Trabalho SEL5752
Rede Neural MultiLayerPerceptron em VHDL

Aluno: Vitor Martins Barbosa
e-mail: vitor_martinsb@usp.br
Mátricula: 12099991
Professor: Maximiliam Luppe
Disciplina: Dispositivos Reconfiguráveis e Linguagem de Descrição de Hardware

Sumário

1.0 INTRODUÇÃO

Define-se como sistema digital uma combinação de componentes que são utilizados para realizar operações lógicas com valores discretos. A representação de sistemas digitais pode ser realizada utilizada através do VHDL (Hardware Description Language), as mesmas são bastantes utilizadas no desenvolvimento de tecnologias voltadas para FPGA (*field-programmable gate array*) que são dispositivos lógicos programáveis, podendo utilizar até mesmo processadores ARM (modelos mais modernos).

Atualmente, o desenvolvimento de tecnologias voltadas para aprendizagem de máquina tem tomado forma em todas áreas da engenharia, uma das ferramentas mais utilizadas entre os classificadores não lineares, está a rede neural artificial, que possui diversas aplicações em desenvolvimento [THEODORIDIS e KOUTROUMBAS, 2009].

Na área de processamento de sinais, é essencial a utilização de circuitos para realização de coletas destes sinais e classificadores para definir o comportamento e realizando o reconhecimento de padrões do sinal, assim, a combinação da VHDL com a rede neural, é bem visto para realizar essa atividade.

1.0 OBJETIVO

Utilizando a linguagem descritiva VHDL, realizar o desenvolvimento de uma lógica de um pacote fixo e realizar a criação de um neurônio.

Para realizar o teste, realizando treinamento utilizando o Matlab, é implementado um sistema de reconhecimento do tipo de câncer de mama (benigno ou maligno), através de uma base de dados aberta [WOLBERG, STREET e MANGASARIAN, 2001], levantado os pesos e as *bias*, e considerando uma configuração válida onde se utilize poucas entradas, seria utilizado um sistema para validar utilizando o VHDL.

3.0 REVISÃO BIBLIOGRÁFICA

3.1 COMPONENTES DAS REDES NEURAIS

3.1.1 NEURÔNIO ARTIFICIAL

O neurônio artificial é um modelo baseado no neurônio biológico no qual o neurônio é visto como um sistema computacional [UFSC]. Para isso, deve-se ter em mente as seguintes proposições:

- A atividade de um neurônio é um processo tudo ou nada;
- Um certo número fixo (>1) de entradas devem ser excitadas dentro de um período de adição latente para excitar um neurônio;
- Único atraso significativo é o atraso sináptico;
- A atividade de qualquer sinapse inibitória previne absolutamente a excitação do neurônio;
- A estrutura das interconexões não muda com o tempo.

Matematicamente falando, considera-se n entradas (x_1, x_2, x_3, ..., x_n) e uma saída (y), sendo que para cada entrada são definidos pesos para cada entrada (w_1, w_2, w_3, ..., w_n) que influenciaram no quanto a entrada influência na saída do sistema. Como no sistema nervoso, as entradas do sistema podem ser definidas como a saída de outro neurônio. A Equação 1 define o que seria o somatório de todas entradas multiplicados pelos seus pesos que é conhecida como net do neurônio.

$$net_i(t) = \sum_{j=1}^{n} w_{ij} x_j(t) \qquad (1)$$

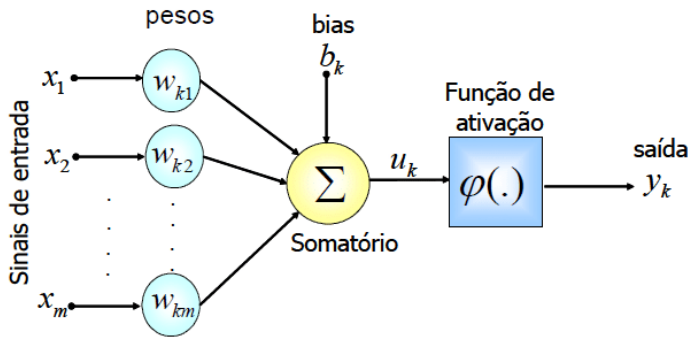
Os pesos w_{ij} são números reais que representam a conexão sináptica da entrada j-ésimo neurônio. Este peso tem dois estados:

- **Excitatória** para w_{ij}>0;
- **Inibitória** para wij<0.

Definido um valor de net_i este valor irá dar origem a saída do neurônio baseado em uma função de ativação, que é responsável por realizar uma transformação não linear considerando as entradas, indicando a probabilidade de que net possuir uma classe definida.

Considerando todo o sistema descrito a representação de um único neurônio pode ser ilustrado como na Figura 1.

Figura 1 - Representação de um neurônio a ser utilizado para uma Rede Neural Artificial.



Fonte: https://www.researchgate.net/figure/Figura-2-Modelo-nao-linear-de-um-neuronio-artificial-Adaptado-de-22_fig1_307834184

3.1.2 FUNÇÃO ATIVAÇÃO

As operações definidas para realizar no neurônio até a multiplicação dos pesos, são consideradas lineares, isso quer dizer que as mesmas realizam a separação das classes utilizando funções lineares, assim, para que esse processo torne possível traçar uma separação de classe não linear, são utilizados função de ativação que podem ter esse comportamento afim de configurar este comportamento, tendo uma variação de valores entre -1 a +1 [CECCON, 2020].

A escolha e variação da função de ativação é um dos papéis fundamentais para obter uma alta taxa de acerto na rede neural e, consequentemente, um projeto com uma boa tomada de decisão para o sistema a ser desenvolvido.

Para o projeto desenvolvido, são utilizadas duas funções de ativação que são apresentadas respectivamente na Equação 2 e Equação 3.

$$\varphi_1(x)\begin{cases} x < -4 \rightarrow f(x) = 0 \\ -4 < x < 0 \rightarrow f(x) = 0.03125x^2 + 0.25x + 0.5 \\ x = 0 \rightarrow f(x) = 0.5 \\ 0 < x < 4 \rightarrow f(x) = -0.03125x^2 + 0.25x + 0.5 \\ x > 4 \rightarrow f(x) = 1 \end{cases} \quad (2)$$

