Implementation of an artificial neuron in VHDL

Javier Gómez Casado
2008

Abstract

The purpose of this thesis is to study and simulate the behaviour of a biological neuron, how nervous impulses, the so-called spikes, are transmitted through the body of the neuron, from the dendrites to the axon. And with this knowledge, to emulate this behaviour with the target getting a system, in VHDL, that works exactly in the same way that a biologic neuron. For it, it is important to know how, why and in what circumstances a neuron is fired.

From an electronic point of view dendrites of a neuron are considered as inputs, and the axon as an output. Nervous impulses are interpreted as electrical signals.

The artificial neuron will be used in other applications, so that, the codification of the program that emulates the neuron will have be extrapolated to other systems to get a complete neural network.

With the codification in VHDL of the artificial neuron a FPGA can be programmed for future applications of computational neuroscience as well as artificial intelligence.

Contents

List of figures

1. Project aims

The human nervous system has been treated by numerous studies. Not only from a medical point of view, as treatments for prevention of diseases, but also, from a technological point of view, trying to emulate the behaviour of the biological neuron. And based on this, modelling an artificial neuron can be used to develop future applications in computational neuroscience, as well as in artificial intelligence.

Artificial neural networks simplify the behaviour of the human brain, so, their applications are used in different fields as industrial automation, medicinal applications, robotics, electronics, security, transport, military, etc.

Applications in pattern recognition like recognition of fingerprints or control of missiles use systems based on neural networks among other techniques.

The following research is based exactly in this issue, to understand the real behaviour of a biological neuron and according to this being able to model an artificial neuron that works in a similar way. When this artificial neuron will be developed, it will be used in future applications through complete neural networks for applications as commented previously.

A neuron can be compared to a black box composed of few inputs and an output. Like an electrical circuit that makes the addition of the different signals that receives from other units and obtain in the output according to the result of the addition with relation to the threshold. The artificial neuron is an electronic device that responds to electrical signals
An example of the use of artificial neurons in the industry is, for example, CCortex, building by CorticalDB, which is a massive spiking neural network simulation of the human cortex and peripheral systems. CCortex represents up to 20 billion neurons and 20 trillion connections [1].

According to the manufacturer, the development of CCortex will enable a wide range of commercial products that will transform and enhance global business relationships, with advanced capabilities in pattern recognition, verbal and visual communication, knowledge acquisition, and decision-making capabilities. These products will have widespread applications in the fields of artificial intelligence, communications, medical modelling, and database and search technologies.

These are only a little sample of the abilities that can be implemented starting with the development of the artificial neuron.

2. Biological background

First at all, we must study and understand the different parts and the behaviour of neurons. Functionally, a neuron can be divided in three parts:

Soma: The body of the neuron.
Dendrites: Branches that transmit the action potentials from others neurons to the soma.
Axon: A branch that transmits the action potential from the neuron to other cell.

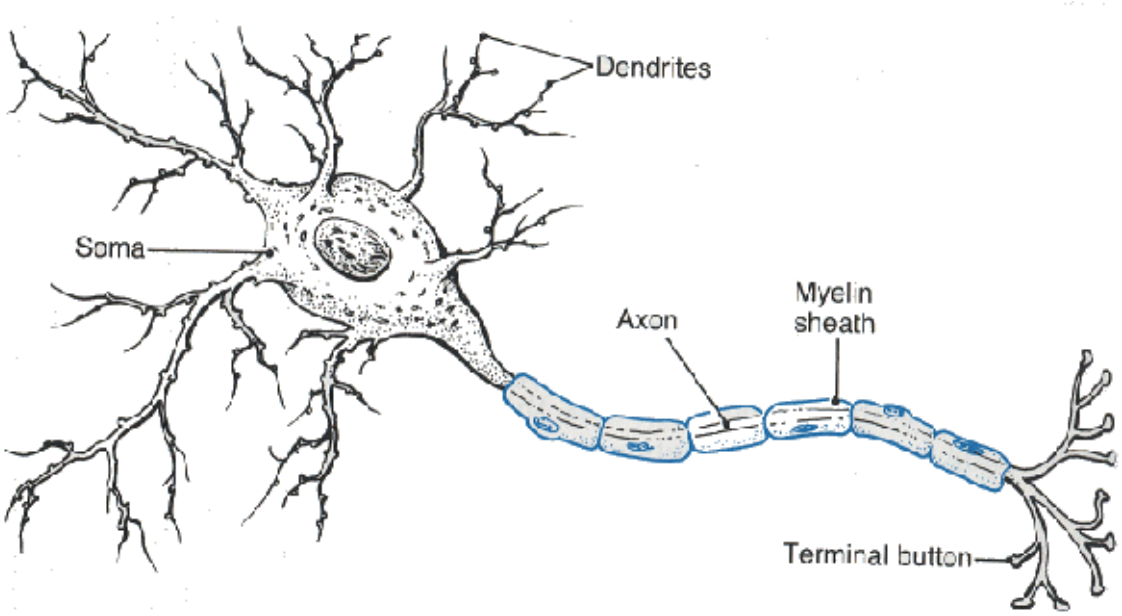


Figure 1: Parts of a neuron. *All main parts of a neuron can be appreciated in this picture, dendrites, soma and axon.*

The soma is the central unit of the neuron, it receives signals from others neurons across the different dendrites. If the addition of these signals is over of a certain threshold an output signal, a spike, is generated. The spike is propagated by the axon to other neurons. The contact site between the axon of the first neuron and the dendrites of the next neurons is called synapse.

The spikes are short electrical pulses. The amplitude of theses spikes, are about 100 mV and they have duration of 1 or 2 ms [2].

A spike train is a chain of action potentials emitted by a neuron, the importance of action potentials is in the timing and the numbers of spikes, the form of the action potential does not carry any information, since they all look almost the same. The spike is the elementary unit of signal transmission.

When a neuron is fired, it generates a action potential, this action potential is transmitted and processed in the soma of the next neuron, where action potential from

others neurons are arriving. The soma considers all the input from the different neurons and makes a non-linear addition of these signals. The sum of this addition gives the membrane potential of the neuron, which is transmitted along the axon to target neurons.

During the summation process, not all the inputs have the same relevance, thus, some are more important (represented as higher weight values) whereas other inputs are less important (lower weight values). One of the characteristics of the biological as well as artificial neuronal network is their capacity of learning, for that, the inputs have adaptive weights.

