Proceedings of the IV Brazilian Conference on Neural Networks - IV Congresso Brasileiro de Redes Neurais
pp. 013-018, July 20-22, 1999 - ITA, São José dos Campos - SP - Brazil

Uso de um Ambiente Codesign para a Implementação de Redes Neurais

Rolf F. Molz[1], Paulo M. Engel[2], Fernando G. Moraes[3]

[1]Depto. de Informática – UNISC - Caixa Postal 188 - Santa Cruz do Sul - RS. CEP: 96815-900.
[2]Inst. de Informática – UFRGS - Caixa Postal 15064 - POA - RS. CEP: 91501-970.
[3]Inst. de Informática – PUC - Av. Ipiranga, 6681 - prédio 30 / bloco 4 - POA - RS. CEP: 90619-900
E-mails: rolf@dinf.unisc.br , engel@inf.ufrgs.br, moraes@inf.pucrs.br

Abstract

This paper presents a hardware implementation of neural networks into reprogrammable devices, FPGAs. It was used for this implementation the LIRMM board, which allows to develop both hardware and software designs, using C and VHDL languages, respectively. A feedforward neural network was implemented, using backpropagation learning. Emphasis is given for the net calculus, since intensive multiplications and additions are needed to compute this function. At the neuron level, the size of arithmetic operators was reduced by correctly coding the synaptics weights and input patterns, as well as replacing multipliers by adders and look-up-tables. At the network level, two techniques were employed to neuron synchronization. The synchronous solution resulted in a correct neural network implemented in hardware. After this implementation, the simulation of the neural network is presented. This work is organized as follows: section 1 introduces the use of neural network; section 2 presents the hardware-software environment used; section 3 details the neural network implementations (digital, analog, hybrid), our network and the simulation; and finally, our conclusions and future work.

1. Introdução

As redes neurais têm sido largamente utilizadas em muitos campos interdisciplinares, tanto para desenvolvimento como para aplicação. Estas são habilitadas a solucionar uma grande variedade de problemas que possuem uma solução muito difícil por outros métodos. As redes neurais são especialmente úteis para problemas que necessitam do emprego da característica de generalização. Os exemplos de aplicação se estendem do campo comercial às áreas de pesquisas que demonstram cada vez mais a sua aplicabilidade.

No campo de implementação de redes neurais encontram-se basicamente dois métodos, que são: software e hardware. A implementação de redes neurais por software possui a vantagem de ser rápida e facilmente implementável, enquanto que a implementação por hardware geralmente recai num processo mais demorado e mais difícil de ser implementado.

Contudo, estas diferenças estão diminuindo graças ao avanço tecnológico que nos é imposto. Com o surgimento e evolução dos dispositivos reprogramáveis podemos realizar a implementação de redes neurais em um curto intervalo de tempo. Entretanto ainda existem dificuldades quanto a este tipo de implementação.

Com isto, este artigo tem como objetivo apresentar uma implementação em hardware de redes neurais artificiais por meio do uso de dispositivos configuráveis alocados em um ambiente de codesign.

2. Ambiente de Codesign

Neste artigo propomos o desenvolvimento de uma implementação em hardware de redes neurais artificiais. Será utilizado como ambiente de desenvolvimento uma placa que nos possibilita o emprego de técnicas de codesign.

Para tanto, adotou-se uma placa desenvolvida no Laboratório de Informática, Robótica e Microeletrônica de Montpellier, a qual é denominada de LIRMM (*Logic Inside Reconfigurable Micro Machine*) [1].

Esta placa foi desenvolvida de modo a se obter um ambiente de prototipação rápido baseado em um processador de sinais (DSP Texas TMS 320C40), para a prototipação de *software*, e dois circuitos reconfiguráveis (FPGA), para a prototipação de *hardware*.

Esta plataforma permite o desenvolvimento conjunto de *hardware/software* (codesign), dada a presença de um processador (projeto de *software*) e dispositivos programáveis (projeto de *hardware*). O processador de sinais é programado após a geração e avaliação de um código gerado em linguagem C. Os circuitos reconfiguráveis são programados por meio de ferramentas de mapeamento de *hardware* e síntese lógica (Synopsys, Xilinx).

