Digital Hardware Implementation of Artificial Neurons Models Using FPGA

Rafid Ahmed Khalil rafidmori@yahoo.com

Sa'ad Ahmed Al-Kazzaz saad_kazzaz@yahoo.com

Department of Electrical Engineering, University of Mosul, Mosul, Iraq

Abstract

This paper present the digital implementation of multiply-accumulate (MAC) circuit of artificial neuron using FPGA (Field Programmable Gate Array) including three types of nonlinear activation functions: hardlims, satlins and tansig. A VHDL hardware description Language codes are used to implement the neuron using XC3S500E-FG320 Xilinx FPGA device. The simulation results obtained with Xilinx Foundation 8.2i software are presented. The results are analyzed in terms of usage percentage of chip resources and maximum working frequency.

تنفيذ الكيان المادي الرقمي لنماذج خلايا عصبية اصطناعية باستخدام مصفوفة البوابات المبرمجة حقليا

رافد احمد خليل **سعد أحمد القزاز**

قسم الهندسة الكهربائية, جامعة الموصل

الخلاصة

يقدم هذا البحث طريقة لتنفيذ الكيان المادي الرقمي المبني على مصفوفة البوابات المبرمجة حقليـــاً (FPGA) لدائرة ضرب - تجميع (MAC) لخلية عصبية اصطناعية ولثلاثة أنواع مختلفة من الدوال التفعيل الغير خطية :- (hardlims , satlins , tansig) . حيث تم تطوير برمجيات لتنفيذ الخلايا العصبية الاصطناعية على شريحة (FPGA نوع XC3S500E-FG320 إنتاج شركة Xilinx) . ان كافة نتائج المحاكات لإشـــارات الإدخـــال والإخـــراج المختلفة التي عرضت في هذا البحث ، تم الحصول عليها باستخدام بيئة العمـــل البرمجيـــة Xilinx Foundation 8.2iوالخاصة بشريحة (FPGA) المستخدمة. وقد تم تحليل كافة نتائج التنفيذ المادي بالاعتماد على النسب المئوية لاستهلاك الموارد المادية لشريحة (FPGA) وكذلك القيمة العظمى لتردد الاشتغال لكل نموذج منفذ .

Received 10 Dec. 2007 **Accepted 16 May 2008**

1. Introduction

Artificial neural networks (ANNs) have been used successfully in solving pattern classification and recognition problems, function approximation and predictions. Their processing capabilities are based on their highly, parallel and interconnected architecture. Such characteristics make their implementation enormous challenging, and also very costly, due to the large amount of hardware required [1].

Digital implementation of ANNs may be performed using different tools such as custom design, digital signal processor (DSP), programmable logic …etc. Among them, programmable logic offers low cost, powerful software development tools and true parallel implementation [2].

Field Programmable Gate Array (FPGA) are a family of programmable device based on an array of configurable logic blocks (CLBs), which gives a great flexibility in prototyping, designing and development of complex hardware real time systems . The structure of a FPGA can be described as an "array of blocks" connected togerther via programmable interconnections. The main advantage of FPGA is the flexibility that they afford [3]. Xilinx Inc. introduced the world's first FPGA, the XC2064 in 1985. The XC2064 contained approximately 1000 logic gate. Since then, the gate density of Xilinx FPGAs has increased thousands times [4]. Recently there is a lot of interest in the FPGA realization of neural networks which is reported by many researchers [1, 5-8].

In the present work, we introduced the design of an artificial neuron models based on a XC3S500E Xilinx FPGA device. The XC3S500E Xilinx FPGA device has high gate density i.e. 500,000 logic gate and many features, as illustrated below [9], which are necessary for neural implementation:

- Fast logic enable the design of compact and fast arithmetic functions (i.e., multiplication and addition).
- Look up tables can be used as RAMs and ROMs.
- Combinational functions have up to ten inputs within configurable logic blocks (CLBs), and delays are very small and almost independent on the number of variable.
- Very high routing capabilities allows successful implementation of critical path delays, even for complex neural network [1].

The Very high speed integrated circuit Hardware Description Language (VHDL) is heavily used by large corporations, majority of companies as well as universities for FPGA programming.VHDL was first adopted as language standard in 1987, with a major revision occurring in 1993, 2001 [10]. VHDL is very powerful (HDL) but very complex syntax language. VHDL simplifies the development of complex system such as ANNS, because it is possible to model and simulate a digital system from a high level of abstraction with important facilities for modular design [2].

The purpose of this work is to design an artificial neuron model. The model consist of two stages .The first is the MAC stage for multiplication and accumulation of parallel inputs and weights values . And the second is the nonlinear activation function for the output signal .Three types of activation functions were considered : symmetrical hard limiter, symmetric saturating linear, and hyperbolic tangent sigmoid (referred to as hardlims, satlins, and tansig in Matlab software package receptivity).

2. Mathematical model of an artificial neuron

The common Mathematical model of an artificial neuron is shown in Fig.(1) [11] .

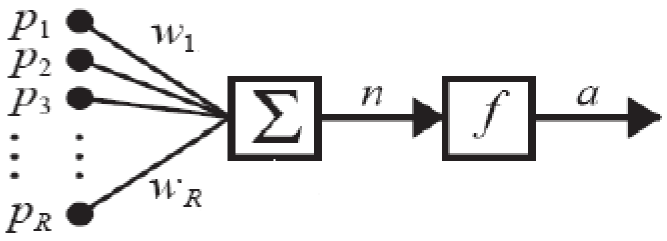


Fig.(1) mathematical model of artificial neuron .

The neuron output can be written as :

$$a = f\left(\sum_{j=1}^{R} w_j\, p_j\right) \qquad \text{………………..…. (1)}$$

where p_j is the input value and w_j is the corresponding weight value , a is the output of the neuron, and $f\ (\ )$ is a nonlinear activation function. Typically the activation function is chosen by the designer for specific training algorithm , and then the weights will be adjusted by some learning rule so that the neuron input / output relationship meet some specific goal.

3. VHDL design of the neuron

It is important to design the neuron without activation function as common part in designing a complete neuron with any activation function based on FPGA The design affect the utilization ratio of the chips area and the processing speed directly. The structure of the neuron can be realized in many ways , mainly considering the degree of the parallel computation needed .

The proposed VHDL structural diagram for hardware implementation of neuron is shown in Fig (2). The structure contains two shift registers , one shifters hold the weights , while the other holds the inputs (shift register with data load capability) .This approach is appropriate for general purpose neuron (i.e., with programmable weights) .It employs only one input to load all weights (thus saving on chip pins) . The weights are shifted in sequentially until the register is loaded with its weight . The weights are then multiplied by the input and accumulated to produce the desired output .