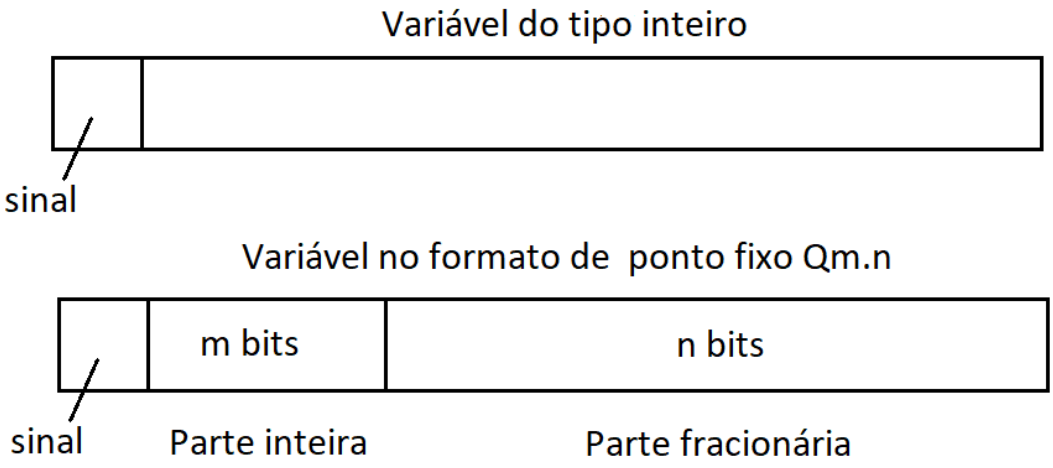
$$\varphi_2(x)\begin{cases} x < -4 \rightarrow f(x) = -1 \\ -4 < x < 0 \rightarrow f(x) = 0.0625x^2 + 0.5x \\ x = 0 \rightarrow f(x) = 0.5 \\ 0 < x < 4 \rightarrow f(x) = -0.0625x^2 + 0.5x \\ x > 4 \rightarrow f(x) = 1 \end{cases} \quad (3)$$

3.2 PONTO FIXO

3.2.1 REPRESENTAÇÃO DOS PONTOS FIXOS

Define-se como valor de ponto fixo é um número real escalonado por um fator específico. A Figura 2 apresenta como é representado o formato da representação numérica real e para *Q-Format* (também conhecida como $Q_{m.n}$). A faixa de valores é definida por $[-2^m, 2^m-2^{-n}]$ e a resolução é 2^{-n}.

Figura 2 – Representação do ponto fixo como *Q-format*.



Fonte: https://www.embarcados.com.br/entendendo-a-aritmetica-em-ponto-fixo/

A representação de um número real (x) para *Q-format* (X) é realizado utilizando a Equação 2, assim como o contrário é verdadeiro, podemos obter o valor de *Q-format*.

$$X = x.2^n \quad (3)$$

$$x = \frac{X}{2^n} \quad (4)$$

Os passos para realizar a conversão para o valor real é definido abaixo:

1. Considerando número real x, o valor binário b da mesma é definida por $b=x.2^F$, onde F é a dimensão francional do número real x;

2. Considerar o arredondamento do valor x (x.round);

3. O valor x.round deve ser convertido para binário (x.round.bin);

4. O valor x.round.bin, necessita de n bits para representar a variável b.

3.2.2 PACOTES DE PONTO FIXO E OPERAÇÕES

Utilizando as técnicas de função para operações entre pontos fixos, inteiros e reais, foi desenvolvido um pacote para realizar o tratamento dos sistemas, afim de que se possa desenvolver um neurônio para ser utilizada na rede neural a ser desenvolvida utilizando o VHDL. A Tabela 1 apresenta as escolhas de delimitação e criação do tipo fixed para utilizar o *Q-format.*

Tabela 1 – Criação de tipos para realização de operações

Constantes e Tipos	Descrição
CONSTANT max_ind: INTEGER := 15;	Definição de número máximo de índices do ponto fixo.
CONSTANT min_ind: INTEGER := -15;	Definição de número de mínimos de índices de ponto fixo.
TYPE fixed IS ARRAY(INTEGER RANGE <>) OF BIT;	Tipo fixo com range de inteiro (equivalente a um bit).
TYPE fixed_vector IS ARRAY(NATURAL RANGE<>, INTEGER RANGE <>) OF BIT;	Vetor de pontos fixos.
TYPE matrix IS ARRAY (NATURAL RANGE <>, NATURAL RANGE <>) OF BIT;	Matrix de pontos de fixos (utilizado para multiplicação).
SUBTYPE fixed_range IS integer RANGE min_ind TO max_ind;	Grupo de bits formando o ponto fix.

A Tabela 2 apresenta as principais funções e as descrições da sua utilidade para as operações desenvolvidas.

Tabela 2 – Funções utilizadas para realização de operações com variáveis do tipo fixo

Funções	Descrição
FUNCTION MAX (arg_L, arg_R: INTEGER) RETURN INTEGER;	Retorna o maior valor entre arg_L e arg_R
FUNCTION MIN (arg_L, arg_R: INTEGER) RETURN INTEGER;	Retorna o menor valor entre arg_L e arg_R
FUNCTION COMP1_FIXED(arg_L: fixed) RETURN fixed;	Retorna o complemento de 1de arg_L
FUNCTION ADD_SUB_FIXED (arg_L, arg_R: fixed; c: BIT) RETURN fixed;	Realiza a soma ou subtração entre arg_L e arg_R
FUNCTION MULT_FIXED (arg_L, arg_R: fixed) RETURN fixed;	Retorna a multiplicação entre arg_L e arg_R
FUNCTION to_fixed (arg_L: INTEGER; max_range: fixed_range := max_ind; min_range: fixed_range := min_ind) RETURN fixed;	Realiza a conversão de inteiro para ponto fixo

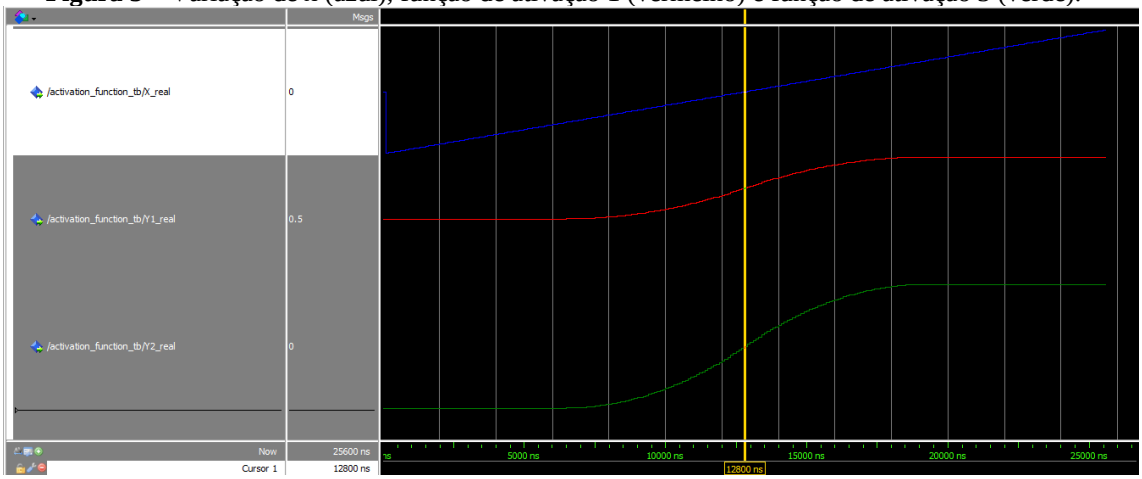
FUNCTION to_integer (arg_L: fixed) RETURN integer;	Realiza a conversão de ponto fixo para inteiro
FUNCTION "+"(arg_L, arg_R: fixed) RETURN fixed; FUNCTION "+"(arg_L: fixed; arg_R: INTEGER) RETURN fixed; FUNCTION "+"(arg_L: INTEGER; arg_R: fixed) RETURN fixed; FUNCTION "+"(arg_L: fixed; arg_R: REAL) RETURN fixed; FUNCTION "+"(arg_L: REAL; arg_R: fixed) RETURN fixed;	Operação soma utilizando o caractere “+”
FUNCTION "-"(arg_L, arg_R: fixed) RETURN fixed; FUNCTION "-"(arg_L: fixed; arg_R: INTEGER) RETURN fixed; FUNCTION "-"(arg_L: INTEGER; arg_R: fixed) RETURN fixed; FUNCTION "-"(arg_L: fixed; arg_R: REAL) RETURN fixed; FUNCTION "-"(arg_L: REAL; arg_R: fixed) RETURN fixed;	Operação subtração utilizando o caractere “-”e realizando o tratamento entre tipos
FUNCTION "*"(arg_L, arg_R: fixed) RETURN fixed; FUNCTION "*"(arg_L: fixed; arg_R: INTEGER) RETURN fixed; FUNCTION "*"(arg_L: INTEGER; arg_R: fixed) RETURN fixed; FUNCTION "*"(arg_L: fixed; arg_R: REAL) RETURN fixed; FUNCTION "*"(arg_L: REAL; arg_R: fixed) RETURN fixed;	Operação de multiplicação utilizando o caractere “*” e realizando o tratamento entre tipos
FUNCTION to_fixed (arg_L: REAL; max_range, min_range: fixed_range) RETURN fixed;	Realiza a conversão de real para ponto fixo
FUNCTION to_real (arg_L: fixed) RETURN REAL;	Realiza a conversão de ponto fixo para real