2.1. Arquitetura da Placa LIRMM

Nesta seção abordaremos de forma sucinta a arquitetura da placa que foi utilizada para implementação.

Como foi comentado no início desta seção, a placa LIRMM é composta por um DSP e por duas FPGAs.

A ULA do DSP permite realizar em paralelo operações de soma e multiplicação, favorecendo a

implementação de, por exemplo, filtros digitais (FIR e IIR). Este DSP utiliza seis portas de comunicação para realizar comunicações inter-processadores de alta velocidade, sendo que cada porta está habilitada a realizar transferências a uma taxa de 20 Mbytes/s de dados bidirecionais.

A placa possui dois FPGAs XC4013 (Xilinx). Estes FPGAs possuem, de forma individual, 576 blocos lógicos configuráveis (CLB) e 208 pinos que podem ser interligados aos componentes periféricos.

Para aumentar a flexibilidade da placa LIRMM, as duas FPGAs estão interconectadas por um barramento de 34 bits, permitindo com isto, particionar o *hardware* necessário entre as duas FPGAs. Além disto, para incorporar uma maior flexibilidade, há uma memória RAM estática local de 128Kbytes para cada FPGA.

A frequência de oscilação do *clock* pode ser ajustada para 20Mhz ou 40Mhz. Isto permite uma execução em tempo real para determinadas aplicações. Para esta implementação a frequência do clock está ajustada para 20Mhz.

Na figura 2.1 pode-se visualizar a estrutura geral da placa de prototipação rápida LIRMM.

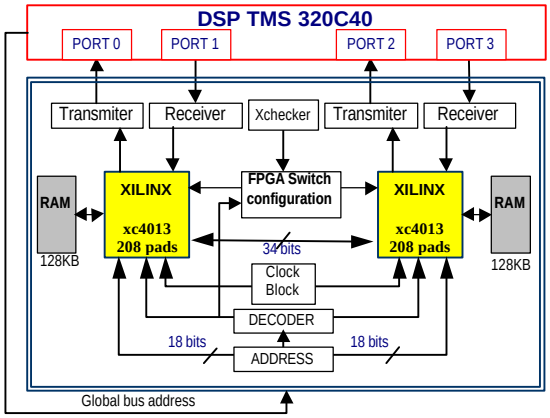

Figura 2.1: Estrutura geral da placa LIRMM

O DSP envia dois sinais básicos que são utilizados para realizar a comunicação entre o DSP e as duas FPGAs. Estes são chamados de:

- CSTRB (*communication port strobe*) – este sinal indica que o DSP disponibilizou dados válidos na porta de comunicação.
- CRDY (*communication port ready*) – este sinal indica que o DSP recebeu dados pela porta de comunicação.

A palavra a ser enviada ou recebida pelo DSP é de 32 bits, que são enviados ou recebidos em 4 blocos seriais de 8 bits.

Como pode ser observado na figura 2.1, as comunicações entre os dispositivos são realizadas de forma individual, isto é, cada FPGA possui um canal somente para emissão e outro canal somente para recepção de dados. Estes canais de comunicação são implementados em hardware através de códigos previamente descritos em VHDL e ocupam menos de 40 CLBs, o que representa somente 7% do número total de CLBs existentes em cada uma das FPGAs [1].

3. Implementação de Redes Neurais em FPGA

Na implementação por *hardware*, o tempo de desenvolvimento de um protótipo é maior comparado ao tempo utilizado em uma implementação por *software*. Contudo, devido a alta taxa de processamento paralelo que pode ser obtida, torna-se ideal para aplicações que envolvam o processamento de sinais em tempo real.

As redes neurais artificiais podem ser implementadas em *hardware* semi-dedicado ou dedicado. Para a implementação em *hardware* dedicado pode-se abordar a forma do projeto sob três focos distintos. O primeiro foco utiliza técnicas digitais de implementação, [2], o segundo foco aborda as técnicas analógicas, como pode ser examinado em [3], [4], [5], [6], [7] e [8]; e por fim, o terceiro foco se utiliza das técnicas híbridas, como em [9] e em [10], tendo a implementação composta em parte digital e em parte analógica.