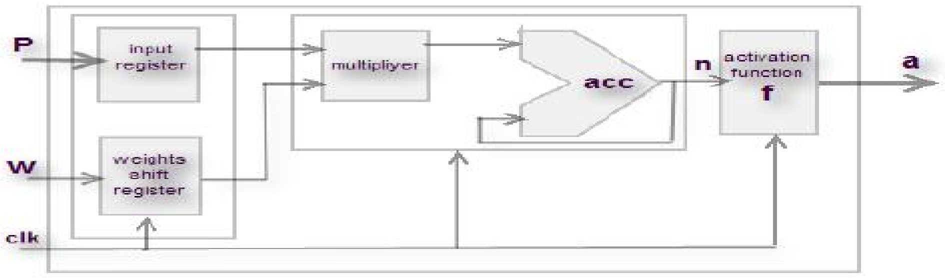
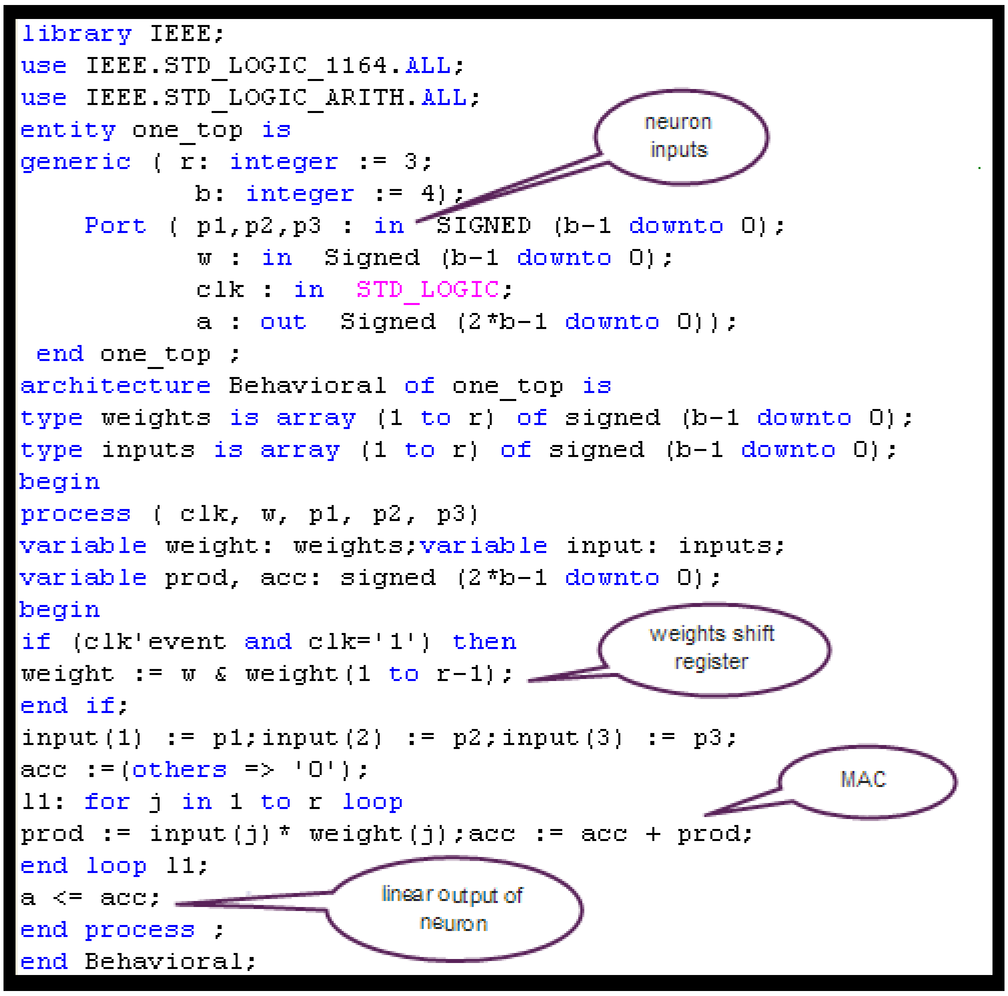


Fig.(2) VHDL structural diagram for neuron implementation.

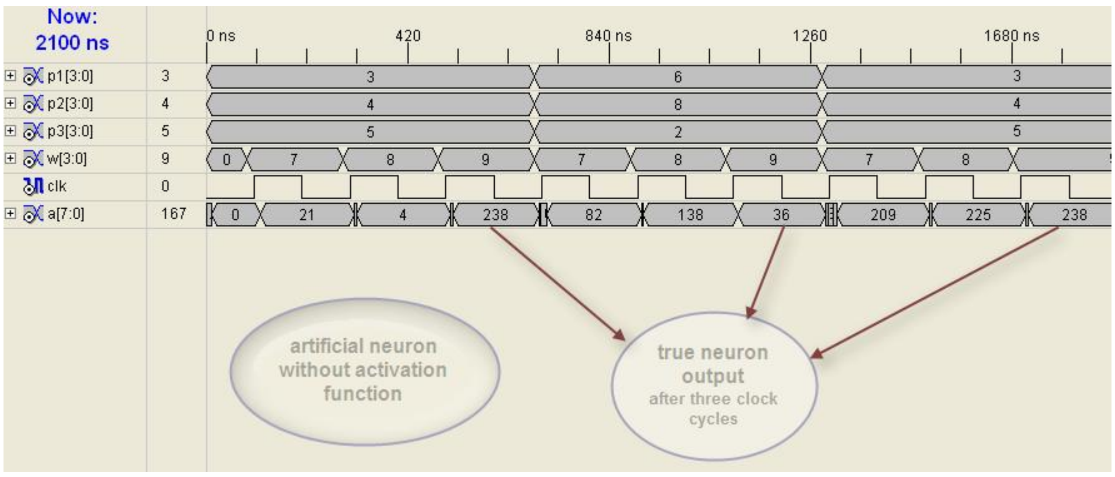
The VHDL code used for the implementation of a neuron without activation function is presented in table (1).

Table (1) VHDL code for implementing neuron without activation function (linear neuron).