4.0 Resultados e Discussões

4.1 Funções de Ativação e Teste do Pacote Fixo

Utilizando as funções de ativação apresentada nas Equações e 1 e 2 e variando x entre os valores de -8 a 8 é possível avaliar o comportamento graficamente da função de ativação que é apresentada na Figura 3.

Figura 3 – Variação de x (azul), função de ativação 1 (vermelho) e função de ativação 3 (verde).



Fonte: Autor

Através desta análise de comportamento do gráfico, foi possível averiguar o bom funcionamento do pacote criado para utilizar o ponto fixo, visto que o mesmo utilizou as operações adequadas para realizar a mesma.

A faixa amarela, indica que no ponto onde x=0 as funções ativação atinge os valores esperados para o seu comportamento, assim, é possível utilizar tanto a função ativação quanto o pacote de ponto fixo para realizar a construção do neurônio.

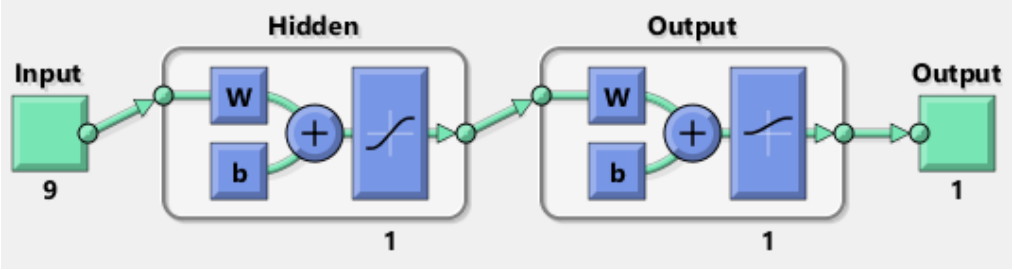
4.2 Rede Neural - MLP

4.2.1 Treinamento

O treinamento da Rede Neural foi realizado utilizando o Matlab, o algoritmo que é apresentado, no anexo I deste documento, realiza o treinamento e, consequentemente, possibilita realizar o levantamento dos pesos e bias para realizar a rede neural artificial.

A Figura 4 apresenta a configuração utilizada, a mesma foi tomada de decisão pelo fato de ter uma taxa de acerto relativamente boa (aproximadamente 85,75% com desvio padrão de 3,00%) e utilizar poucas entradas para serem implementadas no VHDL.

Figura 4 – Configuração utilizada para o realizar o treinamento.



Fonte: Autor

Por conta das dimensões dos dados, foram considerados 9 entradas e, consequentemente, haverá 9 pesos e uma bias na camada oculta e a função de ativação apresentada na Equação 2. Para a camada de saída foram utilizadas apenas um neurônio e a função de ativação apresentada na Equação 3.

4.2.2 RESULTADOS

A Tabela 3 apresentada o resultado dos pesos obtidos utilizando o treinamento, os mesmos são utilizados para realizar a validação, onde são inseridos como entrada no código desenvolvido no VHDL.

Tabela 3 – Dados obtidos a serem inseridos como entradas no VHDL

Camada	**Info.**	**Dados**								
Oculta	**Peso**	-1,3688	-1,3308	-1,3595	-1,3696	-0,2197	-2,3010	-0,0300	-1,5020	1,2731
Oculta	**Bias**	-4,389378								
Saída	**Peso**	-2,800294								
Saída	**Bias**	-0,830327								

Através dessa informação e do algoritmo desenvolvido em VHDL, é possível inserir qualquer dado de entrada para realizar a validação do tipo de câncer. A Tabela 5 apresenta essa informação e a saída da informação levantada, indicando o tipo do câncer, onde 1, é maligno e 0 é benigno. Como é a realização da validação, a entrada 1 foi inserido uma informação rotulada e, a mesma indica um câncer benigno, enquanto a entrada 2 é o dado de uma informação maligno, em ambos os casos, as informações não foram utilizadas no treinamento, afim de evitar qualquer tipo de redundância no sistema.

Tabela 4 – Resultados obtidos da validação utilizando o VHDL

Tipo de Dado	**Dados**								
Entrada 1	0,0000	0,0000	0,0000	0,0000	0,5000	0,0000	1,0000	0,0000	0,0000
Saída 1	0.6551611								
Entrada 2	0,0000	1,0000	0,7143	0,5714	0, 4286	0, 8571	0,8571	0,0000	0, 7143
Saída 2	0,1284179								

Os resultados se mostraram satisfatório, onde os algoritmos desenvolvidos desempenharam o seu papel, sendo possível implementar a Rede Neural para validar no VHDL e o mesmo atua conforme esperado, identificando um valor de saída 1 que apresenta o posicionamento claro de uma classe dado a classificação realizada. Entretanto a busca pela melhor configuração se faz necessário afim de tornar a apresentação de um resultado mais seguro

5.0 CONCLUSÃO

Através desse sistema desenvolvido, foi possível avaliar corretamente e com clareza o sistema, o mesmo está disponível em um repositório do github para realização de testes. Os resultados se mostraram promissores, onde os algoritmos desenvolvidos se comportaram da maneira adequada.

Vale ressaltar que há melhorias a podem ser realizadas para nesse sistema para trabalhos futuros, talvez utilizando um classificador linear, como o método baseados nos métodos dos quadrados mínimos, tornando possível realizar operações de treinamento e validação dentro da linguagem VHDL, possibilitando até mesmo o levantamento de pesos naquele sistema.