O estilo de projeto utilizando técnicas analógicas é popular para a implementação de redes neurais, pois, através desta técnica obtém-se circuitos compactos capazes de realizar um processamento assíncrono de alta velocidade. Porém possui um problema quanto à exatidão dos valores de tensão exigidos na implementação. Este problema é abordado em [8].

Esta dificuldade, aliada ao alto tempo necessário para a validação de um sistema implementado em *hardware* analógico, nos conduziu ao desenvolvimento deste trabalho, onde se pretende investigar novas soluções para implementações de redes neurais em *hardware*, conciliando uma solução que adote a melhor performance, simplicidade e baixo custo. A solução que será estudada é a implementação digital de redes neurais artificiais em dispositivos configuráveis do tipo FPGA utilizando um ambiente de codesign.

Muitos trabalhos foram publicados, mostrando a viabilidade técnica deste tipo de implementação [11], [12], [13], [14], [15] e [16].

Em todos estes artigos salienta-se como principal problema de implementação o produto existente no cálculo do *net* , que é valor de saída que cada neurônio calcula antes de ser aplicada a função de ativação. Como outro problema que deve ser considerado temos a aplicação da função de ativação. Esta função de ativação, para determinadas topologias de redes neurais, torna a implementação mais difícil e as vezes menos exata. Tal problema será discutido mais adiante.

A equação (1), apresenta o cálculo do valor do *net* de um neurônio básico.

$$net_i = \sum_{j=0}^{n} W_{ij} * I_j \qquad (1)$$

Na figura 3.1 é apresentado um diagrama básico de um neurônio. Através da equação (1) e da figura 3.1 pode-se verificar a necessidade de multiplicações no processo de propagação em uma rede neural. Também verifica-se a necessidade da aplicação de uma função de ativação, que pode ser, por exemplo, uma função sigmoidal para o caso de uma rede neural *feedforward* com aprendizado por *backpropagation*.

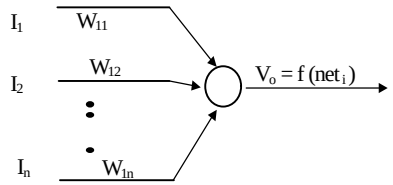

Figura 3.1: Diagrama básico de um neurônio.

Para solucionar o problema da multiplicação, pode-se adotar técnicas de multiplicação que visam diminuir o tamanho do circuito resultante. Dentre as técnicas mais empregadas está a multiplicação serial, onde se implementa somente um bloco multiplicativo [17]. Com isto, os padrões de entrada necessitam ser serializados de forma a realizar a operação de multiplicação a nível de bit, e não a nível de palavra.

Outro método empregado é a utilização da regra da aritmética distribuída. Este processo de implementação é empregado aproveitando-se da forma construtiva dos FPGAs, as quais utilizam *lookup tables* [11].

Por fim, o método de multiplicação que foi empregado neste trabalho aborda a multiplicação como operações de deslocamento, sendo que para tal, os valores utilizados devem ser múltiplos da potência de dois [12].

Com relação a função de ativação na saída de cada neurônio, foi adotada a implementação da função de ativação por meio de *lookup tables*. Através deste método são armazenados valores previamente definidos que serão as respostas para determinadas entradas.

Antes de examinarmos com mais detalhes a implementação propriamente dita, iremos verificar um ponto importante na implementação de redes neurais em *hardware*. Este ponto diz respeito a notação numérica dos valores de entrada dos neurônios, saída dos neurônios e dos pesos. Como foi salientado acima, este trabalho adotou como procedimento multiplicativo a operação de deslocamentos binários sucessivos, sendo que para tal, todos os números envolvidos devem ser múltiplos de potência de dois.

