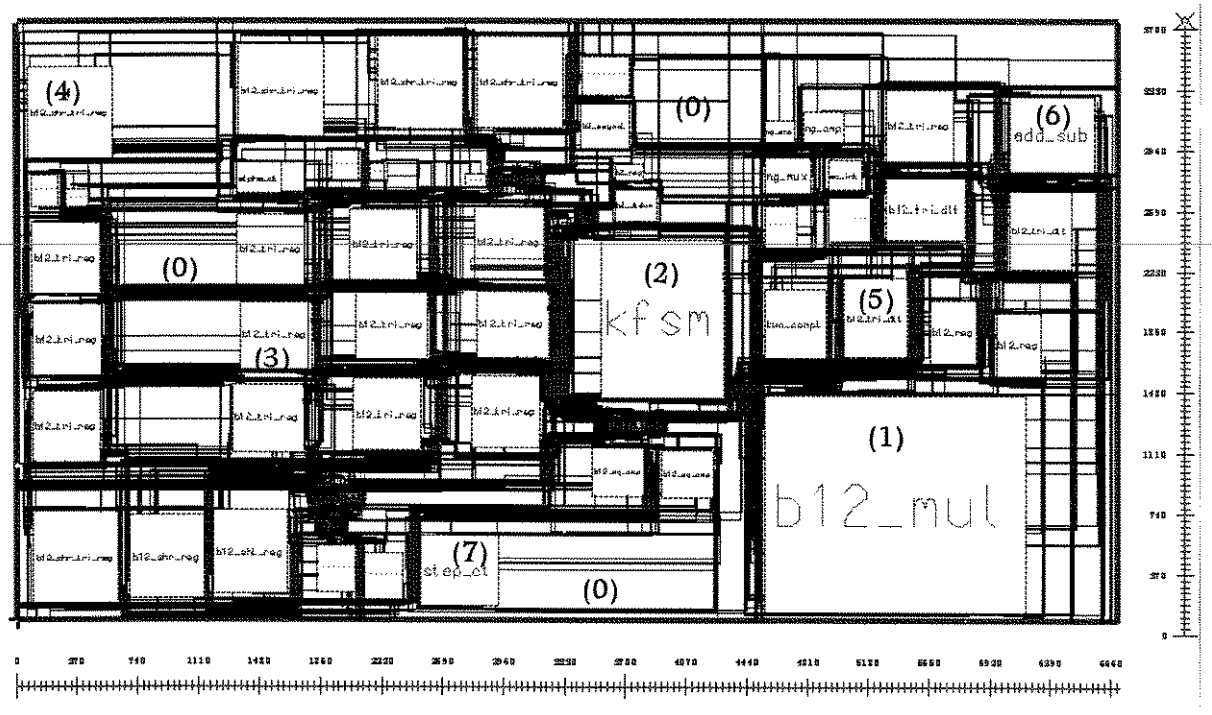


Figura 7.3. Primeira versão do layout de *Neuron*, incluindo-se a hierarquia.

Na figura anterior, observa-se a distribuição dos componentes após a execução da função de roteamento automático, realizada por *Auto Route* [96]. Observam-se vários espaços dispersos não aproveitados completamente pela ferramenta. No entanto, é possível ter uma primeira impressão da área ocupada por cada um dos blocos funcionais mais relevantes.

Os blocos funcionais correspondem aos descritos no segundo nível da composição hierárquica ilustrada na figura 7.2. O nome do componente é utilizado na base de dados da ferramenta CAD, mantendo-se a nomeação nas diferentes fases do projeto, desde a captura esquemática ou geração por síntese a partir do VHDL até o *layout*.

Label	Nome do Componente	quant	Descrição	Área ocupada de silício (μm)	%
(0)	---	-	área não ocupada por componente	---	---
(1)	b12 mul	1	Multiplicador de 12 bits	1.600 × 1.330	8.6
(2)	kfsm	1	Máquina de Estados	753 × 970	3.0
(3)	b12 tri reg	13	registrador de 12 bits com *buffer tri-state.*	415 × 440	9.6
(4)	b12 shr tri reg	5	registrador de deslocamento (dir.) de 12 bits com *buffer tri-state*	520 × 560	5.9
(5)	b12 tri dlt	3	*latch* **d** de 12 bits com *buffer tri-state*	390 × 490	2.3
(6)	add sub	1	Unidade aritmética de 12 bits de soma/subtração (compl. a 2)	520 × 460	1.0
(7)	step ct	1	Contador de nº. de passos do SOFM	470 × 440	0.8

Tabela 7.1. Componentes em *Neuron* com maior porcentagem de ocupação de área.

A Tabela 7.1 representa os componentes mais relevantes de *Neuron* (em termos do seu tamanho e funcionalidade). Por cada componente, mostra-se a área correspondente ocupada e a porcentagem de ocupação em relação à área total do núcleo do *chip*. A soma da área dos 7 componentes desta tabela representa 31% da área total do núcleo. Na Tabela 7.2 representa-se o resto dos outros componentes, não identificados na figura 7.3.

No. item	Item	Descrição	Área ocupada de silício (μm^2)	%
(1)	Tabela 7.1	Soma da área dos 7 componentes da Tabela 7.1	7.707.510	31.0
(2)	Resto dos blocos funcionais	Blocos Funcionais não identificados na Tabela 7, descritos parcialmente na figura 7.2	1.804.180	7.3
(3)	*Glue-Logic*	Lógica adicional utilizada na interconexão dos blocos funcionais e para solução da *fan-out/in*	89.250	0.4
	Total	Soma das áreas dos itens (1)-(3)	9.600.940	38.7

Tabela 7.2. Todos os componentes em *Neuron* e a sua porcentagem de ocupação de área.

Somando-se as áreas de todos os componentes da célula, obtem-se um total de 9.6 mm^2, representando menos do 40 % da área total do núcleo do *chip*. Isto significa que mais do 60% da área total do núcleo é utilizada para interconexão dos blocos e em silício não aproveitado. Foram realizadas diversas variantes do *layout* ilustrado na fig. 7.3, mas não foram obtidos resultados satisfatórios: a área não aproveitada foi reduzida em um factor pouco significante. Decidiu-se então eliminar a hierarquia realizando uma operação de *flatting* [96] no *layout* final de *Neuron*. Esta versão *flat* é descrita a seguir.

7.3.2. Eliminação da hierarquia no *Layout.*

Na ferramenta de projeto utilizada [96], uma célula constitui o elemento de projeto básico, sendo uma estrutura independente, que pode ou não possuir hierarquia. O conceito pode ser visualizado na figura 7.4, onde uma estrutura A-B é formada por 3 níveis hierárquicos : L0, L1 e L2. A composição da célula C em L1 é visível unicamente no nível L2, descendo na estrutura hierárquica do bloco A em L0. Na variante *flat* do projeto de *layout* de uma célula, a estrutura de C seria diretamente visualizada no nível topo único L0.

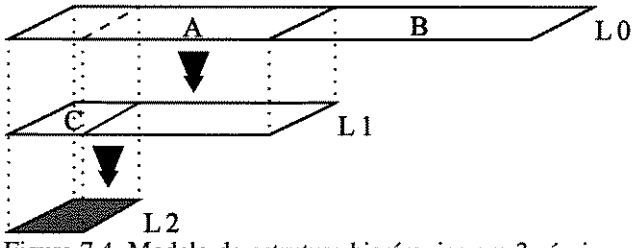


Figura 7.4. Modelo de estrutura hierárquica em 3 níveis no projeto do *layout* do bloco A-B.

Na versão descrita em 7.3.1, a estrutura hierárquica de *Neuron*, usada como fonte lógica para a realização do *layout*, manteve-se inalterada, resultando em baixa taxa de aproveitamento da área disponível. Na versão *flattened*, a hierarquia existente na fonte lógica do circuito foi mantida, mas decidiu-se pela sua eliminação ao construir os elementos básicos (células) na realização do *layout*. Os blocos presentes na versão anterior foram desta forma eliminados. A estrutura final foi compactada, formando-se células independentes que o roteador conectou com maior flexibilidade, o custo é a perda da visualização organizada que permite a estrutura hierarquizada (vide figura 7.5).

Na figura 7.5 representa-se o *layout* para a nova versão de *Neuron*, incluindo-se os *pads* de proteção que permitem a interconexão do *chip* com o exterior. A área total ocupada, com os *pads* (implementados por *standard cells* AMS [94] IB15, OB33), é de 4,75 mm × 5,00 mm = 23,75 mm^2, o *chip* contem um número de gates equivalentes aproximado de 5000.

Nesta versão, a área correspondente ao núcleo resultou em 11.244.816 μm^2, i.e., aprox. 11,25 mm^2. Se considerarmos os mesmos dados da versão anterior para os blocos funcionais de *Neuron*, pode-se verificar que a soma total da área dos blocos mais os componentes de lógica adicional (total=9,6 mm^2), representa agora aproximadamente 85% do núcleo. Por tanto, a área dedicada à interconexão (mais a área de silício não aproveitada), foi reduzida desde 60%, na primeira versão, para aprox.15%, nesta última.

Deve-se levar em conta que a ferramenta toma como ponto de partida as fontes lógicas e o *layout* original de cada bloco funcional, desagregando-o e recolocando-o na área disponível. Não existe necessariamente o *floorplanning* otimizado por blocos funcionais e um mesmo bloco pode ser desagregado em diferentes regiões do *chip*.

Área do *chip* e custo total (fabricação + encapsulamento) : 4.75 x 5.0 mm^2 (incluindo *PADs*) = 1600 Francos F./ mm^2 x 23.75 mm^2 : Francos F. 38.000 (aprox. US$ 8000).

