VLSI Implementation of Neural Network

Jitesh R. Shinde
Research Scholar & IEEE member, Nagpur, India Email:-shindejitesh@gmail.com

Suresh Salankar
Professor, G.H.Raisoni College of Engineering, Nagpur, Maharashtra, India

Abstract — This paper proposes a novel approach for an optimal multi-objective optimization for VLSI implementation of Artificial Neural Network (ANN) which is area-power-speed efficient and has high degree of accuracy and dynamic range.

A VLSI implementation of feed forward neural network in floating point arithmetic IEEE-754 single precision 32 bit format is presented that makes the use of digital weights and digital multiplier based on bit serial architecture.

Simulation results with 45 nm & 90 nm tech file on Synopsis Design Vision Tool, Aldec’s Active HDL tool, Altera’s Quartus tool & MATLAB showed that the bit serial architecture (TYPE III) based multiplier implementation and use of floating point arithmetic (IEEE -754 Single Precision format) in ANN realization may provide a good multi-objective solution for VLSI implementation of ANN.

Keyword — Artificial Neural Network (ANN), bit serial architecture (type III) based multiplier, array multiplier, floating point arithmetic, multi-layered artificial neural network (MNN), Neural Network (NN), multi-layer perceptron (MLP).

2. ARTIFICIAL NEURAL NETWORK

ANN is an information-processing system wherein neurons process information [3].

An artificial neuron forms the basic unit of artificial neural networks. The basic elements of an artificial neuron are (1) a set of input nodes, indexed by, say, 1, 2, ... N, that receives the corresponding input signal or pattern vector, say \(x=(P_1, P_2, ... , P_N)^T\); (2) a set of synaptic connections whose strengths are represented by a set of weights, here denoted by \(w=(w_1, w_2, ..., w_l)^T\); and (3) an activation function ‘a’ that relates the total synaptic input to the output (activation) of the neuron.

The main components of an artificial neuron are illustrated in Figure 2.1.

The total synaptic input, \(a\), to the neuron is given by the inner product of the input and weight vectors:

\[
 a = \sum_{j=1}^{N} W_j P_j ;
\]

where we assume that the threshold of the activation is incorporated in the weight vector. The output activation, \(y\), is given by

\[
 y = f(a);
\]

Fig.2.1: Structural diagram of simple neuron

The multi-layer perceptron (MLP) or muti-layer artificial neural network (MNN) is a feed forward neural network consisting of an input layer of nodes, followed by two or more layers of perceptrons, the last of which is the output layer. The layers between the input layer and output layer are referred to as hidden layers. MLPs have been applied successfully to many complex real-world problems consisting of non-linear decision boundaries. Three-layer MLPs have been sufficient for most of these applications [3, 4] and its block diagram representation is shown in figure 2.2.
3. DESIGN ISSUES

1. Data Representation: An important issue during hardware implementation of neural network is to strike a balance between the need for reasonable precision and cost associated with in logic overhead with increased precision [5]. So selection of proper arithmetic scheme viz. fixed point arithmetic scheme or floating point arithmetic is important.

- The fixed point arithmetic scheme will be advantageous in application where degree of precision is not important and thus in such application it may provide a good multi-objective solution for optimizing cost of hardware and speed simultaneously. But fixed point arithmetic scheme does not provide a better option for dynamic weight re-configurability because as input changes, data bus size and hence the entire logic need to be reconfigured every time as the number of bits used to represent weights are varied in order to enhance the precision and accuracy of system. Moreover, weights need to be processed first (i.e. truncated and rounded off) before applying to system and hence thereby increasing the simulation time.

- The floating point arithmetic scheme (IEEE 754: Single precision format (32 bit) or Double precision format (64 bit)) offers the greatest amount of dynamic range and eliminates the need of processing the weights, and thereby making it good choice for neural network based applications where high degree of precision is desired. But the hardware implementation for floating point arithmetic is costlier and the speed of processing is low due to double calculations i.e., separate calculation for mantissa and exponent.