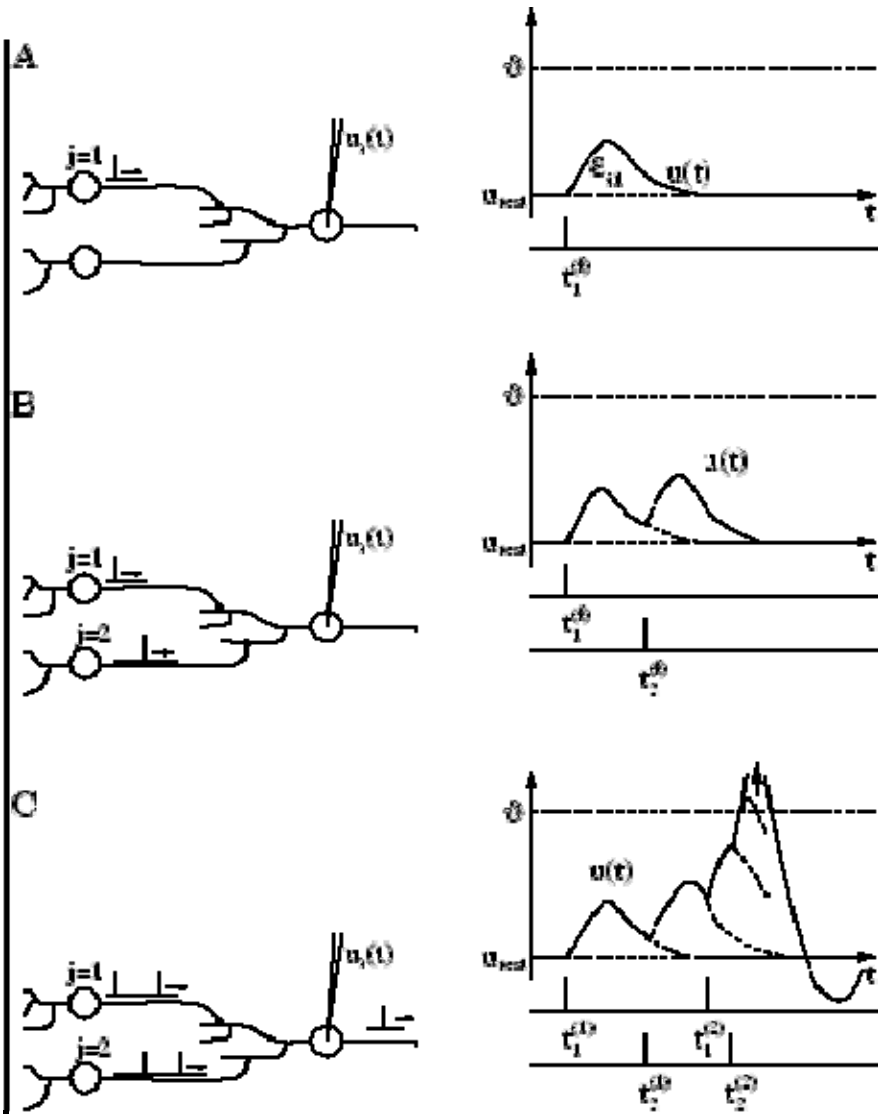


Figure 2: Potential actions. *The neuron of the picture receives signal from two inputs, j=1 and j=2, at the moment that a spike is received, an EPSP (excitatory postsynaptic potential) is generated (case A). A spike from the second presynaptic neuron arrives, then, another EPSP is generated that is added to the first one (case B). Finally, after others EPSP generated by the arriving of other spikes, the addition of the potentials is over the threshold, as a consequence an action potential is triggered (case C) and starts a large positive pulse(arrow).*

Amplitudes of the presynaptic potentials are in the range of one milivolt, and the threshold or critical value for spiking initiation is over 20 or 30 mV [3].

Most of the cases about 20 or 50 spikes in a short period of time are needed to overall the threshold. When the addition of presynaptic potentials is over this range a spike is generated and transmitted along the axon to the next postsynaptic neurons.

When it happens, at the same moment, the neuron needs a period up to approximately 2 millisecond to recover the capacity to fire again, this time is period is know as the absolute refractory period. During this time the neuron ignores the arriving of the spikes from other dendrites, when this time is over, the neuron comes back to work in the same way as the previous one [4].

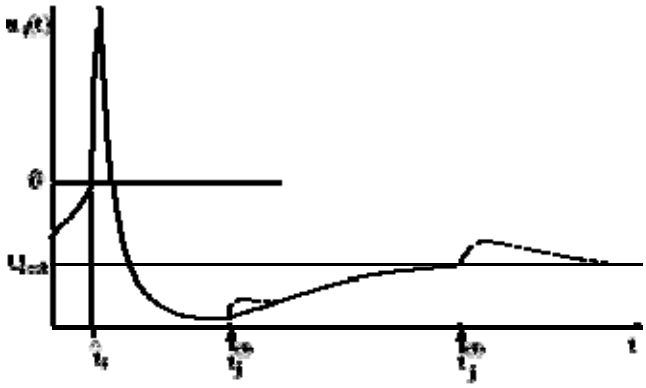


Figure 3: Refractory time. *This is the waveform of a postsynaptic potential. After the pulse generated when the potential is over the threshold the voltage returns to a value the resting potential known as refractory time, during this time, the neuron ignores the arriving of others spikes.*

Inside a neuron we have to recognize two kinds of potentials, they are the presynaptic potential and the postsynaptic potential.

The presynaptic potential is the potential generated in others neurons, called spike that is transmitted through the dendrites to the neuron. On the other hand, the postsynaptic potential is the potential generated in the own neuron and transmitted to the next neuron through the axon.

We can define the presynaptic potentials as nervous impulses that arrive to the neuron and the postsynaptic potential as the action generated in the neuron that is propagated to the rest of neurons.

3. Analysis of the problem

The purpose of this thesis work is to make a model that implement the behaviour of a neuron. This model will have some inputs that will emulate the dendrites of the neuron and a pulse will be obtained as a response, which will be propagated to other neurons across the axon.

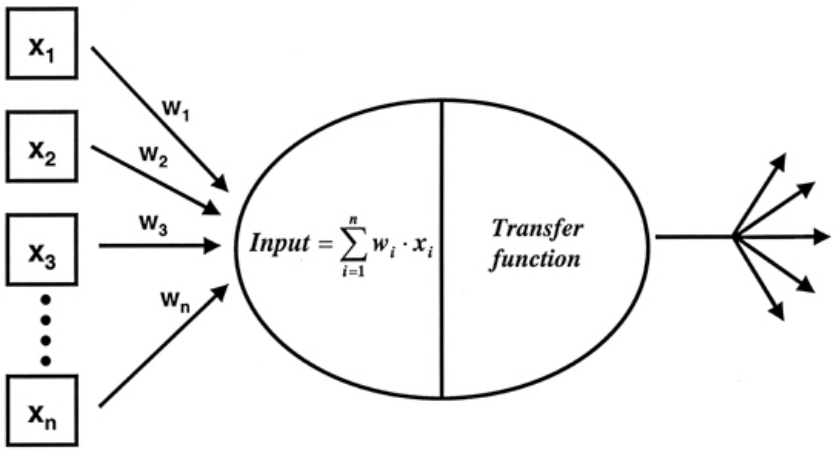


Figure 4: Analysis of a neuron. *In this picture we can see the simplified behaviour of a neuron. The inputs ($x_1, x_2, x_3…x_n$) and their respective weights ($w_1, w_2, w_3…w_n$). In the processing unit or soma, firstly the addition of all inputs is realized and after that a signal is generated to be transmitted to others neurons.*

The system must be able to receive the spikes from others neurons, to process these inputs emulating the events that happen when a spike is arrived, making the comparison between the threshold and the addition of potentials and generated a potential action when it is needed. Besides, we have to consider others factors, like the effects of the absolute refractory time, and the importance of the different weights in the response of the system.

The generated code will be used to program a FPGA (Field Programmable Gate Array), for that, we have to consider all signals, spikes, potential action, etc. as digital signals. For example, the arriving of a spike will be considered as the arriving of a digital pulse.

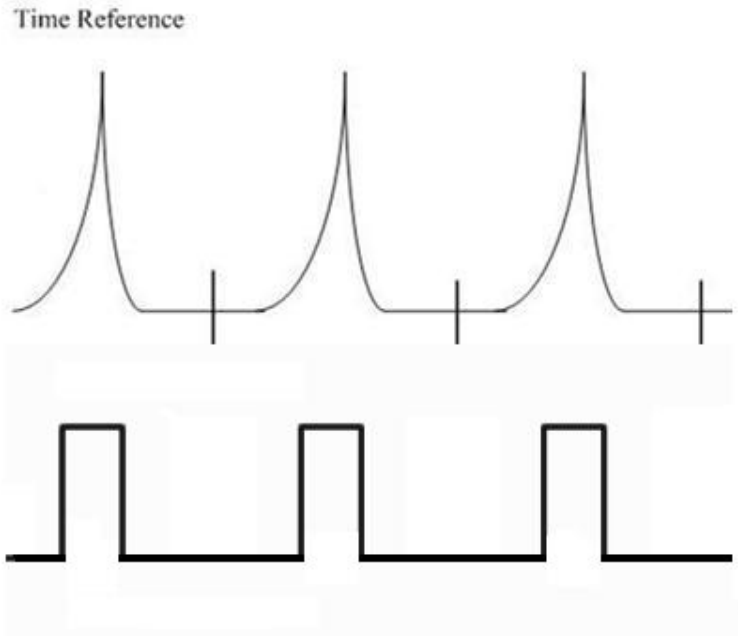


Figure 5: Spike Vs Pulse. *In a real neuron the waveform of the spikes that arrive to a neuron are like shown in the picture. For modelling the artificial neuron we only use digital signals, for that, all spikes will be interpreted as pulses.*

In the same way, the resultant wave will be represented by a pulse. When the neuron was active the output will be a '1' logic and in others cases the output will be a '0' logic.