No caso de redes neurais, torna-se evidente o uso de valores não inteiros. Neste caso, para se evitar a utilização da notação em ponto flutuante, o que acarretaria um aumento do número de CLBs empregados, adotou-se uma alternativa que é bem apresentada em [12]. Abaixo, tabela 2.1, estão apresentados os possíveis valores utilizados para a entrada e saída dos neurônios. Estes números estão codificados em binário e compreendem o intervalo fechado pelos números reais –1 e +1.

Esta codificação representa um número em ponto flutuante de três bits, sendo 1 bit para mantissa (m) e 2 bits para expoentes (e). Os correspondentes números são calculados por meio da seguinte expressão:

$$\text{Número} = -1^{m} * 2^{-e} \qquad (2)$$

Durante o restante deste artigo adotamos a simbologia de se colocar a letra "b" após uma seqüência binária para se indicar um número em notação binária.

Valor Codificado	Valor Real
100b	-1
101b	-2^{-1}
110b	-2^{-2}
111b	-2^{-3}
011b	2^{-3}
010b	2^{-2}
001b	2^{-1}
000b	1

Tabela 2.1 – Possíveis valores para a entrada e saída dos neurônios.

Além destes valores da entrada e saída dos neurônios, os valores dos pesos sinápticos também devem ser múltiplos de potência de dois. Para tais valores, foi adotado um número codificado em 10 bits com ponto decimal fixo, sendo 1 bit para representar o sinal, 4 bits para representar a parte inteira e 5 bits para representar a parte decimal. Por exemplo:

Número Codificado	Valor Real
0 0000 11000	+0,75
1 0001 01000	-1,25

Tabela 2.2 – Exemplo da codificação empregada.

O número de bits existentes para a codificação do peso sináptico fornece uma amplitude que se estende do número 15,96875 a –15,96875.

É importante conhecer a faixa dos valores dos pesos sinápticos para adequar convenientemente o número de bits necessários para a implementação. Se isto não for atentado, corre-se o risco de saturar os valores dos pesos sinápticos. O número de bits da parte decimal foi escolhido de forma empírica.

3.1. Estudo de Caso

O problema adotado para ser solucionado é o conhecido problema da função lógica XOR. Para este problema, adotamos uma rede neural de três camadas. Na figura 3.2, está apresentado o diagrama da rede neural adotada.

Esta rede é composta de dois neurônios na entrada, três neurônios na camada escondida e um neurônio na camada de saída. A camada de entrada não é implementada, visto que esta camada teria a função de acondicionar os sinais à rede. Como isto é realizado pelo *software*, esta camada não é implementada. O

aprendizado desta rede foi realizado pelo algoritmo de *backpropagation*.

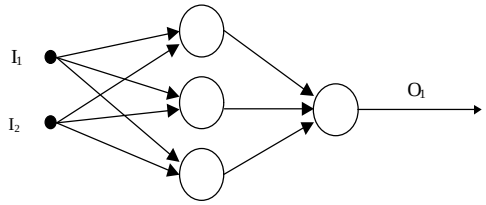

Figura 3.2: Diagrama da rede neural adotada.

O aprendizado da rede neural adotada, figura 3.2, foi realizado de modo *off-line*, com isto, a rede neural implementada ocupa um número mínimo de CLBs. Este aprendizado foi implementado em linguagem C.

Após o término do aprendizado, os pesos são repassados para a descrição VHDL da rede neural. Com isto, a descrição passa pelas fases de síntese em hardware e ao final, a placa de prototipação é colocada no modo de execução. Neste momento, os pesos são transferidos para os respectivos neurônios, permanecendo constantes durante os testes da rede neural.

Um dos problemas da implementação é a aplicação da função de ativação sobre o valor do *net*. Seria muito difícil e custoso realizar uma implementação de uma função do tipo sigmoidal em FPGA.

O problema foi resolvido por meio da utilização de *lookup tables* existentes nos FPGAs empregados. Como isto não deixa de ser uma discretização, nota-se claramente um erro nos valores obtidos como resposta. Para compensá-los e considerá-los no momento do aprendizado, o algoritmo do *backpropagation* foi modificado, de modo que, no programa desenvolvido para realizar o aprendizado, a função de ativação aplicada não seria uma função matemática, mas sim, uma tabela onde os valores foram previamente definidos. Com isto, o aprendizado também levou em consideração este erro.