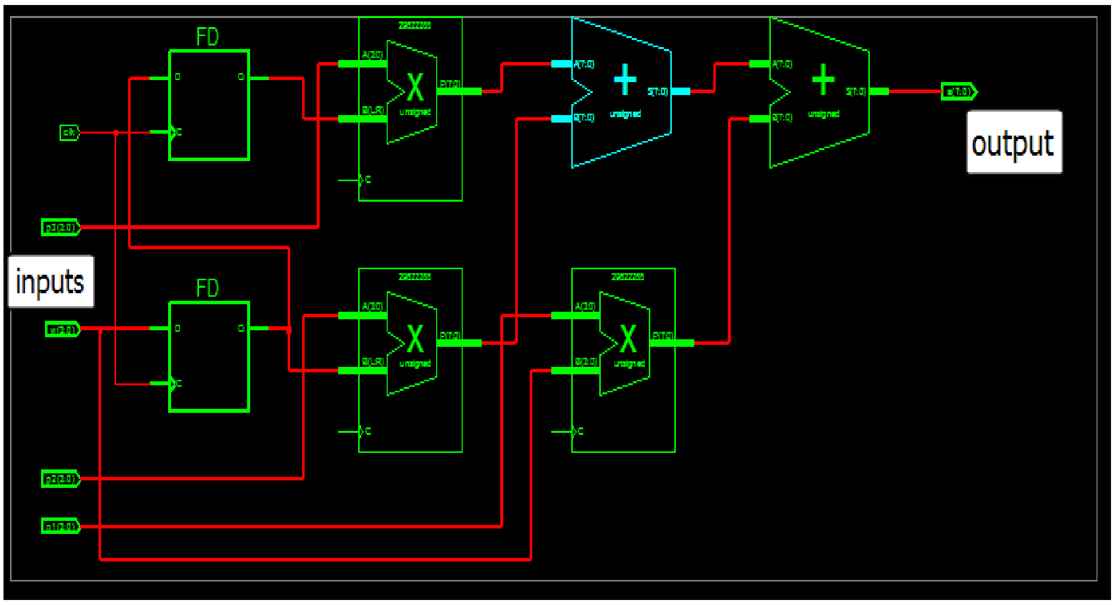


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity one_top is
generic ( r: integer := 3;
           b: integer := 4);
    Port ( p1,p2,p3 : in  SIGNED (b-1 downto 0);
              w : in  Signed (b-1 downto 0);
              clk : in  STD_LOGIC;
              a : out  Signed (2*b-1 downto 0));
 end one_top ;
architecture Behavioral of one_top is
type weights is array (1 to r) of signed (b-1 downto 0);
type inputs is array (1 to r) of signed (b-1 downto 0);
begin
process ( clk, w, p1, p2, p3)
variable weight: weights;variable input: inputs;
variable prod, acc: signed (2*b-1 downto 0);
begin
if (clk'event and clk='1') then
weight := w & weight(1 to r-1);
end if;
input(1) := p1;input(2) := p2;input(3) := p3;
acc :=(others => '0');
l1: for j in 1 to r loop
prod := input(j)* weight(j);acc := acc + prod;
end loop l1;
a <= acc;
end process ;
end Behavioral;
```

Simulation results is shown in Fig.(3a) .The neuron has three 4- bit input each .since a SIGNED data representation was employed , the range of the input values and weights runs from (-8 to 7) and the range of the 8-bit output runs from (-128 to 127) . The first input vector applied to the neuron has the values $p_1=3,\ p_2=4,\ and\ \ p_3=5$, since there are three wights , three clock cycles are needed to shift them in ,as shown in Fig (3 a) .The values chosen for the weights were $w_3=7, w_2=8, w_1=9$. Note that 9 is in indeed -7 , and 8 is -8 because data type used here is SIGNED . Consequently, the weight have been all loaded , the system immediately gives its output, i.e., $a = p_1 w_1 + p_2 w_2 + p_3 w_3 = (3)(-7) + (4)(-8) + (5)(7) = -18$ represent as $256 - 18 = 238$. The neuron output for the second input vector [6 8 2] is 36. Fig.(3b) shows the RTL (register transfer level) hardware schematic circuit for implementing linear neuron.



(a)



(b)

Fig.3 (a) Time diagram , (b) Hardware circuit, for implementing an 8- bit linear artificial neuron (without activation function).

4. VHDL design of activation functions

The activation function in Fig.(1) may be linear or nonlinear function of n. Aparticular activation function of neuron is chosen to satisfy specification of the training algorithm that the neural network is attempted to run. In this work, three of the most commonly used activation functions are hardware implemented on FPGA using VHDL language.

4.1 The symmetrical hard limit activation function

The symmetric hard limit transfer function referred to as " hardlims " in matlab . It is used to classify input into two distinct categories , and can be defined as follows [11]:

$$a = \begin{cases} -1 & n \prec 0 \\ 1 & n \geq 0 \end{cases} \qquad \text{……………………..(2)}$$

This function is shown in Fig (4) . it is used with MLP, and Hopfield neural networks **:-**

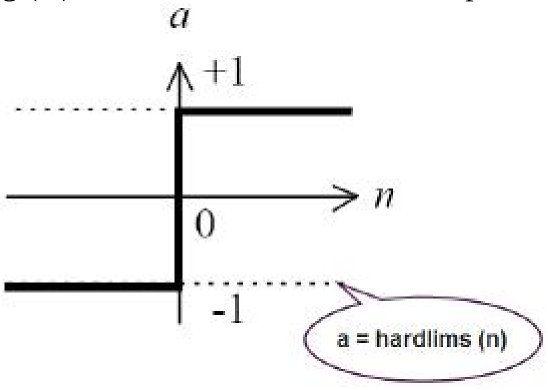
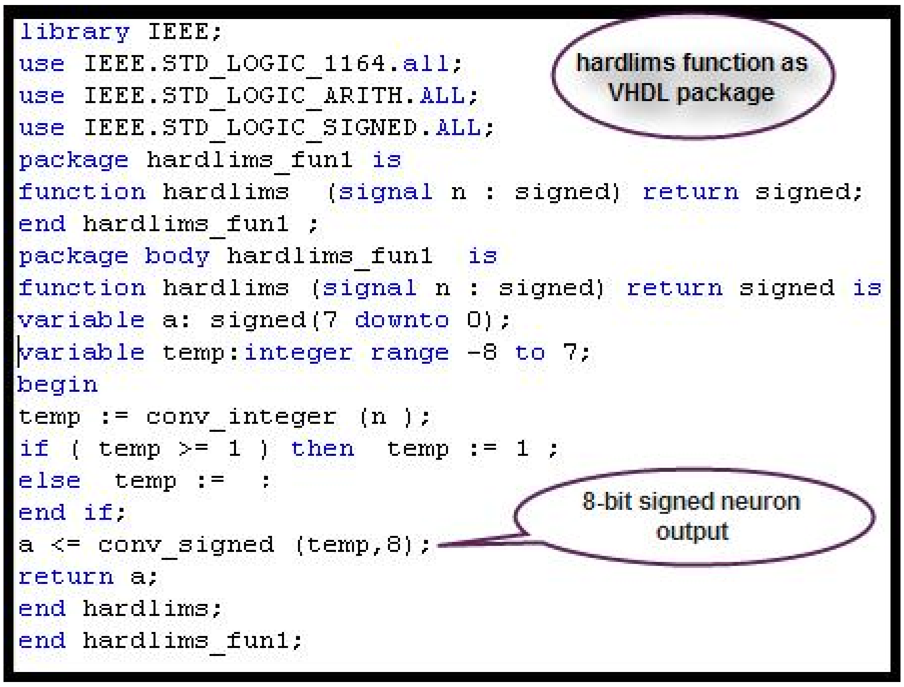


Fig.(4) Symmetrical hard limit activation function

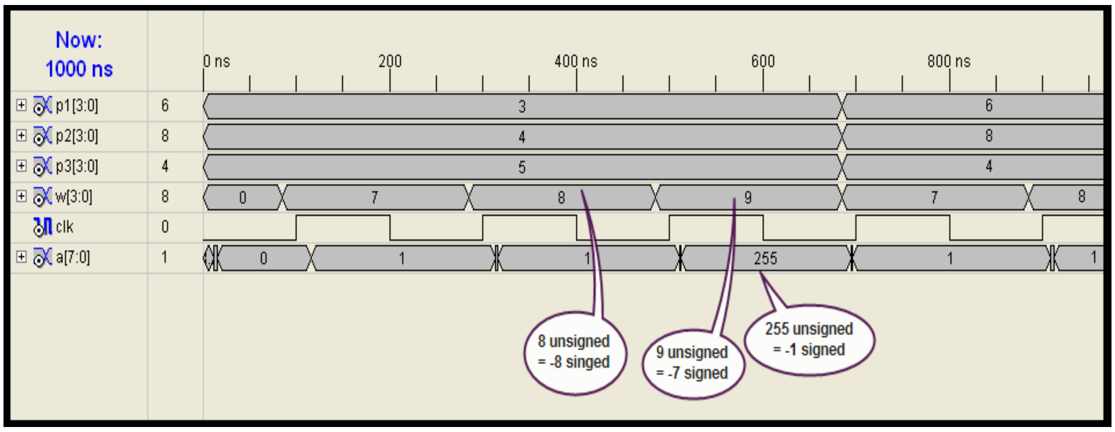
The VHDL code used to implement this function is shown in table (2), as VHDL package.

Table (2) VHDL code for implementing hardlims function as a package .

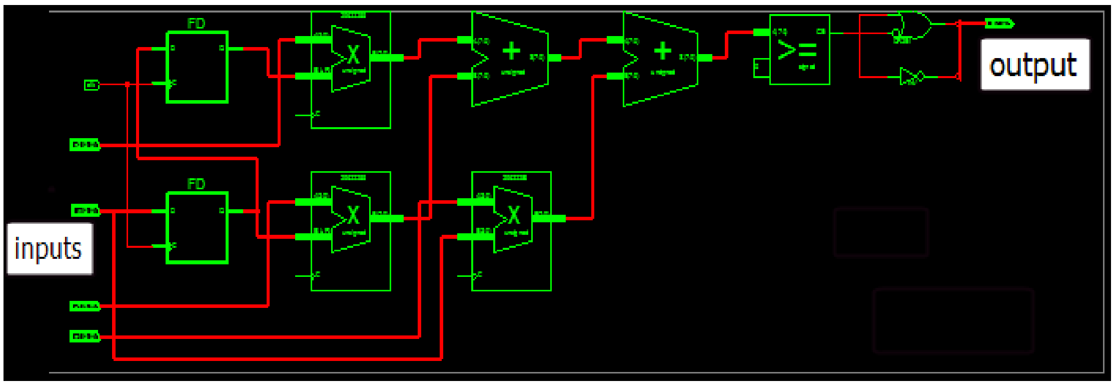


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
package hardlims_fun1 is
function hardlims  (signal n : signed) return signed;
end hardlims_fun1 ;
package body hardlims_fun1  is
function hardlims (signal n : signed) return signed is
variable a: signed(7 downto 0);
variable temp:integer range -8 to 7;
begin
temp := conv_integer (n );
if ( temp >= 1 ) then  temp := 1 ;
else  temp :=  :
end if;
a <= conv_signed (temp,8);
return a;
end hardlims;
end hardlims_fun1;
```

