Proceedings of the VI Brazilian Conference on Neural Networks - VI Congresso Brasileiro de Redes Neurais
pp. 103–108, June 2–5, 2003 – Centro Universitário da FEI, São Paulo - SP - Brazil

Desenvolvimento de uma Rede Neural LVQ em Linguagem VHDL para Aplicações em Tempo-Real

Mauricio Kugler [1], Jorge Tortato Júnior [2], Heitor S. Lopes [3]
[1,3] Laboratório de Bioinformática, [2] Núcleo de Pesquisa em Engenharia Simultânea
Centro Federal de Educação Tecnológica Paraná – CEFET-PR
Av. 7 de setembro, 3165 – 80230-901 Curitiba (PR) – Brasil
E-mails:mauricio@kugler.com, op3542@siemens.com.br, hslopes@cpgei.cefetpr.br

Abstract

The complexity of embbeded systems has been growing progressively, demanding implementations of real-time signal processing algorithms, usually not feasible with conventional processors. This paper describes the development of a LVQ neural network, to be implemented using a programmable logic device (FPGA). The use of a combinatorial distance-comparing block allows the execution of this task in a single clock cicle, reducing classification time of input sample. The source code, written in VHDL language, is based on a finite-state machine. It is presented a study about memory use and speed of the device as the network parameters are changed. As an example, a 128 dimension LVQ network was implemented, with two classes and 16 subclusters. The classification took 334 ms with a 25 MHz clock.

1. Introdução

Nos sistemas embarcados atuais, cada vez mais é necessário o processamento em tempo real de grande quantidade de informações. Porém, algoritmos de processamento e classificação de sinais e imagens (como por exemplo, as redes neurais) exigem grande capacidade computacional. Esta capacidade nem sempre está disponível em processadores apropriados ao uso em sistemas dedicados, seja por motivos técnicos (consumo de corrente elevado, grandes dimensões físicas, excesso de componentes periféricos) ou econômicos.

Uma alternativa cada vez mais utilizada para o desenvolvimento de sistemas embarcados dedicados de grande capacidade de processamento é a implementação de tais algoritmos em dispositivos de lógica programável (PLD – *Programable Logic Device*). Os PLDs são dispositivos semicondutores que podem ser programados para executar uma funcionalidade específica [1]. Diferente dos microprocessadores e microcontroladores, que executam um *software* com instruções seqüenciais em uma arquitetura pré-definida (Von Neumann, Harvard, etc), os PLD têm sua arquitetura interna definida pelo projetista, permitindo que a estrutura e as funcionalidades do *hardware* do dispositivo sejam particularizadas para o sistema em questão. Uma das grandes vantagens dos PLD é a possibilidade de se definir vários blocos de *hardware* que operam em paralelo (combinacionais e/ou seqüenciais), aumentando muito a capacidade computacional do sistema. Atualmente, os PLDs de alta capacidade baseados em memórias voláteis são chamados FPGAs (*Field Programmable Gate Array* - Matriz de Portas Programáveis em Campo).

Este trabalho mostra a implementação de uma rede neural LVQ (*Learning Vector Quantization* – Quantização de Vetores por Aprendizagem) em linguagem VHDL (*Very High Speed Integrated Circuits Hardware Description Language* - Linguagem de Descrição de *Hardware* de Circuitos de Velocidade Muito Alta) [2]. Este tipo de rede neural, muito usada em trabalhos sobre reconhecimento de padrões [3], é particularmente interessante para a implementação em *hardware* devido ao fato da rede ser baseada no cálculo de distância geométrica entre as amostras e os vetores de referência. Esta característica elimina a necessidade de multiplicadores, blocos estes que ocupam grande quantidade de lógica dos componentes programáveis, além de necessitarem de uma grande quantidade de ciclos de *clock* para a realização dos cálculos.