Outra modificação no aprendizado da rede neural é a adequação dos pesos obtidos durante as iterações para pesos cujos valores são múltiplos de potência de dois. Estes pesos adequados devem respeitar a codificação comentada no item anterior. Percebe-se que com esta adequação, os pesos também transportam erros que conduzem a uma convergência mais demorada do processo de aprendizagem.

Faixa dos valores de *net*	Valores da f(*net*)
x ≤ -1,25	100b
-1,25 > x ≤ -0,625	101b
-0,625 > x ≤ -0,3125	110b
-0,3125 > x ≤ -0,15625	111b
-0,15625 > x ≤ 0,15625	011b
0,15625 > x ≤ 0,3125	010b
0,3125 > x ≤ 0,625	001b
x > 0,625	000b

Tabela 3.1 – Valores de *net* e os valores da f(*net*).

A tabela 3.1, apresenta os valores de saída da função de ativação, f(*net*), para a faixa dos valores de *net* que são aplicados na entrada da função de ativação.

As referências bibliográficas sobre redes neurais artificiais que comentam o problema da função lógica XOR apresentam como solução uma rede com somente dois neurônios na camada escondida. Tal número de neurônios na camada escondida, foi testado de forma inicial, mas a rede não convergiu para um resultado satisfatório. Para tanto, a camada escondida foi adicionada em um neurônio. Um dos motivos desta ampliação do número de neurônios na camada escondida deve-se à discretização da função de ativação sigmoidal adotada no aprendizado, e que, por sua vez, será adotada no momento da implementação. Outro motivo é a adequação dos pesos sinápticos para valores que são múltiplos da potência de dois.

A rede neural foi descrita em VHDL, utilizando-se como ambiente de simulação o programa da Aldec, ActiveVHDL. Para as implementações, utilizaram-se a ferramenta de síntese lógica da Synopsys, FPGAExpress, e a ferramenta de síntese física da Xilinx, Foundation Express 1.5.

Esta rede foi inicialmente implementada no modo assíncrono, isto é, sem a presença de qualquer tipo de sinal de *clock*. Na figura 3.3 pode-se verificar a estrutura dos blocos que compõem este tipo de implementação.

Na figura 3.3 observa-se que após dado um sinal de START, ocorre a propagação do sinal de forma assíncrona. Contudo, há um re-sincronismo no final da primeira camada, permitindo com isto o início da camada seguinte. Tal re-sincronismo é realizado por uma porta lógica.

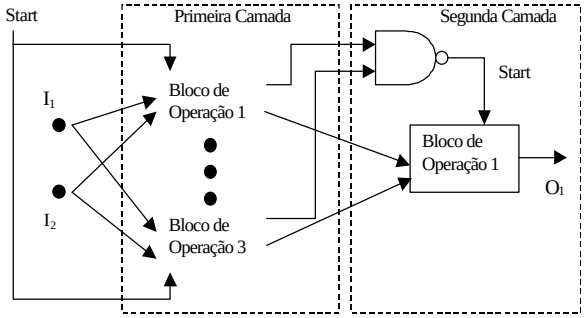

Figura 3.3: Implementação assíncrona.

Este tipo de implementação é simples, pois não há nenhuma parte de controle, facilitando a programação e obtendo-se um circuito menor. Porém, tal implementação não foi continuada devido a problemas de estabilização dos sinais nas entradas das duas camadas, resultando em falhas durante a implementação.

É importante lembrar que tal circuito funciona perfeitamente durante a simulação, mas não opera de forma condizente quando implementado. Tal motivo deve-se ao fato de que no momento da simulação não

são considerados os tempos de estabilização dos sinais envolvidos.

Com este problema de estabilização dos sinais nas entradas dos neurônios, passou-se para uma descrição síncrona da rede. Na figura 3.4 observa-se o diagrama em blocos da rede no modo síncrono.

Com a descrição da rede neural no modo síncrono, percebe-se claramente a distinção entre a parte de controle e a parte operativa.

