UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Projeto Final - Descrição de um Neurônio Artificial em VHDL

Linguagem e Descrição de Hardware (SEL0632)

Prof. Dr. Maximiliam Luppe

Alunos:

Luiza Sossai de Souza - 10748441

Gustavo Simões Belote - 10748584

São Carlos

10 de agosto de 2021

Sumário

1 Neurônio Artificial

O modelo de neurônio artificial é inspirado em um neurônio biológico para tentar replicar a alta capacidade de processamento existente no cérebro humano. O neurônio real é constituído principalmente por:

- Dendritos: estruturas em que ocorre a chegada de informação ao neurônio,
- Corpo celular: estrutura em que a informação é processada;
- Axônio: estrutura que transmite a informação processada para os neurônios seguintes.

1.1 Modelo matemático

Baseado nisso, um neurônio artificial contém várias entradas j, as quais recebem, cada uma, um sinal x_j. Ao contrário das sinapses do cérebro humano, em neurônios artificiais as entradas podem adquirir valores positivos e negativos. Cada um desses sinais por sua vez é multiplicado por um peso sináptico w_{kj}, sendo k associado ao neurônio e j ao terminal de entrada da sinapse. Por fim é realizado a seguinte soma soma dos mesmo por meio de uma junção aditiva de modo que:

$$u_k = \sum_{j+1}^{m} w_{kj} x_j \tag{1}$$

Por fim, esse sinal u_k é processado de modo a formar uma saída com valor finito. Isso ocorre por meio da função de ativação f. Além da função de ativação é aplicado um bias b_k de modo que a saída y_k é dada por:

$$y_k = f(u_k + b_k) \tag{2}$$

Em termos de descrição matemática, esse bias pode ser realocado na entrada do neurônio, fazendo-se uma entrada $x_0 = 1$ com peso $w_{k0} = b_k$. Desse modo, a expressão de y se torna:

$$v_k = u_k + b_k \tag{3}$$

$$y_k = f(v_K) \tag{4}$$

1.2 Função de Ativação

A função de ativação também faz parte do modelo matemático de um neurônio, mas, devido à sua importância, decidiu-se reservar um seção para explicá-la com mais detalhes.

O objetivo original das Redes Neurais Artificiais é simular o processo de aprendizado do cérebro humano computacionalmente. Nesse contexto, o seu mecanismo de funcionamento é amplamente inspirado nos processos bioquímicos envolvidos nas atividades dos neurônios. No sistema nervoso humano, estímulos chegam aos neurônios na forma de impulsos elétricos e, dependendo de sua intensidade, eles são ou não transmitidos aos neurônios seguintes. A função de ativação funciona de maneira semelhante em Redes Neurais Artificiais recebendo sinais de entrada e modificando-os de maneira que possam ser passados a diante (como saída da rede ou como entrada para um próximo neurônio).

Na prática, a função de ativação é o que permite que a rede aprenda relações e padrões de alta complexidade envolvendo o conjunto de dados de interesse. Isso está intimamente ligado com o fato de que a função de ativação é o componente responsável pela possibilidade de se adicionar não-linearidade na rede, de acordo com a expressão matemática que a define.

Outro aspecto importante relacionado à função de ativação é a sua capacidade de limitar o valor das saídas a um certo intervalo que permite o funcionamento adequado do sistema computacional utilizado. Isso se torna especialmente relevante à medida que a magnitude da rede vai aumentando, quando há uma grande quantidade de parâmetros a serem considerados, de forma que os resultados das somas e multiplicações podem resultar em valores demasiadamente elevados.

Existem algumas funções tradicionalmente muito estudadas e utilizadas em Redes Neurais Artificiais. Algumas delas são apresentadas a seguir: a função limiar, a função limiar por partes e a função sigmóide.

