

IMPLEMENTAÇÃO DE PERCEPTRON SINGLE-LAYER EM FPGA USANDO VHDL

FPGA IMPLEMENTATION OF A SINGLE-LAYER PERCEPTRON USING VHDL

DIOGO FIM

FAESA - Graduando em Engenharia da Computação

diogofimcv@gmail.com

FERNANDO NOLASCO ABREU

FAESA - Graduando em Engenharia da Computação

fermmando_fna@hotmail.com

VICTOR MARQUES MIRANDA

Prof. Msc. Centro Universitário FAESA

victor.miranda@faesa.br

RESUMO

A crescente evolução das FPGAs tem apontado novos rumos para o desenvolvimento das Redes Neurais Artificiais (RNAs), uma vez que permitem a implementação dessas na execução de tarefas com alto grau de paralelismo, como as múltiplas operações matemáticas dos seus neurônios constituintes, o que é inviável de se reproduzir por *software* em processadores sequenciais. A quantidade de recursos a nível de *hardware* disponíveis nas FPGAs permite a implementação de vários processos simultaneamente e vem a contribuir para redução do tempo de processamento, sobretudo quando da expansão da rede projetada. Sem contar que as FPGAs se constituem excelentes alternativas em aplicações onde a execução em tempo real é crucial. Propõe-se, nesse trabalho, a implementação de uma rede do tipo *Perceptron Single-Layer* em FPGA, utilizando VHDL como forma de descrição e síntese. Como a rede será aplicada em problemas de classificação, a topologia constituída de um único neurônio se mostra suficiente. Implementações em *software*, quando do treinamento da rede ao usar o Matlab, e em *hardware*, para a sua síntese e validação, são combinadas. A FPGA pode ser vista como uma prova de conceito, ou mesmo como um produto final, a baixo custo computacional e financeiro. A

programação do *hardware*, via VHDL, permite ao usuário acrescentar aplicações ao longo do tempo, atendendo a novas necessidades e melhorias específicas para determinada função do dispositivo. Concomitantemente, destaca-se a potencialidade deste trabalho para fins didáticos, buscando fomentar projetos futuros correlacionados e trazer maior aprofundamento nos estudos da Eletrônica Digital e Microprocessada e das RNAs.

Palavras-chave: Redes Neurais Artificiais. FPGA. VHDL

ABSTRACT

The evolution of the Field-Programmable Gate Arrays (FPGAs) has pointed new methods for the development of Artificial Neural Networks (ANN), as they allow the implementation of these networks in the execution of tasks with a high degree of parallelism, such as the multiple mathematical operations of its neurons, which is impracticable to replicate by software-based implementation for sequential processors. The amount of hardware resources available in the commercialized FPGAs allows the implementation of several processes simultaneously and contributes to reduce the processing time, especially considering the network growth. Besides, FPGAs are excellent alternatives in applications where real time processing is crucial. This work aims to the FPGA implementation of a Single-Layer Perceptron, using VHDL as a form of description and synthesis. The chosen topology of a single-neuron proves to be enough, regarding the network is going to be applied to solve classification problems. Software implementations, in order to perform the network training using the Matlab, and hardware implementations, for its synthesis and validation, are combined. The FPGA may be seen as a concept proof, or even as a final product, at low computational and financial cost. The hardware programming, through VHDL, allows the user to add applications whenever he desires, meeting new needs e specific improvements to certain device function. Concomitantly, the potential of this work for didactic purposes stands out, intending to foment correlated future research projects and to bring up advanced knowledges in Digital and Microprocessed Electronics and also in ANN.

Keywords: Artificial Neural Network. FPGA. VHDL

1. INTRODUÇÃO

Com os avanços tecnológicos recorrentes, novas possibilidades surgem, assim como soluções mais elaboradas, mais inteligentes para problemas complexos e desafiadores. Entre esses avanços surgiram as Redes Neurais Artificiais (RNAs), nome este dado pelo fato de essas estruturas se constituírem e se comportarem de forma similar ao cérebro humano.

As RNAs são constituídas por unidades de processamento individuais – baseadas nos neurônios – que se comunicam através das sinapses. Elas são capazes de aprender, permitindo que uma tarefa seja realizada sem conhecimento de regras pré-definidas. A generalidade deste conceito faz com que as RNAs sejam adequadas para uma ampla gama de aplicações, não restritas ao contexto industrial e acadêmico, mas também voltadas para diversas áreas do conhecimento, como reconhecimento de padrões, como voz, biometria, imagens e caracteres; classificação, análise ou prospecção de dados; previsões do mercado financeiro; diagnóstico médico; sensoriamento remoto; dentre outras.

A primeira RNA foi concebida em 1943 por Warren McCulloch e por Walter Pitts, respectivamente neuroanatomista e cientista, época em que o objetivo principal da pesquisa foi demonstrar que a máquina poderia resolver problemas de maneira a se assimilar a um cérebro humano (KOVÁCS, 2006). Biologicamente falando, o mecanismo de produção e transporte de sinais de um neurônio a outro é um fenômeno fisiológico bem compreendido, mas como esses sistemas individuais cooperam para formarem estruturas paralelas e complexas com grande capacidade de processamento, armazenamento e transmissão de informação é um fenômeno que desperta cada vez mais a atenção.

Os trabalhos com redes neurais têm sido motivados, a princípio, pela presença nestas das capacidades de aquisição e manutenção do conhecimento e de reconhecimento, próprias do cérebro humano e que em muito diferem da maneira pela qual os computadores digitais convencionais processam as informações. O cérebro humano é um computador paralelo, não-linear e altamente complexo (KOVÁCS, 2006). Aliado a isso, outra motivação é o poderio de processamento das redes neurais, o qual reside em sua estrutura extremamente paralela, redundante, massivamente distribuída e em sua habilidade de aprender e generalizar. O ajuste de alguns parâmetros modifica a capacidade da rede em achar a melhor combinação para a solução de um determinado problema. Assim, as redes neurais representam sistemas adaptativos, auto-organizáveis, assim como cada neurônio constituinte.

De acordo com Silva, Spatti e Flauzino (2010), as RNAs passaram a ser fortemente estudadas a partir do início dos anos 90, sobretudo, em virtude de suas características acima expostas e de suas múltiplas possibilidades de aplicação em várias áreas, e apresentam ainda um enorme potencial para pesquisas.

A maioria das redes neurais existentes atualmente opera em máquinas sequenciais, devido, principalmente, ao menor custo quando comparadas às alternativas. No entanto, tais máquinas são incapazes de reproduzir uma característica inerente às redes neurais: a execução de operações em paralelo como, por exemplo, as de adição e multiplicação. Sendo assim, RNAs têm sua capacidade de processamento limitada devido à sua implementação em sistemas que utilizam *hardwares* seriais (CORIC; LATINOVIC; PAVASOVIC, 2000). Além disso, a diminuição do tempo de processamento é uma questão fundamental, por exemplo, em aplicações que necessitem de uma grande quantidade de cálculos em tempo reduzido, e já se torna tema de discussão e pesquisa no meio científico (SUGAHARA; OIDA; YOKOYAMA, 2009).

Nesse contexto, com as evoluções observadas na última década em FPGAs, tanto na redução dos preços, quanto no aumento da capacidade e redução de tempo de processamento, estas constituem uma excelente alternativa à implementação de RNAs com execução de múltiplas tarefas em paralelo ao nível de *hardware*.

Uma FPGA (acrônimo, em português, de “Arranjo de Portas Programáveis em Campo”) consiste em um circuito integrado projetado para ser configurado por um usuário consumidor ou projetista após a fabricação – daí advém o termo “programável em campo”. A possibilidade de criar um *hardware* que pode ser rapidamente customizado, a qual é uma das vantagens e características mais importantes das FPGAs, permite sintetizar um sistema em *hardware* com flexibilidade semelhante ao *software*.

Além disso, implementações em FPGAs mantêm o paralelismo característico do *hardware*, o que é bastante adequado para RNAs. Este paralelismo permite implementações de alta velocidade, alta performance e alto poder de processamento. Por se tratar de *hardware*, é possível ter, por exemplo, inúmeros cálculos rodando em paralelo e entregando os resultados num mesmo ciclo de processamento, algo completamente impossível para um *software* realizar. Além disso, como as RNAs tem a habilidade de aprender com o ambiente e se adaptar constantemente e as FPGAs possibilitam a criação de sistemas de alta performance com

flexibilidade, essas se mostram cada vez mais atrativas para a implementação das RNAs (CAVUSLU; KARAKUZU; SAHIN, 2006).

É perceptível que desde que surgiram em 1985, as FPGAs têm gerado novas possibilidades por parte dos pesquisadores e estudiosos, atraindo, cada vez mais, a atenção dos interessados em RNAs e se integrando a novas áreas do conhecimento. Ao longo dos anos, tem-se verificado um grande aumento de desempenho e integração nos componentes lógicos em FPGA permitindo implementar sistemas lógicos complexos num único *chip*. O desenvolvimento de sistemas digitais exigentes e complexos pode implicar a implementação de circuitos com milhões de portas lógicas que incluem memórias, interfaces rápidas e outros componentes de alto desempenho.

Uma das abordagens para o projeto e implementação desse nível de complexidade é a utilização de linguagens de descrição de *hardware*, no presente caso o VHDL (*Very High Speed Integrated Circuits Hardware Description Language*). Essa linguagem permite a um projetista de sistemas a descrição e síntese de circuito em uma FPGA, desenvolvendo *hardware* de forma simples e rápida, com a grande vantagem de ser possível efetuar testes, simulações e múltiplas implementações. Com isso, as fases de desenvolvimento se tornam mais rápidas e os custos que uma produção física de várias evoluções do sistema poderia ter até atingir o produto final são diminuídos.

Esse tipo de dispositivo permite a criação de arquiteturas funcionais personalizadas, dinamicamente reconfiguradas, como em Oliveira (2017), de acordo com a necessidade do projeto ou aplicação, como, por exemplo, diante da necessidade de expansão da rede neural. Em outras palavras, com FPGA, aliada à linguagem VHDL, é possível criar uma arquitetura computacional e o programa/instruções que irão controlar a arquitetura criada.

Assim, levando em consideração estes fatos motivadores, o presente trabalho objetiva implementar uma RNA do tipo *Perceptron* de camada simples (*Perceptron Single-Layer*) em *hardware* FPGA, usufruindo de sua característica de paralelismo, utilizando a linguagem VHDL. Além de especificamente: evidenciar as vantagens da utilização das FPGAs para a implementação, em *hardware*, das RNAs; evidenciar as vantagens de utilização da linguagem de descrição de *hardware*, VHDL, em projetos como esse; definir o modelo de FPGA e seu respectivo *kit* de desenvolvimento que atenda aos requisitos de projeto; projetar e desenvolver uma topologia de *Perceptron Single-Layer* e aplicá-la em problemas de classificação de cunho

interdisciplinar; traçar o tipo e o conjunto de dados para o treinamento da rede, assim como para os testes e sua validação; treinar a RNA projetada em *software*; e validar a RNA projetada em *hardware*.

Acredita-se que, em breve, as tecnologias de *hardware* reconfigurável estarão cada vez mais próximas e presentes na vida do cidadão comum. Assim como você atualiza seu Sistema Operacional do celular ou do computador, já pensou na possibilidade de atualizar seu processador? Fabricantes como a Intel já têm se posicionado neste cenário, lançando processadores da linha Intel Xeon, aplicáveis em computação de alto desempenho, que possuem FPGAs incorporadas junto ao processador. No que tange às tecnologias da Intel, por exemplo, o uso das FPGAs se encaixa na aplicação de coprocessadores, ou seja, na criação de sistemas para auxílio à execução de processos, como criptografia e visão computacional, por exemplo. Estima-se, portanto, que é apenas uma questão de tempo até que essa tecnologia seja cada vez mais incorporada em processadores de propósito geral. Sem deixar de falar de áreas como *Deep Learning*, *Machine Learning* e *Big Data*, que têm descoberto nas FPGAs fortes aliados para a aceleração de processamento dos recursos necessários.