A parte de controle é responsável pelo sincronismo da parte operativa. Esta é formada por dois registradores de deslocamento, que estabelecem tempos de estabilização e de processamento. Com a parte de controle, conseguiu-se estabelecer um tempo fixo de estabilização dos sinais que são aplicados à entrada dos neurônios e, portanto, a parte operativa realiza sua operação com dados válidos. Além disto, a parte de controle também sincroniza o tempo de processamento da parte operativa, havendo com isto, o correto acionamento da camada seguinte.

Estes tempos foram estabelecidos em função do tempo crítico obtido pela ferramenta de análise de *timing*.

Como pode ser visto pela figura 3.4, observa-se que há uma parte de controle para cada camada de neurônios na rede neural. Com isto, conseguiu-se uma implementação altamente confiável e sem um aumento significativo do número de CLBs ocupados.

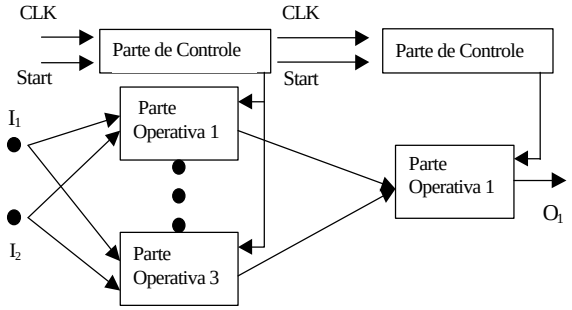

Figura 3.4: Implementação síncrona.

A partir deste momento, vamos descrever o funcionamento da rede implementada de uma forma mais detalhada. Para tal descrição, iremos detalhar os sinais que estão presentes na figura 3.5. Tal figura demonstra uma simulação da rede para um determinado padrão apresentado às suas entradas.

O padrão escolhido para esta simulação foi o padrão codificado 100b para ambas as entradas. Consultando a tabela 3.1, vemos que este valor codificado representa o número –1. Assim, temos duas entradas com valor igual a –1, as quais aplicadas a entrada de uma porta lógica XOR, devem resultar em um número negativo.

Após ter-se programado o DSP em linguagem C e o FPGA com o seu arquivo de configuração, iniciou-se os testes de funcionamento da rede neural a nível de *hardware*.

O programa que foi instalado no DSP realiza a interface entre o PC e a placa de prototipação. Este programa solicita números que são transmitidos ao FPGA que, por sua vez, realiza o processamento e os devolve ao DSP. Após, o DSP retorna o resultado ao PC que os mostra no vídeo.

No início do processamento, gera-se um sinal de inicialização, INIT. Neste sinal é que há a transferência dos pesos sinápticos para os respectivos neurônios.

Um dos principais sinais envolvidos na transferência de dados entre o DSP e o FPGA é o sinal FR. Quando há este sinal, a rede neural lê o sinal *valor*, o qual possui o valor que será aplicado às entradas dos neurônios pertencentes a primeira camada. A primeira camada implementada é a camada escondida, visto que os sinais já estão acondicionados de forma correta. Assim não se necessita da camada de entrada da rede neural. No caso da figura 3.5, o valor que será aplicado a rede neural é o valor 44h ou em binário, 0100 0100b.

Este sinal FR sincronizado com o sinal de *clk*, ativa a parte de controle da primeira camada da rede. Esta parte de controle, após 4 ciclos de *clk*, ativa o sinal GUARDA_I. Este sinal faz com que os dados apresentados às entradas da rede sejam lidos pelos neurônios pertencentes a esta camada. A partir deste momento, os valores lidos são estabilizados novamente por 2 ciclos de *clk*. Terminado este tempo de estabilização, o neurônio começa o seu processamento de forma assíncrona.

Em paralelo, existe um *shift register* que gerencia toda a operação dos neurônios pertencentes a esta camada. No seu término, após 10 ciclos de *clk*, os resultados dos neurônios já estão calculados, sinais SAIDA_I1, SAIDA_I2 e SAIDA_I3. Juntamente com isto, há a geração do sinal INICIA_PROX. Este sinal, inicializa a parte de controle da próxima camada.

