

Tese

IMPLEMENTAÇÃO DE UMA MATRIZ DE NEURÔNIOS DINAMICAMENTE E PARCIALMENTE RECONFIGURÁVEL PARA DESCRIÇÃO DE TOPOLOGIAS DE REDES NEURAIS ARTIFICIAIS MULTILAYER PERCEPTRONS

Carlos Alberto de Albuquerque Silva

Natal, agosto de 2015

UFRN / Biblioteca Central Zila Mamede.
Catalogação da Publicação na Fonte

Silva, Carlos Alberto de Albuquerque.

Implementação de uma matriz de neurônios dinamicamente reconfigurável para descrição de topologias de redes neurais artificiais *multilayer perceptrons* / Carlos Alberto de Albuquerque Silva. – Natal, RN, 2015.

81 f. : il.

Orientador: Prof. Dr. Adrião Duarte Doria Neto.

Coorientador: Prof. Dr. José Alberto Nicolau de Oliveira

Tese (Doutorado) – Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica.

1. Redes neurais artificiais – Tese. 2. – Tese. 3. Sistemas parcialmente reconfiguráveis – Tese. 3. *Field Programmable Gate Array* (FPGA) – Tese. 4. Redes *multilayer perceptrons* – Tese. I. Doria Neto, Adrião Duarte. II. Oliveira, José Alberto Nicolau de. III. Universidade Federal do Rio Grande do Norte. IV. Título.

RN/UF/BCZM

CDU 004.032.26

CARLOS ALBERTO DE ALBUQUERQUE SILVA

**IMPLEMENTAÇÃO DE UMA MATRIZ DE NEURÔNIOS DINAMICAMENTE
RECONFIGURÁVEL PARA DESCRIÇÃO DE TOPOLOGIAS DE REDES NEURAIS
ARTIFICIAIS MULTILAYER PERCEPTRONS**

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Elétrica da UFRN (Área de Concentração: Engenharia de Computação), como requisito para a obtenção do título de Doutor.

Orientador: Prof. Dr. Adrião Duarte Doria Neto
Co-orientador: Prof. Dr. José Alberto Nicolau de Oliveira

NATAL
2015

CARLOS ALBERTO DE ALBUQUERQUE SILVA

**IMPLEMENTAÇÃO DE UMA MATRIZ DE NEURÔNIOS DINAMICAMENTE
RECONFIGURÁVEL PARA DESCRIÇÃO DE TOPOLOGIAS DE REDES NEURAIS
ARTIFICIAIS MULTILAYER PERCEPTRONS**

**Tese de Doutorado apresentada ao Programa de
Pós-graduação em Engenharia Elétrica da
UFRN (Área de Concentração: Engenharia de
Computação), como requisito para a obtenção
do título de Doutor.**

Aprovada em: _____ / _____ / _____

Prof. Dr. Adrião Duarte Doria Neto
(Orientador)

Prof. Dr. José Alberto Nicolau de Oliveira
(Co-orientador)

Prof. Dr. Jorge Dantas de Melo

Prof. Dr. David Simonetti Barbalho

Prof. Dr. Danniell Lopes

Profa. Dra. Karla Darlene Nepomuceno Ramos

NATAL
2015

Dedico a Deus, por Sua presença em todos os momentos da minha vida, deixando-me mais forte para superar os desafios, a minha amada esposa Simone Karine por todo seu apoio, ajuda e compreensão e a minha amada filha Letícia, que sirva de incentivo para seu crescimento profissional e pessoal. E aos meus pais, que me deram não somente a vida, mas, principalmente, a minha educação e condições para crescer na vida.

AGRADECIMENTOS

Agradeço a Deus, por iluminar e guiar meu caminho durante este percurso.

Aos meus pais, Francisco e Ivanilda, por serem os principais responsáveis por esta conquista e pelo contínuo apoio em todos os momentos, ensinando-me, sobretudo, a importância da construção e coerência de meus próprios valores.

Aos meus irmãos, Alex e Fábio e a meus familiares, pelo amor, carinho, força e por acreditarem em mim.

Ao meu tio José Alberto, pelo apoio, pela ajuda, paciência e por contribuir para este trabalho, ensinando e esclarecendo assuntos relacionados a temática em questão em vários momentos de meus estudos.

A minha esposa, amiga e companheira, Simone Karine, por todo amor, pela ajuda, pelas sábias palavras de conselhos e pelos incentivos prestados.

A minha filha Letícia por seus lindos sorrisos que não me faziam desistir nunca.

Ao meu orientador, Dr. Adrião Duarte Doria Neto, pela orientação com suas ideias, implicações e críticas construtivas, e pela oportunidade de, com suas argumentações científicas e sugestões essenciais para o desenvolvimento deste trabalho, enriquecer meus conhecimentos.

Aos professores Dr. David Simonetti Barbalho e Dr. Jorge Dantas de Melo, pelos comentários e significantes sugestões que favoreceram a construção deste trabalho.

Aos amigos e colegas do Programa de Pós-Graduação em Engenharia Elétrica e de Computação - PPGEEC da UFRN, que contribuíram com sua amizade, partilhando comigo ideias, fomentando discussões que favoreceram meu aprendizado.

Agradeço e ao Programa de Recursos Humanos - PRH14 da Agência Nacional de Petróleo - ANP, pelo auxílio financeiro.

E a todos que me ajudaram, diretamente ou indiretamente, neste percurso.

MUITO OBRIGADO!

Missão dada é missão cumprida.
(Capitão nascimento)

RESUMO

As Redes Neurais Artificiais (RNAs), que constituem uma das ramificações da Inteligência Artificial (IA), estão sendo empregadas como solução para vários problemas complexos existentes nas mais diversas áreas. Para a solução destes problemas, torna-se indispensável que sua implementação seja feita em hardware. Em meio às estratégias a serem adotadas e satisfeitas durante a fase de projeto e implementação das RNAs em hardware, as conexões entre os neurônios são as que necessitam de maior atenção. Recentemente, encontram-se RNAs implementadas tanto em circuitos integrados de aplicação específica (*Application Specific Integrated Circuits - ASIC*) quanto em circuitos integrados configurados pelo usuário, a exemplo dos *Field Programmable Gate Array* (FPGAs), que possuem a capacidade de serem reconfigurados parcialmente, em tempo de execução, formando, portanto, um Sistema Parcialmente Reconfigurável (SPR), cujo emprego proporciona diversas vantagens, tais como: flexibilidade na implementação e redução de custos. Tem-se observado um aumento considerável no uso de FPGAs para a implementação de RNAs. Diante do exposto, propõe-se a implementação de uma matriz de neurônios reconfigurável para a descrição de topologias de redes neurais artificiais *multilayer perceptrons* (MLPs) em FPGA, com a finalidade de favorecer a realimentação e o reuso de processadores neurais (*perceptrons*) usados em uma mesma área do circuito. Propõe-se ainda, uma rede de comunicação capaz de realizar o reuso de neurônios artificiais. A arquitetura do sistema proposto permitirá configurar várias topologias de redes MLPs por meio da reconfiguração parcial do FPGA. Para permitir essa maleabilidade de configurações de RNAs, um conjunto de componentes digitais (*datapath*) e um controlador foram desenvolvidos para executar instruções que definirão cada topologia MLP para a malha neural.

Palavras-chave: Redes Neurais Artificiais. MLP. FPGA. Sistemas Parcialmente Reconfiguráveis.

ABSTRACT

The Artificial Neural Networks (ANN), which is one of the branches of Artificial Intelligence (AI), are being employed as a solution to many complex problems existing in several areas. To solve these problems, it is essential that its implementation is done in hardware. Among the strategies to be adopted and met during the design phase and implementation of RNAs in hardware, connections between neurons are the ones that need more attention. Recently, are RNAs implemented both in application specific integrated circuits's (Application Specific Integrated Circuits - ASIC) and in integrated circuits configured by the user, like the Field Programmable Gate Array (FPGA), which have the ability to be partially rewritten, at runtime, forming thus a system Partially Reconfigurable (SPR), the use of which provides several advantages, such as flexibility in implementation and cost reduction. It has been noted a considerable increase in the use of FPGAs for implementing ANNs. Given the above, it is proposed to implement an array of reconfigurable neurons for topologies Description of artificial neural network multilayer perceptrons (MLPs) in FPGA, in order to encourage feedback and reuse of neural processors (perceptrons) used in the same area of the circuit. It is further proposed, a communication network capable of performing the reuse of artificial neurons. The architecture of the proposed system will configure various topologies MLPs networks through partial reconfiguration of the FPGA. To allow this flexibility RNAs settings, a set of digital components (datapath), and a controller were developed to execute instructions that define each topology for MLP neural network.

Keywords: Artificial Neural Networks. MLP. FPGA. Partially Reconfigurable systems.

LISTA DE ILUSTRAÇÕES

Figura 1 -	Neurônio Artificial.....	17
Gráfico 1 -	Transformação afim produzida pela presença de um bias.....	19
Figura 2 -	Neurônio artificial e o bias como sendo peso da sinapse (w_0).....	19
Figura 3 -	Rede Neural Artificial com camada única alimentada adiante.....	21
Figura 4 -	Rede alimentada adiante com uma camada oculta e uma camada de saída...	22
Figura 5 -	Rede recorrente.....	23
Figura 6 -	Flexibilidade versus desempenho dos sistemas programáveis.....	28
Figura 7 -	Benefícios da computação reconfigurável.....	29
Figura 8 -	Estrutura dos primeiros PLD; (a) PLA; (b) PAL.....	30
Figura 9 -	Arquitetura básica de uma CPLD.....	32
Figura 10 -	Arquitetura dos CPLDs recentes.....	33
Figura 11 -	Aumento da capacidade lógica e memória embarcada nos FPGAs em uma década.....	34
Figura 12 -	Arquitetura geral de um FPGA.....	35
Figura 13 -	Recursos de Roteamento em um FPGA.....	37
Figura 14 -	Modelos de áreas reconfiguráveis. (a) e (b) 2D; (c) e (d) 1D.....	38
Figura 15 -	Analogia para a reconfiguração parcial e dinâmica de um FPGA.....	41
Gráfico 2 -	Função sigmóide com base na aproximação linear por partes.....	51
Gráfico 3 -	Aproximação linear otimizada (vermelho) vs sigmóide.....	52
Gráfico 4 -	Aproximação PLAN (vermelho) vs sigmóide.....	53
Gráfico 5 -	Aproximação de 2ª ordem vs sigmóide.....	54
Figura 16 -	Fluxo da reconfiguração parcial.....	58
Figura 17 -	Diagrama de Blocos.....	59
Figura 18 -	bloco NEURON.....	60
Figura 19 -	bloco NEURONNET.....	61
Figura 20 -	bloco NEURONFNET.....	62
Figura 21 -	Rede de conexão.....	63
Figura 22 -	Arquitetura completa.....	64
Figura 23 -	Controlador (MdE).....	65
Figura 24 -	Datapath.....	67
Gráfico 6	Função <i>sinc</i>	70
Figura 25 -	Imagens usadas na classificação de padrões pela rede MLP.....	71
Figura 26 -	Topologia da rede para a resolução do problema do XOR.....	71
Figura 27 -	Topologia da rede em hardware para a resolução do problema do XOR.....	72
Figura 28 -	Resultado da simulação do XOR.....	73
Figura 29 -	Partição estática da malha neural.....	74

LISTA DE TABELAS E QUADROS

Quadro 1 -	Funções de ativação e seus respectivos gráficos.....	20
Quadro 2 -	Evolução dos PLDs.....	32
Quadro 3 -	Características das Tecnologias de Programação.....	36
Tabela 1 -	Aproximação PLAN.....	53
Tabela 2 -	Comparativos de áreas ocupadas pelo Bloco NEURON e pela rede MLP....	68
Tabela 3 -	Comparativos de áreas ocupadas pelo Bloco NEURON e pela rede MLP....	69
Quadro 4 -	Resultados comparativos da malha neural utilizada para execução da função XOR.....	72
Quadro 5 -	Taxas de ocupação do FPGA para a partição estática.....	73
Quadro 6 -	Taxas de ocupação do FPGA para a partição reconfigurável.....	73

LISTA DE SIGLAS E ABREVIATURAS

ASIC	<i>Application Specific Integrated Circuits</i>
CAD	<i>Computer Aided Design</i>
CI	Circuitos Integrados
CLB	<i>Configurable Logic Block</i>
CPLD	<i>Complex PLD</i>
CPU	<i>Central Processing Units</i>
CR	Computação Reconfigurável
DCM	<i>Digital Clock Manager</i>
DSP	<i>Digital Signal Processing</i>
EEPROM	<i>Electrically-Erasable Programmable Read-Only Memory</i>
EMAC	<i>Cores Ethernet Media Access Control</i>
EMC	<i>External Memory Controller</i>
FF	<i>Foundation Fieldbus</i>
FPGA	<i>Field Programmable Gate Array</i>
FSL	<i>Fast Simplex Link</i>
GAL	<i>Generic Array Logic</i>
HDL	<i>Hardware Description Language</i>
HF	Hardware Fixo
HP	Hardware Programável
I/O	<i>Input/Output</i>
IA	Inteligência Artificial
ICAP	<i>Internal Configuration Access Port</i>
IOB	<i>Input/Output Block</i>
ISP	<i>In System Programability</i>
JTAG	<i>Joint Test Action Group</i>
RTL	<i>register-transfer level</i>
LUT	<i>Look Up Tables</i>
MdE	Máquina de Estados
MGR	matriz geral de rotas
MGT	<i>Multi-Gigabit Transceiver</i>
MLP	<i>Multilayer Perceptron</i>
PAL	<i>Programmable Array Logic</i>
PALCE	<i>PAL CMOS electrically erasable/programmable device</i>
PLA	<i>Programmable Logic Array</i>
PLDs	<i>Programmable Logic Divices</i>
RAM	<i>Random Access Memory</i>
RDP	Reconfiguração Dinâmica parcial
RNAs	Redes Neurais Artificiais
RTR	<i>Run-Time Reconfiguration</i>
SPLDs	<i>Simple PLDs</i>
SPR	Sistemas Parcialmente Reconfiguráveis
SRAM	<i>Static Random Access Memory</i>
SW	Software
UART	<i>Universal Asynchronous Receiver/Transmitter</i>
ULA	Unidade Lógica Aritmética

SUMÁRIO

1	INTRODUÇÃO	12
2	REDES NEURAIS ARTIFICIAIS	16
2.1	NEURÔNIO ARTIFICIAL.....	17
2.1.1	Tipos de função de ativação	20
2.2	ARQUITETURAS DE REDES NEURAIS.....	21
2.2.1	Rede neural artificial alimentada adiante com camada única	21
2.2.2	Rede neural artificial com múltiplas camadas	22
2.2.3	Rede recorrente	23
2.3	PROCESSOS DE TREINAMENTO E ASPECTOS DE APRENDIZAGEM.....	23
2.3.1	Aprendizado supervisionado por retropropagação do erro	24
2.4	ALGORITMOS INTELIGENTES E REDES NEURAIS.....	25
3	COMPUTAÇÃO RECONFIGURÁVEL	27
3.1	DISPOSITIVOS RECONFIGURÁVEIS.....	29
3.2	CPLDs (Complex PLDs).....	32
3.3	FPGAs (Field Programmable Gate Arrays).....	33
3.3.1	Tecnologias de Programação	35
3.3.2	Blocos Lógicos Configuráveis (Configurable Logic Block – CLB)	36
3.3.3	Recursos de Roteamento	37
3.3.4	Áreas reconfiguráveis	38
3.3.5	Tipos de Reconfiguração	38
3.3.6	Dispositivos Xilinx Virtex-6	42
4	REDES NEURAIS ARTIFICIAIS EM FPGAs	45
4.1	CLASSIFICAÇÃO DAS REDES NEURAIS IMPLEMENTADAS EM FPGA.....	45
4.1.1	Algoritmo de Aprendizagem	46
4.1.2	Representação dos Dados	47
4.1.3	Estratégias de redução dos Multiplicadores	48
4.1.4	Aproximação da função de ativação	49
4.1.5	Paralelismo em redes neurais	54
4.1.5.1	Dispositivos Paralelos.....	55
5	MATERIAIS E MÉTODOS	57
5.1	FLUXO DE PROJETO PARA RECONFIGURAÇÃO PARCIAL.....	58
5.2	DESCRIÇÃO ESTRUTURAL DO NEURÔNIO EM FPGA.....	59
5.3	REDE DE CONEXÃO.....	62
5.4	ARQUITETURA COMPLETA.....	64
5.4.1	Controlador	65
5.4.2	Datapath	66
6	RESULTADOS E DISCUSSÕES	68
	CONCLUSÃO	75
	REFERÊNCIAS	77

1 INTRODUÇÃO

No cenário industrial contemporâneo, projetar, construir e gerenciar “informação” como estratégia de apoio no monitoramento de sistemas e no controle de processos é de extrema importância para a indústria como um todo, sobretudo em áreas onde há processos que ocorrem sob incertezas e com dados incompletos. Lidar com tal estratégia tem exigido dos sistemas computacionais respostas em intervalos de tempo cada vez menores ou até mesmo em tempo real. Observa-se também que algumas aplicações, ao processarem “informação”, apresentam uma grande demanda por recursos computacionais quando executadas em soluções sequenciais.

Neste contexto, os requisitos de desempenho de muitas aplicações não são atendidos quando executadas em sistemas computacionais convencionais, que possuem processadores *single-core* ou *multi-core*, ambos, baseados na arquitetura de Von Neumann. Essa arquitetura não incorpora nenhum tipo de inteligência nas ações da máquina, ela apenas executa os comandos que lhe são transmitidos em forma de algoritmo (MONTEIRO, 2007).

Caso houvesse a possibilidade de incorporar a extração e o armazenamento de conhecimento nas arquiteturas de computadores, estas poderiam interpretar e relacionar dados e instruções por meio de decisões inteligentes. Com isso, as arquiteturas passariam a apresentar um comportamento similar ao do cérebro humano. Foi buscando essa semelhança com o cérebro humano que, em 1943, surgiu o primeiro trabalho sobre os sistemas conexionistas, fruto das pesquisas realizadas por Warren McCulloch e Walter Pitts. (BRAGA; CARVALHO; LUDEMIR, 2007; HAGAN; DEMUTH; BEALE, 1996).

Estes sistemas baseiam-se na crença de que o comportamento inteligente só pode ser alcançado por meio do maciço processamento paralelo e distribuído, tal qual acontece nas conexões neurais do sistema nervoso central dos seres humanos. Esta linha conexionista é praticada por meio da construção de sistemas baseados em Redes Neurais Artificiais (RNA) (HAYKIN, 2001).

Inspiradas no processamento em paralelo distribuído e nas conexões do sistema neural biológico, as RNAs constituem um dos paradigmas computacionais para a concepção de sistemas computacionais capazes de simular tarefas intelectuais complexas, diferentemente dos paradigmas computacionais convencionais que são constituídos por uma lógica sequencial ou por uma lógica paralela ao nível de *threads* (TAFNER, 2010).

Assim, as RNA são sistemas paralelos distribuídos formados por unidades de processamento simples, denominadas de neurônios artificiais, que calculam determinadas funções matemáticas, normalmente não-lineares (BRAGA; CARVALHO; LUDEMIR, 2007).

Diante do exposto, parece natural que as RNA sejam desenvolvidas para algum tipo de arquitetura paralela. Entretanto, a sua implementação é comumente praticada em máquinas que adotam a arquitetura de Von Neumann, fazendo uso de todos os benefícios deste modelo (BRAGA, 2005). Por outro lado, este modelo apresenta algumas limitações decorrentes da execução sequencial das instruções e das limitações físicas dos barramentos que ligam o processador com a memória (STALLINGS, 2002).

Devido às limitações impostas pelo modelo de Von Neumann, muitas implementações de RNA passaram a ser desenvolvidas em hardware, buscando assim, alcançar o paralelismo intrínseco destas redes. Um exemplo de hardware usado nas implementações das RNA é o *Field Programmable Gate Array* (FPGA).

O FPGA apresenta-se como uma solução intermediária que combina a velocidade do hardware dedicado com a flexibilidade alcançada com o software, buscando preencher os espaços existentes entre o software e o hardware, a fim de atingir um desempenho maior que o da solução por software, enquanto mantém um nível de flexibilidade maior que o do hardware. É um tipo de circuito integrado programável pelo usuário, via software, e pode ser utilizado tanto em um produto final quanto para fins de prototipagem, apresentando vantagens competitivas, principalmente, ao reduzir o tempo de lançamento de novos produtos no mercado.

Como exemplos de produção científica referentes à temática em estudo, pode-se referenciar, no âmbito internacional, o trabalho desenvolvido por Bako *et al.* (2012), que apresenta a implementação de uma RNA em FPGA para classificação do sinal EEG. Em seu desenvolvimento, o autor faz uso da técnica *Codesign* (software/hardware), juntamente, com a técnica de reconfiguração parcial do FPGA. Finker *et al.* propõem, em seu trabalho, o desenvolvimento de um neurônio artificial capaz de modificar seus parâmetros em tempo real, tais como: números de entradas e funções de ativação. Para isso, os autores fazem uso da técnica de reconfiguração parcial do FPGA da Xilinx. Já, no âmbito nacional, têm-se os trabalhos de Lopes (2009) e de Prado *et al.* No primeiro trabalho o autor apresenta uma abordagem para a implementação de arquiteturas complexas de redes neurais em FPGA, fazendo uso de vários processadores em um único chip. Já no segundo trabalho os autores propõem uma arquitetura em hardware, descrita em VHDL, desenvolvida para embarque de RNA do tipo *Multilayer Perceptron* (MLP), com possibilidade de reuso de processadores

neuronal (neurônios artificiais – *Perceptrons*), que utilizam sempre a mesma área de silício.

Diante desta conjuntura, pode-se observar a relevância desta temática, visto que o estudo sobre RNAs embarcadas em FPGAs atuando como sistemas dinamicamente reconfiguráveis tornam-se, portanto, soluções promissoras para diversas aplicações e sistemas devido ao seu potencial de redução da área do silício.

O emprego dos FPGAs como plataforma para a implementação de RNA em circuitos integrados, tem permitido explorar o alto poder de processamento, o baixo custo, a facilidade de programação e a capacidade de reconfiguração do circuito, permitindo que a rede se adapte a diferentes aplicações. No entanto, existem obstáculos para uma adoção mais genérica desse tipo de implementação, os quais estão relacionados ao desenvolvimento do neurônio artificial, juntamente com suas estruturas internas: multiplicadores em paralelo; funções de ativação e; de outras partes que eventualmente possam ser necessárias, dentre elas a implementação de um bloco multiplicador acumulador. (SILVA, 2010).

Para contornar tais obstáculos e conseguir mais flexibilidade na configuração topológica das redes neurais, faz-se necessário a utilização de FPGAs que possibilitem a reconfiguração dinâmica e parcial das funções e interconexões neuronais. A reconfiguração dinâmica parcial (RDP) se dá de forma seletiva, em qualquer porcentagem dos recursos reconfiguráveis do FPGA, em qualquer instante de tempo, e com o dispositivo funcionando, permitindo a reconfiguração de uma parte do circuito sem prejudicar o funcionamento do restante.

Diante do exposto, este trabalho tem como objetivo implementar uma matriz de neurônios dinamicamente e parcialmente reconfigurável para a descrição de topologias de redes neurais artificiais *multilayer perceptron* em FPGA, com a finalidade de favorecer a realimentação e o reuso de processadores neurais usados em uma mesma área do circuito.

Em seu delineamento, esta tese está estruturada em sete seções, que guardam, em si, peculiaridades, e a coerência necessária às exigências da investigação, de modo a preservar sua coesão.

Na primeira seção realiza-se a apresentação formal da proposta de estudo, sua fundamentação, seu objetivo e seus direcionamentos.

Na segunda seção é apresentada a base teórica para compreensão das Redes Neurais Artificiais, os paradigmas de aprendizagem, suas arquiteturas, os tipos de função de ativação e trabalhos a elas relacionados.

Na seção três são apresentados os conceitos básicos da computação reconfigurável, alguns paradigmas, suas vantagens e desvantagens; são analisados alguns dispositivos

reprogramáveis, em especial o FPGA, a metodologia utilizada em sua programação e os tipos de reconfiguração.

Na seção quatro são abordadas algumas implementações de redes neurais artificiais em FPGAs que permitem reconfiguração parcial.

Já na seção cinco, encontra-se o detalhamento da arquitetura do sistema embarcado proposto, o seu desenvolvimento e as otimizações realizadas.

Na seção seis são apresentados os dados dos testes, simulações e resultados, das implementações realizadas na arquitetura proposta.

A seção sete encerra a tese, com a apresentação da conclusão, das contribuições, das dificuldades encontradas para a implementação da arquitetura da malha neural, das soluções adotadas e sugestões para trabalhos futuros, que poderão dar continuidade a este trabalho, tendo como base o neurônios artificial e a rede de conexão desenvolvidos em hardware.