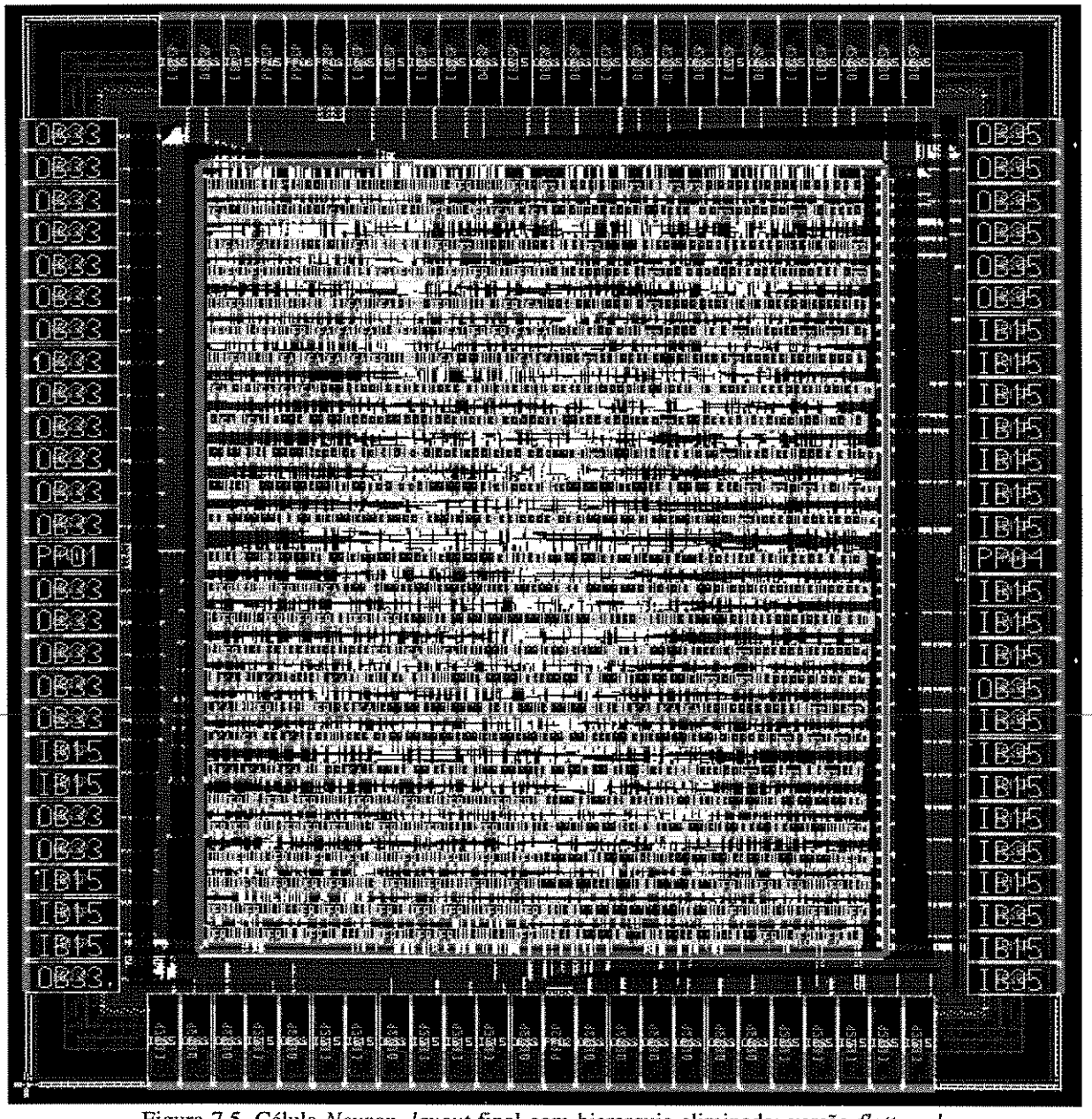
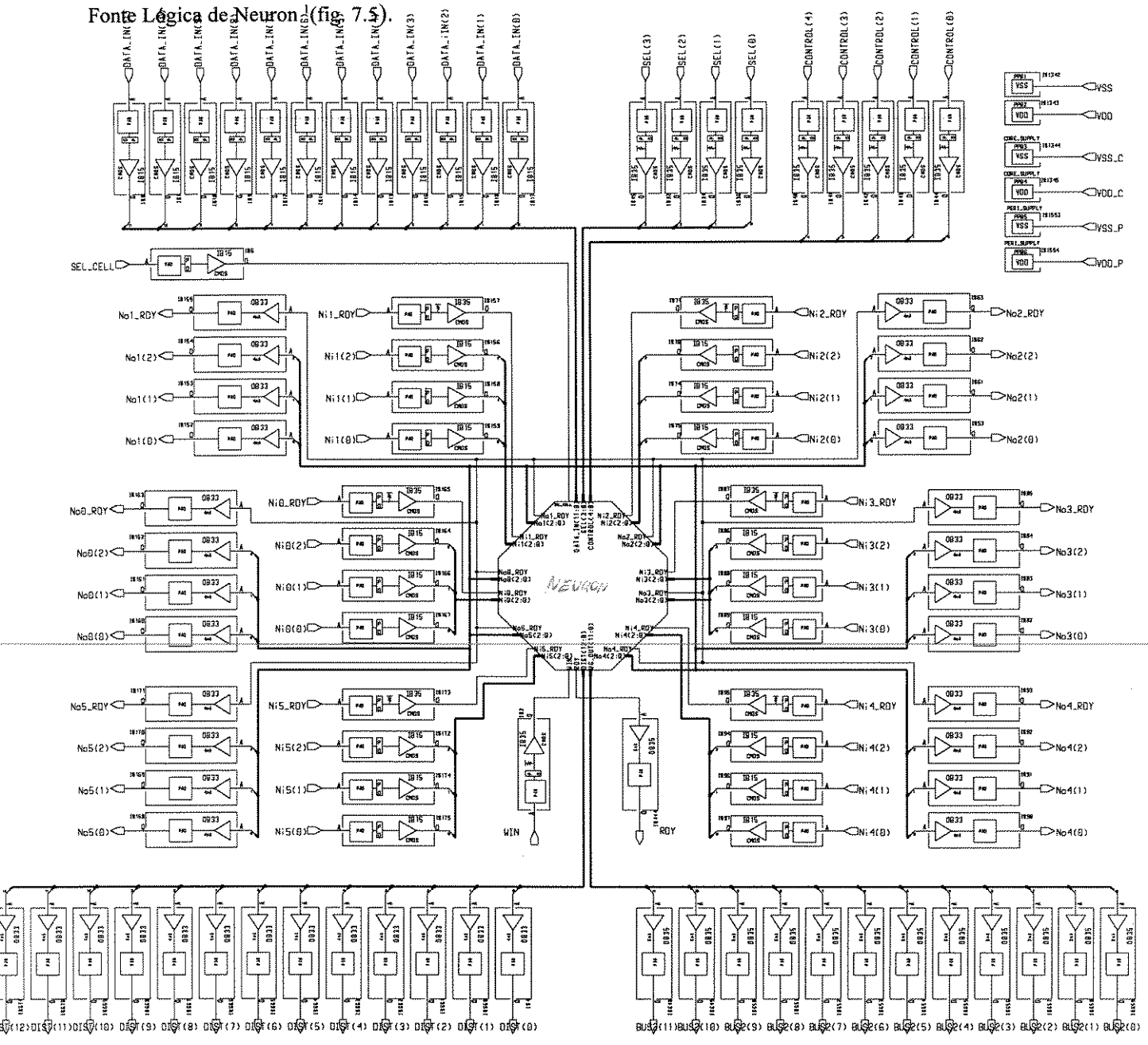


Figura 7.5. Célula *Neuron, layout* final com hierarquia eliminada: versão *flattened.*

Sobre esta última versão da implementação de *Neuron* foi realizada uma extração de parâmetros incorporando as capacitâncias parasitas das linhas de comunicação e outras fontes de atraso. O diagrama esquemático representando o circuito utilizado como fonte lógica ao *layout* da fig. 7.5 é representado na fig. 7.6.

Fonte Lógica de Neuron[1] (fig. 7.5).



[1] Figura 7.6. Diagrama Esquemático utilizado como fonte lógica do *layout* na fig. 7.5.

7.4. Simulações.

Serão apresentados unicamente os resultados mais relevantes, relativos aos ciclos 1, 2 e 3 da FSM descrita no capítulo 6. Apresentar-se-ão também as formas de onda relativas ao elemento básico que implementa o bloco WTA.

Para verificar a operação do algoritmo serão utilizados como dados de entrada os apresentados na tabela 7.3, utilizando como modelo a Tabela 6.2. A freqüência de operação do relógio é de 40 ns (25 Mhz).

Parâm.	T1	T2	T3	T4	T5	T6	T7	T8
Addr.	0	1	2	3	4	5	6	7
@t (ns)	120	160	200	240	280	320	360	400
Dado	0C0	1CE	28E	39C	45C	56A	62A	738
Parâm.	K1	Tmin	Rmax	Dim	Wg0	Wg1	Wg2	Wg3
Addr.	8	9	A	B	C	D	E	F
@t (ns)	440	480	520	560	600	640	680	720
Dado	400	03E	006	002	6C6	0A2	0	0

Tabela 7.3. Parâmetros programados externamente no estado inicial S_0.

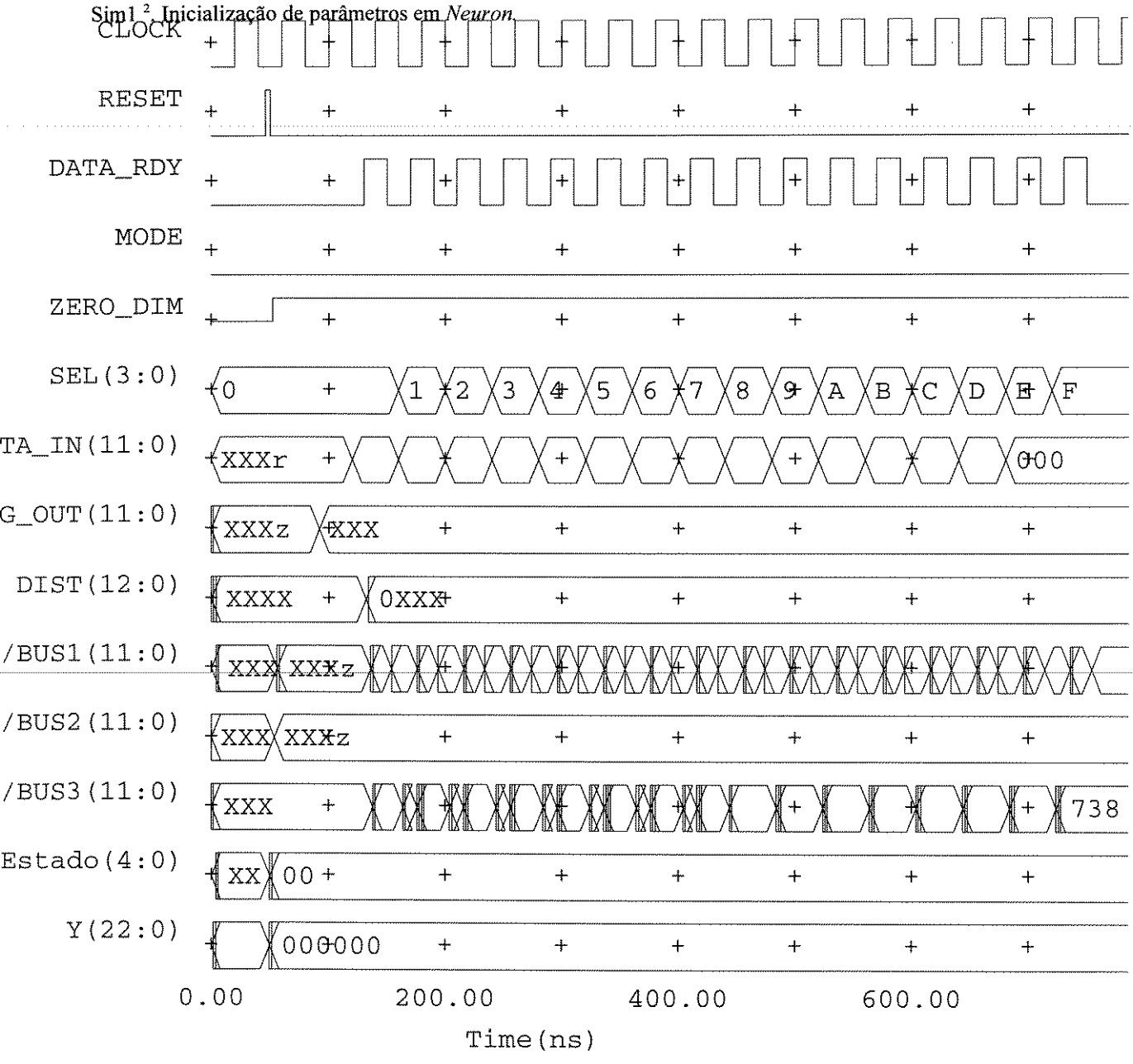
Os parâmetros da Tabela 7.2, representados em hexadecimal, permitem a execução de SOFM para vetores **x** e **w** de dimensão 2. Em todos os diagramas de tempo mostrados a seguir, os estados da FSM são representados em *código Gray*, da forma apresentada na Tabela 7.4.

Estado	S_0	S_1	S_2	S_3	S_4	S_5	S_6
c. Gray	00	10	18	08	0C	1C	14
Estado	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}
c. Gray	04	06	16	1E	0E	0A	1A
Estado	S_{14}	S_{15}	S_{16}	S_{17}	S_{18}	S_{19}	S_{20}
c. Gray	12	02	03	13	1B	0B	0F
Estado	S_{21}	S_{22}	S_{23}	S_{24}	S_{25}	S_{26}	
c. Gray	1F	17	07	05	15	1D	

Tabela 7.4. Codificação Gray para os estados da Máquina de Estados - FSM.