For simulating purposes the neuron inputs p_i and weight w_i were represented as signed 4-bit, the output of MAC n as signed 8-bit, and neuron output a represented as std-logic 1-bit . The time diagram for artificial neuron with hardlims activation function is shown in Fig.(5a). Fig.(5b) shows the RTL hardware schematic circuit for implementing hardlims neuron.



(a)



(b)

Fig.5 (a) Time diagram, (b)Hardware circuit ,of implementing an 8-bit neuron with hardlims activation function .

4.2 The saturating linear activation function

The output of saturating linear activation function " satlins ", can be defined as follows [11] :

$$a = \begin{cases} -1 & n \prec -1 \\ n & -1 \le n \le 1 \\ 1 & n \succ 1 \end{cases} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

as illustrated in Fig (6) . Neuron with this activation function are used in the ADALINE neural networks .

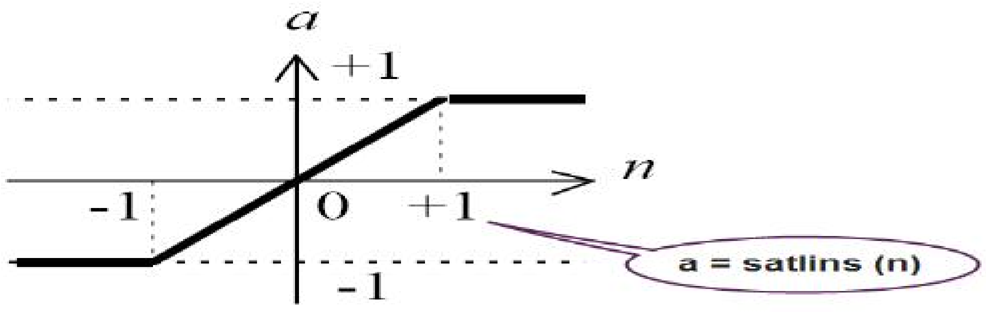


Fig. (6) Saturating linear activation function.

Table 3 shows the VHDL codes (as a package) used for implementing the " saltins " activation function.