2. Redes neurais LVQ

A Quantização de Vetores de Aprendizagem (LVQ - *Learning Vector Quantization*) [4] é um método de treinamento de redes neurais para classificação de padrões no qual cada saída da rede representa uma classe em particular (porém, várias saídas também podem ser usadas para uma única classe). Cada classe é referenciada a um vetor de pesos, os quais representam os centros de *clusters* que definem as superfícies de decisão das classes. Uma classe pode ser definida por um ponto único ou por um conjunto de pontos (para uma melhor representação no caso de superfícies de decisão irregulares). Este tipo de rede assume que um conjunto de padrões de treinamento de classificação conhecida é disponível, junto com uma distribuição inicial dos vetores de referência.

No treinamento, a classe T previamente definida de cada amostra de entrada x é comparada à classe C representada pelo centro do *cluster* w mais próximo de cada amostra. O valor de w é atualizado da seguinte forma:

- Se $T = C$ então $w_{novo} = w_{anterior} + \alpha \cdot [x - w_{anterior}]$ (uma amostra semelhante ao *cluster* o atrai);
- Se $T \neq C$ então $w_{novo} = w_{anterior} - \alpha \cdot [x - w_{anterior}]$ (uma amostra diferente do *cluster* o afasta);

onde α é o fator de aprendizagem da rede neural. O treinamento é executado para todas as variáveis de entrada repetidas vezes, sempre com a ordem das variáveis alterada para uma seqüência aleatória, podendo ser encerrado por um número máximo de iterações ou pela estabilidade dos *clusters* [5].

Após o treinamento, uma rede neural LVQ se torna, basicamente, em um comparador de pontos. Cada amostra de entrada da rede será classificada como pertencente à classe do centro de *cluster* mais similar à amostra [6]. A medida de similaridade ou de dissimilaridade entre dois pontos x e y pode ser implementada como a distância geométrica entre esses dois pontos. Das várias normas de distância existentes, a mais comum é a medida ponderada métrica d_p:

$$d_p(x,y) = \left(\sum_{i=1}^{n} w_i \cdot |x_i - y_i|^p \right)^{1/p} \quad (1)$$

sendo n é o número de dimensões do espaço em questão, p a ordem da medida de similaridade e w_i um coeficiente de ponderação [7].

Os casos mais habituais são as distâncias Euclidiana e de *Manhattan*, quando faz-se, respectivamente, p=2 e p=1, com pesos $w_i = 1$ (medida não-ponderada). Apesar da distância Euclidiana ser mais habitual, a simplificação dos cálculos quando da utilização da distância de *Manhattan* a torna mais eficiente para aplicações em tempo-real, reduzindo a:

$$d_1(x,y) = \sum_{i=1}^{n} |x_i - y_i| \quad (2)$$

A distância de *Manhattan* foi a medida de similaridade utilizada na implementação do bloco de cálculo de distância, o qual é a base da rede neural LVQ em *hardware* deste trabalho.

3. Descrição de redes neurais LVQ em linguagem VHDL

Diversos trabalhos procuram determinar uma forma genérica de implementação de redes neurais em *hardware*, porém, sem tentar aproveitar as características particulares de cada tipo de rede neural [8]. Neste trabalho, a característica particular de classificação por comparação de distância geométrica possibilita uma enorme redução do número de elementos lógicos e do tempo de processamento.

O bloco básico de cálculo de distância é mostrado na Figura 1. Seqüencialmente, as dimensões de cada centro de *cluster* são aplicadas na entrada do subtrator, juntamente com as dimensões da amostra a ser classificada. O módulo da diferença destes valores, que representa a distância dos pontos na dimensão em questão, é somada aos valores acumulados de distância das outras dimensões no primeiro registrador. Todas estas operações são realizadas em apenas um ciclo de *clock*. A saída deste registrador é também aplicada à entrada de um comparador, que tem na sua segunda entrada o valor da menor distância total da amostra a um *cluster* que foi encontrada até o momento. Se a nova distância for menor que a armazenada, esta é carregada no segundo registrador. Esta operação também gasta apenas um ciclo de *clock*. Realizadas seqüencialmente para todos os *clusters*, estas operações permitem que seja encontrado o *cluster* de menor distância à amostra, classificando-a segundo a sua classe.