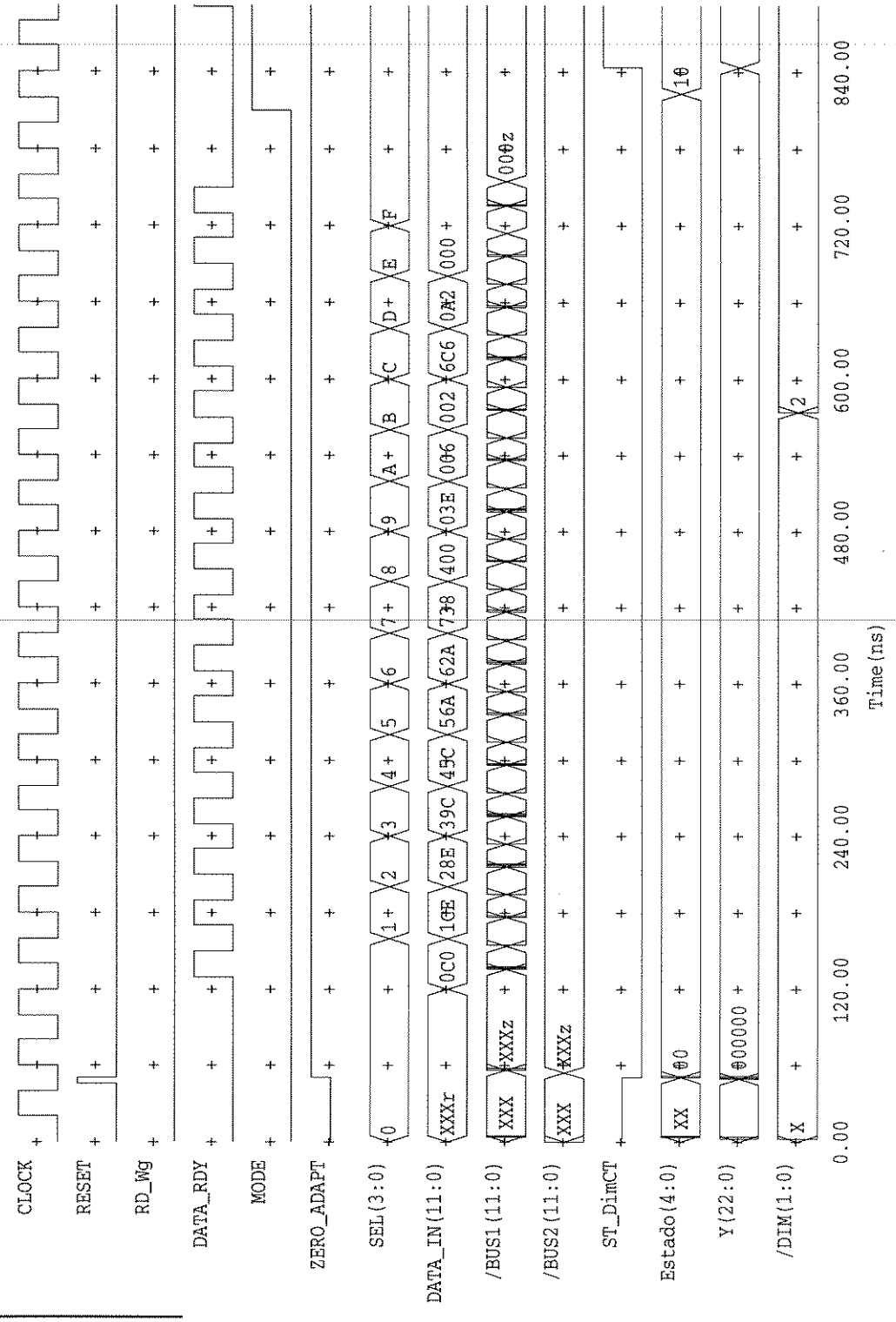
Os dados são apresentados em hexadecimal, utilizando a representação numérica da fig. 6.4.

Sim1 [2]. Inicialização de parâmetros em *Neuron*.

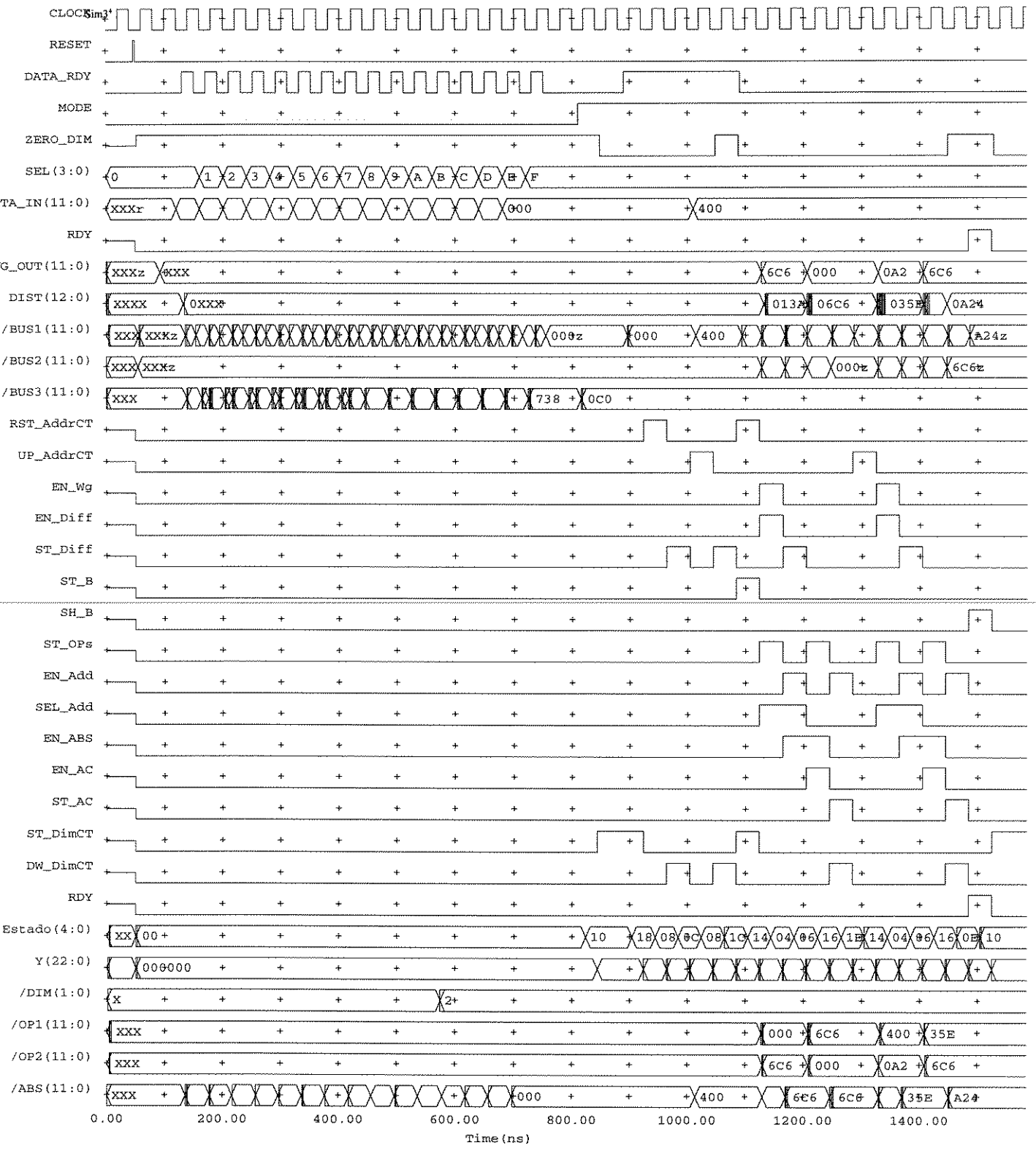


[2] Fig. 7.7. Diagrama de tempo da carga de parâmetros para inicialização de *Neuron*.

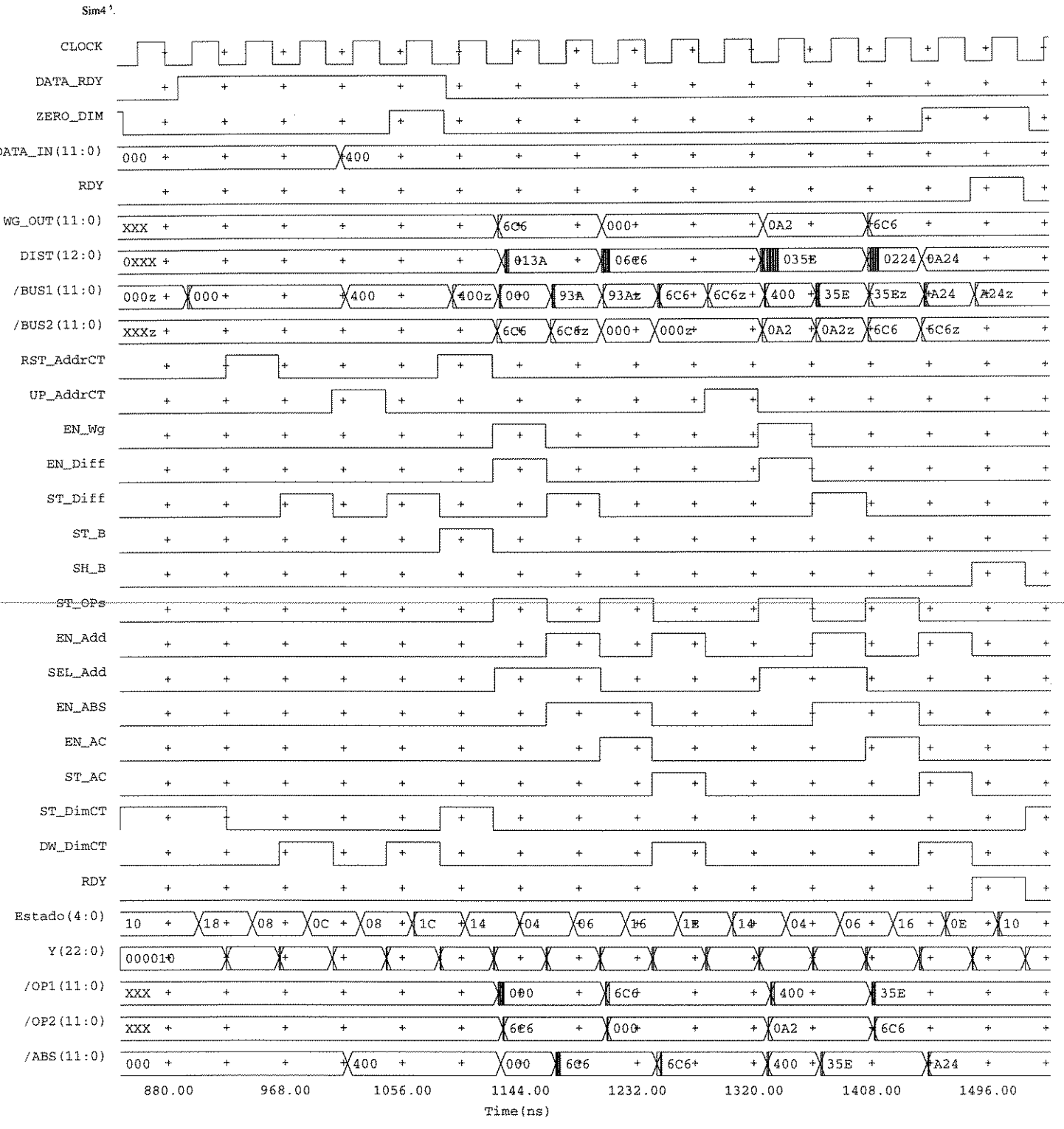
Sim 2[3].



[3] Fig. 7.8. Inicialização de *Neuron*, com dados ingressados pelo barramento *Data_In*.

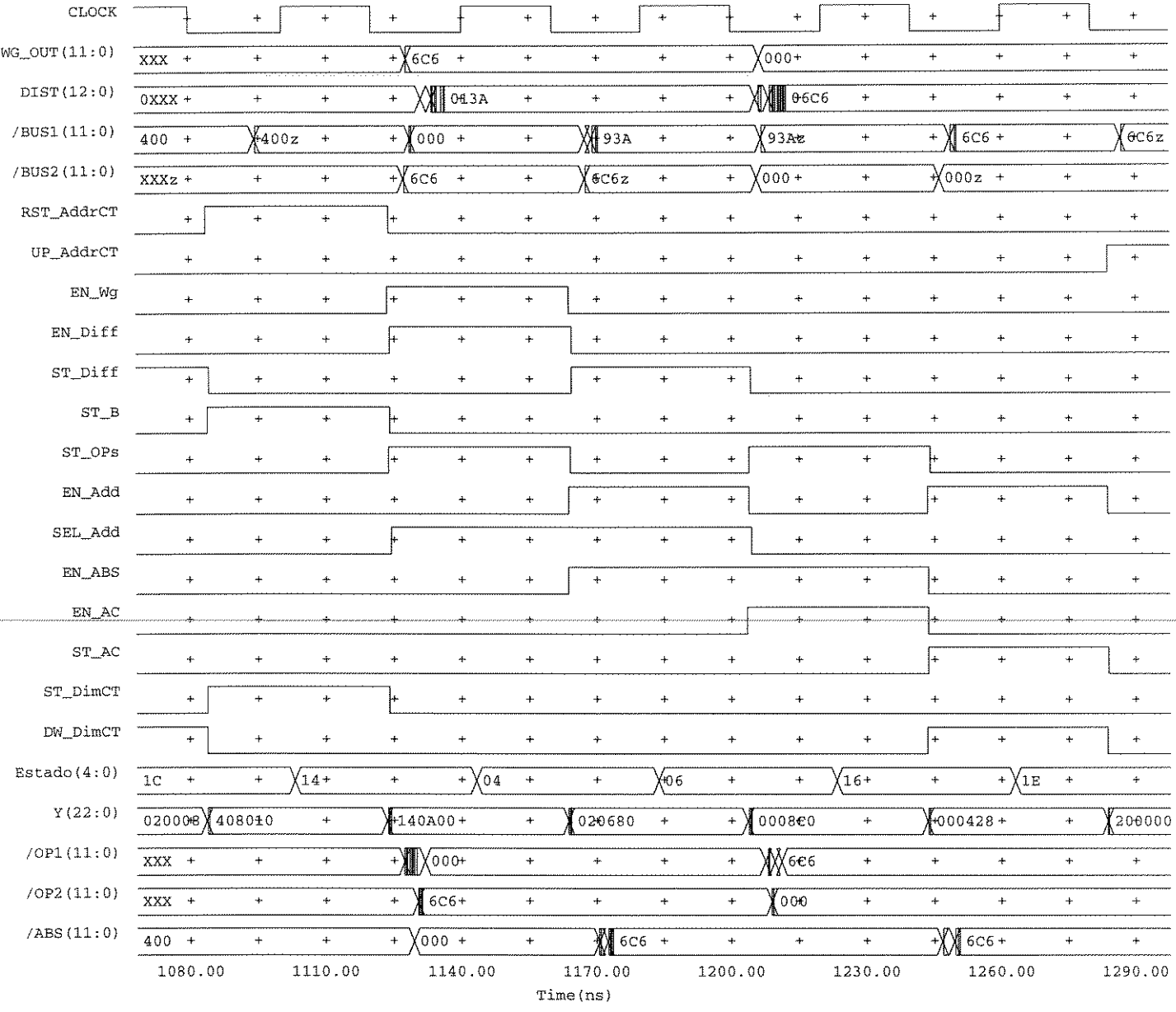


[4] Fig. 7.9. Carga de parâmetros e Cálculo da distância Manhattan.