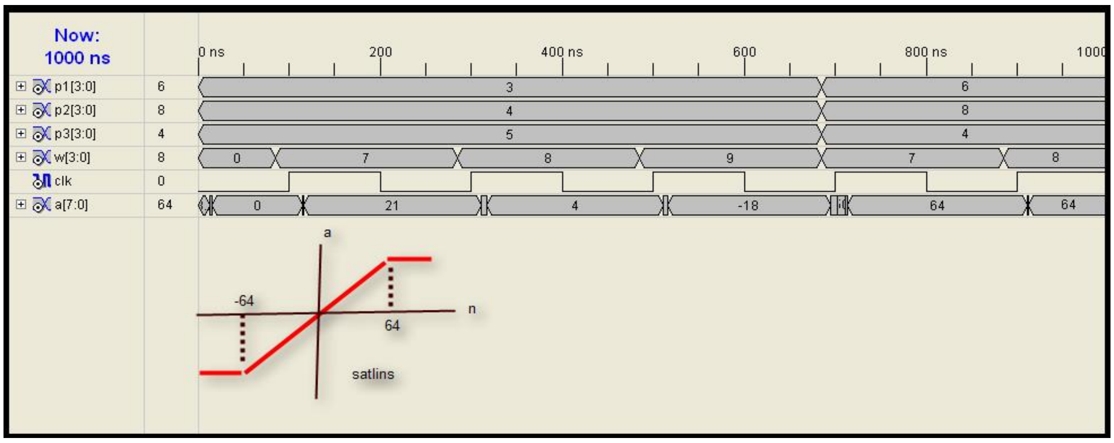
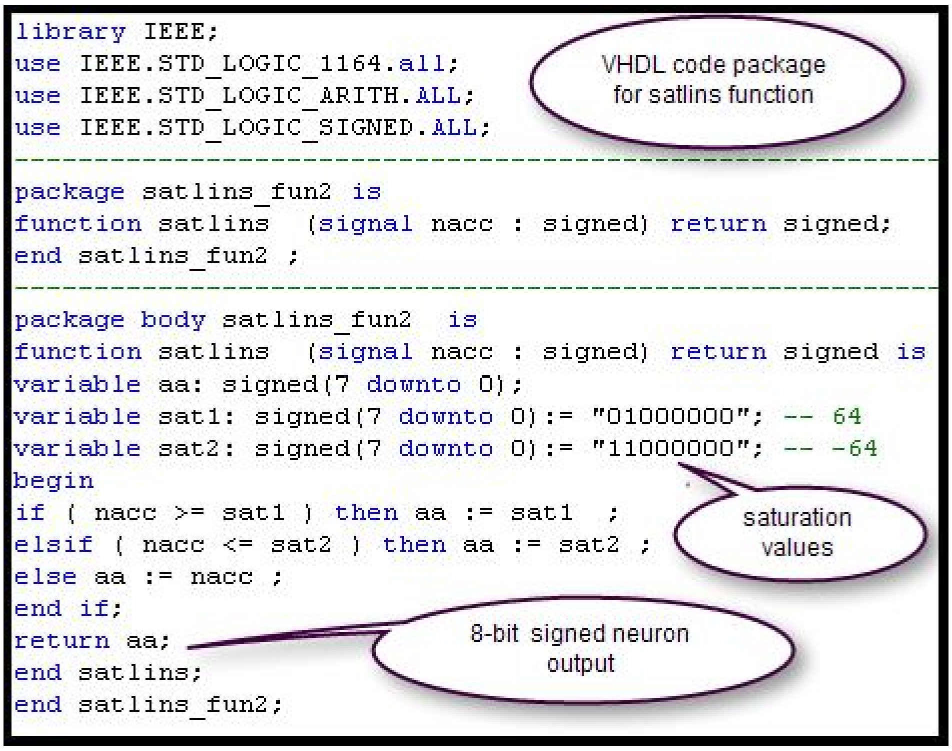


Table (3) VHDL code for implementing satlins activation function as a package .

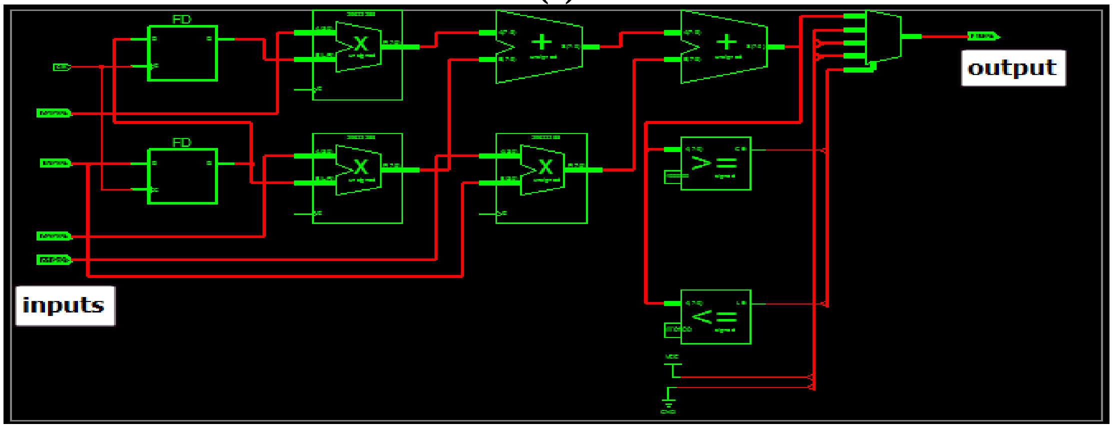


```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
--------------------------------------------------------------------
package satlins_fun2 is
function satlins  (signal nacc : signed) return signed;
end satlins_fun2 ;
--------------------------------------------------------------------
package body satlins_fun2  is
function satlins  (signal nacc : signed) return signed is
variable aa: signed(7 downto 0);
variable sat1: signed(7 downto 0):= "01000000"; -- 64
variable sat2: signed(7 downto 0):= "11000000"; -- -64
begin
if ( nacc >= sat1 ) then aa := sat1  ;
elsif ( nacc <= sat2 ) then aa := sat2 ;
else aa := nacc ;
end if;
return aa;
end satlins;
end satlins_fun2;
```

VHDL code package for satlins function
saturation values
8-bit signed neuron output

The resulthng simulation timing diagram of the neuron with satlins activation function is shown in Fig.(7a) . where 4-bit are assigned to the input p_j , 4- bit to the weight w_j , 8-bit to the input of the activation function, and 8- bit to the output a . Note that all data are of singed type. It can be seen that the output of the neuron a equal the output of MAC n as long as it is lower to the saturation level .Once the sum equals or becomes greater than the saturation level the output remains constant and equal to the saturation level . Fig.(7b) shows the RTL hardware schematic circuit for implementing satlins neuron.

(a)



(b)

Fig .7 (a) Time diagram , (b) hardware circuit, of implementing neuron with satlins activation function.

4.3 Hyperbolic Tangent Sigmoid activation function

The Hyperbolic tangent sigmoid (tansing) activation function is shown in Fig (8) . This function takes the input (which may have any value between plus and minus infinity) and the output value into the range - 1 to 1 , according to the expression [11]:

$$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}} \qquad \text{................ ..... (4)}$$

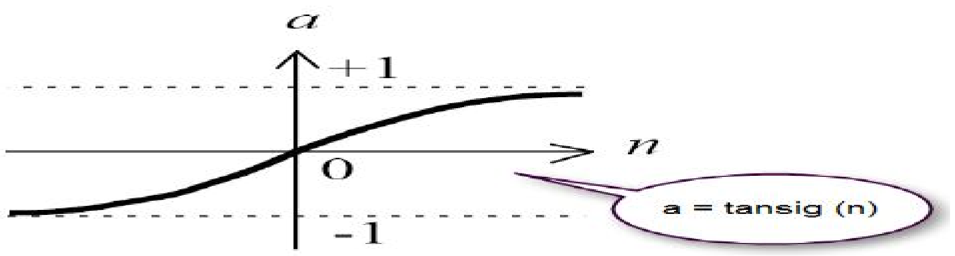


Fig.(8) Hyperbolic tangent sigmoid (tansig) activation function.

