Departamento de Matemática e Engenharias

Dissertação de Mestrado
Engenharia de Telecomunicações e Redes

Implementação em *hardware* de uma Rede Neuronal com um microprocessador embebido em FPGA

Orientador: Professor Doutor Fernando Manuel Rosmaninho Morgado Ferrão Dias
Autora: Gisnara Rodrigues Hoelzle

Funchal – Setembro 2009

Agradecimentos

A DEUS

“Quando amamos e acreditamos do fundo de nossa alma, em algo, nos sentimos mais fortes que o mundo, e somos tomados de uma serenidade que vem da certeza de que nada poderá vencer nossa fé.”

Ao ORIENTADOR

“O que vai adiantar desistir?” Poucas… mas sábias palavras.

Obrigado pela paciência, serenidade e principalmente por ter acreditado em mim.

Aos PROFESSORES

“… O mestre que caminha à sombra do templo, rodeado de discípulos, não dá de sua sabedoria, mas sim de sua fé e de sua ternura. Se ele for verdadeiramente sábio, não vos convidará a entrar na mansão de seu saber, mas antes nos conduzirá ao limiar de vossa própria mente. Àqueles que dedicaram seu tempo e sua experiência para que minha formação fosse também um aprendizado de vida, meu carinho e homenagem.”

Ao meu FILHO

Desculpa pelas minhas ausências e pelo isolamento nas infindáveis horas de estudo; pelos momentos em que precisou de mim; pela atenção que não lhe foi dada devidamente e as alegrias e tristezas que não compartilhamos. Agradeço o sorriso e seus olhares que me deu a certeza para seguir em frente.

Ao meu MARIDO

Um estímulo para minha luta. Saiba que, junto de você, recebo mais do que dou e aprendo muito mais do que ensino. Tantas foram as vezes que meu cansaço e preocupações foram sentidos e compartilhados por você, numa união que me incentivava a prosseguir.

À minha FAMÍLIA

Como agradecer sabedoria, confiança e aprendizagem… mesmo de longe pude sentir toda a energia. Ofereço-lhes esta vitória!

Aos COLEGAS

“Há sempre alguém na vida da gente. Alguém que nos apareceu sorrindo, alguém que nos apareceu cantando. Há sempre alguém na vida da gente…Alguém que nos deixou chorando, alguém que nos deixou sofrendo. Há sempre alguém na vida da gente impossível de se esquecer.”

Resumo

O objectivo deste trabalho é a implementação em *hardware* de uma Rede Neuronal com um microprocessador embebido, podendo ser um recurso valioso em várias áreas científicas. A importância das implementações em *hardware* deve-se à flexibilidade, maior desempenho e baixo consumo de energia.

Para esta implementação foi utilizado o dispositivo FPGA *Virtex II Pro* XC2VP30 com um *MicroBlaze soft core*, da *Xilinx*. O *MicroBlaze* tem vantagens como a simplicidade no *design*, sua reutilização e fácil integração com outras tecnologias.

A primeira fase do trabalho consistiu num estudo sobre o FPGA, um sistema reconfigurável que possui características importantes como a capacidade de executar em paralelo tarefas complexas. Em seguida, desenvolveu-se o código de implementação de uma Rede Neuronal Artificial baseado numa linguagem de programação de alto nível. Na implementação da Rede Neuronal aplicou-se, na camada escondida, a função de activação tangente hiperbólica, que serve para fornecer a não linearidade à Rede Neuronal. A implementação é feita usando um tipo de Rede Neuronal que permite apenas ligações no sentido de saída, chamado Redes Neuronais sem realimentação (do Inglês *Feedforward Neural Networks* - FNN). Como as Redes Neuronais Artificiais são sistemas de processamento de informações, e as suas características são comuns às Redes Neuronais Biológicas, aplicaram-se testes na implementação em *hardware* e analisou-se a sua importância, a sua eficiência e o seu desempenho. E finalmente, diante dos resultados, fez-se uma análise de abordagem e metodologia adoptada e sua viabilidade.

Palavras-chave:

Rede Neuronal, FPGA, *MicroBlaze*, Implementação em *Hardware*

Abstract

The objective of this project is the hardware implementation of a neural network with an embedded microprocessor, which can be a valuable resource for many scientific areas. The importance of the implementation in hardware is due to the flexibility, higher performance and low power consumption.

To carry out this implementation, the device FPGA Virtex II Pro XC2VP30 was used with a microprocessor MicroBlaze soft core, from Xilinx. The MicroBlaze has advantages such as simplicity in the design, high reusability and easy integration with other technologies.

The first part of the work consisted of a study about the FPGA, a reconfigurable system that has important features like the ability to execute complex tasks in parallel. Next the code for the implementation of an Artificial Neural Network based on a high-level language was developed. In the Artificial Neural Network implementation the activation functions of the hidden layer on hyperbolic tangents which provide the network non-linearity. The implementation is done using a type of Neural Network that allows connections only to the exit, called Feedforward Neural Networks - FNN. As Artificial Neural Networks are information processing systems and their features are common to Biological Neural Networks, tests were applied to the hardware implementation and these tests analyzed the importance, the efficiency and performance. At the last stage, with the results obtained, the approach and methodology used were analized and checked for viability.

Key-words:

Neural Network, FPGA, MicroBlaze, Hardware Implementation

Índice

1. INTRODUÇÃO ... 1
1.1. ORIGEM DO TRABALHO ... 1
1.2. OBJECTIVO GERAL ... 2
1.3. OBJECTIVOS ESPECÍFICOS ... 2
1.4. O PROBLEMA A SER INVESTIGADO ... 2
1.5. JUSTIFICATIVA PARA O ESTUDO ... 2
1.6. ORGANIZAÇÃO DA DISSERTAÇÃO ... 3
2. REDES NEURONAIS ... 4
2.1. INTRODUÇÃO ... 4
2.2. CONCEITOS BÁSICOS ... 5
2.2.1. Neurónios Naturais ... 5
2.2.2. Neurónios Artificiais ... 6
2.3. REDES NEURONAIS ARTIFICIAIS ... 7
2.3.1. História ... 8
2.3.2. Aplicações ... 10
2.3.3. Tipos de Redes Neuronais Artificiais ... 11
2.3.4. Funções de Activação ... 12
2.3.5. Processos de Aprendizagem ... 13
3. IMPLEMENTAÇÃO DE REDES NEURONAIS EM HARDWARE ... 16
3.1. INTRODUÇÃO ... 16
3.2. TIPOS DE IMPLEMENTAÇÃO DE REDE NEURONAL ... 16
3.3. COMPARAÇÃO FPGA E ASIC ... 17
3.4. ESTADO DA ARTE ... 18
4. DESENVOLVIMENTO DA IMPLEMENTAÇÃO ... 22
4.1. ESPECIFICAÇÕES DO HARDWARE E DO MICROPROCESSADOR ... 22
4.1.1. Sistema Reconfigurável (Field Programmable Gate Array) ... 22
4.1.2. Microprocessador MicroBlaze ... 25
4.2. IMPLEMENTAÇÃO DA REDE NEURONAL DO PROJECTO ... 27
4.2.1. Ferramentas Utilizadas ... 28
4.2.2. Aspectos Gerais dos Módulos da Rede Neuronal ... 29
4.2.3. Implementação da Rede Neuronal ... 35
5. TESTES E RESULTADOS ALCANÇADOS ... 40
6. CONCLUSÃO E TRABALHOS FUTUROS ... 56
6.1. CONCLUSÃO ... 56
6.2. TRABALHOS FUTUROS ... 57

Lista de Figuras

Figura 2.1 - Rede Neuronal do Cérebro Humano ... 4
Figura 2.2 - Sinais na célula nervosa e seus principais elementos ... 6
Figura 2.3 - Neurónio natural ... 7
Figura 2.4 - Neurónio artificial (modelo simplificado) ... 7
Figura 2.5 - Neurónio *McCulloch-Pitts* ... 8
Figura 2.6 - Rede de *Perceptrons* proposta por *Rosenblatt* ... 9

Figura 2.7 - Tipos de Redes Neuronais Artificiais ……………………………..…………………………...…11
Figura 3.1 - Arquitectura genérica de um FPGA ....……………………………………………………...17
Figura 4.1 - Componentes da Plataforma FPGA ML310 ..………………………………………………23
Figura 4.2 - Cabo null modem:(a) Cabo utilizado (b) Ligações dos pinos ..…………………………….25
Figura 4.3 - Diagrama de blocos do *MicroBlaze* …………..……………………………………………26
Figura 4.4 - Fluxo da Ferramenta EDK……………………..……………………………………………29
Figura 4.5 - Diagrama de blocos da Rede Neuronal ……………………………………………………..30
Figura 4.6 - Janela *Base System Builder* ……………………………………………………...……….35
Figura 4.7 - *Linker Script Generator* …………….…………………….………………………………37
Figura 4.8 - Cabo JTAG ………………………..…………………………………………………...38
Figura 4.9 - Ligações dos sinais do JTAG …..…...………………………………………………………39
Figura 5.1- Vista esquemática do forno ………………………………………………………………40
Figura 5.2 - Modelo de Rede Neuronal For6700……………..…………………………………………..41
Figura 5.3 - Modelo de Rede Neuronal FORGA001……………..………………………………………41
Figura 5.4 - Saída do Modelo RN For6700 no *HyperTerminal* ….……………………………………………42
Figura 5.5 - Saída do Modelo RN FORGA001 no *HyperTermina l.*……………………………………..43
Figura 5.6 - Saída da Rede Neuronal For6700 (*Microsoft Office Excel 2007*) ….……………………….....44
Figura 5.7 - Saída da Rede Neuronal FORGA001 (*Microsoft Office Excel 2007*) . ……………………………..44
Figura 5.8 - Modelo For6700/ID050300 …. ……………………………………………………………….48
Figura 5.9 - Modelo For6700/ID13100A …………………………………………………………...……49
Figura 5.10 - Modelo For6700/DT030103 …. ……………………………………………………………..50
Figura 5.11 - Modelo FORGA001/ID050300 …………………………………………………………...51
Figura 5.12 - Modelo FORGA001/ID13100A …. .…………………………………………………………..52
Figura 5.13 - Modelo FORGA001/DT030103 ….. ……………………………………………………………53

Lista de Tabelas

Tabela 2.1 - Funções de Activação ………………………………………………………………………13
Tabela 4.1 - Características da Família *Virtex-II Pro* ……………………………………………………….24
Tabela 4.2 - Módulos que implementam a Rede Neuronal ……………………………………………………30
Tabela 5.1 – Modelo For6700/ID050300……………………………………………………………………48
Tabela 5.2 – Modelo For6700/ID13100A……………………………………………………………...49
Tabela 5.3 – Modelo For6700/ DT030103…………………………………………………………………..50
Tabela 5.4 – Modelo FORGA001/ID050300……………………………………………………………..51
Tabela 5.5 – Modelo FORGA001/ID13100A ……………………………………………………...…52
Tabela 5.6 – Modelo FORGA001/DT030103……………………………………………………………….53
Tabela 5.7 - Erro Médio Quadrático dos Modelos de RNs ……………………………………………...54
Tabela 5.8 - Tempo de Processamento dos Modelos de RNs …..………………………………………...54

CAPÍTULO 1

1. INTRODUÇÃO

"A mente que se abre a uma nova ideia jamais volta ao seu tamanho original." – Albert Einstein – cientista. (1879-1955)

Pretende-se, neste trabalho, implementar uma Rede Neuronal Artificial em *hardware*, utilizando um FPGA (*Field Programmable Gate Array*) com o microprocessador *MicroBlaze* embebido.

Autores como Lippmann [1] e Morgado Dias [2] constataram que as soluções com RNAs (Redes Neuronais Artificiais) alcançam melhores resultados na fase de implementação com um *hardware* específico do que a implementação mais comum usando um computador pessoal ou *workstation*. A existência destas soluções em *hardware* é de extraordinária importância para áreas como Ciências Aplicadas e Ciências da Saúde.

As Redes Neuronais Artificiais são sistemas distribuídos paralelamente, visto que têm a capacidade de receber ao mesmo tempo várias entradas e distribuir essas entradas de maneira organizada. Dessa forma, as informações armazenadas e compartilhadas em todas as unidades de processamento das RNAs melhoram o desempenho e a fiabilidade nos sistemas implementados [2].

1.1. Origem do Trabalho

Nos anos 80, coincidindo com a crescente evolução e disponibilidade de computadores, um grande interesse em tecnologias de Redes Neuronais fez-se notar, tendo sido possível, nesta época, desenvolver aplicações de reconhecimento de padrão visual e da fala, previsão financeira e muitas outras áreas.

No entanto, colocar em prática as técnicas de computação neuronal dedicada proporcionou soluções para áreas em que não seria viável a instalação de um PC (*Personal Computer*)/*workstation*, como a execução de um *software* de RN (Rede Neuronal) em *robots* autónomos para usos industriais e de exploração.

Neste trabalho, para realização da implementação da Rede Neuronal, escolheu-se o sistema reconfigurável FPGA com um microprocessador embebido. O FPGA é um elemento de lógica programável, configurado por um vector de *bits* denominado *bitstream* e carregado para a memória interna do componente.

Tendo em conta que o custo dos FPGAs está a reduzir, projectos de sistemas embebidos estão sendo apresentados como novas oportunidades para a criação acelerada de aplicações em *software* usando plataformas de *hardware* programáveis. Autores como *Pellerin* e *Thibault* [3] descrevem, a partir de uma perspectiva de *hardware*, estas novas plataformas que estão a preencher eficazmente a lacuna entre o *software* programável de sistemas baseados nos tradicionais microprocessadores e aplicações específicas em plataformas baseadas de *hardware* com funções personalizadas. E descrevem também, a partir de uma perspectiva de

software, os avanços das ferramentas de projecto e as metodologias para plataformas baseadas em FPGA que permitem a rápida criação de algoritmos de *hardware* acelerado.

1.2. Objectivo Geral

O principal objectivo desta dissertação é a implementação em *hardware* de uma Rede Neuronal Artificial com o microprocessador *MicroBlaze* embebido em FPGA.

1.3. Objectivos Específicos

A implementação da RNA (Rede Neuronal Artificial) deverá fornecer um sinal à saída do sistema, que activará os componentes acoplados, assemelhando-se ao que acontece na Rede Neuronal Biológica.

Levar-se-ão em consideração parâmetros importantes na implementação de uma RNA em *hardware* com o microprocessador embebido, como o número de entradas e saídas, o número de neurónios, o número de ligações a cada neurónio, o número de camadas, a representação numérica e precisão para os sinais envolvidos.

A implementação da RNA em *hardware* será um recurso para projectos nas áreas das Ciências Aplicadas como a Engenharia e Ciências da Saúde como a Biomedicina.

1.4. O problema a ser Investigado

Implementar a Rede Neuronal Artificial no FPGA com o microprocessador *MicroBlaze* embebido requer verificar as potencialidades do *hardware* e do *software* escolhidos e adquiridos para o trabalho.

A implementação deverá avaliar o grau de proximidade relativo ao resultado procurado em cada caso proposto, escolhendo-se de seguida um valor que converge para uma solução óptima.

Os neurónios da rede consistem de elementos base extremamente simples como somadores, multiplicadores e acumuladores e tornar-se-á necessário programá-los.

1.5. Justificativa para o Estudo

Desde os anos 40, biólogos projectam a implementação das actividades mentais através de programas computacionais. O modelo de mente mais aproximado com a realidade biológica é o da Rede Neuronal Artificial. As RNAs estão na base dos chamados computadores neuronais.

Tornou-se uma motivação para o desenvolvimento das máquinas independentes do controlo humano e que proporcionariam um funcionamento eficiente na resolução de tarefas, através de um processamento paralelo distribuído.

Na literatura científica observam-se vários projectos relacionados com as RNAs, como o FACETS (*Fast Analog Computing with Emergent Transient States*) que está a construir um computador neuronal capaz de emular o cérebro humano. Neste projecto, os pesquisadores conseguiram, até o momento, o que eles chamam de "cérebro num *chip*" [4].

1.6. Organização da Dissertação

Esta dissertação está organizada em seis capítulos e cinco anexos.

Na introdução, capítulo 1, é apresentada a origem do trabalho proposto, objectivos, o problema a ser investigado, justificativa para o estudo e a organização da dissertação.

Descrevem-se no capítulo 2 os fundamentos teóricos das Redes Neuronais, a sua história, a sua aplicabilidade, os tipos de Redes Neuronais Artificiais, as funções de activação e os processos de aprendizagem.

Faz-se no capítulo 3 uma abordagem das implementações de Redes Neuronais Artificiais, comparação entre FPGA e ASIC (*Application Specific Integrated Circuit*) e estado da arte.

Apresentam-se no capítulo 4 as especificações do *hardware* e do microprocessador embebido no FPGA, as ferramentas utilizadas, os aspectos dos módulos implementados e a implementação da Rede Neuronal.

Os testes do trabalho e os resultados alcançados são descritos no capítulo 5.

Finalmente, conclui-se no capítulo 6 o trabalho e apresentam-se trabalhos futuros, tendo em vista as inovações tecnológicas.

Nos anexos, encontram-se especificações do *hardware*, mencionadas neste trabalho, relatório de utilização dos recursos de *hardware*, o diagrama de blocos do sistema embebido, a sequência utilizada no trabalho das ferramentas incluídas no EDK (Embedded Development Kit) até o FPGA, pseudo-código e o código de implementação da Rede Neuronal Artificial.

2. REDES NEURONAIS

"If I have seen farther than others, it is because I have stood on the shoulders of giants." - Isaac Newton – (1642-1727)

As Redes Neuronais são modelos simplificados dos subsistemas que compõem o cérebro. Estas redes têm um enorme potencial de aprendizagem e são capazes de evoluir de uma forma semelhante à aprendizagem feita pelo ser humano: por tentativa e erro [2].

2.1. Introdução

A ideia de implementar a mente humana iniciou-se com uma tentativa de simular o cérebro e para isto foi necessário estudar as actividades dos neurónios [5]. Desenvolveram-se modelos matemáticos para descrever o comportamento complexo dos neurónios. Embora fáceis de serem modelados em detalhes, os neurónios podem ser complicados demais para se implementar.

Os neurónios são células básicas que transmitem impulsos eléctricos e estes são fundamentais para o funcionamento do sistema nervoso. Todas as funções e movimentos do organismo relacionam-se com o funcionamento dessas pequenas células.

Os neurónios ligam-se uns aos outros através de sinapses, e juntos formam uma grande rede, a que chamamos REDE NEURONAL. Rede essa que proporciona uma grande capacidade de processamento e armazenamento de informação.

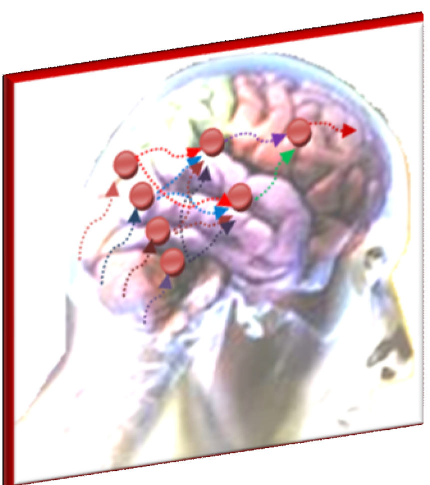


Figura 2.1 – Rede Neuronal no Cérebro Humano [6]

As Redes Neuronais possuem funções de processamento paralelo distribuído (PDP – *Parallel Distributed Processing*) [7] e estão dispostas na forma de um grafo direccionado, com restrições e definições próprias.

2.2. Conceitos Básicos

As Redes Neuronais Artificiais reproduzem o funcionamento lógico da Rede Neuronal Biológica, por isso são caracterizadas como modelos inspirados na estrutura neuronal de organismos inteligentes que adquirem conhecimento com a experiência [8]. No entanto, uma grande Rede Neuronal Artificial chega a possuir centenas ou milhares de unidades neuronais, enquanto que o cérebro de um ser humano alcança biliões de neurónios.

2.2.1. Neurónios Naturais

O sistema nervoso é composto por 86 biliões de células nervosas [9] ou neurónios. A informação é processada através de um evento conhecido como impulso nervoso. O impulso nervoso é a transmissão de um sinal codificado de um dado estímulo ao longo da membrana do neurónio, a partir do ponto em que ele foi estimulado.

Dois tipos de fenómenos estão envolvidos no processamento do impulso nervoso: os eléctricos e os químicos. Os processos eléctricos propagam um sinal dentro do neurónio e os processos químicos transmitem o sinal de um neurónio a outro [10].

Os impulsos recebidos pelos neurónios podem ser excitadores ou inibidores, ou seja, eles podem ajudar ou impedir a ocorrência de um impulso de saída. Nos neurónios naturais a excitação deve superar a inibição numa quantidade a que chamamos limiar do neurónio, para que seja gerado um impulso de saída [10]. Esta é uma condição de activação para as Redes Neuronais Biológicas que não transmitem sinais negativos e que têm a sua frequência medida pelos impulsos não contínuos e positivos.

Os principais elementos que constituem os neurónios são as dendrites, o corpo celular, também chamado de soma, e o axónio. Estes elementos estão ilustrados na figura 2.2.

As dendrites são um conjunto numeroso de prolongamentos especializados na função de receber os estímulos do meio ambiente, de células epiteliais sensoriais ou de outros neurónios [10]. O corpo celular representa o centro da célula e é responsável por receber e combinar as informações dos outros neurónios. O axónio é um prolongamento único, responsável por ligar-se a inúmeras outras entradas de diferentes neurónios e assim transmitir os estímulos para outras células.

A ligação entre um axónio de um neurónio e uma dendrite de outro é denominada sinapse. Desta forma, os axónios e dendrites não chegam a entrar em contacto directo. Existe uma substância química neurotransmissora que, em quantidade suficiente, permite que o impulso nervoso atravesse a separação. As ligações sinápticas são fundamentais na memorização da informação no cérebro humano [8].

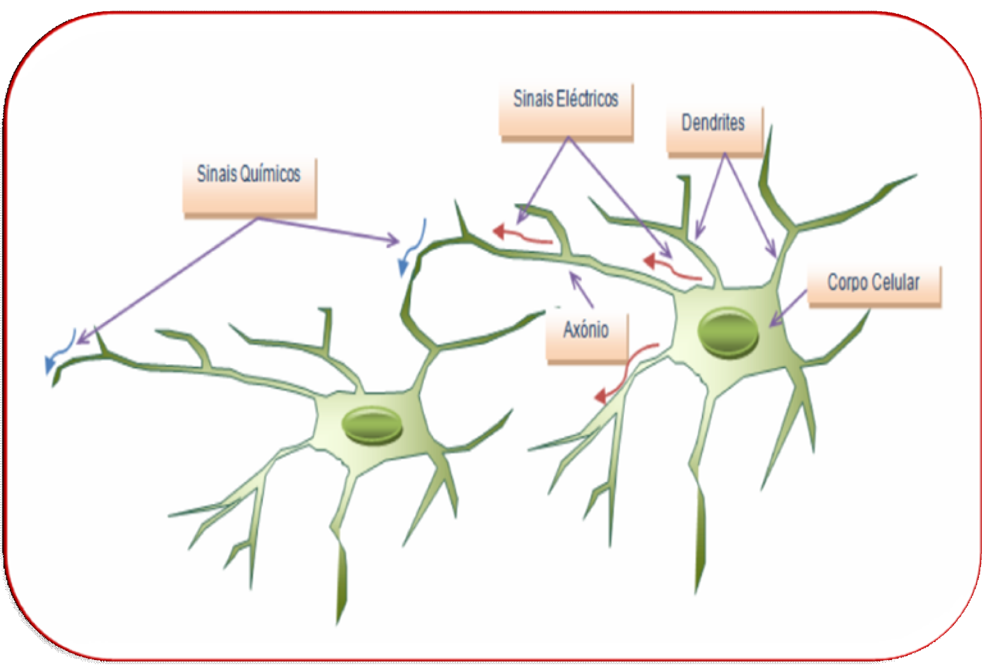


Figura 2.2 - Sinais na célula nervosa e seus principais elementos

2.2.2. Neurónios Artificiais

Os neurónios artificiais são implementados em módulos que simulam o funcionamento do sistema nervoso do ser humano. Estes módulos funcionam de acordo com os elementos em que foram inspirados, ou seja, exercem um esquema de entrada, processamento e saída de informação.

O neurónio artificial possui várias entradas podendo receber sinais de outros neurónios, sinais de fora da rede ou entradas fixas. A cada ligação de entrada no neurónio artificial tem-se um peso associado. Estes pesos modelam a característica dos neurónios naturais de possuírem ligações que propagam estímulos de maneira variável.

Os valores de entrada são multiplicados pelos pesos associados a cada entrada, e os resultados são aplicados a uma função de soma. O valor obtido é processado por uma função de activação, que define o valor da saída do neurónio.

As figuras 2.3 e 2.4 mostram um modelo de neurónio natural em comparação a um neurónio artificial [11].

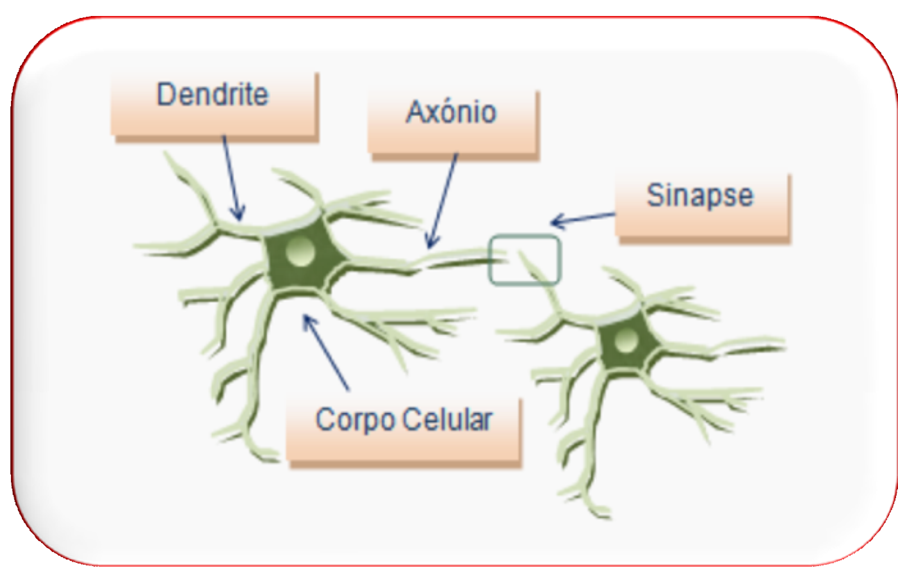


Figura 2.3 - Neurónio natural

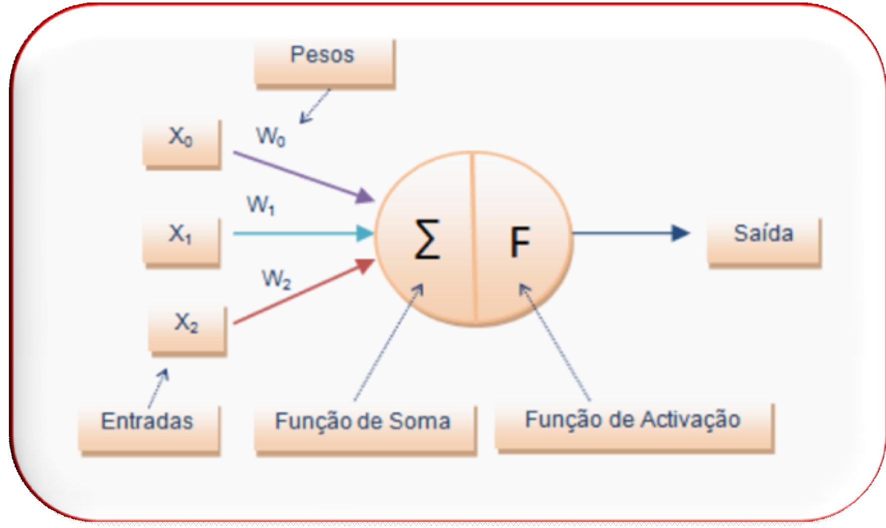


Figura 2.4 - Neurónio artificial (modelo simplificado)

2.3. Redes Neuronais Artificiais

As Redes Neuronais Artificiais consistem num método de solucionar problemas da inteligência artificial [7]. Estas redes constroem um sistema de circuitos num modelo inspirado no cérebro.

O sistema de neurónios artificiais com ligações sinápticas está disposto em camadas. As camadas são classificadas em três grupos: camada de entrada (*input*), camada escondida (*hidden*) e camada de saída (*output*).

A camada de entrada recebe os estímulos de fontes externas do sistema. A camada escondida é aquela onde é realizado o processamento do sistema construído, por meio de ligações ponderadas. A camada de saída, para além do processamento, envia sinais para fora do sistema, sinais que podem afectar componentes motores acoplados a esta.

Os neurónios da camada escondida são muito importantes numa Rede Neuronal Artificial, visto que sem a presença destes neurónios internos seria impossível a resolução de problemas linearmente não separáveis, como é o caso do OU Exclusivo (XOR).

2.3.1. História

O biólogo *Warren McCulloch* e o matemático *Walter Pitts* foram os primeiros a apresentarem um equivalente matemático para o funcionamento dos neurónios humanos, em 1943 [12] e [13].

A figura 2.5 mostra a representação do modelo matemático de *McCulloch* e *Pitts*, onde:

- $\mathbf{X_1 \ldots X_p}$ - Vector de valores de entrada;
- $\mathbf{W_1 \ldots W_p}$ - Vector de pesos associados às entradas;
- $\sum_1^p \mathbf{Xi * Wi}$ - Função de entrada, que multiplica o valor das entradas pelos respectivos pesos;
- **f(a)** - Função de activação, que evidenciará se o valor de entrada foi suficiente ou não para gerar algum dado na saída;
- **Y** - Função de saída, que irá conter o resultado das operações efectuadas no neurónio.

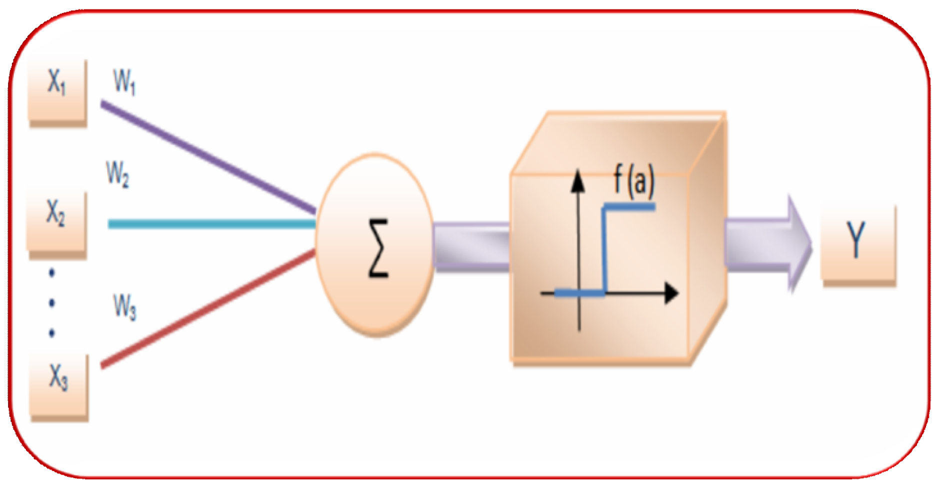


Figura 2.5 - Neurónio *McCulloch-Pitts*

O neurónio de *McCulloch-Pitts* é um dispositivo binário ou bipolar. A saída do neurónio representa um de dois estados possíveis, ou seja, 0 e 1 ou -1 e 1, e as suas entradas possuem ganhos arbitrários que podem ser excitadoras ou inibidoras [13].