2. Analog versus Digital Neural network

- The computational density of chip is defined as amount of computation per unit silicon. The computational density of analog neural network is 10 to 100 times greater than that of digital neural network because complex non-linear operations such as multiply, divide and tangent can be implemented with handful of transistors in a analog network in comparison to digital network which requires hundreds of transistor or even thousands of transistor to perform the same operation. Thus, precision of analog neural network is directly proportional to area of the chip.

- The speed of analog neural network is inversely proportional to area of chip. Smaller the area of chip less will be the time taken by signal to propagate to output or from one circuit component to other. Moreover, the parasitic capacitance will also be less and thereby further enhancing the speed of the circuit.

- The power consumption of analog circuit is directly proportional to the speed at which circuit operates. A large percentage of power consumed is dissipated as heat during normal operation of the circuit. For given speed and circuit architecture, efficiency of power dissipation decreases with the area of chip but this will be accompanied by degradation in the precision of circuit.

- One common approach to reduce power requirement of analog circuit is to reduce operating voltages. However the immediate effect will be reduce dynamic range of all signals in circuits and hence affecting the precision of circuit.

- Digital neural network are inherently robust for effects such as substrate noise, power supply variation, radiation, matching, noise, drift, mobility reduction and so on. In analog networks, these effects can be minimized but at the cost of increased power consumption and area of circuit.

- An analog network must be full custom design. Digital Designs are flexible since it allows software control and arbitrary level of precision (low to high, and fixed or floating point). They can be generated from logic description of its function.

- Loading of digital weights is easy in comparison to loading of analog weights since no feedback is required.

- Analog neural networks are harder to scale with new processes and require total redesign. Digital neural networks scale easily with new processes and require no redesign.

Above conclusions drawn from (6, 7, 8, 9) suggest that analog neural networks are suitable for classification problems where minimum power consumption is main design constraint i.e. single objective optimization goal and digital neural networks may be suitable for classification problems where design constraints are precision, area, power and speed i.e. multi-objective optimization.

3. Multiplier Unit

Neural network processing comprises a large number of multiplications operations (equation 2.1). Thus, the performance of digital neural network on large extent depends on how the multipliers are realized in digital neural network. A key design issue in efficient realization of multiplier block will be trade-off between precision,
area, speed and power consumption of the circuit [1, 2, 10, and 11]. MCM (Multiple Constant Multiplication) approach may not be suitable in digital neural network because here requirement is that weight needs to be fixed. So, dynamic weight adjustment will not be possible because it may require redesigning entire block [1, 2].

4. DESIGN APPROACH FOR OPTIMIZATION
To realize multi-objective optimized i.e. high degree of precision, area-speed-power optimized VLSI implementation for ANN following approaches were selected:
- Floating point arithmetic scheme (IEEE 754-single precision (32 bit) format to have high degree of precision.
- Digital neural network may enable in realizing the multi-objective optimization goal.
- Array multiplier or multiplier unit based on bit serial architecture or digit serial architecture (Type 3) (14, 15). Block schematic of multiplier unit based on bit serial architecture and digit serial architecture are shown in figure 4.1, 4.2& 4.3 respectively.

Bit - serial arithmetic and communication is efficient for computational processes, allowing good communication within and between VLSI chips and tightly pipelined arithmetic structures. It is ideal for neural networks as it minimizes the interconnect requirement by eliminating multi - wire busses [16].

Comparative analysis of multiplier (N*N) with respect to multiplicand data size ‘A’ & multiplier data size of N=8 in both cases are shown in table 4.1.

![Fig.4.1: An example of 4x4 Array multiplier](image)

![Fig.4.2: Bit-serial type-III multiplier with word-length of 4 bits](image)

![Fig.4.3: Digit Cell for type-III multiplier](image)

Table (4.1): Comparison of Multipliers

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type of Multipliers 8 x 8 bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MUL1 (Array)</td>
</tr>
<tr>
<td>G1</td>
<td>N^2=64</td>
</tr>
<tr>
<td>G2</td>
<td>N(N-1)=12</td>
</tr>
<tr>
<td>Pipelining</td>
<td>Absent</td>
</tr>
<tr>
<td>Speed</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Area</td>
<td>High</td>
</tr>
<tr>
<td>Dynamic Power Dissipation</td>
<td>Moderate</td>
</tr>
</tbody>
</table>

Where description of notations G1, G2 used in above table is as follows:
- G1 => approximate number of AND gates required for partial product implementation.
- G2 => approximate number of Full Adders required.
- Digit size D=N/W=4. No. of folding W=2.