2 REDES NEURAIS ARTIFICIAIS

O cérebro humano é um órgão que consiste aproximadamente de 100 bilhões de neurônios que são conectados por cerca de 100 trilhões de conexões, formando assim, o sistema mais complexo que se tem conhecimento. As conexões existentes no cérebro permitem formar uma rede de neurônios, capaz de processar maciças informações em paralelo e distribuídas.

Esta rede de neurônios, ou rede neural, tem a responsabilidade de realizar o processamento das funções cognitivas básicas, assim como, a execução das funções sensorimotoras e autônomas. Além disso, esta rede tem a capacidade de reconhecer padrões e relacioná-los, usar e armazenar conhecimento por experiência, além de interpretar informações. No entanto, a estrutura individual dos neurônios, a topologia de suas conexões e o comportamento em conjunto desses elementos biológicos formam a base para o estudo das Redes Neurais Artificiais (RNAs).

Pesquisas envolvendo RNAs são motivadas, desde o início, pela importância do cérebro humano no processamento das informações, as quais são completamente distintas das processadas por um computador digital convencional. O cérebro pode ser visto como um computador altamente complexo, não-linear e paralelo, o qual possui a capacidade de organizar os seus elementos estruturais, os neurônios, de forma a executar seus processamentos de modo mais eficiente (HAYKIN, 2001). Já as RNAs tentam reproduzir as funções das redes biológicas, buscando implementar seu comportamento funcional e sua dinâmica.

Inspiradas no processamento em paralelo distribuído e nas conexões do sistema neural biológico, as RNAs constituem um dos paradigmas computacionais, no campo da Inteligência Artificial (IA), para a concepção de sistemas computacionais capazes de simular tarefas intelectuais complexas, diferentemente dos paradigmas convencionais que são constituídos por uma lógica sequencial (modelos de Von Neumann) (TAFNER, 2010).

As RNAs são sistemas de processamentos paralelos distribuídos, formados por unidades neuronais (neurônios artificiais), que, trabalhando em conjunto, são capazes de resolver problemas matemáticos de características não-lineares. Tais unidades são alocadas em uma ou mais camadas, com um grau de interconexão similar à estrutura cerebral, cuja transferência de informação entre as unidades ocorre em tempos específicos, dentro de uma margem de sincronização (NASCIMENTO; YONEYAMA, 2004; BRAGA; CARVALHO; LUDERMIR, 2007).

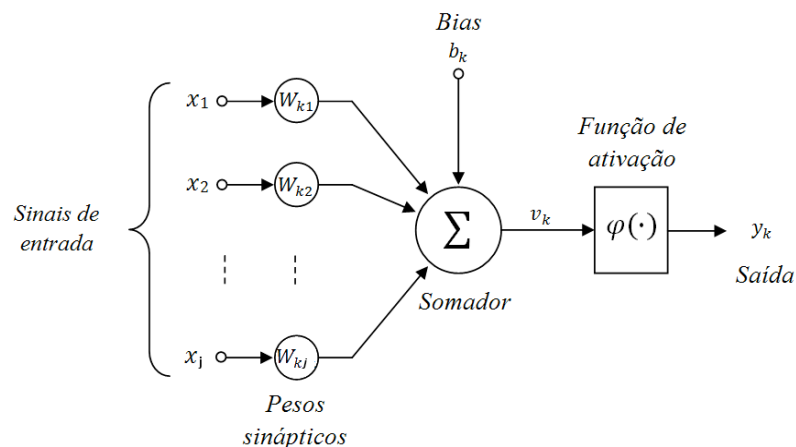
O poder computacional intrínseco em uma RNA é extraído, inicialmente, de sua estrutura maciçamente paralela. E, posteriormente, da sua capacidade de aprender e de generalizar a informação aprendida. Ou seja, a rede aprende a partir de um conjunto reduzido de padrões e produz respostas coesas para os dados que não estavam presentes na fase de treinamento. A auto-organização, o mapeamento de entrada-saída, a adaptabilidade e o processamento temporal são outras características presentes em uma rede neural que, aliadas às anteriores, fazem dela uma ferramenta hábil na resolução de problemas complexos (HAYKIN, 2001; BRAGA; CARVALHO; LUDERMIR, 2007).

2.1 NEURÔNIO ARTIFICIAL

As Redes Neurais Artificiais (RNAs) são idealizadas, até hoje, a partir do conhecimento adquirido ao longo dos anos sobre os sistemas nervosos biológicos e do próprio cérebro humano. Por essa razão, as unidades computacionais denominadas neurônios artificiais são modelos que visam representar, de maneira simplificada, os neurônios biológicos por meio de uma formulação matemática.

O modelo matemático de neurônio artificial mais simples, que engloba as características essenciais de uma rede neural biológica, isto é, o paralelismo e a alta conectividade, foi proposto e desenvolvido pelos pesquisadores McCulloch e Pitts (1943) (Figura 1), sendo ainda o mais empregado nas diferentes arquiteturas de RNAs (HAYKIN, 2001).

Figura 1 – Neurônio Artificial.



Fonte: HAYKIN, 2001, pág. 36.

Conforme o modelo proposto na Figura 1, são identificados três elementos básicos em

sua composição:

1. Um conjunto de *sinapses*, com cada uma sendo caracterizada por um peso. Assim, para todo sinal x_j na entrada da *sinapse* j vinculada ao neurônio k será ponderado um peso sináptico w_{kj} . Com relação ao peso sináptico w_{kj} , é importante observar que o primeiro índice refere-se ao neurônio em questão, enquanto que o segundo faz alusão ao terminal de entrada da sinapse ao qual o peso se refere;
2. Um combinador linear, empregado para executar o somatório dos sinais produzidos pelo produto dos pesos sinápticos pelas entradas fornecidas ao neurônio;
3. Uma função de ativação, responsável por associar o sinal resultante do combinador linear, conhecido como potencial de ativação, a um valor de saída, podendo aplicar não-linearidade ou restrição. Essa função define, ainda, uma restrição ao intervalo permissível de amplitude do sinal de saída a um valor finito. Geralmente essa limitação fica entre $[0, 1]$ ou $[-1, 1]$, por questão de normalização da informação (HAYKIN, 2001).

No neurônio artificial da Figura 1 é possível identificar, também, um *bias* representado por b_k , o qual é empregado para aumentar ou diminuir a entrada líquida da função de ativação, dependendo de seu valor positivo ou negativo. Assim, a descrição matemática de um neurônio artificial k pode ser representada pelas equações 1 e 2:

$$u_k = \sum_{j=1}^m w_{kj} x_j \quad (1)$$

e

$$y_k = \varphi(u_k + b_k) \quad (2)$$

com,

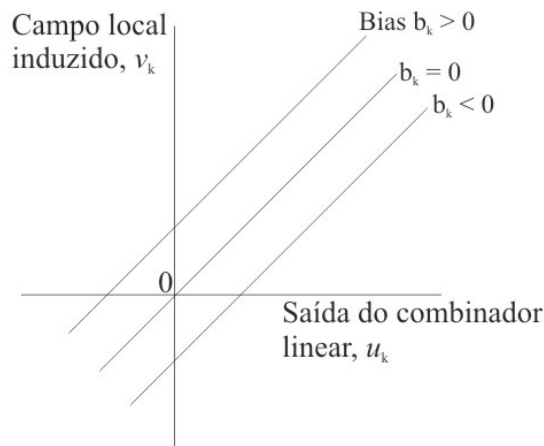
- x_1, x_2, \dots, x_m sendo os sinais de entrada;
- w_1, w_2, \dots, w_{km} sendo os pesos sinápticos do neurônio k ;
- u_k representando a saída do combinador linear;
- b_k representando o *bias*;
- φ representando a função de ativação;
- y_k sendo o sinal de saída do neurônio.

O emprego do *bias* b_k tem a finalidade de realizar uma transformação afim à saída u_k

do combinador linear do neurônio artificial da Figura 1, demonstrado na Equação 3. Mesmo que o valor do *bias* b_k seja positivo ou negativo, a relação entre o potencial de ativação v_k do neurônio k e a saída do combinador linear u_k é alterada na forma mostrada no Gráfico 1. Com isso, pode-se notar, como resultado da transformação afim, que o gráfico de v_k em função de u_k não passa mais pela origem.

$$v_k = u_k + b_k \quad (3)$$

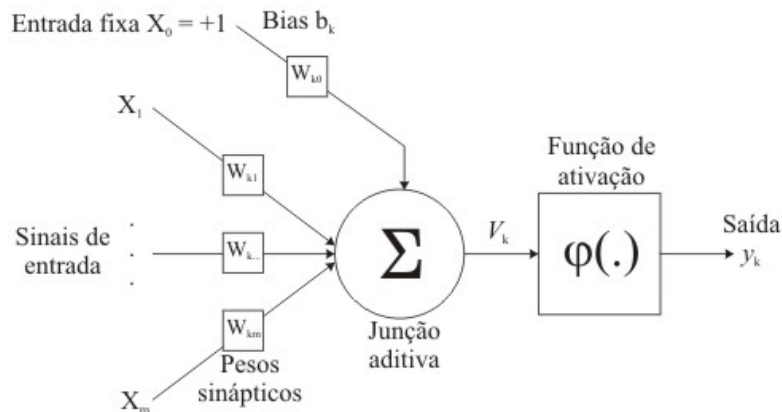
Gráfico 1 – Transformação afim produzida pela presença de um *bias*.



Fonte: HAYKIN, 2001, pág. 37.

Pode-se representar o neurônio artificial k de uma outra forma. Para isto, considera-se o *bias* como sendo o peso da sinapse (w_0), que receberá como sinal de entrada um valor fixo em “+1”, visualizado na Figura 2. Essa forma simplifica a representação matemática do modelo do neurônio artificial e a implementação dos algoritmos de aprendizado.

Figura 2 – Neurônio artificial e o *bias* como sendo peso da sinapse (w_0).

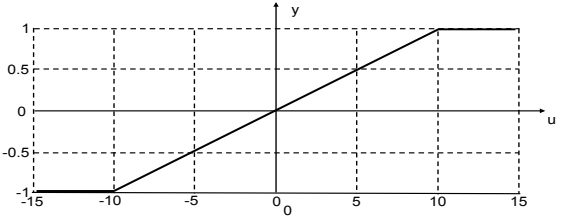
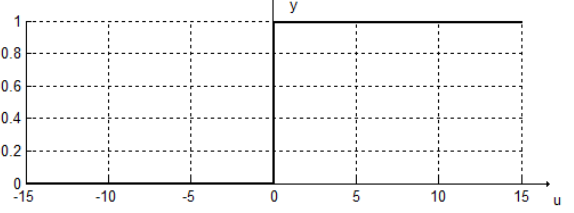
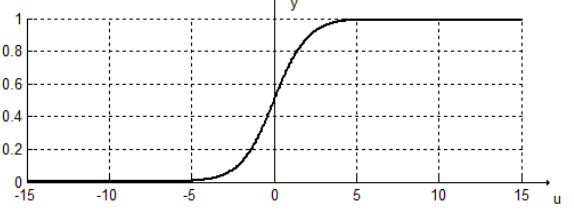
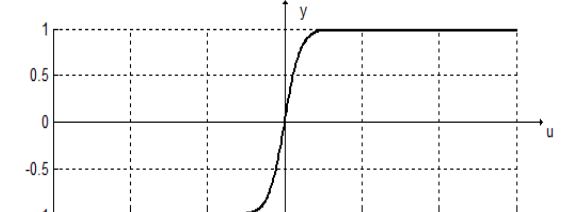


Fonte: HAYKIN, 2001, pág. 38.

2.1.1 Tipos de função de ativação

O valor de saída do neurônio artificial, em termos de campo local induzido, é determinado por uma função de ativação, representada por $\varphi(v)$, cujo propósito é manter a saída do neurônio artificial dentro do domínio de definição da mesma. Tal função pode assumir várias formas dentre os dois grupos principais, isto é, funções parcialmente diferenciáveis e funções totalmente diferenciáveis. No Quadro 1, pode-se visualizar algumas das principais funções de ativação com seus respectivos gráficos.

Quadro 1 – Funções de ativação e seus respectivos gráficos.

Função de Ativação	Gráfico da Função de Ativação
Rampa simétrica $\varphi(v) = \begin{cases} a, & \text{se } v_k > a \\ v, & \text{se } a \leq v_k \leq a \\ -a, & \text{se } v < a \end{cases}$	 <p>(a) Gráfico de ativação rampa simétrica.</p>
Limiar $\varphi(v) = \begin{cases} 1, & \text{se } v_k \geq 0 \\ 0, & \text{se } v_k < 0 \end{cases}$	 <p>(b) Gráfico de ativação degrau bipolar.</p>
Sigmóide $\varphi(v) = \frac{1}{1+\exp(-av)}$	 <p>(c) Gráfico de ativação logística.</p>
Tangente hiperbólica $\varphi(v) = \frac{1-\exp(-av)}{1+\exp(-av)}$	 <p>(d) Gráfico de ativação tangente hiperbólica.</p>

No Quadro 1, as funções de ativação parcialmente diferenciáveis, representadas pelos Gráficos (a) e (b), são aquelas que possuem pontos cujas derivadas de primeira ordem são

inexistentes. Entretanto, as funções de ativação totalmente diferenciáveis, consideradas nos Gráficos (c) e (d), apresentam derivadas de primeira ordem que são conhecidas em todos os pontos de seu domínio de definição (SILVA, 2010).

2.2 ARQUITETURAS DE REDES NEURAIIS

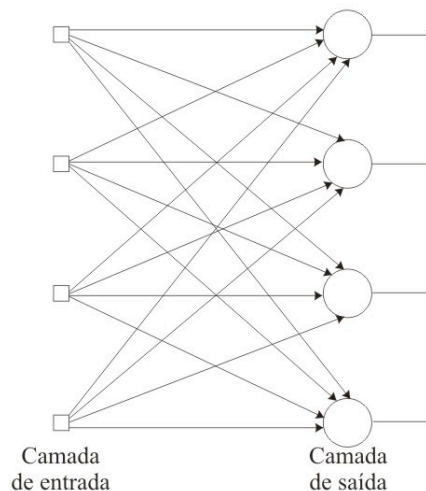
A questão fundamental na concepção de uma RNA, para uma dada aplicação ou resolução de problema, é a escolha de sua arquitetura. Nessa escolha, define-se a forma como seus neurônios artificiais devem ficar dispostos uns em relação aos outros. Essas disposições são essencialmente estruturadas através do direcionamento das conexões sinápticas dos neurônios artificiais (HAYKIN, 2001).

As principais arquiteturas RNAs, considerando a disposição de seus neurônios, assim como as formas de interligação entre eles e a constituição de suas camadas, podem ser analisadas a seguir.

2.2.1 Rede neural artificial alimentada adiante com camada única

Essa RNA, ilustrada na Figura 3, caracteriza-se por apresentar uma arquitetura com uma camada de entrada de dados, seguida por uma única camada de neurônios, que é responsável pelo processamento dos sinais de entrada e pela geração dos sinais de saída. O fluxo de informações é unidirecional, ou seja, passa da camada de entrada em direção à camada de saída. A nomenclatura não considera a camada de entrada, porque não existe nela a possibilidade de computação de dados.

Figura 3 – Rede Neural Artificial com camada única alimentada adiante.



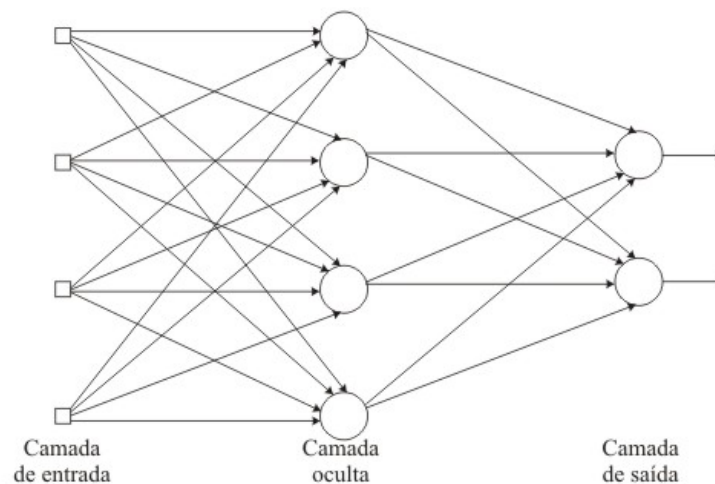
Fonte: HAYKIN, 2001, pág. 36.

2.2.2 Rede neural artificial com múltiplas camadas

Diferentemente das RNAs pertencentes à arquitetura anterior, as RNAs dessa classe são conhecidas como *Perceptrons* multicamadas (*multilayer Perceptron* - MLP), por admitirem a presença de uma ou mais camadas ocultas e nós computacionais, chamados de neurônios ocultos. Esses neurônios têm a função de intervir entre a entrada externa e a saída da rede de modo útil. Acrescentando-se uma ou mais camadas ocultas, a rede torna-se apta a extrair estatísticas de ordem elevada de algum desconhecido processo aleatório subjacente, responsável pelo comportamento dos dados de entrada, processo sobre o qual a rede está tentando adquirir conhecimento. A RNA adquire uma perspectiva global do processo aleatório, apesar de sua conectividade local, em virtude do conjunto adicional de pesos sinápticos e da dimensão adicional de interações neurais proporcionada pelas camadas escondidas.

Para esse tipo de arquitetura, os sinais de entrada da rede são fornecidos pelos nós de fonte da camada de entrada aos neurônios da primeira camada oculta, que, por sua vez, processarão esses sinais produzindo as saídas que servirão como entrada para a próxima camada oculta ou para a camada de saída. O conjunto de saídas produzido pelos neurônios da camada de saída estabelece a resposta global da rede. A Figura 4 representa o esquema de uma rede neural com a presença de uma camada oculta.

Figura 4 – Rede alimentada adiante com uma camada oculta e uma camada de saída.



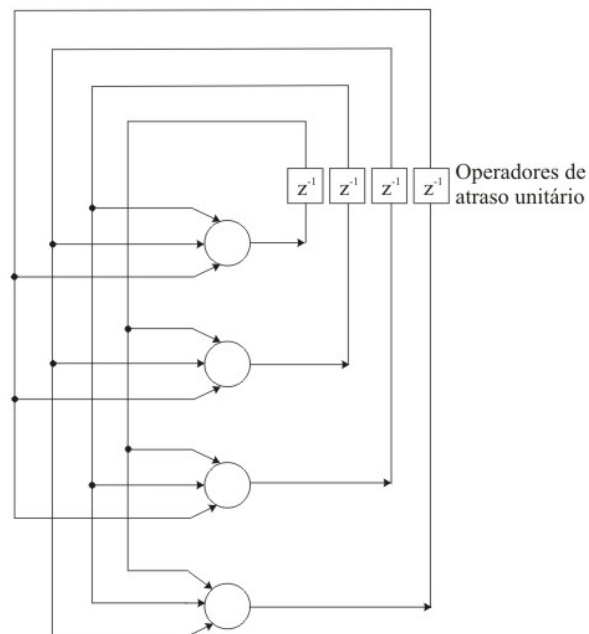
Fonte: HAYKIN, 2001, pág. 48.

2.2.3 Rede recorrente

É uma arquitetura de RNAs que se distingue das demais, descritas anteriormente, por apresentar laços de realimentação, com ou sem camadas ocultas. A presença desses laços tem um impacto intenso na capacidade de aprendizagem e de desempenho da rede. Além disso, os laços de realimentação compreendem o uso de ramos particulares associados com elementos de atraso unitário, resultando, portanto, em um processamento dinâmico de informações.

Assim, por intermédio dos laços de realimentação, as RNAs com este tipo de arquitetura geram saídas atuais levando-se também em consideração os valores das saídas anteriores. Essa arquitetura de rede é ilustrada na Figura 5.

Figura 5 – Rede recorrente.



Fonte: HAYKIN, 2001, pág. 48.

2.3 PROCESSOS DE TREINAMENTO E ASPECTOS DE APRENDIZAGEM

Uma das propriedades mais relevantes de uma rede neural artificial é a sua capacidade de aprender a partir da apresentação de amostras que expressam o comportamento do sistema e de generalizar a informação aprendida. Assim, a rede será então capaz de gerar uma saída próxima da desejada a partir de quaisquer dados introduzidos em suas entradas. Isto é realizado através de um processo iterativo de ajustes empregados aos seus parâmetros livres (pesos sinápticos e níveis de *bias*). O término do aprendizado ocorre quando a rede neural obtém uma solução generalizada para uma classe de problemas, segundo algum critério

preestabelecido.

Quanto ao método de aprendizagem, as RNAs se classificam em dois paradigmas principais: aprendizado supervisionado e aprendizado não-supervisionado.

O aprendizado supervisionado caracteriza-se pela existência de um agente externo, responsável por apresentar um conjunto de amostras dos sinais de entrada e a respectiva saída desejada, analisando a saída calculada pela rede e checando-a com a saída desejada. Caso a saída da rede não seja igual ou próxima ao valor desejado, os pesos sinápticos e níveis de *bias* precisam ser ajustados iterativamente, sob a influência do sinal de erro, de forma a minimizar a diferença entre a saída gerada pela rede e a saída desejada.

Já no aprendizado não-supervisionado, não há necessidade do agente externo para verificar o processo de aprendizado. Nesse processo, apenas os padrões de entrada são fornecidos para a rede, ao contrário do aprendizado supervisionado, cujos padrões de treinamento possuem pares de entrada e saída. Durante esse aprendizado, verifica-se a existência de regularidades nos dados de entrada, por meio de suas características estatisticamente mais relevantes (BRAGA; CARVALHO; LUDERMIR, 2007).

O término do método de aprendizagem se dá, basicamente, quando a RNA encontra uma solução para todas as entradas fornecidas. Entretanto, outros parâmetros, também, podem ser levados em consideração, como, por exemplo, a definição de uma taxa de erro a ser atingida ou o número máximo de ciclos.

2.3.1 Aprendizado supervisionado por retropropagação do erro

Em relação aos diversos algoritmos de aprendizado supervisionado vigentes na literatura, o algoritmo adotado pelo presente trabalho foi o da retropropagação do erro (*back-propagation*), que é o mais aplicado para treinar as redes *perceptrons* de múltiplas camadas.

O processo de desenvolvimento do algoritmo *back-propagation* envolve duas etapas. A primeira delas ocorre após as entradas serem apresentadas à rede e propagadas adiante até que um valor de saída seja computado pela última camada da rede. Esse valor deverá ser comparado com o valor da saída desejada a fim de minimizar o erro da resposta atual em relação à saída desejada.

Na segunda etapa, o erro deverá ser propagado em direção à camada de entrada, ou seja, à camada contrária à de saída. Nessa etapa, o objetivo principal é a atualização dos parâmetros livres (pesos sinápticos), por meio da retropropagação do erro, podendo, ao término das duas etapas, serem apresentadas novas entradas à rede neural.

2.4 ALGORITMOS INTELIGENTES E REDES NEURAIAS

A Literatura da área dissemina várias pesquisas focadas na formulação de algoritmos fundamentados em técnicas de inteligência artificial, capazes de armazenar o conhecimento, aplicá-lo na resolução de problemas e adquirir novos conhecimentos por meio da experiência. Com relação a esses três aspectos, destaca-se o desenvolvimento de sistemas inteligentes baseados em redes neurais artificiais.

No entanto, os sistemas inteligentes, frequentemente, necessitam de uma rede de dispositivos distribuídos como, por exemplo, sensores, atuadores e controladores, que englobem amplas funcionalidades em termos de comunicação da rede, de integração dos seus elementos e do processamento das informações obtidas. Esses dispositivos devem empregar interfaces padronizadas e protocolos de comunicação, resultando assim, em dispositivos reconfiguráveis e dotados de autonomia (CAGNI JÚNIOR, 2007). Um exemplo de dispositivo inteligente são os *Software Sensors*, que empregam códigos computacionais capazes de inferir valores a partir de medidas de diferentes sensores, realizando algum cálculo ou até mesmo algum tipo de controle.

Além disso, o trabalho publicado por Silva (2005) expõe o desenvolvimento de uma solução capaz de executar uma rede neural artificial no ambiente de redes, para a automação industrial *Foundation Fieldbus* (FF), baseado em blocos funcionais, para a compensação das medidas de temperatura através de *Software Sensors*.

Com relação à área de detecção de falhas, destaca-se o trabalho realizado por Fernandes *et al.* (2006), no qual foi desenvolvido um sistema inteligente firmado em redes neurais artificiais, para a verificação e a classificação de falhas no controle de nível, em uma planta composta de dois tanques, cuja escoação gravita de um tanque para o outro. Três redes neurais foram desenvolvidas, duas delas simulando os tanques de nível, e a terceira recebendo os sinais de saída das redes iniciais.

Além desse estudo, há, também, o trabalho publicado por Cagni Júnior (2007), que consiste na descrição de um sistema composto por uma placa processadora de sinais (DSP) e um supervisor, cuja principal finalidade é a correção dos dados aferidos por um medidor de vazão do tipo turbina. A correção é realizada por um algoritmo inteligente baseado em uma rede neural artificial.