Para comparação, um microprocessador que siga o princípio das máquinas RISC (*Reduced Instruction Set Computer* – Computador de Conjunto de Instruções Reduzido) gastaria, no melhor caso, 5 ciclos de *clock* para executar a subtração condicional e a soma do acumulador de distância. Este número, porém, é ainda maior na maioria dos casos, pois não leva em conta os problemas de conflitos de controle (que inclui o preenchimento do *pipeline* do processador), conflitos estruturais (seqüências de instruções que geram ciclos maiores de execução) e dependência de dados (o que exige que o resultado de uma operação seja reescrito nos registradores para ser utilizado) [9].

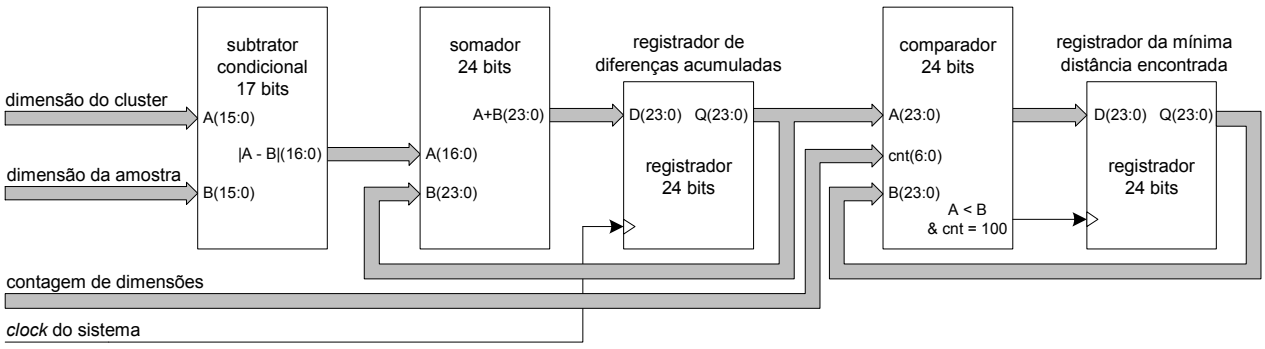


Figura 1: Módulo de cálculo e registro de distância mínima

O esquemático do sistema completo é mostrado na Figura 2. Este diagrama apresenta uma memória ROM para o armazenamento dos centros de *clusters* (os quais são definidos *a priori* por sessões de treinamento da rede neural) e uma memória FIFO para o recebimento das amostras durante a classificação da amostra anterior. O controle do sistema é realizado pela máquina de estados mostrada na Figura 3. Nesta, os estados de S1 e S2 correpondem à transferência da amostra de entrada da FIFO para a RAM interna. Os estados de S3 a S8 realizam as comparações sucessivas de cada dimensão da amostra de entrada e dos vetores de referência, armazenando o vetor de referência de menor distância em relação à amostra de entrada. A rede LVQ da Figura 3 é configurada para 100 dimensões e 20 *subclusters* por classe, com duas classes. O resultado final da classificação é fornecido em S9.