Comparative analysis suggests that bit serial architecture (Type III) provides better trade off to realize multi-objective optimization approach for VLSI Implementation of digital neural network.

i. Criteria for FPGA selection & high performance:
- Any FPGA is suited to bit serial design. The first consideration is whether there are sufficient logic elements and I/O resources to support the design. Most bit serial designs have a very low routing complexity, so the routing resources are not an issue.
- The entire design should be synchronous, including the sets and resets.
V. DESIGN & IMPLEMENTATION

In our work, we had designed and implemented VLSI
digital ANN viz. ANN using array multiplier approach
(fnn1test1) & ANN using multiplier approach based on
bit serial architecture (Type III) (bsfn1_test1).
The experimental setup flow used in this research work is
as follows:

Step 1: IMPLEMENTATION IN MATLAB: The data
that used in this project was acquired from University of
California, Irvine (UCI Machine Learning Repository).
The first objective of this study was to determine whether
an Iris flower is of versicolour, virginica or setosa. And
also to identify the fit training settings for building
classification model for Iris. The attributes information is
given in Table 5.1.

Table 3.1: Description of attributes

<table>
<thead>
<tr>
<th>No</th>
<th>Physical Attribute</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sepal length</td>
<td>Used as Input</td>
</tr>
<tr>
<td>2</td>
<td>sepal width</td>
<td>Used as Input</td>
</tr>
<tr>
<td>3</td>
<td>Petal length</td>
<td>Used as Input</td>
</tr>
<tr>
<td>4</td>
<td>Petal width</td>
<td>Used as Input</td>
</tr>
<tr>
<td>5</td>
<td>Target</td>
<td>Used as Output:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Setosa</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Versicolour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Virginica</td>
</tr>
</tbody>
</table>

After observing data set it was observed that only two
attributes viz. petal length and petal width were sufficient
to classify whether an Iris flower is of versicolour,
virginica or setosa. Thus, by reducing dimension of input
data set we were able to optimize the simulation runtime
considerably in MATLAB.
To ensure a correct comparison of different types of
neural networks, the division of input data into training,
validation and test sets is performed by independent part
of code and the division result is stored. The partitioning
of input data is performed randomly with a certain ratio
of input entities to be stored as training set, validation
set and test set. The experimental settings for this done in
MATLAB to classify data using neural toolbox is shown
in figure 4.1 and 4.2 respectively and result i.e. trained
graphs obtained showing 100 % fit is shown in figure
5.1, 5.2 and 5.3 respectively.

Train graph of iris data indicates type of iris. The plot
describes the attribute and classes with target and
distribution the iris data. Basically the training purpose is
to identify the fit training settings for model.
STEP II : IMPLEMENTATION in VHDL: The weights of stage 1 (fw11, fw22) and stage 2 (fw33) (refer fig 3.2) obtained from this trained graph were first converted into IEEE 754 single precision 32 bit binary format in MATLAB and were used in stage 1 and stage 2 of VLSI implementation of neural network shown in figure (5.4).

ANN in this research work uses three layer feed forward neural network architecture which has two input neuron, fifteen hidden neuron and one output neuron. Inputs to the ANN are a 32 bit number in IEEE-754 single precision format, represent attributes petal length and petal width of iris flower and output is 32 bit number in IEEE-754 single precision format, represent a prediction to which category flower belongs.

The flow the input data in generalized block schematic structure of IEEE 754 single precision multiplier block used is shown in figure (5.5).

Unpack block unpacks incoming data (31 down to 0) into three parts viz. sign bit (MSB 31st bit), exponent (30 down to 23) and mantissa (22 down to 0). This blocks maps 23 bit mantissa into 32 bit by appending zeros at LSBs.