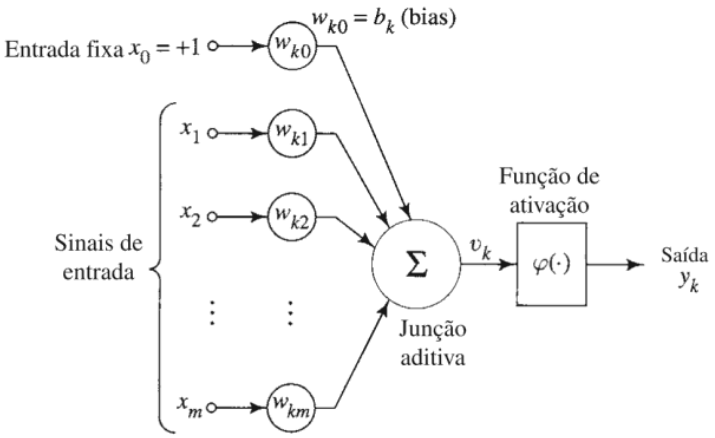

Figura 1: Representação de um neurônio artificial

1.2.1 Função limiar

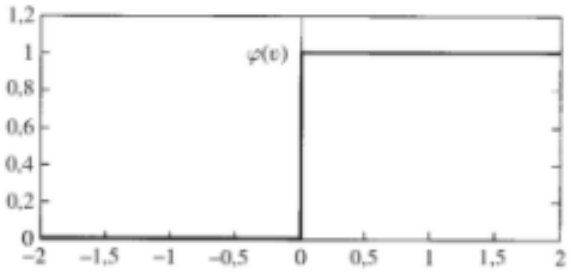

Figura 2: Função limiar

A função limiar retorna 1 sempre que o resultado de v_k é maior do que ou igual a 0 , e retorna zero caso contrário. Matematicamente:

$$f(v_k) = \begin{cases} 1, & v_k \geq 0 \\ 0, & v_k < 0 \end{cases}$$

1.2.2 Função linear por partes

A função linear por partes aproxima da função de um amplificador não linear seguindo a seguinte equação:

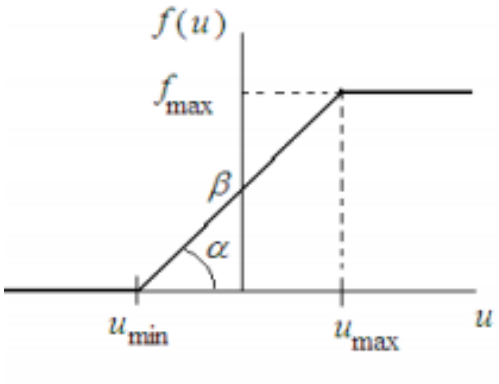

Figura 3: Função linear por partes

$$f(v_k) = \begin{cases} f_{max}, & v_k \geq v_{max} \\ \alpha v_k + \beta, & v_{min} < v_k < v_{max} \\ 0, & v \leq v_{min} \end{cases}$$

Quando o fator de ampliação α tende ao infinito a função linear por partes se transforma na função limiar, já se a função de ampliação é tal que não leve o sistema a saturação é criado um combinador linear.

1.2.3 Função sigmóide

A função sigmóide é uma função diferenciável, diferente das funções apresentadas anteriormente. Sua saída é um sinal crescente em formato de s e sua lei de formação pode ser representada por várias equações. Uma possibilidade empregada usualmente é dada pela seguinte equação:

$$f(v_k) = \frac{1}{1 + e^{-\alpha v_k}}$$

Nessa equação a representa a inclinação da reta, com exceção da origem, onde a inclinação é $a/4$. A função sigmóide é sem dúvida a mais utilizada como função de ativação em redes neurais.

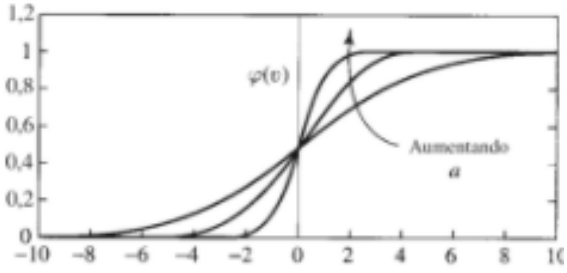

Figura 4: Função sigmóide

1.2.4 Funções de ativação ímpares

Em nenhuma das funções foi utilizado valores negativos na saída, porém fraquentemente é vantojaso que se tenha uma função de ativação ímpar, cuja saída esteja no intervalo $[-1, 1]$. Uma forma de obter isso é usar a função tangente hiperbólica (tanh) na definição da sigmoide e fazer $f_k(v_k) = 0$, se $v_k = 0$, e $f_k(v_k) = -1$, se $v_k < 0$ na descrição da função linear.

1.3 Rede neural *Multilayer Perceptron*

A maneira como os neurônios estão dispostos na rede neural é chamada de arquitetura da rede, a qual pode assumir diversas formas. A arquitetura da rede é importante para definir qual método de aprendizagem será empregado, isto é, a forma como a rede será treinada e como os pesos serão ajustados. Neste trabalho, iremos abordar somem a arquitetura Multilayer perceptron.