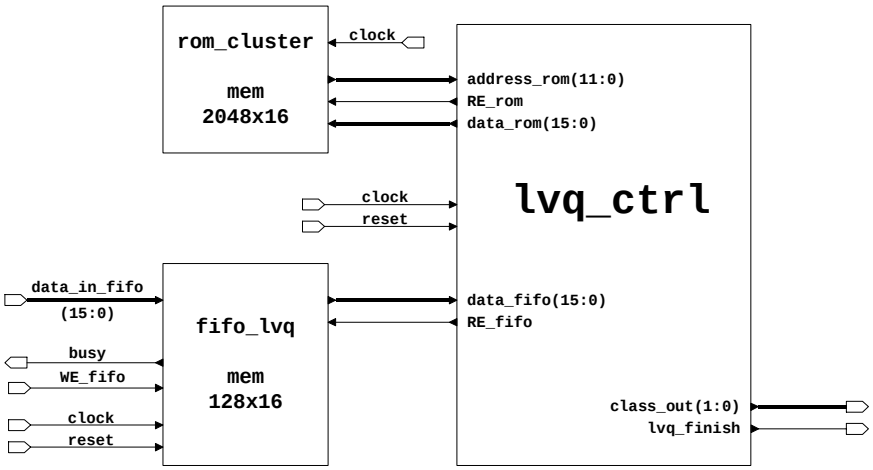


Figura 2: Diagrama do sistema completo da rede neural LVQ em *hardware*

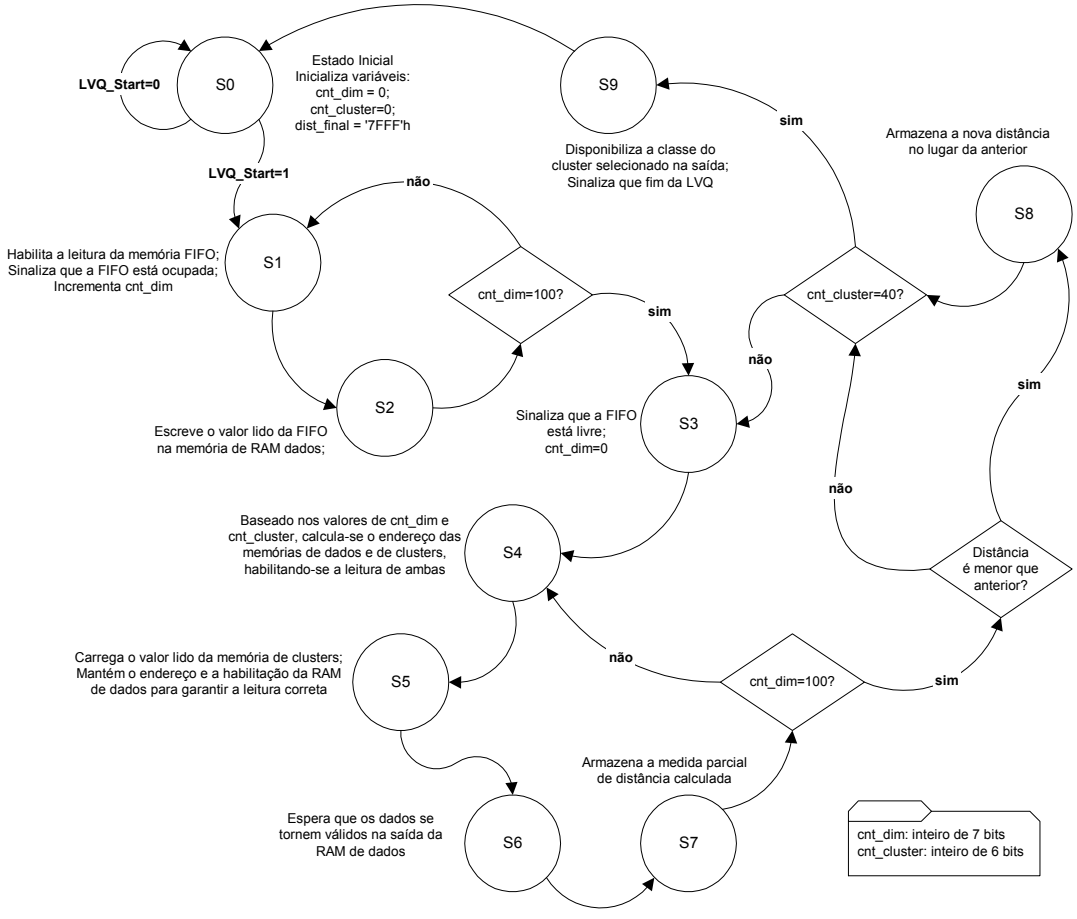


Figura 3: Máquina de estados de controle da rede neural LVQ

4. Resultados

Na implementação em *hardware* de uma rede neural LVQ, o fator limitante é a grande quantidade de memória necessária para o armazenamento dos vetores de referência. Nas simulações realizadas, foi implementada uma rede de 64 dimensões, duas classes e 16 *subclusters* por classe. A síntese foi realizada em um dispositivo ACEX1k100, da Altera, que possui 49152 *bits* de memória interna e 4992 células lógicas. Para a compilação dos circuitos descritos, foi utilizado o *software* Leonardo (Exemplar) e para a síntese física e simulações foram utilizados os softwares MAX+plus II e Quartus II (Altera).

Na Figura 4 é mostrado o preenchimento inicial da memória FIFO de entrada. Foi gravado um vetor igual a [1, 2, 3, 4, 5, 6, 0, 0, 0, ...,0] para ser comparado aos vetores de referência (na simulação, o valor inicial dos dados da memória FIFO é '0', porém, em uma execução real, todos os valores necessitam ser preenchidos). Apenas para exemplo, os dois primeiros vetores da ROM contém os dados [0, 1, 2, 3, 4, 5, 0, 0, 0, ..., 0] e [1, 2, 3, 4, 5, 6, 0, 0, 0, ..., 0] respectivamente. Com estes vetores, dois casos podem ser testados: um centro de *cluster* mais próximo da origem do que o vetor fornecido e um centro de *cluster* igual ao vetor de entrada, este de classe diferente do primeiro. A execução da rede LVQ é iniciada pelo sinal *lvq_start*, após o qual os dados da FIFO são transferidos para a RAM interna. Durante esta transferência, o sinal *fifo_busy* indica que a FIFO está bloqueada para escrita.