4. Model

The implementation of this neuronal model will be carried out in VHDL language.
VHDL language is a combination of VHSIC (Very High Speed Integrated Circuit) and HDL (Hardware Description Language) [5].
We have used this language because, in a future, this code will be valid to programme a FPGA following applications. This language is used to program and design digital circuits, for that, all signals will be digital signals.

The way to cope with the problem is using a top-down method, that is, to divide a complex design in easier designs or modules, each module is redefined with greater details or divided in more subsystems. For that, you have in the first view a general idea of the system, and if you go down through the subsystems you can see with more detail how each block works.

For the developing of the system we have used a system based in an artificial neuron with only three dendrites as inputs, actually, the numbers of simultaneous spikes, in a short period of time, needed to excite a neuron are about 50 but we think that 3 dendrites are enough to show how a neuron works. To get that these 3 dendrites will be able to fire the output, we will have to increase the value of the weights, associated with the dendrites. They are defined like variables, can let the possibility of change the values to make a dendrite more significant than others.

One of the most important characteristics of the neural networks is the ability of learning. For neurons, learning is the modification of the induced behaviour for the interaction with the environment and like result of experiences that drives to establishment of new response models to extern stimulus.
In artificial neuronal networks, the knowledge is represented in the weights of the connections between the neurons.The process of learning implies some numbers of changes in these connections. The neuronal network learns modifying the values of the weights of the network.

The use of weights is very important weights are associated to the synapses and can increase or decrease the signals that arrive to a synapse.
As mentioned earlier, we have defined 3 variable weights as inputs, every weight is associated with a dendrite to let that the global neuronal network let the ability of learning.

To first view our system will have:
- 3 inputs referred to the dendrites from others neurons.
- 1 output referred to the axon. When the output would be fired, the output will have to generate a pulse to propagate it to others neuron through the axon.

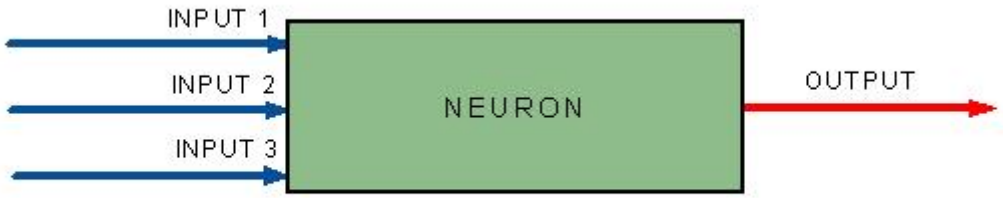


Figure 6: Neuron with 3 inputs.

In a deeper analysis, we have to include other important blocks. We have to introduce more inputs to get the system can learn, they are the weights.
For the other hand, also, it is necessary to introduce a signal of clock in the system. All the signals will be digital signals, for that, it is recommended a signal that synchronizes the global system.

- 3 inputs referred to the weights of the connection between other neurons and our neuron. Every weight is in relation with each dendrite.
- 1 input of clock. It is necessary a clock signal to make synchronous the system

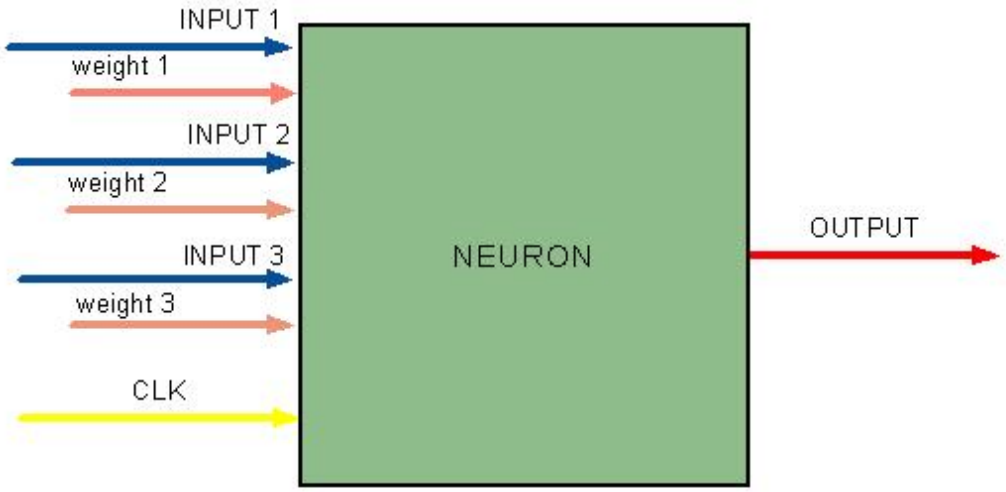


Figure 7: Neuron with weight inputs

When a spike arrives, the soma has basically two functions, to generate the potential action according to the input and to compare if the addition of all potential actions in this instance of time is over a threshold, in that case, it will have to generate a pulse through the axon.

We are going to consider these actions as:

- Potential generator. When receive a pulse starts to increase the potential.
- Addition and comparison. Add all the potential and compare with the threshold.

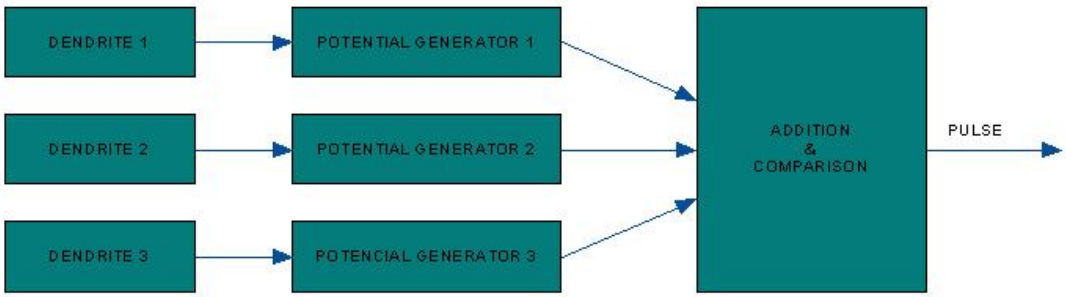


Figure 8: Functions of a neuron. *The functions of the processor unit of a neuron are to generate a potential according to the arriving of spikes and evaluate these potential to generate an output potential.*

Now, we are going to define the potential generator block, when a spike arrives, inside the soma an excitatory postsynaptic potential (EPSP) is generated, this potential follows a exponential function determinate by the alpha function $f(x) = x * e^{-x}$

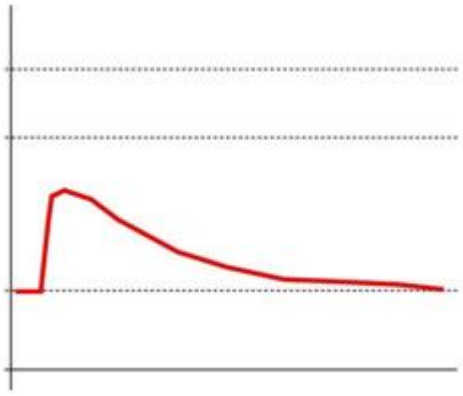


Figure 9: Real EPSP

With this waveform we can observe that the amplitude changes in relation with the time, for that, it is necessary to implement the Potential generator block with two others blocks, one block that works as a timer, it must to start the count in real time when a spike arrives, and reset the count when it would be necessary.

After this block, it will be other block that implements de alpha function. The mission of this block is receive the resultant value of the timer counter and process it

considering the respective weight to obtain an value that represent the value of the potential action in this determinate instant of time.