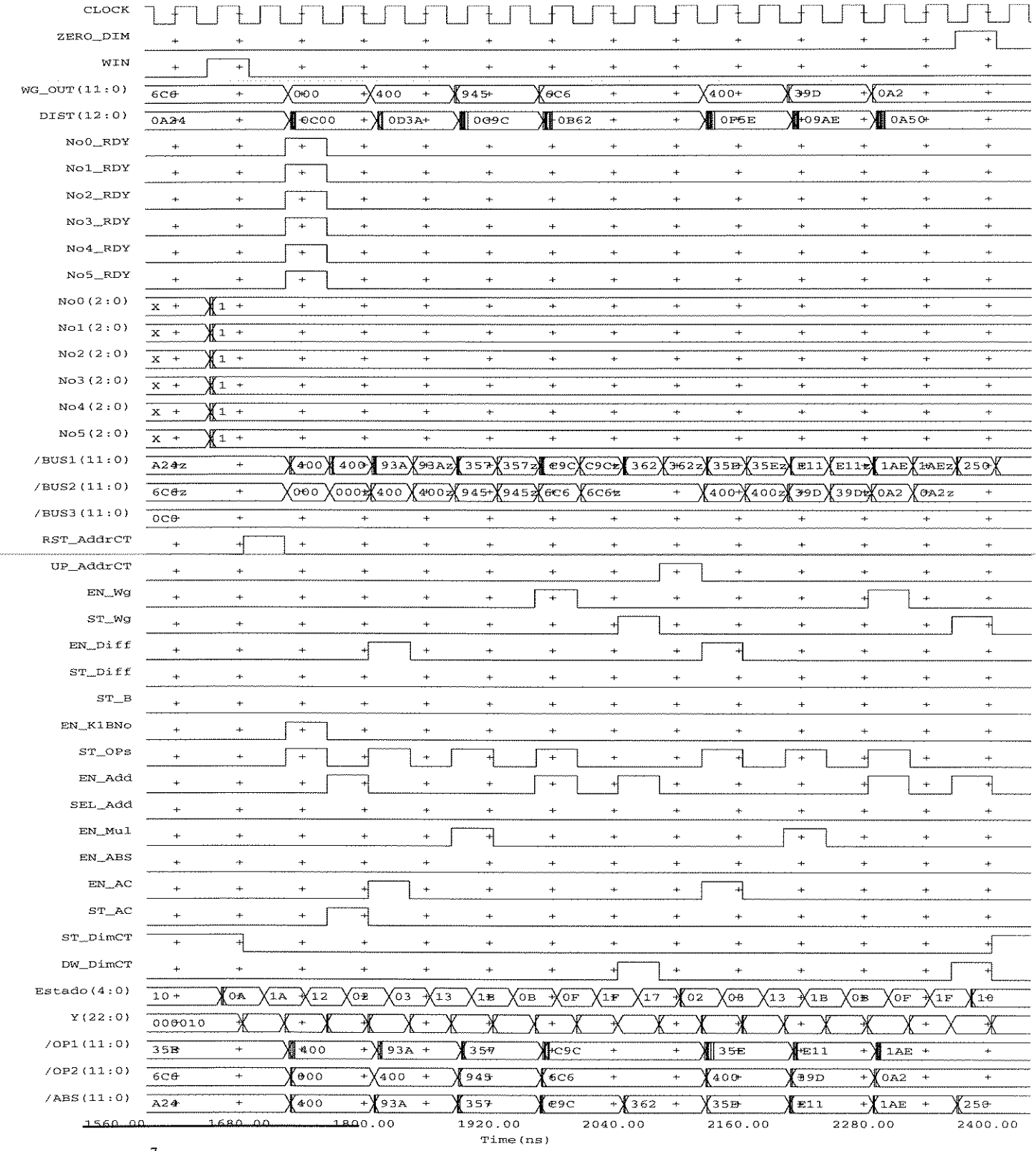
[5] Fig. 7.10. Operação de cálculo da distância Manhattan. Estados S_1-S_{11}.

Sim5 [6]. Detalhe na op. de cal. de dist.



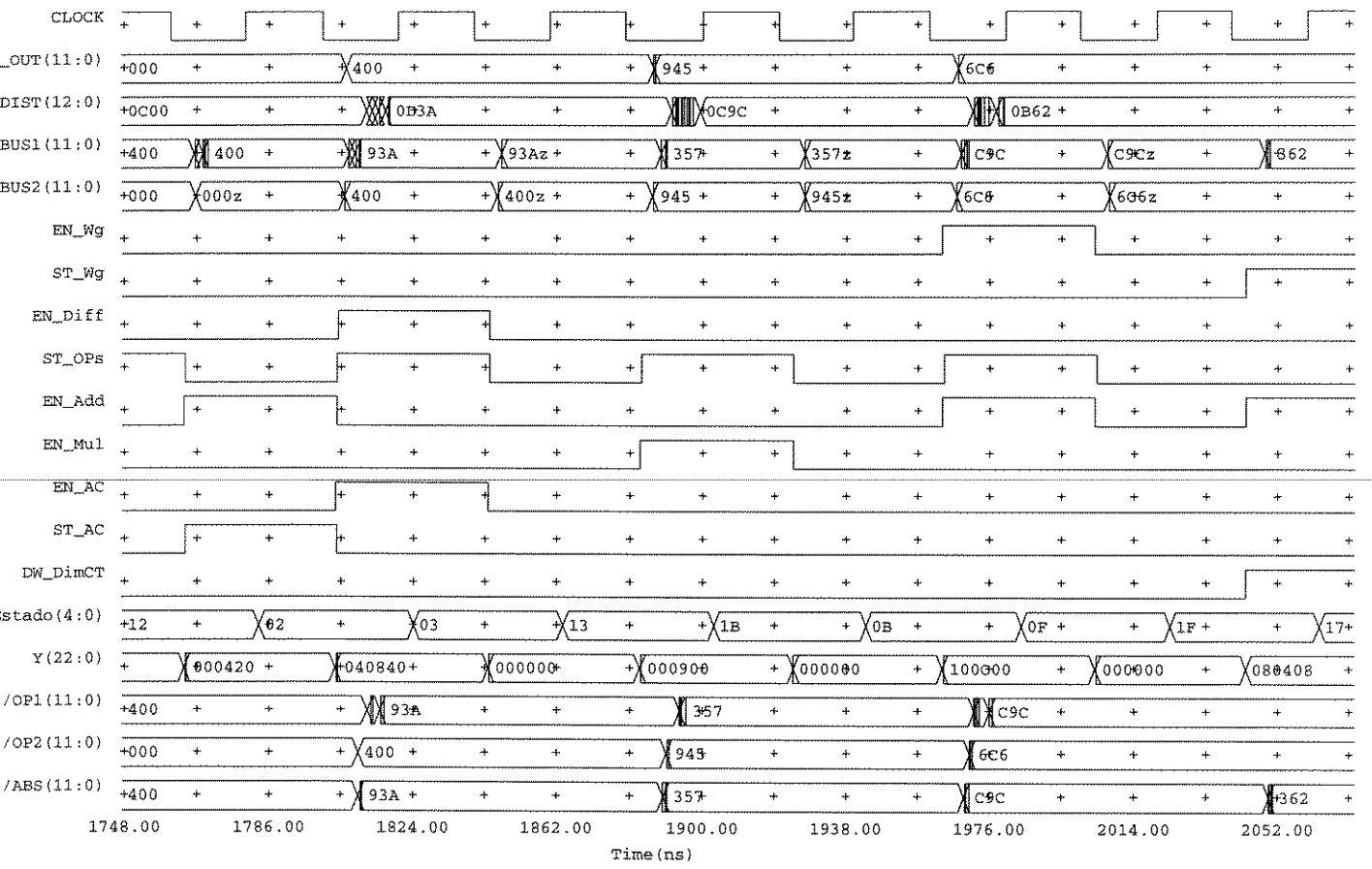
[6] Fig. 7.11. Detalhe na operação de cálculo da distância Manhattan.

Sim6 [7]. At. do vetor **w**.

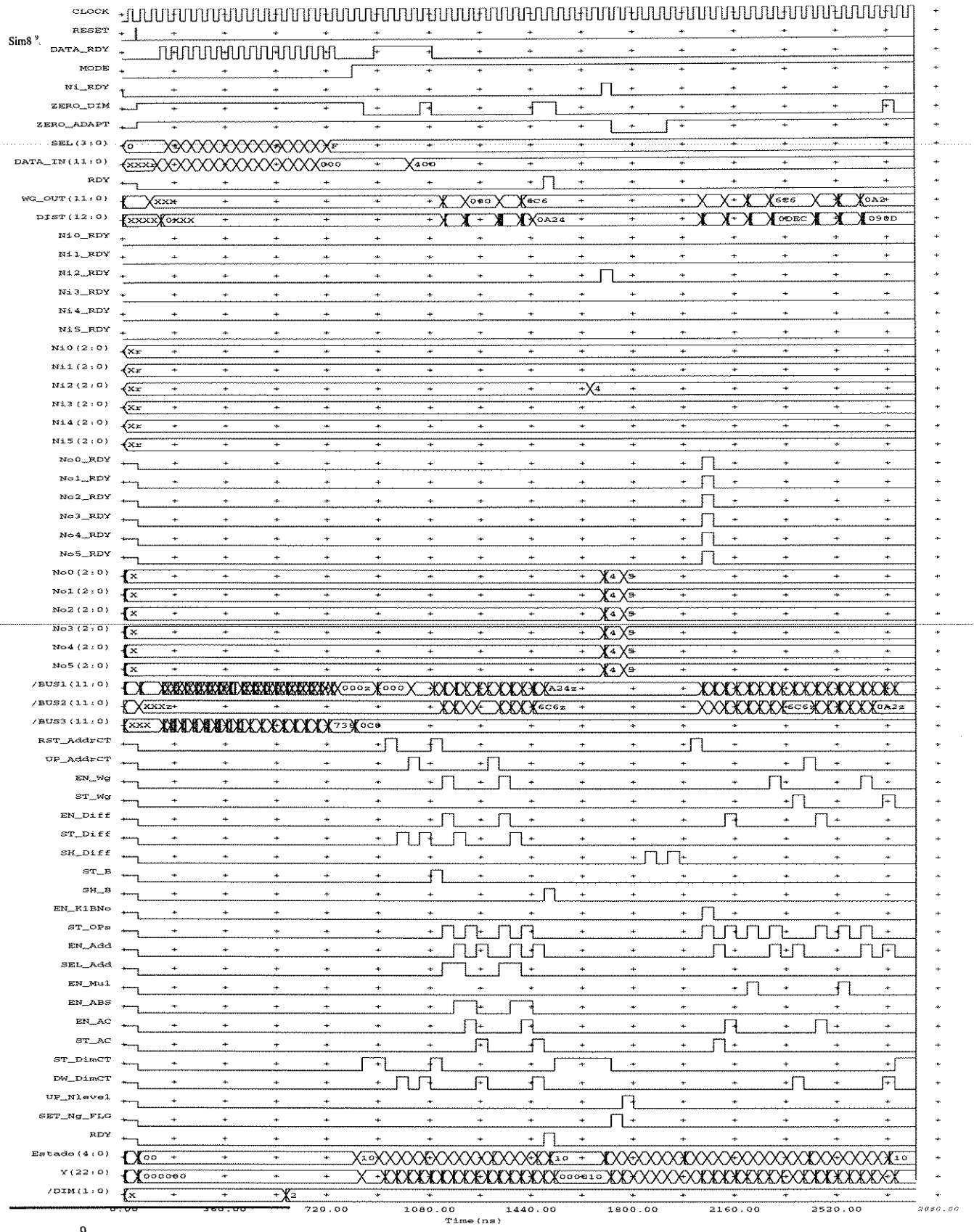


[7] Fig. 7.12. Operação de atualização de pesos, no vetor $\mathbf{w} = [w_0 w_1]$. Caso da célula ganhadora..