Esta seção apresentou a base teórica sobre RNAs, partindo da explanação do neurônio matemático, desenvolvido por McCulloch e Pitts, até sua incorporação em uma arquitetura neural artificial. Além disso, foram teorizados os tipos de função de ativação, as arquiteturas

existentes e os processos de aprendizagem. Outros estudos abordados foram relacionados aos algoritmos inteligentes e às redes neurais artificiais. Na seção seguinte será discutido os conceitos da computação reconfigurável, os quais servirão de base para a construção de uma arquitetura dinamicamente reconfigurável.

3 COMPUTAÇÃO RECONFIGURÁVEL

O crescente aumento do uso da computação em quase todas as áreas do conhecimento, sobretudo nas relacionadas às ciências exatas e engenharias, vem proporcionando soluções para problemas complexos, os quais geram uma grande demanda de recursos computacionais necessários para o armazenamento, a recuperação, a transmissão e o processamento de informações. Além disso, para muitos desses problemas, as soluções devem ser alcançadas em intervalos de tempo cada vez menores ou mesmo em tempo real (MESQUITA, 2002).

Para boa parte desses problemas complexos, as soluções implementadas podem ser classificadas em dois paradigmas distintos: o primeiro relacionado às soluções implementadas em Hardware Fixo (HF), ou *Hardwired*; e o segundo, relacionado às soluções implementadas em Hardware Programável (HP) através de Software (SW) (MARTINS et al., 2009). Entretanto, ambos os paradigmas apresentam desvantagens relacionadas ao desempenho, à flexibilidade, ao custo.

O paradigma HF apresenta como desvantagens a falta de flexibilidade e a sua subutilização quando se trata de aplicações diversas. Isso impossibilita que os componentes do hardware sejam alterados após sua fabricação, impedindo, assim, adaptações futuras. Em relação ao paradigma de HP, a principal desvantagem está relacionada ao desempenho durante a execução de uma aplicação, o qual, muitas vezes, é insatisfatório.

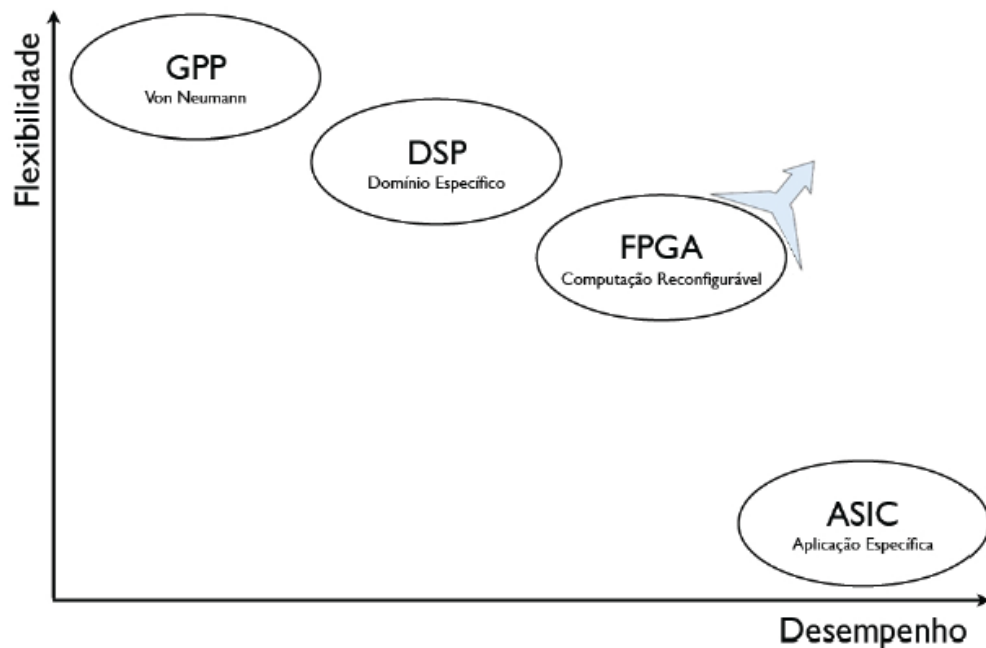
Por causa das desvantagens apresentadas pelos dois paradigmas, a Computação Reconfigurável (CR) apresenta-se como uma solução intermediária que combina a velocidade do hardware com a flexibilidade do software, por meio da abordagem de co-projeto em hardware-software, que define quais partes de um sistema serão implementadas em hardware ou em software (ARAGÃO; ROMERO; MARQUES, 2009).

Para Skliarova e Ferrari (2003), a CR baseia-se em dispositivos lógicos reprogramáveis que podem atingir um desempenho elevado e, ao mesmo tempo, fornecer a flexibilidade da programação em nível de portas lógicas.

Bobda (2007), por sua vez, em termos básicos, afirma que a CR é o estudo da computação que envolve os dispositivos reconfiguráveis, incluindo arquiteturas, algoritmos e aplicações, objetivando preencher os espaços existentes entre o software e o hardware, a fim de atingir um desempenho maior que o da solução por software, enquanto mantém um nível de flexibilidade maior que o do hardware.

A CR combina o desempenho do hardware dedicado com a flexibilidade alcançada com os softwares. Entretanto, enquanto que os componentes de software estão limitados às arquiteturas dos microprocessadores, a CR permite obter arquiteturas adaptadas às aplicações específicas. Assim, observa-se que a CR, que faz uso de sistemas programáveis, está posicionada entre as arquiteturas de Von Neumann e os Circuitos Integrados de Aplicação Específica (*Application Specific Integrated Circuit – ASIC*) - (Figura 6).

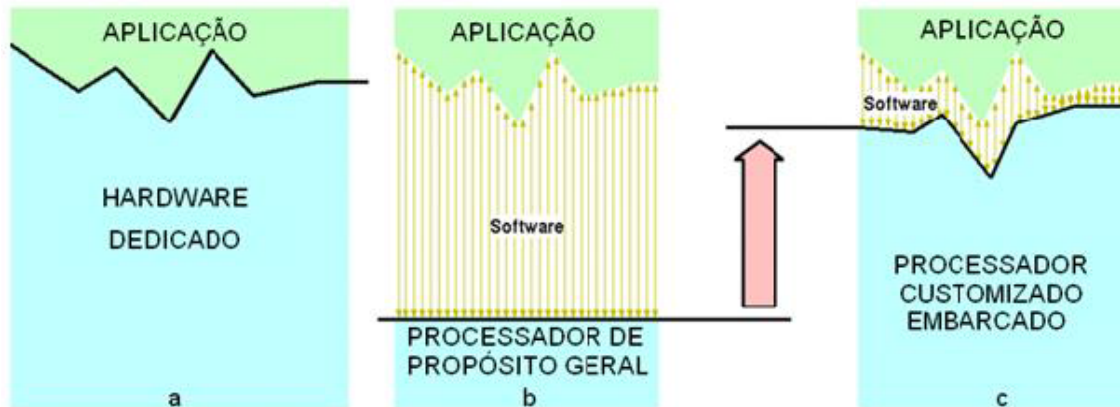
Figura 6 – Flexibilidade versus Desempenho dos Sistemas Programáveis.



Fonte: LOPES, 2012.

O emprego da CR em várias aplicações de sistemas digitais tem permitido a obtenção de altos desempenhos com baixo consumo de potência, quando comparado às aplicações desenvolvidas em software, e com qualidade inferior àquelas obtidas em implementações de Circuitos ASICs, em termos de área, performance e potência. Esta afirmação será melhor compreendida após a análise da Figura 7, página 29.

Figura 7 – Benefícios da Computação Reconfigurável.



Fonte: LOPES, 2012.

Como se pode observar, o primeiro cenário refere-se a um circuito integrado de aplicação específica (ASIC), desenvolvido, sobretudo, para realizar uma tarefa específica. Esta abordagem proporciona excelente desempenho e nenhuma flexibilidade. O segundo cenário faz alusão ao uso de software em um hardware de propósito geral. Tal abordagem é a mais flexível, pois permite a alteração da funcionalidade do sistema apenas com o ajuste do software. No entanto, as particularidades deste hardware não comportam alto desempenho quando comparadas às do hardware dedicado. Já a computação reconfigurável (CR), apresentada no terceiro cenário, é capaz de alcançar o desempenho oferecido pelo hardware, enquanto mantém a flexibilidade proporcionada pelo software (MENOTTI, 2010).

Sob esta visão, pode-se assegurar que um sistema computacional e os dispositivos lógicos usados como blocos construtivos podem apresentar uma estrutura e um comportamento fixos, programáveis, configuráveis ou reconfiguráveis (MARTINS et. al., 2012).

3.1 DISPOSITIVOS RECONFIGURÁVEIS

A rápida evolução dos recursos de concepção de circuitos integrados (CI), tanto na área de processos quanto na área de *Computer Aided Design* (CAD), tornou possível o surgimento de dispositivos com lógica programável (*Programmable Logic Devices* – PLDs).

Os dispositivos reconfiguráveis são dispositivos programáveis capazes de ser configurados para produzir o comportamento de um circuito lógico em hardware (GOMES; CHARÃO; VELHO, 2009). Estes dispositivos apresentam-se como alternativa tecnológica

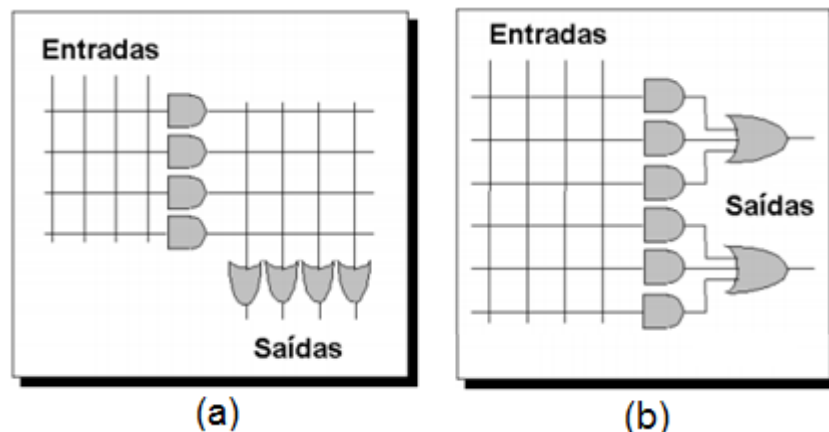
capaz de implementar inúmeras aplicações de propósito geral, enquanto mantêm as vantagens de desempenho de aplicações em dispositivos dedicados.

O emprego destes circuitos se dá, naturalmente, nos projetos de prototipagem, visto que, em sua maioria, esses circuitos podem ser reprogramados e testados nas etapas preliminares do projeto, permitindo, assim, grande economia de tempo e dinheiro (SILVA, 2010).

A utilização destes dispositivos em sistemas digitais modernos tem crescido rapidamente, pois permitem o desenvolvimento de novos produtos sem tanto investimento de tempo e/ou dinheiro, assim como, facilitam a evolução de projetos que já se encontram em produção.

Os PLDs surgiram em meados da década de 1970 como uma alternativa para programação lógica de dispositivos digitais em nível de hardware. Um *Programmable Logic Array* (PLA), o primeiro PLD desenvolvido, era constituído de dois níveis de portas lógicas, um plano de portas *wired-AND* seguido por um plano de portas *wired-OR*, conforme ilustrado na FIGURA 8 (a).

Figura 8 – Estrutura dos primeiros PLDs; (a) PLA; (b) PAL.



Fonte: RIBEIRO, 2002.

As principais deficiências dos PLAs eram o alto custo de fabricação e o baixo desempenho em termos de velocidade (RIBEIRO, 2002). Para superar estas deficiências, foram desenvolvidos os dispositivos *Programmable Array Logic* (PAL), constituídos de um único plano de portas *wired-AND* que alimentam portas OR, conforme FIGURA 8 (b).

Segundo Costa (2011), os primeiros PLDs - PAL (*Programmable Array Logic*) ou PLA (*Programmable Logic Array*) - utilizavam apenas portas lógicas convencionais. Em seguida, no final dos anos de 1970, surgiram PLDs com flip-flop em cada saída do circuito, o

que permitiu a implementação de funções sequenciais simples. No início da década de 1980, novos circuitos lógicos - portas lógicas e multiplexadores - foram adicionados a cada saída do PLD, surgindo uma nova estrutura de PLD chamada GAL *Generic Array Logic* (GAL) ou PALCE (PAL CMOS *electrically erasable/programmable device*). Estes dispositivos passaram a ser chamados coletivamente de SPLDs (*Simple PLDs*), conforme ilustrado no Quadro 2 abaixo, cujas características mais importantes eram o baixo custo e o alto desempenho.

Quadro 2 – Evolução dos PLDs.

PLDs	SPLDs	PAL (meados da década de 1970)
		PLA (meados da década de 1970)
		PAL/PLA registrados (final anos 1970)
		GAL/PALCE (início da década de 1980)
	CPLDs (meados da década de 1980)	
	FPGAs (meados da década de 1980)	

Fonte: PEDRONI, 2010.

A dificuldade em aumentar a capacidade da arquitetura dos SPLDs residia no fato da estrutura dos planos lógicos programáveis aumentar muito com o acréscimo no número de entradas do dispositivo. A saída foi integrar vários SPLDs em um mesmo chip com interconexões programáveis para interligar os SPLDs (RIBEIRO, 2002). Assim, em meados da década de 1980, vários dispositivos GAL foram fabricados no mesmo chip, utilizando um esquema de roteamento sofisticado, surgindo assim o que passou a se chamar CPLD (*complex PLD*).

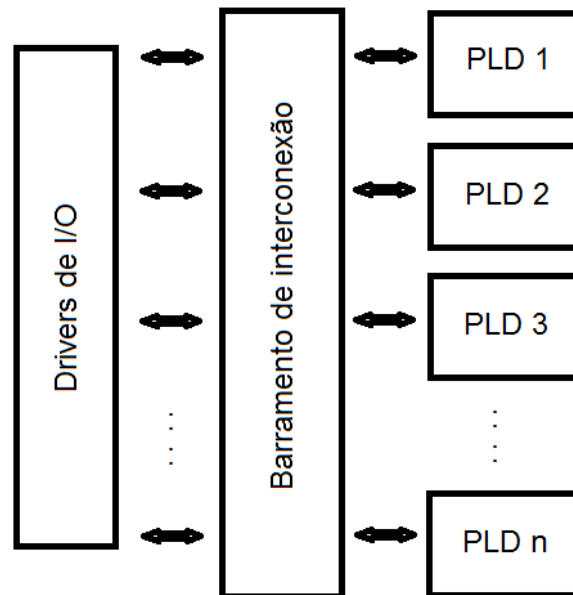
Os CPLDs possuem capacidade lógica equivalente a até 50 (cinquenta) SPLDs, mas estender esta arquitetura para densidades maiores requer uma abordagem diferente. Com um arranjo de elementos de circuitos não conectados, chamados de blocos lógicos, e recursos de interconexão entre blocos lógicos programáveis pelo usuário, em meados dos anos 1980, surgiram as *Field Programmable Gate Arrays* (FPGAs) visando seu emprego em projetos complexos e de alto desempenho, como chaves comutadoras de alta complexidade, transceptores *gigabit*, HDTV e outras aplicações em telecomunicações (COSTA, 2011).

FPGAs diferem dos CPLDs em arquitetura, tamanho, características embutidas, desempenho, tecnologia, custo e tipo de memória empregada (SRAM), o que exige uma memória de configuração (*configuration memory*) não volátil para programação do hardware ao inicializá-lo.

3.2 CPLDs (Complex PLDs)

A arquitetura básica de um CPLD é formada por vários PLDs (GALs, geralmente) num mesmo *chip* (Figura 9), se comunicando através de um esquema de interconexões programáveis, *drivers* de I/O mais sofisticados, suporte JTAG (porta para programação e teste definida pelo *Joint Test Action Group*) e uma grande quantidade de pinos de I/O para utilização pelo usuário.

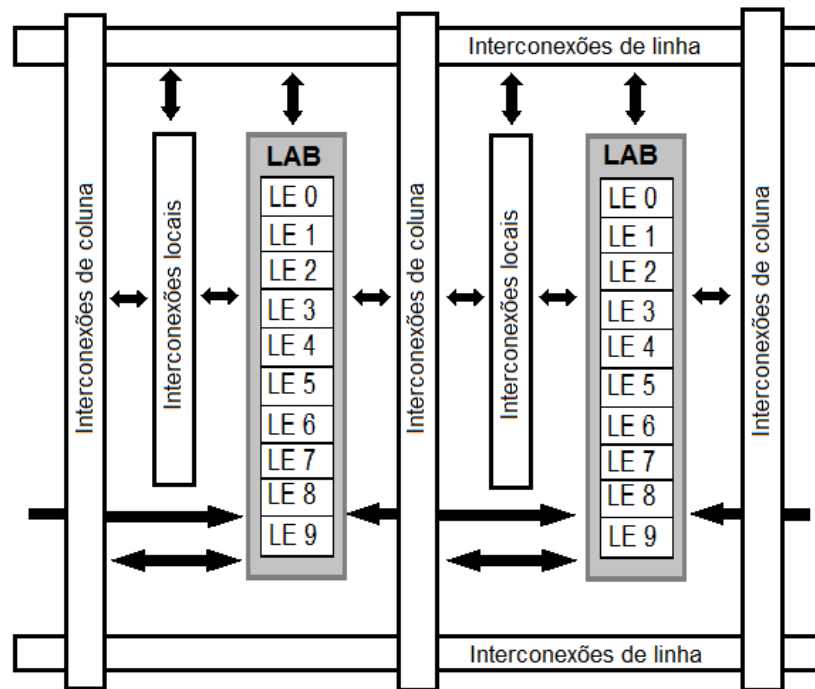
Figura 9 – Arquitetura básica de uma CPLD.



Fonte: PEDRONI, 2010.

Os CPLDs mais recentes são mais versáteis que os antecessores que possuíam a arquitetura baseada em GAL. Como ilustrado na Figura 10, página 33, um LAB (em vez de GAL), é constituído por um conjunto de 10 *logic elements* (LEs) com interconexões mais sofisticadas, locais e globais, o que o torna mais parecido com um FPGA simplificado do que com um CPLD tradicional.

Figura 10 – Arquitetura dos CPLDs recentes.



Fonte: PEDRONI, 2010.

3.3 FPGAs (Field Programmable Gate Arrays)

Atualmente, os dispositivos mais empregados na computação reconfigurável (CR) são os *Field Programmable Gate Array* (FPGAs), os quais são, inicialmente, projetados para a prototipação de circuitos.

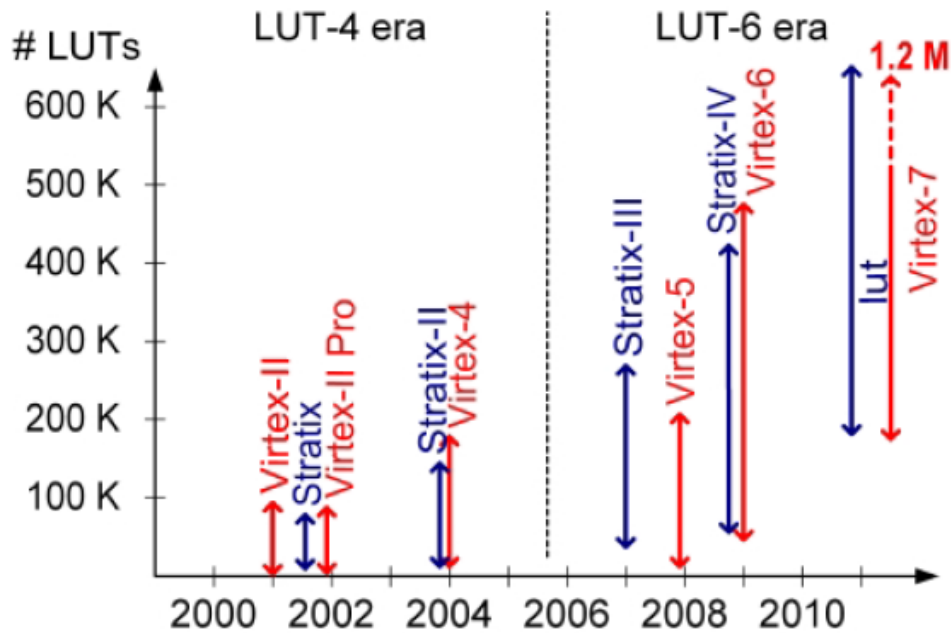
Os FPGAs são um tipo de circuito integrado programável pelo usuário, via software, e podem ser utilizados tanto em um produto final quanto para fins de prototipagem. Ele tem sido uma alternativa aos Circuitos Integrados de Aplicação Específica (*Application Specific Integrated Circuit* – ASIC) devido ao baixo custo do desenvolvimento e por possibilitar uma redução do tempo de lançamento de novos produtos no mercado.

Os primeiros FPGAs eram relativamente simples e continham apenas alguns milhares de portas lógicas. Atualmente, eles oferecem milhões de portas lógicas, alguns integrados a componentes embarcados, tais como *Digital Signal Processing* (DSP), blocos de memórias internas, *Central Processing Units* (CPUs) e outros blocos dedicados, formando-se assim um sistema completo numa única pastilha (PELLERIN; THIBAUT apud LOPES, 2012).

Na Figura 11, página 34, é possível observar nos FPGAs Stratix 5 da Altera e Virtex-7 da Xilinx, o dobro de capacidade lógica e memória embarcada, se comparados aos FPGAs

Stratix e Virtex-II, respectivamente da Altera e Xilinx, fabricados uma década antes (KOCK; TORRESEN apud LOPES, 2012).

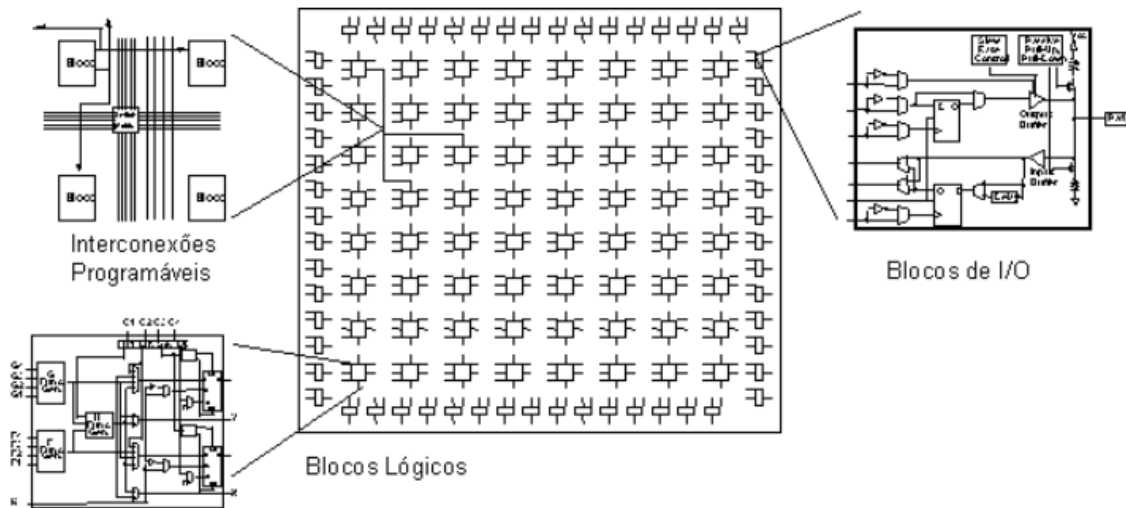
Figura 11 – Aumento da capacidade lógica e memória embarcada nos FPGAs em uma década.



Fonte: KOCK; TORRESEN apud LOPES, 2012.

A arquitetura geral de um FPGA é composta de uma matriz de Blocos Lógicos Configuráveis (*Configurable Logic Block* – CLB), como pode ser visto na Figura 12, página 35, em vez de uma pilha de blocos como ocorre com os CPLDs, em quantidade maior, porém menores e mais sofisticados. A organização dos CLBs se dá de forma bidimensional e a comunicação entre eles ocorre através de interconexões programáveis, distribuídas entre as linhas e as colunas dos blocos lógicos na forma de trilhas horizontais e verticais. A borda externa é composta de blocos especiais com capacidade para realizar operações de entrada e saída (*Input/Output Block* – IOB). Além desses elementos básicos, há também a estrutura de configuração que permite fixar o comportamento dos blocos lógicos e o caminho das interconexões (PEDRONI, 2010) (TOCCI, 2007).

Figura 12 – Arquitetura geral de um FPGA.



Fonte: RIBEIRO, 2012.

3.3.1 Tecnologias de Programação

Um FPGA é configurado através de comutadores programáveis eletronicamente, cujas propriedades – tamanho, tecnologia, capacitância e resistência – afetam o desempenho e determinam a volatilidade e a capacidade de reconfiguração (reprogramação), devendo ser consideradas no início do projeto visando a escolha da arquitetura mais adequada para a aplicação pretendida (RIBEIRO, 2002).