Por fim, a importância do tema se dá também ao promover avanços na academia, na área de sistemas eletrônicos embarcados, microprocessados e de dispositivos programáveis, ao se utilizar uma plataforma reconfigurável para desempenhar funções das mais simples às mais complexas.

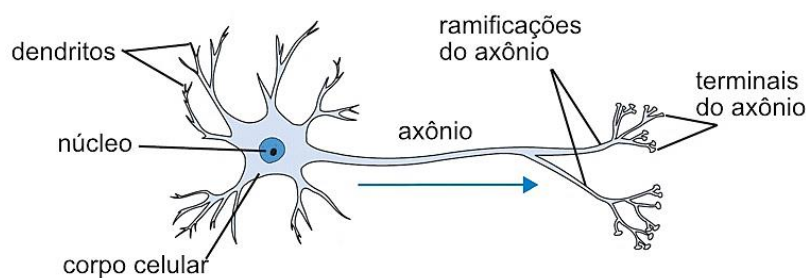
2. REVISÃO DA LITERATURA

2.1. O Perceptron

O cérebro humano é formado por um conjunto de células amplamente interconectadas, os neurônios, formando um sistema complexo com capacidade de aprendizado, chamado rede neural. Eles têm um papel essencial na determinação do funcionamento e comportamento do corpo humano e do raciocínio.

Embora haja diferentes morfologias de neurônios, todos são compostos por dendritos, corpo celular (ou soma) e axônio, como se percebe na Figura 1.

Figura 1 – Neurônio Biológico



Fonte: STANFORD CS CLASS

No neurônio biológico, os sinais são impulsos elétricos excitatórios ou inibitórios que entram através dos dendritos, passam pelo corpo celular e são transmitidos ao longo do axônio do neurônio, este que se comunica com outros neurônios através das suas sinapses. Ou seja, os pulsos de entrada são combinados e se seu potencial atinge um certo nível, o neurônio é ativado e passa o sinal através das sinapses para os próximos neurônios a ele conectados.

As Redes Neurais Artificiais (RNAs) surgiram na década de 40, mais precisamente em 1943, quando o neurofisiologista Warren McCulloch e o matemático Walter Pitts, da Universidade de Illinois, fizeram uma analogia entre as células nervosas e o processo eletrônico num artigo publicado no *Bulletin of Mathematical Biophysics*, e apresentaram o primeiro modelo de neurônios artificiais (KOVÁCS, 2006).

Desde esse marcante acontecimento, novas e mais sofisticadas propostas de modelos de neurônios têm sido feitas. Dentre elas, destaca-se o *Perceptron*, o qual foi criado em 1958 por Rosenblatt, sendo a forma mais simples de se implementar um neurônio. Ele consiste em um modelo matemático que recebe várias entradas x_1, x_2, \dots e produz uma única saída binária. Rosenblatt propôs uma regra simples para calcular a saída. Ao contrário do neurônio de McCulloch e Pitts, que só trabalhava com entradas binárias, as entradas do *Perceptron* englobavam números reais, aumentando a gama de problemas que poderiam ser resolvidos por este neurônio. Rosenblatt introduziu, ainda, pesos, w_1, w_2, \dots , também números reais, expressando o quanto cada entrada influenciaria na saída, os quais incluem ainda um sinal de *bias* (θ) que adiciona um grau de liberdade ao neurônio. Este sinal funciona como um peso adicional multiplicado a uma entrada de valor -1 . Os pesos e *bias* são ajustáveis através de treinamento, permitindo que o neurônio seja usado em vários problemas diferentes.

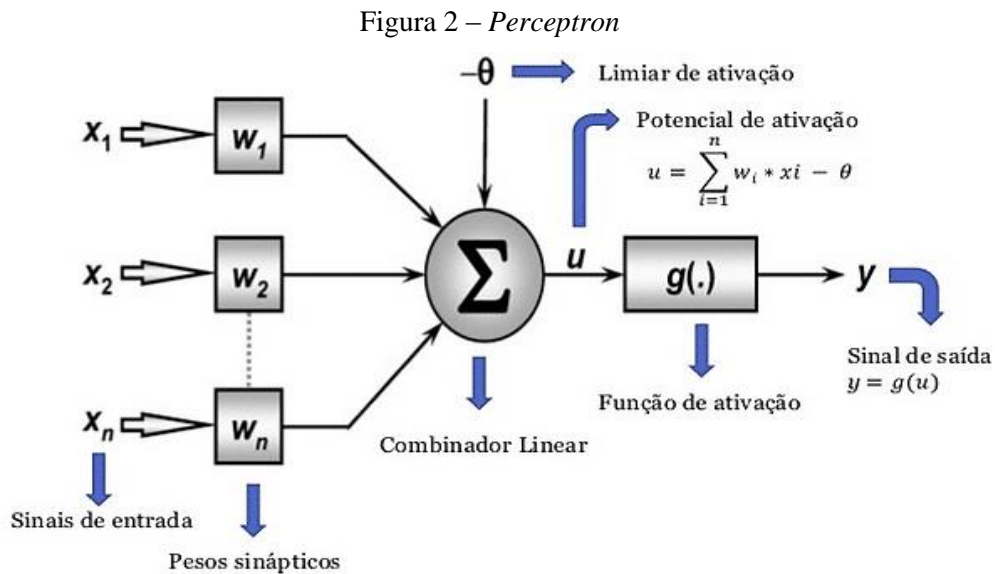
As entradas dos neurônios são multiplicadas pelos seus respectivos pesos e, em seguida, somadas ao *bias*, chegando a um valor u , como expresso na Equação (1):

$$u = \sum_{i=1}^n x_i \cdot w_i - \theta \quad (1)$$

Este valor u é, então, usado como entrada de uma função de ativação do tipo Degrau, g . Esta função gera 1 como saída se o valor de u é maior ou igual a um valor limiar (*threshold*) ou 0, caso contrário. Assim como os pesos, o *threshold* é um número real que é um parâmetro do neurônio. Colocando em termos algébricos mais precisos, tem-se:

$$y = g\left(\sum_{i=1}^n x_i \cdot w_i - \theta\right); \quad y = g(u) = \begin{cases} 1, & \text{se } u \geq 0 \\ 0, & \text{se } u < 0 \end{cases} \quad (2)$$

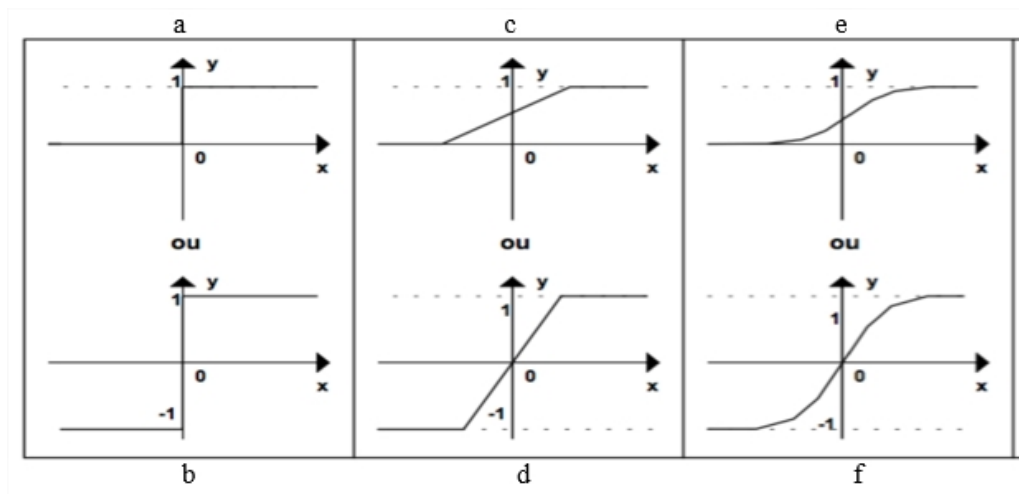
A função de ativação controla a saída do neurônio e tem como objetivo limitá-la em um dado intervalo e, se necessário, inserir uma não linearidade ao neurônio. A escolha da função de ativação pode mudar consideravelmente o comportamento do neurônio, sendo assim uma etapa importante no desenvolvimento das RNAs. A Figura 2 mostra um neurônio artificial do tipo *Perceptron*.



Fonte: PALMIERE (2016).

Uma evolução natural do *Perceptron* é permitir outras funções de ativação para classificar sistemas não-linearmente separáveis. Surgem então *Perceptrons* onde as funções de ativação podem ser de vários tipos e não só a função degrau. A Figura 3 ilustra, resumidamente, as principais funções de ativação.

Figura 3 – Funções de ativação. (a) Degrau (*Step Function*). (b) Sinal ou Degrau Simétrica (c) Rampa (*Ramp Function*). (d) Rampa Simétrica. (e) Sigmoidal Unipolar (ou Logística). (f) Função Sigmoidal Bipolar (ou Tangente Hiperbólica)



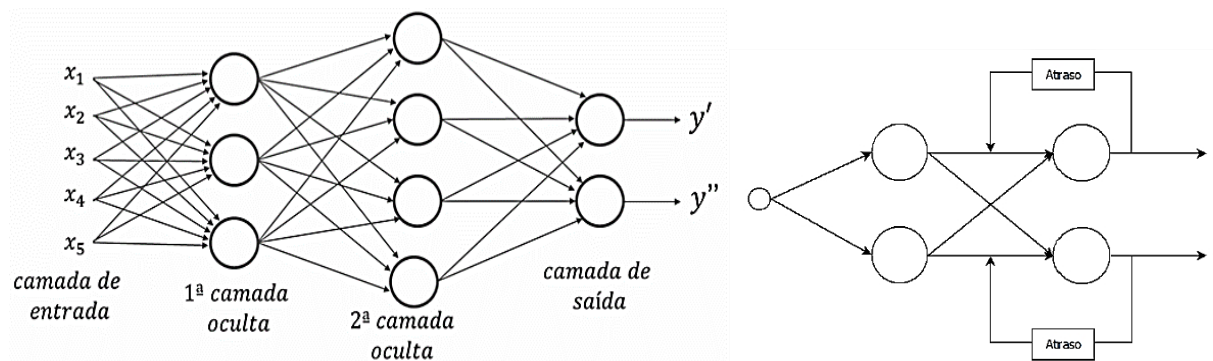
Fonte: Adaptado de KOVÁCS (2006).

2.2. Redes neurais artificiais (RNAs)

Apesar de o *Perceptron* sozinho ser capaz de resolver somente problemas linearmente separáveis em somente duas classes, ao se criar RNAs com mais de um neurônio pode-se resolver problemas mais complexos.

As RNAs compostas por uma única camada de *Perceptron* são chamadas de *Single-Layer Perceptron* (SLP), e as compostas por várias camadas, chamadas de *Multilayer Perceptron* (MLP). Nessas, a saída de cada neurônio é ligada às entradas dos neurônios da camada seguinte, propagando o dado da entrada da rede até a saída. Redes como estas, nas quais a informação se propaga da entrada para a saída, somente em um sentido, constitui uma topologia de rede do tipo *Feedforward*. Por outro lado, a topologia de rede que possui laços de realimentação, ou seja, que permite que os sinais percorram ambas as direções, é dita do tipo Recorrente. Essas topologias são mostradas na Figura 4.

Figura 4 – Redes *Feedforward* (à esquerda) e Recorrentes (à direita)



Fonte: Adaptado de KOVÁCS (2006).

2.3. Arranjos de portas programáveis em campo – FPGAS

Os primeiros dispositivos lógicos programáveis (*Programmable Logic Device* – PLD) foram introduzidos por volta da década de 70. A ideia principal era construir circuitos lógicos que fossem programáveis, mas diferentes dos microprocessadores que possuem *hardware* fixo (PEDRONI, 2010).

Visando projetos mais complexos e de elevado desempenho, foram lançadas as FPGAs. FPGA é um acrônimo para Arranjos de Portas Programáveis em Campo (*Field Programmable Gate Arrays*) e se constituem em dispositivos compostos por elementos lógicos reprogramáveis, conectados entre si via conexões que podem ser ajustadas, formando uma estrutura regular, paralela e hierárquica. Estes blocos e conexões estão em total controle do usuário e podem ser programados da maneira que for necessário para atender a aplicações específicas, de modo a se adaptar para padrões emergentes ou para mudanças de requisitos.