The tansig activation function is commonly used in multilayer neural networks that are trained by the backpropagation algorithm , since this function is differentiable [11] . The tansig function is not easily implemented in digital hardware because it is consists of an infinite exponential series [2] .Many researchers use a lookup table to implement the tansig function. The draw back of using lookup table is the great amount of hardware resources needed [1,5]. A simple second order nonlinear function presented by kwan [12] , can be used as an approximation to a sigmoid function . This nonlinear function can be implemented directly using digital techniques . The following equation is a second order nonlinear function which has a tansig transition between the upper and lower saturation regions:

$$f(n) = \begin{cases} n(B - g.n) & for \;\; 0 \le n \le L \\ n(B + g.n) & for \; -L \le n \prec L \end{cases} \qquad \text{……………………(4)}$$

where B and g represent the slope . and the gain of the nonlinear function $f(n)$ between the saturation regions -Ll and L. The block diagram of the sigmoid activation function implementation using this process is shown in Fig.(9).

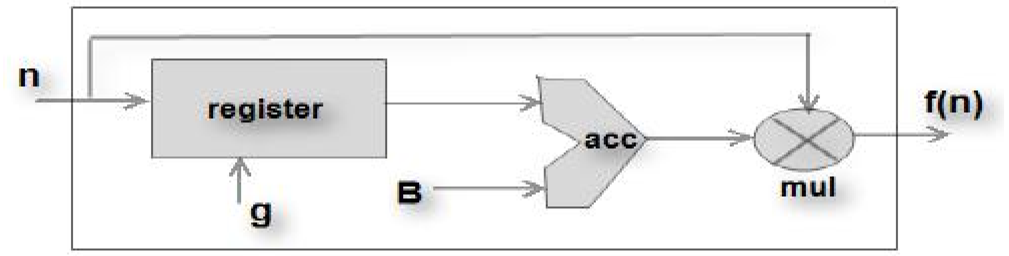
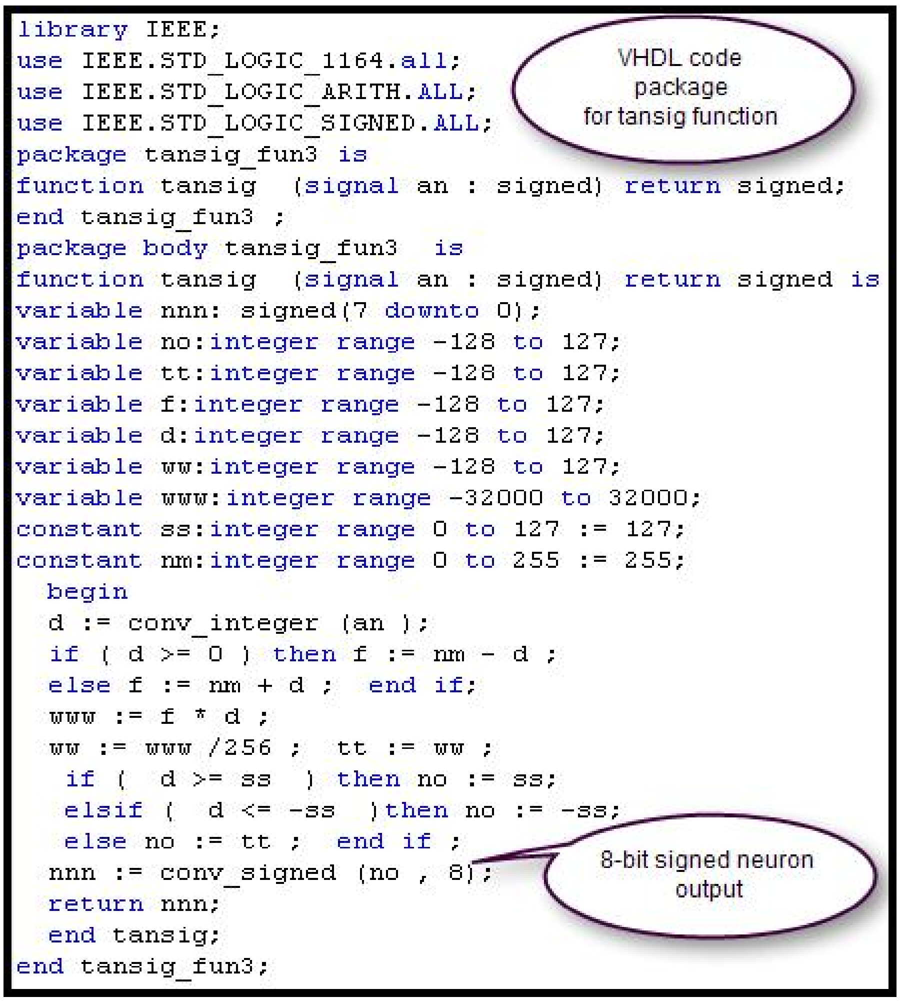


Fig. (9) block diagram of the tansig activation function implementation.

The VHDL code for an approximated tansig function as a package is given in Table (4). The input parameters have been set for an integer range from 0 to 255.

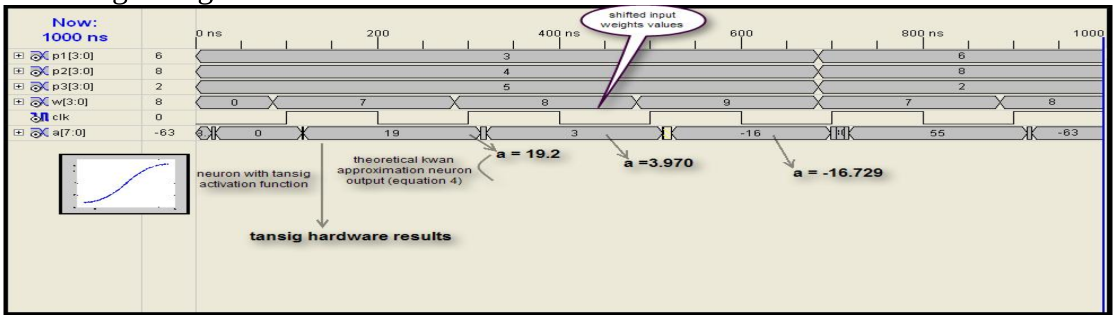
Table(4) VHDL code for implementation tansig activation function as a package.