Em 1947, *McCulloch* e *Pitts* exploraram a possibilidade de construir redes que efectuassem o reconhecimento de padrões visuais. Eles investigaram a habilidade de humanos e animais de reconhecer diferentes modos de apresentação de um mesmo objecto e como as múltiplas transformações de uma imagem (*input*) poderiam gerar uma representação canónica (*standard*) [5].

O neurofisiologista *Donald Hebb* publicou, em 1949, o livro "*The Organization of Behavior*" [14] que apresentava pela primeira vez a formulação de uma regra de aprendizagem específica para as sinapses dos neurónios [15]. *Hebb* notou que as ligações sinápticas do cérebro são continuamente modificadas à medida que um organismo aprende novas tarefas, criando assim agrupamentos neuronais.

Em 1957, após um longo período de investigação sem obter muitos resultados, o *Perceptron* foi criado por *Frank Rosenblatt*. O *Perceptron* é um tipo de Rede Neuronal Artificial com neurónios dispostos em camadas [5] e [16]. A figura 2.6 apresenta a Rede de *Perceptrons* proposta por *Rosemblatt*.

Rosenblatt escreveu, em 1958, o livro "*Principles of Neurodynamics*" [17]. Nesse livro descreveu o modelo dos *Perceptrons*, em que os neurónios eram organizados em camadas de entrada e saída, onde os pesos das ligações eram adaptados a fim de se atingir a eficiência sináptica, usada no reconhecimento de caracteres. Esta eficiência sináptica numa Rede Neuronal Biológica significa que o cérebro aprendeu com os estímulos fornecidos.

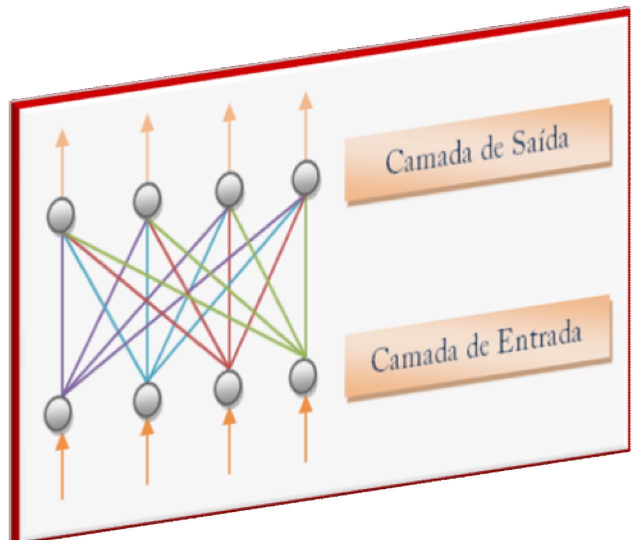


Figura 2.6 - Rede de *Perceptrons* proposta por *Rosenblatt*

Em 1960, *Widrow* e *Hoff* apresentaram o LMS (*Least Mean-Square*) [18], um algoritmo adaptativo para Redes Neuronais, também conhecido como Regra Delta e usaram-no para desenvolver um modelo neuronal linear muito simples, o *Adaline* (*Adaptive Linear Element*) [19].

Rosenblatt e *Widrow* estavam cientes das limitações do *Perceptron* de camada única e do *Adaline*, que se diferenciavam apenas pelo processo de aprendizagem. Apesar do entusiasmo, esses pesquisadores não conseguiram desenvolver algoritmos de aprendizagem para redes mais complexas.

O interesse em Redes Neuronais Artificiais diminuiu com a publicação do livro *Perceptrons*, por *Minsky* e *Papert*, em 1969. Por intermédio de uma análise matemática rigorosa, estes dois investigadores mostraram que havia certos problemas que o *Perceptron* não poderia resolver, como no caso da operação lógica OU Exclusivo (XOR) [5] e [19].

Um período de pesquisa silenciosa seguiu-se durante 1969 a 1982, quando poucos trabalhos foram publicados. Com o surgimento de recursos tecnológicos, nos anos 80, foram introduzidos novos conceitos sobre Redes Neuronais.

Um dos conceitos foi do físico *John Hopfield*, em 1982, que introduzia o uso da mecânica estatística, que esclarecia os comportamentos dos sistemas com elevado número de entidades constituintes, e com isto explicava as operações e convergência de algumas Redes Neuronais com Realimentação (*Feedback*).

O desenvolvimento do algoritmo de *Backpropagation*, em 1986, utilizado para treinar o *Perceptron* de múltiplas camadas, foi também um dos conceitos novos introduzidos. O algoritmo de *Backpropagation* foi descoberto por *Rumelhart*, *Hinton* e *Williams* [20]. Entretanto, a publicação mais influente foi o livro "*Parallel Distributed Processing*", editado por *David Rumelhart* e *James McClelland* [21].

A utilização da aprendizagem por *Backpropagation* de múltiplas camadas tornou-se o mais popular para o treino de *Perceptrons* devido a sua simplicidade computacional e eficiência [22] .

2.3.2. Aplicações

As Redes Neuronais Artificiais têm grande aplicabilidade em diversas áreas, como a medicina, o controlo, a economia, a agricultura e a meteorologia. Estas redes são consideradas como aproximadores universais, ou seja, podem aproximar qualquer função, e isto foi provado por investigadores como *Cybenko* [23], *Hornik, Stinchcombe* e *White* [24]. As RNAs possuem outras características significativas, como a generalização, a adaptabilidade e a uniformidade [19].

Algumas aplicações das RNAs estão abaixo descritas:

- Tecnologia de reconhecimento de voz;
- Tecnologia de reconhecimento de caracteres – OCR (*Optical Character Recognition*);
- Detecção de fraudes de cartões de crédito;
- Mercados financeiros;
- Controlo de processos industriais e automação;
- Robôs que desarmam bombas;

- Robôs que sentem empatia [25];
- Previsões climáticas;
- Diagnóstico médico;
- Análise de aroma e odor – Nariz electrónico.

2.3.3. Tipos de Redes Neuronais Artificiais

As Redes Neuronais Artificiais são classificadas de acordo com suas características e propriedades.

Do ponto de vista da topologia existem apenas dois tipos de RNAs: a Rede Neuronal sem realimentação ou *Feedforward Neural Network* e a Rede Neuronal com realimentação ou *Feedback Neural Network*. Na figura 2.7, apresentam-se alguns tipos de RNAs [19].

As RNAs sem realimentação são as redes constituídas habitualmente por elementos base iguais, os neurónios, que estão dispostos em camadas e ligados de forma a que o sinal flua da entrada para a saída sem realimentação e sem ligações laterais [2].

As RNAs com realimentação são as redes que incluem, pelo menos, uma ligação de uma camada mais próxima da saída para uma camada menos próxima da saída ou uma ligação entre neurónios da mesma camada [2].

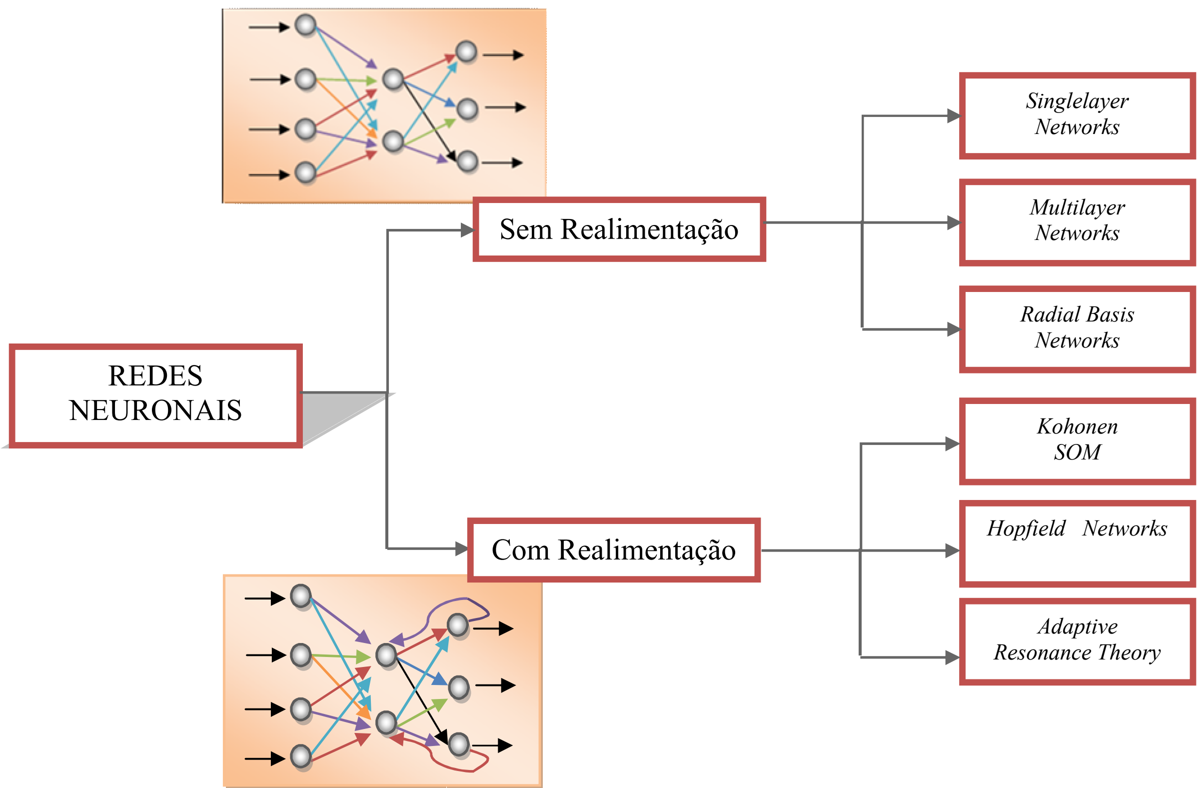


Figura 2.7 - Tipos de Redes Neuronais Artificiais

Redes Neuronais sem realimentação

A Rede Neuronal do tipo *Perceptron* (*Singlelayer*) foi o modelo pioneiro das Redes Neuronais Artificiais. Esta rede está limitada à classe de problemas linearmente separáveis, utiliza apenas a função de activação *Heaviside*, que origina uma saída do tipo binário [15].

O *Perceptron Multilayer* é uma extensão do simples *Perceptron*, capaz de trabalhar com problemas mais complexos, como os problemas não linearmente separáveis. Este avanço foi possível através da utilização de, no mínimo, uma camada entre a entrada e a saída da rede [15].

A Rede Neuronal *Radial Basis* utiliza o processo de aprendizagem supervisionado ou híbrido. Esta rede utiliza as funções de activação gaussiana e linear, podendo ser aplicada no controlo de processos em indústrias e previsão nos mercados financeiros [19].

Redes Neuronais com realimentação

A Rede Neuronal de *Kohonen*, também conhecida como auto-organizável, é inspirada nos mapas corticais. Nesta rede os neurónios competem entre si para responder a um estímulo apresentado. O processo de aprendizagem utilizado é não supervisionado [15].

A Rede Neuronal de *Hopfield,* também conhecida como Memória Associativa, armazena padrões que são recuperados a partir de estímulos de entrada. O armazenamento de tais padrões é realizado através do processo de aprendizagem não supervisionado [15].

A *Adaptive Resonance Theory* (ART) é um modelo de RN que realiza o processo de aprendizagem não supervisionado e foi desenvolvida por *Stephen Grossberg* e *Gail Carpenter* [26]. Esta rede permite que um exemplo de treino modifique um agrupamento somente se o grupo estiver suficientemente próximo do exemplo de treino, caso contrário um novo grupo é criado.

2.3.4. Funções de Activação

A função de activação tem como entrada o somatório dos estímulos recebidos pelo neurónio, ponderados pelos seus respectivos pesos sinápticos e estabelece o estado corrente de activação do neurónio [27].

Um neurónio genérico implementa a função:

$$\boldsymbol{Y} = \boldsymbol{F} \left(\sum_{(\boldsymbol{i},\boldsymbol{n})} \boldsymbol{W}_{\boldsymbol{in}} \boldsymbol{I}_{\boldsymbol{i}} \right) \tag{2.1}$$

onde I_i representa a i-ésima entrada e W_{in} o peso correspondente e F representa a função de activação que deve ser diferenciável ou não diferenciável apenas num conjunto finito de pontos [2].

As funções de activação frequentemente utilizadas na implementação de Rede Neuronais Artificiais estão apresentadas na tabela 2.1, onde y representa a saída e x representa a entrada.

Tabela 2.1 - Funções de Activação [2]

Nome	Função	Função
Heaviside	$x < 0 \longrightarrow y = 0$ $x \geq 0 \longrightarrow y = 1$	
Heaviside Simétrico	$x < 0 \longrightarrow y = -1$ $x \geq 0 \longrightarrow y = 1$	
Linear	$y = x$	
Linear com Saturação	$x < 0 \longrightarrow y = 0$ $0 < x < 1 \longrightarrow y = x$ $x > 1 \longrightarrow y = 1$	
Linear Simétrico com Saturação	$x < -1 \longrightarrow y = -1$ $-1 < x < 1 \longrightarrow y = x$ $x > 1 \longrightarrow y = 1$	
Log-Sigmoidal	$y = \frac{1}{1+e^{-x}}$	
Tangente Hiperbólica	$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	

2.3.5. Processos de Aprendizagem

As Redes Neuronais Artificiais possuem a capacidade de aprender através de exemplos. Este processo de aprendizagem, também designado por algoritmo de treino, é iterativo e traduz-se pelo ajuste dos pesos que compõem a RNA. Estes pesos sinápticos são modificados dependendo do algoritmo de aprendizagem.

De entre os principais tipos de procedimentos para a aprendizagem, pode-se citar [28]: aprendizagem supervisionada, aprendizagem semi-supervisionada e aprendizagem não supervisionada.

Aprendizagem supervisionada é aquela em que a Rede Neuronal Artificial recebe um conjunto de entradas padronizadas e seus correspondentes padrões de saída. Ajustam-se os pesos sinápticos até que o erro entre os padrões de saída gerados pela rede tenha um valor desejado.

Aprendizagem semi-supervisionada, também chamada de aprendizagem híbrida, é aquela em que a Rede Neuronal Artificial pode ter uma aprendizagem supervisionada e aprendizagem não supervisionada, ou seja, numa camada pode trabalhar com um tipo de aprendizagem e em outra camada trabalha com outro tipo de aprendizagem.

Aprendizagem não supervisionada é aquela em que não existe uma supervisão do "pesquisador". A própria rede encontra regularidades nos dados de treinos.

A subsecção seguinte ilustra o algoritmo de aprendizagem supervisionado mais conhecido: o *Backpropagation* [22].

Backpropagation

Backpropagation é o algoritmo mais utilizado para treino de Redes Neuronais multicamadas. Este algoritmo é baseado na aprendizagem supervisionada por correcção de erros, ou seja, a partir do erro na saída, calcula o erro referido a cada neurónio da Rede Neuronal.

O processo de aprendizagem por correcção de erros utiliza algoritmos no intuito de alcançar o menor valor de erro possível. Na descrição a seguir do algoritmo de *Backpropagation*, utiliza-se o i como o índice das unidades de entradas, sendo X_1 a unidade de entrada com índice 1. De forma análoga utiliza-se a notação Z_j para unidades escondidas e Y_k para unidades de saída [22] e [29].

O peso entre a unidade de entrada de índice i e a unidade escondida de índice j é representado por v_{ij}, e entre a unidade escondida j e a unidade de saída k será w_{jk}.

O algoritmo de *Backpropagation* é dividido em três fases: a propagação, o cálculo do erro e o de propagação inversa. Como se trata de um algoritmo de treino supervisionado, para cada vector de entrada $E = \{ E_1,E_2,...,E_i \}$ é informado uma saída esperada $S = \{ S_1,S_2,\ldots,S_k \}$.

Na fase de propagação inversa são feitos os seguintes passos:

1. Das unidades de entrada até as unidades de saída:

- Cada unidade de entrada recebe um sinal de entrada;
- O sinal de entrada é multiplicado pelo peso $v_{ij;}$
- O resultado da multiplicação passa por uma função de activação nas unidades escondidas;
- O resultado da função de activação é multiplicado pelos pesos w_{jk};
- Esse resultado passa pela função de activação na unidade de saída;
- O sinal de saída é o vector formado pelos sinais enviados pelas unidades de saída.

O sinal de saída obtido é comparado com o sinal de saída padrão para uma dada entrada, que a partir desta comparação são efectuados os ajustes dos pesos.

2. Das unidades de saída até as unidades de entrada:
   - O erro é calculado a partir do resultado esperado;
   - Este erro é utilizado no factor de ajuste dos pesos entre as unidades.

Na unidade de saída, o factor de ajuste dos pesos é calculado de acordo com a equação 2.2, em que y_in_k é a entrada da unidade e f a função de activação.

$$\delta_k = (S_k - y_k).f'(y_in_k) \tag{2.2}$$

Este factor só é utilizado na correcção dos pesos entre a unidade de saída e as unidades escondidas. Para a correcção dos pesos entre as unidades escondidas e as unidades de entrada é utilizada a equação 2.3. Nesta equação é utilizada a mesma notação da equação 2.2, com o acréscimo dos símbolos m, que é o número de unidades de saída e do símbolo z_in_j que é a entrada da unidade escondida j.

$$\delta_j = \left(\sum_{k=1}^{m} \delta_k . w_{jk} \right).f'(z_in_j) \tag{2.3}$$

Depois de calculado todos os factores de correcção, os pesos são actualizados seguindo as equações 2.4 e 2.5, onde α representa o factor de correcção e z_j representa a saída da unidade escondida j.

$$v_{ij} = v_{ij} + \alpha . \delta_j .E_i \tag{2.4}$$

$$w_{jk} = w_{jk} . \alpha . \delta_k .z_j \tag{2.5}$$

3. IMPLEMENTAÇÃO de REDES NEURONAIS em HARDWARE

"Nem tudo que se enfrenta pode ser modificado, mas nada pode ser modificado até que seja enfrentado." – Albert Einstein – cientista. (1879-1955)

3.1. Introdução

A implementação de Redes Neuronais Artificiais tem vindo a fazer parte de muitos projectos científicos como em receptores FH-CDMA (*Frequency Hopping – Code Division Multiple Access*) [30], em veículos autónomos [31], em análise de risco de crédito [32] e outros. São projectos específicos e dedicados à solução de problemas complexos que a princípio seriam insolúveis. Estas redes tornam-se vantajosas pela tolerância a falhas, uma vez que possuem informação distribuída entre os seus elementos, ou seja, mesmo em caso de falha pode-se recuperar parte do funcionamento visto que a informação está presente nos neurónios não afectados.

3.2. Tipos de Implementação de Rede Neuronal

Os tipos de implementação de uma Rede Neuronal são apresentados na literatura com diversas classificações. Uma classificação geral é apresentada utilizando as seguintes classes: neurocomputadores, placas aceleradoras, *chips*, elementos de biblioteca e microcomputadores embebidos [2].

As implementações em *hardware* são as mais exploradas pelos pesquisadores, pois estas alcançam melhores resultados em termos de velocidade de processamento, possuem maior fiabilidade e os custos são mais reduzidos. Nos artigos [33] e [34] são apresentados estudos de várias soluções de implementação em *hardware*, visando a categoria de Neuro-*chips* e concluem que não existe uma escolha ideal, mas sim uma solução mais adequada para cada caso.

Neste trabalho é proposto a implementação em um FPGA com o microprocessador embebido, sendo que esta escolha teve como relevância o preço do *hardware*. Surgindo nesta implementação, com o *MicroBlaze* embebido, a possibilidade de verificar a fácil utilização deste microprocessador e analisar o desempenho do processamento da rede.

Esta opção de implementação com o microprocessador embebido, tornou-se possível com a perspectiva positiva das novas ferramentas de projecto, que utilizam métodos específicos permitindo uma rápida criação de algoritmos de *hardware* e também com o nível de abstracção que obtemos quando utilizamos linguagem de programação de alto nível, como a linguagem C.

3.3. Comparação FPGA e ASIC

O FPGA (*Field Programmable Gate Array*) é um tipo de componente que possui um conjunto de recursos configuráveis, serve para implementar circuitos digitais, como processadores, *interfaces*, controladores e decodificadores. O ASIC (*Application Specific Integrated Circuit*) é um circuito integrado construído para executar uma tarefa específica.

A implementação de *chips* do tipo ASIC é um processo que requer alta tecnologia para o seu desenvolvimento, ou seja, só é possível realizar este processo em empresas especializadas. O custo destes *chips* depende directamente da quantidade a ser fabricada e normalmente o valor é alto, o que torna o FPGA a melhor opção para ser utilizado na fase de projecto e verificação do sistema. A tecnologia dos FPGAs vem maximizar o desempenho dos projectos, o que traz fiabilidade e baixo custo.

No FPGA, o engenheiro descreve o *hardware* a ser configurado e após compilação do sistema é criado um ficheiro de *bits*, denominado *bitstream*, que é utilizado para programá-lo. No ASIC, o engenheiro desenha todo o circuito que pretende, como por exemplo utilizando elementos de biblioteca e a programação só pode ser feita uma vez, na sua fase de projecto.

A estrutura geral de um FPGA, apresentada na figura 3.1, possui três tipos de recursos: os blocos de I/O (IOB – *Input Output Blocks*) que ligam os pinos de entrada e saída do dispositivo; os blocos lógicos (CLB – *Configurable Logic Blocks*) compostos por *flip-flops* e lógica combinatória e um conjunto de chaves de interligações organizadas como canais de roteamento horizontal e vertical entre as linhas e as colunas dos blocos lógicos [35].

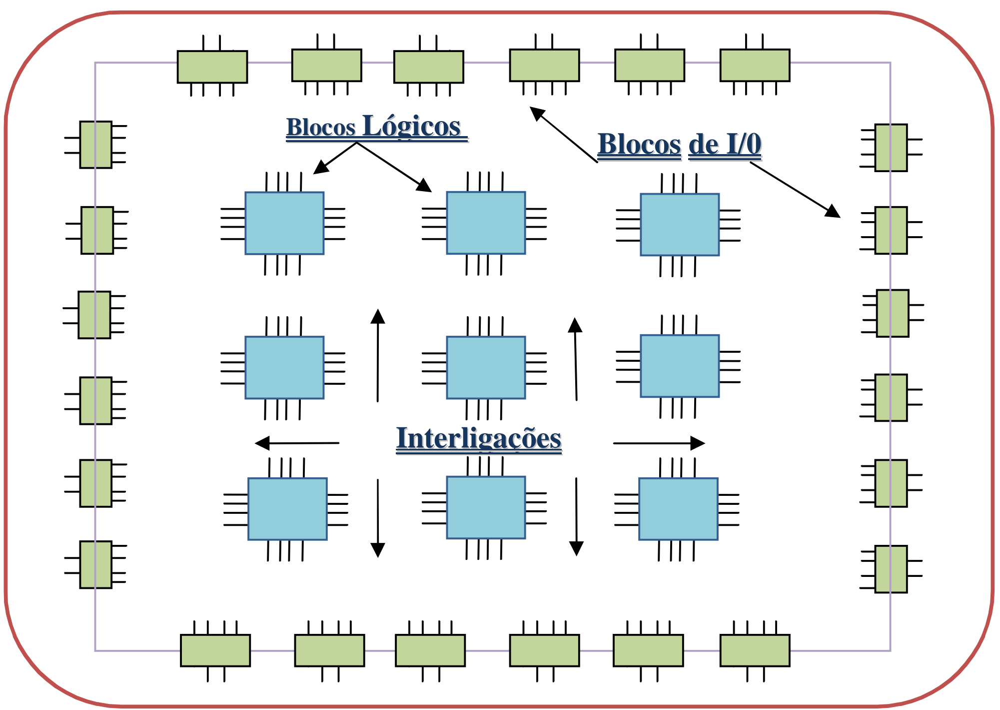


Figura 3.1 - Arquitectura genérica de um FPGA

Actualmente existem muitos fabricantes de dispositivos FPGA, como *Xilinx*, *Altera*, *Actel*, *Quicklogic*, *Atmel* e *Cypress*. Estas empresas estão sempre a fazer inovações tecnológicas, aumentando a capacidade lógica e a velocidade dos dispositivos.

3.4. Estado da Arte

Na literatura científica, existem muitos artigos e trabalhos importantes relacionados com a implementação de Redes Neuronais Artificiais em *hardware*. Alguns trabalhos se preocuparam no estudo da notação numérica em vírgula flutuante e na aplicação da função de activação. Alguns cientistas dedicaram implementações no intuito de adequar materiais para novos pesquisadores, capturando assim o estado da arte.

Numa implementação em *hardware*, a utilização da notação numérica em vírgula flutuante é muito importante, pois permite a utilização directa dos pesos aplicados à Rede Neuronal. Na notação em vírgula flutuante, a sua representação permitiu obter um maior intervalo de valores com o mesmo número de *bits* quando comparado, por exemplo, com a notação de vírgula fixa.

A função de activação tangente hiperbólica, embora simples de se implementar em *software*, é muito complicada em *hardware*. O que se verifica na maior parte das implementações, encontradas na literatura, o uso simplificado de funções de activação.

No artigo "*A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function*" [27] foi descrita a implementação de uma Rede Neuronal em FPGA, usando o tipo *Feedforward Neural Networks*, a notação escolhida foi a vírgula flutuante com 32 *bits* e a função de activação utilizada foi a tangente hiperbólica, tendo sido reduzida a apenas uma exponencial. Aplicou-se um algoritmo de aproximação linear que conseguiu uma alta precisão na função de activação. Os resultados obtidos nesta implementação foram satisfatórios, sendo considerada uma das melhores soluções para a função de activação com um erro máximo de 10^{-8}.

O desenvolvimento de uma arquitectura de *hardware* para uma Rede Neuronal em tempo real foi descrito no artigo "*Design of a pipelined hardware architecture for real-time neural network computations*" [36], que utilizou a notação de vírgula flutuante com 24 *bits* e tendo obtido bons resultados.

Em [37] encontra-se uma outra solução para a função de activação, em que se baseia numa série de *Taylor*, esta solução atingiu um erro máximo de 0.51%. No artigo, descreveu-se o desenvolvimento de sensores de *software*, em que a Rede Neuronal foi modulada com o uso da função de activação tangente hiperbólica e foi utilizada a notação de vírgula flutuante com 17 *bits*.

No artigo "*Feasibility of Floating-Point Arithmetic in FPGA Based Artificial Neural Networks*" [38], fez-se uma abordagem de uma implementação de RNAs em FPGA, utilizando uma precisão mínima permitida de 16 *bits*, com vírgula fixa, tendo em conta a precisão e a qualidade de desempenho da rede. No entanto, ocorreram erros de quantização nos cálculos com a limitação da vírgula fixa, como solução aplicou-se notação de vírgula flutuante para a execução do algoritmo *Backpropagation*.

Os primeiros resultados publicados, onde se avaliaram as soluções de *hardware* disponíveis no mercado para a implementação de Redes Neuronais, realizaram-se em 2003 [33] e em 2004 [34].

Nestas publicações de 2003, "*Commercial hardware for artificial neural networks: A survey*", e 2004, "*Artificial neural networks: A review of commercial hardware*", do ponto de vista do utilizador, a primeira etapa para escolher uma solução de hardware é uma especificação resumida do que é pretendido. Esta especificação inclui o tipo de RN (FNN, RBF – *Radial Basis Function*, *Kohonen*, etc.), o número dos neurónios, o número de entradas e de saídas, o número de ligações a cada neurónio, a precisão, a velocidade de operação ou o desempenho e outras características que podem ser mais ou mais menos importantes dependendo da aplicação. No artigo, foram tratados somente *hardware* da categoria de Neuro-*chips* com ênfase especial para as soluções comercialmente disponíveis e concluíram no estudo que não há uma escolha ideal, mas sim uma solução mais adequada para cada caso. Seguem abaixo alguns exemplos de soluções de *hardware* comercial analisados:

Implementações analógicas

ETANN - o *chip* da Intel, 80170NX, também designado por rede neuronal analógica treinável electricamente (do inglês *Electrically Trainable Analog Neural Network* - ETANN), foi o primeiro a estar disponível comercialmente. Armazena os pesos como carga eléctrica em portas flutuantes e usa multiplicadores de *Gilbert* de 4-quadrantes. Tem 64 neurónios e consegue um desempenho de 2 GCPS.
Synaptics Silicon Retina - este *chip* é um caso especial na implementação de *hardware* de RNs porque em vez de implementar uma arquitectura convencional tenta emular os neurónios biológicos da forma mais aproximada possível.

"Estas implementações analógicas têm a vantagem de obter velocidade elevada e alta densidade, ao mesmo tempo que apresentam inconvenientes como armazenamento dos pesos, estabilidade com variações da temperatura e obtenção de multiplicadores lineares sobre uma gama de operação alargada." [33] e [34]

Implementações digitais

MD-12201 - este *chip* da *Micro Devices* foi uma das primeiras implementações de *hardware* comercial para RNs. Contém oito neurónios com funções da activação de *Heaviside* e oito pesos de dezasseis *bits* com apenas um *bit* de entrada. Os multiplicadores usados são do tipo *bit-serial*.
NLX-4202 - cada *chip* contém dezasseis neurónios com precisão variável. Os dezasseis *bits* disponíveis para as entradas e os pesos podem ser seleccionados como dezasseis elementos de um *bit*, quatro elementos de quatro *bits*, dois elementos de oito *bits* ou um de dezasseis *bits*. As funções de activação são definidas pelo utilizador e está disponível realimentação interna.
Lneuro-1 - composto por dezasseis elementos de processamento, este *chip* tem também dezasseis *bits* de precisão variável para os pesos como o NLX-420. Os pesos devem ser guardados na memória *cache* existente no *chip* (1Kbyte) e as funções de activação são guardadas externamente.
Lneuro-2.3 - o Lneuro 2.3 é uma evolução do Lneuro 1. É constituído por 12 elementos de processamento que podem ser usados de forma paralela ou em modo série. Cada elemento de processamento contém 128 registos de 16 *bits* que podem ser usados para armazenar os pesos

ou os estados. Os elementos de processamento contêm também 16 a 32 *bits* de resolução no multiplicador, uma ALU de 32 *bits* e um shift-barrel. Pode ser usado microcódigo para adaptar o *chip* às aplicações.
Inova N640003 - este *chip* contém 64 elementos de processamento, cada um com um multiplicador de inteiros de 9 por 16 *bits*, um acumulador de 32 *bits* e 4 *Kbytes* de memória para o armazenamento dos pesos. O circuito inclui linhas de controlo e de dados comuns para cada processador o que torna mais fácil a combinação de diversos *chips*.
HNC100-NAP4 - o *Hecht-Nielsen Computers* 100 *Neurocomputer Array Processor* é um processador de vírgula flutuante de 32 *bits*. A escolha de uma representação de vírgula flutuante de 32 *bits* limita o número de elementos de processamento a 4. Os pesos são armazenados exteriormente.
Hitachi WSI - existem dois circuitos diferentes desenvolvidos pela *Hitachi* que têm algumas características comuns e algumas características distintas. Em comum têm a precisão de 9 por 8 *bits* e o tipo de arquitectura. A parte distinta diz respeito ao treino, ao número dos neurónios e à memória disponível para as sinapses.
RC Module NM6403 - este circuito é composto por processadores RISC (*Reduced Instruction-Set Computer*) de 32/64 *bits* e por um co-processador de 64-*bit* para suportar operações vectoriais de comprimento variável.
Siemens MA-16 - este *chip* executa operações rápidas de matrizes 4x4 com elementos de 16 *bits*. Tem 16 elementos de processamento complexos e os pesos e as funções da activação são armazenados exteriormente.
Nestor/Intel Ni 10005 - este circuito desenvolvido em conjunto pela *Nestor* e pela *Intel*, pode armazenar até 1024 protótipos com 256 dimensões e 5 *bits* por dimensão.
IBM ZISC036 - o IBM *Zero Instruction Set Computer* contém 36 neurónios com uma precisão de 8 *bits*. O *chip* tem treino integrado e inclui facilidades para ligar em série vários *chips*.
Silicon Recognition ZISC 78 - este circuito está preparado para fazer reconhecimento automático de padrões e é composto por 78 neurónios. Os vectores da entrada podem ter 1 a 64 componentes de 8 *bits*.