Uma rede MultiLayer Perceptron os neurônios são divididos em camadas, sendo elas:

- Camada de entrada: camada que recebe os sinais de entrada da rede;
- Camadas ocultas: camadas intermediárias de processamento de dados;
- Camada de saída: camada onde a informação final processada é adquirida.

A rede Multilayer perceptron é uma das mais utilizadas, pois consegue distinguir padrões que não são linearmente separáveis. Assim, esse modelo é amplamente usado

para resolução de problemas não lineares. Essa arquitetura normalmente adota a função sigmóide como função de ativação.

Nesta arquitetura, cada camada (a partir da segunda) só recebe como sinal de entrada as saídas da camada anterior a ela, sendo assim, não ocorre realimentação nem conexão entre neurônios pertencentes a uma mesma camada. Outra característica dessa arquitetura é que as camadas estão totalmente conectadas, isto é, um neurônio tem ligações com todos os neurônios da camada anterior a ele.

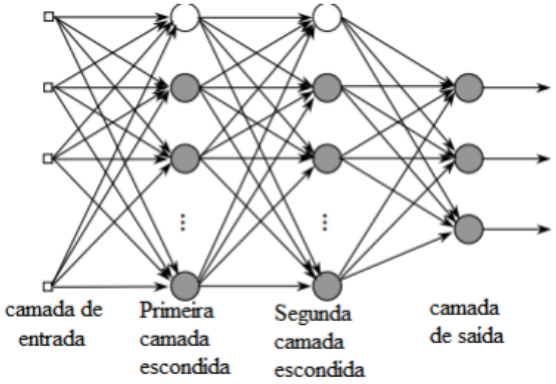

Figura 5: Rede Multilayer Perceptron

O método de treinamento utilizado é supervisionado e conhecido como algoritmo de retropropagação de erro (error back-propagation). Nele, um sinal é aplicado na rede e gera uma saída. Essa saída é comparada com a resposta desejada conhecida e, a partir disso, gera-se um sinal de erro. Esse sinal de erro é propagado para trás através da rede (na direção oposta ao fluxo dos sinais de entrada), de modo que os pesos da rede são ajustados de acordo com uma regra de correção do erro, visando tornar a saída real da rede cada vez mais próxima da saída ideal desejada.

2 *Q format*

Q format é um padrão de representação de números decimais que usa estruturas de hardware que teoricamente só conseguiriam suportar números inteiros. Esse padrão se baseia em separar a palavra de bits em uma porção para representar a parte inteira do número e a porção restante para representar a parte decimal,

frequentemente reservando o bit mais significativo para indicar o sinal do número.

Como a priori se tem liberdade para definir quantos bits representam a parte inteira e quantos representam a parte decimal, uma palavra de n bits poderia ser equivalente a diversas possibilidades de números decimais (dependendo da posição à qual o separador decimal estiver associado). Por isso, ao empregar o *Q format*, é essencial que o formato utilizado seja especificado. Isso pode ser feito de duas formas:

- $Q_{m.n}$ - onde m indica o número de dígitos associados à parte inteira e n indica o número de dígitos associados à parte decimal;
- Q_n - onde m indica o número de dígitos associados à parte decimal (nesse caso, assume-se que o tamanho da palavra de bits é conhecido ou pré-definido).

Assim, para se converter uma palavra de bits representada em *Q format* para o seu valor decimal equivalente, os dígitos à esquerda do separador decimal devem ser multiplicados pelas potências não negativas de 2 e os dígitos à direita do separador decimal pelas potências negativas de 2. Por exemplo, uma palavra de bits $d_{n+m-1}...d_1d_0$ no formato $Q_{m.n}$ (ou Q_n de $m+n$ bits, equivalentemente) pode convertida para seu valor decimal N_{10} correspondente da forma:

$$N_{10} = \sum_{i=0}^{n+m} di\ 2^{-n+i} \tag{5}$$

2.1 Números com sinal em *Q format*

Ao empregar o *Q format* quando precisamos representar números que podem ser positivos ou negativos, devemos levar em consideração:

- O bit mais significativo da palavra é reservado para indicar o sinal (e é denominado bit de sinal);
- Caso o bit mais significativo seja 0, o número é positivo e, caso seja 1 o número é negativo;

- Números negativos são representados usando seu complemento de 2.