6.0 ANEXO

6.1 ANEXO I – TRATAMENTO DE BASE DE DADOS E TREINAMENTO

```
clear all;
close all;

filename = 'bcw_info.txt';
fileID = fopen(filename);
Data = textscan(fileID,'%f %f %f %f %f %f %f %f %f %f %f');
ID_Data = Data{1};
Rot_Data = Data{end};
num_Data = length(ID_Data);
Data_benign = zeros(1,9);
Data_malignant = zeros(1,9);
Rot_benign = zeros(1,1);
Rot_malignant = zeros(1,1);
Data_info = zeros(num_Data,9);

for k = 2:10
    Data_info(:,k-1)=Data{k};
end

k_b=0;
k_m=0;
for k = 1:length(Data_info)

    A=Data_info(k,:);
    B = (A-min(A))/(max(A)-min(A));
    Data_info(k,:)=B;

    if Rot_Data(k)==2
        k_b=k_b+1;
        Rot_Data(k)=0;
        Data_benign(k_b,:)=B;
        Rot_benign(k_b)=0;
    elseif Rot_Data(k)==4
        k_m=k_m+1;
        Rot_Data(k)=1;
        Data_malignant(k_m,:)=B;
        Rot_malignant(k_m)=1;
    end
end

Data_train = [Data_benign(1:150,:);Data_malignant(1:150,:)]';
Data_val = [Data_benign(151:170,:);Data_malignant(151:170,:)]';
Rot_train = [Rot_benign(1:150) Rot_malignant(1:150)];
Rot_val = [Rot_benign(151:170) Rot_malignant(151:170)];
save('x.mat','Data_train');
save('t.mat','Rot_train');

x = Data_train;
t = Rot_train;

trainFcn = 'trainscg';

hiddenLayerSize = 1;
net = patternnet(hiddenLayerSize, trainFcn);
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

[net,tr] = train(net,x,t);

y = net(Data_val);
e = gsubtract(Rot_val,y);
performance = perform(net,Rot_val,y);
tind = vec2ind(Rot_val);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);
```

```
view(net)
w1 = net.IW{1};
fprintf('\n Peso Camada Oculta: ');
fprintf(' %f ',w1);
w2 = net.LW{2};
fprintf('\n Peso Camada Saída: ');
fprintf(' %f ',w2);
b1 = net.b{1};
fprintf('\n Bias Camada Oculta: ');
fprintf(' %f ',b1);
b2 = net.b{2};
fprintf('\n Bias Camada Saída: ');
fprintf(' %f ',b2);

cont_0=0;
cont_1=0;
for k = 1:length(y)
   if k<length(y)*0.5
       if y(k)<0.5
          cont_0=cont_0+1;
       end
   else
      if y(k)>0.5
          cont_1=cont_1+1;
      end
   end
end
perc_res=100*(cont_0+cont_1)/length(y);
fprintf('\n Taxa de acerto: %f \% \n',perc_res);

y_teste_B=net(Data_val(:,1));
y_teste_M=net(Data_val(:,end));
```