0	0	2,9	0,15956734
0	0	3	0,14936121
0	0	3,1	0,13965253
0	0	3,2	0,13043905
0,1	0,09048374	3,3	0,12171445
0,2	0,16374615	3,4	0,11346912
0,3	0,22224547	3,5	0,10569084
0,4	0,26812802	3,6	0,0983654
0,5	0,30326533	3,7	0,09147705
0,6	0,32928698	3,8	0,08500893
0,7	0,34760971	3,9	0,07894345
0,8	0,35946317	4	0,07326256
0,9	0,36591269	4,1	0,06794797
1	0,36787944	4,2	0,06298142
1,1	0,36615819	4,3	0,0583448
1,2	0,36143305	4,4	0,0540203
1,3	0,35429133	4,5	0,04999048
1,4	0,34523575	4,6	0,04623844
1,5	0,33469524	4,7	0,0427478
1,6	0,32303443	4,8	0,03950279
1,7	0,31056199	4,9	0,03648826
1,8	0,297538	5	0,03368973
1,9	0,28418038	5,1	0,03109341
2	0,27067057	5,2	0,02868613
2,1	0,2571585	5,3	0,02645545
2,2	0,24376695	5,4	0,02438954
2,3	0,23059534	5,5	0,02247724
2,4	0,21772309	5,6	0,02070804
2,5	0,2052125	5,7	0,019072
2,6	0,1931113	5,8	0,01755982
2,7	0,18145488	5,9	0,01616272
2,8	0,17026818	6	0,01487251

Table1: Potentials in function of time. *This table shows the values that takes the potential action in different time slots.*

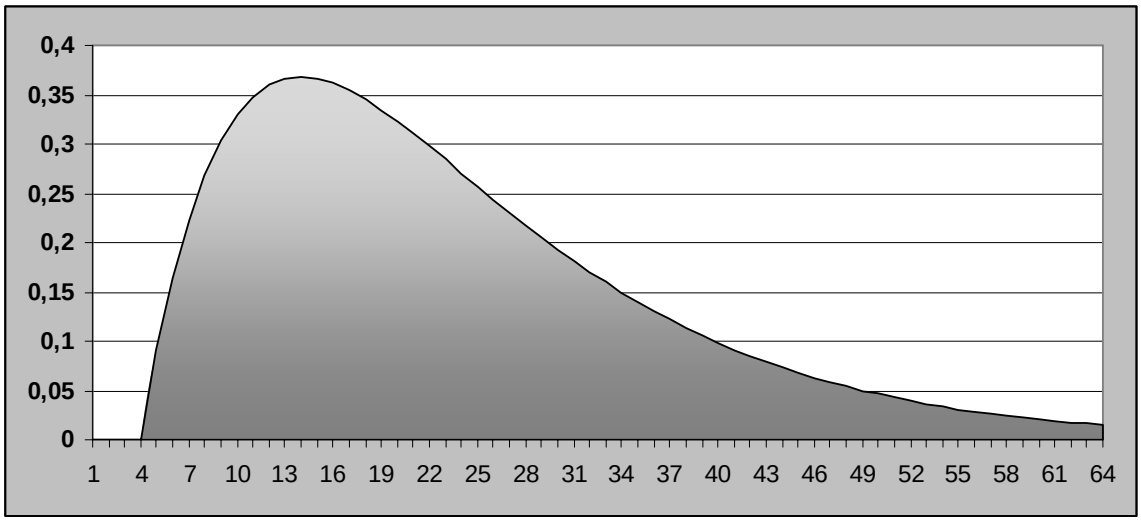


Figure 10: Modelled EPSP. *Following the alpha function $f(x) = x * e^{-x}$ and related with the time we can emulate the waveform of a real EPSP.*

In Table 1 and Figure 10 we can observe different values that can take the alpha function for different instants of time.

For each dendrite we will need one potential generator, for that, for the three dendrites will be 3 timers and 3 Alpha functions.

In these blocks is when we will use the weight inputs, according to the weight of the input the potential obtained by the alpha function will increase o decrease, making the synaptic connection more or less important.

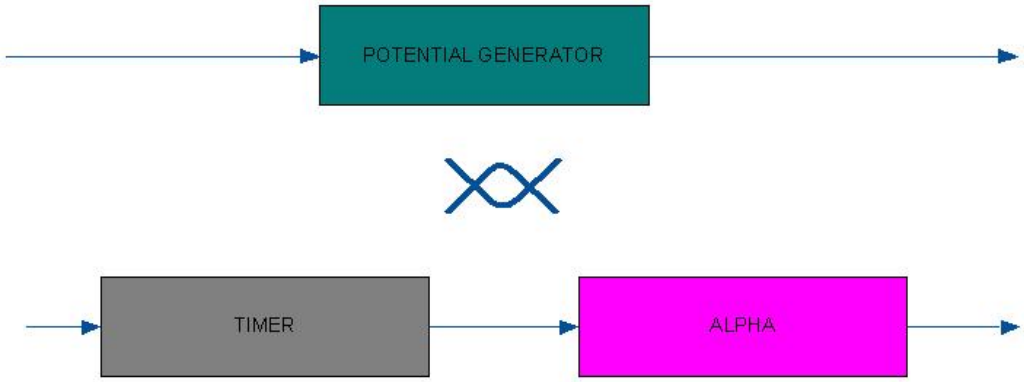


Figure 11: Splitted potential generator. *Potential generator is in function of time, for that to implement the potential generator block we need two blocks, timer block to generate a signal that temporizates and the alpha block to make the operations.*

The following block is continuously making the addition of all potentials and comparing the result with the threshold, defined as 20 mV. If the result is positive it

sends a pulse indicating that the neuron is excited. When the addition of all potentials is not over the threshold the output is a '0' logic. This block can be implemented with only a block that makes the addition and comparison.

On the other hand, there is a factor that we have not considered yet. It is the refractory time. When a neuron is fired, during a period of time it is unable to generate another spike, this time period is the refractory period and it has duration of about 2 ms.

To solve this problem we have used another block named inhibit block. This block works as a timer. When a neuron is fired, it sends a signal to the potential generators to inhibit and reset any count. At the same time, it starts a count, until the timer is over 2 ms, it keeps sending the inhibit signal. When the counter overloads, it disables the inhibit signal. The potential generator is able to generate potential when a pulse arrives again.

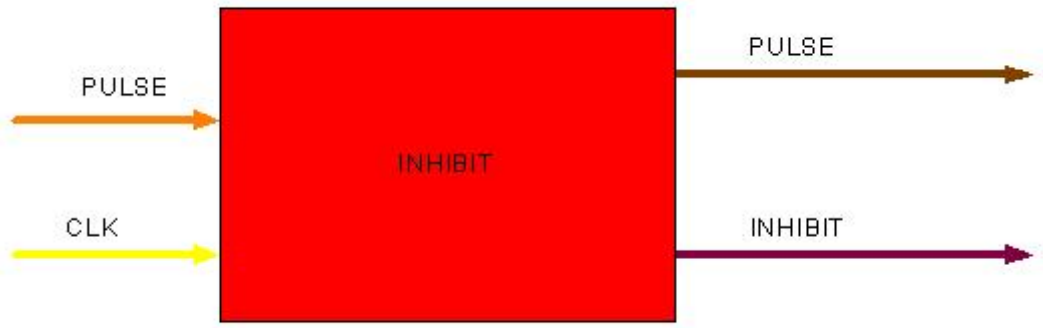


Figure 12: Inhibit block

We have to create a new block, inhibit block, and modify the potential generator adding it a new input, an enable input, when this input is active the timer does not count and the output of alpha block always will be "0" logic.

Figure 13: Potential generator with inhibit signal

5. Solution

To get the artificial neuron we have split the system in some blocks, as it is explained previously. The first step is program and probe these blocks by separated, verify that carry out with the specifications, after that, realize the interconnection between the block to check if the resultant system works as we expected.

First of all we are going to codify the Timer:

5.1 TIMER:

Requirements:
We need a system that when it detects a pulse (the rise edge), starts a count of real time. It will also have an enable input, when this input is active, the timer will stop count, and the count would be reset. When this signal will be inactive the timer will be able to count again when receive other rise edge.

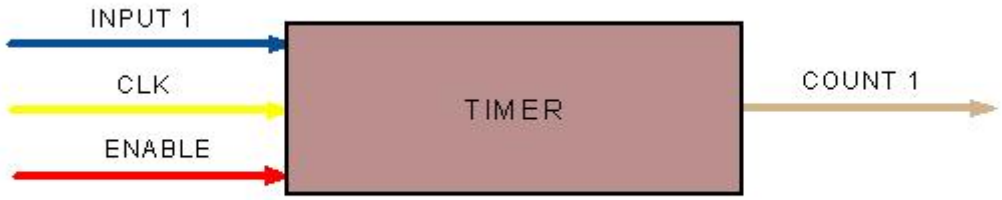


Figure 14: Timer block

Inputs: Input1: External input from others neurons
CLK: External input, used to synchronize the count.
Enable: Internal input, from the inhibit block.
Outputs: Count1: Internal output, to alpha block.