Em contraste à maioria dos *chips* que se encontra no dia-a-dia, como por exemplo, aqueles que acompanham as televisões, celulares, etc., os quais já vêm todos pré-programados (*chips* do tipo ASIC – *Application Specific Integrated Circuits*), isto é, com as suas funcionalidades todas definidas e pré-configuradas no ato de fabricação, as FPGA's constituem uma nova categoria de *hardware* reconfigurável, as quais têm as suas funcionalidades definidas exclusivamente pelos usuários e não pelos fabricantes.

Segundo o site especializado Grand View Research (2020), o valor de mercado das FPGAs estava estimado em 9 bilhões de dólares em 2019. A sua crescente aplicação em uso militar, aeroespacial, na eletrônica para consumidores e na indústria automotiva, é fator crítico para o crescimento de seu uso em novos segmentos e, com isso, projeta-se um aumento significativo do seu uso no mercado nos próximos anos.

Uma FPGA é dividida em três componentes principais: blocos lógicos configuráveis (CLB) e de funções fixas; blocos de E/S (IOB); e chaves de interconexão (ou de comutação).

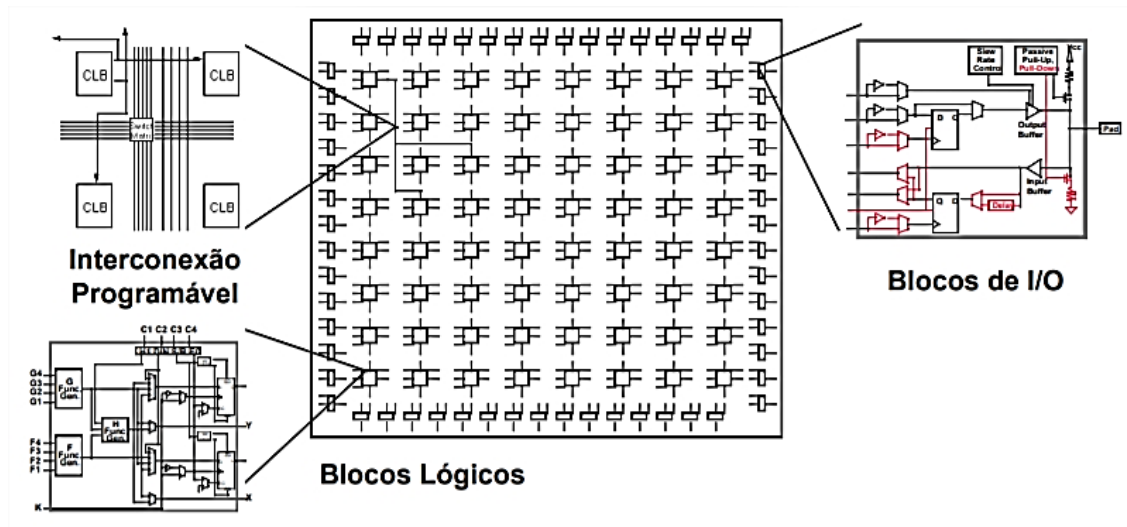
Os blocos lógicos se dividem em três Categorias de Granularidade, de acordo com as suas capacidades lógicas:

- **Pequena:** portas lógicas de duas entradas ou um multiplexador 4×1 e um *flip-flop*;
- **Média:** duas ou mais LUTs (*Look-Up Tables*), geralmente de 4 entradas, e dois ou mais *flip-flops*; e

- **Grande:** pequenos microprocessadores e/ou ALUs (*Arithmetic Logic Units*) e/ou memórias.

A LUT é basicamente uma memória pré-programada que fornece uma saída dado um conjunto de variáveis de entrada; assim, não realiza operação lógica alguma, apenas consulta a tabela verdade da função que na memória foi programada (ZAGHETTO; PRADO; TAVARES, 2003).

Figura 5 – Estrutura de uma FPGA



Fonte: CODÁ (2014).

2.4. VHDL

Em geral, FPGAs provêm circuito customizado através de linguagens de descrição de *hardware* para aplicações de *design* específico. Dentre elas, destaca-se a linguagem VHDL (*Very High Speed Integrated Circuit Hardware Description Language*).

O VHDL foi originalmente desenvolvida como uma ferramenta de apoio do projeto VHSIC (*Very High Speed Integrated Circuit*) da Agência de Projetos de Pesquisa Avançada de Defesa (DARPA – *Defense Advanced Research Projects Agency*) dos EUA, em meados da década de 1980, para documentar o comportamento de uma nova linha de circuitos integrados que compunham os equipamentos vendidos às Forças Armadas americanas.

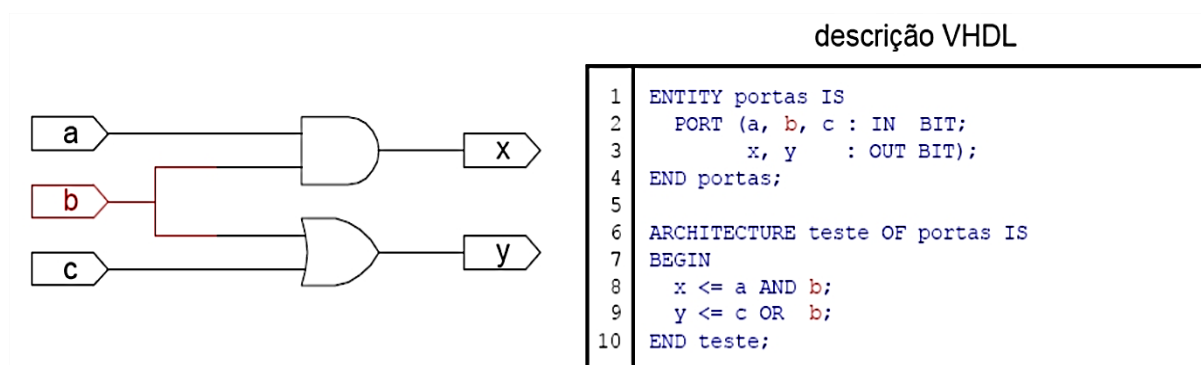
Uma vez que o projeto VHSIC era de alta prioridade militar e havia dezenas de fornecedores envolvidos, o Departamento de Defesa (DoD) estava preocupado principalmente com as questões de portabilidade, documentação e compreensibilidade dos projetos. Cada um destes fornecedores atuava desenvolvendo partes dos projetos ou mesmo fornecendo componentes que viriam a se encaixar em outros sistemas maiores. Desta forma o DoD optou por buscar

desenvolver uma linguagem que servisse como base para troca de informações sobre estes componentes e projetos. Uma linguagem que, independente do formato original do circuito, pudesse servir como uma descrição e documentação eficientes do circuito, possibilitando aos mais diferentes fornecedores e participantes entender o funcionamento das outras partes, padronizando a comunicação.

O desenvolvimento do VHDL serviu inicialmente aos propósitos de documentação do projeto VHSIC, porém, após o sucesso inicial do seu uso, a sua definição foi posta em domínio público, o que levou a ser padronizada pelo IEEE (*Institute of Electrical and Electronic Engineers*) em 1987, ampliando ainda mais a sua utilização e novas alterações sendo propostas (PEDRONI, 2010).

Atualmente o VHDL é utilizado nos meios industriais e acadêmicos, para: documentar, especificar, simular, e sintetizar circuitos digitais. O seu uso apresenta as seguintes vantagens: favorece projeto *top-down*, onde projetos complexos partem de um nível de especificação mais elevado para um mais baixo; permite descrever *hardware* em diversos níveis de abstração e mesclá-los em um mesmo código; suporte a projetos com múltiplos níveis de hierarquias (a descrição geral do circuito pode consistir na interligação de outras descrições menores); projeto independente da tecnologia, ou seja, um projeto pode ser criado sem antes ter que se escolher o dispositivo; comandos executados concorrentemente (com exceção de regiões específicas no código); possibilidade de, através de simulação, se verificar o comportamento do sistema digital; redução no tempo de projeto e no custo; diminuição do volume de documentação, já que um código bem comentado em VHDL substitui com vantagens os diagramas esquemáticos e a descrição funcional do sistema; facilidade na atualização e portabilidade dos projetos.

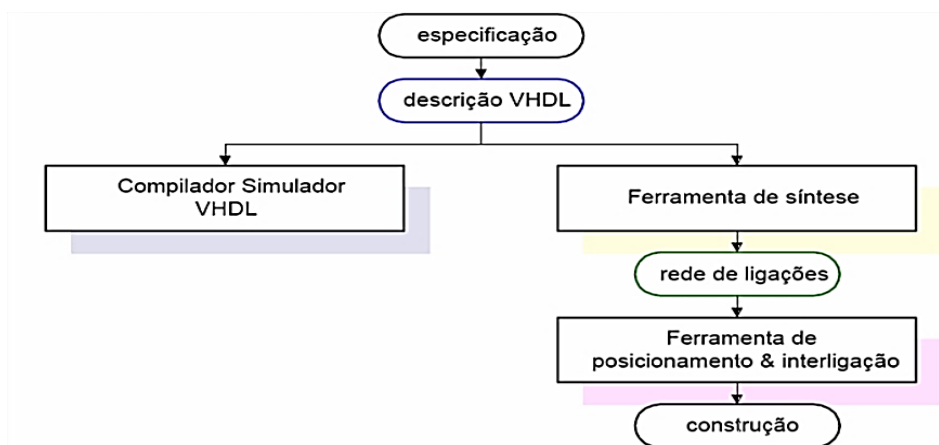
Figura 6 – Estrutura básica de um código em VHDL



Fonte: Adaptado de D'AMORE (2005).

Na figura 7 são apresentadas as etapas de um projeto utilizando VHDL.

Figura 7 – Etapas de um projeto utilizando VHDL



Fonte: Adaptado de D'AMORE (2005).

Uma descrição VHDL é gerada a partir das especificações do projeto. Então, esta descrição é submetida a um simulador que faz a verificação de correspondência entre a especificação e o código. A mesma descrição também é interpretada por uma ferramenta de síntese, esta, por sua vez, infere as estruturas necessárias para um circuito que corresponda à descrição prévia. D'Amore (2005) destaca que o resultado dessa etapa é um arquivo contendo uma rede de ligações de elementos básicos disponíveis na tecnologia do dispositivo empregado. Este arquivo é a base de dados que a ferramenta que realiza o posicionamento e interligações dos componentes utiliza. Por sua vez, a saída desta ferramenta é um arquivo que contém os dados necessários para a construção do dispositivo.