Todos os FPGAs são baseados em elementos de memória para manter a programação do dispositivo. Há três tecnologias comuns utilizadas para implementar memória para os *bits* de configuração em um FPGA:

1. *Flash* (EEPROM): A memória flash é uma variação moderna da EEPROM (*Electrically-Erasable Programmable Read-Only Memory*) e sua grande vantagem é a capacidade de reprogramação e retenção dos dados com o chip já instalado, chamado de *In System Programmability* (ISP);
2. Antifuse: possui tamanho reduzido, baixa capacitância quando não configurada e baixa resistência quando programada, o que tem reflexo no seu melhor desempenho em relação às demais tecnologias de programação de FPGAs. Entretanto, não permite reconfiguração; e,
3. *Static Random Access Memory* (SRAM): é uma memória volátil, o que exige uma memória externa para configuração do *chip* quando da inicialização do mesmo.

Ocupa mais espaço no *chip*, mas é reconfigurada mais rapidamente se comparada com outras tecnologias como EEPROM.

FLASH é uma memória não-volátil e retém os dados de configuração mesmo após a energia ser desligada do dispositivo, enquanto a SRAM é volátil e tem de ser programada a partir de uma memória externa cada vez que a energia é aplicada ao dispositivo. Em *antifuse*, a configuração do dispositivo é definida através da criação de ligações permanentes nas células de configuração. Tal como acontece com *flash*, *antifuse* requer passos extras de fabricação sobre o processo CMOS padrão (Quadro 3).

Quadro 3 – Características das Tecnologias de Programação.

Tecnologia	Reconfigurabilidade	Volatilidade	Processos Extras
<i>Antifuse</i>	Não	Não	3
EEPROM	Sim, na placa	Não	>5
SRAM	Sim, na placa	Sim	0

Fonte: PEDRONI, 2010.

Os únicos dispositivos no mercado que suportam configuração rápida do circuito são baseados em tecnologia SRAM.

3.3.2 Blocos Lógicos Configuráveis (*Configurable Logic Block* – CLB)

Os blocos lógicos dos FPGAs podem ser simples quanto um transistor ou complexos quanto um microprocessador. A funcionalidade dos CLBs está associada à implementação de funções lógicas básicas, como o AND, o OR, o XOR e o NOT, como também de funções combinacionais e sequenciais mais complexas, tais como: codificadores, decodificadores, multiplexadores, *Look Up Tables* (LUTs) e funções matemáticas.

A granularidade de um sistema reconfigurável é um dos fatores importantes quando falamos de hardware reconfigurável. Como descrito por Ribeiro (2002), há três categorias de granularidade que são usadas para classificar os FPGAs quanto à capacidade lógica dos CLBs:

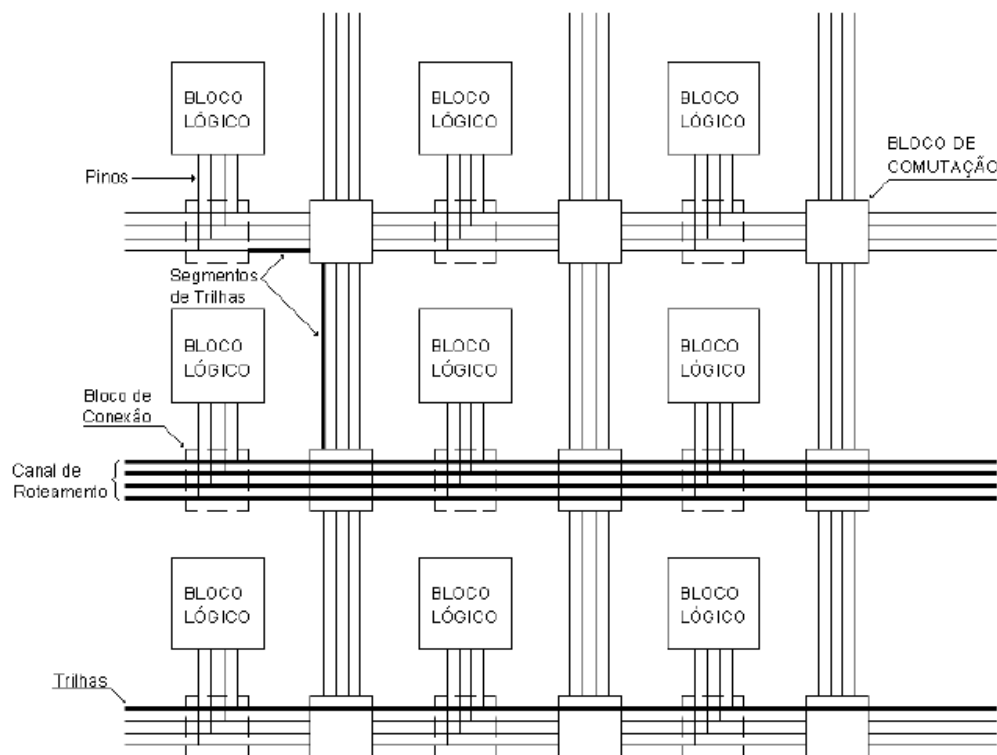
1. Granularidade Fina: há um grande número de blocos lógicos pequenos e simples contendo transistores ou portas lógicas básicas. Entretanto, devido à grande quantidade de blocos lógicos há necessidade de um grande número de interconexões programáveis, o que sobrecarrega o roteador além de deixá-lo lento e levá-lo a ocupar mais espaço no *chip*;

2. Granularidade Média: frequentemente, possuem duas ou mais LUTs e dois ou mais flip-flops. A maioria das arquiteturas de FPGA implementa funções lógicas em LUTs de 4 entradas;
3. Granularidade Grossa: podem conter blocos lógicos que implementam Unidades de Lógica e Aritmética (ULAs) e/ou pequenos microprocessadores.

3.3.3 Recursos de Roteamento

A maneira como os comutadores programáveis e segmentos de trilhas são configurados para realizar a interconexão de blocos lógicos dependem dos recursos de roteamento dos FPGAs e são compostos por duas estruturas básicas, conforme Figura 13.

Figura 13 – Recursos de Roteamento em um FPGA.



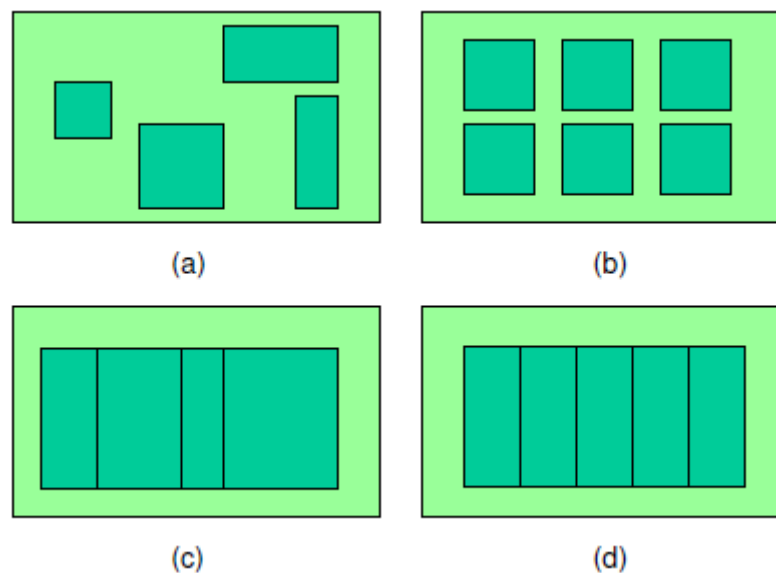
Fonte: RIBEIRO, 2002.

A primeira se refere ao bloco de conexão, presente em todas as arquiteturas de FPGA, e é responsável pela conexão dos pinos de I/O (entrada e saída) de um bloco lógico com os segmentos de trilhas dos canais de roteamento. A segunda estrutura é o bloco de comutação responsável pela ligação entre os seguimentos de trilhas horizontais e verticais.

3.3.4 Áreas reconfiguráveis

Áreas reconfiguráveis são partes do circuito reservadas para receber uma ou mais tarefas a serem executadas por um FPGA. Estas tarefas são representadas por retângulos que devem utilizar determinadas áreas reconfiguráveis, sendo bastante úteis para estabelecer restrições de regiões e roteamento das tarefas do FPGA (SANT'ANNA, 2006). Dependendo da forma com que estas áreas reconfiguráveis são arranjadas no FPGA, existem duas classificações, conforme mostra a Figura 14:

Figura 14 – Modelos de Áreas Reconfiguráveis. (a) e (b) 2D; (c) e (d) 1D.



Fonte: SANT'ANNA, 2006.

- Modelo 2D: é o modelo mais flexível e com maior taxa de utilização pois permite a definição de áreas reconfiguráveis em qualquer região do dispositivo. Neste modelo, deve-se evitar a fragmentação de tarefas em áreas dispersas do circuito.
- Modelo 1D: permite a definição de áreas reconfiguráveis apenas em colunas (parcialmente reconfigurável) o que pode interferir em tarefas distintas alocadas na mesma coluna quando da reconfiguração.

3.3.5 Tipos de Reconfiguração

Por causa da alta densidade de portas lógicas, dos recursos de roteamento, da rápida velocidade de reconfiguração e da possibilidade de configuração e reconfiguração, os FPGAs podem ser classificados de acordo com sua configurabilidade (LIMA; MARQUES, 2002):

1. **Programável:** abrange todos os tipos de FPGAs;
2. **Reconfigurável:** nesta categoria são incluídos todos os tipos de FPGAs, exceto os "one-time programming", que só podem ser configurados uma só vez;
3. **Parcialmente reconfigurável:** permite uma reconfiguração seletiva, enquanto o restante do dispositivo permanece inativo e retém a sua informação de configuração;
4. **Dinamicamente reconfigurável:** permite a reconfiguração de uma parte do circuito sem prejudicar o funcionamento do restante (circuito ativo). Em síntese, é possível configurar alguns blocos lógicos e segmentos de roteamento, enquanto outras partes do hardware continuam funcionando, sem interrupção.

Uma das formas de configurar um FPGA é por meio da elaboração de uma arquitetura dinamicamente reconfigurável, com um microcontrolador fornecendo o arquivo de configuração do FPGA no instante adequado (COMPTON; HAUCK, 2002).

As principais características da reconfiguração dinâmica são:

- Reaproveitamento dos recursos de hardware: com a reconfiguração da mesma parte do FPGA inúmeras vezes, viabiliza-se o uso de uma área do FPGA menor do que a necessária para um projeto estático;
- Exploração da simultaneidade: permite a preservação do tempo de execução, pois, ao contrário da arquitetura de Von Neumann, várias subtarefas podem ser realizadas simultaneamente no FPGA. Assim, operações que não possuem dependência de outras podem ser executadas simultaneamente.

Porém, há também desvantagens e dificuldades na utilização da reconfiguração dinâmica:

- Tempo de reconfiguração elevado: para todos os componentes da lógica reconfigurável, esse tempo torna-se elevado, variando na faixa de μ s a centenas de ms;
- A quantidade reduzida de software e ferramentas para síntese, simulação, depuração e teste;
- A complexidade acrescentada ao ciclo de projeto dos sistemas dinamicamente reconfiguráveis.

Essa complexidade, tanto na implementação como na verificação funcional dos sistemas reconfiguráveis, deve-se ao aumento na quantidade de portas lógicas em um FPGA, sendo, portanto, necessária a utilização de *Hardware Description Language* (HDLs), como Verilog e VHDL, para o desenvolvimento de sistemas, que é um processo que demanda mais

tempo de desenvolvimento se comparado ao tempo para se programar sistemas em alto nível, como os que utilizam as linguagens C/C++ (LOPES, 2012).

A implementação, em geral, requer decisões e conhecimentos tanto da micro-arquitetura do FPGA quanto das linguagens HDLs, além dos conhecimentos necessários das ferramentas dos fabricantes dos FPGAs. Ainda, dependendo da implementação, torna-se necessário explorar alternativas de arquiteturas e divisão de tarefas do projeto entre o hardware e o software.

Ainda, a reconfiguração dinâmica pode ser dividida em dois tipos distintos: reconfiguração total e reconfiguração parcial.

A reconfiguração dinâmica total realoca todos os recursos dos FPGAs em um estágio de configuração. Cada estágio é configurado para ocupar todos os recursos do FPGA, desta forma, o tamanho do *bitstream* (dados de configuração) é o mesmo durante cada configuração (GONZALEZ, 2006; KOJIMA, 2007).

Na reconfiguração dinâmica parcial, a configuração se dá em qualquer porcentagem dos recursos reconfiguráveis do dispositivo, em qualquer instante de tempo. Para isto, é preciso apenas carregar as tarefas necessárias em cada instante de tempo enquanto o dispositivo ainda está ativo, reduzindo, portanto, o tempo na troca das configurações (GONZALEZ, 2006; KOJIMA, 2007). Com isto, é possível uma realocação dinâmica de hardware mais eficiente.

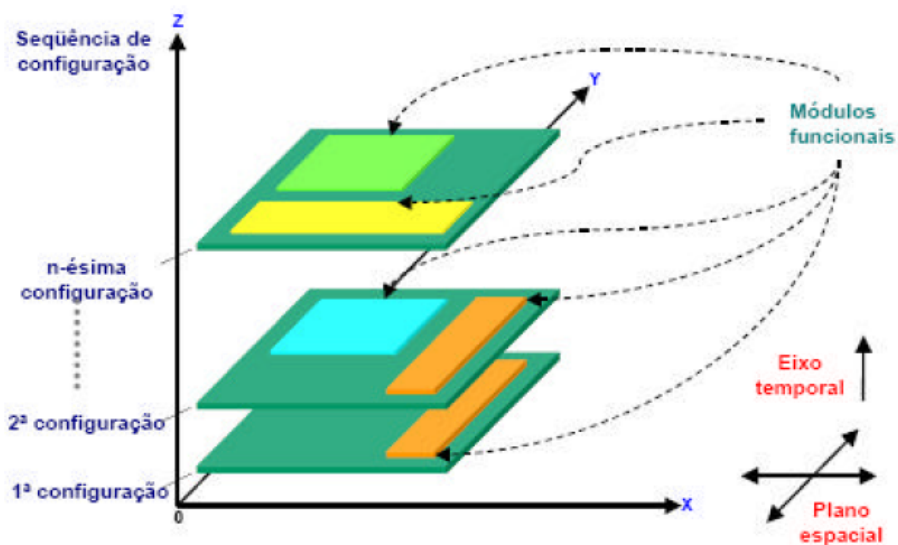
O emprego de FPGAs que usam reconfiguração parcial e dinâmica em tempo de execução – RTR (*Run-Time Reconfiguration*), fornece flexibilidade para diferentes tipos de aplicações ao possibilitar a reprogramação da lógica dos CLBs ou do roteamento mais de uma vez e com o circuito em operação (SEIXAS, 2007).

Segundo Bobda (2007), na implementação de um sistema no modo RTR, nem a computação nem a sequência de configuração do projeto são conhecidas em tempo de compilação, assim, os requisitos para implementar uma dada tarefa são conhecidas apenas em tempo de execução. RTR implica em desfragmentação do dispositivo e comunicação entre os módulos recentemente aplicados. Assim, a reconfiguração parcial dinâmica de FPGAs exige dois tipos de particionamento:

1. Particionamento espacial: forma espacial com que os módulos de hardware serão distribuídos na lógica reconfigurável;
2. Particionamento temporal: definição dos intervalos de tempo em que cada módulo de hardware deverá estar presente no FPGA, já que algum módulo reconfigurável pode ocupar recursos no dispositivo, já ocupado por outro.

A Figura 15 mostra esta característica numa analogia em 3 dimensões, onde os eixos X e Y representam a configuração dos CLBs e do roteamento (particionamento espacial), enquanto o eixo Z (particionamento temporal) representa as várias configurações de tarefas (módulos funcionais) possíveis em momentos distintos. Na Figura 15 é descrito um exemplo de reconfiguração parcial onde um FPGA possui uma configuração (1ª) já definida, mas com a chegada de uma nova configuração (2ª), e após a reconfiguração parcial, a nova configuração passa a fazer parte do FPGA juntamente com a parte que não foi alterada. Desta forma, as possibilidades de uso de um FPGA com reconfiguração parcial e dinâmica podem ser maiores do que o recurso fisicamente (hardware) disponível no dispositivo (SEIXAS, 2007).

Figura 15 – Analogia para a reconfiguração parcial e dinâmica de um FPGA.



Fonte: SEIXAS, 2007.

A reconfiguração parcial dinâmica é útil quando um projeto não precisa ocupar todo o hardware de um FPGA em um dado momento, ou quando apenas parte do projeto necessita ser alterada.

A reconfiguração parcial e dinâmica é realizada com dados de configuração chamados de *bitstream*, os quais podem ser descarregados no dispositivo por uma porta de configuração. A ideia por trás da reconfiguração parcial dinâmica é realizar a reconfiguração apenas mudando algumas partes dos *bits* de configuração sendo executadas no FPGA.

O maior problema da reconfiguração parcial dinâmica é justamente produzir o *bitstream* de reconfiguração. Há pelo menos duas formas para fazer a extração do *bitstream*, representando um dado módulo (BOBDA apud LOPES, 2012):

- A primeira abordagem, chamada de construtiva, permite fazer a implementação de um dado módulo separadamente, usando ferramentas comuns de desenvolvimento, e então juntar esse módulo no *bitstream* completo ou construí-lo como somas de módulos parciais.
- A segunda possibilidade consiste primeiramente em gerar o *bitstream* completo separadamente. As partes fixas, bem como as partes reconfiguráveis, são implementadas como componentes e juntadas em outro *bitstream*. As diferenças entre os dois *bitstreams* estão nas partes reconfiguráveis e, então, os dois *bitstreams* são computados para se obter o *bitstream* parcial dinâmico.

Mesmo com tantas vantagens para a produção de sistemas flexíveis e adaptativos, existem poucos dispositivos que permitem reconfiguração em tempo de execução (RTR), principalmente devido à complexidade das arquiteturas e sua reconfiguração parcial, agravado pela pouca disponibilidade de ferramentas de CAD para projeto, depuração e testes (SEIXAS, 2007). A Xilinx, por meio da família Virtex, é uma das únicas empresas que produzem FPGAs com reconfiguração parcial e dinâmica em tempo de execução.

3.3.6 Dispositivos Xilinx Virtex-6

A família de componentes Virtex-6 da empresa Xilinx® é uma geração de FPGAs superior às famílias anteriores (Virtex, Virtex-2, Virtex-3, Virtex-4 e Virtex-5), possui alta performance e permite reduzir o consumo de energia em até 50% em relação a família Virtex-2. Os dispositivos FPGAs da família Virtex-6 são divididos em três plataformas (XILINX, 2012):

1. Virtex-6 LXT: para aplicações de alto desempenho;
2. Virtex-6 SXT: para aplicações *Digital Signal Processing* (DSP) de alto desempenho;
3. Virtex-6 HXT: para aplicações de plataforma embutidas de alto desempenho.

Todos os dispositivos Virtex-6 implementam as seguintes funcionalidades:

- Blocos de I/O (IOBs): servem de interfaces entre os CLBs e os pinos do componente;
- Blocos lógicos configuráveis (CLBs): proporcionam elementos funcionais para a lógica sequencial e concorrente;

- Blocos de RAM: elementos de armazenamento de 36 *Kbits* dedicados de memória RAM do tipo *dual-port*;
- *XtremeDSP slices*: com 18 *bit* x 18 *bit* multiplicadores dedicados, Somador integrado e Acumulador de 48 *bits*;
- Blocos *Digital Clock Manager* (DCM): solução completa para a compensação dos atrasos de distribuição do *clock*.

Outra funcionalidade disponibilizada no FPGA Virtex-6 é a matriz geral de rotas (MGR) que fornece um vetor de chaves de roteamento entre os componentes. Cada elemento programável é acoplado a uma matriz de chaves, o que permite múltiplas conexões com a MGR. O conjunto de interconexões programáveis é hierárquico. Todos os elementos programáveis, incluindo os recursos de roteamento, são controlados por valores carregados e armazenados em células de memória estática.

Os dispositivos Virtex-6 suportam:

- Transceptor de dados seriais de alta velocidade *RocketIO Multi-Gigabit Transceiver* (MGT) com taxas de transmissão acima dos 10 Gb/s por canal;
- CPU RISC embutida IBM PowerPC 405 (450 MHz);
- *Cores Ethernet Media Access Control* (EMAC) de 10/100/1000 Mb/s.

Em termos de reconfiguração dinâmica, os dispositivos Virtex-6 apresentam as seguintes vantagens em relação às famílias anteriores:

- Número de I/Os e *slices* maiores, permitindo o desenvolvimento de aplicações mais complexas e de tamanho maior;
- Frequência de reconfiguração: 100 MHz. A família Virtex-II apresenta frequência máxima de reconfiguração de 50 MHz;
- Interface de Configuração (*SelectMAP*): 8 ou 32 *bits*. A família Virtex-II possui Interface de Comunicação (*SelectMAP*) menor: 8 *bits*;
- *Bus macros* que permitem a ligação dos sinais da direita-para-esquerda, da esquerda-para-direita, de cima-para-baixo e de baixo-para-cima, possibilitando uma flexibilidade maior para o desenvolvimento de projetos dinamicamente reconfiguráveis.

Ao longo desta seção, foram introduzidos os tópicos básicos da computação reconfigurável e alguns de seus paradigmas, com vantagens e desvantagens. Também foram abordados alguns dispositivos reconfiguráveis, em especial o FPGA, a metodologia utilizada

em sua programação e os tipos de reconfiguração. Na próxima seção, serão abordados os conceitos para implementar RNAs em FPGA.

4 REDES NEURAIIS ARTIFICIAIS EM FPGAs

Uma rede neural artificial (RNA) é uma rede distribuída e paralela composta por unidades de processamento não-lineares, interligadas por camadas. Paralelismo, modularidade e adaptação dinâmica são três características computacionais normalmente associadas as RNAs. Mas, o número de operações e a complexa rede de conexões das RNAs as tornam difíceis de serem implementadas em FPGA.

As propostas de implementações de RNAs embarcadas em FPGAs existentes, e as que ainda estão por vir, têm de lidar com os problemas usuais impostos por esse paradigma. Estes problemas tornam-se mais agudos por causa de restrições específicas inerentes aos FPGAs, tais como: consumo de área e tempo de processamento.

Para contornar tais restrições, FPGAs baseados em arquiteturas parcialmente reconfiguráveis tornam-se adequados para implementar RNAs cujo propósito seja explorar a simultaneidade e a reprogramação destes dispositivos para adaptar os pesos e as topologias de uma RNA. Com essa nova era de dispositivos parcialmente reconfiguráveis, tem-se observado o surgimento de novas estratégias de implementação de RNAs embarcadas em FPGA.

Diante do exposto, estratégias empregadas em trabalhos anteriores foram examinadas e classificadas de acordo com as decisões de projeto em cada caso. Estas classificações proporcionarão meios para identificar vantagens e desvantagens de cada estratégia.

4.1 CLASSIFICAÇÃO DAS REDES NEURAIIS IMPLEMENTADAS EM FPGA

Todas as implementações de RNAs em FPGA, de alguma forma, tentam explorar características como: paralelismo e configurabilidade. Saber identificar e empregar estas características possibilita explorar diferentes implementações para os seguintes recursos:

- Algoritmo de Aprendizagem;
- Representação dos Dados;
- Esquemas de redução de Multiplicador;
- Ligação das sinapses;
- Função de ativação.

4.1.1 Algoritmo de Aprendizagem

A etapa de aprendizado de uma RNA consiste de um processo iterativo de ajustes dos parâmetros de entrada da rede, os pesos sinápticos das conexões, que guardam, ao final do processo, o conhecimento que a rede adquiriu do ambiente externo (MENDEL; MCLAREN *apud* GONÇALVES, 2013). Entretanto, a realização desta etapa direta no FPGA é muitas vezes considerada difícil e inútil, uma vez que necessita de alguns operadores específicos e uma precisão elevada, a fim de evitar perda de eficiência.

Com base nisto, várias implementações de redes neurais artificiais em FPGA têm sido propostas usando diversas regras de aprendizado, especialmente com o intuito de habilitar o aprendizado *on-chip*. Dependendo de onde é realizado o treinamento, a rede pode assumir uma das três classificações a seguir (CHAVES, 2010):

- ***Off-chip learning***: O treinamento *off-chip* ocorre com a implementação do algoritmo de aprendizagem num hardware de propósito geral. Os pesos resultantes do processo de treinamento são quantizados e então carregados no *chip* para fazer somente a propagação *forward*. Com base nisto, pode-se afirmar que o treinamento *off-chip* é, naturalmente, o mais escolhido quando nenhum aprendizado dinâmico é necessário;
- ***Chip-in-the-loop learning***: Neste caso a rede neural artificial implementada em hardware é usada durante o treinamento, mas apenas na propagação *forward*. O cálculo do novo peso é feito em um computador, o qual carrega os pesos no FPGA após cada iteração de treinamento.
- ***On-chip learning*** - O treinamento da rede neural artificial é feito inteiramente *on-chip* o qual oferece a possibilidade de treinamento contínuo. Isto significa que os valores dos pesos sinápticos são representados com precisão limitada. Simulações têm demonstrado que o algoritmo de *backpropagation* é altamente sensível ao uso de pesos sinápticos com precisão limitada e que o treinamento falha quando a precisão do peso sináptico é menor que 16 *bits*. Isto ocorre, sobretudo, por causa da atualização dos pesos sinápticos serem menores que o passo de quantização, o que impede a substituição dos mesmos. Para reduzir a área do *chip* necessária para armazenamento do peso e superar a precisão limitada do sistema, é desejado um favorecimento na redução do número de valores de pesos permitidos.