6.2 ANEXO II – PACOTE PONTO FIXO

```
----------------------------
--Vitor Martins Barbosa

--12099991

--Descrição:

-- Exercío de Aula 12
----------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.MATH_REAL.ALL;

PACKAGE fixed_package IS
        CONSTANT N_BIT: INTEGER := 10;
        CONSTANT max_ind: INTEGER := 15;
        CONSTANT min_ind: INTEGER := -15;
        TYPE fixed IS ARRAY(INTEGER RANGE <>) OF BIT;
        TYPE matrix IS ARRAY (NATURAL RANGE <>, NATURAL RANGE <>) OF BIT;
        SUBTYPE fixed_range IS integer RANGE min_ind TO max_ind;
        FUNCTION MAX (arg_L, arg_R: INTEGER) RETURN INTEGER;
        FUNCTION MIN (arg_L, arg_R: INTEGER) RETURN INTEGER;
        FUNCTION COMP1_FIXED(arg_L: fixed) RETURN fixed;
        FUNCTION ADD_SUB_FIXED (arg_L, arg_R: fixed; c: BIT) RETURN fixed;
        FUNCTION MULT_FIXED (arg_L, arg_R: fixed) RETURN fixed;
        FUNCTION to_fixed (arg_L: INTEGER; max_range: fixed_range := max_ind;
                                                        min_range: fixed_range := min_ind) RETURN fixed;
        FUNCTION to_integer (arg_L: fixed) RETURN integer;
        FUNCTION "+"(arg_L, arg_R: fixed) RETURN fixed;
        FUNCTION "+"(arg_L: fixed; arg_R: INTEGER) RETURN fixed;
        FUNCTION "+"(arg_L: INTEGER; arg_R: fixed) RETURN fixed;
        FUNCTION "+"(arg_L: fixed; arg_R: REAL) RETURN fixed;
        FUNCTION "+"(arg_L: REAL; arg_R: fixed) RETURN fixed;
        FUNCTION "-"(arg_L, arg_R: fixed) RETURN fixed;
        FUNCTION "-"(arg_L: fixed; arg_R: INTEGER) RETURN fixed;
        FUNCTION "-"(arg_L: INTEGER; arg_R: fixed) RETURN fixed;
        FUNCTION "-"(arg_L: fixed; arg_R: REAL) RETURN fixed;
```

```
		FUNCTION "-"(arg_L: REAL; arg_R: fixed) RETURN fixed;
		FUNCTION "*"(arg_L, arg_R: fixed) RETURN fixed;
		FUNCTION "*"(arg_L: fixed; arg_R: INTEGER) RETURN fixed;
		FUNCTION "*"(arg_L: INTEGER; arg_R: fixed) RETURN fixed;
		FUNCTION "*"(arg_L: fixed; arg_R: REAL) RETURN fixed;
		FUNCTION "*"(arg_L: REAL; arg_R: fixed) RETURN fixed;
		FUNCTION to_fixed (arg_L: REAL; max_range, min_range: fixed_range) RETURN fixed;
		FUNCTION to_real (arg_L: fixed) RETURN REAL;
		FUNCTION Activation (X : fixed) RETURN fixed;
END fixed_package;

PACKAGE BODY fixed_package IS
--Internas
		--Auxiliares aritmeticas
		--MAX
			FUNCTION MAX (arg_L, arg_R: INTEGER) RETURN INTEGER IS
			BEGIN
				IF arg_L > arg_R THEN
					return arg_L;
				ELSE
					return arg_R;
				END IF;
			END MAX;
		--MIN
			FUNCTION MIN (arg_L, arg_R: INTEGER) RETURN INTEGER IS
			BEGIN
				IF arg_L < arg_R THEN
					RETURN arg_L;
				ELSE
					RETURN arg_R;
				END IF;
			END MIN;
		--COMP1_FIXED
			FUNCTION COMP1_FIXED(arg_L: fixed) RETURN fixed IS
				VARIABLE arg_L_COMP1: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
			BEGIN
				FOR k in arg_L'LOW TO arg_L'HIGH LOOP
					arg_L_COMP1(k) := NOT(arg_L(k));
				END LOOP;
				RETURN arg_L_COMP1;
			END COMP1_FIXED;
		--ADD_SUB_FIXED
			FUNCTION ADD_SUB_FIXED (arg_L, arg_R: fixed; c: BIT) RETURN fixed IS
				VARIABLE s: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
				VARIABLE v: BIT;
			BEGIN
				v := c;
				FOR k in arg_L'LOW TO arg_L'HIGH LOOP
					s(k) := (arg_L(k) XOR arg_R(k)) XOR v;
					v := (arg_L(k) AND arg_R(k)) OR (v AND (arg_L(k) OR arg_R(k)));
				END LOOP;
				RETURN s;
			END ADD_SUB_FIXED;

		--MULT_FIXED
			FUNCTION MULT_FIXED (arg_L, arg_R: fixed) return fixed IS
				CONSTANT M: INTEGER := arg_L'LENGTH;
				CONSTANT N: INTEGER := arg_R'LENGTH;
				VARIABLE Mij: matrix(0 TO M-1, 0 TO M+N-1);
				VARIABLE Cij: matrix(0 TO M-1, 0 TO M+N);
				VARIABLE Pij: matrix(0 TO M, 0 TO M+N);
				VARIABLE blinha: fixed(M+N-1 DOWNTO 0);
				VARIABLE P: fixed(M+N-1 DOWNTO 0);
			BEGIN

				blinha := (M+N-1 downto N => '0') & arg_R;

				initCij: FOR i IN 0 TO M-1 LOOP
					Cij(i, 0) := '0';
				END LOOP initCij;

				initPij1: FOR i IN 0 to M LOOP
					Pij(i, 0) := '0';
				END LOOP initPij1;

				initPij2: FOR j IN 1 TO M+N-1 LOOP
```

```
                                        Pij(m, j) := '0';
                                END LOOP initPij2;

                                Mijcol: FOR i IN M-1 DOWNTO 0 LOOP
                                        Mijrow: FOR j IN M+N-1 DOWNTO 0 LOOP
                                                Mij(i,j) := arg_L(i) and blinha(j);
                                        END LOOP Mijrow;
                                END LOOP Mijcol;

                                Pijcol: FOR i IN M-1 DOWNTO 0 LOOP
                                        Pijrow: FOR j IN 0 TO M+N-1 LOOP
                                                Pij(i,j+1) := Pij(i+1,j) XOR Mij(i,j) XOR Cij(i,j);
                                                Cij(i,j+1) := (Pij(i+1,j) AND (Mij(i,j) OR Cij(i,j))) OR (Mij(i,j) AND
Cij(i,j));
                                                --SOMA: som PORT MAP(Pij(i+1,j),Cij(i,j),Mij(i,j),Pij(i,j+1),Cij(i,j+1));
                                        END LOOP Pijrow;
                                END LOOP Pijcol;

                                initPi: FOR i IN M+N-1 DOWNTO 0 LOOP
                                        P(i) := Pij(0,i+1);
                                END LOOP initPi;

                                RETURN P;
                        END MULT_FIXED;

--Externas
        --Conversao de tipo
                --to_fixed
                FUNCTION to_fixed (arg_L: INTEGER; max_range: fixed_range := max_ind;
                                                                min_range: fixed_range := min_ind) RETURN
fixed IS
                        VARIABLE int_res        : integer := arg_L;

                        VARIABLE res                    : fixed (max_range DOWNTO min_range);
                        VARIABLE res_0                  : fixed (max_range DOWNTO min_range);

                BEGIN
                        FOR k IN min_range TO max_range LOOP
                                res(k) := '0';
                                res_0(k) := '0';
                        END LOOP;

                        IF int_res<0 THEN
                                int_res:=int_res*(-1);
                        END IF;

                        IF int_res = 1 THEN
                                res(min_range) := '1';
                        ELSE
                                FOR k IN min_range TO max_range-1 LOOP
                                        IF int_res >= 1 AND int_res >= -1 THEN
                                                IF (int_res MOD 2) = 0 THEN
                                                        res(k) := '0';
                                                ELSE
                                                        res(k) := '1';
                                                END IF;
                                                int_res := INTEGER(int_res / 2);
                                        ELSE
                                                res(k) := '0';
                                        END IF;

                                END LOOP;
                        END IF;

                        IF arg_L < 0 THEN
                                res := ADD_SUB_FIXED(COMP1_FIXED(res),res_0,'1');
                        END IF;

                        RETURN res;
                END     to_fixed;

                --to_integer
                FUNCTION to_integer (arg_L: fixed) RETURN INTEGER IS
                        VARIABLE v_arg_L: fixed(arg_L'RANGE);
                        VARIABLE res: INTEGER := 0;
                BEGIN
```

```
		v_arg_L := arg_L;

		IF arg_L(arg_L'LEFT) = '1' THEN
			v_arg_L := COMP1_FIXED(arg_L);
		END IF;

		FOR k IN v_arg_L'LEFT DOWNTO 0 LOOP
			IF v_arg_L(k) = '1' THEN
				res := (2**k) + res;
			END IF;
		END LOOP;

		IF arg_L(arg_L'LEFT) = '1' THEN
			res := -1*(res+1);
		END IF;

		RETURN res;
END to_integer;
--Aritmeticas