3. METODOLOGIA

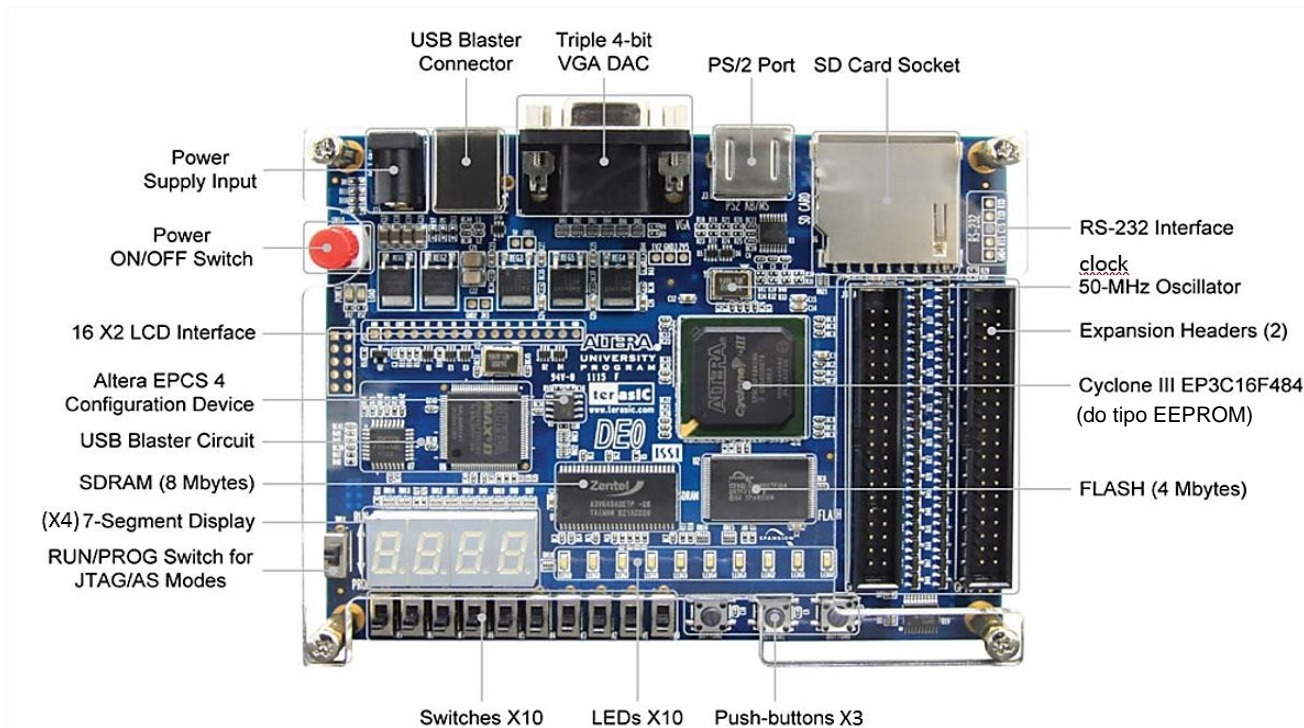
3.1. Kit de desenvolvimento DE0

O kit de desenvolvimento DE0 possui como principais componentes: a placa Altera DE0 (fabricada pela Terasic) e nela embarcada a FPGA Altera Cyclone III 3C16, esta que possui como características: 15408 LEs (Elementos Lógicos); 56 blocos de memória incorporados; 504K bits de RAM (*Random Access Memory*); 56 multiplicadores incorporados; e 346 pinos de entrada e saída para total o usuário. Cada bloco lógico da FPGA é chamado de LAB (*Logic Array Blocks*), que é constituído por 16 unidades básicas conhecidas como LEs (*Logic Elements*), os quais são formados por uma memória LUT que implementa a lógica combinacional, registradores (contendo *flip-flops*) e lógicas reconfiguráveis para interconexões entre os LEs adjacentes (ALTERA CORPORATION, 2012).

A placa possui tamanho compactado, mas possui todo o *hardware*, além da própria FPGA, e

características essenciais necessários para o usuário projetar uma gama variada de projetos e sobretudo para atendimento aos objetivos do presente projeto, como se percebe na Figura 8.

Figura 8 – Placa DE0 e seus componentes de *Hardware* (com a FPGA Altera Cyclone III 3C16)



Fonte: TERCASIC TECHNOLOGIES (2011).

O Apêndice A ilustra o diagrama de bloco da placa DE0. Para fornecer a máxima flexibilidade para o usuário, todas as conexões são feitas através do dispositivo Cyclone III FPGA. Assim, ele pode configurar a FPGA para implementar qualquer projeto de sistema.

Além disso, o *kit* DE0 dispõe de outras ferramentas que foram essenciais para o desenvolvimento do projeto, sendo uma delas o Quartus II Web Edition, um ambiente para análise e síntese de projetos digitais, disponibilizado para *download* diretamente no site do fabricante Altera, gratuitamente; o manual e *datasheet* da placa; e o painel de controle para teste dos componentes da placa.

O painel de controle possibilita ao usuário conectar o computador à placa DE0 através de uma conexão USB; acessar vários componentes na placa através do computador; e testar e verificar a funcionalidade desses componentes. O código que executa o controle dessas funções é pré-implementado dentro da FPGA.

O Quartus II dispõe de suporte para várias FPGAs; possui uma interface gráfica para simulação;

análise e síntese de circuitos, a qual faz uso de diagrama de blocos ou de um editor de texto com suporte a VHDL e Verilog (UNICAMP, 2010); e uma tela para conectar as entradas e saídas lógicas do circuito com os componentes da placa. Além disso, possibilita o carregamento dos dados de configuração (esses que servem para inicializar e controlar corretamente o dispositivo, como conexões de roteamento e outras funções programáveis), diretamente na memória SRAM da FPGA usando a configuração *JTAG (Joint Test Action Group)* – modo *RUN*.

Para maiores detalhes relacionados aos principais componentes do *kit* DE0, vide Apêndice A.

O *kit* DE0 foi escolhido levando em consideração aspectos importantes do projeto, como: utilização em uma variedade de situações-problemas práticos, suas características suficientes de *hardware* e de *software*, sua escalabilidade e capacidade de fomentar projetos futuros relacionados e também de suporte didático, os quais vão desde, minimamente, a simples lógicas, aquisição de dados e controle a cálculos matemáticos, processamento de sinais e aplicações que demandam processamento paralelo e de tempo real.

3.2. Implementação da RNA do tipo *Perceptron*

A implementação em *software* traz como vantagem uma grande facilidade e velocidade no desenvolvimento, já que várias ferramentas já estão disponíveis no mercado, como por exemplo o software MATLAB que tem uma *Toolbox* para o desenvolvimento e treinamento de RNAs de várias topologias diferentes. A facilidade no design de novas RNAs faz com que este tipo de implementação traga uma flexibilidade ao desenvolvimento, permitindo que RNAs de tipos e tamanhos diferentes possam ser implementadas no mesmo dispositivo. Porém, como se trata de um processador sequencial simulando o funcionamento de uma RNA, que é um sistema massivamente paralelo e hierárquico, a velocidade de operação e a capacidade de processamento se tornam limitadas. A diferença nesses aspectos é mais notável quando grandes RNAs são necessárias e, também, quando o sistema em que a RNA se encontra exige operação em tempo real. Para demais aplicações, as implementações em *software* e em *hardware* podem ser combinadas.

Cria-se, então, um sistema que capaz de se adaptar às eventuais mudanças de requisitos e deixa-se a cargo do usuário decidir qual arquitetura de RNA usar (DEOTALE, 2014). Esta adaptação trazida pela reconfiguração da rede em *hardware* pode ser feita a tempo de execução, de maneira rápida, o que é importante para dispositivos embarcados. De qualquer forma, a

reconfiguração traz um certo impacto no tempo de execução, já que um tempo deve ser gasto para fazer mudanças no circuito. Felizmente, dependendo das aplicações, como as aqui propostas, este impacto no tempo pode ser desconsiderado.

Como as FPGAs tem um tamanho e uma quantidade de blocos lógicos limitados, há uma limitação no tamanho que a RNA pode atingir, já que cada neurônio, se implementado de maneira totalmente paralela, precisa de uma grande quantidade de multiplicações e de memória para guardar seus pesos. Como se sabe, as operações de multiplicação demandam circuitos complexos, o que faz com que a implementação de RNAs em FPGAs com grande número de neurônios seja difícil.

Nesse sentido, a SLP (*Single-Layer Perceptron*), uma topologia de RNA baseada em uma única camada, se mostrou suficiente para os problemas aqui trabalhados. Nela, o número de entradas e pesos sinápticos podem variar de acordo com o problema, assim como a quantidade de neurônios na camada e sua função de ativação.

Assim, a implementação em *software* foi útil para o teste e identificação da melhor arquitetura do *Perceptron* para um dado problema, assim como para o treinamento da rede (abordagem *offline*), para que, em seguida, pudesse ser implementada em *hardware* (abordagem *online*).

Como esse trabalho focou em problemas de classificação, entre duas classes diferentes, linearmente separáveis, optou-se pelo uso de um único neurônio, que gerava uma única saída binária (-1 ou 0 e 1) e recebia uma quantidade variável de entradas conforme a aplicação. Sua ativação foi dada pela função sinal ou degrau e seu *bias*, associado a uma entrada de valor -1 .

3.3. Treinamento da rede

O treinamento supervisionado é um método adaptativo em que, a cada instante, quando uma entrada é aplicada à rede neural, a saída correspondente esperada é conhecida e também fornecida à rede. A resposta computada pela rede é, então, observada e comparada com aquela desejada e o erro entre elas é medido. Esse erro computado pode ser usado para realizar o ajuste dos parâmetros nas conexões em cada camada, até se achar a melhor combinação de pesos que permita que ambas as respostas se aproximem umas às outras, refletindo no aprendizado da rede.

Para o treinamento da rede, a base de dados utilizada na aplicação trabalhada foi extraída de um repositório público, o *UCI Machine Learning Repository*.

A fim de evitar que o treinamento se tornasse tendencioso e de se acelerar a convergência do processo de aprendizado da rede, visto que a variação do erro a cada iteração se demonstrou pequena, foi feito um embaralhamento prévio da base de dados. E na sequência, a fim de evitar que a rede ficasse viciada nos dados da base (efeito de memorização), essa foi, então, dividida de modo a englobar 70% dos dados para o treinamento em *software* e 30% para testes e validação da rede na FPGA. Esses percentuais foram definidos experimentalmente.

3.4. Algoritmo de treinamento do *Perceptron*

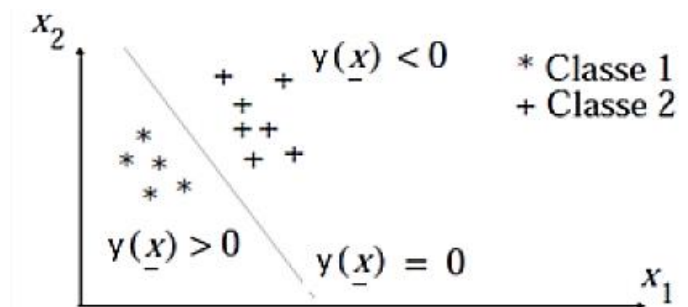
Durante o processo de aprendizagem, para cada entrada $x(k) = [x_1(k), x_2(k), \dots, x_m(k)]^T$, $k = 1, 2, \dots, p$, sendo p o número de pares entrada-saída do conjunto de treinamento, m o número de elementos do vetor de entrada, o *Perceptron* deve ter seus pesos modificados de modo a gerar uma saída $y_1(k)$ igual a uma saída desejada (*target*) $d_1(k) \in \{-1, 1\}$, ou seja:

$$y_i^{(k)} = \text{sign}(w_i^T x^{(k)}) = \text{sign}\left(\sum_{j=1}^m w_{ij} x_j^{(k)}\right) = d_i^{(k)} \quad (3)$$

onde $w_i = [w_{i1}, w_{i2}, \dots, w_{im}]^T$ é o vetor de pesos associado ao neurônio i .

Em outras palavras, é necessário separarmos linearmente o espaço de entrada por meio de um hiperplano, de modo que entradas associadas à classe (c_1), as quais geram +1 como saída ($y > 0$), sejam separadas, por meio de um plano ($y = 0$), daquelas entradas associadas à outra classe (c_2), as quais geram saída -1 ($y < 0$), como mostra a Figura 9.

Figura 9 – Hiperplano separador



Fonte: MIRANDA, 2010

Cada dado do conjunto de treinamento (entrada) da base é processado individualmente em sequência, sendo o vetor de pesos ajustado incrementalmente a cada iteração. O mesmo segue a seguinte regra de adaptação:

$$\Delta w_1 = \eta \cdot [d_i - y_i] \cdot x^{(k)} = \eta \cdot [d_i - \text{sign}(w_i^T \cdot x^{(k)})] \cdot x^{(k)}$$

$$\rightarrow w_i \cdot (\tau + 1) = w_i \cdot (\tau) + \begin{cases} d_i x^{(k)}, & \text{se } y_i \neq d_i \\ 0, & \text{se } y_i = d_i \end{cases} \text{ e } \eta = 0.5 \quad (4)$$

Se $\Delta w_1 = 0 \rightarrow y_i = d_i$ para todas as entradas, ou seja, todos os padrões são classificados corretamente, o algoritmo para e converge para o valor do vetor de pesos obtido. O algoritmo de treinamento do *Perceptron* é esquematizado na Figura 10.