"As implementações digitais, em geral, têm as seguintes vantagens: armazenamento dos pesos em memória, facilidade de integração com outras aplicações, maior facilidade de implementar os algoritmos de treino e precisão dentro do número de *bits* dos operandos e dos acumuladores. De uma forma simplificada, quando comparadas com as soluções analógicas, as soluções digitais podem ser consideradas mais precisas. Por outro lado as implementações digitais são mais lentas (devido à necessidade de conversão de entradas analógicas e à operação dos multiplicadores digitais) e apresentam mais dificuldades para implementar as funções da activação." [33] e [34]

Implementações analógicas e digitais - híbridas

AT&T ANNA - AT&T *Artificial Neural Network Arithmetic Logic Unit* é um circuito digital do ponto de vista do utilizador mas é analógico no seu interior. Os pesos são armazenados como carga de condensadores que é refrescada periodicamente.
Este *chip* fornece um número variável de neurónios (16 a 256) e não tem nenhum algoritmo de treino incluído.
Bellcore CLNN -32 — este circuito apresenta semelhanças com o ANNA, uma vez que as entradas são também digitais e o processamento interno é analógico. Neste caso o treino com

o algoritmo de *Boltzmann* está disponível internamente. Implementa redes realimentadas com todas as ligações.

Mesa Research Neuroclassifier - este circuito, construído pelo instituto de pesquisa *Mesa da Universidade de Twente*, tem entradas analógicas e saídas com pesos digitais de 5 *bits*. A velocidade que é reivindicada atinge os 21 GCPS (do Inglês, *Connection Per Second*), que é a taxa de desempenho mais elevada encontrada nas soluções comerciais.

Ricoh RN-200 - este chip implementa um método diferente de emular RNs, baseado na taxa de impulsos e na largura dos impulsos. Esta é uma evolução do *Ricoh* RN-100, que tinha treino disponível com *Backpropagation* modificado. Um conjunto de 12 *Ricoh* RN-100 foi usado para aprender como balançar um pêndulo 2-D em apenas 30 segundos. Esta versão tem 16 neurónios cada um com 16 sinapses e consegue um desempenho de 3 GCPS.

"As implementações híbridas aparecem como um meio para tentar obter o melhor dos sistemas analógicos e dos sistemas digitais. Tipicamente as entradas/saídas são digitais para facilitar a integração em sistemas digitais, enquanto internamente algum ou todo o processamento é analógico." [33] e [34]

Em [39], [40] e [41] foram apresentadas soluções alternativas para o cálculo da função tangente hiperbólica utilizando o algoritmo CORDIC. Este algoritmo, que surgiu para converter coordenadas rectangulares em coordenados polares, pode ser utilizado para a implementação de funções trigonométricas. A utilização directa desse algoritmo não garantiu a convergência necessária, mas pôde ser usado com a restrição adequada dos ângulos.

As implementações de Redes Neuronais em *hardware* com o *MicroBlaze* embebido não são abundantes na literatura, existe um artigo intitulado como *Neural Network Processor for a FPGA* [42] que utilizou este microprocessador embebido. A rede foi implementada no FPGA *Spartan 3*, da *Xilinx*, e possui as seguintes especificações: oito entradas, oito neurónios na camada escondida e um neurónio na camada de saída. Utilizou-se a arquitectura *Perceptron Multilayer* e no treino da rede, externo ao FPGA, foi utilizado o algoritmo de *Backpropagation*. O resultado do tempo de processamento por entrada padrão foi de 1ms e a ocupação dos recursos do FPGA foi de 81% dos *slices* e 45% dos LUTs (*Look up Tables*) de quatro entradas e com uma frequência máxima do *MicroBlaze* de 174 MHz.

Para o projecto a desenvolver nesta dissertação, a implementação da Rede Neuronal será genérica, não se quantificando as especificações de entradas e de pesos, ou seja, deve poder trabalhar com qualquer estrutura e com qualquer número de entradas. Será desenvolvida no *Virtex-II Pro*, que possui uma capacidade maior do que o *Spartan* e um melhor desempenho. O treino da rede não faz parte do objectivo deste projecto.

4. DESENVOLVIMENTO da IMPLEMENTAÇÃO

"Somente seres humanos excepcionais e irrepreensíveis suscitam ideias generosas e acções elevadas." - Autor desconhecido.

A plataforma escolhida para a implementação da Rede Neuronal foi um FPGA com o *MicroBlaze* embebido, sendo uma opção relacionada com a flexibilidade que facilita o desenvolvimento do projecto, com o maior desempenho relativamente ao tempo de processamento da RNA a ser implementada e baixo consumo de energia dos FPGAs.

Neste capítulo, apresentam-se as especificações do FPGA, do *MicroBlaze* e descrevem os detalhes importantes no desenvolvimento da implementação, bem como as ferramentas utilizadas.

4.1. Especificações do Hardware e do Microprocessador

Utilizou-se neste projecto a plataforma ML310 com um FPGA *Xilinx Virtex-II Pro* XC2VP30 FF896 (*Flip-Chip Fine-Pitch* BGA *Package*) *speed grade* -6, da *Xilinx*. O *MicroBlaze* é o microprocessador embebido no FPGA. No anexo B, apresenta-se um diagrama de blocos do ML310 e de seus periféricos.

4.1.1. Sistema Reconfigurável (Field Programmable Gate Array)

A placa ML310, utilizada neste projecto, é composta por muitos componentes, tais como: *switches*, *jumpers*, 4 *slots* PCI (*Peripheral Component Interconnect*) para expansão de outros tipos de dispositivos de *hardware*, 256 MB de DDR DIMM (*Double Data Rate Dual Inline Memory Module*), 512 MB de memória *flash* e interfaces RS232 (*Recommended Standard-232*), USB (*Universal Serial Bus*), IDE (*Integrated Drive Electronics*), JTAG (*Join Test Action Group*) e rede 10/100 *Ethernet*. Na figura 4.1, são apresentados alguns componentes utilizados no desenvolvimento deste trabalho.

O *System ACE Reset Switch* (SW1) é um interruptor para o dispositivo *System ACE CF controller (Advanced Configuration Environment Compact Flash)*, e este configura o XC2VP30 no ML310, através da interface JTAG [43].

O *System ACE Configuration DIP Switch* (SW3) é o *switch* que controla os 3 pinos na configuração do endereço no *System ACE CF controller*. Encontram-se, no anexo B, os detalhes do SW3. Os endereços CFGADDR0, CFGADDR1 E CFGADDR2 são marcados no SW3 como posições 1, 2 e 3, respectivamente e por *default* SW3 = 000 [43].

O *Serial Port FPGA UART* (*Universal Asynchronous Receiver/Transmitter*) (J4) é uma das portas RS232 disponível no ML310, liga-se directamente ao XC2VP30 FPGA por meio de 10 pinos, conforme é apresentado no anexo B.

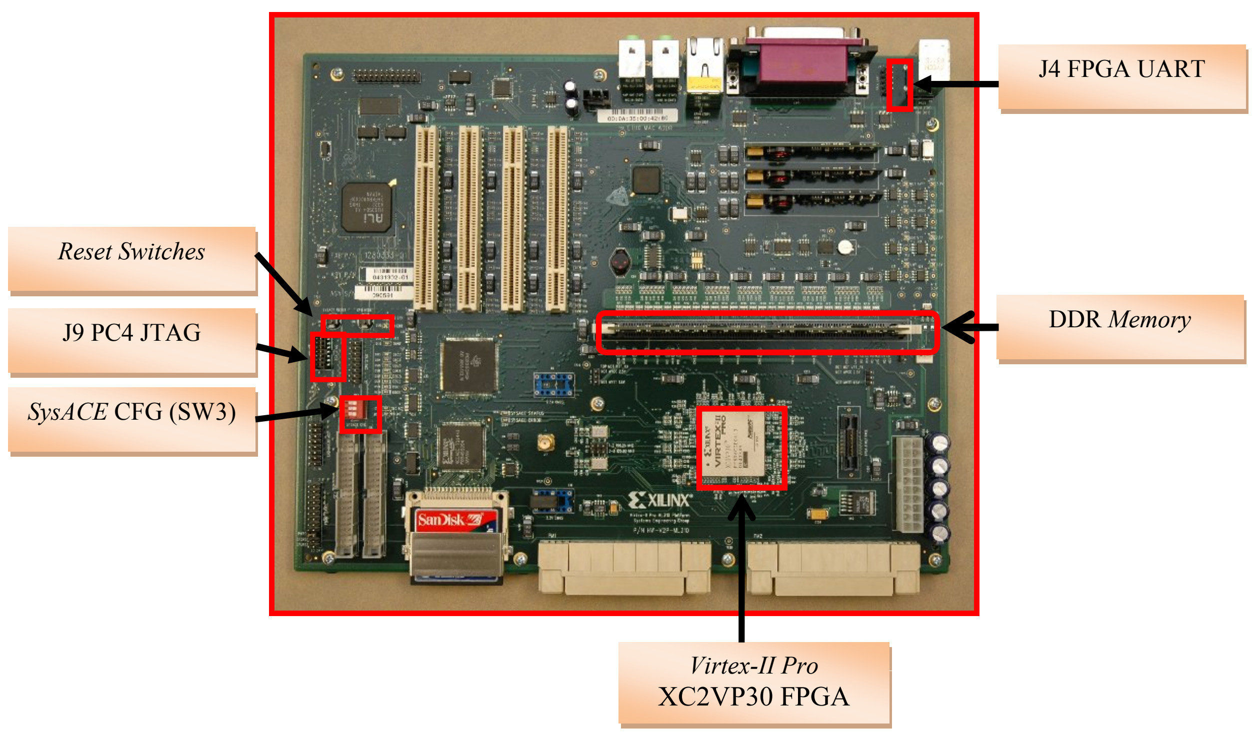


Figura 4.1 - Componentes da Plataforma FPGA ML310 [43]

A placa ML310 possui uma memória DDR-SDRAM (*Double Data Rate Synchronous Dynamic Random Access Memory*) com 256 MB.

O *Parallel Cable IV JTAG* (J9), também conhecido como ficha PC4 JTAG, possui instruções próprias definidas. Esta ficha ligada ao *hardware* permite testar, configurar e verificar a cadeia de componentes do seu circuito. Na placa ML310, esta ficha liga-se inicialmente ao *System ACE CF controller*, que passa as ligações JTAG até o XC2VP30 FPGA. Encontra-se, no anexo B, um diagrama das ligações entre a ficha JTAG, *System ACE CF controller* e o XC2VP30 FPGA [43].

A família *Virtex-II Pro* possui características importantes, o que pode ser observado na tabela 4.1, onde ocorre um aumento significativo nos números de células lógicas e de *slices*. A *slice* é a menor unidade lógica configurável nesta família.

A plataforma deste projecto, *Virtex-II Pro XC2VP30*, é considerada como uma solução tecnicamente sofisticada de silício, possuindo 30.816 células lógicas, 13.696 *slices* e é capaz de implementar sistemas com flexibilidade e custo reduzido [44]. Os dispositivos do *Virtex-II Pro XC2VP30* também implementam circuitos multiplicadores dedicados, os *Xtreme Multipliers*, implementam *RocketIO Transceivers* que suportam velocidades de transferência de dados até 3.125 Gbps, *PowerPC* 405, microprocessador com arquitectura do tipo RISC e blocos de RAM (*Random Access Memory*).

Tabela 4.1 - Características da Família *Virtex-II Pro* [44]

Device	XC2VP2	XC2VP7	XC2VP20	XC2VP30	XC2VP50	XC2VP100
Células Lógicas	3.168	11.088	20.880	30.816	53.136	99.216
PPC 405	0	1	2	2	2	2
RocketIO Transceiver	4	8	8	8	16	20
BRAM (KB)	216	792	1.584	2.448	4.176	7.992
Xtreme Multiplers	12	44	88	136	232	444
Slices	1.408	4.928	9.280	13.696	23.616	44.096

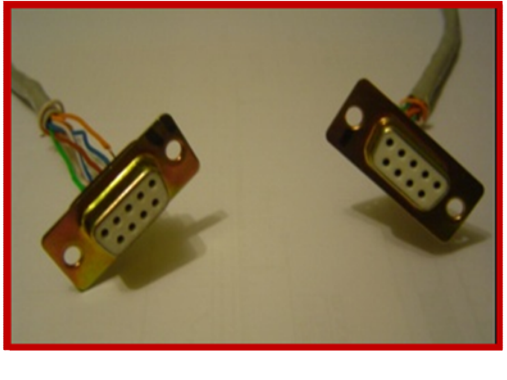
Os primeiros testes, deste trabalho, realizaram-se no intuito de diagnosticar o funcionamento da placa ML310. Estes testes foram verificados através do *HyperTerminal*, que é um programa de comunicações do Sistema Operativo *Windows*.

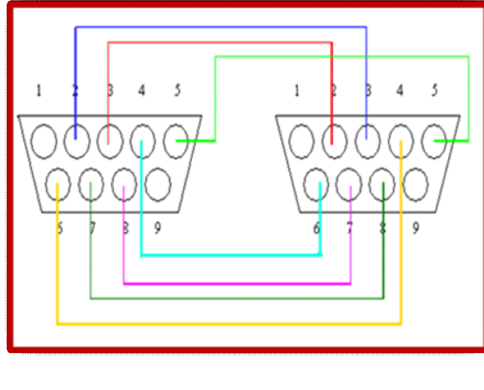
Iniciou-se a comunicação ligando a placa com um cabo *null modem serial* e verificou-se a existência de 8 (oito) configurações possíveis, como: ACE (*Advanced Configuration Environment*) – *loader*; *MontaVista Linux* 3.1; *VxWorks* 5.5; QNX (*Queue Nicks Unix*) *Demo; Linux* EDK (*Embedded Development Kit*) *Base Build; VxWorks* EDK *Base Build e User Configuration* A e *User Configuration* B.

A configuração ACE - *loader* identifica a comunicação da porta série com o cabo *null modem serial*, as configurações *Linux* 3.1, *VxWorks* 5.5 e QNX são sistemas operativos opcionais para se trabalhar com a placa e as *Users Configurations* A e B são opções usadas para o caso do utilizador ter outras configurações já definidas.

Realizaram-se testes simples de diagnóstico, dentre os quais o modo *MontaVista Linux 3.1*, onde se verificaram as comunicações de entradas e saídas com periféricos e através de comandos como *ls* e *ipconfig*, que observaram o sistema de ficheiros e configurações de rede, respectivamente.

As figuras 4.2.a e 4.2.b apresentam o cabo *null modem serial*, construído e utilizado neste projecto, e as ligações dos pinos.





(a) (b)

Figura 4.2 - Cabo *null modem*: (a) Cabo utilizado (b) Ligações dos pinos [45]

Descrevem-se, abaixo, as ligações dos pinos [46] no cabo *null modem serial*:

- pino 2 (RD – *Receive Data*) liga ao pino 3 (TD – *Transmit Data*) – *Receive-Transmit*
- pino 4 (DTR – *Data Terminal Ready*) liga ao pino 6 (DSR – *Data Set Ready*)
- pino 5 (GND - *Ground*) liga ao pino 5 (GND)
- pino 7 (RTS – *Request To Send*) liga ao pino 8 (CTS – *Clear to Send*)

Utilizaram-se os parâmetros 9600-8-N-1 e sem *flow control*, no *HyperTerminal*, ou seja, a comunicação foi feita com a velocidade de 9600 bps, usando palavras de 8 *bits*, sem paridade, um *bit stop* e sem controlo de fluxo.

4.1.2. Microprocessador MicroBlaze

O *MicroBlaze* é um microprocessador com arquitectura do tipo RISC de 32 *bits.* Este microprocessador é um *soft core*, ou seja, possui um conjunto de recursos configurados através de uma linguagem de descrição de *hardware* (HDL-*Hardware Description Language*). O *MicroBlaze* está optimizado para implementações em FPGAs, tendo em vista a simplicidade do *design*, sua reutilização, fácil integração com outras tecnologias e o baixo risco de se tornar obsoleto [47].

O número de *MicroBlazes* que pode ser embebido depende exclusivamente da capacidade do FPGA.

A ferramenta EDK, comercializada pela *Xilinx*, é o ambiente de desenvolvimento do *MicroBlaze*. O EDK possui uma plataforma chamada XPS (*Xilinx Platform Studio*) que permite a criação da arquitectura de *hardware* e a partir desta plataforma, tem-se o SDK (*Software Development Kit*), um *software* baseado no *Eclipse open source standard* que utiliza o compilador GCC (*GNU Compiler Collection*) para compilar o código de programação em linguagem C implementado no *MicroBlaze* [47].

O *MicroBlaze* tem um *design* básico, podendo ser configurado com características dependendo da necessidade do projecto a ser implementado. Esta flexibilidade permite que o utilizador avalie o compromisso entre o desempenho pretendido e a área a ser utilizada no FPGA pelo microprocessador.

Na figura 4.3, apresenta-se um diagrama de blocos do *MicroBlaze*. Os blocos em branco são as funções básicas para o funcionamento do sistema do *MicroBlaze* e os blocos preenchidos na cor laranja são opcionais.

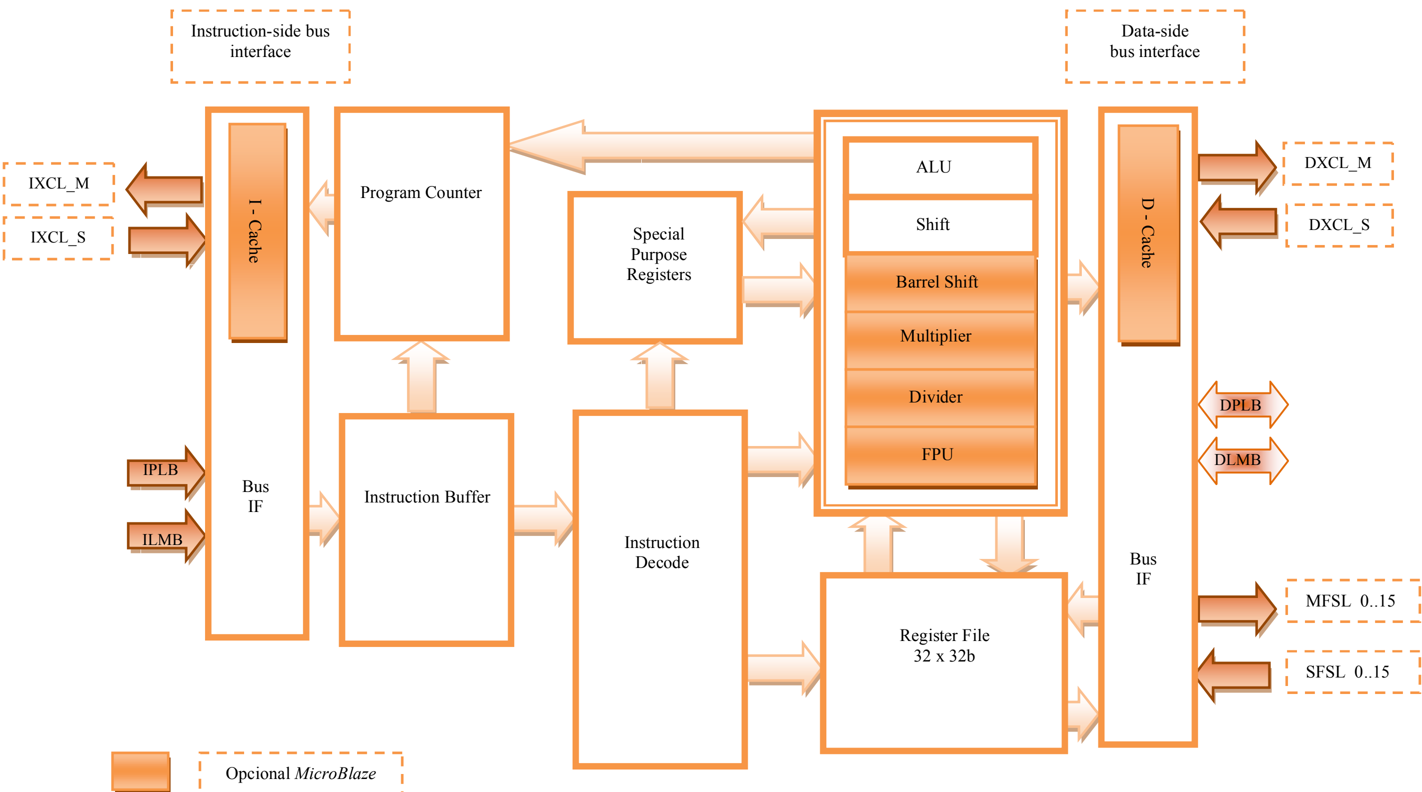


Figura 4.3 - Diagrama de blocos do *MicroBlaze* [47]

As características principais do *MicroBlaze* são [47]:

- Interfaces para os barramentos LMB (*Local Memory Bus*), PLB (*Processor Local Bus*) e FSL (*Fast Simple Link*). O LMB é um barramento síncrono, usado principalmente para aceder memória *on-chip*, o PLB é um barramento compatível com o LMB com acesso à memória *off-chip* e faz a ligação dos periféricos, o FSL é um tipo de protocolo de comunicação de alta velocidade.
- Unidade aritmética e lógica (ALU – *Arithmetic Logic Unit*) onde o processador executa as funções aritméticas e lógicas básicas, nomeadamente as operações

aritméticas com inteiros (adição e subtracção), as operações binárias (*and*, *not*, *or* e *xor*) e as operações de deslocamento de *bit*, que são apresentadas na figura 4.3 como ALU e *Shift*, respectivamente.

- Banco de 32 registos de 32 *bits* para o uso geral.

As características que podem ser acrescentadas no *MicroBlaze* são [47]:

- *Cache* de instruções e *cache* de dados que são memórias de acesso rápido, interno do microprocessador, que permitem evitar o acesso ao dispositivo de armazenamento, que pode ser demorado.
- *Multiplier* e *Divider* permitem executar multiplicações e divisões em menos ciclos de relógio do que utilizando apenas *software*.
- FPU (*Floating Point Unit*), compatível com IEEE-754, permite executar as funções com vírgula flutuante.
- *Barrel Shift* um circuito de lógica combinatória que permite fazer qualquer deslocamento de *bits*, num ciclo de relógio.

4.2. Implementação da Rede Neuronal do Projecto

O tipo de Rede Neuronal utilizada neste projecto foi *Feedforward Neural Network*, que permite apenas ligações no sentido de saída, ou seja, não existe reavaliação recursiva no processamento da rede. Esta opção resulta da sua maior divulgação, sendo utilizada em situações em que se introduzem todos os dados para a solução do problema, na camada de entrada, e que permitem implementar modelos de forma rápida [2].

O código programado para a Rede Neuronal foi desenvolvido no intuito de ser genérico, ou seja, a rede deveria receber uma quantidade variável de entradas e de neurónios. O número de neurónios da camada escondida permitiria o ajuste do tamanho da rede à complexidade do sistema a ser usado [2].

No decorrer do projecto, foram efectuadas alterações nas dimensões da rede implementada. Seguem-se algumas designações de rede testadas: 3-8-1; 4-10-1; 5-3-2; 5-8-3 e 5-20-1. Estas designações quantificam o número de entradas, de neurónios na camada escondida e o neurónio na camada de saída, ou seja, para uma rede 3-8-1, diz-se que possui 3 entradas, 8 neurónios na camada escondida e 1 neurónio na camada de saída.

Para os cálculos internos da função de activação, utilizou-se a função tangente hiperbólica, expressão 4.1, uma vez que esta função forneceria a não linearidade à Rede Neuronal. No entanto, a fim de se tornar mais rápido o processamento da rede no *hardware*, utilizou-se a função numa forma mais reduzida, ou seja, implementou-se a expressão 4.2, que possui apenas uma exponencial e^x. Na camada de saída, utilizou-se a função linear, expressão 4.3. Nas expressões abaixo, $\boldsymbol{y}$ representa a saída e $\boldsymbol{x}$ representa a entrada.

$$Y(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad (4.1)$$

$$Y(x) = 1 - \frac{2}{1 + e^{(2x)}} \tag{4.2}$$

$$Y(x) = x \tag{4.3}$$

Para definir a notação numérica na qual os cálculos da rede seriam baseados, optou-se pela vírgula flutuante com precisão simples, visto que esta permitia a representação de números muito grandes e muito pequenos, para o mesmo número de *bits*. Outra opção seria utilizar a vírgula fixa, mas teria a desvantagem de aumentar consideravelmente o número de *bits* para a mesma gama de valores da notação em vírgula flutuante. E para que as operações fossem executadas, acrescentou-se na configuração do sistema embebido do *MicroBlaze* o FPU (*Floating Point Unit*), já referido anteriormente como uma característica que poderia ser acrescentada neste microprocessador e apresentado na figura 4.3.

4.2.1. Ferramentas Utilizadas

Para o desenvolvimento deste projecto foram utilizadas as ferramentas EDK, Matlab R2007b e *Microsoft Office Excel* 2007, desenvolvidas pela *Xilinx*, *MathWorks* e *Microsoft*, respectivamente.

O projecto foi desenvolvimento na versão 10.1 do XPS, incluído no EDK da *Xilinx*, que apresentava as condições necessárias para as implementações em *hardware*.

Na figura 4.4, apresentam-se detalhes do fluxo da ferramenta EDK e no anexo D, apresenta-se a sequência utilizada no trabalho das ferramentas incluídas no EDK até o FPGA.

A etapa inicial no XPS consistiu em criar a arquitectura de *hardware*, gerando o *Netlist*, onde foi criado uma descrição textual do circuito para as ferramentas de verificação e de implementação do sistema.

Uma vez especificado e verificado o *Netlist*, seguiu-se para a etapa da implementação, onde foi necessário o mapeamento dos componentes para as células lógicas. O processo de mapeamento (*map*) foi seguido pelo posicionamento, roteamento (*place and route*) e após compilação, obteve-se um ficheiro binário (*bitstream*) guardado na memória e que corresponde ao número 1, na figura 4.4. Nesta fase, existe a possibilidade de fazer o *download* directamente para o dispositivo através da interface JTAG.

A etapa seguinte foi dedicada à implementação da Rede Neuronal Artificial desenvolvida na linguagem de programação C. Definiram-se módulos que representavam as entradas, os pesos das ligações, os pesos sinápticos, os neurónios e o funcionamento da rede neuronal. Estes módulos foram programados de uma forma genérica, uma vez que tinha sido proposto testar várias dimensões de rede. Configuraram-se as memórias necessárias e após compilação obteve-se o ficheiro com extensão *elf*, que corresponde ao número 2, na figura 4.4.

Os procedimentos que se seguiram consistiram basicamente em simulações, programação do FPGA e no *debug*, podendo ser observado no número 3, da figura 4.4.

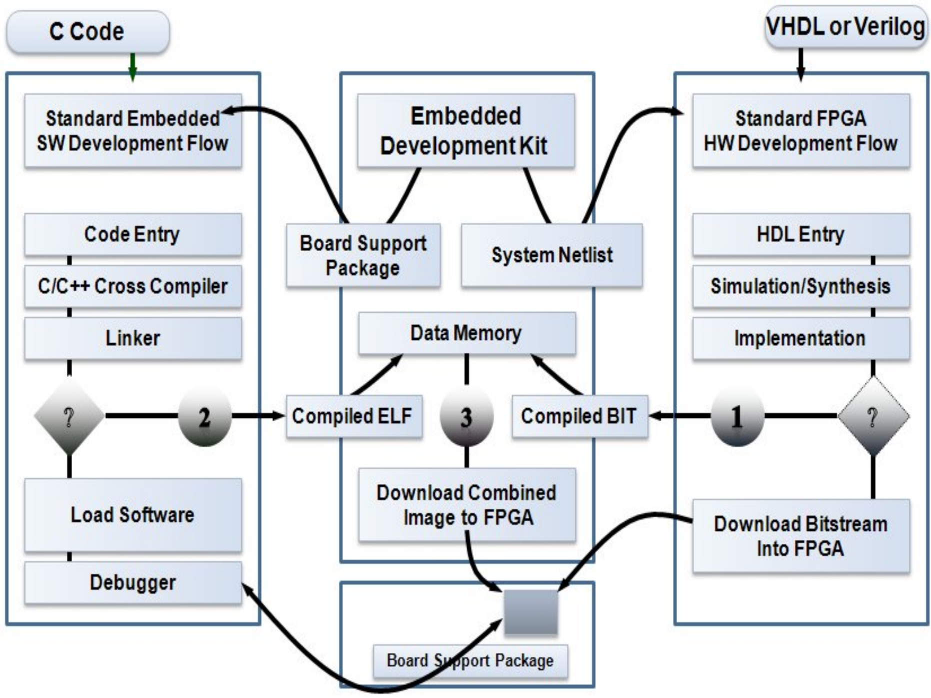


Figura 4.4 - Fluxo da Ferramenta EDK [48]

A ferramenta *Microsoft Office Excel*, da *Microsoft*, foi utilizada para verificar o valor da saída da rede implementada. Foram feitos cálculos matemáticos baseados na expressão 4.4 (secção 4.2.2), que calculava a saída da Rede Neuronal Artificial. As figuras 5.5 e 5.6, do capítulo 5 - Testes e Resultados Alcançados, apresentam os resultados obtidos.

A ferramenta Matlab, da *MathWorks* foi utilizada para comparar a implementação da Rede Neuronal Artificial no FPGA e a implementação da rede no Matlab, que utilizava redes implementadas com uma *toolbox*. Verificou-se também o tempo de processamento destas implementações. Foram utilizados ficheiros de um sistema real de um forno eléctrico, criados num projecto de investigação [2]. No capítulo 5 são descritos os testes e resultados alcançados.

4.2.2. Aspectos Gerais dos Módulos da Rede Neuronal

Neste projecto de *hardware* para RNAs, definiu-se que cada um dos módulos funcionaria de acordo com os elementos que compõem uma rede, ou seja, num esquema de entrada, processamento e saída de informação.

A figura 4.5 apresenta um diagrama de blocos da Rede Neuronal programada em linguagem C e a tabela 4.2 identifica e descreve a funcionalidade dos módulos que implementam a rede.