Code: counters.vhd [6][7]

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;
   use IEEE.std_logic_unsigned.all;
   use IEEE.math_real.all;

entity counters is port(
clk:      in std_logic;
input1: in std_logic;
enable: in std_logic;
count1: buffer real:=0.0
```

```
);
end counters;

architecture archicontador of counters is
begin
process (input1, clk)
variable pulso_rec: real :=0.0;
   begin
  if (enable = '1') then
  count1 <= 0.0;
  pulso_rec :=0.0;
  else
  if (input1'event and input1='1') then
  count1 <= 0.0;
  pulso_rec := 1.0;
  end if;

   if ((clk'event and clk= '1') and pulso_rec= 1.0) then
    count1 <= count1 + 0.1;
  end if;
  end if;
 end process;
end archicontador;
```

The behaviour of this block is the following. The program checks if the signal enable is active, in this case, it blocks the system and reset the counter. Otherwise, the system starts to check if a new pulse is arriving through the input1. In that case, the counter resets and shows to the system that a pulse is been processed. Afterwards, the counter will increment its value until other pulse arrives or the enable signal was active again.

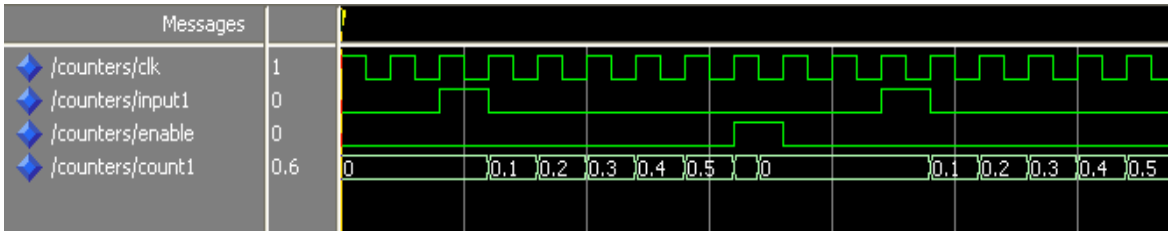


Figure 15: Timer block simulation. *In this simulation we can observe the correct behaviour of the program. When the signal input1 is active the count1 starts, count1 is increased gradually according to the clk signal. When enable is active count1 is reset and wait until the input1 is active again.*

5.2 ALPHA:

Requirements:
This block must be able to use the result of the count, get a potential action follow the alpha function $f(x) = x * e^{-x}$ where "x" is the real time, e.i., the result of the timer counter. The result must also take the value of the weight of this input into the consideration.

Figure 16: Alpha block

Inputs: A: Internal input, from the Timer block.
Weight1: External input associated to the input from others neurons.
RESULT_A: Internal input to Addition/Comparison block.

Code: alpha.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;
   use IEEE.std_logic_unsigned.all;
   use IEEE.math_real.all;

   entity alpha is
      port(A:in real :=3.0;
         RESULTADO: out real:=0.0);
       end entity alpha;

     architecture behavioral of alpha is

          begin

            RESULTADO <= exp(-A)*A*1.0;
       end architecture behavioral;
```

The behaviour of this block is very simple. It takes the value of the previous block and introduces it in the function $f(x) = x * e^{-x}$, at the same time, it multiplies this value with the corresponding weight. The resulting value is put in the output.

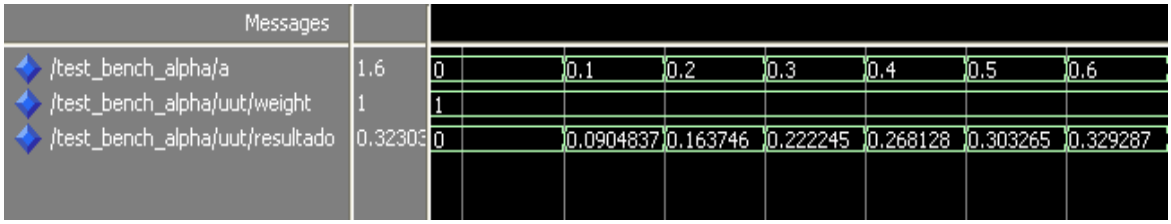


Figure 17: Alpha block simulation. *In this simulation we can observe that the signal "esultado"changes its value according to the changes in the inputs "a" and "weight"*

5.3 ADDITION/COMPARISON:

Requirements:
This block must be able to take the result of the potential actions from the alpha blocks, and in every instant of time, to make the addition of all these potentials. At the same time, the system must take into consideration the comparison between the value of the addition and the threshold. The threshold is the level that determinates when a neuron is excited, fixed to 20 mV. When the result of the addition is over the threshold, the system puts the output to 1.

Figure 18: Addition & Comparison block

Inputs: A1: Internal input, from the Alpha block of the dentrite1.
B1: Internal input, from the Alpha block of the dentrite2.
C1: Internal input, from the Alpha block of the dentrite3.
SAL: Internal output, to Inhibit block.

Code: threshold.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;
   use IEEE.std_logic_unsigned.all;
   entity threshold is
      port(
      A1:in real :=1.0;
      B1:in real :=2.0;
      C1:in real :=0.0;
         SAL: out std_logic);
      end entity threshold;
      architecture behavioral of threshold is
         begin
          SAL<='1' when ((A1+B1)+C1)>1.2 else '0';
       end architecture behavioral;
```

This block sums all the spikes and compares the value with the threshold.

If the addition is over the threshold the output is put to '1', in others cases the output is '0'.

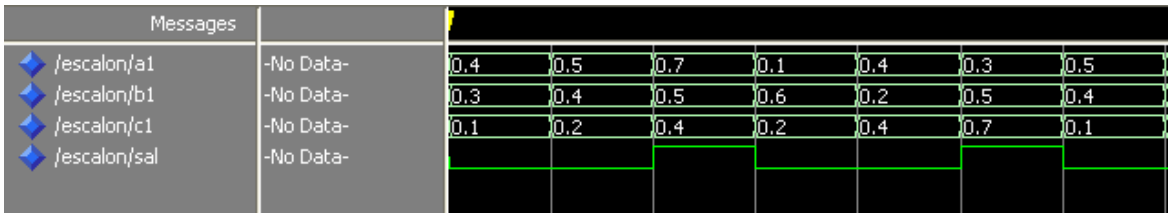


Figure 19: Addition & Comparison block simulation. *In this simulation we can observe that the output "sal" is active only when the summation of all inputs (a1, b1, c1) is over a determinate threshold, in this case, for the simulation 1.2.*

5.4 INHIBIT:

Requirements:
This block must generate a signal that controls the refractory period. When it receives a signal, must generate other signal of a duration de 2 ms (the refractory time).

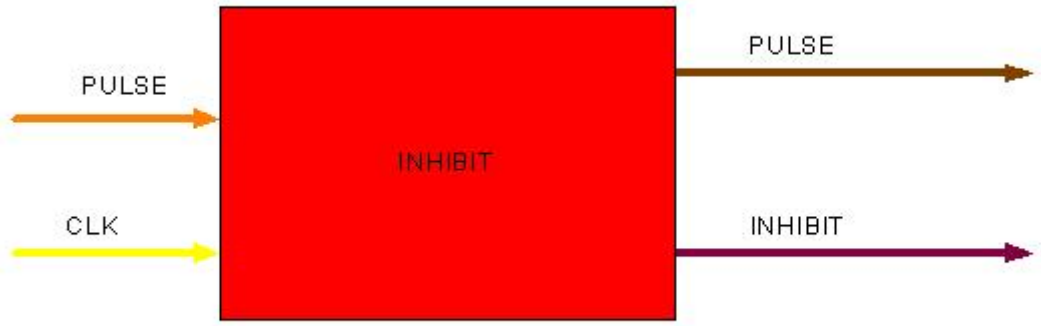


Figure 20: Inhibit block

Inputs: CLK: External input, to synchronize the system.
pul_sal: Internal input, from the addition/comparison block.
Enable: Internal output, to the timer blocks.
PULSE: External output, pulse generated.

Code: inhibit.vhd

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;
   use IEEE.std_logic_unsigned.all;
   use IEEE.math_real.all;

entity contador_inhib is port(
clk:      in std_logic;
```

```
pul_sal: in std_logic;
conta: buffer real:=0.0;
enable: out std_logic;
PULSE: out std_logic
);
end contador_inhib;

architecture archicontador of contador_inhib is
begin
process (pul_sal, clk)
  begin
  PULSE <= pul_sal;
  if (pul_sal'event and pul_sal='1') then
  conta <= 0.0;
  enable<= '1';
  else
   if (clk'event and clk= '1') then
    conta <= conta + 0.1;
    if (conta > 0.8) then
       enable <= '0';
    end if;
  end if;
 end if;
end process;
end archicontador;
```

The behaviour of this block is the following. When receives a rise edge it resets the counter, actives the enable signal and starts the count. The enable signal will be active until the counter of the timer arrives to 2 ms. At this moment, the enable signal is deactivated. When it receives other rise edge the systems will do the same, reset the counter, active the enable and start the count.