Figura 10 – Algoritmo de Treinamento do *Perceptron*

<p>Considere P o conjunto contendo os padrões associados com a saída positiva ($y > 0$) e classe c_1, e N o conjunto contendo os padrões associados com a saída negativa ($y < 0$) e classe c_2.</p>	
<u>start:</u>	o vetor de pesos é inicializado randomicamente em $\tau = 0$;
<u>test:</u>	um padrão $x \in P \cup N$ é inserido na rede de modo a ser classificado:
se todos os padrões foram classificados corretamente por meio de um mesmo vetor de pesos \rightarrow o algoritmo converge e pára (vá para <u>end</u>);	
a) se $x \in P$ e o produto escalar $w \cdot x > 0$ (ângulo entre tais vetores é menor que 90°) \rightarrow padrão classificado corretamente \rightarrow continue testando os outros padrões (<u>test</u>);	
b) se $x \in P$ e o produto escalar $w \cdot x \leq 0$ \rightarrow erro de classificação \rightarrow atualize o vetor de pesos por meio de uma adição (vá para <u>add</u>);	
c) se $x \in N$ e o produto escalar $w \cdot x < 0$ (ângulo entre tais vetores é maior que 90°) \rightarrow padrão classificado corretamente \rightarrow continue testando os outros padrões (<u>test</u>);	
d) se $x \in N$ e o produto escalar $w \cdot x \geq 0$ \rightarrow erro de classificação \rightarrow atualize o vetor de pesos por meio de uma subtração (vá para <u>subtract</u>);	
<u>add:</u>	$w_i(\tau + 1) = w_i(\tau) + x$; $\tau = \tau + 1$ e continue testando os outros padrões (vá para <u>test</u>);
<u>subtract:</u>	$w_i(\tau + 1) = w_i(\tau) - x$; $\tau = \tau + 1$ e continue testando os outros padrões (vá para <u>test</u>);
<u>end:</u>	o algoritmo converge para o vetor de pesos que classifica corretamente todos os padrões. Esse vetor é perpendicular (normal) ao hiperplano linearmente separador do espaço de entrada.

Fonte: MIRANDA, 2010.

Vale destacar que η representa a taxa de aprendizado ou a velocidade que o treinamento converge para a estabilidade, sendo que ele deve estar entre o intervalo $0 < \eta < 1$. Para as soluções dos problemas apresentados no presente trabalho foi utilizado $\eta = 0,5$, pois se demonstrou adequado como forma de incrementar ou decrementar o peso a partir do valor de entrada.

Além disso, quando o valor da taxa de aprendizado é nulo, não ocorre aprendizado da rede, permanecendo inalterado o valor dos pesos; caso contrário, o aprendizado é garantido até que o mesmo se sature, na medida em que os valores dos pesos a serem incrementados em cada iteração não mais se modifiquem, ou, quando da não saturação, o aprendizado é garantido até que o erro desejável seja atingido e, logo, a convergência alcançada.

O código original que implementa o algoritmo de treinamento do *Perceptron* se encontra em Miranda (2020).

3.5. Etapa de validação

Na fase de validação, um conjunto de teste, composto por dados que não foram previamente utilizados no treinamento, é utilizado para determinar a capacidade de generalização da rede. A rede treinada deve ser capaz de reconhecer se novas entradas são similares aos padrões aprendidos para, então, produzir respostas similares. Ela poderá estimar, reconhecer e classificar dados desconhecidos ou incompletos, inferindo soluções e até mesmo capturando relações sutis dentre os dados.

3.6. Descrição da rede usando VHDL para síntese em FPGA

A FPGA interpreta a descrição do *Perceptron* apresentada em VHDL, a partir de suas entidades, arquiteturas e componentes e, a partir dela, sintetiza os circuitos lógicos correspondentes.

Para isso, cada entrada da base de validação, assim como os pesos da rede obtidos após o treinamento são convertidos de valores reais para binários, usando notação de ponto fixo. A quantidade de *bits* usada para a representação da parte inteira dos dados de entrada depende da faixa de variação dos respectivos valores; já para a parte fracionária usa-se a quantidade restante de *bits*, de modo a assegurar maior precisão quando dessa conversão. Como a faixa de variação da parte inteira é limitada, em virtude das próprias características dos dados, a amplitude desses valores é pequena demandando menos *bits* que os usados para a parte fracionária.

Vale destacar que o separador decimal presente na representação binária dos dados tem o objetivo de facilitar a interrelação dos dados convertidos com os originais, mas é omitido na implementação prática do circuito.

Assim, todos os *input switches* da placa acabam sendo utilizados, uma vez que são associados a cada *bit* dos vetores de entrada do *Perceptron*.

Uma vez imputado pelo usuário, através dos *switches*, cada vetor binário de entrada da rede é armazenado em diferentes registradores, os quais contêm um *flip-flop D* para cada *bit* do vetor. Associado a cada registrador, um diferente *pushbutton* da placa (após aplicado o *debounce*) desempenha o papel de *clock*. Inicialmente fornecem nível lógico 1, quando não pressionados, e quando pressionados fornecem 0. Assim, a cada transição de descida do pulso de *clock*, ocorrida na prática quando se aperta um *pushbutton*, gera-se um sinal de sincronismo que memoriza junto à rede o vetor binário de entrada do registrador.

Quanto à representação dos pesos, utilizou-se a mesma quantidade total de *bits* que para os dados de entrada do *Perceptron*, em virtude dos mesmos motivos.

Uma vez registradas todas as entradas de cada linha de teste da base de validação, a saída da rede, para cada uma delas, é calculada, sendo exibida em um LED da placa, e comparada com a respectiva saída esperada.

Os códigos implementados em VHDL se encontram no repositório criado pelos autores, Abreu e Fim (2020), do presente trabalho; já as pinagens das entradas e saídas do sistema neural, em geral, sintetizado se encontram no Apêndice B.

4. RESULTADOS E DISCUSSÃO

4.1. Resultados da etapa de treinamento

O conjunto de dados, no qual o treinamento do *Perceptron* foi baseado, é o *Iris Data Set* (FISHER, 1988), que é mundialmente conhecido e talvez o mais usado em problemas de classificação e reconhecimento de padrões. Originalmente, ele se propõe a classificar a planta do tipo Íris em 3 classes, a partir de medidas de largura e comprimento da sua sépala e da sua pétala, de modo que, para redes com uma saída, como o *Perceptron*, 3 sub-bases podem ser criadas, extraídas da base original, contendo informações referentes a duas classes entre si.

No caso do presente trabalho, uma dessas bases criadas foi utilizada, compreendendo 2 classes (saídas) associadas a 50 amostras de entrada cada, onde cada classe se refere a um tipo da planta íris. Cada atributo de uma determinada amostra representa uma entrada do *Perceptron*, de valor real, e são descritos, sequencialmente, por: x_1 – comprimento (cm) da sépala; x_2 – largura (cm) da sépala; x_3 – comprimento (cm) da pétala; e x_4 – largura (cm) da pétala. Já sua saída é descrita como Iris Setosa $\{y = 1\}$ ou Iris Versicolour $\{y = -1\}$.

A base utilizada para o treinamento se encontra no repositório criado pelos autores, Abreu e Fim (2020), do presente trabalho, e os parâmetros obtidos, na Tabela 1.

Tabela 1 – Parâmetros de Treinamento

Parâmetro	Valor Decimal
Peso sináptico w_1	-0.7
Peso sináptico w_2	9.9
Peso sináptico w_3	-7.8
Peso sináptico w_4	1.6

Limiar de disparo θ	6.0
Pesos iniciais: [2,8,1,5,6] e quantidade de iterações para convergência: 6	

Fonte: Autor

Percebe-se que a rede converge rapidamente, com um pequeno número de iterações necessárias para dividir, pelo hiperplano, os dados com saídas diferentes, o que mostra que a estimativa inicial para o vetor de pesos se faz assertiva para acelerar o aprendizado do *Perceptron*.

4.2. Resultados da etapa de validação

Tabela 2 – Parâmetros da Rede convertidos em Binário

Parâmetro	Valor Binário
Peso sináptico w_1	0000.101100
Peso sináptico w_2	1001.111001
Peso sináptico w_3	0111.110011
Peso sináptico w_4	0001.100110
Limiar de disparo θ	0110.000000

Fonte: Autor

Percebeu-se que todas as entradas da base são positivas e a minoria dos pesos da rede obtidos no treinamento são negativos. Assim, uma estratégia, para se evitar a representação dos dados com *bit* de sinal ou via complemento de dois, consistiu em transferir os termos $-x_i \cdot w_i$ para a outra entrada do comparador, como entradas $+x_i \cdot w_i$ dos módulos somadores, como mostra a Equação (5), a qual implementa a saída do *Perceptron*.

$$y = \begin{cases} 0, & \text{se } x_2 \cdot w_2 + x_4 \cdot w_4 < x_1 \cdot w_1 + x_3 \cdot w_3 + \theta \\ 1, & \text{se } x_2 \cdot w_2 + x_4 \cdot w_4 > x_1 \cdot w_1 + x_3 \cdot w_3 + \theta \end{cases} \quad (5)$$

Note que a Equação (5) possui nenhuma operação de subtração, resultando em maior simplicidade do circuito gerado, e menor número de elementos lógicos para sintetizá-lo na FPGA. O aproveitamento de muitos *bits* na representação dos dados e parâmetros da rede resulta em um maior consumo de recursos da FPGA, uma vez que blocos multiplicadores de grande tamanho serão sintetizados. Ao se representar o número de *bits* dos pesos w_i por N_w e o número de *bits* das entradas x_i por N_x , o número binário resultante do produto $w_i \cdot x_i$ será formado por $N_w + N_x$ *bits*. Porém, isso não se demonstrou um fator crítico nem limitante para o presente projeto (PIÃO, 2012).

Para fins de simplificação, a Tabela 3 mostra um terço representativo dos dados da base de validação utilizados para os testes da rede na FPGA. O restante se encontra no Apêndice C.

Tabela 3 – Dados de Validação e Resultados

Entradas – Saídas	x_1	x_2	x_3	x_4	Saída Teórica	Saída Real
Dados 1	6.9	3.1	4.9	1.5	-1	-1
	110.1110011	011.0001100	100.1110011	001.1000000	0	0
Dados 2	5.2	4.1	1.5	0.1	1	1
	101.0011001	100.0001100	001.1000000	000.0001100	1	1
Dados 3	5.7	2.8	4.1	1.3	-1	-1
	101.1011001	010.1100110	100.0001100	001.0100110	0	0
Dados 4	6.1	3	4.6	1.4	-1	-1
	110.0001100	011.0000000	100.1001100	001.0110011	0	0
Dados 5	4.8	3.1	1.6	0.2	1	1
	100.1100110	011.0001100	001.1001100	000.0011001	1	1
Dados 6	6	2.2	4	1	-1	-1
	110.0000000	010.0011001	100.0000000	001.0000000	0	0
Dados 7	4.9	3.1	1.5	0.1	1	1
	100.1110011	110.0011000	001.1000000	000.0001100	1	1
Dados 8	4.6	3.2	1.4	0.2	1	1
	100.1001100	110.0110010	001.0110011	000.0011001	1	1
Dados 9	5.7	4.4	1.5	0.4	1	1
	101.1011001	100.0110011	001.1000000	000.0110011	1	1
Dados 10	6.3	2.5	4.9	1.5	-1	-1
	110.0100110	010.1000000	100.1110011	001.1000000	0	0

