Implementation of A Neuron Model Using FPGA

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 together 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 an 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].
Configuration of The NN system
2. Mathematical model of an artificial neuron

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

\[ a = f \left( \sum_{j=1}^{R} w_j p_j \right) \]  

(1)

where \( p_j \) is the input value and \( w_j \) is the corresponding weight value, \( a \) is the output of the neuron, and \( f(\cdot) \) 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.
LUTs

- FPGAs

\[ y = f(x), \quad x = \{x_1, x_2, \ldots, x_N\}, \quad N_x \leq 4 \]

\[
\begin{align*}
5 \quad \text{slice} & \quad \rightarrow \quad N_x \leq 5 \\
6 \quad \text{CLB} & \quad \rightarrow \quad N_x \leq 6
\end{align*}
\]
Boolean NN

- **Boolean Neuron**

  \[ y = f(x, w) \]

  \[ y_B = f_B(x_B, w_B) \]

  \( x_B = \{x_1, x_2, ..., x_{N_x} \} \quad x_i \in \{0, 1\} \)

  \( w_B = \{w_1, w_2, ..., w_{N_x} \} \quad w_i \in \{0, 1\} \)

  \( f_B \) - Boolean transfer function

  \( y_B \) - output signal

  \[ f_B, y_B \in \{0, 1\} \]

  General structure of Boolean neuron

Advantages of the BN:
- speeding up of calculation significantly,
- reduction of necessary memory size,
- possibility to map the BN into one single CLB of FPGAs.
Boolean NN

- **Structure**
  - LUT: $N_{k_{zn^2}}$, $N_{N_{Ny^2}} \leq 4$
  - Slice: $N_{k_{zn^2}}$, $N_{N_{Ny^2}} \leq 5$
  - CLB: $N_{k_{zn^2}}$, $N_{N_{Ny^2}} \leq 6$

- **LUT of CLB**
  - $y = f_B(x_B, w_B)$
  - Training algorithm
Example

- Example

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k_1$</th>
<th>$k_2$</th>
<th>$k_3$</th>
<th>$k_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Structure of BNN

```
      k_1
        /|
      /  |
    k_2 /   |
       /    |
    k_3 /      |
       /        |
    k_4 /          |
      /            |
    y_0
```

```
  x_1
   |
  --

  x_2
   |
  --

  x_3
   |
  --
```
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 affects the utilization ratio of the chip's 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.
Neuron Structural Diagram

Fig. (2) VHDL structural diagram for neuron implementation.
The VHDL code used for the implementation of a neuron without activation function presented in table (1).

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

```vhdl
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');
        for j in 1 to r loop
          prod := input(j) * weight(j); acc := acc + prod;
        end loop;
        a <= acc;
      end process;
  end Behavioral;
```
4.3 Hyperbolic Tangent Sigmoid activation function

The Hyperbolic tangent sigmoid (tansig) 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]:

\[
a = \frac{e^n - e^{-n}}{e^n + e^{-n}}
\]

\[\text{.............. .... (4)}\]

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 consists of an infinite exponential series [2]. Many researchers use a lookup table to implement the tansig function. The drawback 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
Implementation of A Neuron Model Using FPGA

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:

\[
f(n) = \begin{cases} 
  n(B - gn) & \text{for } 0 \leq n \leq L \\
  n(B + gn) & \text{for } -L \leq n < L 
\end{cases}
\]  

..........................(4)

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

![Block diagram of the tansig activation function implementation.](image)

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.
VHDL Code of Tanh

Table(4) VHDL code for implementation tansig activation 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 tansig_fun3 is
  function tansig (signal an : signed) return signed is
    variable nnn: signed(? 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;
```
Results

<table>
<thead>
<tr>
<th>Neuron type</th>
<th>Hardlims</th>
<th>satlims</th>
<th>tansig</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Device utilization</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of Slices (4656)</td>
<td>9</td>
<td>11</td>
<td>39</td>
</tr>
<tr>
<td>No. of slice FF (9312)</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>No. of 4 input LUTs (9312)</td>
<td>14</td>
<td>21</td>
<td>72</td>
</tr>
<tr>
<td>No. of bonded IOBs (232)</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>No. of Multiplier (20)</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>No. of GCLKs (24)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Time summary</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max path delay</td>
<td>15.22 ns</td>
<td>16.84 ns</td>
<td>35.95 ns</td>
</tr>
<tr>
<td>Max operating frequency</td>
<td>65 MHz</td>
<td>59 MHz</td>
<td>27.5 MHz</td>
</tr>
</tbody>
</table>

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