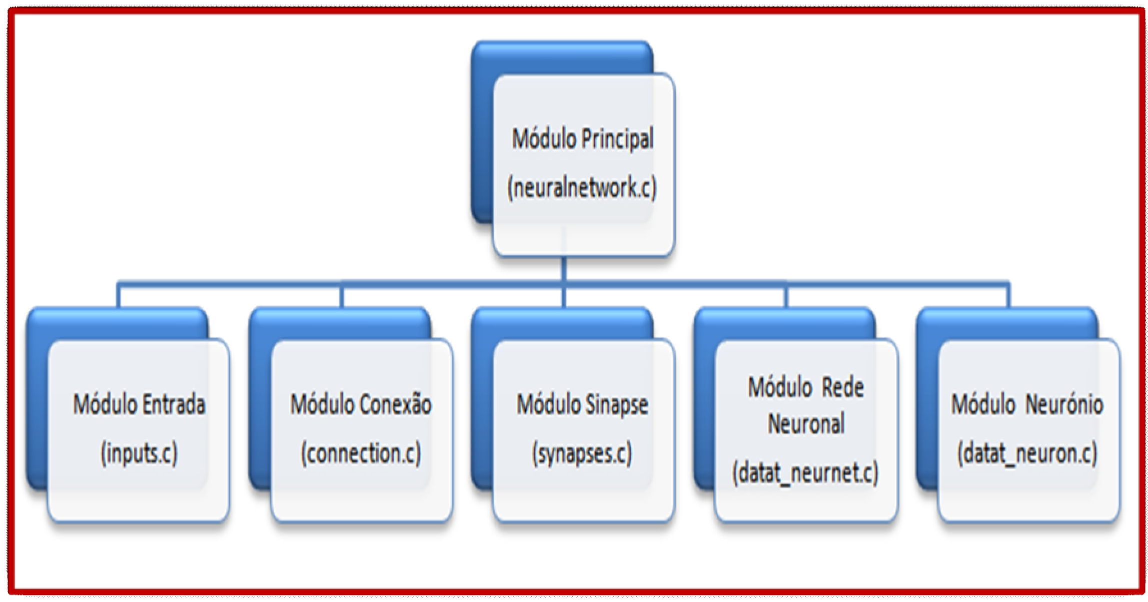


Figura 4.5 - Diagrama de blocos da Rede Neuronal

Tabela 4.2 – Módulos que implementam a Rede Neuronal

Módulo	Descrição
Principal	Programa principal da implementação da Rede Neuronal, contém a função *main ( )*.
Entrada	Módulo que possui o tipo de dados das entradas e suas operações, ou seja, cria-se uma lista ligada que vai recebendo as entradas da rede.
Conexão	Módulo que possui o tipo de dados das ligações da rede e suas operações, ou seja, cria-se uma lista ligada que vai recebendo os pesos das ligações.
Sinapse	Módulo que possui o tipo de dados das sinapses presentes na rede e suas operações, ou seja, cria-se uma lista ligada que vai recebendo os pesos das sinapses.
Neurónio	Módulo que possui o tipo de dados dos neurónios e suas operações, ou seja, cria-se uma lista ligada que vai recebendo os neurónios da rede.
Rede Neuronal	Módulo que possui o tipo de dados da RNA e operações necessárias para o funcionamento da rede, ou seja, neste módulo o utilizador estabelece o tamanho da rede e altera os dados das entradas e dos pesos.

Descreve-se abaixo o pseudo-código do processamento da Rede Neuronal Artificial.

/*
O programa deverá fazer com que a RN funcione, que faça o papel dos neurónios, ou seja, receba dados de entrada, processa os dados e envia um sinal ou estímulo para fora do sistema construído.
***/**

1 - Início de programa
//O programa inicia no **Módulo Principal** (neuralnetwork.c).

2 – VAR inp, neur, outp : **INTEIRO**
entr, weight_con, weight_snp, weight_snpbias : **REAL**

/*

Identificar os dados necessários para o processamento da Rede Neuronal, ou seja, as variáveis:

- inp representa o nº de entradas,
- neur representa o nº de neurónios,
- outp representa o nº de saídas,
- entr representa os valores das entradas,
- weight_con representa os valores dos pesos das ligações,
- weight_snp representa os valores dos pesos das sinapses e
- weight_snpbias representa o valor do peso do Bias.

*/

3 – Criar as listas ligadas

//No **Módulo Entrada** (inputs.c) é criada a lista ligada das entradas da rede.

Função InsertInputList ( );
inicio
/*

Esta função vai inserir o nó no fim da lista, porque é preciso ter as entradas na ordem em que são criadas. Os nós possuem três campos:

- nº de ordem da entrada, do tipo **inteiro,**
- valor da entrada, do tipo **float** e
- **ponteiro** para o próximo nó desta lista.

*/

SE (lista de entrada está vazia) **ENTÃO**
lista_entrada.= novo no;
SENÃO
lista_entrada.= insere fim;
FIM-SE
fim

//No **Módulo Conexão** (connection.c) é criada a lista ligada dos pesos das ligaçoes da rede.

Função connect_InsertList ( );

inicio
/*
Esta função vai inserir o nó no início da lista, porque é mais eficiente. Os pesos das ligações estão ligados à entrada associada. Os nós possuem cinco campos:
- valor do peso da ligação, do tipo **float**,
- nº de ordem da entrada associada, do tipo **inteiro**,
- nº de ordem do neurónio posterior, do tipo **inteiro**,
- nº que identifica se o neurónio posterior é da camada escondida, do tipo **inteiro** e
- **ponteiro** para o próximo nó desta lista.

*/
SE (lista de pesos das ligações está vazia) **ENTÃO**
lista_ligação.= novo no;
SENÃO
lista_ligação.= insere inicio;
FIM-SE
Fim

//No **Módulo Sinapse** (synapses.c) é criada a lista ligada dos pesos das sinapses da rede.

Função InsertSynapseList ( );

inicio
/*
Esta função vai inserir o nó no início da lista, porque é mais eficiente. Os pesos sinápticos estão ligados aos neurónios anteriores. Os nós possuem três campos:
- valor do peso da sinapse, do tipo **float**,
- **endereço** do neurónio anterior e
- **ponteiro** para o próximo nó desta lista.

*/
SE (lista de pesos das sinapses está vazia) **ENTÃO**
lista_sinapse.= novo no;
SENÃO
lista_sinapse.= insere inicio;
FIM-SE
fim

//No **Módulo Neurónio** (datat_neuron.c) é criada a lista ligada dos neurónios da rede.

Função InsertNeuronGrafo ( );

inicio

/*

Esta função vai inserir o nó no fim da lista, porque é preciso ter os neurónios na ordem em que são criados. Os nós possuem quatro campos:

- nº de ordem do neurónio, do tipo **inteiro**,
- nº que identifica se o neurónio é da camada escondida ou da camada de saída, do tipo **inteiro**,
- valor da última activação, do tipo **float** e
- **ponteiro** para o próximo nó desta lista.

*/

SE (lista de neurónios está vazia) **ENTÃO**
lista_neuronio.= novo no;
SENÃO
lista_neuronio.= insere fim;
FIM-SE
fim

4 – Processamento da rede

/*
No **Módulo Rede Neuronal** (datat_neurnet.c) o processamento da rede é calculado, baseado na seguinte expressão [2] e [11]:

$$Y = F \left(\sum_{(m,n)\in S} w_{mn} f_m \left(\sum_{(i,n)\in C} w_{in} I_i \right) \right) \tag{4.4}$$

Sendo que:

S representa o conjunto de pesos das sinapses e

C representa o conjunto de pesos das ligações.
*/

Função AddInputsNeuron ( );
/*
Função que calcula o somatório de todos os pesos das ligações * os valores das entradas respectivas
*/

PARA de 1 até o nº de itens na lista de pesos das ligações
SE (lista de pesos das ligações está vazia) **ENTÃO**
RETURN 0.0;
SE (ao percorrer a lista de pesos das ligações e o peso da ligação iª está indicado para o neurónio da camada escondida e este não foi calculado) **ENTÃO**
somador <- entr * weight_con;
FIM-SE
FIM-SE
RETURN somador
FIM-PARA

Função sigmbip ( );

/*

Função que calcula a tangente hiperbólica dos valores guardados no "somador".
Esta função foi baseada na expressão 4.2 (secção 4.2), que representa a tangente hiperbólica na sua forma reduzida, uma vez que com uma exponencial o processamento da rede é mais rápido.

*/

PARA de 1 até o nº de itens na lista dos neurónios da camada escondida
SE (lista dos neurónios está vazia) **ENTÃO**
RETURN 0.0;
SE (ao percorrer a lista de neurónio e o valor do somador do neurónio iº foi calculado) **ENTÃO**
acumulador <- sigmbip (somador)
FIM-SE
FIM-SE
RETURN acumulador
FIM-PARA

Função AddAntecNeuron ( );

/*

Função que calcula o somatório de todos os pesos das sinapses * acumulador

*/

PARA de 1 até o nº de itens na lista das sinapses
SE (lista de pesos das sinapses está vazia) **ENTÃO**
RETURN 0.0;
SE (ao percorrer a lista de pesos das sinapses e valor do acumulador do neurónio iº foi calculado) **ENTÃO**
somador2 <- weight_snp* acumulador
FIM-SE
FIM-SE
RETURN somador2
FIM-PARA

5 – Fim do Processamento da rede

// O programa termina no **Módulo Rede Neuronal** (datat_neurnet.c).

Função PrintOutputNetwork ( );

/*

Função que calcula a saída da rede.

*/

SE (neurónio de saída) **ENTÃO**
saida <- somador2 + weight_snpbias * 1
FIM-SE
RETURN saida

4.2.3. Implementação da Rede Neuronal

O processo de desenvolvimento da implementação da RN e os diversos recursos utilizados no *MicroBlaze* são descritos nesta secção.

Neste projecto, foi utilizada a plataforma XPS, como já foi citado na secção 4.2.1, para configurar o sistema a nível de utilização de *hardware*, projecto de circuito lógico e a programação da implementação da RNA no *MicroBlaze*.

A configuração inicial do microprocessador *MicroBlaze* foi feita a partir do BSB (*Base System Builder*). A figura 4.6 apresenta uma das janelas do BSB no início de sua configuração.

A frequência de relógio, determinada na configuração do *MicroBlaze*, é de 100 MHz, visando obter o seu melhor desempenho, tendo em conta as suas limitações e a frequência suportada pelo FPGA. Quanto ao método de depuração, optou-se pelo método de depuração por *hardware*, visto que o trabalho envolvia a implementação em *hardware*. A memória local seleccionada foi de 64 KB e acrescentou-se o FPU, já mencionado na secção 4.1.2, para executar as operações necessárias da aplicação implementada e definidas em vírgula flutuante.

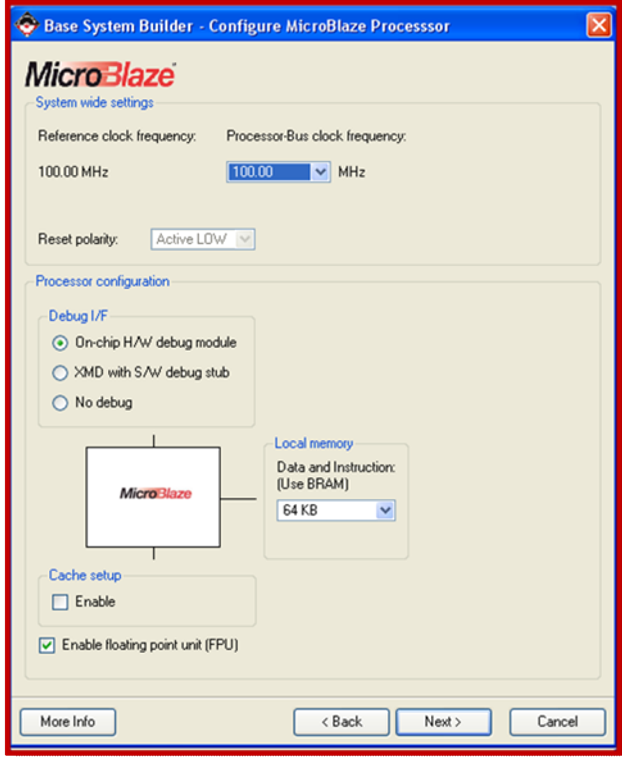


Figura 4.6 - Janela *Base System Builder*

Ao configurar as interfaces de entrada e saída do *MicroBlaze* foram seleccionados os periféricos *RS232-Uart* e *DDR_SDRAM*.

No periférico *RS232-Uart*, utilizaram-se os seguintes parâmetros:

- velocidade de 9600 bps,
- palavras de 8 *bits*,
- sem paridade,
- endereço base do periférico com 64 KB e
- portas para receber e transmitir dados.

No periférico *DDR_SDRAM*, utilizaram-se os seguintes parâmetros:

- endereço base do periférico de 256 MB e
- endereço base do controlador de 64 KB.

O *STDIO* e *STDOUT* foram redireccionados para o *RS232*, deste modo poderia controlar a execução do programa através da consola e também poderia utilizar as funções de entrada e de saída para a detecção de erros. Observou-se que as funções *printf( )* e *scanf( )* ao serem compiladas resultavam num grande número de instruções e, no intuito de ocupar menos memória, em algumas situações foram utilizadas a função *xil_printf( )* para a saída de dados.

O periférico interno *xps_timer_1* foi adicionado à configuração do *MicroBlaze*, uma vez que era necessário calcular o tempo decorrido no processamento da rede implementada. Utilizaram-se os seguintes parâmetros:

- um contador com 32 *bits* e
- endereço base de 64 KB.

No anexo C, apresentam-se descrições dos periféricos *RS232-Uart*, *DDR_SDRAM* e *xps_timer_1*.

Depois de definir a configuração do microprocessador *MicroBlaze*, foram criados os ficheiros do sistema embebido, dentre os quais são: *system.mhs*, *system.mss* e *system.xmp*.

O ficheiro MHS (*Microprocessor Hardware Specification*) possui a especificação do *hardware*. O *software* é especificado no ficheiro MSS (*Microprocessor Software Specification*). O ficheiro com a extensão *xmp* (*Xilinx Microprocessor Project*) possui a informação do projecto criado no XPS.

Para haver interacção entre o FPGA e o exterior foi necessário ligar as portas de entradas e de saídas aos pinos existentes no FPGA. Utilizou-se, como por exemplo, a seguinte sintaxe:

```
Net fpga_0_RS232_Uart_RX_pin LOC=F14;
```

O que significa que a porta `fpga_0_RS232_Uart_RX_pin` está a ser ligada ao pino existente no FPGA com o nome `F14`. Estes nomes estão definidos no *datasheet* da plataforma. No anexo B, apresenta-se a arquitectura do *Virtex II Pro* XC2VP30 onde o pino *F14* é identificado.

No sistema embebido *MicroBlaze*, deste projecto, foram configuradas quatro interfaces de *bus*: dois barramentos PLB (dados e instruções) e dois barramentos LMB (dados e instruções). Estes barramentos já foram analisados na secção 4.1.2. Observou-se no sistema embebido que cada periférico possuía um endereço base (*C_BASEADDR*) para utilização na

escrita e leitura dos registos. No anexo C, apresenta-se o diagrama de blocos do *MicroBlaze*, configurado com suas ligações e a estrutura do sistema deste projecto.

Com o sistema embebido estruturado, configuraram-se no XPS as definições da plataforma de *software*, conforme a necessidade da aplicação a ser implementada. Neste projecto, para que se cumprissem os objectivos foi necessário utilizar o *Xilkernel*.

O *Xilkernel* é uma biblioteca de *software* que permite definir funcionalidades e recursos especiais, ou seja, suporta multi-tarefas, sincronização com semáforos e *mutex* (semáforo binário), comunicação entre processos, tratamento de excepções e interrupções, *timer*, alocação dinâmica de memória e outras. No anexo C, apresentam-se as descrições dos parâmetros do *Xilkernel*.

A biblioteca *xilmfs* também foi incluída na configuração do *Xilkernel*, uma vez que se tornou necessário manipular ficheiros nos testes de comparação de redes com o Matlab.

No *software* SDK, mencionado na secção 4.2.1, desenvolveu-se a aplicação que implementava a Rede Neuronal Artificial, que é especificada na secçao 4.2.2. No anexo E, encontra-se o código programado em linguagem C.

Os dados e códigos foram definidos para a memória externa DDR-SDRAM e guardados no ficheiro de extensão *elf*. Apresentam-se, na figura 4.7, as memórias definidas neste trabalho.

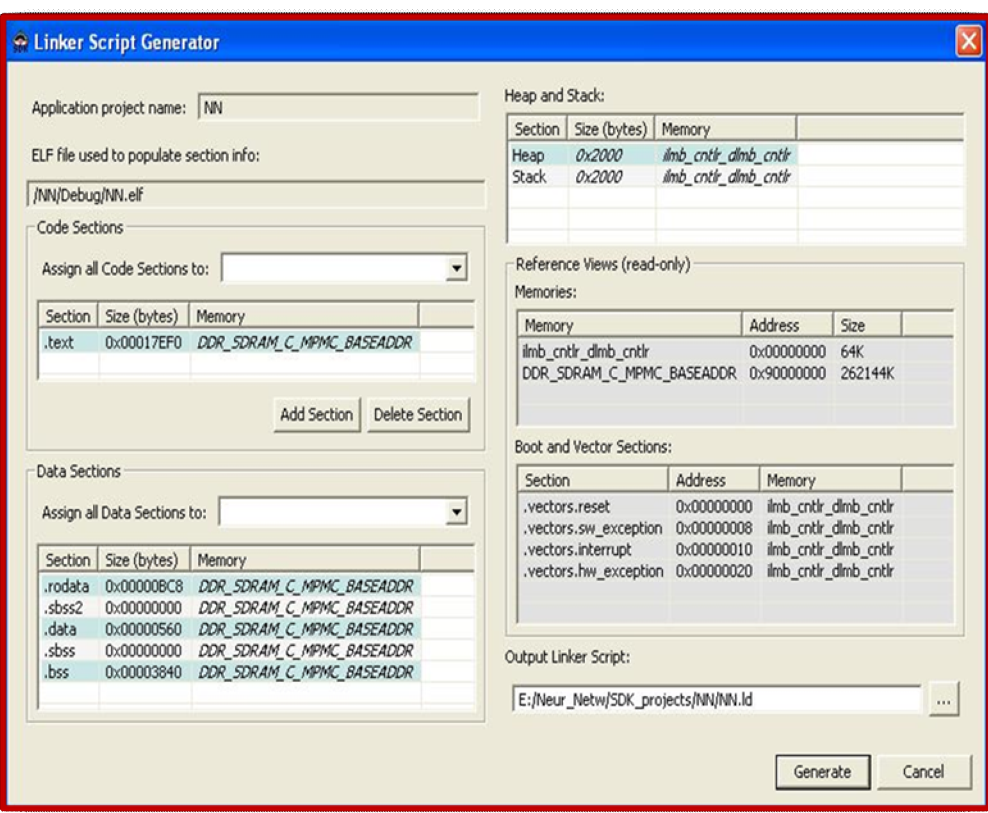


Figura 4.7 - *Linker Script Generator*

Para programar o FPGA e prosseguir com os testes, definiu-se o tipo de ligação ao *hardware*, no XMD (*Xilinx Microprocessor Debugger*) *Debug Options*, com parâmetros do JTAG *Boundary Scan*, que é um método para testar as interligações em circuitos de placas. Verificou-se que o tempo decorrido para a programação do FPGA foi de 49 segundos.

Utilizou-se o cabo JTAG do *Spartan* adaptado para o *Virtex-II Pro*. Esta adaptação foi feita uma vez que a disposição dos seus terminais não coincidia. A figura 4.8 mostra a adaptação feita no cabo JTAG.

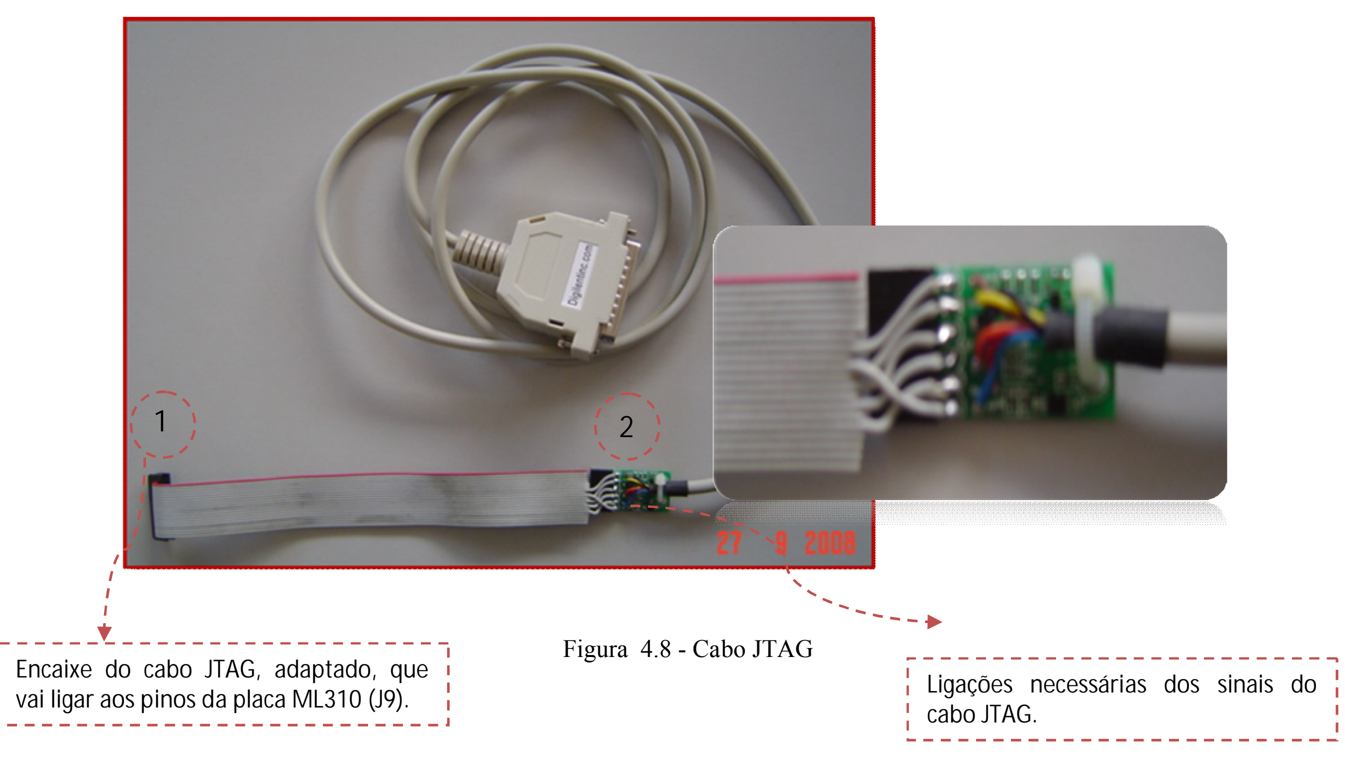


Figura 4.8 - Cabo JTAG

A adaptação, do cabo JTAG, consistiu em soldar as ligações da ficha de encaixe na placa ML310 ao cabo JTAG *Spartan*.

Os sinais do JTAG são descritos abaixo e foram ligados conforme o que é descrito na figura 4.9:

- TDI – JTAG *Data Input*
- TCK – JTAG *Clock Input*
- TMS – JTAG *Test Mode Input*
- TDO – JTAG *Data Output*

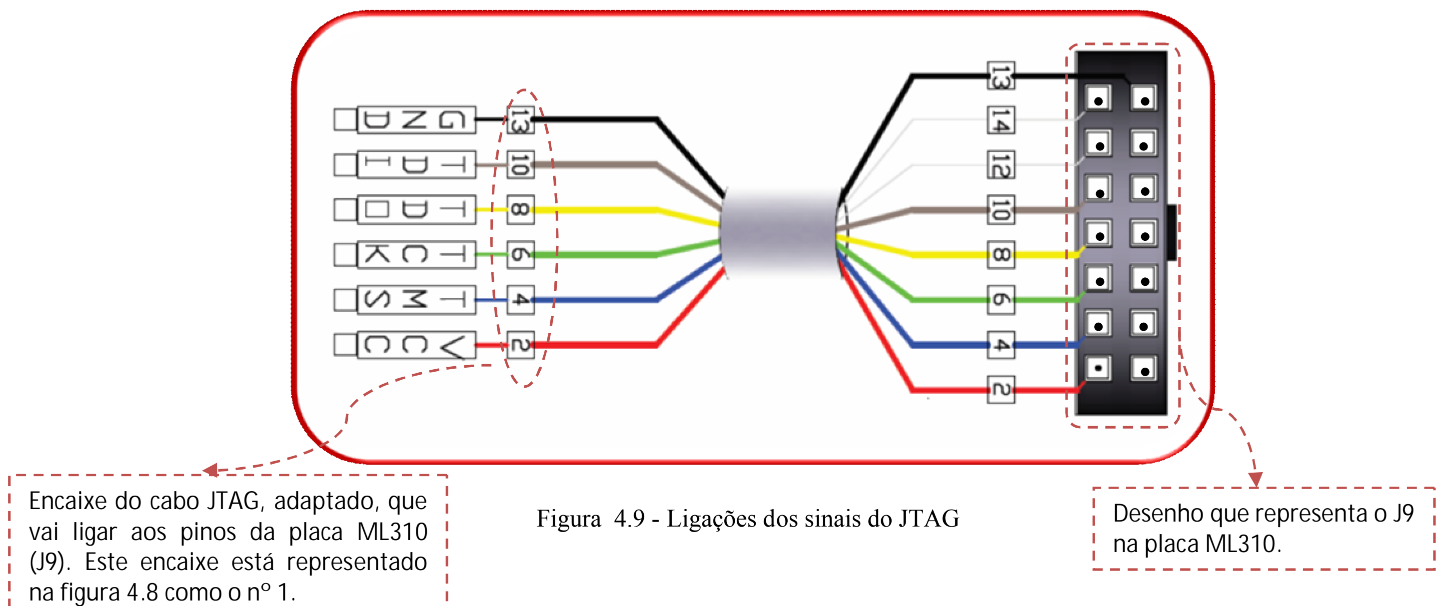


Figura 4.9 - Ligações dos sinais do JTAG

Tendo completado a configuração e programação do FPGA, verificou-se o funcionamento da Rede Neuronal implementada e foi exibido o sinal à saída da rede, no programa *HyperTerminal*.

5. TESTES e RESULTADOS ALCANÇADOS

"A ignorância afirma ou nega veementemente; a ciência duvida." - Voltaire – filósofo iluminista. (1694-1778)

Este capítulo é dedicado aos diversos testes efectuados na Rede Neuronal, implementada e detalhada no capítulo anterior e aos resultados alcançados. Apresentam-se testes que verificam o funcionamento correcto da rede, que modificam a dimensão da rede e avaliam-se as comparações com redes implementadas numa *toolbox*, utilizada com o Matlab e desenvolvida por *Magnus Noorgard* [49].

Os modelos, For6700 e FORGA001, utilizados nos testes da solução desenvolvida são de um forno eléctrico. Este forno eléctrico foi construído no âmbito de um projecto de investigação [2] e que tinha como objectivo desenvolver o forno com elevado grau de controlo dos perfis de temperatura para a indústria cerâmica e do vidro. A zona de operação pretendida foi em torno de 750˚ C, existindo um limite superior de 1200˚ C.

Na figura 5.1, tem-se uma vista esquemática deste forno.

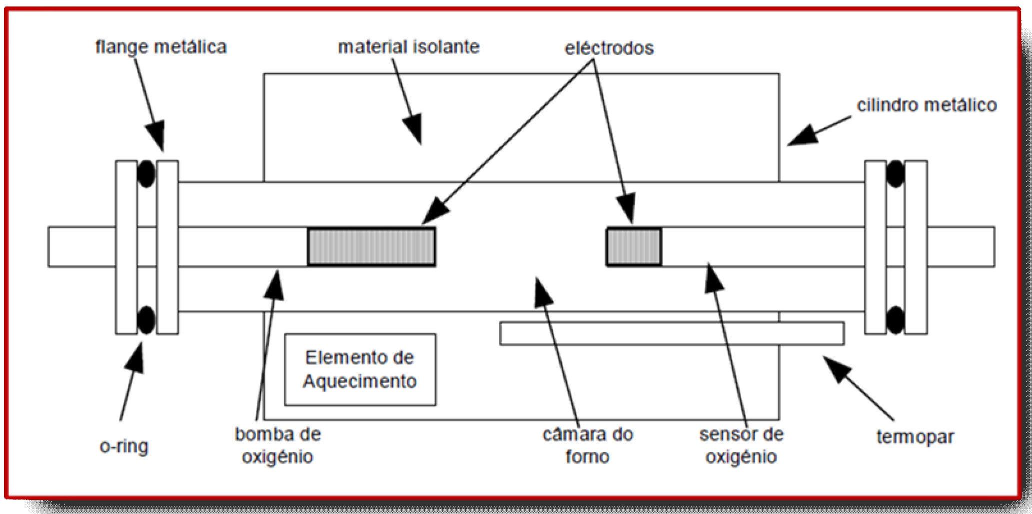


Figura 5.7 - Vista esquemática do forno [2]

O modelo de Rede Neuronal For6700, representado na figura 5.2, possui cinco entradas, três neurónios na camada escondida e um neurónio na camada de saída. Os valores dos pesos das ligações e dos pesos sinápticos são definidos nas variáveis $W1i$ e $w1i$, respectivamente e são descritos na figura 5.6.

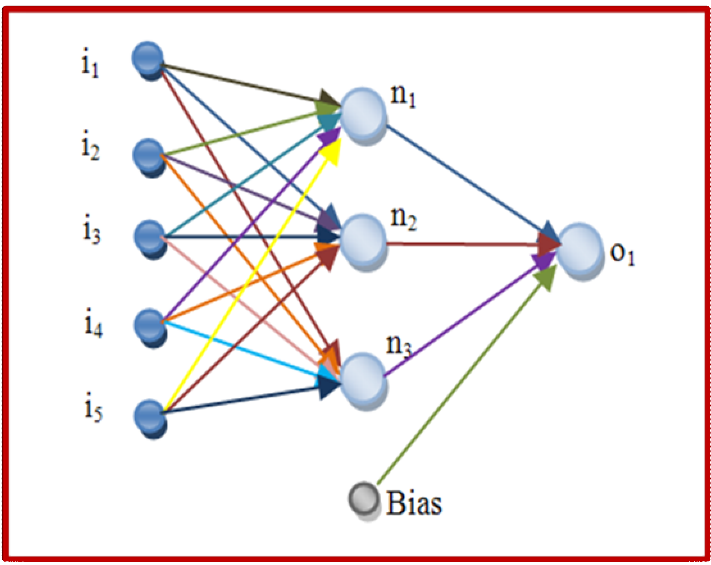


Figura 5.2 - Modelo de Rede Neuronal For6700

O modelo de Rede Neuronal FORGA001, representado na figura 5.3, possui cinco entradas, oito neurónios na camada escondida e um neurónio na camada de saída. Os valores dos pesos das ligações e dos pesos sinápticos são definidos nas variáveis *W1i* e *w1i*, respectivamente e são descritos na figura 5.7.

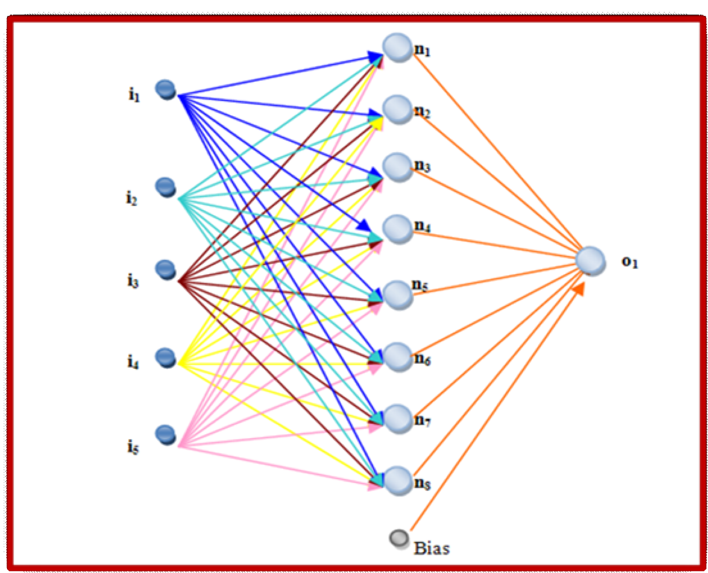


Figura 5.3 - Modelo de Rede Neuronal FORGA001

Para evidenciar o correcto funcionamento da rede implementada, utilizou-se a ferramenta *Microsoft Office Excel 2007*, para realizar os cálculos matemáticos dos dois modelos de rede. As figuras 5.6 e 5.7 apresentam, no *Excel 2007*, a expressão para os cálculos, a notação da rede, valores dos pesos das ligações e dos pesos sinápticos e o valor de saída da rede. Pôde-se observar nestes cálculos que, os valores obtidos são iguais aos valores impressos no *HyperTerminal*, figura 5.4 e 5.5, que correspondem aos valores da solução implementada no *MicroBlaze*.