-- +
FUNCTION "+" (arg_L, arg_R: fixed) RETURN fixed IS
		VARIABLE res: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
BEGIN
		res := ADD_SUB_FIXED(arg_L, arg_R, '0');
		RETURN res;
END "+";
-- -
FUNCTION "-" (arg_L, arg_R: fixed) RETURN fixed IS
		VARIABLE NEG_arg_R:  fixed(arg_R'HIGH DOWNTO arg_R'LOW);
		VARIABLE res: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
BEGIN
		NEG_arg_R := COMP1_FIXED(arg_R);
		res := ADD_SUB_FIXED(arg_L, NEG_arg_R, '1');
		RETURN res;
END "-";

--"+"
FUNCTION "+"(arg_L: fixed; arg_R: INTEGER) RETURN fixed IS
		VARIABLE res: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
		VARIABLE arg_F: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
BEGIN
		arg_F := to_fixed(arg_R);
		RETURN arg_L + arg_F;
END "+";

FUNCTION "+" (arg_L: INTEGER; arg_R: fixed) RETURN fixed IS
		VARIABLE res: fixed(arg_R'HIGH DOWNTO arg_R'LOW);
		VARIABLE arg_F: fixed(arg_R'HIGH DOWNTO arg_R'LOW);
BEGIN
		arg_F := to_fixed(arg_L);
		RETURN arg_F + arg_R;
END "+";

FUNCTION "+" (arg_L: REAL; arg_R: fixed) RETURN fixed IS
		VARIABLE res: fixed(arg_R'HIGH DOWNTO arg_R'LOW);
		VARIABLE arg_F: fixed(arg_R'HIGH DOWNTO arg_R'LOW);
BEGIN
		arg_F := to_fixed(arg_L,arg_R'HIGH,arg_R'LOW);
		RETURN arg_F + arg_R;
END "+";

FUNCTION "+" (arg_L: fixed; arg_R: REAL) RETURN fixed IS
		VARIABLE res: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
		VARIABLE arg_F: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
BEGIN
		arg_F := to_fixed(arg_R,arg_L'HIGH,arg_L'LOW);
		RETURN arg_F + arg_R;
END "+";

--"-"
FUNCTION "-" (arg_L: fixed; arg_R: INTEGER) RETURN fixed IS
		VARIABLE res: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
		VARIABLE arg_F: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
BEGIN
		arg_F := to_fixed(arg_R);
```

```
                        RETURN arg_L - arg_F;
                END "-";

                FUNCTION "-" (arg_L: INTEGER; arg_R: fixed) RETURN fixed IS
                        VARIABLE res: fixed(arg_R'HIGH DOWNTO arg_R'LOW);
                        VARIABLE arg_F: fixed(arg_R'HIGH DOWNTO arg_R'LOW);
                BEGIN
                        arg_F := to_fixed(arg_L);
                        RETURN arg_F - arg_R;
                END "-";

                FUNCTION "-" (arg_L: REAL; arg_R: fixed) RETURN fixed IS
                        VARIABLE res: fixed(arg_R'HIGH DOWNTO arg_R'LOW);
                        VARIABLE arg_F: fixed(arg_R'HIGH DOWNTO arg_R'LOW);
                BEGIN
                        arg_F := to_fixed(arg_L,arg_R'HIGH,arg_R'LOW);
                        RETURN arg_F - arg_R;
                END "-";

                FUNCTION "-" (arg_L: fixed; arg_R: REAL) RETURN fixed IS
                        VARIABLE res: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
                        VARIABLE arg_F: fixed(arg_L'HIGH DOWNTO arg_L'LOW);
                BEGIN
                        arg_F := to_fixed(arg_R,arg_L'HIGH,arg_L'LOW);
                        RETURN arg_L - arg_F;
                END "-";

                FUNCTION "*" (arg_L, arg_R: fixed) RETURN fixed IS
                        VARIABLE res_ext: fixed((arg_R'LENGTH + arg_L'LENGTH - 1) DOWNTO 0);
                        VARIABLE res_ext_2: fixed((arg_R'HIGH + arg_L'HIGH + 1) DOWNTO (arg_R'LOW +
arg_L'LOW));
                        VARIABLE res: fixed((arg_R'HIGH + arg_L'HIGH+1) DOWNTO (arg_R'LOW +
arg_L'LOW));
                        VARIABLE res_0: fixed((arg_R'LENGTH + arg_L'LENGTH)-1 DOWNTO 0);
                        VARIABLE res_0_R: fixed ((arg_R'LENGTH - 1) DOWNTO 0);
                        VARIABLE res_0_L: fixed ((arg_L'LENGTH - 1) DOWNTO 0);
                        VARIABLE arg_L_shift: fixed ((arg_L'LENGTH - 1) DOWNTO 0);
                        VARIABLE arg_R_shift: fixed ((arg_R'LENGTH - 1) DOWNTO 0);
                        VARIABLE res_comp: fixed (arg_L'RANGE);
                BEGIN
                        arg_L_shift := arg_L;
                        arg_R_shift := arg_R;

                        FOR k IN (arg_R_shift'LENGTH-1) DOWNTO 0 LOOP
                                res_0_R(k) := '0';
                        END LOOP;

                        FOR k IN (arg_L_shift'LENGTH-1) DOWNTO 0 LOOP
                                res_0_L(k) := '0';
                        END LOOP;

                        FOR k IN (arg_R_shift'LENGTH + arg_L_shift'LENGTH - 1) DOWNTO 0 LOOP
                                res_0(k) := '0';
                        END LOOP;

                        IF arg_L_shift(arg_L_shift'LENGTH - 1) = '1' AND arg_R_shift(arg_R_shift'LENGTH - 1) =
'1'  THEN
                                res_ext :=
MULT_FIXED(ADD_SUB_FIXED(COMP1_FIXED(arg_L_shift),res_0_L,'1'),
ADD_SUB_FIXED(COMP1_FIXED(arg_R_shift),res_0_R,'1'));
                        END IF;

                        IF arg_L_shift(arg_L_shift'LENGTH - 1) = '1' AND arg_R_shift(arg_R_shift'LENGTH - 1) =
'0'  THEN
                                res_ext :=
MULT_FIXED(ADD_SUB_FIXED(COMP1_FIXED(arg_L_shift),res_0_L,'1'), arg_R_shift);
                                res_ext := ADD_SUB_FIXED(COMP1_FIXED(res_ext),res_0,'1');
                        END IF;

                        IF arg_L_shift(arg_L_shift'LENGTH - 1) = '0' AND arg_R_shift(arg_R_shift'LENGTH - 1) =
'1'  THEN
                                res_ext :=
MULT_FIXED(arg_L_shift,ADD_SUB_FIXED(COMP1_FIXED(arg_R_shift),res_0_R,'1'));
                                res_ext := ADD_SUB_FIXED(COMP1_FIXED(res_ext),res_0,'1');
                        END IF;
```

```
                                IF arg_L_shift(arg_L_shift'LENGTH - 1) = '0' AND arg_R_shift(arg_R_shift'LENGTH - 1) =
'0'  THEN
                                        res_ext := MULT_FIXED(arg_L_shift,arg_R_shift);
                                END IF;

--                              FOR k IN res_comp'HIGH DOWNTO res_comp'LOW LOOP
--                                      res_comp(k) := res_ext(k + 1 + res_comp'LENGTH);
--                              END LOOP;
                                res_ext_2 := res_ext;
                                res_comp := res_ext_2(res_comp'HIGH DOWNTO res_comp'LOW);
                                RETURN res_comp;

                        END "*";

                        FUNCTION "*"(arg_L: fixed; arg_R: INTEGER) RETURN fixed IS
                                CONSTANT M: INTEGER := arg_L'LENGTH;
                                CONSTANT N: INTEGER := arg_L'LENGTH;
                                CONSTANT arg_R_real: REAL := REAL(arg_R);
                                VARIABLE res: fixed(arg_L'RANGE);
                                VARIABLE arg_F: fixed(arg_L'RANGE);
                        BEGIN
                                arg_F := to_fixed(arg_R_real,arg_L'HIGH,arg_L'LOW);
                                res := arg_L * arg_F;
                                RETURN res;
                        END "*";

                        FUNCTION "*"(arg_L: INTEGER; arg_R: fixed) RETURN fixed IS
                                CONSTANT M: INTEGER := arg_R'LENGTH;
                                CONSTANT N: INTEGER := arg_R'LENGTH;
                                CONSTANT arg_L_real: REAL := REAL(arg_L);
                                VARIABLE res: fixed(arg_R'RANGE);
                                VARIABLE arg_F: fixed(arg_R'RANGE);
                        BEGIN
                                arg_F := to_fixed(arg_L_real,arg_R'HIGH,arg_R'LOW);
                                res := arg_F * arg_R;
                                RETURN res;
                        END "*";

                        FUNCTION "*"(arg_L: fixed; arg_R: REAL) RETURN fixed IS
                                CONSTANT M: INTEGER := arg_L'LENGTH;
                                CONSTANT N: INTEGER := arg_L'LENGTH;
                                VARIABLE arg_F: fixed(arg_L'RANGE);