Na Figura 5(a), é mostrada a comparação do vetor de entrada com o primeiro vetor de referência, o qual é um vetor com dimensões menores do que o vetor de entrada. Cada dimensão é comparada seqüencialmente, sendo o endereço das memórias calculado a partir dos três contadores mostrados: contador de dimensões (*cnt_dim*), contador de *subclusters* (*cnt_cluster*) e contador de classes (*cnt_class*). As diferenças de cada dimensão são calculadas no sinal *dif_temp* e depois acumuladas no sinal *dif_acc*. Também se pode observar o final do período de transferência dos dados da memória FIFO. Na Figura 5(b), o vetor de entrada é comparado com um centro de *cluster* idêntico a ele, gerando diferenças acumuladas nulas. A classe temporária muda de estado, já que este último *cluster* pertence à classe contrária do vetor mais próximo anterior.

As diferenças acumuladas de valor 21 são as comparações do vetor de entrada com os valores nulos da FIFO, já que apenas os dois primeiros vetores foram preenchidos.

A indicação de fim de classificação e o sinal de saída indicando a classe final são mostrados na Figura 6. O tempo total de execução da rede neural LVQ para a classificação de um ponto (compreendido entre os sinais *lvq_start* e *lvq_finish*) é de cerca de 334μs, a um *clock* de 25 MHz. A Tabela 1 mostra os tempos de cada estágio do processo e o tempo total de processamento da rede neural LVQ. Este tempo aumenta proporcionalmente com número de pontos que devem ser comparados, número este representado pelo produto entre número de dimensões, número de *subclusters* por classe e número de classes. A Tabela 2 mostra o total de utilização de recursos lógicos da FPGA na síntese do circuito apresentado, bem como os percentuais de utilização de uma FPGA ACEX1k100, da Altera, que estas valores representam.