De todos os algoritmos de aprendizado empregados nas RNAs, o *backpropagation* é comumente o mais empregado para o treinamento em FPGA. Sua implementação em FPGA

expõe alguns desafios a serem solucionados. Um deles refere-se a natureza sequencial de processamento existente entre as camadas. O outro refere-se ao processamento em *pipelining* dos dados durante a fase de treinamento (ELDREDGE, 1994). Este problema surge devido às dependências de atualização dos pesos sinápticos na fase de propagação reversa. Porém, ainda é possível usar *pipelining*, mesmo que seja somente nas funções aritméticas individuais do algoritmo *backpropagation*, o que poderia ajudar a aumentar tanto a transferência de dados quanto a velocidade do *clock* (ELDREDGE, 1994, NICHOLS, 2003).

4.1.2 Representação dos Dados

Dependendo da notação numérica atribuída aos dados da rede, pode-se obter desempenhos consideráveis. Sabe-se, também, que o desempenho é altamente dependente do intervalo de precisão adotado.

A escolha da notação numérica e do intervalo de precisão, em quaisquer arquiteturas de RNAs implementadas em FPGA, é uma das decisões mais importantes durante a fase de projeto. Nessa fase, o projetista faz a escolha por uma das duas notações mais empregadas em projetos digitais – notação em ponto fixo e notação em ponto flutuante.

- **Ponto fixo:** Nesta notação, busca-se um modo para representar não apenas a parte inteira dos dados, mas também a sua parte fracionária. Para tanto, divide-se os n dígitos da representação em dois grupos, separados pelo “ponto separador”, também conhecido como “*radix point*”. Os dígitos do grupo à esquerda do ponto separador são usados para representar a parte inteira do número, enquanto que os dígitos da direita são usados para representar a sua parte fracionária (BARROS, 2008).
- **Ponto flutuante:** Esta notação estende a notação de ponto fixo com o acréscimo de mais um grupo de dígitos, chamado expoente. Com o acréscimo deste grupo de dígitos o número passa a ser representado pelo valor contido no conjunto de dígitos da parte inteira e da parte fracionária, os quais recebem juntos o nome de significando, multiplicado pela base elevada ao valor do campo expoente (BARROS, 2008).

Ambas notações, citadas anteriormente, são consideradas representações numéricas digitais com base na amplitude. As representações numéricas digitais com base na frequência

e a analógica não foram consideradas neste estudo porque promovem o uso de baixa precisão, o que, muitas vezes, é considerado inadequado para a faixa de precisão mínima permitida (MOUSSA; AREIBI; NICHOLS, 2006).

Ponto fixo é a notação numérica mais empregada em aplicações que envolvem RNAs em projetos com FPGA, uma vez que o emprego da notação em ponto flutuante eleva o custo do produto final devido ao aumento no número de *Configuration Logical Blocks* (CLBs) empregados (FERREIRA, 2008). Entretanto, o benefício em adotar ponto flutuante incide sobre os vários graus de precisão baseados na escala de valores numéricos que se está utilizando para os dados (ZHU; SUTTON, 2003, MOUSSA; AREIBI; NICHOLS, 2006).

O maior benefício em se usar ponto fixo para representar números reais reside no fato de os números em ponto fixo seguirem os mesmos princípios da aritmética de números inteiros. Além do mais, migrar para essa notação a partir de uma arquitetura de inteiros não requer nenhuma lógica adicional (FERREIRA, 2008).

A desvantagem do uso do ponto fixo incide no fato de que os números só podem ser representados em um intervalo limitado de valores. Logo torna-se susceptível a ocorrência de erros de *overflow* e de *underflow* (SÁNCHEZ, 2006).

Alternativamente, pode-se aproveitar o que se tem de melhor nas duas notações, ou seja, trabalhar com uma combinação híbrida de aritmética em ponto fixo e em ponto flutuante. Para isso, são separados *bits* para definir mantissa, expoente, sinal e um *offset* fixo, o que permite a demonstração do valor em ponto flutuante (WIIST; KASPER; REININGER, 1998).

4.1.3 Estratégias de redução dos Multiplicadores

A multiplicação é classificada como a operação aritmética que utiliza mais área em uma RNA embarcada em FPGA. Em um esforço para maximizar a densidade de processamento, muitas soluções foram propostas para reduzir o número de multiplicadores, as quais estão listadas e discutidas a seguir:

- **Multiplicadores *bit-serial*** – Este tipo de multiplicador só calcula um *bit* de cada vez, enquanto que um multiplicador totalmente paralelo calcula todos os *bits* simultaneamente. Conseqüentemente, o multiplicador *bit-serial* pode ampliar o sinal representado para qualquer alcance de precisão, enquanto que a área ocupada permanece estática quando implementado em hardware. No entanto, o tempo gasto para realizar a multiplicação de complexidade $O(n^2)$, depende do comprimento da representação do sinal utilizado. O uso de *pipelining* é a maneira de ajudar a

compensar longos tempos de multiplicação e aumentar a taxa de transferência de dados (ELDREDGE, 1994, TAVENIKU; LINDE, 1995).

- **Multiplicadores com alcance de precisão reduzido** – São implementados para reduzir o alcance de precisão dos dados utilizados nas operações de multiplicação. Para Skrbek (1999), esta abordagem não é viável, porque limita o alcance de precisão, ocasionando um efeito negativo nas taxas de convergência.
- **Utilização de um algoritmo de tempo multiplexado** – Este tipo de algoritmo tem sido adotado, em oposição ao alcance de precisão reduzido, para diminuir a quantidade de multiplicadores empregados nos neurônios artificiais (PEREZ-URIBE, 1999, BEUCHAT; HAENNI; SANCHEZ, 1998). O algoritmo de tempo multiplexado, proposto por Eldredge (1994), é a versão mais intuitiva, portanto, a mais empregada nas RNAs com retropropagação. Este algoritmo usa apenas um único multiplicador por neurônio, que deve ser compartilhado entre todas as entradas ligadas a um neurônio particular. Como resultado, o crescimento deste algoritmo em hardware é de complexidade $O(n)$, sendo n , o número de neurônios contidos na RNA. Conseqüentemente, o tempo gasto para realizar a multiplicação é também de complexidade $O(n)$.
- **Uso de "neurônios virtuais"** – A ampliação, em hardware, de qualquer topologia de uma RNA só é viável devido à utilização de elementos de processamento virtuais, ou, simplesmente, neurônios virtuais. Análogo ao conceito de memória virtual empregado nos computadores pessoais de mesa, os neurônios virtuais são utilizados apenas quando necessários, através de operações de *swapping*, ou seja, através da transferência de neurônios virtuais da memória para a plataforma de hardware. Assim, o limite para o número de neurônios passa a ser o tamanho da memória virtual e não mais o tamanho do hardware. Entretanto, este ganho de escalabilidade tem um custo adicional, derivado do tempo extra com a realização do *swapping* (NORDSTROM, 1995).

4.1.4 Aproximação da função de ativação

Quando se trabalha em um projeto que envolve RNAs com FPGAs, deve-se, ainda na fase de projeto, levar em consideração a não linearidade de algumas funções de ativação, tais como: sigmóide e tangente hiperbólica. Estas funções são amplamente utilizadas para o treinamento *on-line*, devido a sua natureza diferenciável. No entanto, a implementação direta

destas funções em FPGA não é uma boa prática de projeto, porque requer uma lógica excessiva no desenvolvimento de uma divisão e de uma série exponencial (CAMPO, FINKER, ECHANOBE *et al.*, 2013).

Para contornar os problemas citados anteriormente, muitas técnicas de aproximação têm sido desenvolvidas. Desde a implementação direta em uma *Look up table* (LUTs) até outras categorias de aproximações como: algoritmo CORDIC, séries polinomiais, lineares por partes, segunda ordem por partes e os mapeamentos de entrada/saída combinacionais (OMONDI; RAJAPAKSE, 2006, FERREIRA, 2008).

A escolha da técnica de aproximação e sua implementação em hardware são aspectos fundamentais que condicionam a precisão da função de ativação e, conseqüentemente, as capacidades de aprendizagem e de generalização de toda a RNA. Além disso, uma precisão muito baixa leva ao mau desempenho, enquanto uma precisão muito alta aumenta, desnecessariamente, os recursos de hardware e reduz a velocidade de processamento (OMONDI; RAJAPAKSE, 2006).

Uma LUT pode ser usada para implementar a função de ativação sigmóide por meio de valores discretos. Os problemas relacionados a este método estão centrados no consumo de área e tempo, o que acaba afetando a velocidade de processamento dos dados. Sabe-se que a implementação direta das funções de ativação sigmóide e tangente hiperbólica em uma LUT aumenta consideravelmente o tamanho do hardware. Para otimizar a área, até certo ponto, as RAMs disponíveis nos FPGAs podem ser usadas para realizar as funções de ativação baseadas nas LUTs. Elas reduzem a área e melhoram a velocidade de acesso aos dados (PANICKER; BABU, 2012).

O algoritmo CORDIC, para implementação em hardware, talvez seja a técnica mais estudada, mas a menos empregada. A vantagem do CORDIC é que o mesmo hardware pode ser usado para várias funções, mas o desempenho resultante é geralmente bastante limitado.

Outra abordagem mais intuitiva, consiste em usar séries polinomiais, como a série de Taylor, para representar qualquer função contínua arbitrária com uma baixa taxa de erro. Para isto, limita-se a ordem do polinômio por meio do particionamento do domínio da função em subintervalos menores. Contudo, essa estratégia não é adequada para se implementar em hardware, devido ao grande número de operações aritméticas (multiplicações e adições) que devem ser executadas para cada valor, gerando, portanto, uma maior quantidade de lógica a ser implementada.

Os autores Basterretxea, Tarela e Del Campo (2004), propõem uma solução eficiente e eficaz para representar a função sigmóide com base na aproximação linear por partes. Esta

solução foi otimizada visando a implementação de RNAs em hardware. Ela é bastante eficiente computacionalmente, pois permite um certo controle da taxa de precisão em relação ao atraso computacional proveniente da recursividade do algoritmo. Esta solução define as seguintes funções:

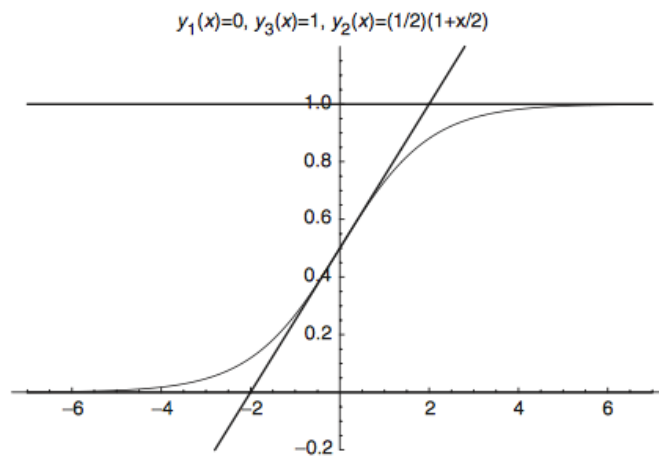
$$y_1(x) = 0 \quad (4)$$

$$y_2(x) = \frac{1}{2} \left(1 + \frac{x}{2} \right) \quad (5)$$

$$y_3(x) = 1 \quad (6)$$

O esquema é como segue no Gráfico 2.

Gráfico 2 - Função sigmóide com base na aproximação linear por partes.



A partir das funções iniciais, o método computa a saída em q passos da seguinte forma:

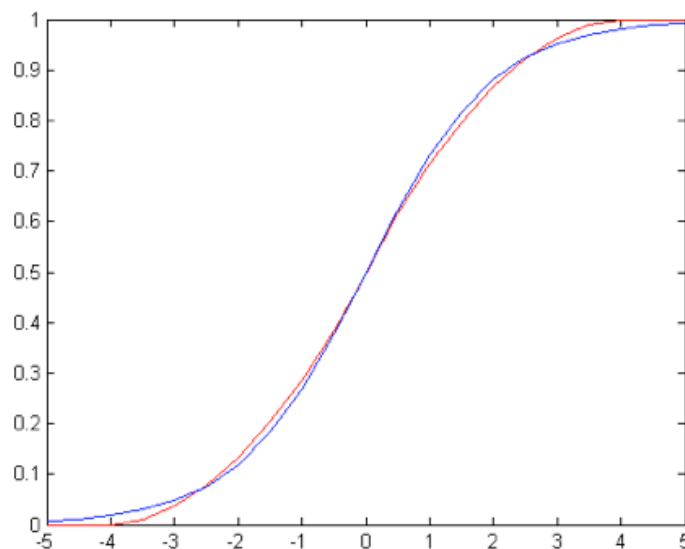
```

 $g(x) = y_1(x) = 0; h(x) = y_2(x) = 1/2(1 + x/2);$ 
for (i =0; i = q; i ++ )
{ $g'(x) = \max [g(x), h(x)];$ 
 $h(x) = 1/2(g(x) + h(x) + \Delta);$ 
 $g(x) = g'(x);$ 
 $\Delta = \Delta/4; \}$ 
 $g(x) = \max [g(x), h(x)];$ 

```

O algoritmo define a parte negativa do domínio da função, havendo uma simetria, rebate-se a parte negativa para a parte positiva. Observa-se, também, que o valor de Δ depende do valor de q . Os autores comentam que a melhor aproximação se dá para $q = 4$ e, por conseguinte, $\Delta = 0.2638$. Ferreira (2008) testou o algoritmo com uma implementação no Matlab, constatando que o erro médio media $1.4539e-017$ e que o erro máximo media 0.0194 , no intervalo de -5 a 5 . No Gráfico 3, é possível visualizar o comparativo da função de aproximação linear otimizada (vermelho) com a função sigmóide (azul).

Gráfico 3 – Aproximação linear otimizada (vermelho) vs sigmóide.



Fonte: FERREIRA, 2008.

O método PLAN, caracteriza-se também como uma aproximação linear por partes. Proposta por Amin, Curtis e Hayes-Gill (1997), a aproximação PLAN (Tabela 1) é um método simples e direto de se implementar, ele emprega portas digitais para realizar a

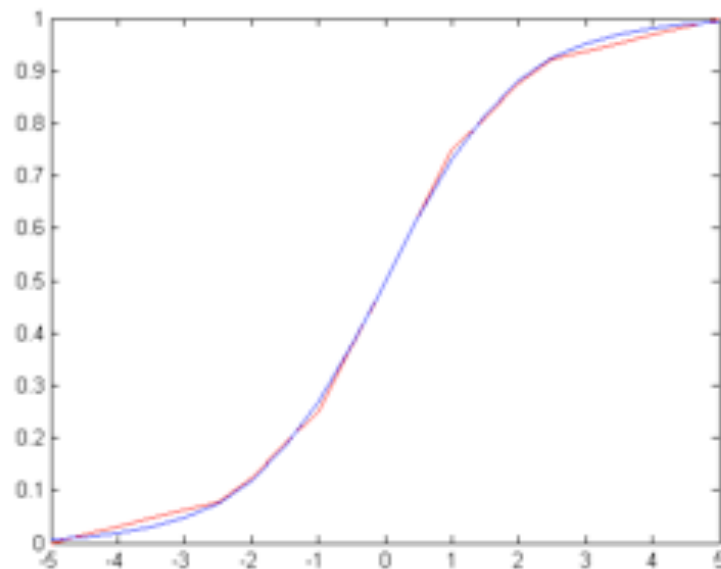
transformação direta de x para y , em que x é a entrada e y é a saída aproximada da função sigmóide. Com outras palavras, os cálculos são realizados apenas sobre o valor absoluto da entrada x .

Tabela 1 – Aproximação PLAN.

Operação	Condição
$Y = 1$	$ X \geq 5$
$Y = 0.03125 \cdot X + 0.84375$	$2.375 \leq X < 5$
$Y = 0.125 \cdot X + 0.625$	$1 \leq X < 2.375$
$Y = 0.25 \cdot X + 0.5$	$0 \leq X < 1$

Como desvantagem, o método PLAN apresenta uma aproximação não suave da função sigmóide. Apesar de parecer grosseiro, este método apresenta uma boa apresentação com erro médio $8.9214e-18$ e um erro máximo de 0.0189, no intervalo de -5 a 5. Graficamente ficam bem evidentes os pontos de intersecção dos segmentos (Gráfico 4).

Gráfico 4 – Aproximação PLAN (vermelho) vs sigmóide.



Fonte: FERREIRA, 2008.

Para representar a função de ativação tangente hiperbólica, pode-se usar uma simples função não linear por partes de segunda ordem. Isso implica que a função se apresente na seguinte forma:

$$y(x) = c_0 + c_1 * x + c_2 * x^2 \quad (6)$$

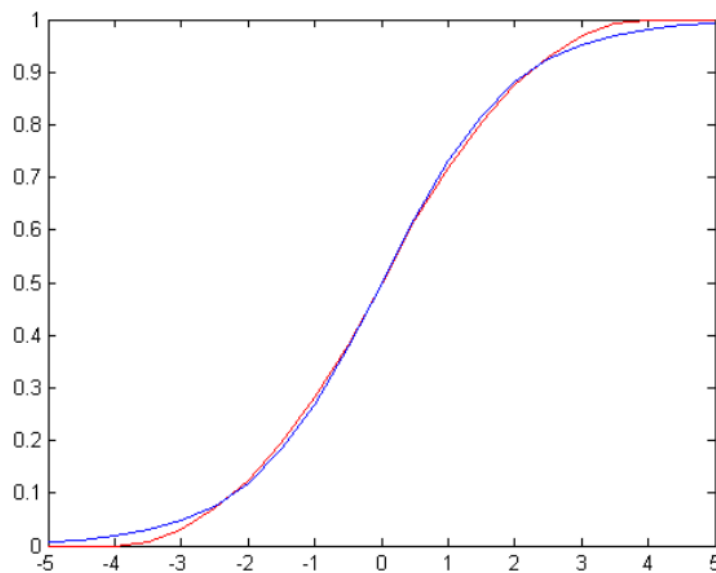
Uma desvantagem desta abordagem é a necessidade de três multiplicações. Zhang, et al. (1996) propuseram uma versão que necessitasse de apenas uma operação de multiplicação, usada para calcular o quadrado, as demais operações de multiplicação foram implementadas usando lógica combinacional. Ademais precisa-se de uma soma.

$$y = \begin{cases} 2^{-1} * (1 - |2^{-2} * x|)^2 & -4 < x < 0 \\ 1 - 2^{-1} * (1 - |2^{-2} * x|)^2 & 0 \leq x < 4 \end{cases} \quad (7)$$

$$(8)$$

A vantagem deste método é fornecer uma aproximação suave (com uma descontinuidade em $x=0$). Essa característica propicia uma possível extensão da arquitetura proposta para incluir a parte do aprendizado baseado num método de gradiente descendente. A implementação forneceu um erro médio de $8.5910e-018$ e uma erro máximo de 0.0215 , como pode ser visualizado no Gráfico 5.

Gráfico 5 – Aproximação de 2ª ordem vs sigmóide.



Fonte: FERREIRA, 2008.

4.1.5 Paralelismo em redes neurais

As RNAs apresentam vários tipos de paralelismo, logo, uma análise mais cautelosa é necessária a fim de determinar quais estruturas são mais adequadas, bem como seu mapeamento em hardware. Sabe-se, no geral, que a implementação de uma rede neural em

hardware totalmente em paralelo não é viável por causa da disposição sequencial das camadas. Os tipos mais específicos de paralelismo presentes nas RNAs são classificados a seguir (OMONDI; RAJAPAKSE, 2006):

- **Paralelismo ao nível de treinamento:** sessões de treinamentos podem correr em paralelo, por exemplo, em processadores SIMD ou MIMD. O paralelismo presente neste nível é geralmente médio, isto é, na ordem das centenas, e, portanto, pode ser quase totalmente mapeado para os FPGAs atuais.
- **Paralelismo ao nível das Camadas:** Em uma rede de múltiplas camadas, as diferentes camadas podem ser processadas em paralelo. Este tipo de paralelismo é baixo, mas pode ser explorado por meio de *pipelining*.
- **Paralelismo ao nível dos neurônios:** Este nível de paralelismo envolve os neurônios artificiais de uma mesma camada, sendo talvez o mais importante nível de paralelismo, em que é possível explorá-lo totalmente.
- **Paralelismo ao nível dos pesos:** Para o cálculo de uma saída em que x_i é uma entrada e w_i é um peso sináptico, os produtos $x_i w_i$ e a soma destes produtos podem ser calculados em paralelo, utilizando, portanto, uma “árvore de somas” com profundidade logarítmica.
- **Paralelismo ao nível de bits:** Este nível de paralelismo acelera a execução de uma instrução aumentando a largura do *datapath*.

4.1.5.1 Dispositivos Paralelos

Implementações rápidas de aplicações de rede neural são úteis por causa do elevado número de operações aritméticas necessárias. Tais implementações podem usar computadores maciçamente paralelos, bem como projetos de hardware digitais ou analógicos. Esta subseção discute rapidamente o emprego de vários dispositivos paralelos possíveis (GIRAU, 2006).

- **Computadores paralelos de propósito geral.** Implementações paralelas de grão fino, realizadas em computadores maciçamente paralelos, apresentam uma certa dificuldade em representar a conectividade dos modelos neurais artificiais, por causa das trocas de informações que são bastante dispendiosas. Já as implementações de grão grosso são aplicadas principalmente para o aprendizado neural, de modo que a sua eficiência sofre com a sequencialidade dos algoritmos de aprendizagem padrão, tais como o gradiente descendente estocástico. Além disso, os computadores maciçamente

paralelos são bastante caros e não podem ser utilizados em aplicações embarcadas. Tais soluções são geralmente preferíveis para estruturas neurais enormes, como computações complexas ou métodos de aprendizagem.

- **Computadores paralelos dedicados.** Neuro-computadores são sistemas paralelos dedicados à computação neural. Eles são baseados nos processadores de sinais digitais ou nos neuro-processadores. A sua utilização esbarra nos quesitos custo e tempo de desenvolvimento. Por causa disso, eles rapidamente tornam-se obsoletos, em comparação com os mais recentes processadores sequenciais.
- **ASICs analógicos.** Muitas implementações de hardwares analógicos foram realizadas. Eles são muito rápidos e de baixa potência, mas apresentam problemas específicos, tais como: precisão, armazenamento de dados e robustez. A aprendizagem “*on-chip*” é outro problema apresentado por este hardware, pois é difícil de se implementar. Este tipo de hardware torna-se uma solução cara e não flexível, como qualquer ASIC, mesmo que alguns métodos de concepção insistam em ser mais flexíveis.
- **ASICs digitais.** Muitos circuitos integrados digitais também foram projetados para redes neurais. Em comparação com *chips* analógicos, eles proporcionam mais precisão e mais robustez, podendo, portanto, lidar com qualquer computação neural. No entanto, projetos que envolvem este tipo de hardware são caros e, também, inflexíveis, ou seja, após o *chip* finalizado o mesmo não pode sofrer mais alterações. Nestes circuitos, geralmente, são implementadas apenas partes da rede neural, visando sua inclusão em sistemas de neurocomputadores. Às vezes, pode-se implementar toda uma rede neural, porém, a arquitetura dessa rede não é mapeada diretamente no *chip*.
- **FPGA.** Em resumo, estruturas regulares de RNAs podem ser, eficientemente, desenvolvidas nos ASICs digitais e analógicos, mas exigem tempos de desenvolvimento elevados. Para contornar esta grande desvantagem, pode-se fazer uso de circuitos integrados programáveis como os FPGAs. Desde o aparecimento destes hardwares programáveis, os algoritmos podem ser implementados em circuitos integrados em um espaço de tempo menor.

Nesta seção, foram apresentadas as bases teóricas que norteiam o desenvolvimento deste trabalho. Foram abordadas características que possibilitam explorar diferentes tipos de implementações de RNAs em FPGA. Na seção seguinte será detalhada a arquitetura do sistema embarcado proposto, o seu desenvolvimento e as otimizações realizadas.

5 MATERIAIS E MÉTODOS

Com o propósito de desenvolver um sistema computacional adaptativo capaz de configurar diferentes topologias, considerando, para isto, uma arquitetura de uma rede neural *Multilayer Perceptrons*. As possibilidades de configurações topologias para esta arquitetura são várias, sendo efetuadas por meio da reconfiguração dinâmica parcial do FPGA e de instruções previamente conhecidas e ordenadas.