Sim7 [8]. Det. na at. de **w**.

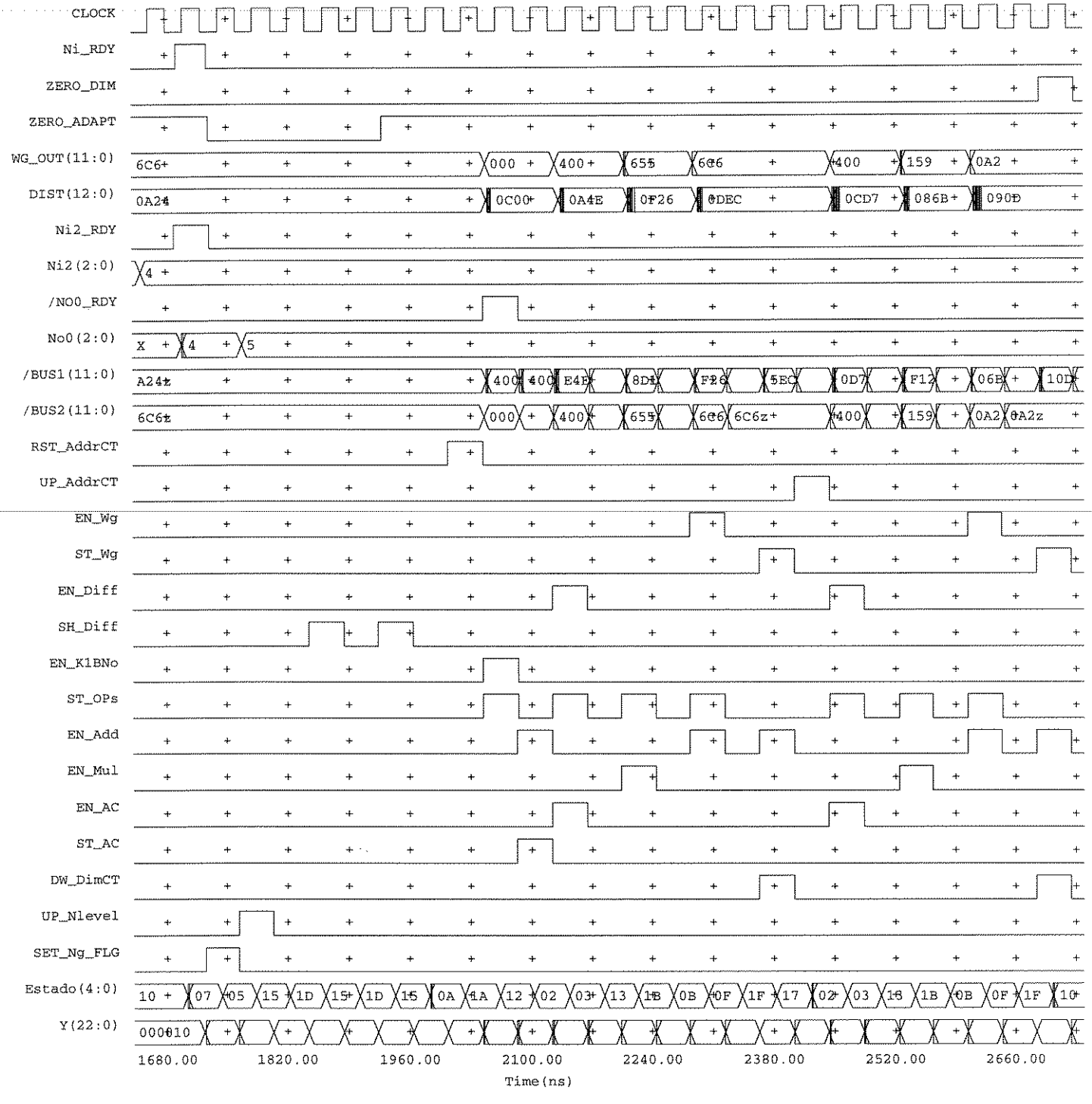


[8] Fig. 7.13. Detalhe na atualização de **w**. Estados S_{14}-S_{21}.



[9] Fig. 7.14. Carga de *Neuron* e atualização de pesos para a célula não ganhadora.

Sim9[10]. At. de w, para c não ganh.

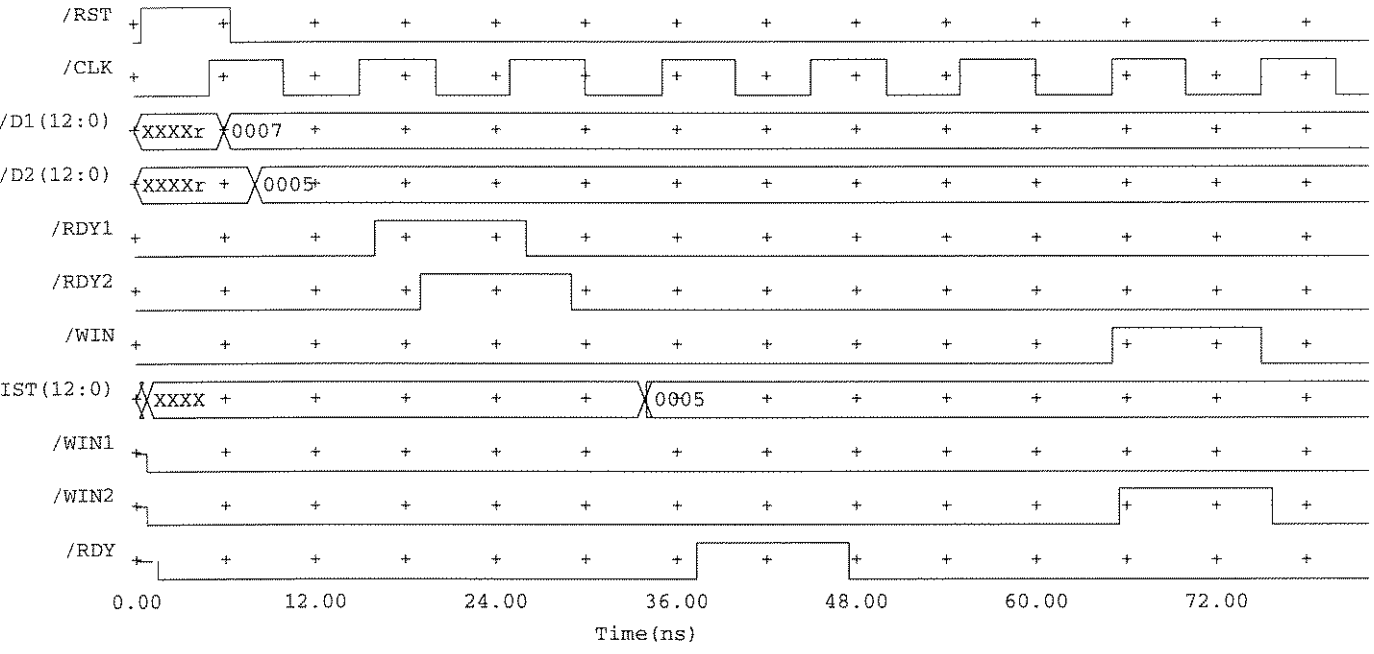


[10] Fig. 7.15. Atualização de pesos para a célula não ganhadora em $N_c(t_k)$. Estados S_{23}-S_{21}.

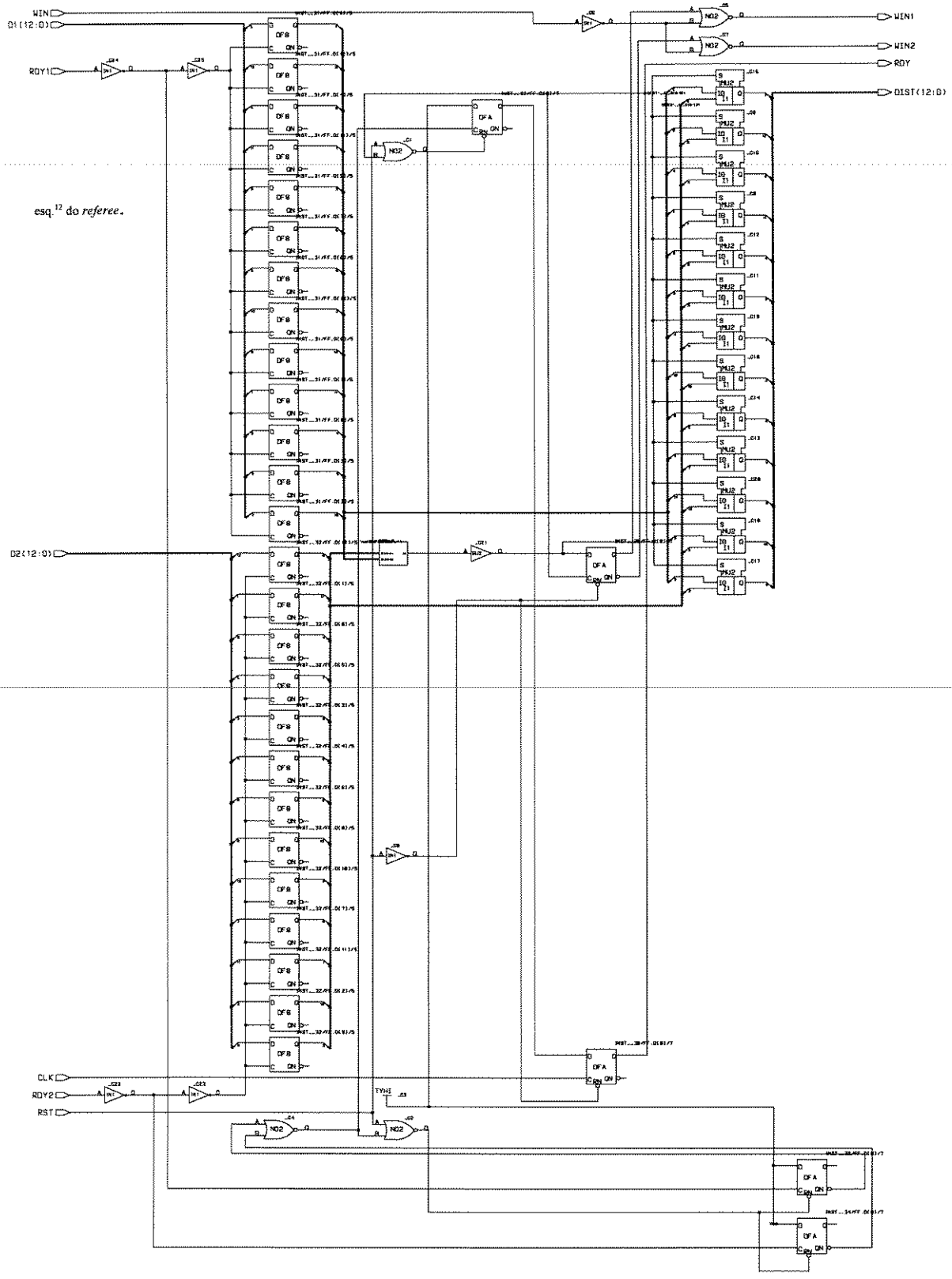
7.5. O componente *Referee* do bloco WTA.

Este componente recebe os valores das distâncias de 2 elementos D1(12:0) e D2(12:0) e coloca na sua saída Dist(12:0) o valor menor entre D1 e D2, armazenando em um *bit* qual foi entrada de valor menor. Ao receber posteriormente um sinal pela entrada *Win*, desde outro elemento no nível inferior (vide árvore de operação do bloco WTA em capítulo 5 e anteriores), o *referee* envia um sinal pela saída de valor menor: Win1 ou Win2. O sinal de *Winner* é desta forma transmitido até a célula ganhadora da rede de N × N células. O resultado da simulação deste componente apresenta-se na figura 7.16[11]. O circuito resultante após a síntese e mapeamento nas *standard cells* de AMS, representa-se na figura 7.17.