Os passos para se obter a representação de um número negativo pelo seu complemento de 2 são: escrever a representação binária do módulo do número, trocar cada um dos bits pelo valor complementar e somar 1. A partir disso, a representação em *Q format* segue as mesmas regras mencionadas anteriormente. Deve-se apenas ter em mente que uma palavra de n bits possuirá na verdade $n-1$ bits numéricos, já que o mais significativo traz informação apenas sobre o sinal do número e não sobre o seu valor absoluto.

Para exemplificar, considere o número binário 11100010 em formato $Q_{5.3}$ com bit de sinal. Como ele é negativo, para sabermos que valor decimal N_{10} ele representa, precisamos determinar seu módulo pelo seu complemento de dois, que é $00011110_{(2)}$. Assim,

$$|N_{10}| = 2^1 + 2^0 + 2^{-1} + 2^{-2} = 3.75$$

Logo, $N_{10} = -3.75$

2.2 Conversões envolvendo *Q format*

A conversão de binário para decimal já foi apresentada anteriormente com a equação 5. Novamente é importante salientar que, caso o número possua bit de sinal, ela é válida na verdade para uma palavra de $n+m+1$ bits.

Já para a conversão de decimal para binário deve se levar em consideração o fator de escala do número em *Q format*, ou seja, o peso do bit menos significativo. Para ilustrar, considere o exemplo anterior. Nele, o peso do bit menos significativo era 2^{-2}, portanto, para resgatarmos o formato usual de um número binário (no qual o bit menos significativo tem peso 1), precisamos primeiramente multiplicar o número decimal por 2^2, arrendondar para o inteiro mais próximo para, então, realizar a conversão de decimal para binário da forma usual. Por exemplo, considere que o número 4.513 deve ser representado no formato $Q_{4.4}$.

- O peso é $p = 2^{-4}$ então o fator de multiplicação é $f = 2^4 = 16$
- O número decimal corrigido é $4.513 \times 16 = 72.208 \approx 72$
- A representação de 4.513 no formato $Q_{4.4}$ é equivalente à representação de 72 em binário: 01001000. Portanto $4.513 = 0100.1000$

2.3 Adição envolvendo *Q format*

Na operação de adição entre dois números em *Q format* é necessário que os ponto decimais estejam alinhados. Na prática, precisamos adicionar digitos à esquerda do número com menos digitos antes do ponto decimal de forma que essa mudança ainda represente o mesmo valor original. Para números positivos, é fácil ver que obtemos esse resultado adicionando zeros à esquerda. Para números negativos (os quais devem estar representados como complemento de 2), devem ser adicionados dígitos 1 à esquerda. Tendo-se feito isso, podemos proceder com a adição da forma usual.

Algumas observações importantes que deve sempre se ter mente antes de realizar uma adição em *Q format*:

- É importante sempre representar os números com 1 bit a mais em relação ao tamanho do maior número (em magnitude) para evitar um possível overflow com o resultado da soma;
- Tendo-se respeitado a condição anterior, caso o resultado exceda o número de bits usados para representar os números, o bit excedente deve ser ignorado.

3 Implementação em VHDL

Para que seja possível implementar os conceitos de Redes Neurais Artificiais e Q Format discutidos anteriormente utilizando a linguagem VHDL, foi necessário criar alguns pacotes e entidades. Aspectos importantes relacionados a eles são apresentados a seguir.

Todos os códigos mencionados estão disponíveis no *gitHub* do grupo, o qual pode ser acessado através do link: `https://github.com/luizasossai/ann_mlp`

3.1 Pacote: fixed_package

Inicialmente foi criado o pacote *fixed_package*. Nele, são definidos novas constantes, subtipos, tipos, funções de conversão e funções matemáticas, além da sobrecarga dos operadores matemáticos básicos para que se possa trabalhar adequadamente com variáveis em Q Format.

3.2 Teste: fixed_package_tb

Para atestar o funcionamento correto dos itens definidos no pacote *fixed_package*, foi desenvolvida uma rotina de teste, o *fixed_package_tb*. Nela, inicialmente foram criadas duas funções necessárias para a realização dos testes: uma função de conversão de *fixed* para *string* (fixed2string) e a sobrecarga do operador de igualdade para a comparação de dois números com formato *fixed*.

Em seguida, essas novas funções foram usadas juntamente com declarações *assert* para avaliar se os valores fornecidos como resultado das funções criadas eram condizentes com os esperados. Em caso de divergência, um *report* faria com que aparece uma mensagem de erro no console do *ModelSim*, mostrando o valor obtido e o valor esperado.