O resultado, após terminado o processamento da segunda camada, é enviado ao DSP através do sinal FR2. Nesta simulação, o resultado é igual a 5 em hexadecimal codificado, o que é igual a 101b, ou através da tabela 3.1, o resultado é igual ao valor -0,5.

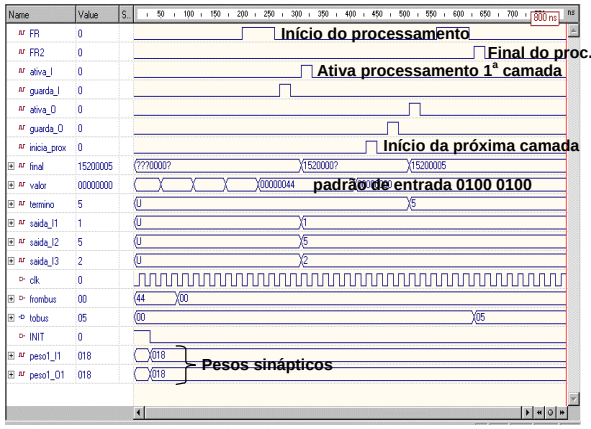

Figura 3.5: Simulação da rede neural implementada.

4. Conclusão

Neste artigo foi apresentado uma implementação de rede neural em dispositivos configuráveis do tipo FPGA, utilizando a placa LIRMM, que é um ambiente prototipação de codesign.

O fator limitante, neste tipo de implementação, é o número de CLBs disponíveis. A implementação utilizou 339 CLBs dos 576 CLBs disponíveis por FPGA. Contudo, hoje já se encontram FPGAs com mais de 200.000 gates. Esta tendência de crescimento nos aponta que, em um futuro próximo, teremos FPGAs com quantidades maiores de CLBs, o que permitirá a implementação de redes neurais complexas, possibilitando um processamento em tempo real.

O ambiente de prototipação, LIRMM, permite uma implementação rápida e também a experimentação de novas técnicas para a implementação de redes neurais. Tais técnicas podem, inclusive, contemplar a utilização do aprendizado por algoritmos genéticos.

Esta facilidade tornou-se evidente devido ao ambiente de prototipação utilizado possuir um DSP que opera em alta velocidade e nos permite a operação de números com grande facilidade.

Embora o aprendizado, neste trabalho, não tenha sido abordado, o mesmo pode ser implementado no DSP, não trazendo nenhum aumento do número de CLBs utilizados na síntese física.

O estudo de caso realizado nos permitiu a verificação de que é importante conhecer como as ferramentas de síntese lógica e física operam. Isto se deve por causa da necessidade de se substituir parte da lógica de descrição da rede neural, por lógica semelhante. Esta troca teve que ser efetivada devido ao não funcionamento da implementação. Com isto viemos a concluir que existem alguns conjuntos de instruções específicos em VHDL que operam corretamente quando implementados em FPGAs.

A descrição da rede neural tornou-se bastante modular, pois a mesma pode ser aumentada ou diminuída no número de neurônios muito facilmente. A descrição no modo síncrono nos permitiu esta modularidade e manteve sua exatidão nos resultados da rede. Com isto, este trabalho implementou e está sugerindo um neurônio básico que pode ser utilizado para outras topologias de redes neurais.

5. Referências Bibliográficas

[1]TORRES, Lionel; PILLEMENT, S.; ROBERT, Michel. **LIRMM: Prototyping Plataform for Hardware/Software Codesign**. Journal of Current Issue in Electronic Modelling, Issue 9, p. 49-59, March 1997.

[2]HASAN, S.M. Rezaul; SIONG, Ng Kang. A Parallel Processing VLSI BAM Engine. **IEEE Trans. on Neural Network**, New York, v.8, n.2, p.424-436, Mar. 1997.

[3]MONTALVO, Antonio J.; GYURCSIK, Ronald S.; PAULOS John J. Toward a General-Purpose Analog VLSI Neural Network with on-chip Learning. **IEEE Trans. on Neural Network**, New York, v.8, n.2, p.413-423, Mar. 1997.