Tabela 1: Tempos de execução dos estágios da rede neural LVQ

Estágio	**Tempo**
Preenchimento da RAM de dados	5,12μs
Comparação de uma dimensão	170ns
Cálculo de um vetor de 64 dimensões	10,28μs
Total (32 *subclusters*)	~334μs

Tabela 2: Utilização de recursos lógicos na síntese da rede neural LVQ

Bloco	**Células lógicas**		**Bits de memória**	
Fifo	26	(0,52%)	1024	(2,08%)
Rom_w	-	-	32768	(66,67%)
Lvq_ctrl	258	(5,17%)	1024	(2,08%)
Total	284	(5,69%)	34816	(70,83%)

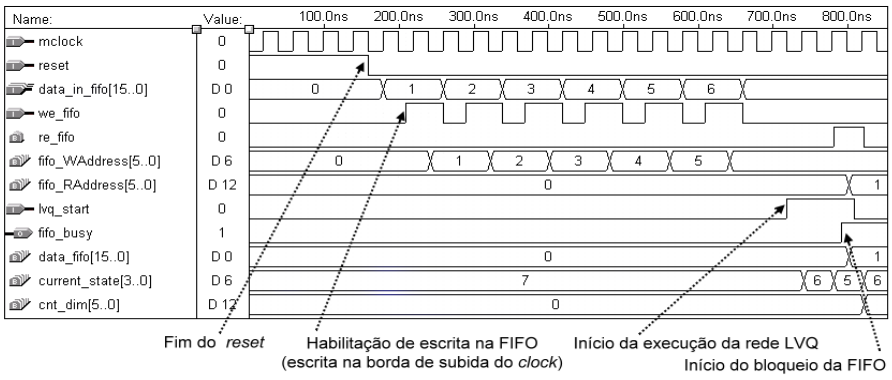
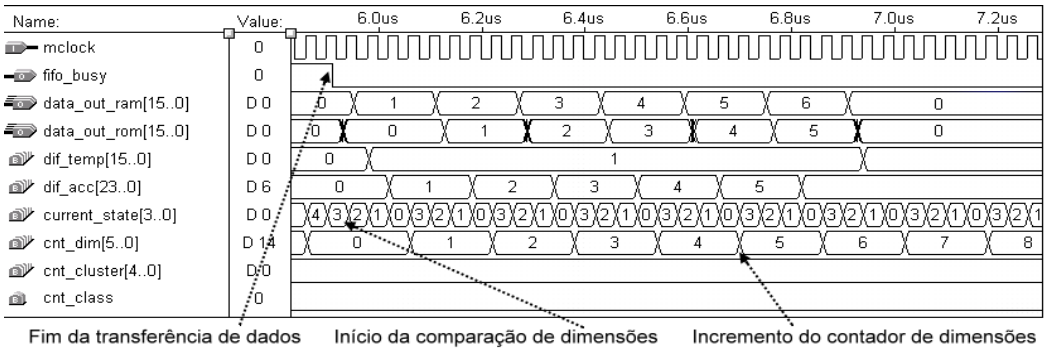
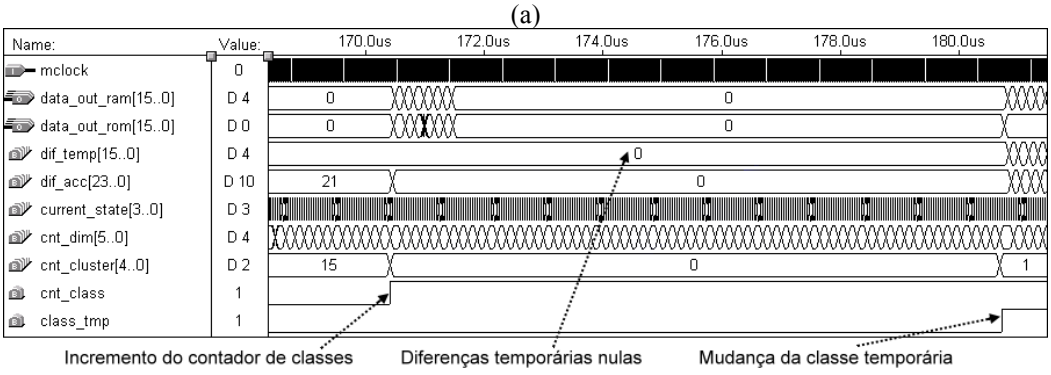