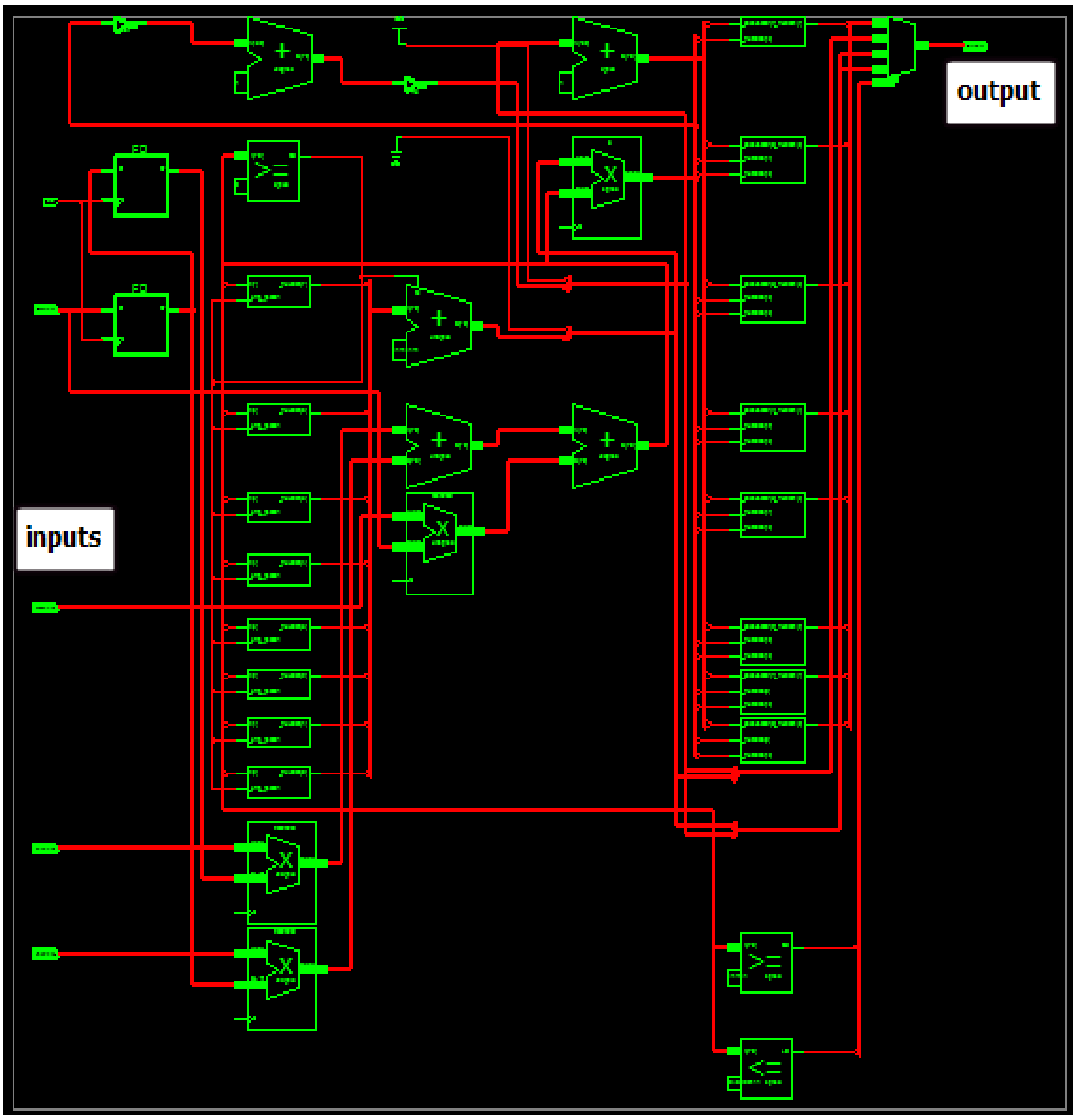
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
package tansig_fun3 is
function tansig  (signal an : signed) return signed;
end tansig_fun3 ;
package body tansig_fun3  is
function tansig  (signal an : signed) return signed is
variable nnn: signed(7 downto 0);
variable no:integer range -128 to 127;
variable tt:integer range -128 to 127;
variable f:integer range -128 to 127;
variable d:integer range -128 to 127;
variable ww:integer range -128 to 127;
variable www:integer range -32000 to 32000;
constant ss:integer range 0 to 127 := 127;
constant nm:integer range 0 to 255 := 255;
  begin
  d := conv_integer (an );
  if ( d >= 0 ) then f := nm - d ;
  else f := nm + d ;  end if;
  www := f * d ;
  ww := www /256 ;  tt := ww ;
   if (  d >= ss  ) then no := ss;
   elsif (  d <= -ss  )then no := -ss;
   else no := tt ;  end if ;
  nnn := conv_signed (no , 8);
  return nnn;
  end tansig;
end tansig_fun3;
```

VHDL code package for tansig function
8-bit signed neuron output

The resulting simulation timing diagram of the neuron with tansig function is shown in Fig (10 a) . The inputs and weights of the neuron is defined as 4- bit and the output is defined as 8- bit and all of singed data type .The approximated parameter of tansig function are of integer data type. Fig.(10b) shows the RTL hardware schematic circuit for implementing tansig neuron.


Now:
1000 ns
p1[3:0] 6
p2[3:0] 8
p3[3:0] 2
w[3:0] 8
clk 0
a[7:0] -63
shifted input weights values
neuron with tansig activation function
theoretical kwan approximation neuron output (equation 4)
a = 19.2
a =3.970
a = -16.729
tansig hardware results

(a)



(b)

Fig .10 (a) Time diagram ,(b) hardware circuit, of implementing neuron with tansig activation function.

In order to make easier visualization of the result of hardware Implementation of tansig function , a comparison between tansing defined by kwan (equation (4)) and the Implemented result is presented in Fig.(11) . This clearly shows that the hardware realization of the tansig function provide a reasonable approximation of this function .

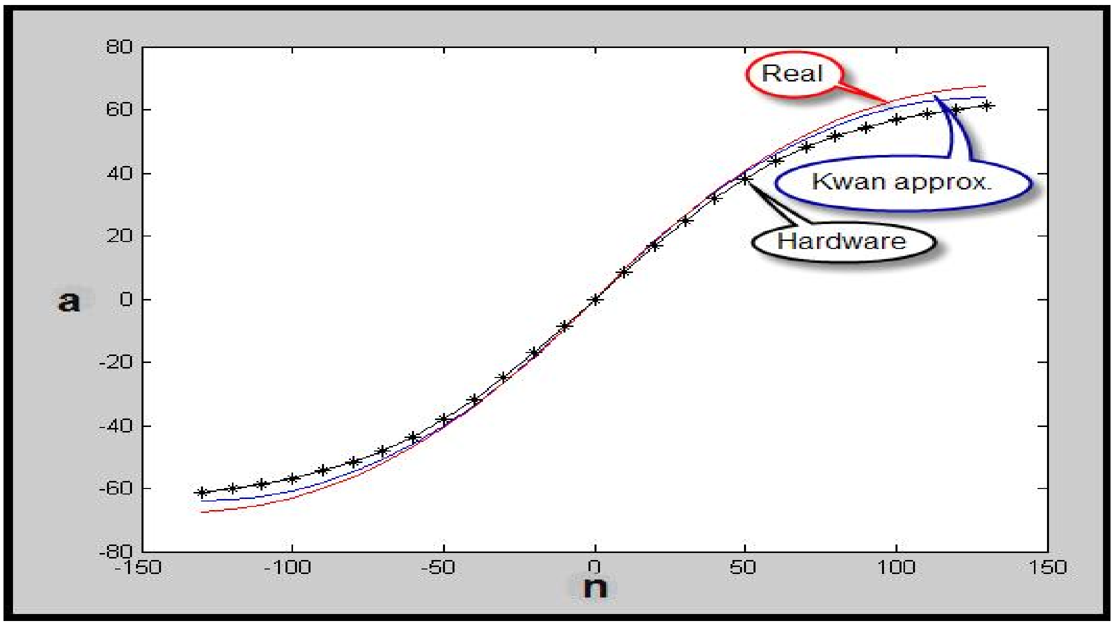
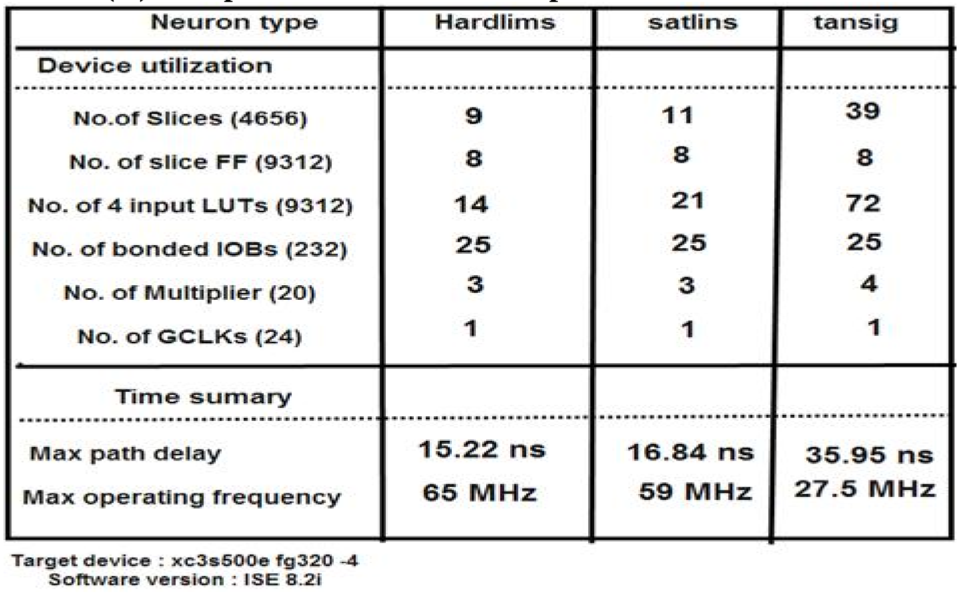


Fig. (11) Tansig activation function hardware results.

5. Synthesis and Implementation Results

As a result of Synthesis and Implementation of artificial neuron with three different activation function (hardlims , satlins , tansig) on a Xilinx XC3S5OOE FPGA device . Table (5) give performance and resource use summary for all implemented 8- bit neurons . As it can be seen , the hardlims function require a very few hardware resource in comparison with the tansig function. The operation speed in all cases gives a good results and shows the advantages of using FPGAs in neural realization.

Table (5) Comparative data for the implemented artificial neurons



Neuron type	Hardlims	satlins	tansig
Device utilization			
No.of Slices (4656)	9	11	39
No. of slice FF (9312)	8	8	8