[4]HSU, Charles C.; GOBOVIC, Desa; ZAGHLOUL, Mona E.; et al. Chaotic Neuron Models and their VLSI Circuit Implementations. **IEEE Trans. on Neural Network**, New York, v.7, n.6, p.1339-1350, Nov. 1996.

[5]WANG, Jun. Analysis and Design of an Analog Sorting Network. **IEEE Trans. on Neural Network**, New York, v.6, n.4, p.962-971, July 1995.

[6]KANE, Jonathan S.; KINCAID, Thomas G. Optoelectronic Winner-Take-All VLSI Shunting Neural Network. **IEEE Trans. on Neural Network**, New York, v.6, n.5, p.1275-1279, Sept. 1995.

[7]MOLZ, Rolf F. **Estudo de Técnicas de Implementação de Redes Neurais Competitivas utilizando tecnologias de Circuito VLSI Analógicos:** Trabalho Invidual. Porto Alegre: CPGCC da UFRGS,1996. (TI-582).

[8]MOLZ, Rolf F. **Proposta de Implementação em Hardware dedicado de Redes Neurais Competitivas com Técnicas de Circuitos Integrados Analógicos.** Dissertação de Mestrado, 103 p. CPGCC, UFRGS, 1998.

[9]HOLLIS, Paul W.; PAULOS, John J. A Neural Network Lerarning Algorithm Tailored for VLSI Implementation. **IEEE Trans. on Neural Network**, New York, v.5, n.5, p.784-791, Sept. 1994.

[10]MASRY, Ezz I.; YANG, Hong-Kui; YAKOUT, Mohamed A. Implementations of Artificial Neural Networks using Current- Mode Pulse width Modulation Technique. **IEEE Trans. on Neural Network**, New York, v.8, n.3, p.532-548, May 1997.

[11]ROSADO, Alfredo; BATALLER, M.; SORIA, E.; CALPE, J.; FRANCÉS, J.V. **A new hardware architecture for a general-purpose neuron based on distributed arithmetic and implemented on FPGA devices**. 321-326p. 1998.

[12]CLOUTIER, Jocelyn; SIMARD, P.Y. **Hardware Implementation of the Backpropagation without Multiplication**. Proc. of the IV International on Microelectronics for Neural Networks and Fuzzy Systems, 46-55p. Turim, Italy. Sept. 1994.

[13]ELDREDGE, James G.;HUTCHINGS, B.L. **Density Enhancement of a Neural Network Using FPGAs and Run-Time Reconfiguration**. IEEE Workshop on FPGAs for Custom Computing Machines, 180-188pp. Napa, CA. April,1994.

[14]ELDREDGE, James G.; HUTCHINGS, B.L. **RRANN: A Hardware Implementation of the Backpropagation Algorithm Using Reconfigurable FPGAs**. IEEE International Conference on Neural Networks. Orlando, FL. July, 1994.

[15]MORGAN, Paul; FERGUSON, A; BOLOURI, H. **Cost-performance analysis of FPGA, VLSI and WSI implementations of a RAM-based neural network**. Proc. of the IV International Conference on Microelectronics for Neural Networks and Fuzzy Systems, 235-243p. Turin, Italy. Sept. 1994.

[16]VALENTINA S.; MICHAEL, G.; OLIVER, M. **A Fast FPGA Implementation of a General Purpose Neuron**. Proc. of the Fourth International Workshop on Field Programmable Logic and Applications. Praga, Czech Republic. Sept. 1994.

[17]GSCHWIND, M.; VALENTINA, S.; MAISCHBERGER, O. **Space Efficient Neural Net Implementation**. Proc. of the Second International ACM/SIGDA Workshop on Field Programmable Gate Arrays. Berkeley, CA. Febr., 1994.

*The author Fernando Moraes would like to gratefully acknowledge the continued support of the "Conselho Nacional de Desenvolvimento Científico e Tecnológico" (**CNPq**), Brazil, to this work. Research grant 522939/96-1.*