Para a reconfiguração dinâmica parcial, deve-se, inicialmente, maximizar a lógica que permanecerá inalterada (estática) e minimizar a lógica que precisa ser modificada durante a execução da aplicação (parcial). Esta maximização, dá-se pelo particionamento da aplicação em blocos funcionais que são, em sua maior parte, comuns a todas configurações utilizadas na implementação da aplicação. Esses blocos representam as partes da configuração que não mudam, e, portanto, podem ser implementadas como um circuito estático.

O segundo passo, na configuração da rede neural, é especificar nas instruções as informações sobre as quantidades de camadas, de neurônios por camada, de entradas, assim como a localização dos pesos sinápticos e *threshold* na memória e como os neurônios artificiais são conectados. Na especificação destas instruções foram utilizadas tanto a metodologia de desenvolvimento *Design RTL* (projeto ao nível de transferência entre registros), que separa parte operativa (*Datapath*) e unidade controladora (Controlador), quanto a metodologia de síntese *Bottom-Up*, que requer um (*netlist*) separado para cada partição reconfigurável do circuito (NEDJAH et al ,2009).

Assim, antes de partir para a descrição do projeto da malha neural, alguns pontos importantes que dizem respeito à notação numérica, ao paralelismo e à função de ativação devem ser esclarecidos.

Quanto à notação numérica, foi utilizada, para os valores de entrada, de saída, dos pesos sinápticos e do *bias*, a notação em ponto fixo, não por falta de suporte aos valores não-inteiros das ferramentas de síntese, mas pelo presumível aumento no número de CLBs a serem empregados, caso fosse utilizada uma notação em ponto flutuante.

Já com relação ao paralelismo, procurou-se preservá-lo entre os neurônios da mesma camada, entre as suas ligações e, internamente, entre os blocos que o compõem.

E quanto à função de ativação, foram implementadas as funções sigmóide e tangente hiperbólica, por meio da técnica de aproximação por interpolação linear.

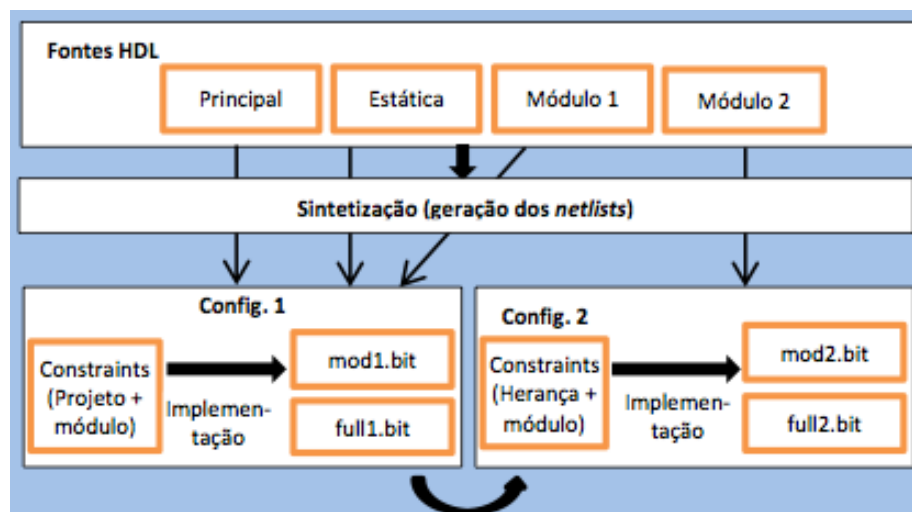
5.1 FLUXO DE PROJETO PARA RECONFIGURAÇÃO PARCIAL

Para o projeto da malha neural foram utilizadas as ferramentas ISE 14.2 e PlanAhead 14.2, ambas da Xilinx, e a metodologia de reconfiguração parcial baseada em módulos. Nos últimos anos, essa metodologia tem passado por grandes mudanças, especialmente depois dos *bus macros*, os quais permitem o roteamento manual necessário para comunicação dos módulos. O PlanAhead, como ferramenta, permite a criação das partições reconfiguráveis no FPGA e das configurações para cada região reconfigurável. Além disso, para que a reconfiguração seja possível, é necessário que o FPGA possua suporte à reconfiguração parcial e dispor de uma porta de comunicação específica para isso, como o *Internal Configuration Access Port (ICAP)*.

Cada configuração reúne a parte estática do FPGA – gerenciadores de clock, processadores, barramentos – com a parte dinâmica de uma região específica. Quando uma configuração altera somente uma única região, ela importa dados sobre pinos das outras partições com as quais ocorre a comunicação. A partir de cada configuração é possível gerar um *bit file* que poderá ser carregado para reconfigurar parcial ou totalmente o FPGA.

O fluxo de um projeto de reconfiguração parcial (Figura 16), adotado por esta Tese, consiste, inicialmente, na descrição das funções que o hardware deve executar por meio de uma linguagem de descrição de hardware. Assim, é possível a sintetização dos módulos para que seja possível conectá-los, então, usando o PlanAhead.

Figura 16 – Fluxo da reconfiguração parcial

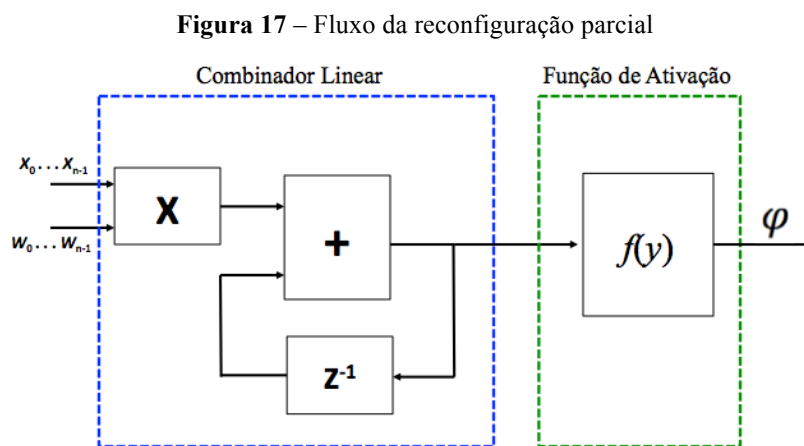


Antes da geração de cada *partial* ou *full bit file*, é necessário adicionar as *constraints* da placa

e do projeto, sendo este o arquivo responsável por definir os *delays* mínimos na comunicação entre os módulos; os pinos aos quais os sinais de entrada e saída estão ligados; as coordenadas das *slices* que delimitam cada região reconfigurável.

5.2 DESCRIÇÃO ESTRUTURAL DO NEURÔNIO EM FPGA

Tendo como referência o neurônio artificial proposto por McCulloch e Pitts (1943) (HAYKIN, 2001), representado na Figura 1, seção 2.1, a concepção em hardware do neurônio artificial (Figura 17), proposta nesse trabalho, seguiu os modelos arquiteturais implementado por SILVA (2009) e PEDRONI (2010).



Fonte: Elaborada pelo autor.

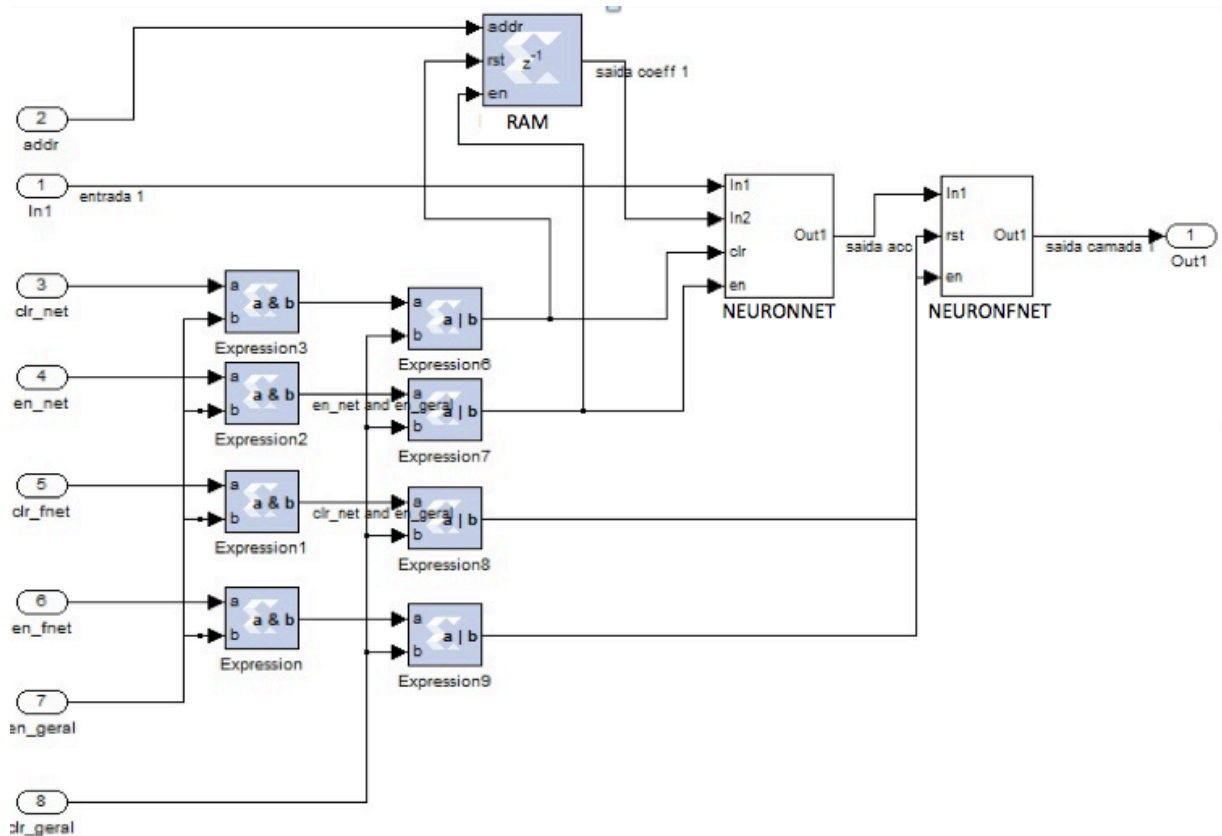
A arquitetura do neurônio artificial, designada NEURON (Figura 18), página 60, foi dividida em dois blocos funcionais: o primeiro bloco é um combinador linear, responsável pelo somatório das entradas ponderadas pelos pesos sinápticos; o segundo bloco é responsável pelo cálculo da função de ativação, denominados, respectivamente de blocos NEURONNET e NEURONFNET. A descrição em VHDL destes blocos baseou-se em uma abordagem de *Design RTL*, com a inclusão de registradores para a sincronia dos fluxos de dados.

O bloco NEURONNET, visualizado na Figura 19, página 61, exibe um fluxo de dados (*data flow*) entre os blocos MULTIPLICADOR e ACUMULADOR. Este bloco calcula o campo local induzido (somatório das multiplicações dos pesos sinápticos com os valores de estrada) com até 16 entradas de 32 *bits* que lhe são impostas.

Como observado no *data flow* do neurônio artificial visualizado na Figura 1, seção 2.1, o cálculo realizado no neurônio é dado pela multiplicação de todas as entradas, x_1 a x_n ,

pelos seus correspondentes pesos sinápticos, w_1 a w_n , sucedido pelo somatório dos valores resultantes destas multiplicações mais o valor do *threshold*. De forma análoga, Silva et al. (2015) implementou um neurônio artificial em FPGA, cuja arquitetura, completamente paralela, tem um multiplicador exclusivo para toda entrada, a fim de realizar simultaneamente a ponderação dos valores de entrada com os pesos sinápticos.

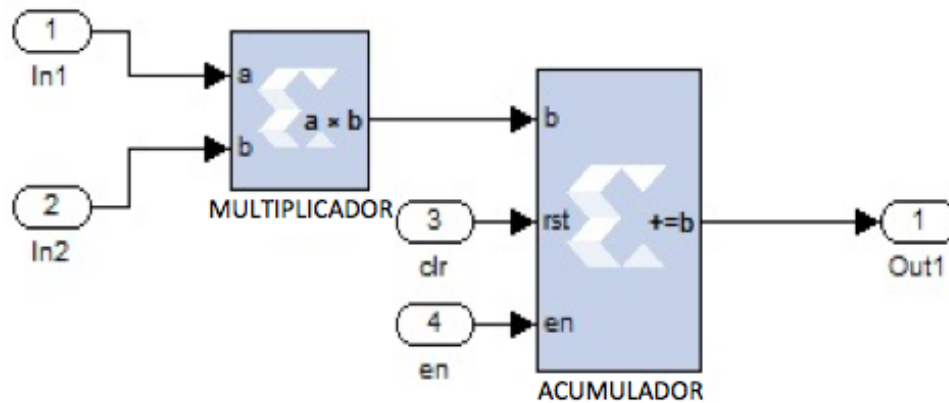
Figura 18 – bloco NEURON



Fonte: *print screen* do bloco NEURON no software Matlab.

Diferentemente, a proposta implementada neste trabalho realiza o cálculo do campo local induzido de forma serial, usando, para isto, os blocos multiplicador, acumulador e registrador, sendo este último interno ao bloco acumulador. Inicialmente, o registrador que armazena os resultados parciais é zerado pelo *reset*. Em seguida, o multiplicador realiza o produto das sucessivas amostras do sinal de entrada x_i pelo respectivo peso sináptico w_i . Por fim, utiliza-se o acumulador para realizar as somas, armazenando-as no registrador na próxima borda positiva do *clock*. Todavia, salienta-se que ao adotar esta estratégia o projeto se beneficiará de mais área de hardware, em compensação perderá em eficiência.

Figura 19 – bloco NEURONNET



Fonte: *print screen* do bloco NEURONNET no software Matlab.

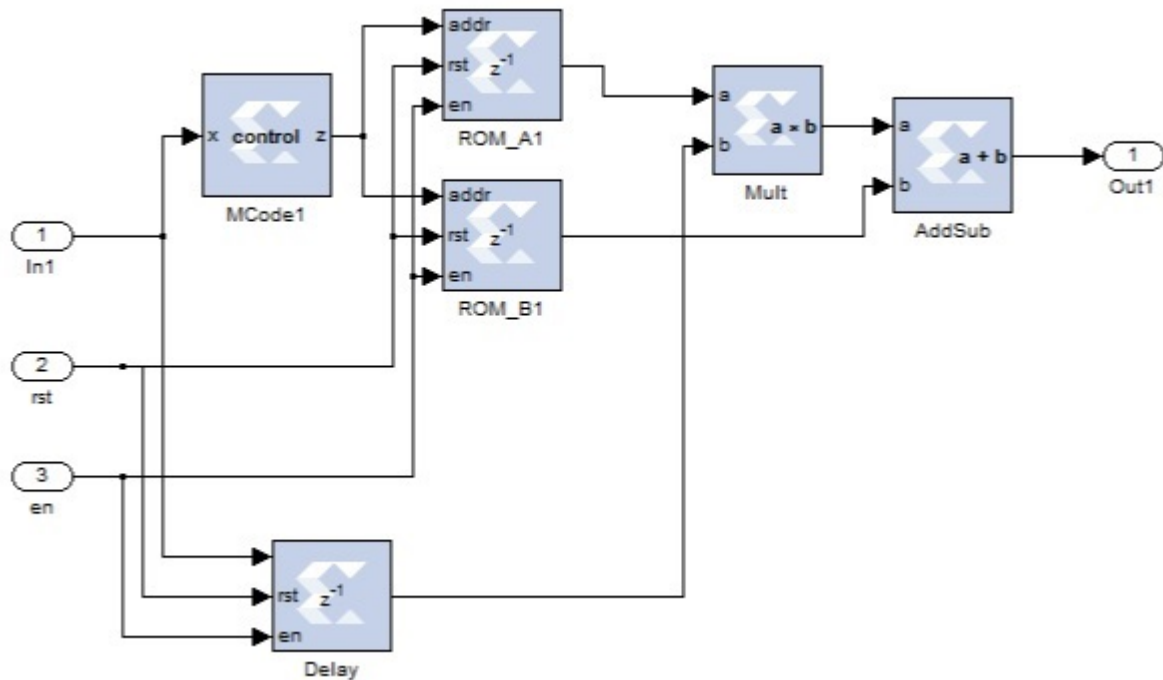
Para a conclusão do bloco NEURON, após o cálculo do campo local induzido, prossegue-se com o cálculo da função de ativação, no bloco NEURONFNET. Para este cálculo, pode-se optar pelo uso das funções de ativação sigmóide ou tangente hiperbólica. A escolha por uma dessas funções, prossegue-se com o carregamento do *bitstream* do bloco NEURON que contém uma dessas funções.

Seguindo o *data flow* do bloco NEURONFNET, visualizado na Figura 20, página 62, até a geração do sinal de saída, pode-se visualizar os blocos que o compõem. Estruturalmente, este bloco é constituído por um comparador, uma unidade de atraso, duas ROMs paralelas de 32 x 21 *bits* de dados, um multiplicador e um somador. Com relação as ROMs, relaciona-se o emprego da ROM_A1 ao armazenamento dos valores correspondentes aos coeficientes angulares de segmentos de retas, usados para a interpolação linear das funções sigmóide e tangente hiperbólica. Já o emprego da ROM_B1, liga-se ao armazenamento dos valores correspondentes aos coeficientes lineares dos mesmos segmentos de reta. Desse modo, os blocos multiplicador e somador ficam responsáveis pelo cálculo de saída do bloco NEURONNET, a partir dos valores obtidos das duas ROMs. O motivo pelo qual se escolheu essa estrutura para simular as funções de ativação sigmóide e tangente hiperbólica está relacionado ao custo de área requerida e, conseqüentemente, a dificuldade de implementá-las em FPGA.

Nesses termos, o cálculo do valor de saída do bloco NEURONFNET é executado da seguinte maneira: a partir do valor de entrada, proveniente do bloco NEURONNET, é definido um endereço no bloco comparador comum às duas ROMs, nas quais estão presentes os correspondentes coeficientes angular e linear do segmento de reta, a serem usados pelos blocos multiplicador e somador, para a geração do sinal de saída.

A saída, independente da função de ativação escolhida, foi montada a partir de uma tabela de 21 pontos, e os seus pontos intermediários foram obtidos por meio de uma interpolação linear. Na aquisição dos 21 pontos, utilizou-se uma técnica de inteligência computacional conhecida como algoritmos genéticos, cuja finalidade é a minimização do erro de cada indivíduo com base em uma função objetivo.

Figura 20 – bloco NEURONFNET



Fonte: *print screen* do bloco NEURONFNET no software Matlab.

5.3 REDE DE CONEXÃO

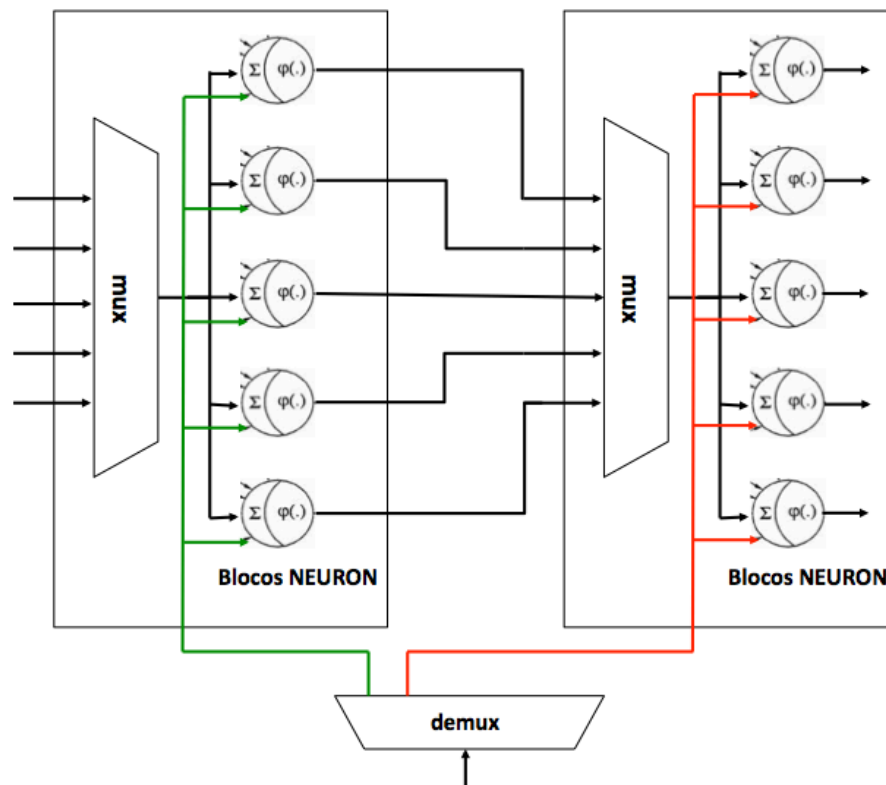
Para este trabalho foram analisadas algumas propostas de redes de interconexão já existentes e largamente utilizadas em projetos de sistemas digitais, tais como: barramento, *crossbar* e multiestágios (PRADO et al, 2010). Embora existisse a possibilidade de se optar por alguma dessas redes, observou-se que elas são apropriadas para aplicações com multiprocessadores acessando bancos de memória e não para aplicações que envolvam RNAs, em que o neurônio artificial de uma camada n precise se conectar com todos os neurônios artificiais da camada $n+1$, preservando, portanto, o paralelismo do processamento de cada neurônio artificial de cada camada específica.

Diante do exposto, houve a necessidade de desenvolver uma rede de conexão própria (Figura 21), página 63, com base em uma topologia em barramento para a transmissão dos sinais de controle oriundos da máquina de estado e que atendesse ao propósito da RDP, cuja

quantidade de camadas e neurônios artificiais não fossem fixas, e sim variáveis dependendo da necessidade da rede neural. Para isto, foram utilizados componentes digitais que permitissem tanto a inserção e a remoção de novos blocos de neurônios quanto a comunicação ordenada entre os neurônios artificiais de uma camada para outra sucessora. Para essa rede de conexão, pode-se ter até 256 neurônios artificiais dispostos em 16 camadas que podem suporta até 16 neurônios.

O processo de funcionamento da Rede de Comunicação proposta está condicionado a adicionar ou a remover blocos de neurônios artificiais através da RDP e a realizar simultaneamente todas as conexões necessárias para conectar as saídas de todos os neurônios artificiais de uma camada n às entradas de todos os neurônios de uma camada $n+1$, por meio de um *mux*.

Figura 21 – rede de conexão



Fonte: Elaborada pelo autor.

Devido o cálculo do campo local induzido ser realizado de forma sequencial no bloco NEURONNET, foi preciso adicionar um *mux* para cada camada existente ou que venha a surgir. Além de receber todos os sinais de saída dos neurônios artificiais de uma camada n , o *mux* serve ainda para selecionar a saída do neurônio x_i da camada $n-1$ para todos os neurônios artificiais da camada n da seguinte forma: em um certo instante de tempo t , todos os neurônios

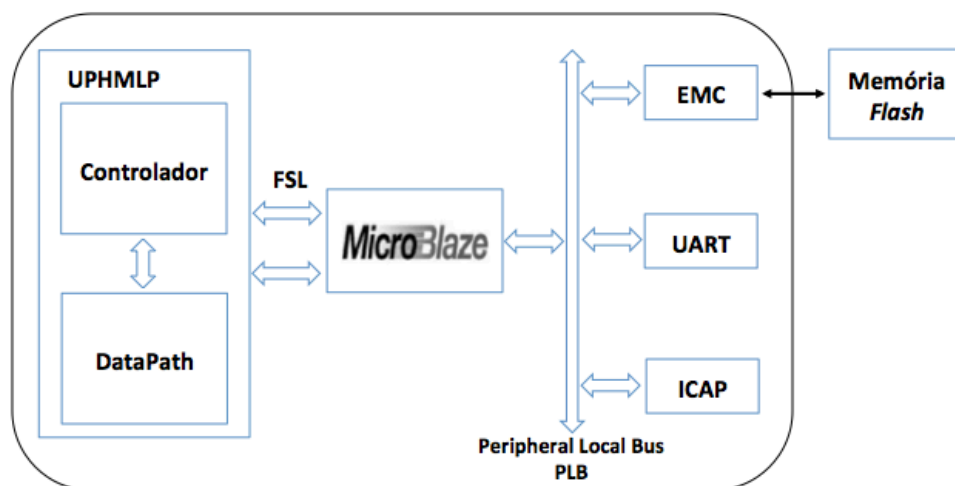
artificiais da camada n receberão como entrada a saída do primeiro neurônio da camada $n-1$; no próximo instante de tempo $t+1$, os neurônios da camada n receberão como entrada o sinal de saída do segundo neurônio da camada $n-1$ e; no último instante de tempo, que equivale a ter alcançado o último neurônio da camada $n-1$, será fornecida como entrada para os neurônios da camada n o sinal de saída do último neurônio.

A responsabilidade de especificar qual a camada estará ativada e quais estarão desativadas em um certo instante de tempo é do *demux* que envia um sinal de *enable*, gerado a partir do bloco contador de camadas.