No. of 4 input LUTs (9312)	14	21	72
No. of bonded IOBs (232)	25	25	25
No. of Multiplier (20)	3	3	4
No. of GCLKs (24)	1	1	1
Time sumary			
Max path delay	15.22 ns	16.84 ns	35.95 ns
Max operating frequency	65 MHz	59 MHz	27.5 MHz

Target device : xc3s500e fg320 -4
Software version : ISE 8.2i

6. Conclusions

The results of this work successfully demonstrate the hardware implementation of artificial neuron with three different activation functions using Xilinx FPGAS . This allows comparisons to be made between the hardware realization of these neurons , which are regarded as basic building block of artificial neural networks .

The tansig function approximation problem has been employed and demonstrate the validity of hardware implementation. The timing diagrams show the accuracy of the results and to enable comparisons with actual result . The operation frequency in all cases is very good and it gives a clear idea of the advantages of using FPGAs, since multiple modules can be working in parallel with a minimum reduction in performance due to the increased number of interconnections.

Finally , it can be say that FPGAs technology and their low cost , and reprogrammability make this approach a very powerful option for implementing ANNS as an alternative to the development of custom hardware.

References

1. Parasovic A., latinovic I . , "A Neural Network FPGA Implementation" . IEEE. 5th seminar on Neural Network Application in Electrical Engineering , September 2000 , PP117 – 120.
2. R. Omondi, C. Rajapakse," FPGA Implementation of Neural Networks", Springer U.S., 2006, ISBN 10-0-387-28485-0.
3. Xilinx , XST User Guide , Xilinx Inc . 2003.
4. S. Brown , J . Rose , "FPGA and CPDL Architectures , A Tutorial ", IEEE Design and Test of Computer , No. 4 , June 1996 , pp.42 – 57.
5. Pavlitov K., Mancler O., " FPGA Implementation of Artificial Neurons" , Electronics, No.9 September , 2004 , pp . 22-24 .
6. J. Blake , L. McDaid , " Using Xilinx FPGAs to Implement Neural Networks and Fuzzy Systems" , Faculty of Eng . , University . of Ulster , Magel college , Northland Rd . Derry , 2005.
7. O. Maischberger ,V. Salapura , " A Fast FPGA Implementation of a General Purpose Neuron ", Technical University , Institute of in formatik , Austria , 2006.
8. Sarich , Moussa , " The Impact of Arithmetic Representation on Implementing MLP – BP on FPGAs : A study " , IEEE Transaction on Neural Network , Vol18 , No.1 , JANNARY 2007 . PP240 – 255.
9. Xilinx , Spartan – 3E starter Kit Board user guide.
10. D. L. Berry, "VHDL programming by examples", McGraw-Hill, fourth edition, 2002.
11. M.Hagan , H . Demuth , M. Beele , " Neural Network Design" , University of Colorado Bookstore, 2002, ISBN : 0- 9717321- 0-8.
12. Kwan , H.K. , " simple sigmoid . like activation function suitable for digital hardware implementation" , Electronic Letters , V.28 , July , 1992 , pp. 1379 – 1380 .

The work was carried out at the college of Engg. University of Mosul