Fonte: Autor

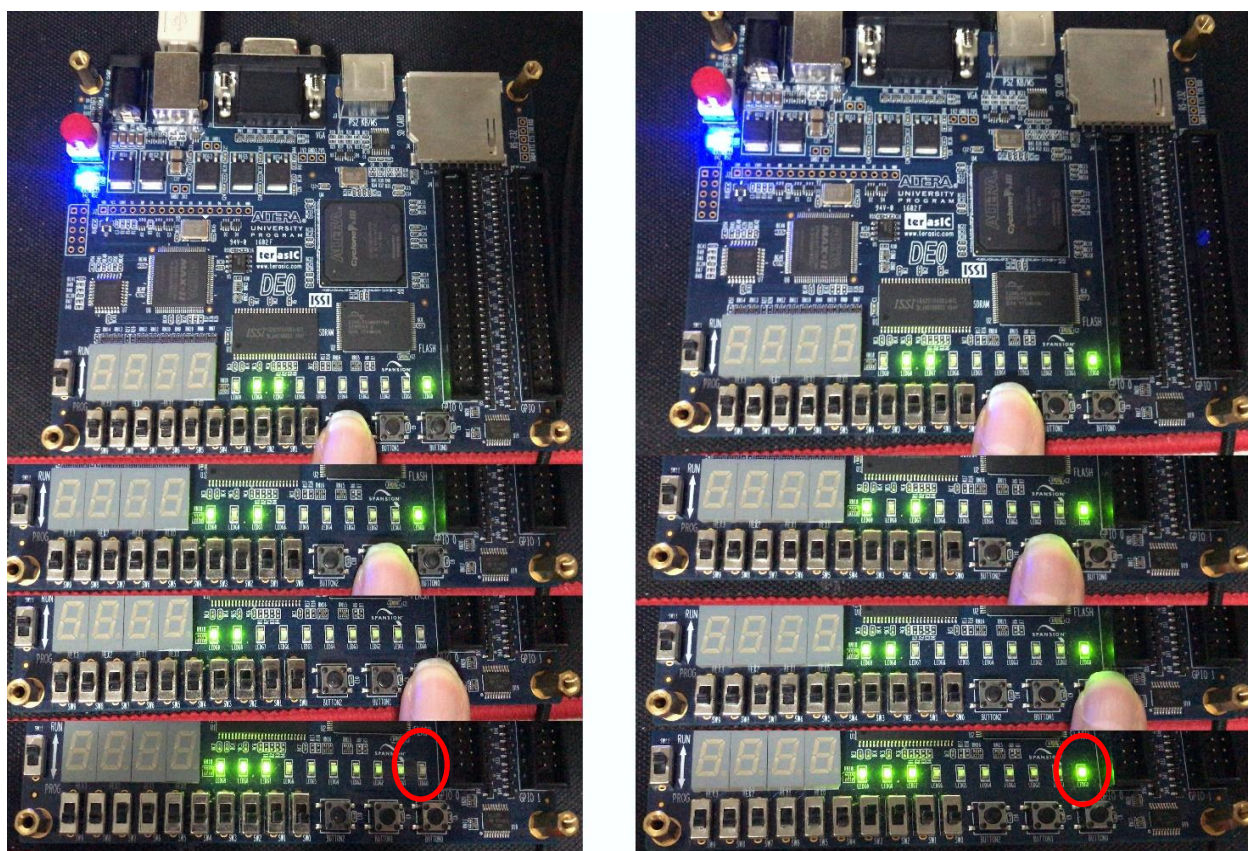
Quatro parâmetros, conhecidos na literatura, são importantes para estabelecer algumas métricas de avaliação dos resultados para problemas de classificação. São eles:

- **Verdadeiro Positivo (VP):** quantidade de classes que deveriam ser positivas e foram classificadas como positivas.
- **Verdadeiro Negativo (VN):** quantidade de classes que deveriam ser negativas e foram classificadas como negativas.
- **Falso Positivo (FP):** quantidade de classes que deveriam ser classificadas como negativas, mas foram classificadas erroneamente como positivas.
- **Falso Negativo (FN):** quantidade de classes que deveriam ser classificadas como positivas, mas foram classificadas erroneamente como negativas.

A partir dos resultados obtidos, percebe-se que o número de acertos da rede foi de 100%, equivalendo a 100% de VP e de VN, assim como 0% de FP e de FN. Isso se deve ao grau de precisão dos dados imputados na rede, durante seu treinamento, representados em notação de ponto fixo e à representatividade suficiente da base de dados utilizada para o treinamento.

A Figura 11 mostra alguns desses testes executados na placa DE0.

Figura 11 – Entradas-Saída de validação do *Perceptron* (à esquerda, resultados testados a partir dos Dados 1 e à direita, resultados testados a partir dos Dados 2) - respectivas saídas circuladas nas imagens (1 = LED aceso; 0 = LED apagado)



Fonte: Autor

5. CONCLUSÃO

A grande utilidade do *kit*, além do aprendizado adquirido, foi a prototipação de um produto e teste de conceito.

O uso de FPGAs para aplicações que se beneficiam do paralelismo e da customização é uma opção válida e competitiva no mercado. A constante evolução destes dispositivos mostra que se tornaram uma opção sólida comparada a outras formas de implementação de RNAs.

Ao se utilizar VHDL como linguagem para se criar a rede do tipo *Perceptron*, algumas vantagens surgem como a facilidade na atualização de projetos, redução do custo de projeto e a simplificação da geração de documentos para o projeto. A utilização do VHDL também simplificou o processo de desenvolvimento, pois facilitou a implementação de registradores, facilitando o armazenamento dos valores das entradas da rede e permitindo a utilização de todos os *switches* da placa para cada entrada, de modo a garantir maior precisão na conversão dos

valores para binário, e simulando em formato de código o comportamento de circuitos digitais com múltiplas portas lógicas.

O código em VHDL, é completamente portátil entre FPGAs diferentes, no caso de se utilizar um modelo mais robusto no futuro. Além disso, essa linguagem também permitiu descrever e sintetizar um único neurônio, isoladamente, e testá-lo na FPGA até que se obtivesse o resultado desejado; o neurônio obtido pode então futuramente ser inserido em um circuito maior, quantas vezes forem necessárias até que se obtenha a rede neural desejada, com múltiplas camadas, ou pode, até mesmo, ser utilizado em diversos projetos distintos, em quantidades distintas, formando diferentes redes neurais dos mais variados tamanhos e complexidade.

O reconhecido problema de classificação da planta Íris foi testado. Este problema se propõe a classificar plantas do tipo Íris em 3 classes, a partir de medidas de largura e comprimento da sépala e da pétala. A base de dados utilizada trabalhou com dados relacionados a 2 dessas classes. Para o treinamento do *Perceptron*, utilizou-se do *software* Matlab. Interessante salientar que, dependendo do problema, como o aqui trabalhado, o algoritmo implementado sempre converge em um número finito de iterações para qualquer vetor de pesos inicializado randomicamente (PALMIERE, 2020).

Durante a etapa de validação, para cada conjunto de entradas, as respectivas saídas geradas pelo *Perceptron* sintetizado no FPGA, foram exatamente iguais às saídas esperadas, o que comprova a eficácia da metodologia empregada e o potencial desse para servir de base a trabalhos futuros.

REFERÊNCIAS

- ABREU, F. N.; FIM, D. **Repositório de códigos e arquivos**. Disponível em: <https://github.com/Diogo-Fernando-TCC/VHDLTCC>. Acesso em 26 maio 2020.
- ALTERA CORPORATION. **Cyclone iii device handbook**. 2012. Disponível em: www.altera.com/literature/hb/cfg/config_handbook.pdf. Acesso em: 01 mar. 2020.
- CAVUSLU, M.; KARAKUZU, C.; SAHIN, S. **Neural network hardware implementation using fpga**. In: Neural Information Processing, 2006.
- CORIC, S., LATINOVIC I., PAVASOVIC A. **A neural network fpga implementation**. 5th seminar on Neural Network Applications in Electrical Engineering, 2000.
- CODÁ, L. M. R. **Dispositivos lógicos programáveis**. Departamento de Engenharia Elétrica e de Computação (EESC) – USP, 2014.
- D'AMORE, R. **VHDL: descrição e síntese de circuitos digitais**. LTC, 2005. p 656.

DEOTALE, P. D.; DOLE, L. **Design of fpga based general purpose neural network**. In.: IEEE. Information Communication and Embedded Systems (ICICES), 2014. pp. 1–5.

FISHER, R. A. **Iris plants database**. 1988. Disponível em: <http://archive.ics.uci.edu/ml/datasets/Iris>. Acesso em: 20 mar. 2020.

GRAND VIEW RESEARCH. **Field programmable gate array market size, share & trends analysis report**, 2020. Disponível em: <https://www.grandviewresearch.com/industry-analysis/fpga-market>. Acesso em: 10 mar. 2020.

KOVÁCS, L. Z. **Redes neurais artificiais: fundamentos e aplicações**, 2006.

MIRANDA, V. M. **Perceptron simples: algoritmo de aprendizagem aplicado a 2 classes**, 2010.

OLIVEIRA, J. G. M. **Uma arquitetura reconfigurável de rede neural artificial utilizando fpga**. 2017. Dissertação de Mestrado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Itajubá, Itajubá-MG.

PALMIERE, S. E. **Rede perceptron de uma única camada**. 2016. Disponível em: <https://www.embarcados.com.br/rede-perceptron-de-uma-unica-camada>. Acesso em: 02 mar. 2020.

PEDRONI, V. A. **Eletrônica digital moderna e vhdL**. Elsevier Editora Ltda, 2010. p.787.

PIÃO, S. S. **Implementação de rede neural artificial em fpga utilizando vhdL**. 2012. Trabalho de Conclusão de Curso, Engenharia Elétrica, Escola de Engenharia de São Carlos da Universidade de São Paulo, São Carlos-SP.

SILVA, I. N.; SPATTI, D. H.; FLAUZINO, R. A. **Redes neurais artificiais para engenharia e ciências aplicadas**, 2010.

STANFORD CS CLASS. **Convolutional neural networks for visual recognition**. Disponível em: <https://cs231n.github.io/neural-networks-1>. Acesso em: 20 maio 2020.

SUGAHARA, K.; OIDA, S.; YOKOYAMA, T. **High performance fpga controller for digital control of power electronics applications**. Power Electronics and Motion Control Conference, New York, 2009.

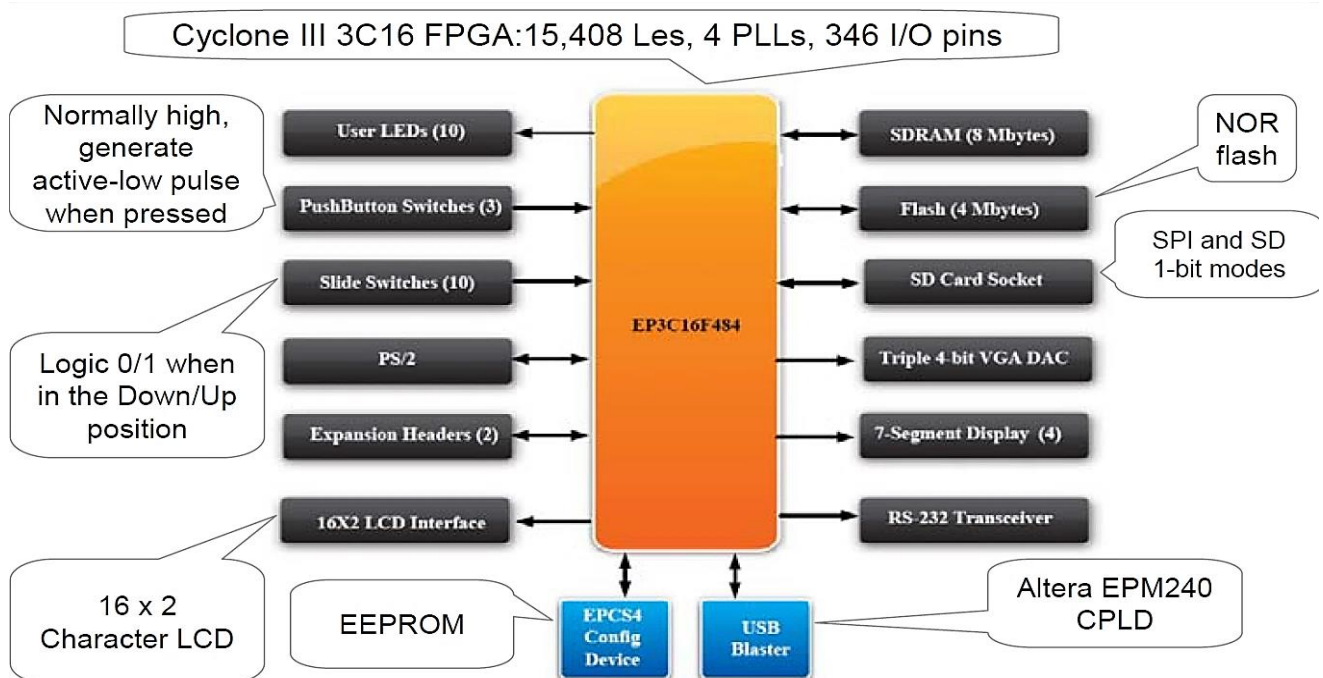
TERASIC TECHNOLOGIES. **DE0 user manual**. 2011. Disponível em: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=56&No=364>. Acesso em: 20 Abril. 2020.