5.4 ARQUITETURA COMPLETA

Com o intuito de simplificar as configurações das RNAs e possibilitar uma maior generalização da arquitetura (Figura 22), decidiu-se por uma solução *co-design*, em que é possível decompor a funcionalidade do sistema em duas partições: o Sistema de Carga e Controle (SCC) e a Unidade de Processamento em Hardware da MLP (UPHMLP).

Figura 22 – Arquitetura completa



Fonte: Elaborada pelo autor.

O SCC, implementado em linguagem C e executado no processador Microblazer de 32 *bits*, é responsável pelo carregamento e controle dos dados, bem como pela impressão dos dados no terminal através do componente *Universal Asynchronous Receiver/Transmitter* (UART). Ao Microblaze foram adicionados também os componentes *External Memory Controller* (EMC), responsável por ligar o processador a memória *flash*, na qual são

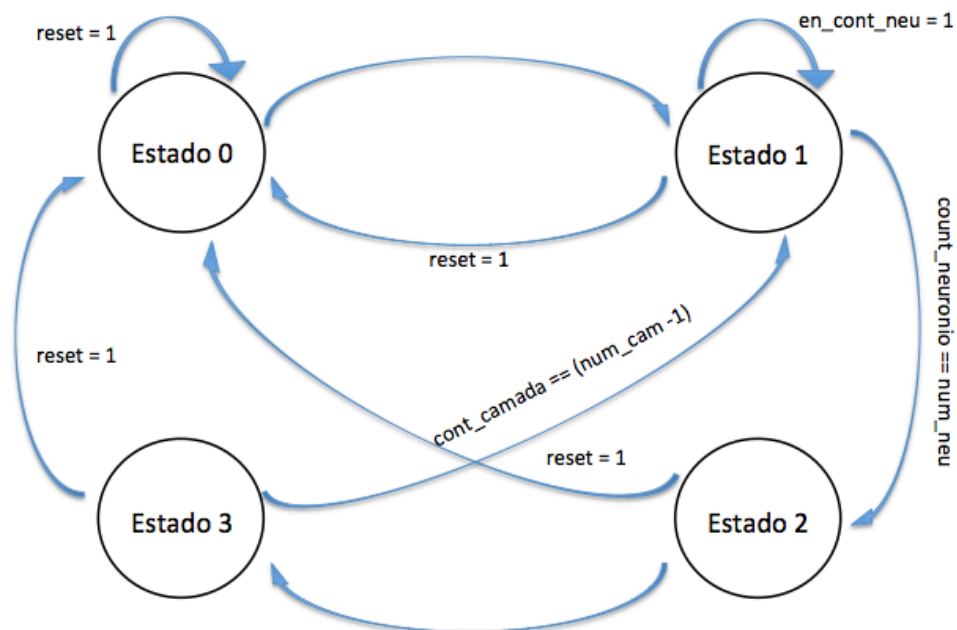
armazenados os arquivos parciais e o *Internal Configuration Access Port* (ICAP), responsável pelo recebimento a partir do processador de um *bitstream* parcial, previamente carregado da memória *flash*, e pela configuração do módulo de hardware correspondente [17].

Já a UPHMLP consiste de dois blocos funcionais implementados na linguagem de descrição de hardware VHDL: o primeiro, representado pelo Controlador, é responsável pela solicitação e recebimentos dos dados ao SCC, por meio da Máquina de Estado (MdE); o segundo, representado pelo *DataPath*, é um núcleo adaptativo e reconfigurável, com uma arquitetura paralela podendo ser formada com até 16 camadas física. Observa-se ainda que a UPHMLP está ligada ao MicroBlaze por meio do barramento *Fast Simplex Link* (FSL), que é um canal unidirecional ponto-a-ponto

5.4.1 Controlador

O modelo computacional *State-Chart* foi empregado na representação da Máquina de Estado (MdE) (Figura 23), para possibilitar hierarquia e facilitar a implementação do Controlador.

Figura 22 – Controlador (MdE)



Fonte: Elaborada pelo autor.

Na MdE (Figura 23), visualizam-se quatro estados. No estado 0 (zero), inicializa-se todos os componentes do sistema, quando o sinal de *reset* for igual a 1, a MdE não mudará de

estado. Caso contrário, se o reset for diferente de 1, a MdE passará para o estado 1.

No estado 1, a MdE habilitará o bloco NEURONNET de todos os neurônios de uma camada, o qual irá realizar a soma ponderada com os pesos sinápticos de forma sequencial. Neste Estado, o bloco NEUROFNET encontra-se desabilitado. Quando o sinal do controlador de neurônio for igual ao número de neurônios por camada, significa que todos os neurônios finalizaram o cálculo do campo local induzido, passando para o Estado 2 .

O Estado 2 é um estado de sincronismo, que tem por finalidade segurar o sinal do bloco NEURONNET de todos os neurônios artificiais de uma camada. Após a finalização do processamento do sinal, passa-se para o último estado, Estado 3.

O Estado 3 caracteriza-se por realizar o cálculo da função de ativação. A execução deste cálculo será realizada até que todos blocos NEURONFET de todos os blocos NEURONS da camada que está sendo executada finalizem o processo.

5.4.2 Datapath

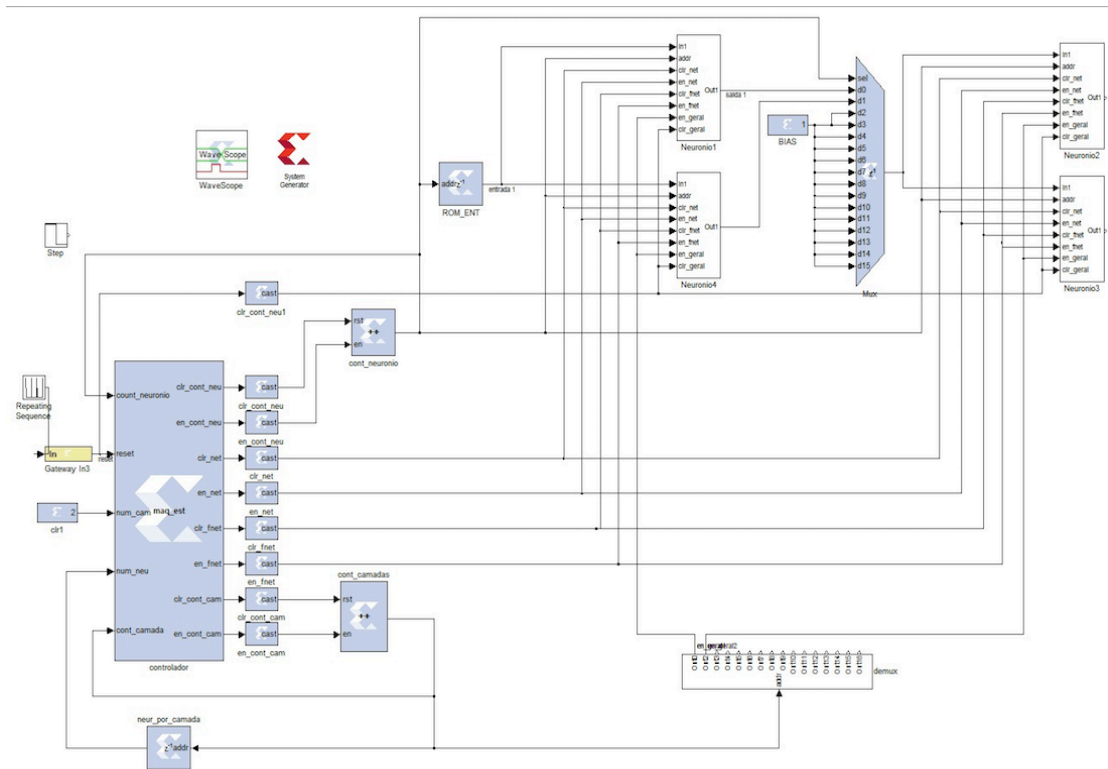
O *Datapath* é composto, inicialmente, de um núcleo adaptativo e reconfigurável, com uma arquitetura paralela interna estruturada para suportar no máximo 16 camadas com até 16 blocos NEURONS, totalizando 256 blocos NEURONS na rede neural. Ele exibe como os componentes estão conectados.

No *Datapath*, pode-se, ainda, acrescentar outros blocos neuronais, por meio dos *bitstreams* parciais, para executar uma ou mais camadas com até 16 blocos NEURONS (neurônios artificiais). A responsabilidade de execução e de coordenação de toda esta modularidade é do Controlador. Ele se encarregará pela dinâmica de inserção e remoção dos neurônios artificiais (blocos NEURONS). Para auxiliar o controlador na tarefa de inserção de blocos NEURONS, troca dos pesos, das entradas e dos *thresholds* foi necessário inserir uma memória BRAM do tipo *dual-port* de 16 *bits* e 64 endereços para leitura e escrita dos pesos sinápticos de cada neurônio a ser utilizado como entrada para a próxima camada.

Na Figura 24, página 67, visualiza-se a arquitetura do *Datapath* e a disposição do fluxo de dados entre os componentes. Observa-se que todos os componentes, exceto os blocos *demux* e *mux*. são controlados por sinais de controle vindos do controlador (Sinais da MdE), por meio de uma estrutura em barramento, cujo canal de comunicação unidirecional é baseado em *FIFO*. A ação que será realizada por cada componente interno será determinada pela MdE, como também seu momento de execução. A MdE mantém o sincronismo dos componentes para que os dados trafeguem corretamente entre eles. O sinal de *clock*, que não

está sendo mostrado na figura, é interno a todos os componentes que requerem sincronismo.

Figura 24 – Datapath



Fonte: print screen do Datapath no software Matlab.

Ainda na Figura 24, observa-se que o bloco contador de neurônios e o contador de camada estão ligados ao controlador (MdE). A finalidade do contador de camada é enviar um sinal de controle para o *demux* poder controlar a camada que ficará ativa e as que ficarão desativadas. O mesmo sinal que sai para o *demux*, servirá de entrada para a MdE e para o bloco neurônio por camada. No bloco neurônio por camada será atribuído a quantidade de neurônios que cada camada conterá, e que a saída deste bloco servirá de entrada para a MdE.

Já o bloco contador de neurônio também envia um sinal de controle, o sinal que sai deste bloco servirá para habilitar em cada instante de tempo um neurônio que enviará seu sinal de saída para as entradas dos neurônios da camada seguinte. No próximo instante de tempo, o controlador de neurônio habilitará a saída do segundo neurônio da primeira camada, que servirá de entrada para todos os neurônios da próxima camada. Todo este processo de seleção de sinais de saída dos neurônios é controlado pelo *mux*.

Nesta seção foram apresentados os métodos para a implementação da arquitetura em FPGA, a partir da descrição estrutural do neurônio artificial e de seus blocos internos. Foram abordados ainda, o funcionamento da arquitetura, sua estrutura física, o controlador e os passos necessários para o controlador rodar redes MLPs. Na próxima seção serão

apresentados os dados dos testes, simulações e resultados, das implementações realizadas na arquitetura proposta.

6 RESULTADOS E DISCUSSÕES

A especificação em VHDL da rede MLP proposta em hardware foi sintetizada para o FPGA Virtex-6 de referência XC6VLX240T, por meio da ferramenta XST *Synthesis* do software Xilinx ISE 13.2. Após a síntese, o Xilinx PlanAhead 13.2 foi utilizado para gerar os *bitstreams* parciais, correspondentes a todas as configurações necessárias para cada região reconfigurável. Já para as simulações, o ISim Simulator foi usado para obtenção dos resultados.

Na Tabela 2 é possível visualizar os comparativos de área de hardware requerida por um único bloco NEURON para diferentes números de entradas, bem como a área de hardware necessária para implementar uma rede MLP em sua totalidade, levando em conta o número máximo de entradas (i), de neurônios por camada (n) e de camadas (l). Estes comparativos foram realizados em termos de áreas necessárias no FPGA, as quais são representadas por *slices*, contendo quatro LUTs e oito *flip-flops*. As taxas de ocupação obtidas pela arquitetura proposta (A) foram comparadas com os resultados das implementações de Prado (2011) que propõe uma arquitetura capaz de executar redes MLP **por meio** do reuso dos neurônios artificiais e de Martins (2011) que apresenta uma arquitetura capaz de alterar qualquer topologia de rede MLP em tempo de execução, mas que só utiliza a função de ativação Sigmóide, enquanto que a arquitetura proposta neste trabalho permite utilizar tanto a função de ativação Sigmóide quanto a Tangente Hiperbólica, em tempo de execução por meio da RDP.

Tabela 2 - Comparativos de áreas ocupadas pelo Bloco NEURON e pela rede MLP.

i	n	l	Área do Neurônio (<i>Slices</i>)			Área da MLP (<i>Slices</i>)		
			(A)	(B)	(C)	(A)	(B)	(C)
2	6	3	10	8	6	24	21	11
4	9	5	16	12	9	84	76	41
8	13	7	22	18	15	452	423	129

Arquitetura Proposta (A) Arquitetura (PRADO, 2011) (B) Arquitetura (MARTINS, 2011) (C)

Na Tabela 3, página 69, resumem-se as considerações temporais relacionadas aos

ciclos de configuração da arquitetura, realizados pelo Controlador, para gerar as topologias de redes neurais MLP por meio da RDP, sendo, em seguida, comparada com a arquitetura proposta por Prado (2011), que embora utilize um hardware distinto, pode-se configurar a mesma topologia de rede MLP, dados de entrada e pesos sinápticos utilizados neste trabalho, semelhanças que não foram encontradas em outros trabalhos correlatos.

Tabela 3 - Comparativos de áreas ocupadas pelo Bloco NEURON e pela rede MLP.

Topologia da rede MLP (entrada/oculta1...n/saída)	Arquitetura (A)			Arquitetura (B)		
	ciclos	Tempo (ms)	<i>slices</i>	ciclos	Tempo (ms)	<i>slices</i>
2 / 25 / 12	327	16.406	6231	389	17.656	7975
8 / 5 / 5 / 5 / 3	129	16.953	4402	166	18.751	7975
8 / 5 / 5 / 5 / 5 / 3	184	18.257	4764	212	19.546	7975

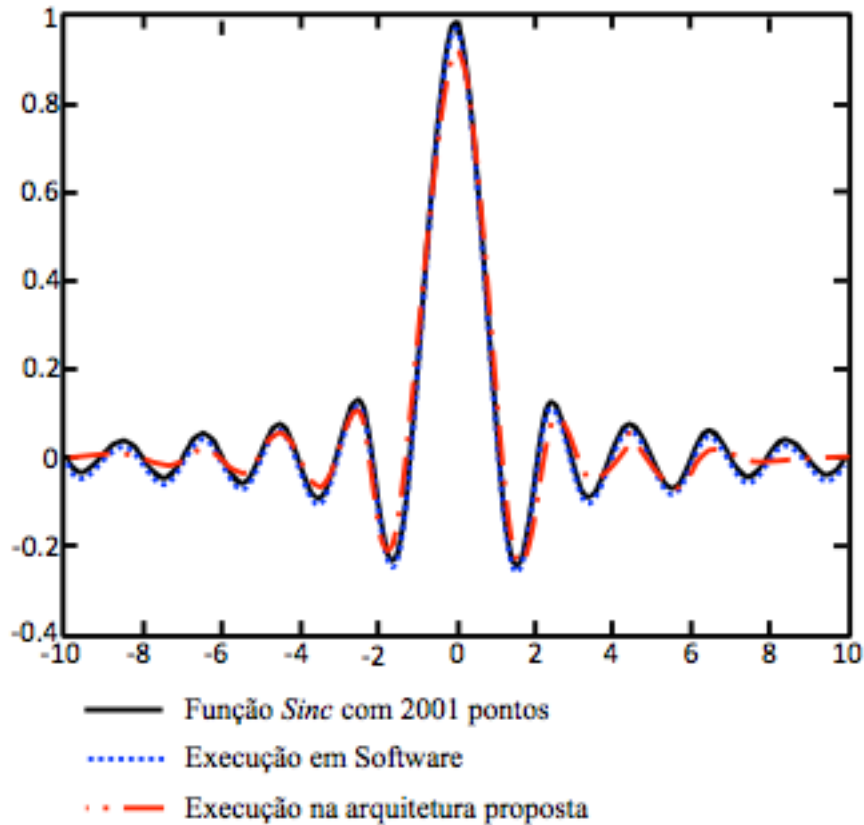
Arquitetura Proposta (A) Arquitetura (PRADO, 2011) (B)

Nota-se que para cada topologia tem-se uma quantidade de ciclos diferente para executar a rede, conseqüentemente o tempo de configuração difere de uma rede para outra.

Com isto, pode-se constatar que o tempo usado para reconfigurar parcialmente a rede neural depende do tamanho do fluxo parcial de *bits* que por sua vez depende do tamanho do bloco reconfigurável. Portanto, o tamanho da região reconfigurável foi selecionado de modo a permitir a configuração de, no máximo, 32 blocos NEURON, com um fluxo de *bits* correspondendo a 45.089 bytes e com um tempo de reconfiguração variando de 6,2 ms para redes abaixo de 50 ciclos e de 13,8 ms para redes com mais de 100 ciclos.

Além das análises comparativas realizadas, a viabilidade da arquitetura do nosso sistema foi submetida a resolver quatro problemas da área de redes neurais artificiais, dentre eles a interpolação da função *sinc* (Gráfico 6), resolução da função XOR e dois problemas de identificação de padrões, aqui chamados de problema da “borboleta” (Figura 25(a)) e de problema da “bandeira” (Figura 25 (b)), nos quais algumas regiões separadas em um plano devem ser classificadas. Todas essas redes MLP foram previamente treinadas (*off-line*) e testadas em software utilizando uma heurística simplificada. A heurística utilizada para definir a topologia foi criar pequenas RNAs, com alguns *perceptrons*, aumentando a quantidade de *perceptrons* até que o erro de saída das RNAs fosse mínimo.

No Gráfico 6, visualiza-se o comparativo da função *sinc* implementada em software com as sínteses das arquiteturas propostas realizadas no FPGA.

Gráfico 6 - Função *sinc*.

Fonte: *print screen* da função *sinc* no software Matlab.

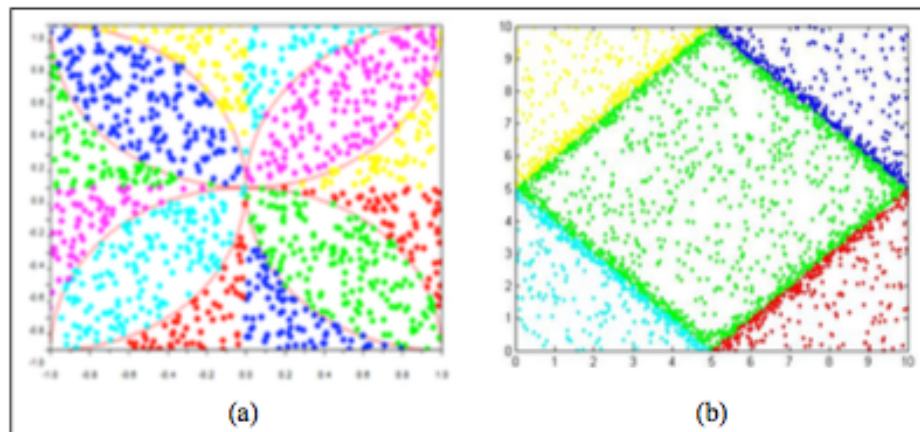
Observa-se, portanto, que fora do intervalo $-3 < x < 3$ os resultados das sínteses divergem da função *sinc* executada em software por estarem utilizando notação numérica em ponto fixo. A primeira arquitetura MLP, sintetizada no FPGA, foi estruturada para operar com 3 neurônios na camada oculta e 1 neurônio na camada de saída, apresentando assim um erro relativo de 0,34%. O erro relativo de 0,16%, presente na segunda arquitetura, foi obtido por causa do acréscimo de mais 2 neurônios na camada oculta por meio da reconfiguração dinâmica parcial, finalizando, portanto, uma arquitetura com 6 neurônios, dos quais 5 neurônios se localizam na camada oculta e 1 neurônio na camada de saída.

Outra arquitetura de rede neural MLP implementada em hardware obteve, para o problema da classificação de padrões (Figura 25 (a)), um nível de acerto de 94,97%, utilizando dois nós na camada de entrada, uma camada oculta com 25 neurônios e 12 neurônios na camada de saída. Entretanto, após a modificação do sistema referido, com a finalidade de adequar uma nova topologia de rede neural MLP (Figura 25 (b)), houve uma

redução no número de blocos NEURON de 25 para 12 na camada oculta e na camada de saída de 12 para 5 neurônios. Diante disto, pode-se constatar o mesmo percentual de acerto da (Figura 25 (a)).

Para as duas classificações, cada neurônio na camada de saída sinaliza uma determinada classe que identifica a região na qual o ponto de entrada se localiza. Para isto, foram usados os pontos situados nas regiões de fronteiras entre as classes, considerados os mais difíceis ou críticos de identificar.

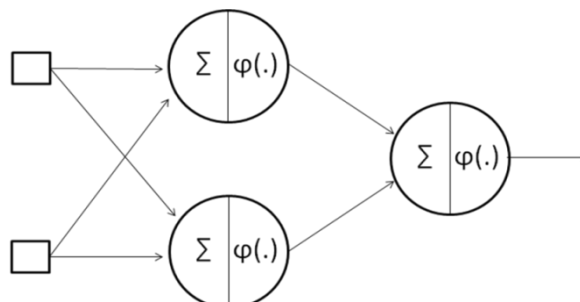
Figura 25 - Imagens usadas na classificação de padrões pela rede MLP.



Fonte: *print screen* da tela do software Matlab.

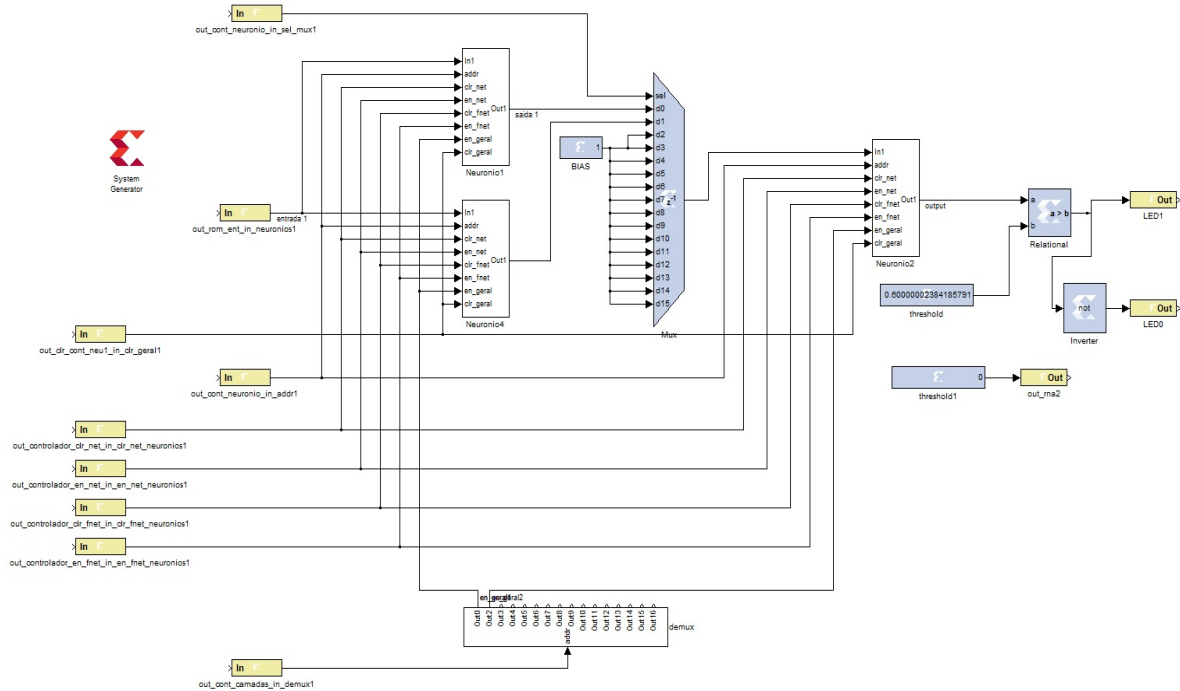
Para a execução do XOR, utilizou-se uma topologia, para a malha neural, com duas entradas, dois neurônios na camada oculta e um neurônio na camada de saída, representado na Figura 26. Apesar de simples, o problema foi suficiente, também, para mais uma vez validar a arquitetura da malha neural em FPGA, visto que os pesos e *bias* foram extraídos do treinamento realizado no MATLAB, utilizando-se de uma topologia neural idêntica à desenvolvida em hardware, página 72, Figura 27.

Figura 26 – Topologia da rede para a resolução do problema do XOR.



Fonte: Elaborado pelo autor.

Figura 27 – Topologia da rede em hardware para a resolução do problema do XOR.