Figura 4: Seqüência de escrita na memória FIFO da rede neural LVQ



(a)



(b)

Figura 5: Comparação do vetor de entrada com dois casos de vetores de referência

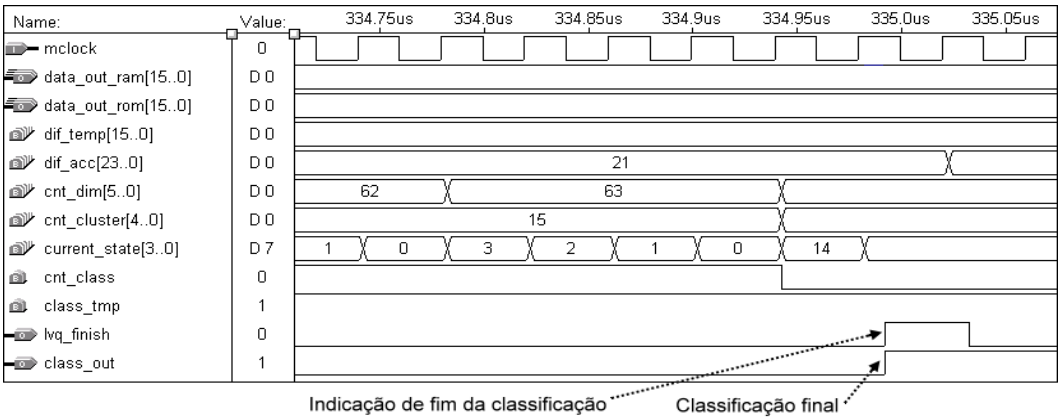


Figura 6: Fim da execução da rede neural LVQ

No código implementado, todos os parâmetros da rede LVQ (número de dimensões, número de *subclusters* por classe e número de classes) podem ser facilmente alterados, tornando-o flexível para uma grande variedade de aplicações.

O aumento do número de dimensões e *subclusters* por classe aumenta apenas a utilização de memória, mas não aumenta consideravelmente do número de células lógicas. Considerando a notação de ponto fixo de 16 bits utilizada, o número de células lógicas necessárias é a soma dos bits utilizados pelas memórias FIFO, RAM e ROM, apresentando um aumento linear, de acordo com a equação 3, a seguir:

$$M_T = 16 \cdot d \cdot (2 + c \cdot s) \qquad (3)$$

onde: M_T é o número total de bits de memória, d é número de dimensões, c é número de classes e s é número de *subclusters* por classe.

A Figura 7 mostra os limites máximos dos parâmetros da rede neural LVQ para a sua implementação em uma FPGA ACEX1k100, considerando seus 49152 *bits* de memória interna. As linhas associadas aos números de classes indicam as combinações dos parâmetros (dimensões e *subclusters* por classe) que utilizam o total (ou próximo do total) de *bits* da memória interna do componente. As linhas relativas a números de classes elevados (128 a 1024), assim como o extremo inferior de cada eixo, são mostrados apenas para ilustração, já que não são valores usuais de implementação de uma rede neural LVQ.