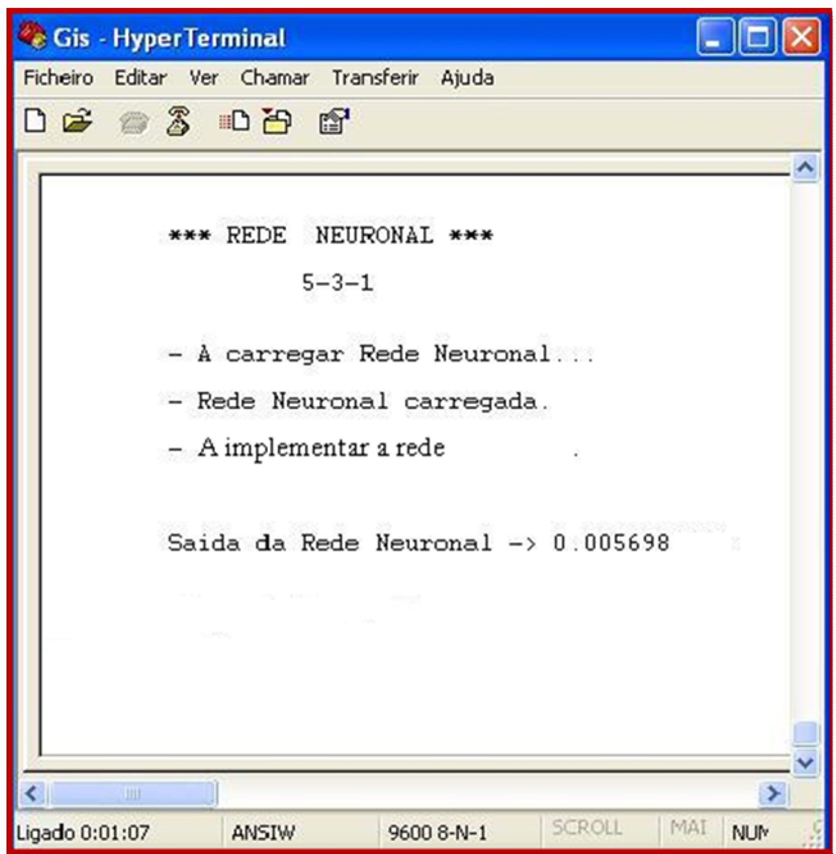


Figura 5.4 – Saída do Modelo RN For6700 no *HyperTerminal*

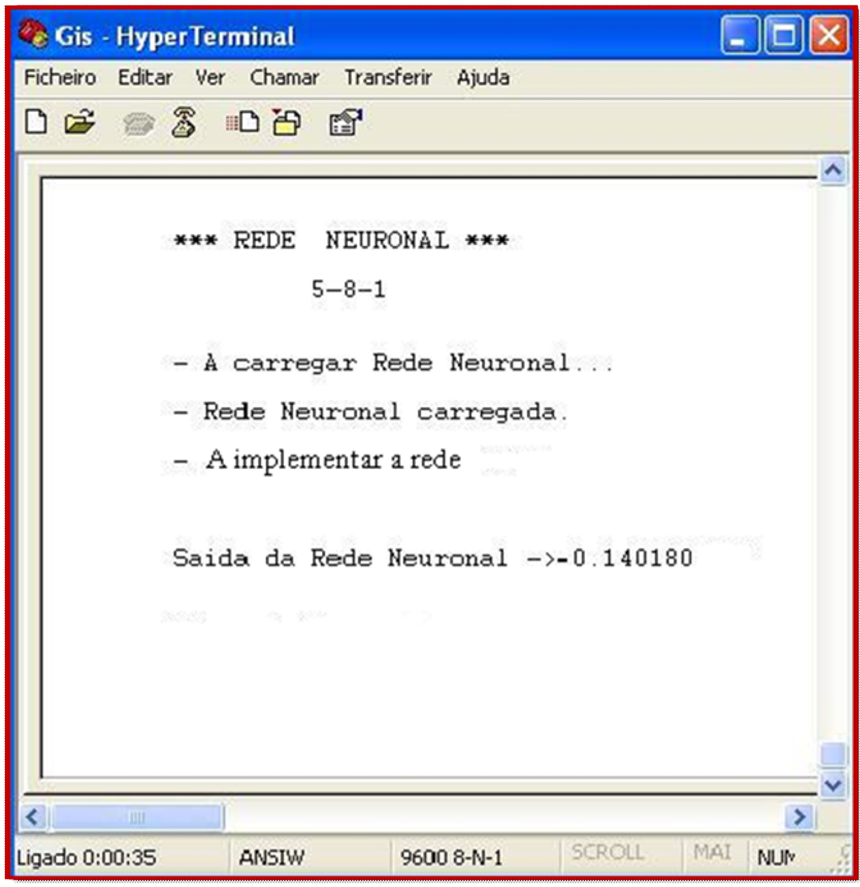


Figura 5.5 – Saída do Modelo RN FORGA001 no *HyperTerminal*

Foram testadas várias dimensões de redes, dentre as quais: 5-3-1, 5-8-1 e 5-20-1. Todas estas redes registaram valores baixos de utilização de memória, ocupando respectivamente 89 KB, 90 KB e 96 KB. Com estas ocupações de memória, podemos concluir que é possível trabalhar com redes de grandes dimensões, visto que a memória externa utilizada DDR-SDRAM e mencionada na secção 4.2.3, possui disponível o total de 256 MB.

Após verificar o bom funcionamento da rede e testar algumas dimensões de rede, fez-se para efeitos de comparação os testes a partir da ferramenta Matlab (versão 7.5). Pretendia-se fazer uma comunicação em série, entre o PC e o FPGA, para se obter um número grande de amostras. Estes testes de comunicação sofreram grandes atrasos e foram seriamente comprometidos, limitando o desempenho. Apesar desta situação, a solução implementada foi submetida a algumas comparações numa aplicação desenvolvida no Matlab, ou seja, a implementação da Rede Neuronal no FPGA realizou os seus testes e guardou num ficheiro os resultados para serem comparados com a solução das mesmas redes no Matlab.

Todos os testes, realizados no Matlab, também foram submetidos ao teste de sistema real do forno eléctrico [2], utilizando os ficheiros de modelos de Redes Neuronais Artificiais For6700 e FORGA001.

Cálculos da Saída da Rede Neuronal

Modelo For6700

Expressão

$$Y = F\left(\sum_{(m,n)\in S} w_{mn} f_m \left(\sum_{(i,n)\in C} W_{in} I_i \right) \right)$$

Notação

5 - 3 - 1

Pesos Ligações

	0,161993724781256	-0,061380199805640	0,001231074946929	-0,000428482832594	-0,546939657614124
$W1i=$	0,252808485955575	-0,133269545102912	0,001782202360917	-0,001787358552108	0,783536893041092
	-0,425370292195393	-0,156222631720689	-1,243009150655790	-1,118934426692030	-2,629129910497620

Pesos Sinápticos

$w1i=$	8,017540803246350	5,734809538026070	0,000026102819197	0,239985073830717

Cálculos

Y = -3,989005204849200 + 3,7547446650716700 + -0,0000261015750689 + 0,2399850738307170

Y = 0,005698432478113

Figura 5.6 - Saída da Rede Neuronal For6700 (*Microsoft Office Excel 2007)*

Cálculos da Saída da Rede Neuronal

Modelo FORGA001

Expressão

$$Y = F\left(\sum_{(m,n)\in S} w_{mn} f_m \left(\sum_{(i,n)\in C} W_{in} I_i \right) \right)$$

Notação

5 - 8 - 1

Pesos Ligações

$W1i$					
	1,429423440962900	1,488331811574410	1,491153263076050	0,558153932302629	0,872862776195729
	-0,383374665736380	-0,182986412536246	0,184468341754496	-0,004501052988985	1,295921196939340
	-0,715264576379988	-0,462758883584153	-0,143135844310413	-1,573603222796760	0,495284513616319
	0,883529543356063	0,942516804705757	1,383051806691100	1,078396922227960	0,303718652788134
	-1,516311631063890	-1,072708884315280	-1,817945512647910	-0,565191646202408	-0,871876377473462
	-0,520446815943277	0,335778269556382	-0,053957620953193	-0,000594040044206	-0,835193115042525
	-0,292137987991599	-0,207046588813645	0,156068038021719	-0,005362672910837	0,160622884320281
	1,117597911545120	0,719896073276892	1,119139942086680	1,028297045054380	0,124926196118447

Pesos Sinápticos

$w1i$

0,092184242493317	-1,646964017017540	-0,000020100381732	0,000646314915296	0,092408339397892	-3,601633055766160	-1,122837903491260	-0,000408117327076	-0,864112053465601

$Y =$ **-0,140180475122541**

Figura 5.7 - Saída da Rede Neuronal FORGA001 (*Microsoft Office Excel 2007*)

Descreve-se abaixo o pseudo-código da comparação das implementações das RNAs no FPGA e no Matlab.

/*
O programa recebe ficheiros, calcula a saída RN no Matlab, compara os valores e calcula o erro médio quadrático entre as implementações no FPGA e no Matlab.
*/

Carregar os valores de referência
/*
`ID050300` é o ficheiro de referência para a rede, seus valores serão utilizados nos cálculos da aplicação.
*/
```
load ID050300
```

Criar as amostras para o teste
```
Y=kiln_temp(1:5800);
```

Filtrar as sequências de entrada e de saída
```
Am = [1 2];
Bm = [2];
```

Carregar o modelo de RN utilizado no FPGA
/*
`For6700` é o ficheiro do modelo de rede neuronal. Este possui os valores do tamanho da rede e os valores dos pesos das ligações e dos pesos sinápticos.
*/
```
load For6700
```

Receber o ficheiro com os valores de saída da implementação no FPGA
/*
`fopen` é a função que abre o ficheiro recebido do FPGA, com os resultados do modelo implementado no FPGA.
*/
```
fid = fopen ('Yfpga1.mat');
```

/*
`textscan` é a função que importa os dados, em vírgula flutuante de precisão simples, recebidos do ficheiro gerado no FPGA.
*/
```
c = textscan(fid,'%f');
```

/*
`cell2mat` é a função que converte em uma simples matriz os dados importados do FPGA.
*/
```
Yfpga = cell2mat(c);
```

/*
`fclose` é a função que fecha o ficheiro recebido do FPGA.
*/
```
fclose(fid);
```

Calcular a saída da RN
/*
`nnoutput` é a função de uma *toolbox*, que calcula a saída da Rede Neuronal, os seus parâmetros foram definidos pelo ficheiro For6700.

`NetDeff` - estrutura da rede,
`W1f` - valores dos pesos das ligações,
`W2f` - valores dos pesos sinápticos e
`finputs` - valores das entradas da rede.
*/

```
para i=3 até o número_de_iterações
     ynew = nnoutput(NetDeff,W1f,W2f,finputs);
fim
```

Calcular o erro médio quadrático entre as implementações do FPGA e do Matlab

```
error = y - Yfpga;
errqd = (error'*error)/length(error)
```

O código de programação desenvolvido no Matlab está incluído no anexo E e refere-se à aplicação em que foi utilizado o modelo de Rede Neuronal For6700, representado na figura 5.2.

Todos os modelos testados no Matlab tinham as mesmas características da rede implementada no FPGA, ou seja, para efeito de comparação todos os dados eram iguais, tais como: as entradas da rede, os pesos aplicados e os ficheiros de referência.

Utilizaram-se três ficheiros de referência na rede testada no Matlab, que foram ID050300, ID13100A e DT030103. E na rede implementada no FPGA, estes três ficheiros de referência tiveram os nomes de U1s_1, U1s_2 e U1s_3. Estes ficheiros possuem valores aleatórios, em vírgula flutuante, e variam quantitativamente. Estes valores são utilizados nos cálculos para o processamento da rede implementada.

Nas figuras 5.8 a 5.13, apresentam-se os perfis de temperatura definidos para o forno eléctrico [2]. Estes perfis correspondem às características do forno, identificando o sistema construído.

Na figura 5.8, apresentam-se as saídas das implementações da Rede Neuronal Artificial no FPGA e no Matlab, em que foram utilizados o modelo de rede For6700 e os valores de referência do ficheiro ID050300. Para este teste, calculou-se o erro médio quadrático, obtendo o valor de 8.3660 e^{-20}.

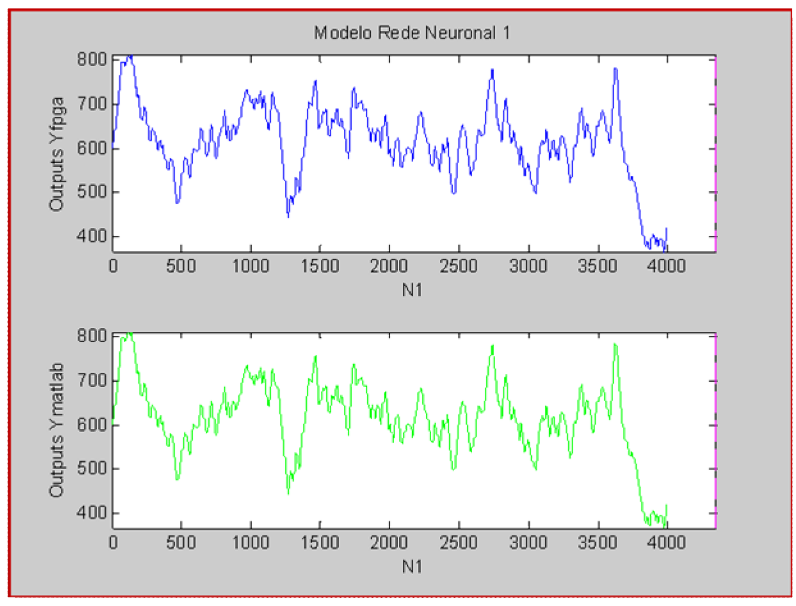


Figura 5.8 - Modelo For6700/ID050300

Na tabela 5.1, encontram-se 13 amostras com os resultados obtidos na implementação da rede no FPGA e no Matlab, em que foram utilizados o modelo de rede For6700 e os valores de referência do ficheiro ID050300. As entradas da rede são representadas por **y(i-1)**, **y(i-2)**, **U1s(i-1)** e **U1s(i-2)**, onde **y** corresponde ao conjunto dos valores de saída da rede e **U1s** corresponde ao conjunto dos valores fornecidos pelo ficheiro de referência ID050300.

Tabela 5.1 - Modelo For6700/ID050300

Amostra	Entradas				Resultado Matlab	Resultado FPGA
	y(i-1)	y(i-2)	U1s(i-1)	U1s(i-2)		
1	0,0056984325	0,0000000000	1,1402107613	1,1402107613	0,0159856677	0,0159856680
2	0,0159856677	0,0056984325	1,1402107613	1,1402107613	0,0299558574	0,0299558570
3	0,0299558574	0,0159856677	1,1402107613	1,1402107613	0,0468687209	0,0468687210
4	0,0468687209	0,0299558574	1,1402107613	1,1402107613	0,0661189683	0,0661189680
5	0,0661189683	0,0468687209	1,1402107613	1,1402107613	0,0872116604	0,0872116600
6	0,0872116604	0,0661189683	1,1402107613	1,1402107613	0,1097422352	0,1097422350
7	0,1097422352	0,0872116604	1,1402107613	1,1402107613	0,1333802429	0,1333802430
8	0,1333802429	0,1097422352	1,1402107613	1,1402107613	0,1578560528	0,1578560530
9	0,1578560528	0,1333802429	1,1402107613	1,1402107613	0,1829499581	0,1829499580
10	0,1829499581	0,1578560528	1,1402107613	1,1402107613	0,2084832311	0,2084832310
11	0,2084832311	0,1829499581	1,1402107613	1,1402107613	0,2343107676	0,2343107680
12	0,2343107676	0,2084832311	1,1402107613	1,1402107613	0,2603150345	0,2603150350
13	0,2603150345	0,2343107676	1,1402107613	1,1402107613	0,2864010887	0,2864010890

Na figura 5.9, apresentam-se as saídas das implementações da Rede Neuronal Artificial no FPGA e no Matlab, em que foram utilizados o modelo de rede For6700 e os valores de referência do ficheiro ID13100A. Para este teste, calculou-se o erro médio quadrático, obtendo o valor de 8.5372 e^{-20}.

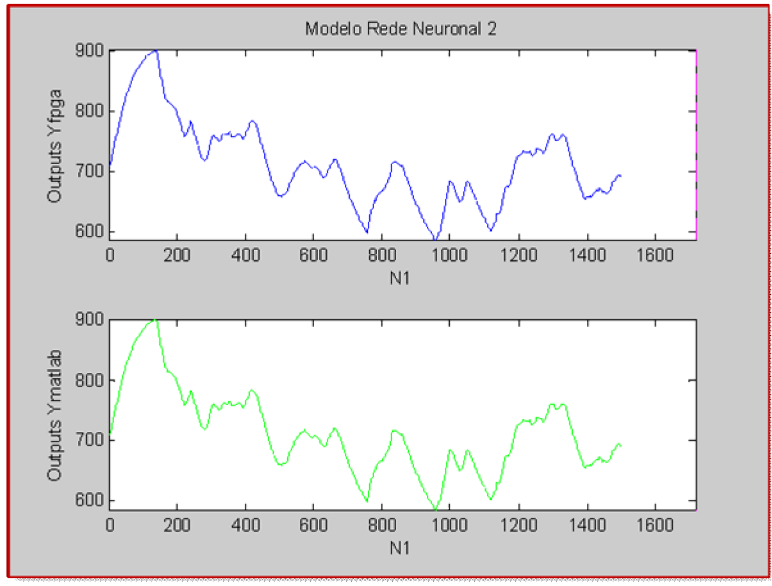


Figura 5.9 - Modelo For6700/ID13100A

Na tabela 5.2, encontram-se 13 amostras com os resultados obtidos na implementação da rede no FPGA e no Matlab, em que foram utilizados o modelo de rede For6700 e os valores de referência do ficheiro ID13100A. As entradas da rede são representadas por **y(i-1)**, **y(i-2)**, **U1s(i-1)** e **U1s(i-2)**, onde **y** corresponde ao conjunto dos valores de saída da rede e **U1s** corresponde ao conjunto dos valores fornecidos pelo ficheiro de referência ID13100A.

Tabela 5.2 – Modelo For6700/ID13100A

Amostra	Entradas				Resultado Matlab	Resultado FPGA
	y(i-1)	y(i-2)	U1s(i-1)	U1s(i-2)		
1	0,0089787735	0,0000000000	1,8198835111	1,8198835111	0,0251910028	0,0251910030
2	0,0251910028	0,0089787735	1,8198835111	1,8198835111	0,0472100361	0,0472100360
3	0,0472100361	0,0251910028	1,8198835111	1,8198835111	0,0738676490	0,0738676490
4	0,0738676490	0,0472100361	1,8198835111	1,8198835111	0,1042063718	0,1042063720
5	0,1042063718	0,0738676490	1,8198835111	1,8198835111	0,1374413425	0,1374413420
6	0,1374413425	0,1042063718	1,8198835111	1,8198835111	0,1729294954	0,1729294950
7	0,1729294954	0,1374413425	1,8198835111	1,8198835111	0,2101444848	0,2101444850
8	0,2101444848	0,1729294954	1,8198835111	1,8198835111	0,2486561526	0,2486561530
9	0,2486561526	0,2101444848	1,8198835111	1,8198835111	0,2881136393	0,2881136390
10	0,2881136393	0,2486561526	1,8198835111	1,8198835111	0,3282314467	0,3282314470
11	0,3282314467	0,2881136393	1,8198835111	1,8198835111	0,3687779096	0,3687779100
12	0,3687779096	0,3282314467	1,8198835111	1,8198835111	0,4095656497	0,4095656500
13	0,4095656497	0,3687779096	1,8198835111	1,8198835111	0,4504436651	0,4504436650

Na figura 5.10, apresentam-se as saídas das implementações da Rede Neuronal Artificial no FPGA e no Matlab, em que foram utilizados o modelo de rede For6700 e os valores de referência do ficheiro DT030103. Para este teste, calculou-se o erro médio quadrático, obtendo o valor de 8.1097 e^{-20}.

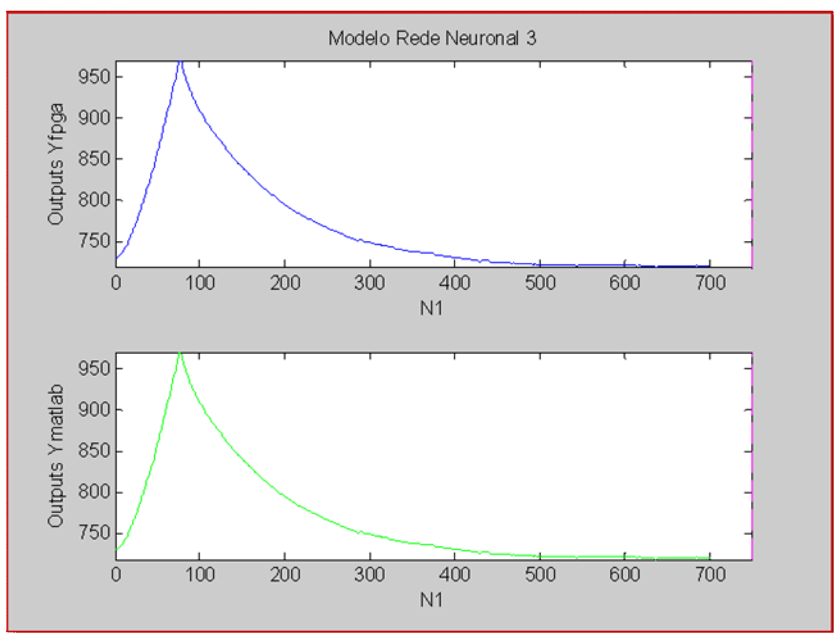


Figura 5.10 - Modelo For6700/DT030103

Na tabela 5.3, encontram-se 13 amostras com os resultados obtidos na implementação da rede no FPGA e no Matlab, em que foram utilizados o modelo de rede For6700 e os valores de referência do ficheiro DT030103. As entradas da rede são representadas por **y(i-1)**, **y(i-2)**, **U1s(i-1)** e **U1s(i-2)**, onde **y** corresponde ao conjunto dos valores de saída da rede e **U1s** corresponde ao conjunto dos valores fornecidos pelo ficheiro de referência DT030103.

Tabela 5.3 - Modelo For6700/ DT030103

Amostra	Entradas				Resultado Matlab	Resultado FPGA
	y(i-1)	y(i-2)	U1s(i-1)	U1s(i-2)		
1	0,0058658071	0,0000000000	1,1307522804	0,9932881944	0,0174011017	0,0174011020
2	0,0174011017	0,0058658071	1,0560076412	1,1307522804	0,0313380053	0,0313380050
3	0,0313380053	0,0174011017	0,8232718947	1,0560076412	0,0447292138	0,0447292140
4	0,0447292138	0,0313380053	1,0060897376	0,8232718947	0,0620343855	0,0620343860
5	0,0620343855	0,0447292138	0,8455379821	1,0060897376	0,0787928123	0,0787928120
6	0,0787928123	0,0620343855	0,9388899555	0,8455379821	0,0976598172	0,0976598170
7	0,0976598172	0,0787928123	0,9746774620	0,9388899555	0,1178738459	0,1178738460
8	0,1178738459	0,0976598172	1,0991832265	0,9746774620	0,1404750664	0,1404750660
9	0,1404750664	0,1178738459	1,1109498708	1,0991832265	0,1640594171	0,1640594170
10	0,1640594171	0,1404750664	1,1370058921	1,1109498708	0,1886269353	0,1886269350
11	0,1886269353	0,1640594171	1,0666150219	1,1370058921	0,2127802158	0,2127802160
12	0,2127802158	0,1886269353	1,0312985692	1,0666150219	0,2366654931	0,2366654930
13	0,2366654931	0,2127802158	1,2610879458	1,0312985692	0,2636111171	0,2636111170

Na figura 5.11, apresentam-se as saídas das implementações da Rede Neuronal Artificial no FPGA e no Matlab, em que foram utilizados o modelo de rede FORGA001 e os valores de referência do ficheiro ID050300. Para este teste, calculou-se o erro médio quadrático, obtendo o valor de 8.4874 e^{-20}.

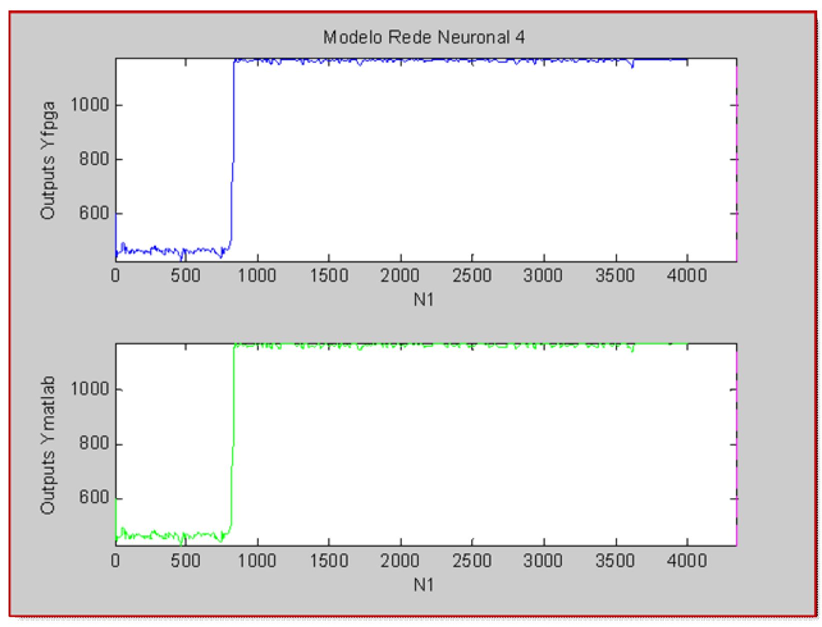


Figura 5.11 Modelo FORGA001/ID050300

Na tabela 5.4, encontram-se 13 amostras com os resultados obtidos na implementação da rede no FPGA e no Matlab, em que foram utilizados o modelo de rede FORGA001 e os valores de referência do ficheiro ID050300. As entradas da rede são representadas por **y(i-1)**, **y(i-2)**, **U1s(i-1)** e **U1s(i-2)**, onde **y** corresponde ao conjunto dos valores de saída da rede e **U1s** corresponde ao conjunto dos valores fornecidos pelo ficheiro de referência ID050300.

Tabela 5.4 Modelo FORGA001/ID050300

Amostra	Entradas				Resultado Matlab	Resultado FPGA
	y(i-1)	y(i-2)	U1s(i-1)	U1s(i-2)		
1	-0,1401804751	0,0000000000	1,1402107613	1,1402107613	-0,3313922283	-0,3313922280
2	-0,3313922283	-0,1401804751	1,1402107613	1,1402107613	-0,5425674356	-0,5425674360
3	-0,5425674356	-0,3313922283	1,1402107613	1,1402107613	-0,7545327422	-0,7545327420
4	-0,7545327422	-0,5425674356	1,1402107613	1,1402107613	-0,9521495107	-0,9521495110
5	-0,9521495107	-0,7545327422	1,1402107613	1,1402107613	-1,1281045029	-1,1281045030
6	-1,1281045029	-0,9521495107	1,1402107613	1,1402107613	-1,2820758231	-1,2820758230
7	-1,2820758231	-1,1281045029	1,1402107613	1,1402107613	-1,4043749742	-1,4043749740
8	-1,4043749742	-1,2820758231	1,1402107613	1,1402107613	-1,4725486789	-1,4725486790
9	-1,4725486789	-1,4043749742	1,1402107613	1,1402107613	-1,4737803817	-1,4737803820
10	-1,4737803817	-1,4725486789	1,1402107613	1,1402107613	-1,4217075497	-1,4217075500
11	-1,4217075497	-1,4737803817	1,1402107613	1,1402107613	-1,3508159407	-1,3508159410
12	-1,3508159407	-1,4217075497	1,1402107613	1,1402107613	-1,2952339709	-1,2952339710
13	-1,2952339709	-1,3508159407	1,1402107613	1,1402107613	-1,2707103148	-1,2707103150

Na figura 5.12, apresentam-se as saídas das implementações da Rede Neuronal Artificial no FPGA e no Matlab, em que foram utilizados o modelo de rede FORGA001 e os valores de referência do ficheiro ID13100A. Para este teste, calculou-se o erro médio quadrático, obtendo o valor de 8.7831 e^{-20}.

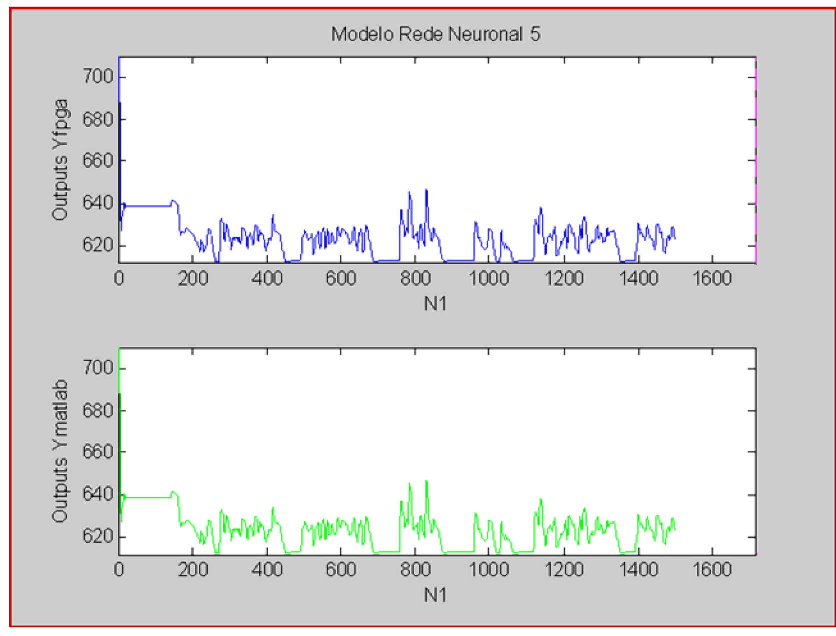


Figura 5.12 Modelo FORGA001/ID13100A

Na tabela 5.5, encontram-se 13 amostras com os resultados obtidos na implementação da rede no FPGA e no Matlab, em que foram utilizados o modelo de rede FORGA001 e os valores de referência do ficheiro ID13100A. As entradas da rede são representadas por **y(i-1)**, **y(i-2)**, **U1s(i-1)** e **U1s(i-2)**, onde **y** corresponde ao conjunto dos valores de saída da rede e **U1s** corresponde ao conjunto dos valores fornecidos pelo ficheiro de referência ID13100A.

Tabela 5.5 Modelo FORGA001/ ID13100A

Amostra	Entradas				Resultado Matlab	Resultado FPGA
	y(i-1)	y(i-2)	U1s(i-1)	U1s(i-2)		
1	-0,2086080246	0,0000000000	1,8198835111	1,8198835111	-0,4777052499	-0,4777052500
2	-0,4777052499	-0,2086080246	1,8198835111	1,8198835111	-0,7476306126	-0,7476306130
3	-0,7476306126	-0,4777052499	1,8198835111	1,8198835111	-0,9767881466	-0,9767881470
4	-0,9767881466	-0,7476306126	1,8198835111	1,8198835111	-1,1344844908	-1,1344844910
5	-1,1344844908	-0,9767881466	1,8198835111	1,8198835111	-1,2091324522	-1,2091324520
6	-1,2091324522	-1,1344844908	1,8198835111	1,8198835111	-1,2123165612	-1,2123165610
7	-1,2123165612	-1,2091324522	1,8198835111	1,8198835111	-1,1702120121	-1,1702120120
8	-1,1702120121	-1,2123165612	1,8198835111	1,8198835111	-1,1112643444	-1,1112643440
9	-1,1112643444	-1,1702120121	1,8198835111	1,8198835111	-1,0577457088	-1,0577457090
10	-1,0577457088	-1,1112643444	1,8198835111	1,8198835111	-1,0218038302	-1,0218038300
11	-1,0218038302	-1,0577457088	1,8198835111	1,8198835111	-1,0058076907	-1,0058076910
12	-1,0058076907	-1,0218038302	1,8198835111	1,8198835111	-1,0056890579	-1,0056890580
13	-1,0056890579	-1,0058076907	1,8198835111	1,8198835111	-1,0148444879	-1,0148444880

Na figura 5.13, apresentam-se as saídas das implementações da Rede Neuronal Artificial no FPGA e no Matlab, em que foram utilizados o modelo de rede FORGA001 e os valores de referência do ficheiro DT030103. Para este teste, calculou-se o erro médio quadrático, obtendo o valor de 8.4063 e^{-20}.