3.3 Pacote: neuron_pkg

O próximo passo foi criar um pacote com a função de ativação a ser utilizada nos neurônios artificiais. As funções utilizadas foram baseadas em [6]. No artigo consultado, os autores propõem aproximações da função sigmoide por funções com leis de formação matematicamente mais simples, de forma a possibilitar a sua implementação em VHDL.

Porém, em relação às funções de ativação fornecidas na disciplina, foi necessário alterar a condição da estrutura condicional, visto que a fornecida utiliza to_integer

para realizar a comparação. Entretanto, pela maneira que anteriormente havia sido feita a função to_integer no fixed_package, não se considera o arredondamento dos valores, e sim faz o corte da parte não inteira do argumento recebido pela função. Isso causa erros na análise de condição, pois, por exemplo, todo número pertencente ao intervalo $(-1, 0)$ retornará zero, de forma que a estrutura condicional entenderá que este número pertence ao intervalo $[0, 4)$ e, portanto, fornecerá uma resposta incorreta. Por essa razão, foram trocadas as funções to_integer para to_real em todas as suas ocorrências no corpo das funções de ativação.

No pacote são definidas duas possibilidades de função de ativação, porém, para a implementação do neurônio artificial foi usada apenas uma delas (*Activation1*).

3.4 Teste: activation_function_tb

Para verificar o bom funcionamento das funções de ativação, foi disponibilizada uma rotina de teste e um arquivo do tipo *ActiveTclScript*. A partir da compilação da rotina de teste no *ModelSim* e da execução do *script*, deveriam ser obtidas dois gráficos com formato análogo ao de uma função sigmoide e cujos valores na metade do tempo de simulação (12800000 ps) deveriam ser iguais a 0, 0.5 e 0. O resultado obtido foi condizente com o esperado e é mostrado na fugura 6.

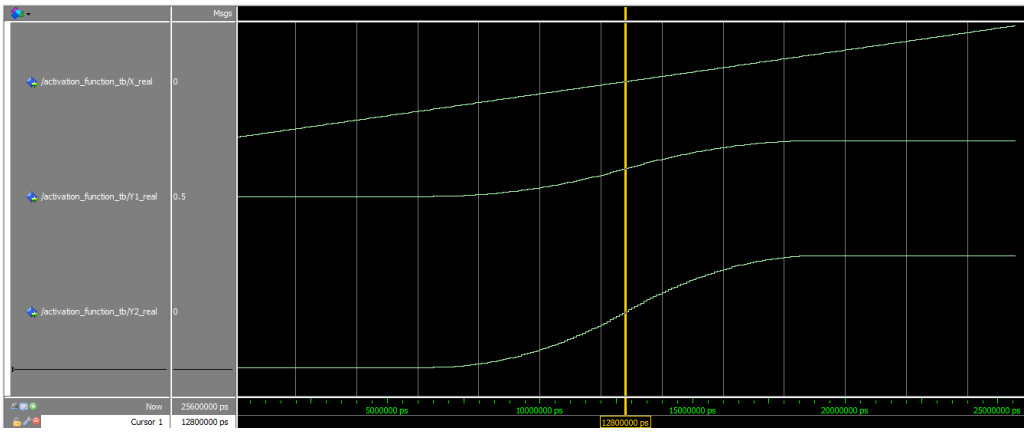

Figura 6: Gráficos obtidos na execução do arquivo de teste para a função de ativação.

3.5 Entidade: neuron

Por fim, criou-se a entidade neuron, que descreve o funcionamento de um neurônio: o neurônio recebe um vetor de entrada X (cujo tamanho pode ser modificado facilmente no código caso se deseje), multiplica o vetor de entrada por um vetor de peos W elemento a elemento. Soma essas valores e subtrai por um *bias* B. Esse valor é aplicado na função de ativação *Activation1* para originar a saída do neurônio.

3.6 Teste: neuron

O teste da entidade foi realizado modificando a inicialização da variável *dendritos* para 2, com o fim de simplificar a análise. A simulação foi realizada no *ModelSim*. Inicialmente definiram-se valores para a entrada X, para os pesos W e para o *bias* B. Além disso, substituiu-se o valor da expressão resultante da combinação linear entre as entradas e os pesos na expressão correspondente da função de ativação *Activaction1*. O resultado então foi convertido para o *Q format* utilizado ($Q_{3.12}$) e comparado com o resultado da simulação.