                                VARIABLE res: fixed(arg_L'RANGE);
                        BEGIN
                                arg_F := to_fixed(arg_R,arg_L'HIGH,arg_L'LOW);
                                res := arg_L * arg_F;
                                RETURN res;
                        END "*";

                        FUNCTION "*"(arg_L: REAL; arg_R: fixed) RETURN fixed IS
                                CONSTANT M: INTEGER := arg_R'LENGTH;
                                CONSTANT N: INTEGER := arg_R'LENGTH;
                                VARIABLE res: fixed(arg_R'RANGE);
                                VARIABLE arg_F: fixed(arg_R'RANGE);
                        BEGIN
                                arg_F := to_fixed(arg_L,arg_R'HIGH,arg_R'LOW);
                                res := arg_F * arg_R;
                                RETURN res;
                        END "*";

                        --to_real
                        FUNCTION to_real (arg_L: fixed) RETURN REAL IS
                                VARIABLE res: REAL := 0.0;
                                VARIABLE arg_F: fixed(arg_L'RANGE);
                                VARIABLE res_0_L: fixed(arg_L'RANGE);
                        BEGIN

                                FOR k IN arg_L'RANGE LOOP
                                        res_0_L(k) := '0';
                                END LOOP;

                                arg_F := arg_L;
                                IF arg_L(arg_L'HIGH)='1' THEN
                                        arg_F := ADD_SUB_FIXED(COMP1_FIXED(arg_F),res_0_L,'1');
                                END IF;
```

```
                FOR k IN arg_F'HIGH - 1 DOWNTO arg_F'LOW LOOP
                        IF arg_F(k) = '1' THEN
                                res := res + (2.0**k);
                        END IF;
                END LOOP;

                IF arg_L(arg_L'HIGH)='1' THEN
                        res := res * (-1.0);
                END IF;

                RETURN res;
        END to_real;

--to_fixed
        FUNCTION to_fixed (arg_L: REAL; max_range, min_range: fixed_range) RETURN fixed IS
                CONSTANT abs_real : REAL := ABS(arg_L);
                CONSTANT min_range_real : REAL := 2.0**(min_range);
                CONSTANT max_range_real : REAL := 2.0**(max_range);
                VARIABLE int_part : REAL := 0.0;
                VARIABLE frac_part: REAL := 0.0;
                VARIABLE int_res: INTEGER := 0;
                VARIABLE res: fixed(max_range DOWNTO min_range);
                VARIABLE res_0: fixed(max_range DOWNTO min_range);
        BEGIN

                IF abs_real < min_range_real OR arg_L=0.0 THEN
                        FOR k IN max_range DOWNTO min_range LOOP
                                res(k) := '0';
                        END LOOP;
                ELSIF arg_L >= max_range_real THEN
                        FOR k IN max_range DOWNTO min_range LOOP
                                res(k) := '1';
                        END LOOP;
                        res(max_range):='0';
                ELSIF arg_L <= -max_range_real THEN
                        FOR k IN max_range DOWNTO min_range LOOP
                                res(k) := '0';
                        END LOOP;
                        res(max_range):='1';
                ELSE
                        int_part := floor(abs_real);
                        frac_part := abs_real - int_part;
                        int_res := INTEGER(int_part);

                        FOR k IN res'RANGE LOOP
                                res(k) := '0';
                        END LOOP;

                        FOR k IN 0 TO res'HIGH-1 LOOP
                                IF int_res >= 1 AND int_res >= -1 THEN
                                        IF (int_res MOD 2) = 0 THEN
                                                res(k) := '0';
                                        ELSE
                                                res(k) := '1';
                                        END IF;
                                        int_res := INTEGER(int_res / 2);
                                ELSE
                                        res(k) := '0';
                                END IF;
                        END LOOP;

                        FOR k IN -1 DOWNTO res'LOW LOOP
                                IF frac_part /= 1.0 THEN
                                        frac_part := frac_part * 2.0;
                                END IF;

                                IF frac_part > 1.0 THEN
                                        res(k) := '1';
                                        frac_part := frac_part - 1.0;
                                ELSIF frac_part < 1.0 THEN
                                        res(k) := '0';
                                ELSIF frac_part = 1.0 THEN
                                        res(k) := '1';
                                        EXIT;
                                END IF;
```

```
                                        END LOOP;

                                END IF;

                                FOR k in res'RANGE LOOP
                                        res_0(k):='0';
                                END LOOP;

                                IF arg_L < 0.0 THEN
                                        res := ADD_SUB_FIXED(COMP1_FIXED(res),res_0,'1');
                                END IF;

                                RETURN res;
                        END to_fixed;

                        -- Parâmetro de entrada:
                        --      X : fixed [0,1]
                        -- Parâmetro de saída:
                        --      SIG : fixed [0,1]
                        -- Retorna:
                        --  / Se    X < -4      SIG =  0.0
                        --  | Se -4 < X < 0     SIG = +0.03125*X**2+0.25*X+0.5
                        --  <  Se    X = 0      SIG = +0.5
                        --  | Se  0 < X < 4     SIG = -0.03125*X**2+0.25*X+0.5
                        --  \ Se    X >= 4      SIG = +1.0

                        FUNCTION Activation (X : fixed) RETURN fixed IS
                                CONSTANT X_LEFT: INTEGER := X'LEFT;
                                CONSTANT X_RIGHT: INTEGER := X'RIGHT;
                                CONSTANT a2p: fixed(X'RANGE) := to_fixed(0.03125, X_LEFT, X_RIGHT);
                                CONSTANT a2n: fixed(X'RANGE) := to_fixed(-0.03125, X_LEFT, X_RIGHT);
                                CONSTANT a1: fixed(X'RANGE) := to_fixed(0.25000, X_LEFT, X_RIGHT);
                                CONSTANT a0: fixed(X'RANGE) := to_fixed(0.50000, X_LEFT, X_RIGHT);
                                CONSTANT maxSIG: fixed(X'RANGE) := to_fixed(0.99999, X_LEFT, X_RIGHT);
                                CONSTANT minSIG: fixed(X'RANGE) := to_fixed(0.00000, X_LEFT, X_RIGHT);
                                VARIABLE SIG: fixed(X'RANGE);
                        BEGIN
                                IF to_real(X) >= 4.0 THEN                       -- Se    X >= 4 SIG = +1.0
                                        SIG := maxSIG;
                                ELSIF to_real(X) < -4.0 THEN                    -- Se    X < -4 SIG =  0.0
                                        SIG := minSIG;
                                ELSIF to_real(X) < 0.0 THEN                     -- Se -4 < X < 0  SIG = (+0.03125*X+0.25)*X+0.5
                                        SIG := (((a2p * X) + a1) * X) + a0;
                                ELSE                                            -- Se  0 < X < 4  SIG = (-0.03125*X+0.25)*X+0.5
                                        SIG := (((a2n * X )+ a1) * X) + a0;
                                END IF;
                                RETURN SIG;
                        END;

END fixed_package;
```