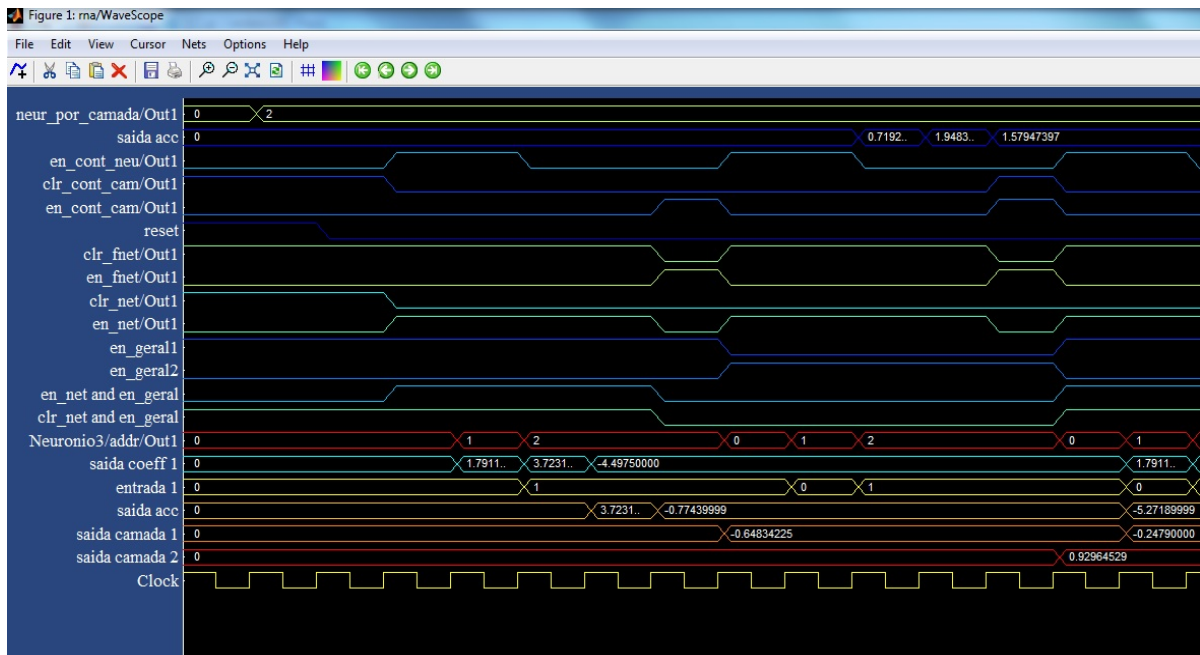
Fonte: *print screen* da tela do software are Matlab.

Observa-se que a implementação da malha neural, a partir do modelo fornecido, viabilizou a resolução do problema do XOR; e os valores das saídas obtidos, visualizados no Quadro 4 validam a implementação feita no FPGA, apresentando uma aproximação extremamente relevante para a saída da malha neural, com erro relativo máximo de apenas 0,25%. Na Figura 28, página 73, pode-se visualizar um dos resultados das simulações do XOR.

Quadro 4 – Resultados comparativos da malha neural utilizada para execução da função XOR.

Entrada	Saída desejada	Saída do MATLAB ponto flutuante	Saída da Malha Neural
00	0	0.0145	0.0163
01	1	0.9187	0.9296
10	1	0.9233	0.9357
11	0	0.0136	0.0144

Figura 28 – Resultado da simulação do XOR.



Fonte: *print screen* da tela do software Matlab.

Nos Quadros 5 e 6 são apresentados o total de dispositivos usados e o total de área ocupada no FPGA xc6vlx240t-1ff1156 pela malha neural na resolução do XOR. Como se pode observar tanto na partição estática (Figura 29) quanto na partição reconfigurável (Figura 27), as taxas de uso são baixas, evidenciando, dessa forma, que podem ser implementadas redes muito maiores.

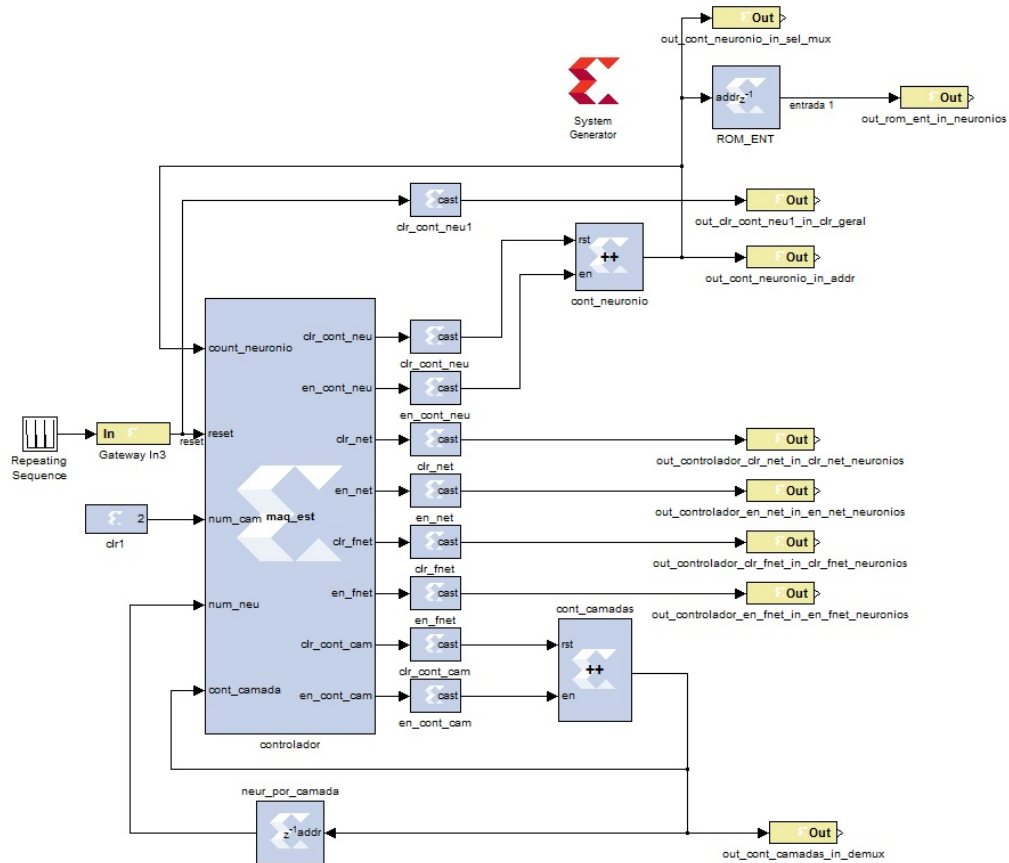
Quadro 5 – Taxas de ocupação do FPGA para a partição estática.

Partição estática	
Dispositivo	xc6vlx240t-1ff1156
Elementos lógicos (slices)	3/301440(0%)
Número de LUTs	11/150720(0%)
Total de Pinos	37/600(6%)
Número de BUFG	1/32(3%)

Quadro 6 – Taxas de ocupação do FPGA para a partição reconfigurável.

Partição reconfigurável	
Dispositivo	xc6vlx240t-1ff1156
Elementos lógicos (slices)	225/301440(0%)
Número de LUTs	685/150720(0%)
Total de Pinos	0/600(6%)

Figura 29 – Partição estática da malha neural.



Fonte: *print screen* da tela do software Matlab.

Nesta seção foram analisados e discutidos os resultados obtidos com a arquitetura final, detalharam-se todos os testes realizados com a arquitetura em todas as suas fases de desenvolvimento. Foram realizados comparativos com arquiteturas em hardware de outros autores. Na próxima seção serão apresentadas as conclusões, algumas considerações finais e encaminhamentos para trabalhos futuros.

CONCLUSÃO

Este estudo mostrou a viabilidade de descrever uma arquitetura para um sistema de redes neurais artificiais, capaz de configurar e reconfigurar várias topologias de redes MLP através do método de reconfiguração parcial presente no FPGA Virtex -6. Com este sistema, a topologia de configuração (número de neurônios por camada e número de camadas) pode ser reprogramado e alterado em campo, sem qualquer esforço extra, por meio de instruções bem simples.

A arquitetura desenvolvida é composta de uma máquina de estados finitos como controlador, um datapath com vários componentes digitais de uso geral e específicos, de blocos NEURONS e de uma rede de comunicação específica para gerenciamento das conexões para o reuso dos blocos NEURONS em uma rede neural artificial do tipo MLP.

Neste contexto, foi realizada uma descrição do bloco NEURON, usando a notação numérica de ponto fixo e empregando a estratégia de reconfiguração parcial para implementar as funções Sigmóide e Tangente Hiperbólica. Para essas funções, os dados foram obtidos a partir de uma interpolação linear, utilizando *Lookup Table*. Ao especificar o bloco NEURON, constituído pelos blocos de NEURONNET e NEURONFNET, verificou-se que, neste estudo, as implementações das redes neurais tornam-se mais modulares, possibilitando, facilmente, aumentar e reduzir o número de neurônios e também a estrutura de rede. Como resultado, foram criados os módulos parciais (*bitstreams* parciais) de redes neurais completas em Virtex-6 FPGA, com a precisão numérica adequada e elevada capacidade de processamento paralelo.

Além disso, o sistema minimiza tanto o tempo de execução quanto a área de silício necessária, uma vez que instancia apenas os neurônios que são necessários para operar em um certo instante de tempo, por meio da RDP. Isto é feito sem deteriorar o tempo de inferência da rede neural. Implementado parte em VHDL, parte com os blocos do DSP Builder da Xilinx, o sistema foi sintetizado e simulado pela ferramenta ISE 13.2 a fim de validar suas funcionalidades. Durante a fase de síntese do projeto foi possível, em relação aos requisitos do sistema, avaliar questões de tempo e área ocupada no FPGA.

A temática em estudo contribui não apenas para o meio industrial, mais, também, para o âmbito acadêmico bem como na de pesquisa em expansão, favorecendo, portanto, o aumento das produções científicas.

Outra contribuição relevante proposta por este trabalho foi a criação de uma rede de comunicação de propósito específico para a malha neural, proporcionando tanto o reuso dos

blocos NEURONS quanto a escalabilidade dos mesmos, por meio dos sinais de controle oriundos da máquina de estado.

As limitações encontradas durante a execução deste trabalho estiveram relacionadas com a notação em ponto flutuante, não suportada pelas ferramentas de síntese, e, também, com a implementação da função de ativação do tipo tangente sigmóide e sigmóide. Com isso, as estratégias adotadas para superar tais limitações foram, primeiramente, usar a notação numérica em ponto fixo. O método utilizado para a realização do cálculo da função foi o uso de *lookup table* para o armazenamento dos valores da função de transferência obtidos, com realização *off-line* do treinamento da rede neural no MATLAB.

Os resultados obtidos pelas topologias MLP, aqui implementadas, permitiram qualificar os métodos e abordagens desenvolvidas neste estudo como capazes de serem transportados da fase de simulação para sistemas de tempo real, em conformidade com os requisitos estabelecidos para o projeto.

Os comparativos com a arquitetura proposta por Prado (2011), exibidos na seção 6, Tabela 3, demonstraram que a malha neural artificial aqui desenvolvida pode atuar em diversas pesquisas que aplicam RNAs, obtendo flexibilidade de topologia MLP com economia de elementos lógicos. É importante ressaltar que a arquitetura é fixa, mas podendo variar, entre 1 e 256, a quantidade de blocos NEURONS em qualquer aplicação em que for usada.

Uma proposição para trabalhos futuros envolvendo este sistema é a realização da fase de treinamento da rede neural em campo, ou seja, o usuário poderia definir um valor de erro máximo, para que não seja ultrapassado. Caso o limiar máximo seja atingido, o sistema iniciará o treinamento *online* da rede neural configurada pelo usuário, após isso, o sistema treinará a rede até que o erro esteja abaixo de um limiar mínimo, também definido pelo usuário.

Outro encaminhamento para trabalhos futuros seria adicionar um conjunto de registradores de atraso internos a rede de comunicação, os quais permitam tanto a realimentação com atraso quanto a manipulação dos dados de entrada em *pipeline*.

REFERÊNCIAS

- AMIN, H.; CURTIS, K.M.; HAYES–GILL, B.R. **Piecewise linear approximation applied to nonlinear function of a neural network**, IEE Proc. Circuits, Devices Sys., 1997, 144, pp. 313–317.
- ARAGÃO, A.; ROMERO, R.; MARQUES, E. **Computação Reconfigurável Aplicada à Robótica (FPGA)**. Disponível em: <<http://www2.eletronica.org/artigos/eletronica-digital/computacao-reconfiguravel-aplicada-a-robotica-fpga>>. Acesso em: 23 nov. 2009.
- BAKO, Laszlo et al. NEURAL NETWORK PARALLELIZATION ON FPGA PLATFORM FOR EEG SIGNAL CLASSIFICATION. **Interdisciplinarity In Engineering**, Romênia, p.370-370, 1 nov. 2012.
- BARROS, A. C. **Implementação em FPGA de um módulo multiplicador e acumulador aritmético de alto desempenho para números em ponto flutuante de precisão dupla, padrão IEEE 754**. 2008. 145 f. Dissertação (Mestrado) – CIN. Ciência da Computação, Universidade Federal de Pernambuco - UFPE, Recife-PE, 2008.
- BASTERRETXEA, K.; TARELA, J. M.; DEL Campo, I. **Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons**, IEEE Proc.-Circuits Devices Syst., Vol. 151, 2004.
- BEUCHAT, J.-L.; HAENNI, J.-O; SANCHEZ, E. **Hardware reconfigurable neural networks**. In 5th Reconfigurable Architectures Workshop (RAW'98), Orlando, Florida, USA, March 30 1998.
- BOBDA, C. **Introduction to Reconfigurable Computing: architectures, algorithms and applications**. Springer, 2007.
- BRAGA, A. L. S. **VANNGen** – Uma ferramenta CAD flexível para a implementação de redes neurais artificiais em hardware, 2005.
- BRAGA, A. P.; CARVALHO, A. C. P. L. F.; LUDEMIR, T. B. **Redes neurais artificiais: teoria e aplicações**. 2. ed. Rio de Janeiro: LTC, 2007.
- BRAGA, A. P.; CARVALHO, A. C. P. L. F.; LUDEMIR, T. B. **Redes neurais artificiais: teoria e aplicações**. 2. ed. Rio de Janeiro: Ltc, 2007. 226 p.
- CAGNI JÚNIOR, Eloi. **Software Inteligente Embarcado Aplicado à Correção de erro na Medição de Vazão em Gás Natural**. 2007. 72 f. Dissertação (Mestrado) - Departamento de Engenharia da Computação, Universidade Federal do Rio Grande do Norte - Ufrn, Natal, 2007.
- CAMPO, I. del; FINKER, R.; ECHANOBE, J.; BASTERRETZEA, K. **Controlled accuracy approximation of the sigmoid function for efficient FPGA-based implementation of artificial neurons**, 2013.

CHAVES, R. M. **Implementação em hardware da função de ativação do neurônio artificial utilizando instrução customizada para o processador NIOS II**. 2010. 129 f. Monografia (Graduação) – Universidade Federal de Lavras-MG, 2010.

COMPTON, K.; HAUCK, S. **Reconfigurable Computing: a survey of systems and software**. ACM Computing Surveys, v. 34, n. 2, june 2002. Disponível em: <<http://www.idi.ntnu.no/emner/tdt22/2011/reconfig.pdf>>. Acesso em: 5 de mai. de 2012.

COSTA, C. da. Elementos de lógica programável com VHDL e DSP: Teoria e Prática. São Paulo: Érica, 2011.

ELDREDGE, J. G. **Fpga density enhancement of a neural network through run-time reconfiguration**. 1994. Master's thesis, Department of Electrical and Computer Engineering, Brigham Young University, May 1994.

FERREIRA, A. P. A. **Implementação de uma arquitetura de Redes Neurais MLP utilizando FPGA**. 2008. 44 f. Monografia (Graduação) – CIN. Ciência da Computação, Universidade Federal de Pernambuco - UFPE, Recife-PE, 2008.

FINKER, R. et al. "Multilevel Adaptive Neural Network Architecture for Implementing Single-Chip Intelligent Agents on FPGAs", 2013 International Joint Conference on Neural Networks (IJCNN), vol., no., pp., 4-9 Aug. 2013.

GIRAU, B. FPGA: concepts and properties. In: OMONDI, A. R.; RAJAPAKSE, J. C. (Ed.). **FPGA implementations of neural networks**. Springer, 2006.

GOMES, V. C. F.; CHARÃO, A. S.; VELHO, H. F. C. **Field Programmable Gate Array - FPGA**. Disponível em: <<http://www.vconrado.com/chr/fpga.pdf>>. Acesso em: 19 dez. 2009.

GONÇALVES, H. S. B. **Desenvolvimento de ferramenta computacional para projetos de redes neurais artificiais**. 2005. 108 f. Dissertação (Mestrado) – Instituto Federal de Ciência e Tecnologia de São Paulo – IFSP, São Paulo, 2013.

GONZALEZ, J. A. Q. **Uma metodologia de projeto para circuitos com reconfiguração dinâmica de hardware aplicada a Support Vector Machines**. Tese de Doutorado. Escola Politécnica da Universidade de São Paulo, São Paulo, 2006.

HAGAN, M.; DEMUTH, H.; BEALE, M. **Neural Network Design**. PWS Publishing Company, Boston, MA, 1996.

HAYKIN, S. **Redes neurais: princípios e práticas**. 2. ed. Porto Alegre: Bookman, 2001.

KOJIMA, L. **Metodologia de projeto de sistemas dinamicamente reconfiguráveis**. Dissertação de Mestrado. Escola Politécnica da Universidade de São Paulo, São Paulo, 2007.

LIMA, A. A.; MARQUES, E. **Reconfigurabilidade Dinâmica e Remota de FPGAs**. Instituto de Ciências Matemática e de Computação – ICMC, Dissertação de Mestrado, USP, Agosto 2002.

LOPES, D. C. **Implementação e avaliação de máquina de comitê em ambiente com múltiplos processadores embarcados em um único chip**. Natal, RN, 2009.

LOPES, J. J. **ChipCflow** – uma ferramenta para execução de algoritmos utilizando o modelo a fluxo de dados dinâmico em hardware reconfigurável; São Carlos, SP, 2012, 233 folhas.

MARTINS et. al. **COMPUTAÇÃO RECONFIGURÁVEL: conceitos, tendências e aplicações**. Disponível em: <http://www.ppgee.pucminas.br/gsd/papers/martins_jai03.pdf>. Acesso em: 16 de abr. 2012.

MARTINS, R. S.; NEDJAH, N.; MOURELLE, L. M. “*Compact Yet Efficient Hardware Architecture for Multilayer-Perceptron Neural Networks*”, Revista Controle & Automação, vol. 22, no. 6, pp. 647-663, 2011.

MARTINS, C. A. P. da S. et al. **Computação Reconfigurável: conceitos, tendências e aplicações**. Disponível em: <http://www.ppgee.pucminas.br/gsd/papers/martins_eri02.pdf>. Acesso em: 05 jun. 2009.

MENOTTI, R. **LALP: uma linguagem para exploração do paralelismo de loops em computação reconfigurável**. Tese de Doutorado. Universidade de São Paulo, São Paulo, 2010.

MESQUITA, D. G. **Contribuições para reconfiguração parcial, remota e dinâmica de FPGAs**. Dissertação de Mestrado. Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002.

MONTEIRO, M. A. **Introdução à organização de computadores**. 5. ed. Rio de Janeiro: LTC, 2007.

MOUSSA, M.; AREIBI, S.; NICHOLS, K. **On the arithmetic precision for implementing back-propagation networks on FPGA: a case study**. In: OMONDI, A. R.; RAJAPAKSE, J. C. (Ed.). Springer, 2006.

NASCIMENTO Jr, C. L.; YONEYAMA, T. **Inteligência artificial em controle e automação**. São Paulo: Edgard Blücher, 2004.

NEDJAH, N.; MARTINS, R. S.; MOURELLE, L. M.; CARVALHO, M. V. S. (2009). *Dynamic MAC-based architecture of artificial neural networks suitable for hardware implementation on FPGAs*, Neurocomputing, Vol. 72, no. 10–12, pp. 2171–2179, Elsevier, Amsterdam.

NICHOLS, K. R. **A reconfigurable computing architecture for implementing artificial neural networks on fpga**. 2003. Master’s thesis, Faculty of Graduate Studies, University of Guelph, Dec 2003.

NORDSTROM T. **On-line localized learning systems part ii - parallel computer implementation**. Research Report TULEA 1995:02, Division of Computer Science and Engineering, Lulea University of Technology, Sweden, 1995.

OMONDI, A. R.; RAJAPAKSE, J. C. **FPGA implementations of neural networks**. Springer, 2006.

PANICKER, M.; BABU, C. **Efficient FPGA implementation of sigmoide and bipolar sigmoide activations functions for multilayer perceptrons**. IOSR Journal of Engineering (IOSRJEN). ISSN: 2250-3021 Volume 2, Issue 6, 1352-1356, Jun 2012.

PEDRONI, V. **Eletrônica digital moderna com VHDL**. Rio de Janeiro: Elsevier, 2010.

PEREZ-URIBE, A. **Structure-Adaptable Digital Neural Networks**. Ph.d. thesis 2052, Logic Systems Laboratory, Computer Science Department, Swiss Federal Institute of Technology-Lausanne, October 1999.

PRADO, R. N. A. et. al. **Arquitetura para aplicações genéricas de redes neurais artificiais com fácil configuração de topologias Multilayer Perceptron em FPGA**. 10th Brazilian Congress on Computational Intelligence (CBIC'2011). Fortaleza, Ceará, Brazil, 2011.

PRADO, R. N. A.; Soares, A. W. A.; Oliveira, J. A. N.; Dória, A. D. N.; Melo, J. D. "Desenvolvimento de uma nova proposta de arquitetura em hardware para aplicações genéricas utilizando redes neurais artificiais". (2010), XVIII Congresso Brasileiro de Automática, vol., no., pp., 12 a 16 set. 2010, Bonito-MS.

RIBEIRO, A. A. L. **Reconfigurabilidade dinâmica e remota de FPGAs**. São Carlos, SP, 2002.

SÁNCHEZ, D. F. **Implementação em VHDL de uma biblioteca parametrizável de operadores aritméticos em ponto flutuante para ser usada em problemas de robótica**. 2006. Dissertação de Mestrado em Sistemas Mecatrônicos, Publicação ENM.DM-30A/09, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 87p.

SANT'ANNA, R. E. **Uma metodologia para escalonamento de tarefas em tempo real em arquiteturas dinamicamente reconfiguráveis**. Recife, PE, 2006, 178 folhas.

SEIXAS, J. L. **Aquarius** – Uma plataforma para desenvolvimento de sistemas digitais dinamicamente reconfiguráveis – Recife, PE, 2007, 168 folhas.

SILVA, C. A. A. **Contribuição para o estudo do embarque de uma rede neural artificial em field programmable gate array (FPGA)**. Natal, RN, 2010.

SILVA, C. A. A. et al. Definição de uma Arquitetura para Configuração de Topologias de Redes Neurais Artificiais utilizando Reconfiguração Parcial em FPGA. Revista IEEE América Latina. v. 13, n° 7, julho 2015.

SKLIAROVA, I. ; FERRARI, A. B. Introdução à computação reconfigurável. **Revista do DETUA**, v. 2, n. 6, p. 1-16, set. 2003. Disponível em: < http://www.ieeta.pt/~iouliia/Papers/2003/1_SF_ETSet2003.pdf>. Acesso em: 15 dez. 2009.

SKRBK, M. **Fast neural network implementation**. Neural Network World, Vol. 9(No. 5):375–391, 1999.

STALLINGS, W. **Arquitetura e organização de computadores**: projeto para o desempenho. 5. ed. São Paulo: Prentice Hall, 2002.

TAFNER, M. A. **As Redes Neurais Artificiais: aprendizado e plasticidade**. Disponível em:<<http://www.cerebromente.org.br/n05/tecnologia/plasticidade2.html>>. Acesso em: 30 mar. 2010.

TAVENIKU, M.; LINDE, A. **A reconfigurable simd computer for artificial neural networks**. Licentiate Thesis No. 189L, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, 1995.

TOCCI, R. J. **Sistemas digitais: princípios e aplicações**. 10. ed. São Paulo: Person Prentice Hall, 2007.

WIIST, H., KASPER, K., REININGER, H. **Hybrid Number Representation for the FPGA-Realization of a Versatile Neuro- Processor**. Euromicro Conference, 1998. Proceedings 24th. Vol. 2, D.O.I 10.1109/EURMIC, vol. 2, pp. 694 -701, 1998.

XILINX. Silicon Devices. Disponível em:<<http://www.xilinx.com/products/devices.htm> >. Acesso em: 26 mar 2012.

ZHANG, M. Et al. *Sigmoid generators for neural computing using piecewise approximations* [Conferência] // IEEE Trans. Comput.. – 1996. – pp. 1045-1049.

ZHU, J.; SUTTON, P. **FPGA Implementations of Neural Networks – a Survey of a Decade of Progress**. In: CHEUNG, P. Y. K.; CONSTANTINIDES, G. A. (Ed.). Field programmable logic and application, Springer, 2003.