3.6.1 Teste 1

Valores de entrada:

- $X(1) = 0.75 = 0000110000000000$
- $X(0) = 0.25 = 0000010000000000$
- $W(1) = 1.25 = 0001010000000000$
- $W(0) = 0.75 = 0000110000000000$
- $B = 0.25 = 0000010000000000$

Realizando a operação $X(1) \cdot W(1) + X(0) \cdot W(0) - B$, obtem-se $0,875$. Dessa forma, basta avaliar o resultado da função de ativação quando ela recebe esse valor como entrada e comparar o resultado com o da simulação do *ModelSim*.

Valores obtidos:

- Simulação: $Y_s = 0000101100011110$
- Cálculo analítico: $Y_a = 0.69482421875$

Efetuando a conversão de qualquer um dos valores, verifica-se que $Y_s = Y_a$.

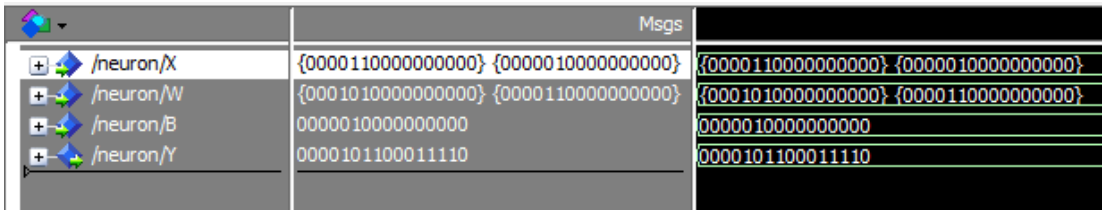

Figura 7: Entidade *neuron* - Primeiro teste.

3.6.2 Teste 2

- $X(1) = 0.5 = 0000100000000000$
- $X(0) = 0.25 = 0000010000000000$
- $W(1) = 1.0 = 0001000000000000$
- $W(0) = 0.5 = 0000100000000000$
- $B = 1.5 = 0001100000000000$

Realizando a operação $X(1) \cdot W(1) + X(0) \cdot W(0) - B$, obtem-se $-0,875$. Dessa forma, basta avaliar o resultado da função de ativação quando ela recebe esse valor como entrada e comparar o resultado com o da simulação do *ModelSim*.

Valores obtidos:

- Simulação: $Y_s = 0000010011100010$
- Cálculo analítico: $Y_a = 0.30517578125$

Efetuando a conversão de qualquer um dos valores, verifica-se que $Y_s = Y_a$.

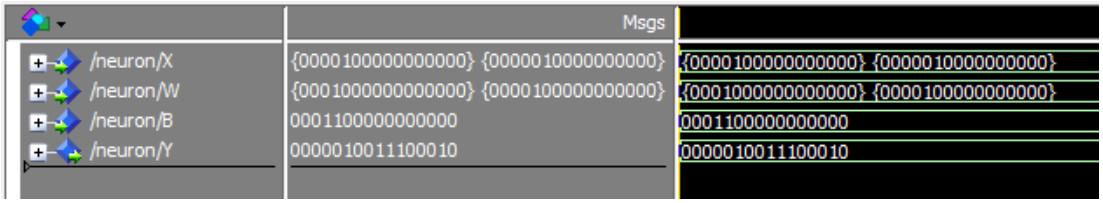

Figura 8: Entidade *neuron* - Segundo teste.

Referências

[1] HAYKIN, Simon. Neural networks: principles and practice. Bookman, v. 11, p. 900, 2001.

[2] FLECK, Leandro et al. Redes neurais artificiais: princípios básicos. Revista Eletrônica Científica Inovação e Tecnologia, v. 1, n. 13, p. 47-57, 2016.

[3] ROQUE, Antônio. Do neurônio biológico ao neurônio das redes neurais artificiais. Acesso em: 21 de maio de 2021. Disponível em: http://sisne.org/Disciplinas/PosGrad/PsicoConex/aula3.pdf

[4] ARAR, Steve. Fixed-Point Representation: The Q Format and Addition Examples. 2017. Acesso em: 20 de maio de 2021. Disponível em: https://www.allaboutcircuits.com/technical-articles/fixed-point-representation-the-q-format-and-addition-examples/

[5] JAIN, Vandit. Everything you need to know about "Activation Functions" in Deep learning models. Acesso em: 02 de agosto de 2021. Disponível em: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253

[6] TSMOTS, Ivan; SKOROKHODA, Oleksa; RABYK, Vasyl. Hardware implementation of sigmoid activation functions using FPGA. In: 2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM). IEEE, 2019. p. 34-38.