UNICAMP. **Altera quartus ii introduction using vhdL design**. 2010. Disponível em: https://www.ic.unicamp.br/~cortes/mc602/tutoriais/tut_quartus_intro_vhdl.pdf. Acesso em: 03 mar. 2020.

ZAGHETTO, A.; PRADO, A. C.; TAVARES, A. **HCPLD - high capacity programmable logic devices**. 2003. Disponível em: https://www.gta.ufrj.br/grad/01_1/pld/hcpld. Acesso em: 21 abr. 2020.

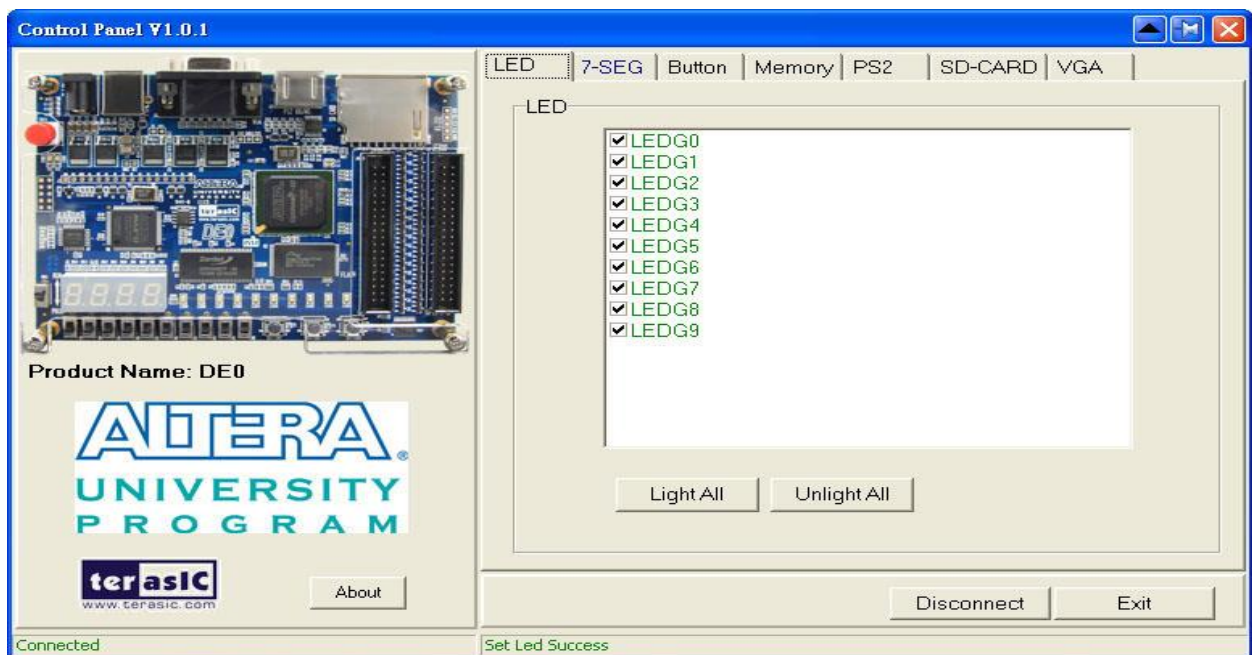
APÊNDICE A – Principais Componentes do *Kit* de Desenvolvimento DE0

Diagrama de Blocos da Placa DE0:



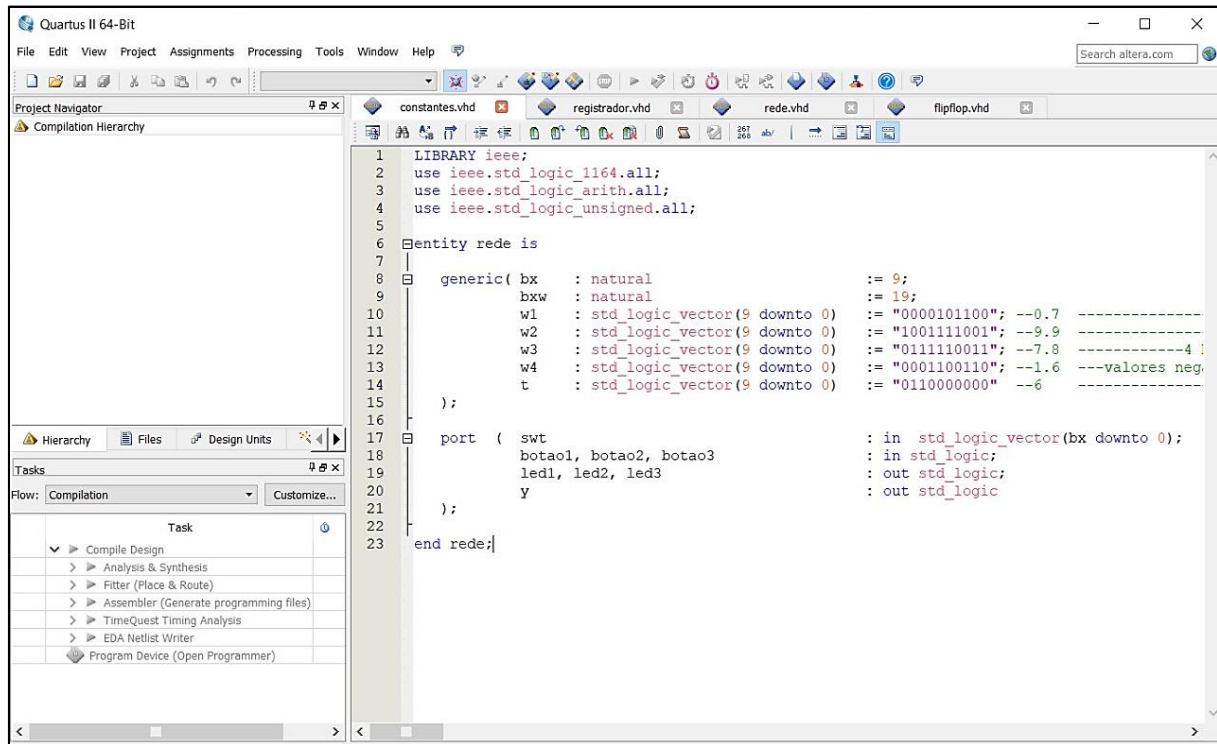
Fonte: Adaptado de TERCASIC TECHNOLOGIES (2011).

Interface Gráfica do Painel de Controle:



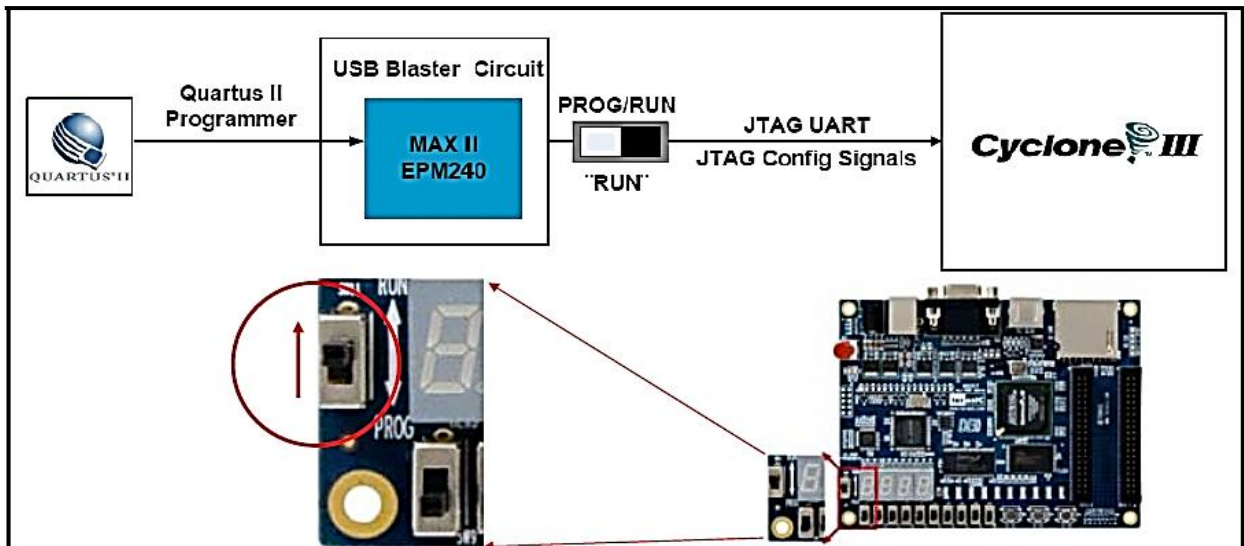
Fonte: Adaptado de TERCASIC TECHNOLOGIES (2011).

Interface Gráfica do Quartus II:



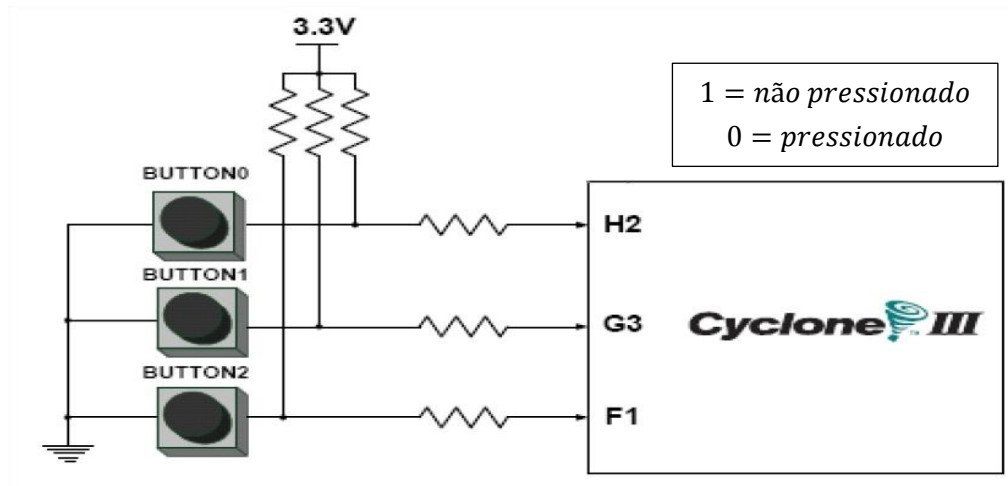
Fonte: Autor.