Área	5,4 mm² (5.395.420,1 µm²), incluindo *pads* de conexão
Freqüencia de operação	20 MHz
Tecnologia	AMS CMOS 0,8 µm, 2 níveis de metal
custo	1.600 Francos Fr./mm² × 5,4 mm² = Fr. fr. 8.640,0



[11] Figura 7.16. Operação do componente *referee*, do bloco WTA.



esq.[12] do *referee*.

[12] Figura 7.17. Esquemático do *referee*, obtido após a síntese/otimização e mapeamento à tecnologia.

7.6. Caracterização de *Neuron* e *Referee.*

7.6.1. Descriçao do procedimento de teste.

O procedimento para a caracterização e verificação dos circuitos fabricados foi a geração de padrões de teste e a análise lógica das respostas aos vetores de teste. Todas as medidas experimentais foram realizadas com o módulo analisador lógico e o gerador de padrões do sistema de análise lógica HP 16500A, de Hewlett-Packard™.

As entradas e saídas do chip foram monitoradas e alimentadas pelo gerador de padrões, acorde com o seguinte esquema de conexões :

SEL (3:0)				Data In (11:0)											
B3 (3)	B3 (2)	B3 (1)	B3 (0)	B2 (7)	B2 (6)	B2 (5)	B2 (4)	B2 (3)	B2 (2)	B2 (1)	B2 (0)	A6 (7)	A6 (6)	A6 (5)	A6 (4)
P1 (3)	P1 (2)	P1 (1)	P1 (0)	P1 (15)	P1 (14)	P1 (13)	P1 (12)	P1 (11)	P1 (10)	P1 (9)	P1 (8)	P1 (7)	P1 (6)	P1 (5)	P1 (4)
Control (0:4)				SC - Ni RDY (0:5)								- Ni2(2:0)			- Win
A6(3)	A6(2)	A6(1)	A6 (0)	A5(7)	A5 (6)	A5 (5)	A5 (4)	A5 (3)	A5 (2)	A5 (1)	A5 (0)	A4 (2)	A4 (1)	A4 (0)	A4 (3)
P2 (0)	P2 (1)	P2 (2)	P2 (3)	P2 (4)	P2 (5)	P2 (6)	P2 (7)	P2 (8)	P2 (9)	P2 (10)	P2 (11)	P2 (14)	P2 (13)	P2 (12)	P2 (15)
Wg Out (11:0) - BUS2											- No0 RDY				
P3 (11)	P3 (10)	P3 (9)	P3 (8)	P3 (7)	P3 (6)	P3 (5)	P3 (4)	P3 (3)	P3 (2)	P3 (1)	P3 (0)	P3 (14)	P3 (13)	P3 (12)	P3 (15)
Data In (11:0)															
P4 (12)	P4 (11)	P4 (10)	P4 (9)	P4 (8)	P4 (7)	P4 (6)	P4 (5)	P4 (4)	P4 (3)	P4 (2)	P4 (1)	P4 (0)	P4 (13)		

Tabela 7.5.Correspondência entre sinais do chip *Neuron* e as entradas/saídas do sistema HP 16500 A.

Na tabela 7.5, B3(3:0), B2(7:0), A6(7:0), A5(7:0), A4(7:0) correspondem aos conectores das saídas do gerador de padrões HP, cada uma de 8 bits. As entradas ao analisador lógico realizam-se através dos conectores (*pods*) da 16 bits: P1(15:0), P2(15:0), P3(15:0) e P4(15:0).

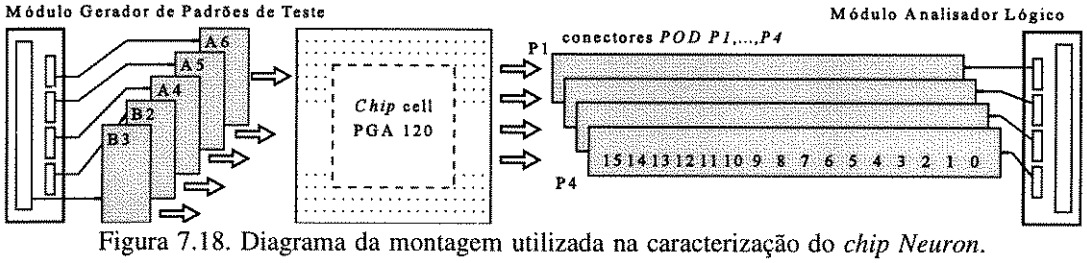


Figura 7.18. Diagrama da montagem utilizada na caracterização do *chip Neuron*.

Na figura 7.18 ilustra-se o esquema utilizado na caracterização dos chips *Neuron*, foram avaliadas 5 versões encapsuladas com PGA120 tipo 13 × 13 (matriz de 13 × 13 pinos). Foram construidas 2 placas de circuito impresso (PCBs) para a realização dos testes : uma para o *chip Neuron*, outra para o *chip* do bloco WTA. Tanto para *Neuron* quanto para o WTA, a freqüência máxima permitida pela montagem foi de 20 MHz, freqüência até onde as amostras medidas responderam perfeitamente. Acima desta frequência de operação não foi possível realizar medidas confiáveis, por causa do uso de circuitos inversores anexos utilizados para a inversão do *clock*, ocasionando-se formas de ondas com excesso de distorção nas entradas do *chip*. A freq. máxima de operação do equipamento utilizado é de 50 MHz.

7.6.2. Metodologia.

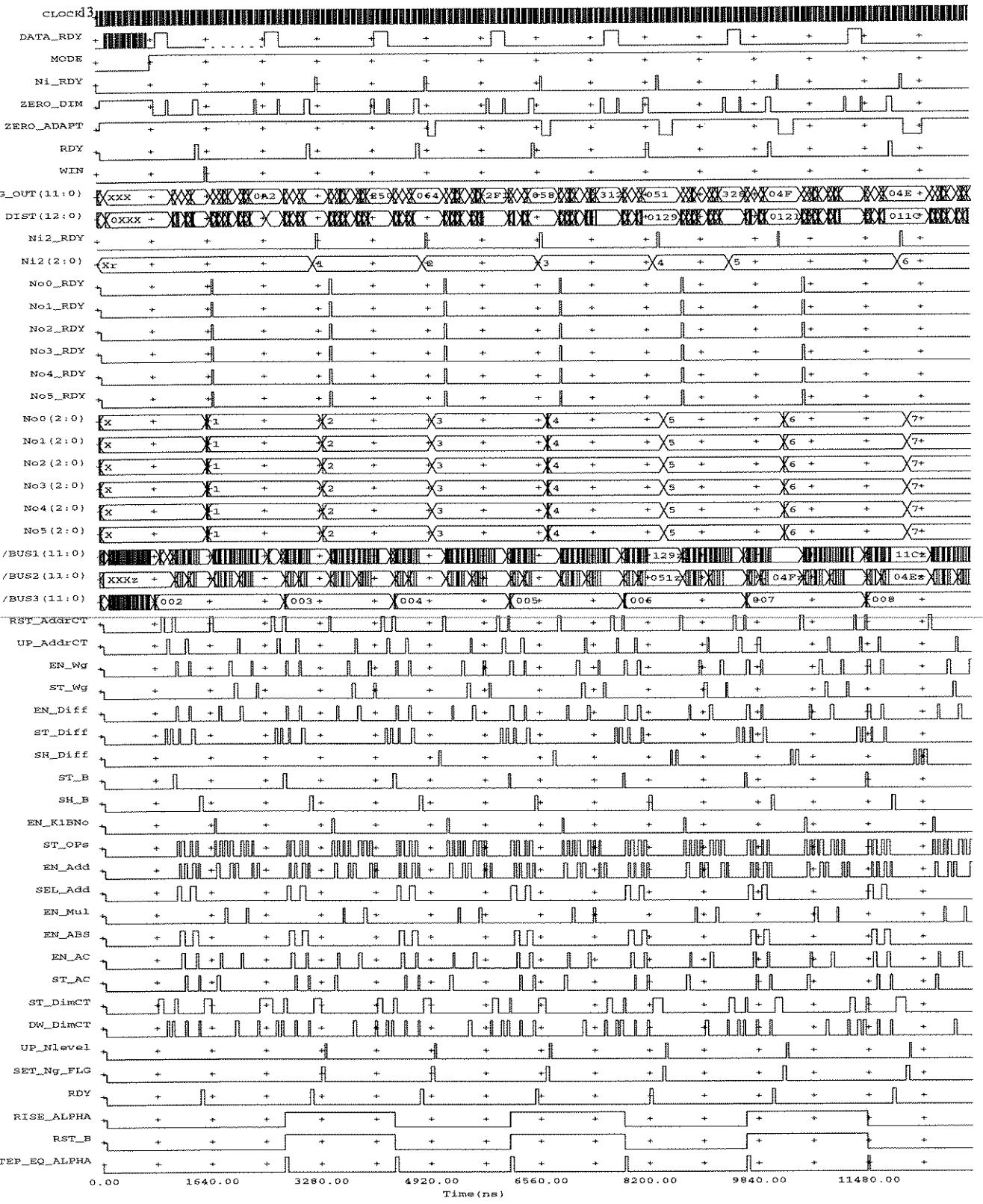
Para ambos os *chips*, a metodologia de testabilidade foi baseada na geração de padrões de teste por meio do módulo gerador de padrões HP 16520A, utilizando os mesmos valores dos vetores de treinamento das simulações realizadas com o *QuickSim* e analisando-se as saídas dos *chips* com o módulo analisador lógico HP 16510A (vide fig. 7.18) do sistema de análise lógica HP 16500A. Os primeiros testes visaram validar a funcionalidade de *Neuron* e do WTA, comparando diretamente as respostas aos padrões de teste com os resultados das simulações.

7.6.3. Resultados.

Os resultados de simulações e medidas em *Neuron* apresentam-se a seguir. Na figura 7.19 e na tabela 7.6, apresenta-se o resultado da simulação para o caso da célula ser ativada por várias vizinhas. A célula corrente localiza-se a distintas distâncias da ganhadora, os resultados destas simulações foram confirmados pelas medidas do analisador lógico.

Tabela 7.6

Tempo	Wg0(11:0) (HEX)	Wg1(11:0)(HEX)	EnWg0	EnWg1	$\alpha(t_k)$(HEX)
2012.4	362	0A2	1	0	400
2332.05	362	250	0	1	400
3772.34	39C	250	1	0	300
4092.05	39C	2F1	0	1	300
5492.05	3A8	2F1	1	0	200
5812.4	3A8	312	0	1	200
7212.05	3AF	312	1	0	180
7532.05	3AF	328	0	1	180
9052.4	3B1	328	1	0	100
9372.05	3B1	32E	0	1	100
10852.4	3B2	32E	1	0	0C0
11172.4	3B2	332	0	1	0C0
12772.4	3B2	332	1	0	080
13092.4	3B2	333	0	1	080



[13]Fig. 7.19. Simulação para o caso de ativação de Neuron pelas células vizinhas (sinal Ni_RDY).

7.6.4. Correção de erro de implementação.

Os resultados de simulações não foram inicalmente concordantes com as medidas realizadas na fase de caracterização. Após a aplicação de uma ampla gama de vetores de teste de *Neuron*, foi detectado um curto-circuito entre um sinal de controle e uma linha de dados na entrada da unidade de valor absoluto. O sinal de habilitação *enable-EN ABS* que ativa o bloco *ABS BLK* (vide secção inferior da figura 6.21 - arquitetura de *Neuron*) foi conectado de forma equivocada com o 6o bit -$\mathbf{b_5}$ (b_{11} b_{10} ... $\mathbf{b_5}$ ...b_0) do barramento BUS1(11:0). O erro foi causado na fase de síntese de *layout* pela ferramenta de roteamento automático e não detectada pelo procedimento de LVS (*Layout Versus Schematics*) por causa do LVS não ter sido realizado em toda a hierarquia do circuito. O procedimento LVS dever ser realizado no chamado modo ***mask*** para cobrir toda a hierarquia e em modo ***direct***, para verificar unicamente o nível hierárquico corrente. O erro foi produzido no nível de metal2 e corrigido por meio de um feixe laser que eliminou o curto-circuito. A funcionalidade de *Neuron* foi verificada satisfatoriamente após esta fase de correção. Uma imagem da secção consertada pelo feixe laser é ilustrada na figura 7.20. Na imagem detalham-se as secções de trilhas de metal-2 queimadas, causantes do problema.