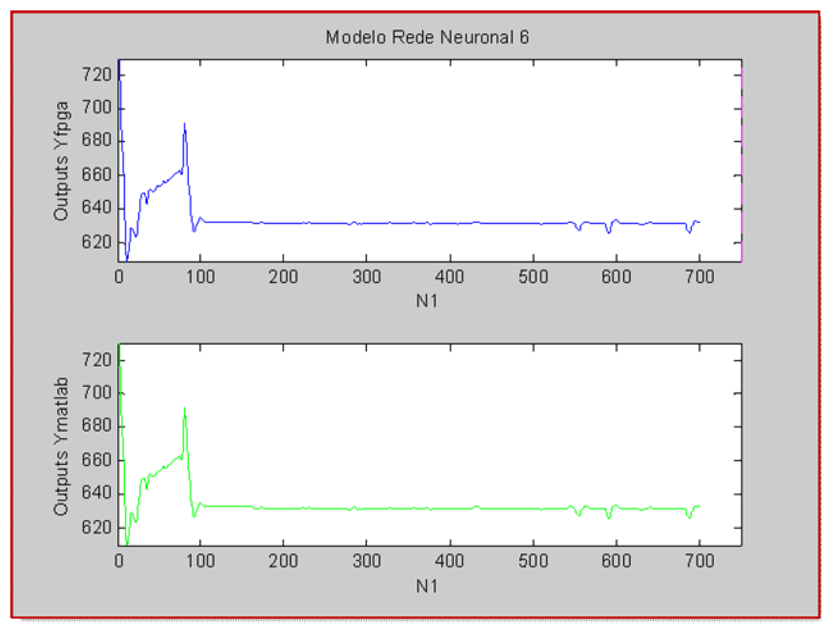


Figura 5.13 Modelo FORGA001/DT030103

Na tabela 5.6, encontram-se 13 amostras com os resultados obtidos na implementação da rede no FPGA e no Matlab, em que foram utilizados o modelo de rede FORGA001 e os valores de referência do ficheiro DT030103. As entradas da rede são representadas por **y(i-1)**, **y(i-2)**, **U1s(i-1)** e **U1s(i-2)**, onde **y** corresponde ao conjunto dos valores de saída da rede e **U1s** corresponde ao conjunto dos valores fornecidos pelo ficheiro de referência DT030103.

Tabela 5.6 Modelo FORGA001/ DT030103

Amostra	Entradas				Resultado Matlab	Resultado FPGA
	y(i-1)	y(i-2)	U1s(i-1)	U1s(i-2)		
1	-0,1248255752	0,0000000000	1,1307522804	0,9932881944	-0,3101249663	-0,3101249660
2	-0,3101249663	-0,1248255752	1,0560076412	1,1307522804	-0,5129023690	-0,5129023690
3	-0,5129023690	-0,3101249663	0,8232718947	1,0560076412	-0,7088318302	-0,7088318300
4	-0,7088318302	-0,5129023690	1,0060897376	0,8232718947	-0,9049111706	-0,9049111710
5	-0,9049111706	-0,7088318302	0,8455379821	1,0060897376	-1,0970617307	-1,0970617310
6	-1,0970617307	-0,9049111706	0,9388899555	0,8455379821	-1,2786930989	-1,2786930990
7	-1,2786930989	-1,0970617307	0,9746774620	0,9388899555	-1,4205458416	-1,4205458420
8	-1,4205458416	-1,2786930989	1,0991832265	0,9746774620	-1,4940304744	-1,4940304740
9	-1,4940304744	-1,4205458416	1,1109498708	1,0991832265	-1,4885562622	-1,4885562620
10	-1,4885562622	-1,4940304744	1,1370058921	1,1109498708	-1,4239806987	-1,4239806990
11	-1,4239806987	-1,4885562622	1,0666150219	1,1370058921	-1,3383513401	-1,3383513400
12	-1,3383513401	-1,4239806987	1,0312985692	1,0666150219	-1,2690709706	-1,2690709710
13	-1,2690709706	-1,3383513401	1,2610879458	1,0312985692	-1,2449091169	-1,2449091170

Verificam-se em todos os testes realizados e na tabela 5.7, do erro médio quadrático, que a precisão obtida, na implementação em FPGA com o *MicroBlaze* embebido, é muito boa. Estes comportamentos muito próximos dos resultados em Matlab não poderiam ser diferentes, uma vez que as redes possuíam os mesmos parâmetros e a implementação com o *MicroBlaze* não introduzia simplificações.

Tabela 5.7 Erro Médio Quadrático dos Modelos de RNs

Erro Médio Quadrático			
Rede Neuronal	Implementação no FPGA / Matlab		
	ID050300	ID13100A	DT030103
For6700	8,3660 e^{-20}	8,5372 e^{-20}	8,1097 e^{-20}
FORGA001	8,4874 e^{-20}	8,7831 e^{-20}	8,4063 e^{-20}

A implementação do microprocessador *MicroBlaze* embebido no FPGA *Virtex-II Pro*, desenvolvida neste trabalho, utilizou 12,89% das *slices* disponíveis e 10,51% das LUTs de quatro entradas e com um limite de frequência em 120,465MHz.

No anexo C, apresenta-se o relatório de utilização dos recursos de *hardware* do sistema embebido e a tabela do limite de frequência do relógio em cada módulo do sistema. Esta frequência máxima é calculada em condições normais de funcionamento e temperatura. No caso do *MicroBlaze* a frequência de 120,465MHz contribuiu para um incremento do desempenho, visto que a frequência máxima do sistema implementado no *Virtex II Pro* é de 100MHz.

Na tabela 5.8, podem ver-se os tempos de processamento dos modelos de rede testados. Foram aplicados para cada modelo duas funções de activação na camada escondida: função tangente hiperbólica e função linear.

Tabela 5.8 Tempo de Processamento dos Modelos de RNs

TEMPO DE PROCESSAMENTO (ms)					
Modelos Testados	Amostras	FPGA		Matlab	
		tanh	linear	tanh	Linear
For6700/DT030103 e FORGA001/DT030103	748	1434	72	62	50
For6700/ID13100A e FORGA001/ID13100A	1714	2147	108	109	104
For6700/ID050300 e FORGA001/ID050300	4348	> 42 s	429	283	250

Com os resultados dos tempos, verificou-se que o processamento em *hardware* foi muito superior ao realizado no Matlab (com um processador de 2,8 GHz), não tendo sido optimizado o sistema implementado.

Observou-se que utilizando a função de activação linear, na camada escondida, o tempo no FPGA foi reduzido, ou seja, a função tangente hiperbólica é determinante no tempo de processamento da rede, influencia directamente nos cálculos dos tempos. Sendo assim, observa-se a necessidade de utilização de uma outra solução para o cálculo da função de activação, para que o *hardware* tenha um melhor desempenho.

Nos últimos testes, utilizando a função tangente hiperbólica, na camada escondida, ocorreram *overflow*, o que se justifica, uma vez que o microprocessador estava a ser executado a 100MHz e possui 32 *bits*.

6. CONCLUSÃO e TRABALHOS FUTUROS

Alguns homens vêem as coisas como são e dizem: "POR QUÊ?"
Eu sonho com coisas que nunca existiram e digo: "POR QUE NÃO?"

6.1. Conclusão

O objectivo principal definido neste trabalho foi atingido, implementou-se em *hardware* uma Rede Neuronal Artificial com um microprocessador embebido e verificou-se que a solução nesta implementação fornecia um sinal de activação à saída, assemelhando-se ao que acontece no cérebro humano.

Neste trabalho, a teoria e a prática orientaram o aumento da investigação em relação ao conhecimento das Redes Neuronais e na utilização de *hardware*.

Puderam verificar-se aspectos importantes quanto à implementação em *hardware/software* entre eles a flexibilidade e o tempo de desenvolvimento do projecto. Verificou-se também a grande potencialidade do *hardware* escolhido para a implementação, observando-se que, após toda a configuração da rede, foram utilizados 33% dos recursos oferecidos pelo FPGA *Virtex-II Pro*.

O sistema desenvolvido foi testado em vários modelos de Rede Neuronal e num sistema real do controlo de temperatura de um forno, que possibilitou verificar o desempenho da rede, a possibilidade de aplicação desta implementação nas diversas áreas científicas e uma precisão muito próxima do desejado. Esta precisão deve-se à opção pela notação numérica de vírgula flutuante, o que permitiu a representação de números muito grandes para uma utilização directa de pesos na Rede Neuronal.

Concluiu-se neste trabalho que o tempo de processamento da rede implementada no *hardware* foi mais lento do que o tempo de processamento num PC, registaram-se valores de 1434 ms, 2147 ms e superior a 42 s, utilizando a função de activação tangente hiperbólica, na camada escondida. Enquanto que no Matlab, usando um processador de 2,8 GHz, registaram-se valores compreendidos entre os 62 ms e 283 ms nos modelos de rede testados. Este resultado é expectável, devido a estar a comparar-se um processador *hard core* com um *soft core* de menor frequência de relógio.

A implementação com o *MicroBlaze* apresenta, no entanto, muitas vantagens como o menor preço do que um PC, o baixo risco de se tornar obsoleto e a fácil integração com outras tecnologias.

O teste adicional sem as funções de activação do tipo tangente hiperbólica, permite verificar que é nesta parte do processamento em que existe mais gasto de tempo. Esta análise permite concluir que é conveniente, como trabalho futuro, implementar esta função em *hardware*. Para esta implementação poderá ser utilizada uma LUT, com maior ou menor resolução ou mesmo um microprocessador *MicroBlaze* adicional para utilizar a função tangente hiperbólica.

A principal contribuição científica deste trabalho foi disponibilizar uma Rede Neuronal Artificial, podendo ser aplicada em diferentes dimensões de rede. A precisão da rede implementada em um *hardware* e a realização dos testes em um sistema real também foram contributos bastante importantes.

6.2. Trabalhos Futuros

Para aumentar os recursos da implementação deste projecto, propõem-se utilizar vários *MicroBlazes* para calcular a saída de forma mais rápida e desenvolver a função de activação tangente hiperbólica com a utilização do algoritmo CORDIC [39], [40] e [41]. Este algoritmo permite o cálculo de funções trigonométricas de uma forma simples, rápida e precisa, sendo possível realizar diversas tarefas de processamento digital de sinais.

As Redes Neuronais Artificiais podem ser aplicadas em diversas áreas e em projectos diversificados, como por exemplo: na área da Biomedicina acrescentaria o treino e a aprendizagem de rede ao trabalho implementado e deste modo desenvolveria a recuperação de algumas actividades perdidas na mente de uma pessoa portadora de *Alzheimer;* na Engenharia possibilitaria, em espaços confinados, a detecção de gases inodoros e venenosos.

[1] R.P.Lippmann. *An introduction to computing with neural nets*. IEEE ASSP Magazine, pages 4—22, 1987.

[2] Morgado Dias. Técnicas de controlo não-linear baseadas em Redes Neuronais: do algoritmo à implementação. Tese de Doutoramento em Engenharia Electrotécnica na Universidade de Aveiro em 2005, disponível na internet em http://dme.uma.pt/morgado/Down/TeseCompletaFMD.pdf, 2008.

[3] David Pellerin and Scott Thibault. *Practical FPGA Programming in C*; Prentice Hall PTR, 2005. ISBN 0-13-154318-0.

[4] Information Society Technologies: FACETS, disponível na internet em http://cordis.europa.eu/ictresults/index.cfm?section=news&tpl=article&ID=90451, Abril de 2009.

[5] J. F. Teixeira. *Mentes e Máquinas*; Artes Médicas, 1998. ISBN 85-7307-329-2.

[6] Ministerio de Educación (España): Proyecto Biosfera, disponível na internet em http://recursos.cnice.mec.es/biosfera/index.htm, Março de 2009.

[7] Grupo de Sistemas Inteligentes: Redes Neurais, disponível na internet em http://www.din.uem.br/ia/intelige/neurais2/, Março de 2008.

[8] Barreto, J. M. (2002). Introdução as Redes Neurais Artificiais, disponível em http://www.inf.ufsc.br/~barreto/tutoriais/Survey.pdf, Junho de 2008.

[9] Cérebro humano: Nem maior nem menor, disponível na internet em http://ambienteacreano.blogspot.com/2009/02/cerebro-humano-nem-maior-nem-menor-do.html, Abril de 2009.

[10] UNL – Universidade Nova de Lisboa: Funcionalidades da Dopamina, disponível em http://www.dq.fct.unl.pt/cadeiras/qpn1/proj/dopamina/funcionalidades/inicial-func.htm, Fevereiro de 2008.

[11] M.T. Hagan, H.B.Demuth and M. Beale. *Neural Network Design*s – A Comprehensive Foundation Second Edition – Simon Haykin – Prentice Hall, 1999.

[12] Absolute Astronomy, disponível na internet em http://www.absoluteastronomy.com/topics/Walter_Pitts, Junho 2008.

[13] W. Mcculloch and W. Pitts. *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Bulletin of Mathematical Biophysics 5 - 1943.

[14] D. Hebb. *The Organization of Behavior*. John Wiley & Sons, New York - 1949.

[15] UFRJ - Universidade Federal do Rio de Janeiro: Redes Neurais, disponível na internet em http://equipe.nce.ufrj.br/thome/p_grad/nn_ic/transp/T3_fundamentos.pdf, Junho 2009.

[16] F. Rosenblatt. *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review, Vol. 65, pp. 386-408. 1958.

[17] F. Rosenblatt. *Principles of Neurodynamics*, Spartan Books, Washington, DC, 1962.

[18] W. Widrow and M. Hoff. *Adaptive switching circuits*. WESCON Convention Record - 1960.

[19] Laboratório Nacional de Computação Científica: Tutorial de Redes Neurais, disponível em http://www.lncc.br/~labinfo/tutorialRN/index.htm, Fevereiro de 2008.

[20] Y.Chauvin and D.E.Rumelhart. *Backpropagation: theory, architectures, and applications*; Lawrence Erlbaum, 1995. ISBN 0-80-581259-8.

[21] D.E. Rumelhart, G.E. Hilton and R.J. Williams. *Learning Internal Representations by Error Propagation.* In D.E. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition 1: Foundations*, volume 1, pages 318-362. M.I.T. Press, Cambridge, MA - 1986.

[22] Akademia Górniczo-Hutnicza: Backpropagation, disponível na internet em http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html, Maio de 2009.

[23] G. Cybenko. Approximation by superposition of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:492-499, 1989.

[24] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximatiors. *Neural Networks*, 2:359-366, 1989.

[25] Inovação Tecnologia: Redes neuronais permitem que robôs sintam empatia, disponível em http://www.inovacaotecnologica.com.br/noticias/noticia.php?artigo=redes-neurais-permitem-que-robos-sintam-empatia – Julho de 2008.

[26] Wikipedia. The Free Encyclopedia: Adaptive Resonance Theory, disponível em http://en.wikipedia.org/wiki/Adaptive_resonance_theory, Dezembro de 2007.

[27] Pedro Ferreira, Pedro Ribeiro, Ana Antunes and Fernando Morgado Dias. A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function. *Neurocomputing Vol. 71, Issues 1-3, Pages 71-77*, December 2007.

[28] Info Wester. Emerson Alecrim: Redes Neurais Artificiais, disponível em http://www.infowester.com/redesneurais.php, Maio de 2008.

[29] UFS – Universidade Federal de Sergipe: Comparação entre truncamento binário e decimal em uma Rede Neuronal Artificial implementada em FPGA, disponível na internet em http://www.dcomp.ufs.br/~lbrunelli/fe/truncamento_0.pdf, Junho de 2008.

[30] SBU - Sistema de Bibliotecas da UNICAMP – Universidade Estadual de Campinas: A utilização de redes neurais artificiais no projeto de receptores FH-CDMA, disponível em http://libdigi.unicamp.br/document/?code=vtls000129614, Fevereiro de 2009.

[31] D.A. Pomerleau. Neural Computation: *Efficient Training of Artificial Neural Networks for Autonomous Navigation*, Vol. 3, N.1, Pages 88-97, 1989.

[32] C.Apté, J.Kastner, editors, Financial Applications. Special Issue, *IEEE Expert, Vol.2, N.3*, 1987.

[33] F.M. Dias, A. Antunes and A.M.Mota. Commercial hardware for artificial neural networks: A survey. *SICICA – 5th IFAC International Symposium on Inteligent Components and Instruments for Control Applications, Aveiro*, 2003.

[34] F.M. Dias, A. Antunes A. Mota. Artificial neural networks: A review of commercial hardware. *Engineering Applications of Artificial Intelligence, IFAC, 17/8:945—952*, 2004.

[35] Xilinx: Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet, disponível na internet em http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf, Novembro de 2007.

[36] J.L. Ayala, A.G. Lomeña, M. López-Vallejo, A. Fernández. Design of a pipelined hardware architecture for real-time neural network computations. *IEEE Midwest Symposium on Circuits and Systems, USA*, 2002.

[37] M. Leon, A. Castro, R. Ascenccio An artificial neural network on a field programmable gate array as a virtual sensor. *Proceedings of the Third International Workshop on Design of Mixed-Mode Integrated Circuits and Applications, Puerto Vallarta, Mexico, pp. 114–117*, 1999.

[38] “Feasibility of Floating-Point Arithmetic in FPGA Based Artificial Neural Networks”, K. Nichols, M. Moussa, S. Areibi, CAINE, *San Diego, CA, 2002, pp. 8–13*.

[39] "Efficient Sigmoid Function for Neural Networks Based FPGA Design", Xi Chen1, Gaofeng Wang, Wei Zhou, Sheng Chang, and Shilei Sun, *ICIC 2006, LNCS 4113, pp. 672-677, Springer- Verlag Berlin Heidelberg* 2006.

[40] "Electronic implementation of a Neural Observer in FPGA technology: application to the control of electric vehicle", Ghariani M., Kharrat M.W, Masmoudin N., Kamoun L., *16th International Conference on Microelectronics*, 2004.

[41] "Application of CORDIC Algorithm to Neural Networks VLSI Design", Meng Qian, *IMACS Multiconference on Computational Engineering in Systems Applications*, 2006.

[42] K.Tabari, M. Boukadoum, A. Bensaoula, D. Starikov. Neural Network Processor for a FPGA- based Multiband Fluorometer Device. *The Internacional Workshop on Computer Architecture for Machine Perception and Sensing*, 2006.

[43] Xilinx: ML310 User Guide, disponível na internet em http://www.xilinx.com/support/documentation/boards_and_kits/ug068.pdf, Outubro de 2007.

[44] Xilinx: Virtex-II Pro and Virtex-II Pro FPGA User Guide, disponível na internet em http://www.xilinx.com/support/documentation/user_guides/ug012.pdf, Outubro de 2007.

[45] Doom9: Null Modem, disponível na internet em http://www.doom9.org/DigiTV/images/dreambox/Null_%20Modem.gif, Fevereiro de 2008.

[46] Lammert Bies, Homepage. RS232 serial null modem cable wiring and tutorial, disponível em http://www.lammertbies.nl/comm/info/RS-232_null_modem.html, Março de 2008.

[47] Xilinx: MicroBlaze Processor Reference Guide, disponível na internet em http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf, Junho de 2008.

[48] Xilinx: EDK Concepts, Tools, and Techniques, disponível na internet em http://www.xilinx.com/support/documentation/manuals/xilinx10/edk_ctt.pdf, Junho de 2008.

[49] Magnus Noorgard. *System Identification and Control with Neural Networks*. PhD thesis, Department of Automation, Technical University of Denmark, 1996.

ANEXOS

ANEXO A – Glossário de Termos e Abreviaturas

ACE CF - *Advanced Configuration Environment Compact Flash*
Adaline - *Adaptive Linear Element*
ALU - *Arithmetic Logic Unit*
ART - *Adaptive Resonance Theory*
ASIC - *Application Specific Integrated Circuit*
ATX - *Advanced Technology Extended*

BGA - *Ball Gride Array*
BSB - *Base System Builder*

CLB - *Configurable Logic Blocks*
CTS - *Clear To Send*

DDR - *Double Data Rate*
DDR DIMM - *Double Data Rate Dual Inline Memory Module*
DDR_SDRAM - *Double Data Rate Synchronous Dynamic Random Access Memory*
DIP - *Dual-Inline Package*
DLMB - *Data Interface Local Memory Bus*
DPLB - *Data Interface Processor Local Bus*
DSR - *Data Set Ready*
DTR - *Data Terminal Ready*
DXCL_M - *Data Interface Xilinx CacheLink (Master)*
DXCL_S - *Data Interface Xilinx CacheLink (Slave)*

EDK - *Embedded Development Kit*
ETANN - *Electrically Trainable Analog Neural Network*

FACETS - *Fast Analog Computing with Emergent Transient States*
FF896 - *Flip-Chip Fine-Pitch 896*
FH-CDMA - *Frequency Hopping – Code Division Multiple Access*
FNN - *Feedforward Neural Network*
FPGA - *Field Programmable Gate Array*
FPU - *Floating Point Unit*
FSL - *Fast Simple Link*

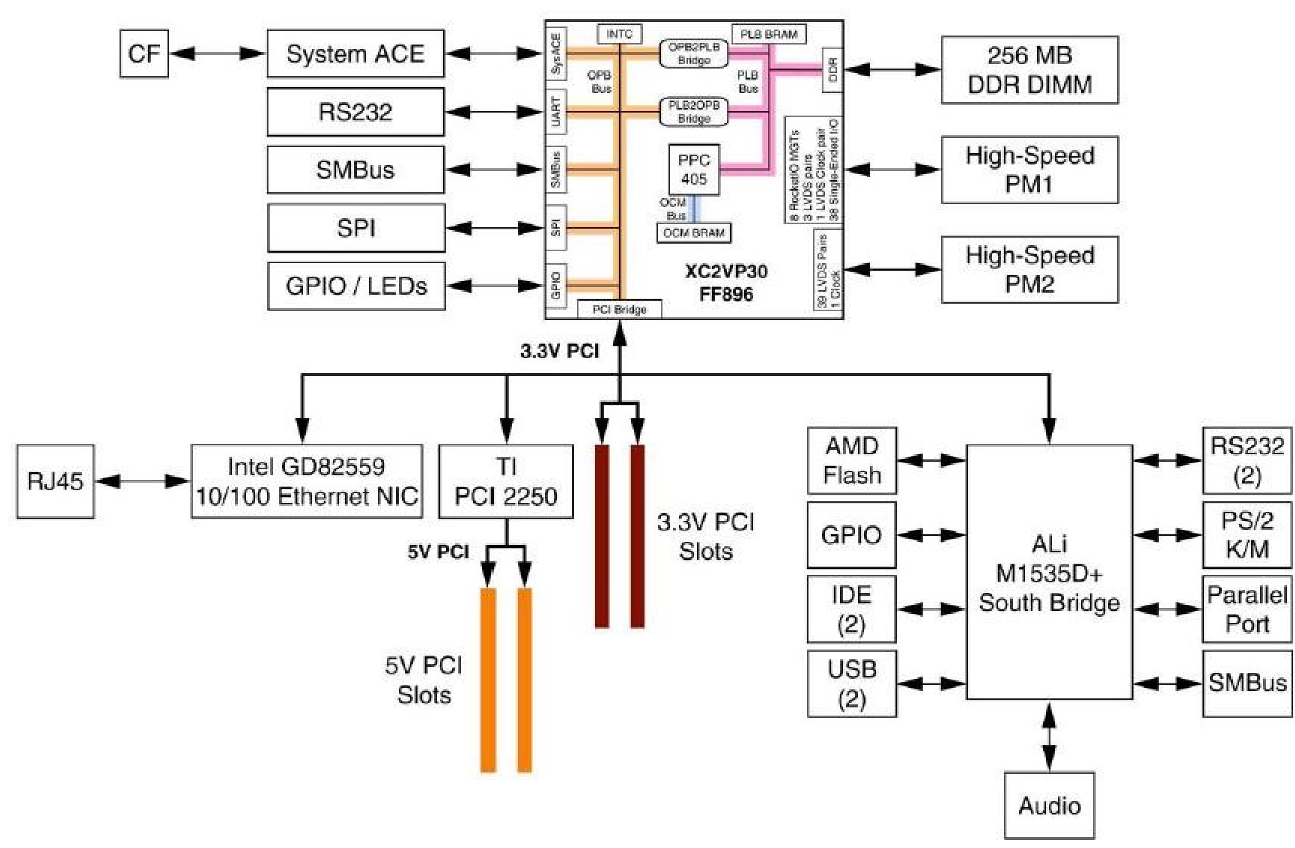
GCC - *Gnu is not Unix Compiler Collection*
GND - *Ground*

HDL	-	*Hardware Description Language*
HW	-	*Hardware*
IDE	-	*Integrated Drive Electronics*
IEEE	-	*Institute of Electrical and Electronics Enginners*
IF	-	*Instruction Fetch*
ILMB	-	*Instruction Interface Local Memory Bus*
IOB	-	*Input Output Blocks*
IPLB	-	*Instruction Interface Processor Local Bus*
IXCL_M	-	*Instruction Interface Xilinx CacheLink (Master)*
IXCL_S	-	*Instruction Interface Xilinx CacheLink (Slave)*
JTAG	-	*Join Test Action Group*
LCD	-	*Liquid Crystal Display*
LMB	-	*Local Memory Bus*
LMS	-	*Least Mean-Square*
LUT	-	*Look up Table*
MHS	-	*Microprocessor Hardware Specification*
MPD	-	*Microprocessor Peripheral Description*
MSS	-	*Microprocessor Software Specification*
OCR	-	*Optical Character Recognition*
OS	-	*Operating System*
PC	-	*Personal Computer*
PCI	-	*Peripheral Component Interconnect*
PC4 JTAG	-	*Parallel Cable IV Join Test Action Group*
PDP	-	*Parallel Distributed Processing*
PLB	-	*Processor Local Bus*
QNX	-	*Queue Nicks Unix*
RAM	-	*Random Access Memory*
RBF	-	*Radial Basis Function*
RD	-	*Receive Data*
RISC	-	*Reduced Instruction-Set Computer*
RN	-	Rede Neuronal
RNAs	-	Redes Neuronais Artificiais
RS232	-	*Recommended Standard-232*
RTS	-	*Request To Send*

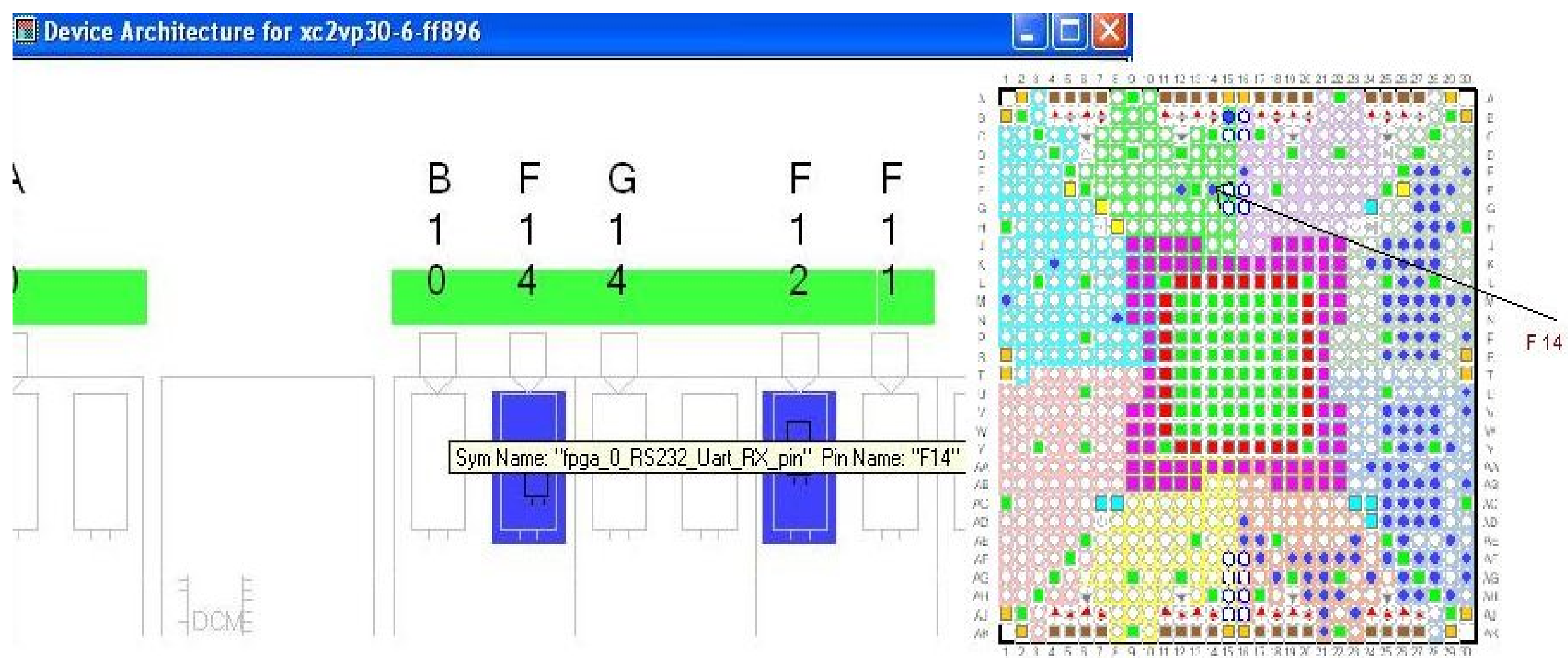
SDK	-	*Software Development Kit*
SOM	-	*Self Organizing Feature Maps*
STDIO	-	*Standard Input*
STDOUT	-	*Standard Output*
SW	-	*Switch*
SW	-	*Software*
TCK	-	*Test Clock Input*
TD	-	*Transmit Data*
TDI	-	*Test Data Input*
TODO	-	*Test Data Output*
TMS	-	*Test Mode Input*
UART	-	*Universal Asynchronous Receiver/Transmitter*
UCF	-	*User Constraints File*
USB	-	*Universal Serial Bus*
XMD	-	*Xilinx Microprocessor Debugger*
XMP	-	*Xilinx Microprocessor Project*
XOR	-	Ou Exclusivo
XPS	-	*Xilinx Platform Studio*