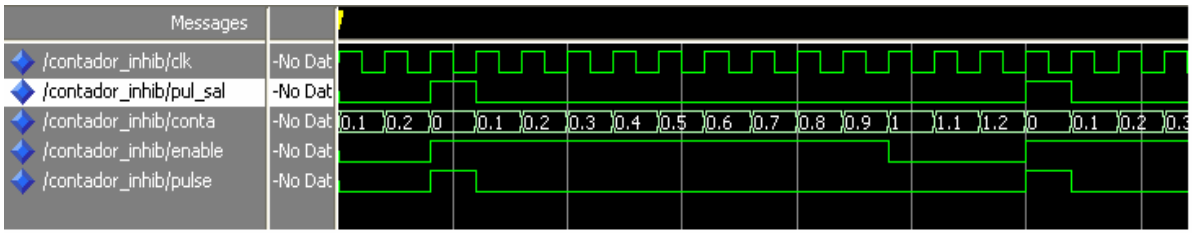


Figure 21: Inhibit block simulation. *In this simulation we can observe the right behaviour of the inhibit block. When the signal "pul_sal" is active, the signal "conta" starts the count. At this moment, the signal "enable" remains active until the count of "conta" arrives to the value 1.0.*

5.5 ASSEMBLING BLOCKS:

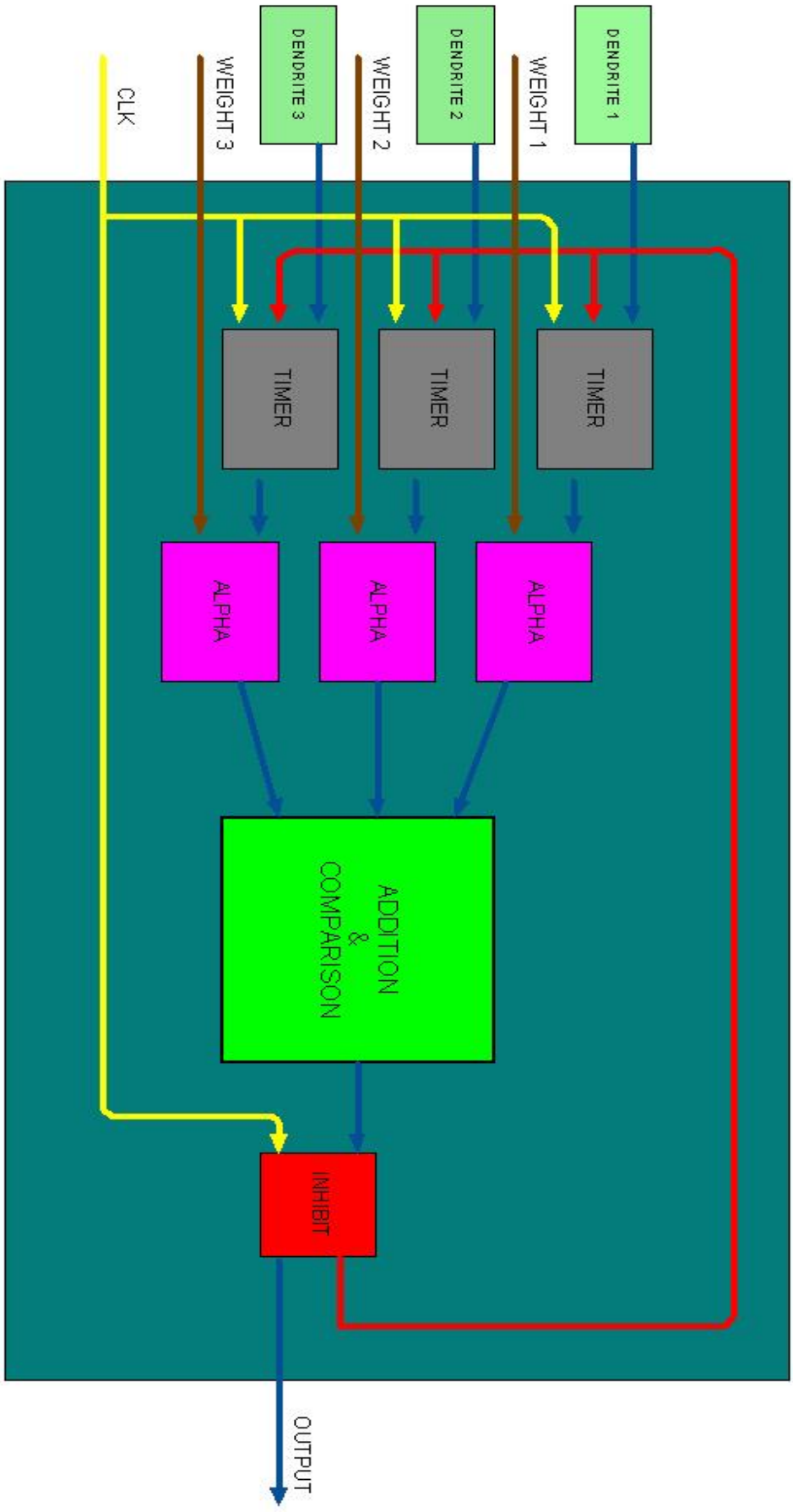
Our system will have 3 dendrites; so, will need 3 potential generators, which means, 3 timers and 3 alpha blocks whose behaviours and structure will be similar that the original block explained before.

Furthermore, to make the interconnection between blocks we need defined some variables that link the outputs of some blocks with the inputs of the followings blocks.

```
SIGNAL x1:real;                    -- Variables to join the other blocks
    SIGNAL x2:real;
    SIGNAL x3:real;
    SIGNAL y1:real;
    SIGNAL y2:real;
    SIGNAL y3:real;
    SIGNAL z:std_logic;
    SIGNAL inh:std_logic;

      BEGIN
        U1a:c1 port map(CLK,neuron1,inh,x1);      -- Interconnection of the
        U1b:c2 port map(CLK,neuron2,inh,x2);      -- different blocks
        U1c:c3 port map(CLK,neuron3,inh,x3);
        U2a:alpha1 port map(x1,Weight1,y1);
        U2b:alpha2 port map(x2,Weight2,y2);
        U2c:alpha3 port map(x3,Weight3,y3);
        U3:threshold port map(y1,y2,y3,z);
        U4:inhibit port map (CLK,z,inh,PULSE);

      END structure;

-- Description: Test to verify quickly if the system meets with the requirements.
-- It emulates the behaviour of the difference inputs that arrives to the neuron modifies
-- The values to check if in all the situations the output is correct.
```

With the aim of simulating the right behaviour of the main program, we need a testbech. A testbench is a virtual environment used to verify a model or a design. We have designed a program that emulates a possible case that could be happened, it represents the possible changes in the different inputs and observe the output to check if the system meets with the requirements. This program is named is tech_bench_global.vhd (Appendix B).

Figure 22: Global system. *Global system is represented in the following figure. We can observe the 3 inputs from the dendrites and their respective weights. The other input showed in the figure is the "clk" input, needed to synchronize the counter of the timer and inhibit blocks.*


CLK
WEIGHT 3
WEIGHT 2
WEIGHT 1
DENDRITE 3
DENDRITE 2
DENDRITE 1
TIMER
TIMER
TIMER
ALPHA
ALPHA
ALPHA
ADDITION
&
COMPARISON
INHIBIT
OUTPUT

6. Conclusions

During several weeks going thought the biological neuron system, we have concluded some important things, as the following ones.