Packlp block packs final result of multiplication obtained after normalization & rounding i.e. its mantissa, exponent and sign bit into IEEE -754 single precision formats.

Unpacklp and packlp block also checks the exponent and mantissa part of inputs for the following conditions (underflow (de-normalized number), overflow, infinity and not a number (NaN) to find the 32 bit input data and hence output (via FPpack block & logic block to predict nature of output) is a valid IEEE 754 single precision floating point number [12, 13].

Table 5.1: Table listing conditions to check nature of input

<table>
<thead>
<tr>
<th>Number</th>
<th>sign</th>
<th>exponent</th>
<th>mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>normalized number</td>
<td>0/1</td>
<td>01 to FE</td>
<td>any value</td>
</tr>
<tr>
<td>De-normalized number</td>
<td>0/1</td>
<td>00</td>
<td>any value</td>
</tr>
<tr>
<td>number (underflow)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zero</td>
<td>0/1</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>infinity (Overflow)</td>
<td>0/1</td>
<td>FF</td>
<td>0</td>
</tr>
<tr>
<td>NaN i.e Not a Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(inf*0 or inf/inf or 0/0 form)</td>
<td>FF</td>
<td>any value but not 0</td>
<td></td>
</tr>
</tbody>
</table>
The IEEE 754 standard partially solves the problem of underflow by using de-normalized representations in which a de-normalized representation is characterized by an exponent code being all 0's, interpreted as having the whole part of the significand being an implied 0 instead of an implied 1[12].

Fig.5.6: Flowchart representing logic used in program to check for underflow condition

Multiplier block performs the multiplication of two input data (mantissa) coming from unpackFP0 & unpackFP1 block and generates mantissa part of final output. Output data bus of multiplier blocks implemented with array multiplier logic 32bitx32bit (figure 4.1) was of 64 bit and total partial product inferred were 64. Output data bus of multiplier block implemented with bit serial logic (figure 4.2) was 24 bit and number of AND gates inferred to implement partial product were 24. Exponent adder blocks add exponent parts of respective exponent parts of input A and B to generate exponent of output Z.

The RTL view of ANN using array multiplier and ANN using bit serial architecture (Type III) based multipliers are shown in figure 5.7 & 5.8 respectively. Each block description is as follows:-

1. **bigfp_fps block** :- The function of this block is to find the largest and smallest number as shown in figure 5.9. The major entities in this block diagram are bigfp_fps, absdiffl block, decsel1 block, barrel_shift_R block, intadd23 block, readadj_mj block and expadder1 block respectively. Each block description is as follows:-

Floating-point numbers are coded as "sign/magnitude", reversing the sign-bit inverses the sign. Consequently the same operator performs as well addition or subtraction according to the two operand's signs.

Floating-point addition progresses in 4 steps:
- Mantissa alignment if A and B exponents are different,
- Addition/subtraction of the aligned mantissas,
- Renormalization of the mantissas sum S if not already normalized
- Rounding of sum S'.

The floating point adder block diagram used is shown in figure 5.9. The major entities in this block diagram are bigfp_fps, absdiffl block, decsel1 block, barrel_shift_R block, intadd23 block, readadj_mj block and expadder1 block respectively. Each block description is as follows:-
Let given numbers are A=0101 and B=1000 where first bit i.e. A (3) & B (3) represents sign bit of given number.

bigtemp <= FP_A when (not(FP_A(3))&FP_A(2 downto 0)) > (not(FP_B(3))&FP_B(3 downto 0)) else FP_B;

smalltemp <= FP_A when (not(FP_A(3))&FP_A(2 downto 0)) < (not(FP_B(3))&FP_B(2 downto 0)) else FP_B;

big_op    <= smalltemp when (FP_A(3) and FP_B(3)) > '0' else bigtemp ;

small_op   <=  bigtemp      when (FP_A(3) and FP_B(3)) > '0'  else smalltemp ;

The result of program will be big_op <=A & small_op <=B.