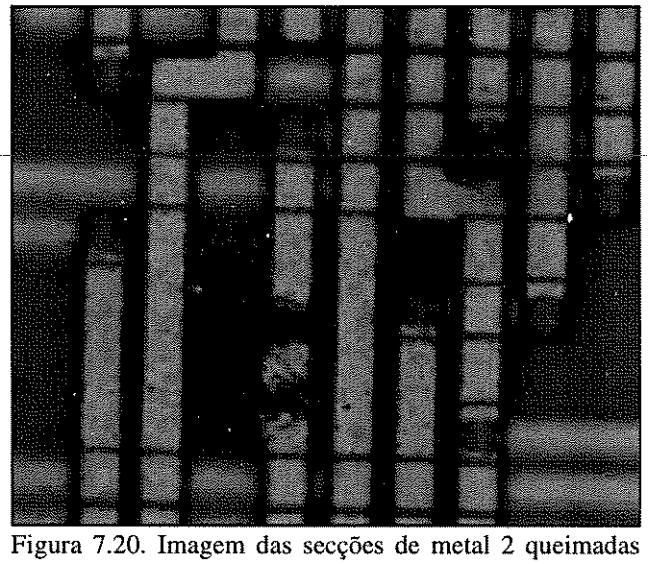


Figura 7.20. Imagem das secções de metal 2 queimadas pelo feixe *laser* para corrigir o defeito de funcionamento de *Neuron*.

Capítulo 8

Conclusão

Este trabalho apresentou o estudo e o projeto em VLSI (*Very Large Scale Integration*) de um circuito dedicado à implementação em *hardware* do algoritmo neural SOFM (*Self-Organizing Feature Map*) proposto por T. Kohonen.

A implementação de sistemas de média complexidade, tal como o caso da célula *Neuron*, acarreta uma variedade de problemas e tomada de decisões relativas à metodologia utilizada, à arquitetura e à tecnologia de implementação. As conclusões mais relevantes obtidas após o desenvolvimento desse projeto são descritas a seguir.

- Metodologia. A metodologia *top-down* não está necessariamente associada a implementações eficientes. A maioria das vezes obteve-se melhores resultados, i.e., estruturas de circuitos mais compactas, com projetos elaborados cuidadosamente de forma manual. No entanto, o mesmo procedimento manual torna-se freqüentemente o gargalo do tempo de execução de projetos, na medida em que aumenta a complexidade dos sistemas. Atualmente, as condições de mercado exigem uma rápida transferência das especificações até a obtenção dos circuitos. Metodologias baseadas na modelagem em alto nível, usando síntese automática revelam-se muito mais eficientes, desde o ponto de vista do tempo de desenvolvimento do produto até a sua colocação no mercado.

- Prototipação. Para a verificação funcional de sistemas de média complexidade, i.e. em torno de 5-50 K*gates*, já é possível a implementação em sistemas FPGAs (*Fiel-Programmable Gate Arrays*), sendo que FPGAs mais avançadas permitem a implementação de circuitos de mais de 100 K*gates*. No entanto, o seu desempenho e aproveitamento de área ainda deixa bastante a desejar se compararmos com implementações em ASICs. Este é um fator crítico em sistemas que devem operar em tempo real, tal como na compressão/decompressão de imagens, equalização adaptativa, etc., onde o desempenho do sistema torna-se prioritário. Por outro lado, nesse projeto, o tempo de retorno da *foundry* levou mais de 7 meses até a recepção dos *chips* para a caracterização. Este atraso é excessivo caso se leve em conta que no atual ritmo de produção comercial internacional, um circuito pode ficar obsoleto em menos de 1 ano.

- Modelagem e Codificação em HDL. Na geração das estruturas de circuitos por meio do processo de síntese automática e otimização, é particularmente crítica a forma de codificação dos programas fontes pela linguagem de descrição de *hardware*. As estruturas geradas variam sensivelmente, dependendo da forma do código original. Para pequenas variações na sintaxe, o sintetizador muitas vezes infere estruturas de *hardware* de forma diferente, com a conseqüente influência no seu desempenho. Todavia detectaram-se diferenças utilizando o mesmo código fonte VHDL sendo sintetizadas por ferramentas diferentes.

- Testabilidade. No projeto de *Neuron*, não foram utilizadas células especiais para a realização de testes, tais como as *scan cells* disponíveis na biblioteca de *standard cells*. O motivo é o incremento da área ocasionado em cada uma destas células, que é praticamente duplicado. Os testes de *Neuron* foram baseados essencialmente em simulações exaustivas após a fase de otimização e mapeamento na tecnologia, verificando desta forma a sua funcionalidade. O procedimento foi repetido com os circuitos fabricados, aplicando-se os mesmos vetores de teste por meio do módulo gerador de padrões do sistema de análise lógica. Para o caso do desenvolvimento de sistemas complexos, é importante contar com ferramentas de verificação formal para aumentar a confiabilidade, especialmente na fase de síntese.

- Tecnologia. A escolha da tecnologia é sem dúvida essencial no desempenho e na otimização da área de silício ocupada . A célula *Neuron* foi desenvolvida em tecnologia CMOS de 1.2 microns (μm). Testes realizados em tecnologia 0.8 μm produziram *layouts* cuja área ocupada foi reduzida até em 50% em relação ao mesmo circuito implementado em 1.2 μm.

Ultimamente, redes neurais auto-organizáveis baseadas no modelo de Kohonen têm sido bastante estudadas e a sua aplicação incrementada em diversas áreas da engenharia. Porém, a implementação e a modelagem por *software* é majoritária e apesar do crescente número de implementações em *hardware*, ainda não é possível ver uma grande utilização de circuitos neurais no mundo real. A evolução da tecnologia em microeletrônica e principalmente das ferramentas de automação de projetos de circuitos e sistemas integrados têm ampliado o número de circuitos de média e grande complexidade, nos quais os sistemas baseados em modelos de processamento neural se incluem. A implementação da célula *Neuron* é importante na medida que permitiu estabelecer critérios práticos de projeto e detectar problemas metodológicos. Além disso, foi possível avaliar as vantagens e desvantagens da arquitetura proposta, sendo adequada para a implementação do algoritmo SOFM sob diferentes variantes. Na implementação de uma rede dedicada a resolver um problema específico, tal como uma operação em tempo real, a arquitetura deve ser re-avaliada, de acordo com a aplicação.

É de particular interesse a integração de circuitos utilizando as vantagens tanto da tecnologia análogica quanto da digital para a realização de sistemas híbridos ou *mixed* em VLSI e ULSI (*Ultra Large Scale Integration*). Um aspecto de particular interesse do autor era a implementação e a avaliação de uma rede de 3×3 células utilizando a técnica MCM (*Multi-Chip-Module*). A implementação não foi concretizada por causa de não serem recebidos os *dies* sem encapsulamento que eram parte do PMU (Projeto Multi Usuário) utilizado.

Bibliografia

Cap 1. Auto-Organização - Retrospectiva

[1] C.W. Kilmister (editor); "Disequilibrium and Self-Organisation"; Contributions to the 2nd and 3rd ISG meetings (ISG-International Study Group on self-organising systems and dissipative structures); Vienna (Aus) 1983 and Windsor (UK) 1985; D. Reidel Pub. Co., Dordrecht, Holland, 1986.

[2] M. Debrun, M.E. Gonzáles, O. Pessoa Jr., "Auto-Organização, Estudos Interdisciplinares", Centro de Lógica Epistemologia e História da Ciência, Unicamp, 1996.

[3] Marshall Yovits & Scott Cameron (editors); "Self-Organizing System"; Proceedings of an Interdisciplinary Conference, 5 and 6 May, 1959, Pergamon Press, NY, 1960.

[4] Teuvo Kohonen; "Self-Organized Formation of Topologically Correct Feature Maps"; Biological Cybernetics, vol. 43, pp.59-69, 1982.

[5] P. Schuster & K. Sigmund; "Self-Organization of Biological Macromolecules and Evolutionary Stable Strategies"; in Dynamic of Synergetic Systems, ed. by H. Haken, pp.156-169, Springer-Verlag, Berlin, 1982.

[6] F. Eugene Yates (editor); "Self-Organizing Systems - The Emergence of Order"; Plenum press, NY, 1987.

[7] V. Vasiliev;"Autowave Processes in Kinetic Systems"; Springer-Verlag, Berlin, 1987.

[8] D.J. Willshaw & C. von der Malsburg, "How patterned neural connections can be set up by self-organization", Proc. of the Royal Society of London, B. Biological Sciences, Vol. 194, No. 1117, Nov., 1976.

[9] W. Ebeling & R. Feister; "Physical Models of Evolution Process - Part IV Evolution and Self-Organization"; in Self-Organization Auto-waves and Structures far from Equilibrium, ed. by V. Krinsky, pp.233-239, Proc. of an Int. Symposium, Pushchino, USSR, July, 1983.

[10] B. Farley; "Self-Organizing Models for Learned Perception"; in Self-Organization Systems, ed. by M. Yovits & S. Cameron, pp.7-27, Pergamon Press, NY, 1960.

[11] Gordon Pask, "The Natural History of Networks"', in Self-Organization Systems, ed. by M. Yovits & S. Cameron, pp.232-263, Pergamon Press, NY, 1960.

[12] F. Rosenblatt, "Perceptual Generalization over Transformation Groups", em *Self Organizing Systems - Proc. of an Interdisciplinary Conf., 5 & 6 May 1959*, ed. by Yovits & Cameron, pp.63-100, Pergamon Press, NY, 1960.

[13] Warren S. McCulloch; "The Reliability of Biological Systems", em *Self Organizing Systems - Proc. of an Interdisciplinary Conf., 5 & 6 May 1959*, ed. by Yovits & Cameron, pp.264-281, Pergamon Press, NY, 1960.

[14] B. Widrow & M. Lehr; "30 Years of Adaptive Neural Networks : Perceptron, Madaline and BackPropagation". *Proc. of the IEEE* Vol. 78(9), pp. 1415-1442, 1990.

Cap 2. O Mapa Auto-Organizável de Kohonen

[15] Teuvo Kohonen, P. Lehtio, J. Rovamo, J. Hyvarinen and K. Bry; "A Principle of Neural Associative Memory"; Neuroscience, vol.2, pp.1065-1076, 1977.

[16] Teuvo Kohonen; "The Self-Organizing Map" ; Proceedings of the IEEE, vol.78, no.9, pp.1464-1480, September 1990.

[17] Jari A. Kangas, Teuvo Kohonen and Jorma T. Laaksonen; "Variants of Self-Organizing Maps" ; IEEE Transactions on Neural Networks, vol.1, no.1, pp. 93-99, March 1990.

[18] Teuvo Kohonen, Erkki Oja and Pekka Lehtio; "Storage and Processing of Information in Distributed Associative Memory Systems" ; in Parallel Models of Associative Memory updated edition, edited by Geoffrey E. Hinton & James A. Anderson, pp. 129-170, Lawrence Erlbaum Ass. publication, U.S.A., 1989.

[19] Teuvo Kohonen; "Self-Organization and Associative Memory"; Springer-Verlag, second edition,1984.

[20] Sun-Ichi Amari; " Mathematical Foundations of Neurocomputing", Proceedings of the IEEE, vol.78, no.9, September 1990.

[21] John C. Eccles; "The Understanding of the Brain", 2nd ed., McGraw-Hill, 1977.

[22] T. Kohonen, K. Mäkisara & T. Saramäki; "Phonotopic Maps - Insightful Representation of Phonological Features for Speech Recognition", Proc. of the 7th Int. Conf. on Pattern Recognition, Montreal 1984.

[23] H. Ritter & T. Kohonen; "Self-Organizing Semantic Maps", Biological Cybernetics, vol. 61, pp. 241-254, 1989.

[24] David Rumelhart, James McClelland & the PDP Research Group, "Parallel Distributed Processing-Explorations in the Microstructure of Cognition, Vol.1: Foundations", A Bradford

Book & The MIT Press, 1988.

[25] T. Kohonen; "Correlation Matrix Memories", IEEE Trans. on Computers, Vol. c-21, no.4, April 1972.

[26] T. Kohonen; "An Introduction to Neural Computing", Neural Networks, Vol 1, pp.3-16, 1988.

[27] T. Kohonen; "Representation of Sensory Information in Self-Organizing Feature Maps, and Relation of these Maps to Distributed Memory Networks", SPIE Vol. 634 - Optical and Hybrid Computing, 1986.

[28] T. Kohonen; "Clustering, Taxonomy and Topological Maps of Patterns", Proc. of the 6th Int. Conf. on Pattern Recognition, Oct. 1982.

[29] J.W.Sammon; "A Nonlinear Mapping for Data Structure Analysis", IEEE Trans. on Computers, Vol. C-18, No.5, pp.401-409, May 1969.

[30] H.Ritter & K.Schulten;"On the Stationary State of Kohonen's Self-Organizing Sensory Mapping", Biological Cybernetics, vol. 54, pp. 99-106, 1986.