The basic behaviour of the biological neuron can be emulated in an artificial neuron.
A biological neuron with their dendrites, soma and axon can be characterized in an artificial neuron as a black box with inputs and an output.

To implement the system the electronic pulses or spikes transmitted through neurons are replaced by digital signals or pulses.

We can get to emulate the potential actions in the presynaptic and postsynaptic reactions using timers and multiplier blocks.

With all these things we get an electronic system that reproduces the behaviour of the biological neuron. The aim is to have the possibility of interconnect more of these artificial neurons to create a complete neuronal network.

In this thesis work we only have focused our efforts to create an only artificial neuron. To complete the global system is not too complicate, to interconnect the artificial neuron with other neuron; the programmer should only connect the output of our neuron with the input of the next neuron, and in this way, with the other previous and next neurons. Besides, the programmer should codify a program that governs the relations between the different weights of the neuronal network, but this part is out of our analysis.

With the VHDL code generated a FPGA can be programmed. Depending of the capacity of the FPGA used, a larger o less number of neuron can be programmed, depending on the same way of the number of interconnections of each neuron.

Therefore, according to the results obtained, we can say that we have designed a system whose performance leads to the achievement of the objectives of the project and at the same time could work as the main base to develop future applications.

7. References

[1] http://www.corticaldb.com/projects.asp, May 2008.

[2] http://icwww.epfl.ch/~gerstner/SPNM/node3.html, March 2008.

[3] http://icwww.epfl.ch/~gerstner/SPNM/node4.html, March 2008.

[4] http://www.encyclopedia.com/doc/1O87-refractoryperiod.html , March 2008.

[5] http://www.vhdl-online.de/tutorial/, April 2008.

[6] Perry, Douglas L. 1998, *VHDL*, third edition, McGraw-Hill, New York, USA, ISBN 0-07-049436-3

[7] http://www.eng.auburn.edu/department/ee/mgc/vhdl.html April 2008

Appendix A

```
----------------------------------------------------------------------------------------------------
-- Title: Modelling spiking neurons in VHDL
-- Project: Thesis work in Mälardalens University
----------------------------------------------------------------------------------------------------
-- File: neuron.vhd
-- Author: Javier Gómez Casado
-- Created: April 2008
----------------------------------------------------------------------------------------------------
-- Description : Implements a neuron in VHDL considering the difference signal created
-- when a neuron is fired.
-- Inputs: neuron1, neuron2, neuron3, Weight1, Weight2, Weight3, CLK.
-- Output: PULSE
----------------------------------------------------------------------------------------------------
-- Notes:
-- Refractory Period -> Time inhibit = 2 ms.
-- Threshold = 20mV
-- Variables weights
----------------------------------------------------------------------------------------------------


library IEEE;                           -- Declaration of libraries
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity neuron is                        -- Implementation of the neuron
port(
neuron1:in std_logic;                   -- 3 inputs, one for each dendrite.
neuron2:in std_logic;
neuron3:in std_logic;
Weight1: in real:=1.0;                  -- 3 variable weights, one for each
Weight2: in real:=2.0;                  -- input
Weight3: in real:=3.0;
CLK: in std_logic;                      -- one clock signal to make the system
                                        -- synchronous
PULSE: out std_logic                    -- one output, represents the axon of the neuron
);
end entity neuron;


architecture structure of neuron is
COMPONENT c1
port(
clk: in std_logic;                      -- Clock signal
input1: in std_logic;                   -- Dendrite 1
```

```
enable: in std_logic;                -- Enable/Disable, depends of the inhibit time
count1: buffer real:=0.0             -- Result of the count
);
END COMPONENT;

COMPONENT c2                 -- The same as c1 but for the dendrite 2
port(
clk: in std_logic;
input2: in std_logic;
enable: in std_logic;
count2: buffer real:=0.0
);
END COMPONENT;

COMPONENT c3                 -- The same as c3 for the dendrite 3
port(
clk: in std_logic;
input3: in std_logic;
enable: in std_logic;
count3: buffer real:=0.0
);
END COMPONENT;

COMPONENT alpha1             -- Generate a output in function of the time for
                             -- the dentrite 1
port(
A:in real :=3.0;
Weight1: in real:=1.0;
RESULT_A: out real:=0.0
);
END COMPONENT;

COMPONENT alpha2             -- For the dentrite 2
port(
B:in real :=3.0;
Weight2: in real:=2.0;
RESULT_B: out real:=0.0
);
END COMPONENT;

COMPONENT alpha3             -- For the dentrite 3
port(
C:in real :=3.0;
Weight3: in real:=3.0;
RESULT_C: out real:=0.0
);
END COMPONENT;

COMPONENT threshold
port(
```

```
A1:in real :=1.0;       -- Signal generates by alpha function over the dendrite1
B1:in real :=2.0;       -- Signal generates by alpha function over the dendrite2
C1:in real :=0.0;       -- Signal generates by alpha function over the dendrite3
SAL: out std_logic      -- Result of the comparison
);
END COMPONENT;

COMPONENT inhibit
port(
CLK: in std_logic;              -- Clock signal
pul_sal: in std_logic;          -- Result of comparison
enable: out std_logic;          -- Signal of inhibit
PULSE: out std_logic;           -- Axon
conta: buffer real:=0.0
);
END COMPONENT;

SIGNAL x1:real;                 -- Variables to join the other blocks
SIGNAL x2:real;
SIGNAL x3:real;
SIGNAL y1:real;
SIGNAL y2:real;
SIGNAL y3:real;
SIGNAL z:std_logic;
SIGNAL inh:std_logic;

BEGIN
U1a:c1 port map(CLK,neuron1,inh,x1);      -- Interconnection of the
U1b:c2 port map(CLK,neuron2,inh,x2);      -- different blocks
U1c:c3 port map(CLK,neuron3,inh,x3);
U2a:alpha1 port map(x1,Weight1,y1);
U2b:alpha2 port map(x2,Weight2,y2);
U2c:alpha3 port map(x3,Weight3,y3);
U3:threshold port map(y1,y2,y3,z);
U4:inhibit port map (CLK,z,inh,PULSE);
END structure;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
entity c1 is
port(
CLK: in std_logic;
input1: in std_logic;
enable: in std_logic;
count1: buffer real:=0.0
);
end c1;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
entity c2 is
port(
CLK: in std_logic;
input2: in std_logic;
enable: in std_logic;
count2: buffer real:=0.0
);
end c2;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
entity c3 is
port(
CLK: in std_logic;
input3: in std_logic;
enable: in std_logic;
count3: buffer real:=0.0
);
end c3;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
entity alpha1 is
port(
A:in real :=0.0;
Weight1: in real :=1.0;                --Variable weights
RESULT_A: out real:=0.0);
end entity alpha1;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
entity alpha2 is
port(
B:in real :=0.0;
Weight2: in real :=2.0;                --Variable weights
RESULT_B: out real:=0.0);
end entity alpha2;
library IEEE;
use IEEE.std_logic_1164.all;
```

```
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
entity alpha3 is
port(
C:in real :=0.0;
Weight3: in real :=3.0;                    --Variable weights
RESULT_C: out real:=0.0);
end entity alpha3;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity threshold is
port(
A1:in real :=1.0;
B1:in real :=2.0;
C1:in real :=0.0;
SAL: out std_logic);
end entity threshold;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
entity inhibit is
port(
CLK: in std_logic;
pul_sal: in std_logic;
enable: out std_logic;
PULSE: out std_logic;
conta: buffer real:=0.0
);
end inhibit;
ARCHITECTURE archicount1 of c1 is       -- Behaviour of the first timer
begin
process (input1, clk)
variable pulso_rec: real :=0.0;
begin
if (enable = '1') then                  -- If the signal of inhibit is active,
count1 <= 0.0;                          -- reset the count
pulso_rec :=0.0;
else
if (input1'event and input1='1') then   -- When the input is active, active
count1 <= 0.0; -- active the count
pulso_rec := 1.0;
end if;
if ((clk'event and clk= '1') and pulso_rec= 1.0) then  -- when the count
count1 <= count1 + 0.1;                                -- is active, with each
end if;                                                -- rise edge of clock
```

```
end if;                                          -- the count goes up
end process;
END archicount1;
ARCHITECTURE archicount2 of c2 is                -- Behaviour of the second timer
Begin                                            -- is the same as the first one but
process (input2, clk)                            -- using the input2
variable pulso_rec: real :=0.0;
begin
if (enable = '1') then
count2 <= 0.0;
pulso_rec :=0.0;
else
if (input2'event and input2='1') then
count2 <= 0.0;
pulso_rec := 1.0;
end if;
if ((clk'event and clk= '1') and pulso_rec= 1.0) then
count2 <= count2 + 0.1;
end if;
end if;
end process;
END archicount2;
ARCHITECTURE archicount3 of c3 is        -- Behaviour of the third timer
Begin                                    -- is the same as the first one but
process (input3, clk)                     -- using the input3
variable pulso_rec: real :=0.0;
begin
if (enable = '1') then
count3 <= 0.0;
pulso_rec :=0.0;
else
if (input3'event and input3='1') then
count3 <= 0.0;
pulso_rec := 1.0;
end if;
if ((clk'event and clk= '1') and pulso_rec= 1.0) then
count3 <= count3 + 0.1;
end if;
end if;
end process;
END archicount3;
architecture archalpha1 of alpha1 is     -- Generate the signal form when a
begin                                    -- dendrite is active
RESULT_A <= exp(-A)*A*Weight1;           --Variable weights
end architecture archalpha1;
architecture archalpha2 of alpha2 is
begin
RESULT_B <= exp(-B)*B*Weight2;           --Variable weights
end architecture archalpha2;
architecture archalpha3 of alpha3 is
```

```
begin
RESULT_C <= exp(-C)*C*Weight3;          --Variable weights
end architecture archalpha3;
architecture archithres of threshold is                  -- Output only is active when the
begin                                                    -- addition of all the input is over
SAL<='1' when ((A1+B1)+C1)>20.0 else '0';        -- the threshold -> 20mV
end architecture archithres;
architecture archinhibit of inhibit is
begin
process (pul_sal, CLK)                          -- When the input is active
begin                                           -- active the output
PULSE <= pul_sal;                               -- until is over the inhibit
if (pul_sal'event and pul_sal='1') then         -- time
conta <= 0.0;
enable<= '1';
else
if (clk'event and clk= '1') then
conta <= conta + 0.1;
--if (conta > time_inhib) then
if (conta > 2.0) then                   -- refractory period -> 2ms
enable <= '0';
end if;
end if;
end if;
end process;
end archinhibit;
```