2. **absdiff1 block**: This block find the larger of the two exponent and outputs the absolute value of the exponent difference as shown in figure 5.11.

3. **decsel1 block**: This block outputs the information how many bit smaller mantissa should be shifted depending on input coming from output of absdiff1 block. The "implicit bit" is added, totaling 24 bits.

4. **barrel_shift_R block**: This block shifts the smaller operand mantissa (small_op (23 down to 0)) towards right by number coming from output of decsel1 block.

5. **intadd23 block**: This block computes the addition of two floating point numbers (24 bit) using 2s-complement arithmetic and result obtained is of 25 bits.(figure 5.11). Here, if addsub= ‘0’, the two numbers are added, else the subtraction is performed using 2s-complement arithmetic.

6. **readi_m1 block**: This block checks 25 bit result i.e. mantissa of intadd23 block and performs following three tasks viz. :
   - It checks each bit of mantissa from MSB part for ‘1’ and outputs that bit position at output i.e. onethloc and thus the exponent is computed from the result of addition block. If for example man1(22)= ‘1’, then
onethloc = “00000001” or man1(0)= ’1’, then onethloc = “00010111” else onethloc = “00011000”.
- It readjusts the mantissa block.
- It outputs addsub =0 if man1(24 and 23 )= '0' else addsub = ‘1’.

7. expadder block: - This blocks performs the addition of exponents viz. big_op number exponent exp1 (7 down to 0) and exponent obtained from readj_m1 block val(7 down to 0). If addsub= ‘0’, the two numbers are added, else the subtraction is performed using 2s-complement arithmetic.

Fig 5.12: Flow graph for logic used in expadder block

VI. RESULTS & COMPARISON
The code for of ANN using array multiplier and ANN using bit serial architecture (Type III) based multiplier were written in Aldec Active HDL tool and synthesized on Altera’s Quartus tool and it was targeted on FPGA Cyclone 2, Device EP2C70F672C6. Later on code was also tested at Backend on Synopsis tool on 45 nm & 90 nm tech file. FPGA implementation of ANN array multiplier & using bit serial architecture (Type III) based multiplier were clearly classifying iris data into 3 categories: - output corresponding to counter’s count value from 1 to 50 represented iris setosa, output corresponding to counter’s count value from 51 to 96 represented iris versicolour and output corresponding to counter’s count value from 97 to 143 represented iris virginica. This was found to be matching with iris flower data set.

The performance comparison of implemented ANN at frontend and backend VLSI design are given in table 6.1 and 6.2 respectively and graphical representation of total cell area and total dynamic power dissipation are shown in figure 6.1 and 6.2 respectively.

The experimental results at Backend VLSI design level indicates that ANN using bit serial architecture Type III based multiplier is better than ANN using array multiplier in:
- Total Cell area saving in 90 nm & in 45 nm: - 62.681 % & 60.65 % respectively.
- Total area saving in 90 nm & in 45 nm: - 63.86 % & 60.65 % respectively.
- Total dynamic power saving in 90 nm & in 45 nm: - 53.81 % & 90.88 % respectively.

Table 6.1: Performance comparison of implemented neural networks at frontend level

<table>
<thead>
<tr>
<th>Parameters</th>
<th>32x32 bit</th>
<th>NN using Array Multiplier</th>
<th>NN using Bit serial architecture based (Type III) Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Logic Elements</td>
<td>63038</td>
<td>62725</td>
<td></td>
</tr>
<tr>
<td>Embedded multipliers</td>
<td>300/300</td>
<td>0/300</td>
<td></td>
</tr>
<tr>
<td>Total dynamic power dissipation(mW)</td>
<td>40.14</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Data arrival time tco(nsec)</td>
<td>300.023</td>
<td>273.496</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Performance comparison of implemented neural networks at backend level

<table>
<thead>
<tr>
<th>Parameters</th>
<th>90 nm tech file</th>
<th>45 nm tech file (no Workload model)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>Bit</td>
<td>Array</td>
</tr>
<tr>
<td>Total Area nm²</td>
<td>2333944</td>
<td>843437</td>
</tr>
<tr>
<td>Total cell area nm²</td>
<td>2193638</td>
<td>818627</td>
</tr>
<tr>
<td>Total dynamic power dissipation(mW)</td>
<td>5.1803</td>
<td>2.393</td>
</tr>
<tr>
<td>Data arrival time tco(nsec)</td>
<td>90.9</td>
<td>1.5</td>
</tr>
</tbody>
</table>

The performance comparison of implemented multiplier at backend VLSI design is given in table 6.3 and 6.4 respectively.