Tipo de Configuração JTAG (modo RUN):



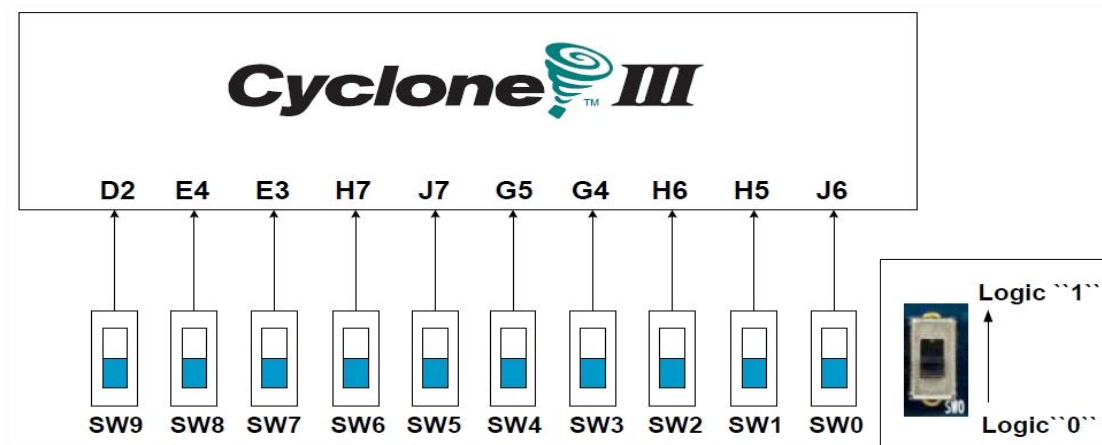
Fonte: Adaptado de TERCASIC TECHNOLOGIES (2011).

Esquema de conexão entre os *pushbuttons* e a FPGA:



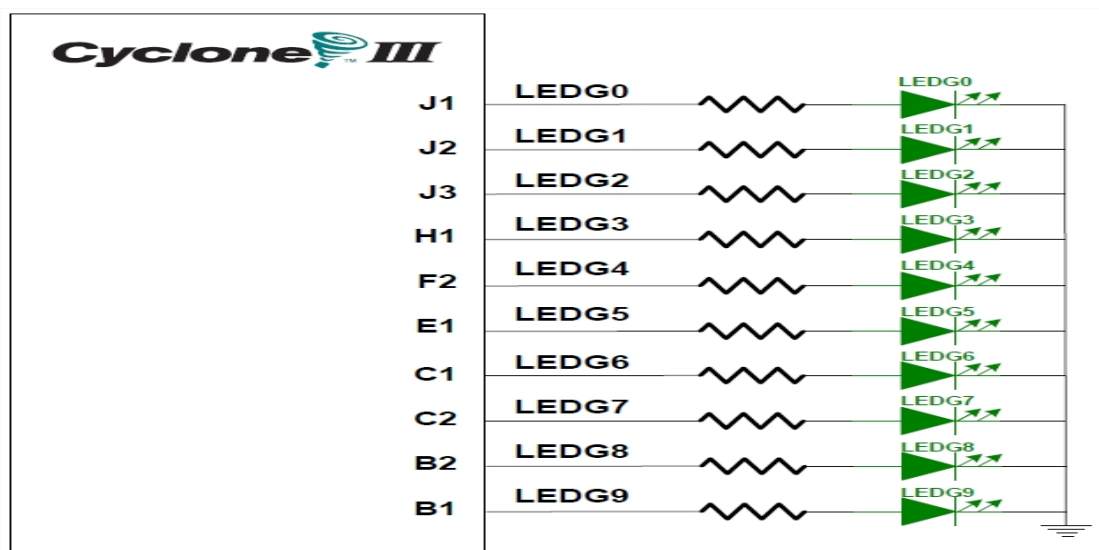
Fonte: Adaptado de TERCASIC TECHNOLOGIES (2011).

Esquema de conexão entre os *slides switches* e a FPGA:




















Fonte: TERCASIC TECHNOLOGIES (2011).

Esquema de conexão entre os LEDs e a FPGA:

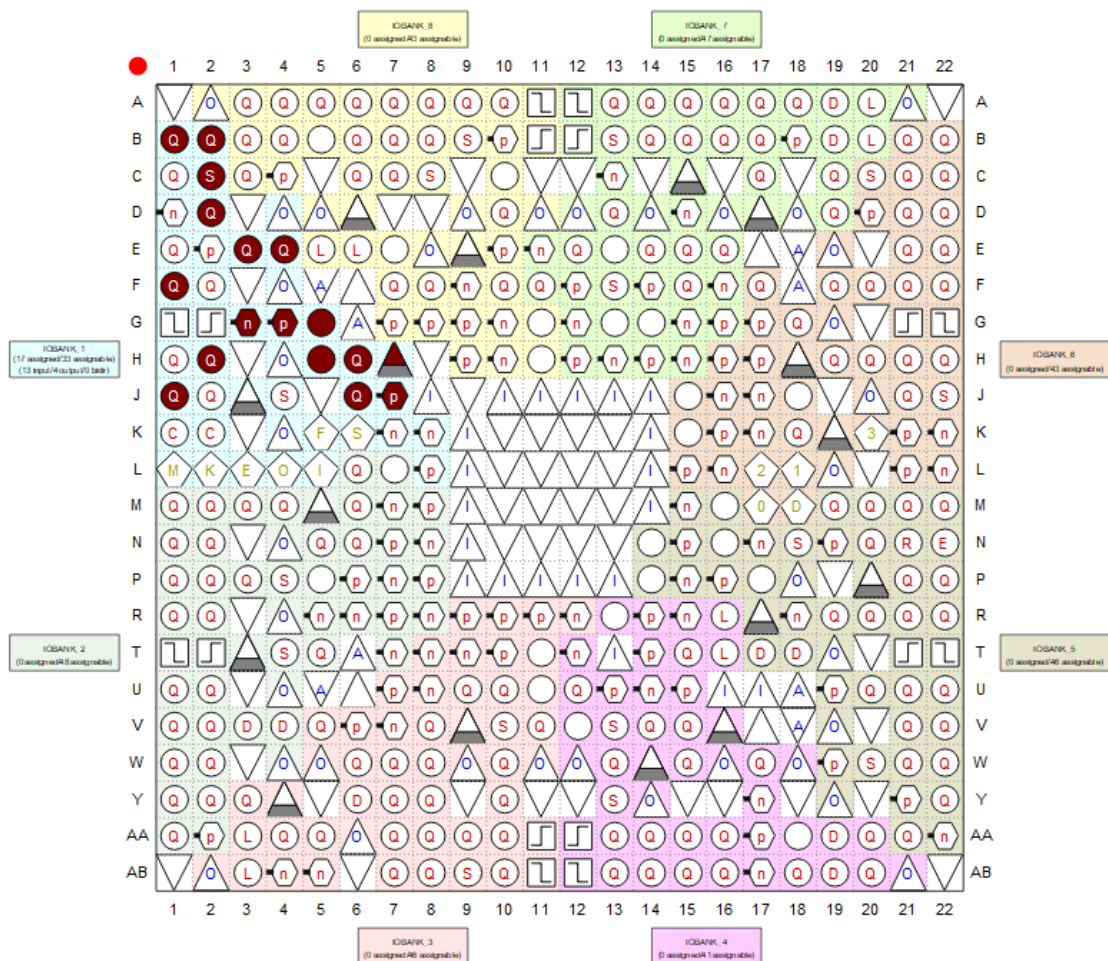


Fonte: TERCASIC TECHNOLOGIES (2011).

APÊNDICE B – Pinagens

Node Name	Direction	Location
 botao1	Input	PIN_F1
 botao2	Input	PIN_G3
 botao3	Input	PIN_H2
 led1	Output	PIN_B1
 led2	Output	PIN_B2
 led3	Output	PIN_C2
 swt[0]	Input	PIN_J6
 swt[1]	Input	PIN_H5
 swt[2]	Input	PIN_H6
 swt[3]	Input	PIN_G4
 swt[4]	Input	PIN_G5
 swt[5]	Input	PIN_J7
 swt[6]	Input	PIN_H7
 swt[7]	Input	PIN_E3
 swt[8]	Input	PIN_E4
 swt[9]	Input	PIN_D2
 y	Output	PIN_J1

Cyclone III - EP3C16F484C6



Fonte: Autor

APÊNDICE C – Tabela Completa com os Dados de Validação do *Perceptron*

(para cada Dado: linha superior – valores em decimal; linha inferior – valores em binário)

ENTRADAS- SAÍDAS	x_1	x_2	x_3	x_4	SAÍDA TEORICA	SAÍDA REAL
Dados 1	6.9	3.1	4.9	1.5	-1	-1
	110.1110011	011.0001100	100.1110011	001.1000000	0	0
Dados 2	5.2	4.1	1.5	0.1	1	1
	101.0011001	100.0001100	001.1000000	000.0001100	1	1
Dados 3	5.7	2.8	4.1	1.3	-1	-1
	101.1011001	010.1100110	100.0001100	001.0100110	0	0
Dados 4	6.1	3	4.6	1.4	-1	-1
	110.0001100	011.0000000	100.1001100	001.0110011	0	0
Dados 5	4.8	3.1	1.6	0.2	1	1
	100.1100110	011.0001100	001.1001100	000.0011001	1	1
Dados 6	6	2.2	4	1	-1	-1
	110.0000000	010.0011001	100.0000000	001.0000000	0	0
Dados 7	4.9	3.1	1.5	0.1	1	1
	100.1110011	011.0001100	001.1000000	000.0001100	1	1
Dados 8	4.6	3.2	1.4	0.2	1	1
	100.1001100	011.0011001	001.0110011	000.0011001	1	1
Dados 9	5.7	4.4	1.5	0.4	1	1
	101.1011001	100.0110011	001.1000000	000.0110011	1	1
Dados 10	6.3	2.5	4.9	1.5	-1	-1
	110.0100110	010.1000000	100.1110011	001.1000000	0	0
Dados 11	7	3.2	4.7	1.4	-1	-1
	111.0000000	011.0011001	100.1011001	001.0110011	0	0
Dados 12	5.1	3.5	1.4	0.2	1	1
	101.0001100	011.1000000	001.0110011	000.0011001	1	1
Dados 13	4.7	3.2	1.3	0.2	1	1
	100.1011001	011.0011001	001.0100110	000.0011001	1	1
Dados 14	4.7	3.2	1.6	0.2	1	1
	100.1011001	011.0011001	001.1001100	000.0011001	1	1
Dados 15	6.6	3	4.4	1.4	-1	-1
	110.1001100	011.0000000	100.0110011	001.0110011	0	0
Dados 16	6.7	3	5	1.7	-1	-1
	110.1011001	011.0000000	101.0000000	001.1011001	0	0
Dados 17	5.1	3.8	1.5	0.3	1	1
	101.0001100	011.1100110	001.1000000	000.0100110	1	1
Dados 18	5.6	3	4.1	1.3	-1	-1
	101.1001100	011.0000000	100.0001100	001.0100110	0	0
Dados 19	4.4	3.2	1.3	0.2	1	1
	100.0110011	011.0011001	001.0100110	000.0011001	1	1
Dados 20	4.5	2.3	1.3	0.3	1	1
	100.1000000	010.0100110	001.0100110	000.0100110	1	1
Dados 21	5	2	3.5	1	-1	-1

	101.0000000	010.0000000	011.1000000	001.0000000	0	0
Dados 22	5	2.3	3.3	1	-1	-1
	101.0000000	010.0100110	011.0100110	001.0000000	0	0
Dados 23	5.8	2.6	4	1.2	-1	-1
	101.1100110	010.1001100	100.0000000	001.0011001	0	0
Dados 24	5.5	2.4	3.8	1.1	-1	-1
	101.1000000	010.0110011	011.1100110	001.0001100	0	0
Dados 25	5.1	3.5	1.4	0.3	1	1
	101.0001100	011.1000000	001.0110011	000.0100110	1	1
Dados 26	5.1	3.7	1.5	0.4	1	1
	101.0001100	011.1011001	001.1000000	000.0110011	1	1
Dados 27	5.7	2.6	3.5	1	-1	-1
	101.1011001	010.1001100	011.1000000	001.0000000	0	0
Dados 28	6.7	3.1	4.7	1.5	-1	-1
	110.1011001	011.0001100	100.1011001	001.1000000	0	0
Dados 29	5	3.4	1.6	0.4	1	1
	101.0000000	011.0110011	001.1001100	000.0110011	1	1
Dados 30	5	3.5	1.3	0.3	1	1
	101.0000000	011.1000000	001.0100110	000.0100110	1	1

Fonte: Autor