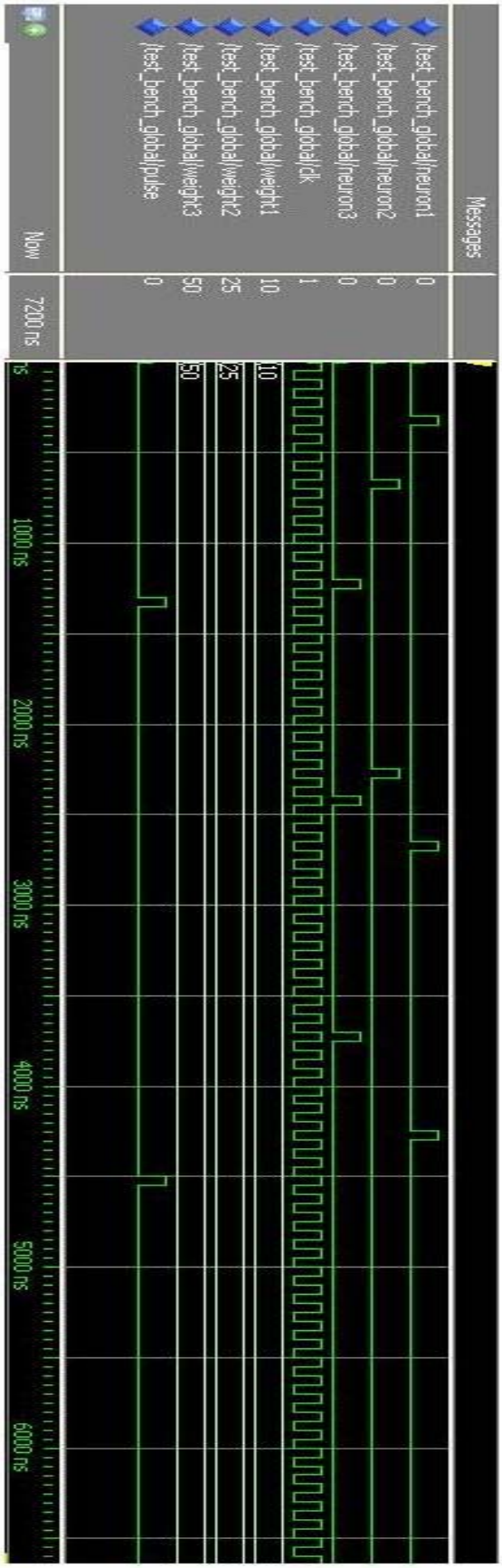
Appendix B

```
---------------------------------------------------------------------------------------------------------
-- Title: Modelling spiking neurons in VHDL
-- Project: Thesis work in Mälardalens University
---------------------------------------------------------------------------------------------------------
-- File: tech_bench_global.vhd
-- Author: Javier Gómez Casado
-- Created: April 2008
---------------------------------------------------------------------------------------------------------
-- Description : Test to verify quickly if the system meets with the requirements.
-- It emulates the behaviour of the difference inputs that arrives to the neuron modifies
-- the values to check if in all the situations the output is correct.
---------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;

ENTITY test_bench_global Is
end test_bench_global;

architecture test_global OF test_bench_global IS
COMPONENT neuron
port (
neuron1:in std_logic; .
neuron2:in std_logic;
neuron3:in std_logic;
Weight1: in real:=1.0;
Weight2: in real:=2.0;
Weight3: in real:=3.0;
CLK: in std_logic;
PULSE: out std_logic
);
end component neuron;

signal neuron1, neuron2, neuron3 , CLK, PULSE: std_logic;
signal Weight1, Weight2, Weight3 : real;

begin
n: neuron port map(
neuron1 => neuron1,
neuron2 => neuron2,
neuron3 => neuron3,
Weight1 => Weight1,
Weight2 => Weight2,
```

```
Weight3 => Weight3,
CLK => CLK,
PULSE => PULSE);

process
begin
CLK <= '1'; -- Clock frequency of 10 Khz
wait for 50 ns;
CLK <= '0';
wait for 50 ns;
end process;
PROCESS
BEGIN
neuron1 <= '0'; -- Check the output for different inputs.
neuron2 <= '0';
neuron3 <= '0';
Weight1 <= 10.0;
Weight2 <= 25.0;
Weight3 <= 50.0;
wait for 300 ns;
neuron1 <= '1';
wait for 50 ns;
neuron1 <= '0';
wait for 300 ns;
neuron2 <= '1';
wait for 50 ns;
neuron2 <= '0';
wait for 500 ns;
neuron3 <= '1';
wait for 50 ns;
neuron3 <= '0';
wait for 1000 ns;
neuron2 <= '1';
wait for 50 ns;
neuron2 <= '0';
wait for 100 ns;
neuron3 <= '1';
wait for 50 ns;
neuron3 <= '0';
wait for 200 ns;
neuron1 <= '1';
wait for 50 ns;
neuron1 <= '0';
wait for 1000 ns;
neuron3 <= '1';
wait for 50 ns;
neuron3 <= '0';
wait for 500 ns;
neuron1 <= '1';
wait for 50 ns;
```

```
neuron1 <= '0';
wait for 400 ns;
wait;
end process;
end;
```

Appendix C


Messages
/test_bench_global/neuron1
/test_bench_global/neuron2
/test_bench_global/neuron3
/test_bench_global/clk
/test_bench_global/weight1
/test_bench_global/weight2
/test_bench_global/weight3
/test_bench_global/pulse
Now
7200 ns
0
0
0
1
10
25
50
0
10
25
50
ns
1000 ns
2000 ns
3000 ns
4000 ns
5000 ns
6000 ns