Table 6.3: Performance comparison of implemented multipliers 32 × 32 bit at backend level

<table>
<thead>
<tr>
<th>Parameters</th>
<th>90 nm tech file</th>
<th>45 nm tech file (no Workload model)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>Bit</td>
<td>Array</td>
</tr>
<tr>
<td>Total area nm²</td>
<td>20677.4997</td>
<td>57</td>
</tr>
<tr>
<td>Total cell area nm²</td>
<td>20007.0145</td>
<td>59</td>
</tr>
<tr>
<td>Total dynamic power dissipation(mW)</td>
<td>0.5913478</td>
<td>0.0275544</td>
</tr>
<tr>
<td>Data arrival time tco(nsec)</td>
<td>19.89</td>
<td>4.19</td>
</tr>
</tbody>
</table>

Table 6.4: Performance comparison of implemented multipliers 32 × 32 bit at backend level

Copyright © 2015 CTTS.IN, All right reserved
VII. CONCLUSIONS

The realization of bit serial architecture Type III based multiplier implemented in floating point arithmetic provides a good trade off in realizing high end applications which is area-speed-power efficient with good precision and dynamic range. The bit serial architecture Type III based multiplier approach suggested in this paper were found to be giving better performance than other promising findings available in literature [17, 18, 19, 20, 21, 22, 23, 24, 25]. Hence, the realization of ANN design using bit serial architecture Type III based multiplier implemented in floating point arithmetic (IEEE 754- single precision format) has presented a new approach for multi-objective optimization of VLSI implementation of neural network. It has also proven to be better alternative over ANN design using array multiplier.

Future research work system by improvising adder unit block if possible and a full characterization of each design option at layout level.

REFERENCE


Copyright © 2015 CTTS.IN, All right reserved 523
[20] Chaitali V. Matey, Dr. S.D. Chede, S.M.Sakhare, “
Design & Implementation of Floating Point Multiplier Using Wallace and Dadda Algorithm”,


[23] Prashant Kumar Sahu, Nitin Meena, “Comparative Study of Different Multiplier Architectures”,


AUTHOR’S PROFILE

**Jitesh R. Shinde** received B.E. Degree in Electronics & Telecommunication Engineering from and M. Tech Degree in VLSI from Rashtrasant Tukadoji Maharaj Nagpur University in 2005 & 2007 receptively. He has also done C-DAC diploma in VLSI Design in 2008. He has also done specialization in IC Layout Engineering certified by University of California, Santacruc Extension and Cadence from TIIT, Hyderabad in 2008. He is doing research in VLSI domain from Rastrasat Tukadoji Maharaj Nagpur University, Nagpur. His current research focuses on VLSI design, Digital Signal Processing, Neural Network.

**Suresh S. Salankar.** received B. E Degree in Power Electronics from Bapurao Deshmukh College of Engineering, Sevagram in 1987. M.Tech in System and Control Engineering from Indian Institute of Technology, Bombay in 1994. Ph.D in Electronics and Telecommunication Engineering from Swami Ramanand Teerth Marathwada University, Nanded, India in 2008. He is currently working as Professor in G. H. Raisoni College of Engineering, Nagpur, India. He has 27 years of teaching experience. He is member of Indian Society for Technical Education, Institution of Engineers (India) and Computer Society of India. His research interests are in the design and evaluation of learning algorithms for Pattern Recognition Applications. It includes, in particular, neural network classifiers, Support Vector Machine Classifiers and Classifier combining strategies. He has published several papers in these areas.

Copyright © 2015 CTTS.IN, All right reserved

524