ANEXO B – Especificações do Hardware

- **Diagrama de blocos do ML310 [43]**

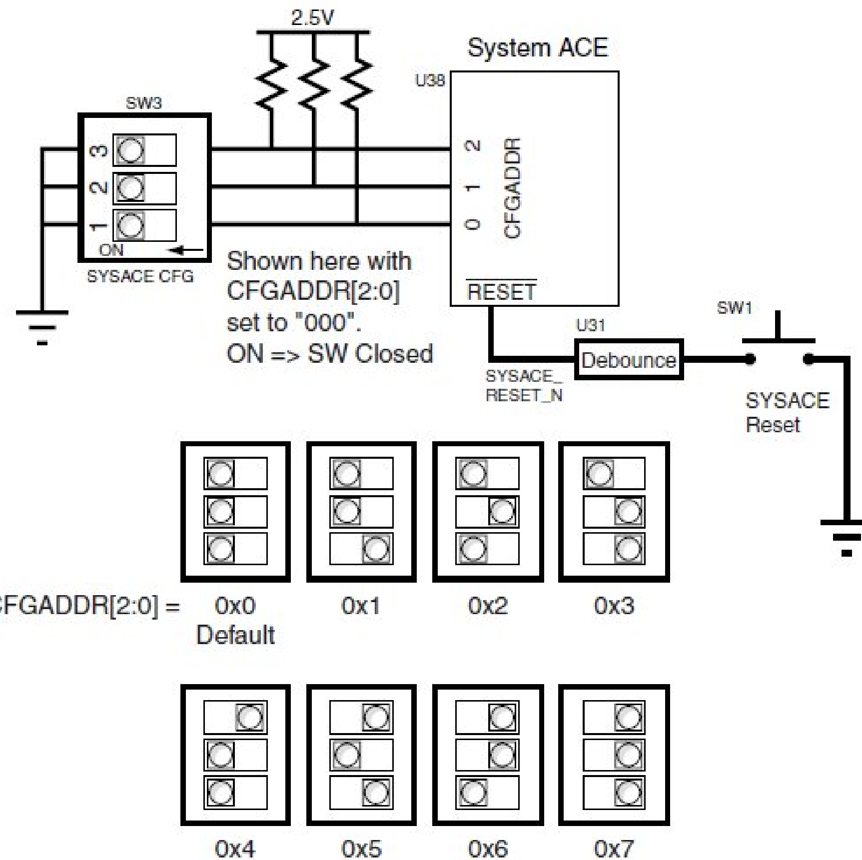


➢ Arquitectura do Virtex II Pro XC2VP30

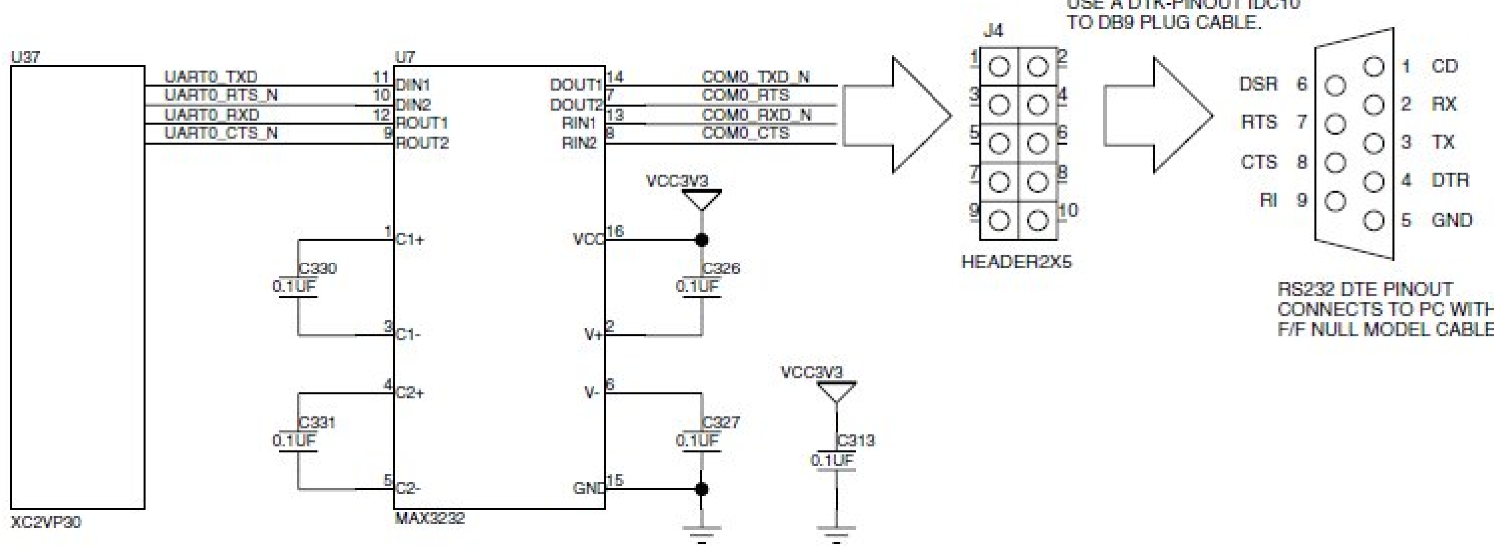


- **SW3 – Detalhes do SysACE CFG *Switch* [43]**

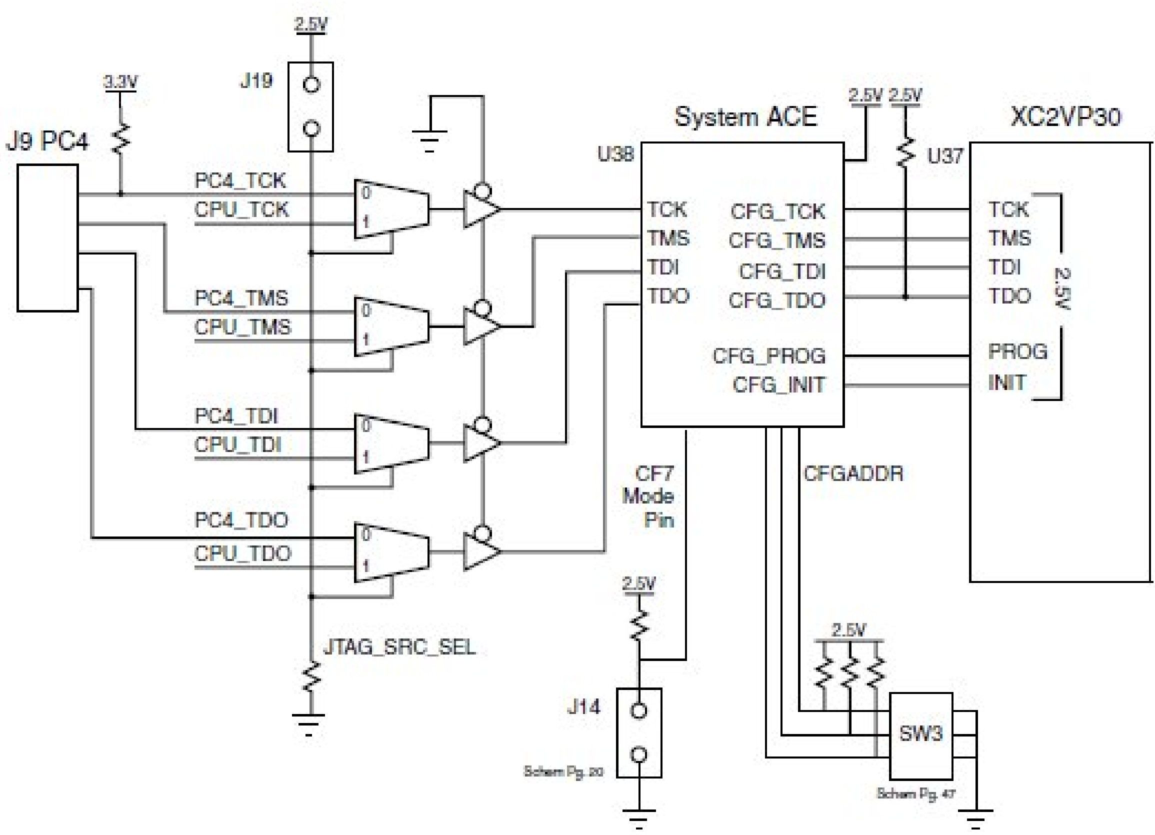
SW3 = 0 0 0 (default)



➢ **Ligação XC2VP30 FPGA e *Serial Port FPGA UART* [43]**

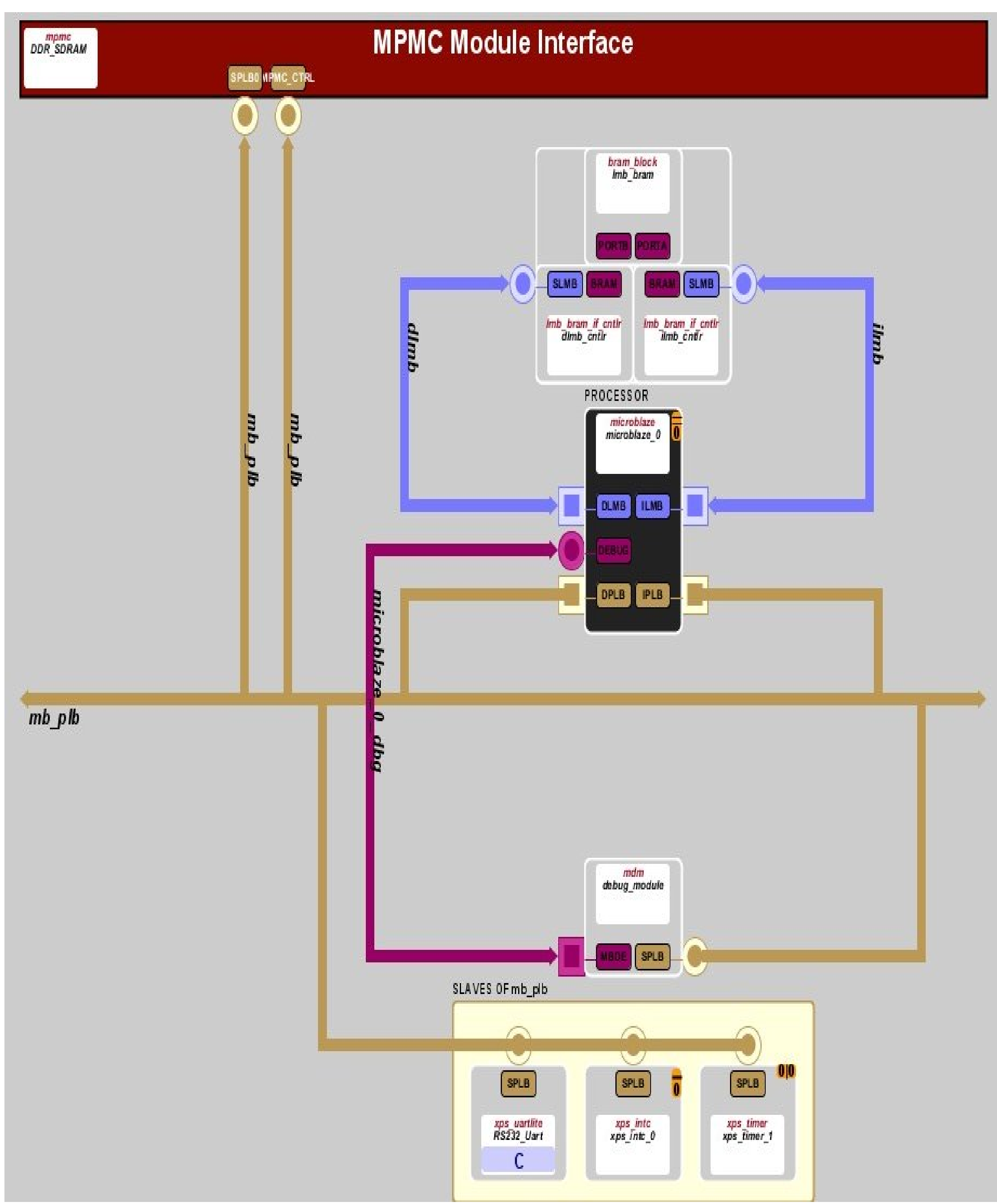


- **Ligações do JTAG para o XC2VP30 e *System ACE CF Controller***

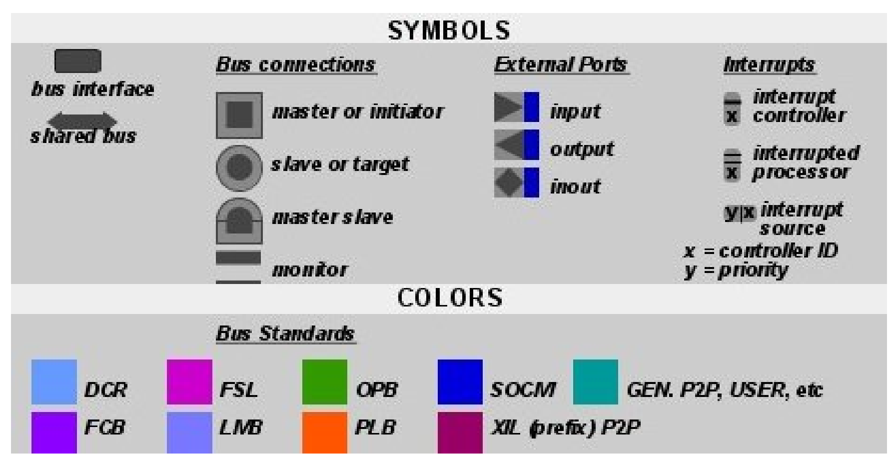


ANEXO C – Relatório do Sistema Implementado

- **Diagrama de blocos do *MicroBlaze***

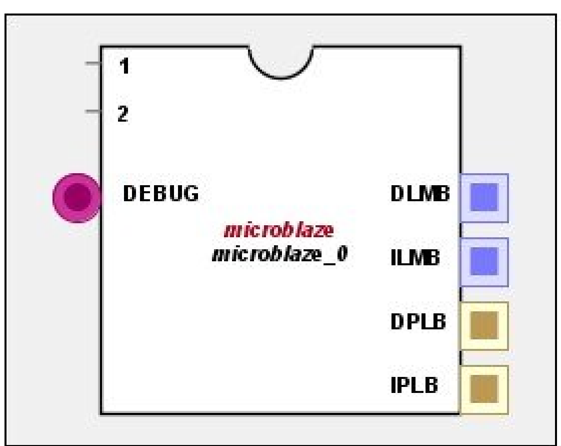


Legenda do Diagrama de blocos do *MicroBlaze*



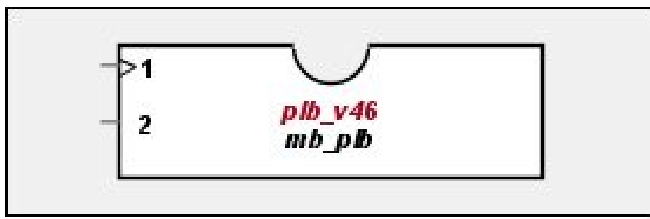
➢ **Microprocessador *MicroBlaze***

Tipo de Recurso	Nº disponível	Nº utilizado	% utilizada
Slices	13696	1766	12,89
Slice Flip Flops	27392	1972	7,20
4 input LUTs	27392	2880	10,51
MULT18X18s	136	7	5,15



- ***Bus***
  - **mb_plb**

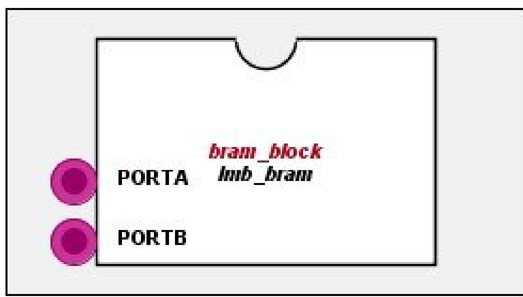
Tipo de Recurso	Nº disponível	Nº utilizado	% utilizada
Slices	13696	301	2,20
Slice Flip Flops	27392	162	0,59
4 input LUTs	27392	525	1,92



➢ **Memória**

- **lmb_bram**

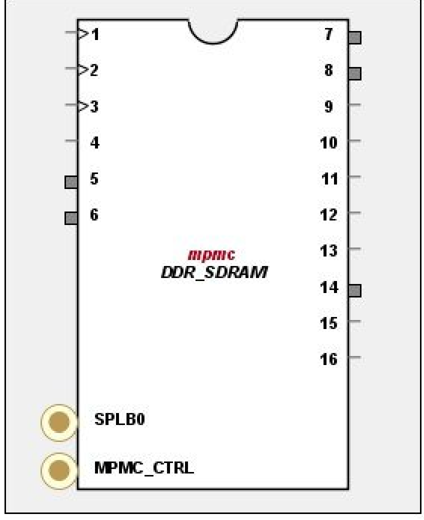
Tipo de Recurso	Nº disponível	Nº utilizado	% utilizada
Slices	13696	0	0,00
BRAMs	136	32	23,53



- **DDR_SDRAM**

Tipo de Recurso	Nº disponível	Nº utilizado	% utilizada
Slices	13696	1863	13,60
Slice Flip Flops	27392	2941	10,74
4 input LUTs	27392	1182	4,32
BRAMs	136	9	6,62

Os parâmetros do periférico *DDR_SDRAM* foram ligados ao *MicroBlaze* através do barramento PLB (mencionado na secção 4.1.2). As alterações de configuração poderão ser feitas no ficheiro *system.mhs*.



- **Tabela da frequência máxima dos módulos do sistema embebido**

MODULE	CLK Port	MAX FREQ
microblaze_0	DCACHE_FSL_OUT_CLK	120.465MHz
microblaze_0	DBG_CLK	120.465MHz
microblaze_0	DBG_UPDATE	120.465MHz
debug_module	debug_module/update	139.237MHz
debug_module	SPLB_Clk	139.237MHz
debug_module	debug_module/drck_i	139.237MHz
DDR_SDRAM	MPMC_Clk0	158.983MHz
DDR_SDRAM	SPLB0_Clk	158.983MHz
DDR_SDRAM	MPMC_CTRL_Clk	158.983MHz
DDR_SDRAM	MPMC_Clk90	158.983MHz
DDR_SDRAM	MPMC_Clk_Mem	158.983MHz
xps_timer_1	SPLB_Clk	180.963MHz
RS232_Uart	SPLB_Clk	192.938MHz
mb_plb	PLB_Clk	241.138MHz
proc_sys_reset_0	Slowest_sync_clk	248.016MHz
xps_intc_0	SPLB_Clk	265.604MHz
ilmb	LMB_Clk	303.674MHz
dlmb	LMB_Clk	303.674MHz
clock_generator_0	CLKIN	303.674MHz

➢ Parâmetros de periféricos

- *RS232-Uart*

```
BEGIN xps_uartlite
      PARAMETER INSTANCE = RS232_Uart
      PARAMETER HW_VER = 1.00.a
      PARAMETER C_BAUDRATE = 9600
      PARAMETER C_DATA_BITS = 8
      PARAMETER C_ODD_PARITY = 0
      PARAMETER C_USE_PARITY = 0
      PARAMETER C_SPLB_CLK_FREQ_HZ = 100000000
      PARAMETER C_BASEADDR = 0x84000000
      PARAMETER C_HIGHADDR = 0x8400ffff
      BUS_INTERFACE SPLB = mb_plb
      PORT RX = fpga_0_RS232_Uart_RX
      PORT TX = fpga_0_RS232_Uart_TX
END
```

- *DDR_SDRAM*

```
BEGIN mpmc
      PARAMETER INSTANCE = DDR_SDRAM
      PARAMETER HW_VER = 4.03.a
      PARAMETER C_MPMC_BASEADDR = 0x90000000
      PARAMETER C_MPMC_HIGHADDR = 0x9fffffff
      PARAMETER C_MPMC_CTRL_BASEADDR = 0x84800000
      PARAMETER C_MPMC_CTRL_HIGHADDR = 0x8480ffff
      BUS_INTERFACE SPLB0 = mb_plb
      BUS_INTERFACE MPMC_CTRL = mb_plb
END
```

- *xps_timer_1*

```
BEGIN xps_timer
      PARAMETER INSTANCE = xps_timer_1
      PARAMETER HW_VER = 1.00.a
      PARAMETER C_COUNT_WIDTH = 32
      PARAMETER C_ONE_TIMER_ONLY = 1
      PARAMETER C_BASEADDR = 0x83c00000
      PARAMETER C_HIGHADDR = 0x83c0ffff
      BUS_INTERFACE SPLB = mb_plb
END
```

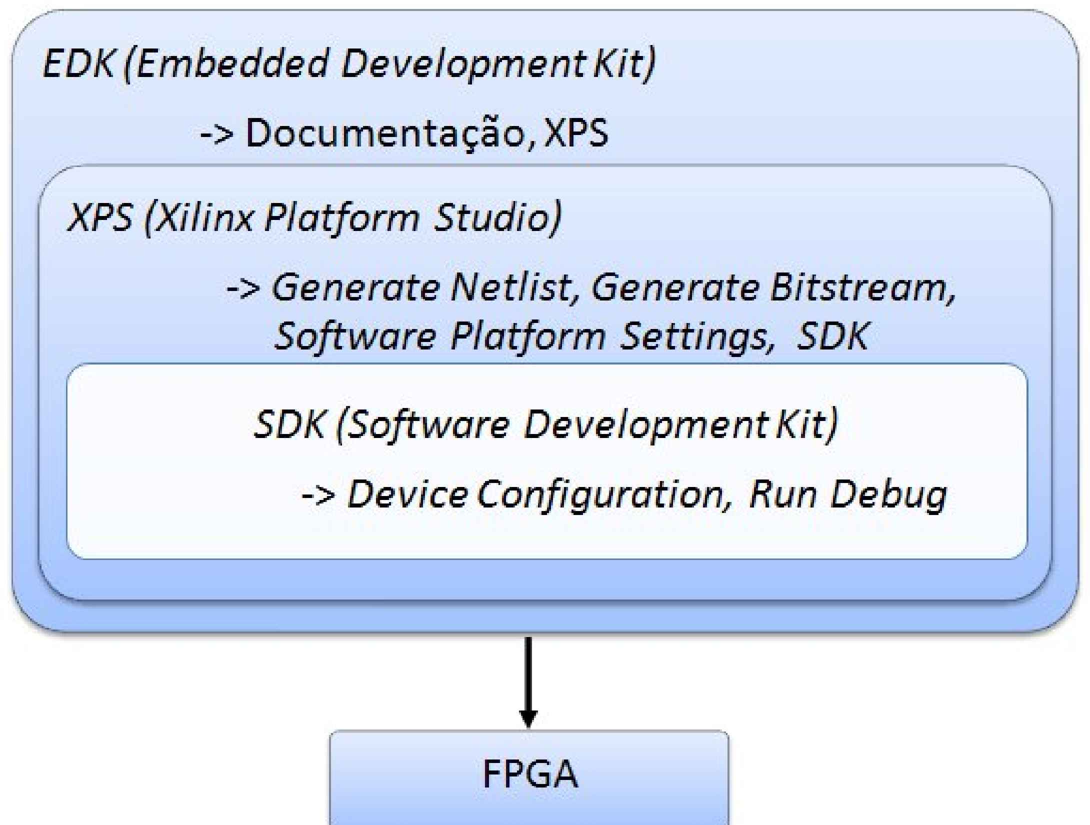
➢ Parâmetros do *Xilkernel*

- *Xilkernel*

```
BEGIN OS
      PARAMETER OS_NAME = xilkernel
      PARAMETER OS_VER = 4.00.a
      PARAMETER PROC_INSTANCE = microblaze_0
      PARAMETER sysintc_spec = xps_intc_0
      PARAMETER stdout = RS232_Uart
      PARAMETER stdin = RS232_Uart
      PARAMETER config_bufmalloc = true
      PARAMETER config_time = true
      PARAMETER systmr_dev = xps_timer_1
      PARAMETER mem_table = ((4,30),(8,20))
END
```

- **Placa ML310 a funcionar**

ANEXO D – Ferramentas da Xilinx



ANEXO E – Código de Implementação da Rede Neuronal

- Programa SDK:

- **neuralnetwork.c**

```
#include "xbasic_types.h"
#include "xparameters.h"
#include "xtmrctr_l.h"
#include "neuralnetwork.h"

int main()
{
    network_loaded = FALSE;

    if(Menu_Principal()==RUN_SUCCESS)
      {
            DeleteNetwork(&network);
            return RUN_SUCCESS;
      }
      DeleteNetwork(&network);
      return RUN_FAILURE;
}

void Menu_Load_network()
{
      XTmrCtr_mEnable(XPAR_TMRCTR_0_BASEADDR,1);
      Xuint32 tic = XTmrCtr_mGetTimerCounterReg(XPAR_TMRCTR_0_BASEADDR,1);

      xil_printf("\r\n\t- A carregar rede...\r\n");
      network_loaded = FALSE;
      if(LoadNetwork(&network)==load_OK)
      {
            xil_printf("\r\n\t- Rede carregada.\r\n");
            network_loaded = TRUE;
        }
}

void im_input_network()
{
   if(network_loaded==TRUE)
     {
      InputsNetwork(&network);
            getchar();
            getchar();
      }
      else
      {
            xil_printf("Nao existe rede na memoria!!\n\n");
      }
}

int Menu_Principal()
{
      char opcMenu;

      while(1)
```

```
	{
		xil_printf("\r\n\t*** REDE  NEURONAL ***\r\n");
		xil_printf("\r\n\t         5-8-1 \r\n");
		Menu_Load_network();
		if(Menu_Implement_Network()==RUN_FAILURE)
			return RUN_FAILURE;
		if(opcMenu=='3')
			return RUN_SUCCESS;
	}
}

int Menu_Implement_Network()
{
	char opcMenu2;

	while(1)
	{
	    xil_printf("\r\n\t- A implementar rede...\r\n");
		if(network_loaded==FALSE)
			{xil_printf("\r\nNao existe rede carregada!!! \r\n");
			}
		im_input_network();
		return RUN_SUCCESS;
	}
}
```

- **inputs.c**

```
#include "inputs.h"

int CriaListInputs(LIST_INPUTS *list,int b_flag)
{
	if((b_flag&(b_insert_end|b_insert_begin|b_insert_ordem_c|b_insert_ord
	em_d))==0)
	{
		list->b_save_flag = b_insert_end;
	}
	else
		list->b_save_flag = b_flag;
	list->first  = NULL;
	return 0;
}

int DeleteListInputs(LIST_INPUTS *list)
{
	while(!ListEmpty_Inp(list))
	{
		DeleteInputListBegin(list);
	}
	list->first = NULL;
	list->b_save_flag=b_insert_end;
	return 0;
}

int ListEmpty_Inp(LIST_INPUTS *list)
{
```

```
	if (list->first==NULL)
		return EMPTY;
	else
		return NOEMPTY;
}

int input_sizeList(LIST_INPUTS *list)
{
	noINPUT *no_tmp=NULL;
	int tam=0;
	if (ListEmpty_Inp(list))
		return tam;
	else
	{
		no_tmp=list->first;
		tam=1;

		while(no_tmp->next!=NULL)
		{
			no_tmp=no_tmp->next;
			tam++;
		}
	}
	return tam;
}

noINPUT* InsertInputList(LIST_INPUTS *list, int tag ,float valor)
{
	noINPUT *no_new = NULL;
	noINPUT *no_tmp = NULL;
	noINPUT *no_ant = NULL;

	if((no_new=(noINPUT *)malloc(sizeof(noINPUT)))==NULL)
		return NULL;
	no_new->tag = tag;
	no_new->dValor = valor;
	no_new->next  = NULL;

	if( ListEmpty_Inp(list) )
	{
		list->first = no_new;
	}
	else
	{
		if(!(list->b_save_flag&b_insert_ordem_c))
		{
			no_tmp = list->first;
			while(no_tmp->next!=NULL)
			{
				no_tmp = no_tmp->next;
			}
			no_tmp->next = no_new;
		}
		else
		{
			no_tmp = list->first;
			no_ant = NULL;
			while((no_tmp!=NULL)&&(tag>no_tmp->tag))
			{
				no_ant = no_tmp;
```

```
                        no_tmp = no_tmp->next;
                    }
                    if(no_ant==NULL)
                    {
                        no_new->next = no_tmp;
                        list->first = no_new;
                    }
                    else
                    {
                        no_new->next = no_tmp;
                        no_ant->next = no_new;
                    }
                }
        }
        return no_new;
}

int DeleteInputIndexList(LIST_INPUTS *list,int n)
{
        int i;
        noINPUT *no_tmp = NULL;
        noINPUT *no_ant = NULL;
        if ( ListEmpty_Inp(list) || (input_sizeList(list)<n)||(n<=0) )
        {
                return 0;
        }
        else
        {
                no_ant = NULL;
                no_tmp = list->first;
                for(i=1 ;(i<n);i++)
                {
                        no_ant=no_tmp;
                        no_tmp=no_tmp->next;
                }
                if (no_ant==NULL)
                {
                        list->first=no_tmp->next;
                }
                else
                {
                        no_ant->next=no_tmp->next;
                }
                free(no_tmp);
        }
        return 1;
}

int DeleteInputListEnd(LIST_INPUTS *list)
{
        noINPUT *no_tmp=NULL;
        noINPUT *no_ant=NULL;
        if (!ListEmpty_Inp(list))
        {
                no_tmp = list->first;
                no_ant=NULL;
                while(no_tmp->next!=NULL)
                {
                        no_ant = no_tmp;
                        no_tmp = no_tmp->next;
```

```
        }
        if(no_ant==NULL)
        {
            free(no_tmp);
            list->first=NULL;
        }
        else
        {
            free(no_tmp);
            no_ant->next=NULL;
        }
    }
    return 0;
}

int DeleteInputListBegin(LIST_INPUTS *list)
{
    noINPUT *no_ant=NULL;
    if (!ListEmpty_Inp(list))
    {
        no_ant = list->first;
        list->first = no_ant->next;
        free(no_ant);
    }
    return 0;
}

int ProcuraInputList(LIST_INPUTS *list, int tag)
{
    int i;
    noINPUT *no_tmp = NULL;
    if (!ListEmpty_Inp(list))
    {
        no_tmp = list->first;
        for(i=1; (no_tmp!=NULL); i++)
        {
            if(no_tmp->tag==tag)
                return i;
            no_tmp=no_tmp->next;
        }
    }
    return 0;
}

noINPUT* InInputIndexList(LIST_INPUTS *list,int n)
{
    int i;
    noINPUT *no_tmp = NULL;
    if (!( ListEmpty_Inp(list)||(input_sizeList(list)<n)||(n<=0) ))
    {
        no_tmp = list->first;
        for(i=1 ;((i<n)&&(no_tmp->next!=NULL));i++)
            no_tmp=no_tmp->next;
    }

    return no_tmp;
}

float ValorInputIndexList(LIST_INPUTS *list,int n)
{
```

```
	noINPUT *no_tmp = NULL;
	no_tmp = InInputIndexList(list,n);
	if(no_tmp==NULL)
		return (0.00);
	else
		return (no_tmp->dValor);
}
```