[31] P. Tavan, H. Grubmuller, and H. Kuhnel; "Self-organization of associative memory and pattern classification:recurrent signal processing on topological feature maps" ; Biological Cybernetics, vol. 64, pp. 95-105, 1990.

[32] V. V. Tolat; "An analysis of Kohonen's self-organizing maps using a system of energy functions" ; Biological Cybernetics, vol. 64, pp. 155-164, 1990.

[33] Z.P.Lo & B.Bavarian; "On the rate of convergence in topology preserving neural networks", Biological Cybernetics, vol. 65, pp. 55-63, 1991.

[34] E.Erwin, K.Obermayer & K.Schulten;"Self-organizing maps: stationary states, metastability and convergence rate", Biological Cybernetics, vol. 67, pp. 33-45, 1992.

[35] E.Erwin, K.Obermayer & K.Schulten; "Self-organizing maps: ordering, convergence properties and energy functions", Biological Cybernetics, vol. 67, pp. 47-55, 1992.

[36] M. Cottrell, J.C.Fort & G. Pagès; "Two or Three Things that we know about the Kohonen Algorithm", Samos/Université Paris 1/6 et Université Nancy; France; ftp from The Neuroprose archive, March 1994.

[37] Stephen Luttrell; "Bayesian Analysis of Self-Organising Maps", Defense Research Agency, Worcestershire, United Kingdom; paper submitted to Neural Computation on 10 th May 1993; ftp from The Neuroprose archive, March 1994.

[38] Michel Verleysen; "Neural Networks and Content-Addressable Memories for Vision: from Theory to VLSI", Thèse presentée en vue de l'obtention du grade de Docteur en Sciences Appliquées, Univ. Catholique de Louvain, promoteur: Paul Jespers, Fevrier 1992.

[39] B. Hochet, V. Peiris, G. Corbaz & M. Declerq; "Implementation of a Neuron Dedicated to Kohonen Maps with Learning Capabilities", Proc. of the 1990 Custom Integrated Circuit Conf., Boston, Mass., May 13-16, 1990.

[40] D. Macq, M. Verleysen, P. Jespers & J.D. Legat; "Analog Implementation of a Kohonen Map with On-Chip Learning", IEEE Trans. on Neural Networks, Vol.4, No.3, May 1993.

[41] T. Serrano-Gotarredona & B. Linares-Barranco; "A Real-Time Clustering Microchip Neural Engine", IEEE Trans. on Very large Scale Integration (VLSI) Systems, Vol.4,No.2, June 1996.

[42] Simon Haykin; "Neural Networks - A Comprehensive Foundation", MacMillan College Publishing Co., NJ, 1994.

[43] T. Villmann, R. Der, M. Hermann & T.M. Martínez; "Topology Preservation in Self-Organizing Feature Maps: Exact Definition and Measurement", IEEE Trans. on Neural Networks, Vol.8, No.2, March 1997.

Cap 3. Análise algorítmica do SOFM.

[44] Marvin Minsky & Seymour Papert; "Perceptrons - An Introduction to Computational Geometry", The MIT Press, 2nd *printing*, Junho, 1972.

[45] B. Kröse e P. van der Smagt; "An Introduction to Neural Networks", Fac. of Mathematic & Computer Science, University of Amsterdam, The Netherlands, 1993.

[46] Risto Miikkulainen; "Self-Organizing Process Based on Lateral Inhibition and Synaptic Resource Redistribution", Proc. of the International Conference on Artificial Neural Networks (ICANN-91), Helsinki, Finland, 1991.

[47] Pierre Brémaud; "An Introduction to Probabilistic Modeling", Springer-Verlag, NY, 1988.

[48] J. Makhoul, S. Roucos & H. Gish; "Vector Quantization in Speech Coding"; Proc. of the IEEE, Vol. 73, No.11, pp. 1551- 1588, Nov 1985.

[49] D.E. Knuth; "The Art of Computer Programming", Vol.1, Add.-Wesley,1973.

[50] Lydia Kronsjö,"Computational Complexity of Sequential and Parallel Algorithms", J. Wiley & Sons, Great Britain, 1987.

[51] B. Moret & H. Shapiro, "Algorithms from P to NP", Vol. I, Design & Efficiency, Benjamin/Cummings Pub., CA, 1991.

[52] P. Purdom & Cynthia Brown, "The Analysis of Algorithms", Holt, Rinehart & Winston, NY,1985.

Cap 4. Primeira fase experimental : Implementação em *software* do SOFM.

[53] Yves Cheneval; "Packlib, an Interactive Environment to Develop Modular Software for Data Processing"; in Lecture Notes in Computer Science, vol 930, Int. Workshop on Artificial Neural Networks, Spain, June 1995.

[54] M. Conway, R. Pausch & K. Passarella; "A Tutorial for SUIT - The Simple User Interface Toolkit"; Computer Science Dpt., The University of Virginia, 1992.

[55] T. Kohonen, K. Raivio, O. Simula, J. Henriksson; "Start-up Behaviour of a Neural Network Assited Decision Feedback Equaliser in a Two-Path Channel", Proceedings of the International Conference on Communications, p. 1523-1527, Chicago, June 14-18, 1992.

[56] K. Raivio, J. Henriksson, O. Simula; "Neural Detection of QAM Modulation in the Presence of Interference", Proceedings of the Int. Conf. on Neural Networks, vol4, pages 1566-1569, Perth, Nov. 27-Dec.1 1995.

[57] J. Makhoul, S. Roucos & H. Gish; "Vector Quantization in Speech Coding"; Proc. of the IEEE, Vol. 73, No.11, pp. 1551- 1588, Nov 1985.

[58] T. Kohonen, K. Raivio, O. Simula, J. Henriksson; "Performance Evaluation of Self-Organizing Map Based Neural Equalizers in Dynamic Discrete-Signal Detection"; Proceedings of the International Conference on Artificial Neural Networks, pages II 1677-1680, Helsinki, June 1991.

[59] A. Oppenhein & R. Schafer; "Digital Signal Processing", Prentice-Hall, 1975.

[60] J. Gordon; "The perceptual attack time of musical tones"; Journal of the Acoustical Society of America, 82(1), pp.88-105,1987.

[61] J. Grey & J. Gordon; "Perceptual Effects of Spectral modifications on musical timbres"; Journal of the Acoustical Society of America, 63(5), pp.1493-1500, 1978.

[62] R. Plomp, "Aspects of Tone Sensation - A Psychophysical Study";. A.P, London.

[63] Jari Kangas; "On the Analysis of Pattern Sequences by Self-Organizing Maps"; Thesis for the Degree of Doctor of Technology, Helsinki Univ. of Technology, Finland, 1994.

Cap 5. Desenvolvimento de um Protótipo de um *Neuroprocessador* para o SOFM.

[64] IEEE "Standard 1076", *Very High Speed Integrated Circuits **H**ardware **D**escription **L**anguage*, 1987.

[65] Douglas L. Perry; "VHDL", McGraw-Hill, USA, 1991.

[66] Raul Camposano; "From Behavior to Structure: High-Level Synthesis", IEEE Design & Test of Computers, Vol.7, No.5, pp.8-19, Oct. 1990.

[67] Raul Camposano; "High-Level Synthesis"; Proceedings of the II Brazilian Microelectronics School, Gramado, RS, Brasil, pp. 93-128, 1992.

[68] Giovanni de Micheli; "High-Level Synthesis of digital Circuits", Advances in Computers, Vol. 37, pp.207-283, 1993.

[69]; Jochen Jess; "High-Level Modelling and Formal Verification"; Proceedings of the II Brazilian Microelectronics School, Gramado, RS, Brasil, pp. 129-138, 1992.

[70] Peter Middelhoek & Sreeranga Rajan; "From VHDL to Efficient and First-Time-Right Designs: A Formal Approach"; ACM Transactions on Design Automation of Electronic Systems, Vol.1, No.2, pp.205-250, April 1996.

[71] Apostolos Dollas & J. Sterling Babcock; "Rapid Prototyping of Microelectronic Systems"; Advances in Computers, Vol. 40, pp. 65-125, 1995.

[72] D. Gajski, S. Narayan, L. Ramachandran, F. Vahid & P. Fung; "System Design Methodologies: Aiming at the 100 h. Design Cycle"; IEEE Transactions on VLSI, Vol. 4, No.1, pp.70-81, March 1996.

[73] S. Carlson; "Modeling Style Issues for Synthesis"; in *Applications of VHDL to Circuit Design*, edited by R. Harr & A. Stanculescu; pp.123-161, Kluwer Ac. Pub. 1991.

[74] A. Achyuthan & M. Elmasry; "Mixed Analog/Digital Hardware Synthesis of Artificial Neural Networks"; IEEE Transactions on Computers-Aided Design of Integrated Circuits ans Systems; Vol. 13, No. 9, pp.1073-1087, Sptember 1994.

[75] M. Kawamata, Y. Iwata & T. Higuchi; "Design and Evaluation of Highly Parallel VLSI Processors for 2-D State-Space Digital Filters Using Hierarchical Behavioral Description Language and Synthesizer"; IEICE Trans. Fundamentals (Japan), Vol. E-75-A, No.7, pp.837-845, July 1992.

Cap 6. A arquitetura do neuroprocessador *Neuron*.

[76] Mentor Graphics Corporation; "Introduction to Autologic™ and Design Synthesis"; Instructor Notes, Software Version 8.4 1, 1994.

[77] B. Lutter, W. Gluns e F.J. Rammig; "Using VHDL for Simulation of SDL Specifications", Reprint from "Proc. of the EURO-DAC/EURO-VHDL 92, Hamburg, Germany, Sept. 1992.

[78] Mentor Graphics Corporation; "AutoLogic Blocks™ Manual", Software Version 8.4 1, 1994.

[79] Zvi Kohavi; "Switching and Finite Automata Theory"; second edition, McGraw-Hill Inc., 1978.

[80] Mentor Graphics Corporation; "QuickSim II™ Manual", Software Version 8.4 1, 1994.

[81] Kai Hwang; "Computer Arithmetic -Principles, Architecture & Design", Wiley & Sons,1979.

[82] Patrick Thiran, V. Peiris, P. Heim & B. Hochet; "Quantization Effects in Digitally Behaving Circuit Implementations of Kohonen Networks"; IEEE Transaction on Neural Networks, Vol.5, No.3, May 1994, pp. 450-458.

[83] Donald E. Knuth; "The Art of Computer Programming", Vol.2/Seminumerical Algorithms, Addison-Wesley Pub. Co., 2nd. printing, 1971.

[84]Jordan L. Holt & Jenq-Neng Hwang; "Finite Precision Error Analysis of Neural Network Hardware Implementations", IEEE Trans. on Computers, Vol. 42, No.3, March 1993, pp.281-290.

[85] Stephen Ward & Robert Halstead Jr.; "Computation Structures", The MIT Press, 1990.

[86] R.F. Lyon; "Two's Complement Pipeline Multipliers"; IEEE Trans. on Comm., April 1976, sec. concise papers, pp. 418-425.

[87] T. Hallin & M. Flynn; "Pipelining of Arithmetic Functions", IEEE Trans. on Computers, aug. 1972, Vol. C-21, short notes, pp.880-886.

[88] Uwe Sparmann & Sudhakar Reddy; "On the Effectiveness of Residue Code Cheking for Parallel Two's Complement Multipliers", IEEE Trans. on Very Large Scale Integration (VLSI) Systems, pp.227-239, Vol.4,No.2,June 1996.

[89] M. Jara, W. Machaca., e F. Damiani, "Design of a 4x4 Kohonen Neural Net - VHDL Description". First IEEE International Caracas Conference on Devices, Circuits & Systems.Dez.12-14, 1995, Caracas, Venezuela.

[90] W. Machaca., M. Jara e F. Damiani, "Design of a Kohonen Neural Net - *Neuron*, VHDL Description". II Workshop IBERCHIP, fevereiro 12-15 1996, EP-USP, São Paulo, Brasil.

Cap 7. Implementação Física e Resultados Experimentais.

[91] SI2, Silicon Integration Initiative Inc, ex-CAD Framework Initiative -http://www.cfi.org/

[92] Mentor Graphics Corporation; "System-1076™ Design and Model Development Manual", Software Version 8.4 1, 1994.

[93] Pradip Jha & Nikil Dutt; "High-Level Library Mapping for Arithmetic Components", IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol.4, No.2, June 1996.

[94]AMS -*Austria Mikro Systeme International AG*; "Standard Cell Databook", Schloss Premstäten, Austria 1995.

[95]AMS -*Austria Mikro Systeme*; "AMS HIT-Kit", versão 2.40 de dezembro de 1995.

[96]Mentor Graphics Corporation; "IC Station Encyclopedia™", Software Version 8.6 1, 1996.