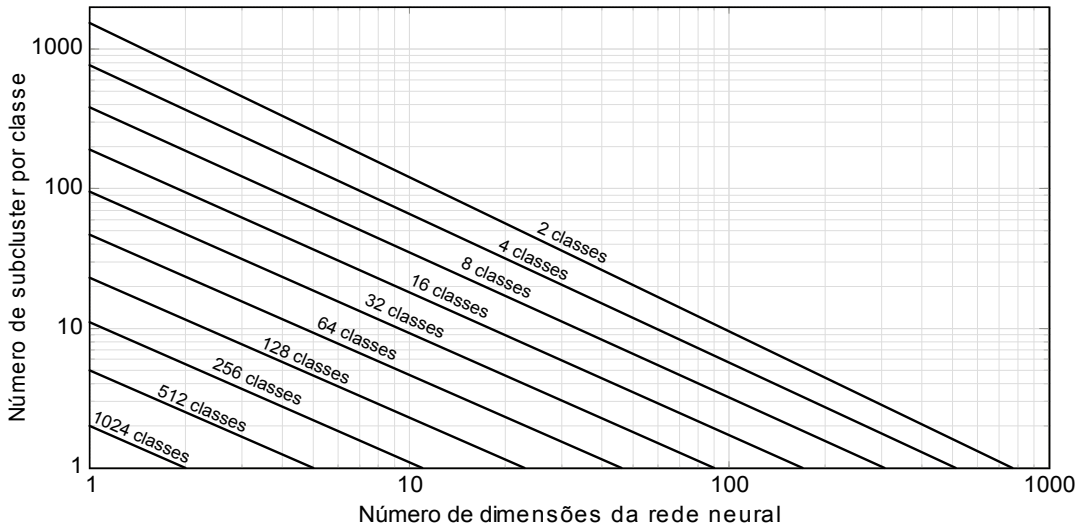


Figura 7: Limites de configuração dos parâmetros da rede LVQ para um dispositivo FPGA ACEX1k100

A velocidade de execução da rede, dada pelo número de ciclos de *clock* necessários para a execução completa da classificação de um vetor de entrada, também tem um crescimento semelhante à área de utilização da FPGA, sendo proporcional ao produto entre o número de dimensões, o número de classes e o número de *subclusters* por classe.

5. Conclusões

A implementação em linguagem VHDL da rede neural LVQ mostrou ser bastante complexa a seqüência de desenvolvimento deste tipo de circuito. Porém, apresentou resultados superiores à capacidade de processamento de microprocessadores RISC de freqüência de *clock* e largura de barramento semelhantes ao que foi utilizado nas FPGAs. O principal fator para essa diferença de eficiência é o fato de processadores seqüenciais gastarem tempo com instruções para o deslocamento de fluxo de execução, teste de condições e para o acesso de dados em memória, tempo este significativo no caso de seqüências repetitivas de cálculo, e que é muito menor no caso de uma implementação em *hardware*, devido à possibilidade de implementação de circuitos combinacionais específicos e blocos paralelos. A eficiência dos circuitos desenvolvidos torna-os possíveis de serem utilizados em várias outras aplicações de sistemas embarcados.

Referências

[1] R.P. Jasinski. *Introdução à Linguagem VHDL*. Relatório Técnico. Laboratório de Microeletrônica – CPDTT - CEFET-PR, Curitiba, 2001, 47 p.

[2] D.L. Perry. *VHDL***.** 3. ed. McGraw-Hill, New York, pages 1-16, 1998.

[3] M. Kugler, H.S. Lopes. Using a chain of LVQ neural networks for pattern recognition of EEG signals related to intermittent photic-stimulation. In: *VII Brazilian Symposium on Neural Networks*, Recife, IEEE Computer Society, pages 164-168, 2002.

[4] T. Kohonen. *Self Organization Maps.* 2. ed. Springer-Verlag, Heidelberg, pages 245-261, 1997.

[5] S. Haykin. *Redes Neurais: Princípios e Prática.* 2. ed. Porto Alegre: Bookman, p. 483-519, 2001.

[6] M. Friedman, A. Kandel. *Introduction to Pattern Recognition: Statistical, Structural, Neural and Fuzzy Logic Approaches.* Imperial College Press, London, pages 55-98, 1999.

[7] S. Theodoridis, K. Koutroumbas. *Pattern Recognition.* Academic Press, London, pages 351-382, 1999.

[8] J.J. Blake, L.P. Maguire, T.M. McGinnity, B. Roche, L.J. McDaid. The implementation of fuzzy systems, neural networks and fuzzy neural networks using FPGAs. *Information Sciences*, 112(1-4):151-168, 1998.

[9] J.L. Hennessy, D.A. Patterson. *Computer Architecture: a Quantitative Approach*. 2. ed. Morgan Kaufmann Publishers, San Francisco, pages 139-146, 1996.

Rev. 15/04/2003