- **connection.c**

```
#include "connection.h"

int CriaList_connect(list_connect *list,int b_flag)
{
	if((b_flag&(b_insert_end|b_insert_begin|b_insert_ordem_c|b_insert_ord
	em_d))==0)
		list->b_save_flag = b_insert_end;
	else
		list->b_save_flag = b_flag;
	list->first  = NULL;
	return 0;
}

int DeleteList_connect(list_connect *list)
{
	while(!connect_ListEmpty(list))
	{
		DeleteConnect_ListBegin(list);
	}
	list->first = NULL;
	list->b_save_flag=b_insert_end;
	return 0;
}

int connect_ListEmpty(list_connect *list)
{
	if (list->first==NULL)
		return EMPTY;
	else
		return NOEMPTY;
}

int connect_sizeList(list_connect *list)
{
	noConnect *no_tmp=NULL;
	int tam=0;
	if (connect_ListEmpty(list))
		return tam;
	else
	{
		no_tmp=list->first;
		tam=1;
		while(no_tmp->next!=NULL)
		{
			no_tmp=no_tmp->next;
			tam++;
```

```
		}
	}
	return tam;
}

noConnect* connect_InsertList(list_connect *list,float weight, int tagA, 
int tagS, int flagS)
{
	noConnect *no_new = NULL;

	if((no_new=(noConnect *)malloc(sizeof(noConnect)))==NULL)
		return NULL;
	no_new->dWeight = weight;
	no_new->tagInputAntec = tagA;
	no_new->tagSuc = tagS;
	no_new->flagSuc = flagS;
	no_new->next  = NULL;
	if (connect_ListEmpty(list))
	{
		list->first = no_new;
	}
	else
	{
		no_new->next = list->first;
		list->first = no_new;
	}
	return no_new;
}

int DeleteConnect_IndexList(list_connect *list,int n)
{
	int i;
	noConnect *no_tmp = NULL;
	noConnect *no_ant = NULL;
	if ( connect_ListEmpty(list) || (connect_sizeList(list)<n)||(n<=0) )
	{
		return 0;
	}
	else
	{
		no_ant = NULL;
		no_tmp = list->first;
		for(i=1 ;(i<n);i++)
		{
			no_ant=no_tmp;
			no_tmp=no_tmp->next;
		}
		if (no_ant==NULL)
		{
			list->first=no_tmp->next;
		}
		else
		{
			no_ant->next=no_tmp->next;
		}
		free(no_tmp);
	}
	return 1;
}
```

```
int DeleteConnect_ListEnd(list_connect *list)
{
	noConnect *no_tmp=NULL;
	noConnect *no_ant=NULL;
	if (!connect_ListEmpty(list))
	{
		no_tmp = list->first;
		no_ant=NULL;

		while(no_tmp->next!=NULL)
		{
			no_ant = no_tmp;
			no_tmp = no_tmp->next;
		}

		if(no_ant==NULL)
		{
			free(no_tmp);
			list->first=NULL;
		}

		else
		{
			free(no_tmp);
			no_ant->next=NULL;
		}
	}
	return 0;
}

int DeleteConnect_ListBegin(list_connect *list)
{
	noConnect *no_ant=NULL;
	if (!connect_ListEmpty(list))
	{
		no_ant = list->first;
		list->first = no_ant->next;
		free(no_ant);
	}
	return 0;
}

int ProcuraConnect_ListSuces(list_connect *list, int tag, int fOutput)
{
	int i;
	noConnect *no_tmp = NULL;
	if (!connect_ListEmpty(list))
	{
		no_tmp = list->first;

		for(i=1; (no_tmp!=NULL); i++)
		{
			if(fOutput==not_priority)
			{
				if(no_tmp->tagSuc==tag)
					return i;
			}
			else
			{
```

```
                        if((no_tmp->flagSuc==fOutput)&&(no_tmp-
>tagSuc==tag))
                              return i;
                  }
                  no_tmp=no_tmp->next;
            }
      }
      return 0;
}

int ProcuraConnect_ListAntec(list_connect *list, int tagAntec)
{
      int i;
      noConnect *no_tmp = NULL;
      if (!connect_ListEmpty(list))
      {
            no_tmp = list->first;

            for(i=1; (no_tmp!=NULL); i++)
            {
                  if(no_tmp->tagInputAntec==tagAntec)
                        return i;
                  no_tmp=no_tmp->next;
            }
      }
      return 0;
}

noConnect* connectIndexList(list_connect *list,int n)
{
      int i;
      noConnect *no_tmp = NULL;
      if (!( connect_ListEmpty(list)||(connect_sizeList(list)<n)||(n<=0) ))
      {
            no_tmp = list->first;

            for(i=1 ;((i<n)&&(no_tmp->next!=NULL));i++)
                  no_tmp=no_tmp->next;
      }
      return no_tmp;
}
```

- **synapses.c**

```
#include "synapses.h"

int CriaListSynapses(LIST_SYNAPSES *list,int b_flag)
{
      if((b_flag&(b_insert_end|b_insert_begin|b_insert_ordem_c|b_insert_ord
      em_d))==0)
            list->b_save_flag = b_insert_end;
      else
            list->b_save_flag = b_flag;
      list->first  = NULL;
      return 0;
}
```

```
int DeleteListSynapses(LIST_SYNAPSES *list)
{
	while(!ListEmpty_Syn(list))
	{
		DeleteSynapseListBegin(list);
	}
	list->first = NULL;
	list->b_save_flag=b_insert_end;
	return 0;
}

int ListEmpty_Syn(LIST_SYNAPSES *list)
{
	if (list->first==NULL)
		return EMPTY;
	else
		return NOEMPTY;
}

int synap_sizeList(LIST_SYNAPSES *list)
{
	noSYNAPSE *no_tmp=NULL;
	int tam;
	if (ListEmpty_Syn(list))
		return tam;
	else
	{
		no_tmp=list->first;
		tam=1;
		while(no_tmp->next!=NULL)
		{
			no_tmp=no_tmp->next;
			tam++;
		}
	}
	return tam;
}

noSYNAPSE* InsertSynapseList(LIST_SYNAPSES *list,float weight, struct
strVerticeNeuron* nEndereco)
{
	noSYNAPSE *no_new=NULL;

	if((no_new=(noSYNAPSE *)malloc(sizeof(noSYNAPSE)))==NULL)
		return NULL;
	no_new->weight = weight;
	no_new->pAntec = nEndereco;
	no_new->next  = NULL;
	if (ListEmpty_Syn(list))
	{
		list->first = no_new;
	}
	else
	{
		no_new->next = list->first;
		list->first = no_new;
	}
	return no_new;
}
```

```
int DeleteSynapseIndexList(LIST_SYNAPSES *list,int n)
{
      int i;
      noSYNAPSE *no_tmp = NULL;
      noSYNAPSE *no_ant = NULL;
      if ( ListEmpty_Syn(list) || (synap_sizeList(list)<n)||(n<=0) )
      {
            return 0;
      }
      else
      {
            no_ant = NULL;
            no_tmp = list->first;
            for(i=1 ;(i<n);i++)
            {
                  no_ant=no_tmp;
                  no_tmp=no_tmp->next;
            }
            if (no_ant==NULL)
            {
                  list->first=no_tmp->next;
            }
            else
            {
                  no_ant->next=no_tmp->next;
            }
            free(no_tmp);
      }
      return 1;
}

int DeleteSynapseListEnd(LIST_SYNAPSES *list)
{
      noSYNAPSE *no_tmp=NULL;
      noSYNAPSE *no_ant=NULL;
      if (!ListEmpty_Syn(list))
      {
            no_tmp = list->first;
            no_ant=NULL;
            while(no_tmp->next!=NULL)
            {
                  no_ant = no_tmp;
                  no_tmp = no_tmp->next;
            }
            if(no_ant==NULL)
            {
                  free(no_tmp);
                  list->first=NULL;
            }
            else
            {
                  free(no_tmp);
                  no_ant->next=NULL;
            }
      }
      return 0;
}

int DeleteSynapseListBegin(LIST_SYNAPSES *list)
```

```
{
      noSYNAPSE *no_ant=NULL;
      if (!ListEmpty_Syn(list))
      {
            no_ant = list->first;
            list->first = no_ant->next;
            free(no_ant);
      }
      return 0;
}

int ProcuraSynapseList(LIST_SYNAPSES *list,struct strVerticeNeuron
*pNeuron)
{
      int i;
      noSYNAPSE *no_tmp = NULL;
      if (!ListEmpty_Syn(list))
      {
            no_tmp = list->first;
            for(i=1; (no_tmp!=NULL); i++)
            {
                  if(no_tmp->pAntec==pNeuron)
                        return i;
                  no_tmp=no_tmp->next;
            }
      }
      return 0;
}

noSYNAPSE* SynapseIndexList(LIST_SYNAPSES *list,int n)
{
      int i;
      noSYNAPSE *no_tmp = NULL;
      if (!( ListEmpty_Syn(list)||(synap_sizeList(list)<n)||(n<=0) ))
      {
            no_tmp = list->first;
            for(i=1 ;((i<n)&&(no_tmp->next!=NULL));i++)
                  no_tmp=no_tmp->next;
      }
      return no_tmp;
}

float WeightSynapseIndexList(LIST_SYNAPSES *list,int n)
{
      noSYNAPSE *no_tmp = NULL;
      no_tmp = SynapseIndexList(list,n);
      if(no_tmp==NULL)
            return (0.00);
      else
            return (no_tmp->weight);
}

struct strVerticeNeuron *AntecSynapseIndexList(LIST_SYNAPSES *list,int n)
{
      noSYNAPSE *no_tmp = NULL;
      no_tmp = SynapseIndexList(list,n);
      if(no_tmp==NULL)
            return ((struct strVerticeNeuron *)NULL);
      else
            return (no_tmp->pAntec);
```

```
}
```

- **datat_neuron.c**

```
#include "datat_neuron.h"

int CriaGrafoNeurons(GRAFO_NEURONS *grafo,int b_flag)
{
	if((b_flag&(b_insert_end|b_insert_begin|b_insert_ordem_c|b_insert_ord
	em_d))==0)
		grafo->b_save_flag = b_insert_end;
	else
		grafo->b_save_flag = b_flag;
	grafo->first = NULL;
	return 0;
}

int DeleteGrafoNeuron(GRAFO_NEURONS *grafo)
{
	while(!GrafoEmpty(grafo))
	{
	   DeleteNeuronGrafoBegin(grafo);
	}
	grafo->first = NULL;
	grafo->b_save_flag=b_insert_end;
	return 0;
}

int GrafoEmpty(GRAFO_NEURONS *grafo)
{
	if (grafo->first==NULL)
		return EMPTY;
	else
		return NOEMPTY;
}

int SizeGrafo(GRAFO_NEURONS *grafo)
{
	VerticeNeuron *verttmp=NULL;
	int tam=0;
	if (GrafoEmpty(grafo))
		return tam;
	else
	{
		verttmp=grafo->first;
		tam=1;
		while(verttmp->next!=NULL)
		{
			verttmp=verttmp->next;
			tam++;
		}
	}
	return tam;
}

VerticeNeuron* InsertNeuronGrafo(GRAFO_NEURONS *grafo, int tag,int flagOut,
double limiar)
```

```
{
      VerticeNeuron *vertnew=NULL;
      VerticeNeuron *verttmp=NULL;

      if((vertnew = (VerticeNeuron *)malloc(sizeof(VerticeNeuron)))==NULL)
            return NULL;
      flagOut &= FLAG_OUTPUT;
      vertnew->tag = tag;
      vertnew->flags = flagOut;
      vertnew->limiar_activation = limiar;
      vertnew->activation = 0.00;
      CriaListSynapses(&vertnew->synapses,b_insert_end);
      vertnew->next  = NULL;
      if (GrafoEmpty(grafo))
      {
            grafo->first = vertnew;
      }
      else
      {
            verttmp = grafo->first;
            while(verttmp->next!=NULL)
            {
                  verttmp = verttmp->next;
            }
            verttmp->next = vertnew;
      }

      return vertnew;
}

int InsertSynapseNeuronAntecGrafo(GRAFO_NEURONS *grafo,
            double weight, int tagN , int flagOutN , int tagA , int
flagOutA)
{
      VerticeNeuron *vertice = NULL;
      VerticeNeuron *verticeAnt = NULL;
      flagOutN&=FLAG_OUTPUT;
      flagOutA&=FLAG_OUTPUT;
      if((tagN==tagA)&&(flagOutN==flagOutA))
            return ERROR;
      vertice = NeuronIndexGrafo(grafo,
ProcuraNeuronGrafo(grafo,tagN,flagOutN));
      if (vertice==NULL)
            return ERROR;

      verticeAnt = NeuronIndexGrafo(grafo,
ProcuraNeuronGrafo(grafo,tagA,flagOutA));
      if (verticeAnt==NULL)
            return ERROR;
      InsertSynapseList(&vertice->synapses, weight, verticeAnt);
      return 0;
}

int DeleteNeuronIndexGrafo(GRAFO_NEURONS *grafo,int n)
{
      int i;
      VerticeNeuron *verttmp = NULL;
      VerticeNeuron *vertant = NULL;
      if ( GrafoEmpty(grafo) || (SizeGrafo(grafo)<n) || (n<=0) )
      {
```

```
            return 0;
      }
      else
      {
            vertant = NULL;
            verttmp = grafo->first;
            for(i=1 ;(i<n);i++)
            {
                  vertant = verttmp;
                  verttmp = verttmp->next;
            }
            if (vertant==NULL)
            {
                  grafo->first=verttmp->next;
            }
            else
            {
                  vertant->next=verttmp->next;
            }
            DeleteListSynapses(&verttmp->synapses);
            free(verttmp);
      }
      return 1;
}

int DeleteNeuronGrafoEnd(GRAFO_NEURONS *grafo)
{
      VerticeNeuron *verttmp=NULL;
      VerticeNeuron *vertant=NULL;
      if (!GrafoEmpty(grafo))
      {
            verttmp = grafo->first;
            vertant=NULL;
            while(verttmp->next!=NULL)
            {
                  vertant = verttmp;
                  verttmp = verttmp->next;
            }
            if(vertant==NULL)
            {
                  DeleteListSynapses(&verttmp->synapses);
                  free(verttmp);
                  grafo->first=NULL;
            }
            else
            {
                  DeleteListSynapses(&verttmp->synapses);
                  free(verttmp);
                  vertant->next=NULL;
            }
      }
      return 0;
}

int DeleteNeuronGrafoBegin(GRAFO_NEURONS *grafo)
{
      VerticeNeuron *vertant=NULL;
      if (!GrafoEmpty(grafo))
      {
            vertant = grafo->first;
```

```
            grafo->first = vertant->next;
            DeleteListSynapses(&vertant->synapses);
            free(vertant);
      }
      return 0;
}

int ProcuraNeuronGrafo(GRAFO_NEURONS *grafo, int tag, int flagOut)
{
      int i;
      VerticeNeuron *verttmp = NULL;
      flagOut =flagOut & (FLAG_OUTPUT | not_priority);
      if (!GrafoEmpty(grafo))
      {
            verttmp = grafo->first;
            for(i=1; (verttmp!=NULL); i++)
            {
                  if(flagOut==not_priority)
                  {
                        if(verttmp->tag==tag)
                              return i;
                  }
                  else
                  {
                        if((verttmp->flags&neuron_OUTPUT)==flagOut)
                              if(verttmp->tag==tag)
                                    return i;
                  }
                  verttmp = verttmp->next;
            }
      }
      return 0;
}

VerticeNeuron* NeuronIndexGrafo(GRAFO_NEURONS *grafo,int n)
{
      int i;
      VerticeNeuron *verttmp = NULL;
      if (!( GrafoEmpty(grafo)||(SizeGrafo(grafo)<n)||(n<=0) ))
      {
            verttmp = grafo->first;
            for(i=1 ;((i<n)&&(verttmp->next!=NULL));i++)
                  verttmp = verttmp->next;
      }
      return verttmp;
}

int NeuronAntec(GRAFO_NEURONS *grafo, int tag,int flagOut)
{
      VerticeNeuron *vertice = NULL;
      flagOut&=FLAG_OUTPUT;
      vertice = NeuronIndexGrafo(grafo,
ProcuraNeuronGrafo(grafo,tag,flagOut));
      if (vertice==NULL)
            return ERROR;
      return(!ListEmpty_Syn(&vertice->synapses));
}

int CleanFlagCalc(GRAFO_NEURONS *grafo)
{
```

```
	int i;
	VerticeNeuron *verttmp = NULL;

	if (!GrafoEmpty(grafo))
	{
		verttmp = grafo->first;
		for(i=1; (verttmp!=NULL); i++)
		{
			verttmp->flags = (~CALCULADO) & verttmp->flags;
			verttmp = verttmp->next;
		}
	}
	return 0;
}
```

datat_neurnet.c

```
#include "xbasic_types.h"
#include "xparameters.h"
#include "xuartlite.h"
#include "xstatus.h"
#include "xtmrctr_l.h"
#include "xilmfs.h"
#include "math.h"
#include "stdio.h"
#include "datat_neurnet.h"
#include "inputs.h"

/*XUartLite UartLite;
XStatus Status;
Xuint8  SendBuffer[TEST_BUFFER_SIZE];
Xuint8  RecvBuffer[TEST_BUFFER_SIZE];

XStatus UartInit( void )
{
    Status = XUartLite_Initialize(&UartLite, XPAR_UARTLITE_0_DEVICE_ID);
    STATUS_CHECK(Status);
    return XST_SUCCESS;
}*/

float entr1           = 0.00;
//float entr2         = atof(RecvBuffer);  //vlr ynew que recebe do Matlab
float entr2           = 0.00;
float entr3           = 0.00;
float entr4           = 0.00;
float entr5           = 1.00;              //vlr que corresponde o Bias
float result          = 0.00;
int size_index        = 4348;
struct mfs_file_block efs[200];
char buf[512];
char buf2[512];
int buflen;
int tmp;
int tmp2;
int num_iter;
Xuint32 tic;
```

```
float sigmbip(float x)
{
   return (1 - (2/(1+exp(2*x))));
 }

float invsigmbip(float y)
{
    return ((log((-2/(y-1))-1))/2);
}

int criaNetwork(NEURAL_NETWORK *network)
{
      network->cria_input  = 0;
      network->cria_neuron = 0;
      network->cria_output = 0;

      CriaListInputs(&network->inputs,b_insert_end);
      CriaList_connect(&network->connections,0);
      CriaGrafoNeurons(&network->neurons,b_insert_end);
      return 0;
}

int DeleteNetwork(NEURAL_NETWORK *network)
{
      DeleteListInputs(&network->inputs);
      DeleteList_connect(&network->connections);
      DeleteGrafoNeuron(&network->neurons);
      network->cria_input  = 0;
      network->cria_neuron = 0;
      network->cria_output = 0;
      return 0;
}

int CalcNetwork(NEURAL_NETWORK *network)
{
    return CleanFlagCalc(&network->neurons);
}

float AddInputsNeuron(NEURAL_NETWORK *network,VerticeNeuron *neuron)
{
      int i;
      int tam;
      noConnect *no_cnt=NULL;
      float total = 0.00;

      tam=connect_sizeList(&network->connections);
      for(i=1;i<=tam;i++)
      {
          if((no_cnt = connectIndexList(&network->connections,i))==NULL)
                  return total;
                  if((no_cnt->tagSuc==neuron->tag) &&
                    (no_cnt->flagSuc==(neuron->flags & FLAG_OUTPUT)))
                  {
                        total+=no_cnt->dWeight *
                        ValorInputIndexList(&network->inputs,
                        ProcuraInputList(&network->inputs,
                        no_cnt->tagInputAntec));
                  }
      }
```

```
	return total;
}

float AddAntecNeuron(NEURAL_NETWORK *network,VerticeNeuron *neuron)
{
	int i, tam;
	noSYNAPSE *no_snps=NULL;
	float total = 0.00;

	tam=synap_sizeList(&neuron->synapses);
	for(i=1;i<=tam;i++)
	{
		if((no_snps = SynapseIndexList(&neuron->synapses,i))==NULL)
			return total;
		total+=no_snps->weight *
			  CalcActivNeuron(network,no_snps->pAntec);
	}
	return total;
}

float CalcActivNeuron(NEURAL_NETWORK *network,VerticeNeuron *neuron)
{
	if(!(neuron->flags&CALCULADO))
	{
		neuron->activation = sigmbip(AddInputsNeuron(network,neuron) +
		                     AddAntecNeuron(network,neuron) -
		                     neuron->limiar_activation);

		neuron->flags |= CALCULADO;
	}
	return (neuron->activation);
}

float ReadFile(int index)
{
	FILE *fdr;
	if((fdr = mfs_file_open("U1s.txt", MFS_MODE_READ))== NULL)
	{
		 xil_printf("Erro abrir ficheiro");
	}
	tmp = mfs_file_read(fdr, &(buf[0]), 512);

	int flag=1;
	char linha[35];
	fgets(linha,30,fdr);
	while (flag!=index )
	{
		 fgets(linha,30,fdr);
		flag++;
	}
	tmp = mfs_file_close(fdr);
	float  nro = atof(linha);

	return nro;
}

int PrintOutputsNetwork(NEURAL_NETWORK *network)
{
	int i;
	int tam;
```

```
	float  weight_snpbias  = 0.239985073830717;

	VerticeNeuron *neuron = NULL;

	tam=SizeGrafo(&network->neurons);

	for(i=1;i<=tam;i++)
	{
		if((neuron = NeuronIndexGrafo(&network->neurons,i))==NULL)
			return ERROR;
		if((neuron->flags&FLAG_OUTPUT)==neuron_OUTPUT)
		{
			float inv = invsigmbip(neuron->activation);
			result = inv + (weight_snpbias*1);

			printf("\r\n\n\tSaida da Rede Neuronal ->
			%.10f\r\n",result);
			fprintf(fdw,"%.9f\n",result);
		}
	}
	entr4 = entr3;
	entr3 = ReadFile(index);
	entr2 = entr1;
	entr1 = result;

	return 0;
}

int CalcOutputsNetwork(NEURAL_NETWORK *network)
{
	int i;
	int tam;
	VerticeNeuron *neuron = NULL;

	tam=SizeGrafo(&network->neurons);

	if((fdw = mfs_file_open("Yfpga1.txt", MFS_MODE_WRITE))== 0)
	{
		xil_printf("Erro");
	}
	tmp2 = mfs_file_write(fdw, buf2, strlen(buf2));

	for(i=1;i<=tam;i++)
	{
		if((neuron = NeuronIndexGrafo(&network->neurons,i))==NULL)
			return ERROR;
		if((neuron->flags&FLAG_OUTPUT)==neuron_OUTPUT)
		{
			CalcActivNeuron(network,neuron);
		}
	}
	PrintOutputsNetwork(network);
	return 0;
}

int InputsNetwork(NEURAL_NETWORK *network)
{
	/*XStatus Status;

	Status = UartInit();
```

```
	if (Status != XST_SUCCESS)
	{
	     print(" -- UART Initialization Error -- \n\r");
	     return(1);
	 }

	unsigned int ReceivedCount;
	ReceivedCount = 0;

	while(!ReceivedCount)
	{
	      ReceivedCount = XUartLite_Recv(&UartLite, RecvBuffer, 16);
	}*/

	int i, index, tam;

	for(index = 1; index <= size_index; index++)
	{
	  noINPUT *nodoinp = NULL;
	  tam    = input_sizeList(&network->inputs);

	  nodoinp = InInputIndexList(&network->inputs,1);
	  nodoinp->dValor = entr1;

	  nodoinp = InInputIndexList(&network->inputs,2);
	  nodoinp->dValor = entr2;

	  nodoinp = InInputIndexList(&network->inputs,3);
	  nodoinp->dValor = entr3;

	  nodoinp = InInputIndexList(&network->inputs,4);
	  nodoinp->dValor = entr4;

	  nodoinp = InInputIndexList(&network->inputs,5);
	  nodoinp->dValor = entr5;

	  CalcNetwork(network);

	  CalcOutputsNetwork(network);
    }
	tic = (XTmrCtr_mGetTimerCounterReg(XPAR_ XPS_TIMER_1_BASEADDR,0)-
	tic);
	tic = tic * 0.00000001; //nº ciclos * 10ns)
	xil_printf("\r\n\n\tTempo de processamento -> %d ms\r\n",tic);

	tmp2 = mfs_file_close(fdw);

	return(!ERROR);
}

int LoadNetwork(NEURAL_NETWORK *network)
{
	XTmrCtr_mEnable(XPAR_XPS_TIMER_1_BASEADDR,0);
	tic = XTmrCtr_mGetTimerCounterReg(XPAR_ XPS_TIMER_1_BASEADDR,0);
	int inp  =  5;    //5-3-1
	int neur =  4;    //total de neuronios
	int outp =  1;

	int i1   =  1;      //entradas
	int i2   =  2;
```

```
      int i3   =  3;
      int i4   =  4;
      int i5   =  5;

      int n1   =  1;      //neuronios na camada escondida
      int n2   =  2;
      int n3   =  3;

      int o1   =  1;     //neuronios na camada saida

      //W2i pesos sinapticos

      float weight_snp1  =  8.01754080324635;
      float weight_snp2  =  5.73480953802607;
      float weight_snp3  =  0.0000261028191972014;

      //w1i pesos conexoes
      float weight_con1  =  0.161993724781256;   //w11
      float weight_con2  = -0.06138019980564;    //w21
      float weight_con3  =  0.00123107494692857; //w31
      float weight_con4  = -0.00042848283259403; //w41
      float weight_con5  = -0.546939657614124;   //w51

      float weight_con6  =  0.252808485955575;   //w12
      float weight_con7  = -0.133269545102912;   //w22
      float weight_con8  =  0.00178220236091744; //w32
      float weight_con9  = -0.00178735855210766; //w42
      float weight_con10 =  0.783536893041092;   //w52

      float weight_con11 = -0.425370292195393;   //w13
      float weight_con12 = -0.156222631720689;   //w23
      float weight_con13 = -1.24300915065579;    //w33
      float weight_con14 = -1.11893442669203;    //w43
      float weight_con15 = -2.62912991049762;    //w53

      criaNetwork(network);
      network->cria_input = inp;
      network->cria_neuron = neur;
      network->cria_output = outp;

//==================================LIMIARES============================
    //neuronios da camada escondida + camada de saida

    if(InsertNeuronGrafo(&network->neurons, n1, neuron_NORMAL,0.00)==NULL){
             DeleteNetwork(network);
             return load_failure;
            }

    if(InsertNeuronGrafo(&network->neurons, n2, neuron_NORMAL,0.00)==NULL){
             DeleteNetwork(network);
             return load_failure;
            }

    if(InsertNeuronGrafo(&network->neurons, n3, neuron_NORMAL,0.00)==NULL){
             DeleteNetwork(network);
             return load_failure;
            }

    if(InsertNeuronGrafo(&network->neurons, o1, neuron_OUTPUT,0.00)==NULL){
              DeleteNetwork(network);
```

```
                  return load_failure;
                  }

//=====================================SINAPSES==============================
      //neuronios da camada escondida

      if(InsertSynapseNeuronAntecGrafo(&network->neurons,weight_snp1,o1,
neuron_OUTPUT,n1,neuron_NORMAL)==ERROR){
            DeleteNetwork(network);
            return load_failure;
            }

      if(InsertSynapseNeuronAntecGrafo(&network->neurons,weight_snp2,o1,
neuron_OUTPUT,n2,neuron_NORMAL)==ERROR){
            DeleteNetwork(network);
            return load_failure;
            }

    if(InsertSynapseNeuronAntecGrafo(&network->neurons,weight_snp3,o1,
neuron_OUTPUT,n3,neuron_NORMAL)==ERROR){
            DeleteNetwork(network);
            return load_failure;
            }

//======================================CONEXOES===========================
      // entrada -> neuronios camada escondida

    if(ProcuraInputList(&network->inputs, i1)==0)
            {
                  if(InsertInputList(&network->inputs, i1, 0.0)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
                  if(connect_InsertList(&network->connections,weight_con1,
i1, n1, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
              if(connect_InsertList(&network->connections,weight_con6, i1,
n2, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
            if(connect_InsertList(&network->connections,weight_con11, i1,
n3, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }

         }

    if(ProcuraInputList(&network->inputs, i2)==0)
            {
                  if(InsertInputList(&network->inputs, i2, 0.0)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
                  if(connect_InsertList(&network->connections,weight_con2,
i2, n1, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
```

```
                        return load_failure;
                        }
            if(connect_InsertList(&network->connections,weight_con7, i2,
n2, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
            if(connect_InsertList(&network->connections,weight_con12, i2,
n3, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
           }

    if(ProcuraInputList(&network->inputs, i3)==0)
            {
                  if(InsertInputList(&network->inputs, i3, 0.0)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
                  if(connect_InsertList(&network->connections,weight_con3,
i3, n1, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
            if(connect_InsertList(&network->connections,weight_con8, i3,
n2, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
            if(connect_InsertList(&network->connections,weight_con13, i3,
n3, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
          }

    if(ProcuraInputList(&network->inputs, i4)==0)
            {
                  if(InsertInputList(&network->inputs, i4, 0.0)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
                  if(connect_InsertList(&network->connections,weight_con4,
i4, n1, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
            if(connect_InsertList(&network->connections,weight_con9, i4,
n2, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
            if(connect_InsertList(&network->connections,weight_con14, i4,
n3, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
            }
```

```
      if(ProcuraInputList(&network->inputs, i5)==0)
            {
                  if(InsertInputList(&network->inputs, i5, 0.0)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
                  if(connect_InsertList(&network->connections,weight_con5,
i5, n1, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
            if(connect_InsertList(&network->connections,weight_con10, i5,
n2, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
                  if(connect_InsertList(&network->connections,weight_con15,
i5, n3, neuron_NORMAL)==NULL){
                        DeleteNetwork(network);
                        return load_failure;
                        }
            }

            return load_OK;
}
```

- Programa Matlab:

- **COM_FPGA.m**

```
clear all
clear functions
close all

%This is the reference and should be changed for different tests
load ID050300
U=u(1:5800);
Y=kiln_temp(1:5800);

UTX = U;
YTX = Y;

%filtrar as sequencias de entrada e de saida
%filtro em s
Am = [1 2];                 % Filter denominator
Bm = [2];                   % Filter numerator

%conversão para z
[Znum,Zden] = c2dm(Bm,Am,0.2,'zoh');

%testar o efeito do filtro na sequencia de dados
YTXff=filtfilt(Znum,Zden,YTX);
UTXff=filtfilt(Znum,Zden,UTX);

%duplicar o efeito do filtro
```

```
YTXf=filtfilt(Znum,Zden,YTXff);
UTXf=filtfilt(Znum,Zden,UTXff);

% Scaling - Retirar média e dividir pelo desvio padrão
meanu=mean(UTXf);
meany=mean(YTXf);
stdu=std(UTXf);
stdy=std(YTXf);
UTXfs=dtrend(UTXf);
YTXfs=dtrend(YTXf);
UTXfs=UTXfs' / stdu;
YTXfs=YTXfs' / stdy;

% Scaling - Retirar média e dividir pelo desvio padrão
meanu=mean(U);
meany=mean(Y);
stdu=std(U);
stdy=std(Y);
Us=dtrend(U);
Ys=dtrend(Y);
Us=Us' / stdu;
Ys=Ys' / stdy;

%Divisão dos dados em duas partes (para sequencia de treino e teste)
N2 = length(Us);
N1 = floor(N2*3/4);
Y1s = Ys(1:N1);
U1s = Us(1:N1);
Y2s = Ys(N1+1:N2);
U2s = Us(N1+1:N2);

% This must be changed for tests with diferent models
load For6700

% create variables for the outputs of the inverse and direct model
y=zeros(N1,1);

% Receive file (model in the FPGA)
fid = fopen ('Yfpga1.mat');
c = textscan(fid,'%f');
Yfpga = cell2mat(c);
fclose(fid);

%get a zero clock
%zero_clock = clock;
%set start time
%start_time = etime(clock,zero_clock);

t = cputime;

% Calculate the inputs for a second order forward model

for i=3:N1  % 4348

         finputs =[];
        % We are calculating first the forward model.
        % Since y is created as zeros the first sample will be zero

        finputs=[y(i-1);y(i-2);U1s(i-1);U1s(i-2)];
```

```
        % Place here the instruction to send the new y to the FPGA
        ynew = nnoutput(NetDeff,W1f,W2f,finputs);
        y(i)= ynew;

end
%get end time
%end_time = etime(clock, zero_clock);
%print the results
%total_time = end_time - start_time

total_t = cputime-t


error = y - Yfpga;
errqd = (error'*error)/length(error)

%Processo de escala inverso
Yfpga2  = Yfpga(1:4000)*stdy + meany;
Ymatlab = y(1:4000)*stdy + meany;

subplot(211)
plot(Yfpga2); hold on
plot([N1 N1],[min(Yfpga2) max(Yfpga2)],'m--'); hold off
axis([0 N1 min(Yfpga2) max(Yfpga2)])

title(' Modelo Rede Neuronal 1 ')
xlabel('N1')
ylabel(' Outputs Yfpga ')

subplot(212)
plot(Ymatlab, 'g'); hold on
plot([N1 N1],[min(Ymatlab) max(Ymatlab)],'m--'); hold off
axis([0 N1 min(Ymatlab) max(Ymatlab)])

xlabel('N1')
ylabel(' Outputs Ymatlab ')
drawnow
```