6.3 ANEXO III – PACOTE NEURON

```
USE work.fixed_package.all;

PACKAGE neuron_pkg IS

        TYPE fixed_vector IS ARRAY(NATURAL RANGE<>, INTEGER RANGE <>) OF BIT;

        FUNCTION Activation1(X : fixed) return fixed;
        FUNCTION Activation2(X : fixed) return fixed;

END PACKAGE;

PACKAGE BODY neuron_pkg IS


        -- Funcao de Transferencia (ou Ativacao), baseada no trabalho:
        --  I. Tsmots, O. Skorokhoda and V. Rabyk, "Hardware Implementation of Sigmoid Activation Functions using
        --  FPGA", IEEE 15th Int. Conf. on the Experience of Designing and Application of CAD Systems (CADSM),
        --  Polyana, Ukraine, 2019, pp. 34-38, doi: 10.1109/CADSM.2019.8779253.
        --  ref: https://ieeexplore.ieee.org/abstract/document/8779253
        --
        -- Parametro de entrada:
```

```
        --              X : fixed [0,1]
        -- Parametro de saida:
        --              SIG : fixed [0,1]
        -- Retorna:
        --  / Se      X < -4        SIG =  0.0
        --  | Se -4 < X < 0        SIG = +0.03125*X**2+0.25*X+0.5
        --  <  Se      X = 0        SIG = +0.5
        --  | Se  0 < X < 4        SIG = -0.03125*X**2+0.25*X+0.5
        --  \ Se      X >= 4       SIG = +1.0

        FUNCTION Activation1 (X : fixed) RETURN fixed IS
                CONSTANT X_LEFT: integer := X'left;
                CONSTANT X_RIGHT: integer := X'right;
                CONSTANT a2p: fixed(X'range) := to_fixed(0.03125, X_LEFT, X_RIGHT);
                CONSTANT a2n: fixed(X'range) := to_fixed(-0.03125, X_LEFT, X_RIGHT);
                CONSTANT a1: fixed(X'range) := to_fixed(0.25000, X_LEFT, X_RIGHT);
                CONSTANT a0: fixed(X'range) := to_fixed(0.50000, X_LEFT, X_RIGHT);
                CONSTANT maxSIG: fixed(X'range) := to_fixed(0.99997, X_LEFT, X_RIGHT);
                CONSTANT minSIG: fixed(X'range) := to_fixed(0.00000, X_LEFT, X_RIGHT);
                VARIABLE SIG: fixed(X'range);
        BEGIN
                IF to_integer(X) >= 4 THEN                      -- Se      X >= 4 SIG = +1.0
                        SIG := maxSIG;
                ELSIF to_integer(X) < -4 THEN-- Se      X < -4 SIG =  0.0
                        SIG := minSIG;
                ELSIF to_integer(X) < 0 THEN  -- Se -4 < X < 0  SIG = (+0.03125*X+0.25)*X+0.5
                        SIG := (0.03125 * (X * X)) + (0.25000 * X) + a0;
                ELSE                                                                    -- Se  0 < X < 4  SIG = (-
0.03125*X+0.25)*X+0.5
                        SIG := ((-0.03125) * (X * X)) + (0.25000 * X) + a0;
                END IF;
                RETURN SIG;
        END;

        -- Funcao de Transferencia (ou Ativacao), adaptada do trabalho:
        --  I. Tsmots, O. Skorokhoda and V. Rabyk, "Hardware Implementation of Sigmoid Activation Functions using
FPGA,", 2019
        --  IEEE 15th Int. Conf. on the Experience of Designing and Application of CAD Systems (CADSM), Polyana,
Ukraine, 2019
        --  pp. 34-38, doi: 10.1109/CADSM.2019.8779253.
        --  ref: https://ieeexplore.ieee.org/abstract/document/8779253
        -- Parametro de entrada:
        --              X : fixed
        -- Parametro de saida:
        --              SIG : fixed
        -- Retorna:
        --  / Se      X < -4        SIG = -1.0
        --  | Se -4 < X < 0        SIG = +0.0625*X**2+0.5*X
        --  <  Se      X = 0        SIG =  0.0
        --  | Se  0 < X < 4        SIG = -0.0625*X**2+0.5*X
        --  \ Se      X > 4        SIG = +1.0
        FUNCTION Activation2 (X : fixed) RETURN fixed IS
                CONSTANT X_LEFT: integer := X'left;
                CONSTANT X_RIGHT: integer := X'right;
                CONSTANT a2p: fixed(X'RANGE) := to_fixed(0.0625, X_LEFT, X_RIGHT);
                CONSTANT a2n: fixed(X'RANGE) := to_fixed(-0.0625, X_LEFT, X_RIGHT);
                CONSTANT a1: fixed(X'RANGE) := to_fixed(0.50000, X_LEFT, X_RIGHT);
                CONSTANT maxSIG: fixed(X'RANGE) := to_fixed(1.00000, X_LEFT, X_RIGHT);
                CONSTANT minSIG: fixed(X'RANGE) := to_fixed(-1.00000, X_LEFT, X_RIGHT);
                VARIABLE SIG: fixed(X'RANGE);
        BEGIN
                IF to_integer(X) >= 4 THEN      -- Se      X >= 4 SIG = +1.0
                        SIG := maxSIG;
                ELSIF to_integer(X) < -4 THEN-- Se      X < -4 SIG = -1.0
                        SIG := minSIG;
                ELSIF to_integer(X) < 0 THEN  -- Se -4 < X < 0  SIG = (+0.0625*X+0.5)*X
                        SIG := (a2p * X + a1) * X;
                ELSE                                    -- Se  0 < X < 4  SIG = (-0.0625*X+0.5)*X
                        SIG := (a2n * X + a1) * X;
                END IF;
                RETURN SIG;
        END;

END neuron_pkg;
```

Bibliografia

ARAR, S. **allaboutcircuits**, 2017. Disponivel em: <https://www.allaboutcircuits.com/technical-articles/fixed-point-representation-the-q-format-and-addition-examples/>. Acesso em: 19 maio 2021.
CECCON, D. Funções de ativação: definição, características, e quando usar cada uma. **iaexpert**, 2020. Disponivel em: <https://iaexpert.academy/2020/05/25/funcoes-de-ativacao-definicao-caracteristicas-e-quando-usar-cada-uma/>.

THEODORIDIS, S.; KOUTROUMBAS, K. **Pattern Recognition**. [S.l.]: [s.n.], 2009.

UFSC. gsigma. Disponivel em: <https://www.gsigma.ufsc.br/~popov/aulas/rna/neuronio_artificial/index.html#:~:text=O%20neur%C3%B4nio%20artificial%20%C3%A9%20um,Esquema%20do%20neur%C3%B4nio%20biol%C3%B3gico.>. Acesso em: 19 maio 2021.

WOLBERG, D. W. H.; STREET, W. N.; MANGASARIAN, O. L. UCI. **Machine Learning Repository**, 2001. Disponivel em: <http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Prognostic%29>. Acesso em: 2021.