Journal of Engineering and Development, Vol. 14, No. 3, September (2010) **ISSN 1813-7822**

Complete Neural Network on a Single FPGA Chip

Assist. Prof. Dr. Dhafer R. Zaghar
Department of Computer & Software Engineering/College of Engineering/University of Al-Mustansiriyah /Iraq, Email-drz_raw@yahoo.com

Abstract

This paper presents a hardware implementation approach for Neural Networks (NNs) on a Programmable System-On-Chip. This is an intrinsic online evolution system that can be genetically evolved and adapted to change in input data patterns dynamically without any need for multiple Field Programmable Gate Array (FPGA) reconfigurations to accommodate various network structure/parameter changes. This will remove a considerable bottleneck for performance.

The hardware implementation of NN using FPGA has two main problems. First it is required a large cost because it has a large number of multipliers, lock up tables (LUTs) and adders. Second the additional error that generate from the truncation of numbers when each value in software has minimum 64-bit while it has in hardware maximum 16-bit.

This paper discusses combinations methods to reduce the cost and increase the speed of NN and propose a novel approaches to removes a considerable bottleneck and reduce the cost of a NN to plausible range under FPGA hardware.

Key words : Neural networks, System-On-Chip, FPGA, multiplier, carry lookahead adder, ripple adder, activation function, LUT, piecewise.

الخلاصـــــــة

هذا البحث يناقش اساليب البناء المادي (hardware implementation) للشبكات المخيه (Neural Networks) باستخدام اسلوب بناء النظام في قطعه واحده قابله للبرمجه (Programmable System-On-Chip). وهذا الاسلوب يعطي نظام مرن و قابل للتعديل دون الحاجه الى عدة قطع من مصفوفة البوابات الواسعه القابله للبرمجه (FPGA) و هذا بدوره يؤدي الى ازالة معظم العقبات التي تقلل من كفاءه النظام. ان اسلوب البناء المادي للشبكات المخيه باستخدام مصفوفة البوابات الواسعه القابله للبرمجه يملك مشكلتان اساسيتان الاولى انه يتطلب حجم كبير بسبب العدد الكبير من وحدات الضرب و الجدوله و الجمع. اما المشكله الثانيه فهي نسبه الخطأ الكبيره التي تنتج من عمليه تقليص سعه الرقم و التي تتمثل ب 64 خانه على الاقل في حالة البرامجيات ولكنها في حالة البناء المادي لا تتعدى 16 خانه. ان هذا البحث سيناقش طرق مركبه لتقليل الكلفه و زيادة السرعة للشبكات المخيه لغرض ازالة العقبات و تقليص حجمها بحيث تصبح ذات حجم يمكن بناءه في قطعة واحده من مصفوفه البوابات الواسعه القابله للبرمجه.

1- Introduction

An artificial neural network is a network of fully or partially interconnected information processing elements called artificial neurons, primarily used for input data classification/approximation/prediction problems.

These networks are massively parallel with inherent concurrency that can be exploited using custom digital hardware designs to achieve higher performance than equivalent software implementations. The trial-and-error based training algorithms for obtaining an optimal network requires frequent changes to the network structure and parameters such as synaptic weights and biases. Even the popular multilayer perceptron (MLP) models for NNs have no general solution to determine an optimal network [1].

Massive connection between neurons in this fully connected network makes MLP less flexible for implementation using digital hardware such as ASICs (Application-Specific Integrated Circuits) or FPGAs. Every small change in network structure, such as addition of a neuron to a hidden layer, can result in significant changes to routing structure warranting modifications in hardware design, which comes at a significant cost of time and resources. Due to these limitations the training algorithms are traditionally run in software to find optimal network structure and parameters and the network thus obtained is frozen in an ASIC to achieve higher connections per second (CPS) processing speeds. To gain the flexibility of software and the processing speeds of digital hardware many researchers have proposed FPGA implementations of artificial neural networks, relying heavily on multiple FPGA reconfigurations [2]. The overhead of FPGA reconfiguration typically requires on the order of a few milliseconds depending on the specific reconfiguration method used. This delay may eliminate any learning speedup advantage if the goal is to develop an online adaptable network. Also, this approach requires that the updated networks have been pre-designed and stored to reconfigure the FPGAs dynamically.

There has been a strong need for dynamically configurable hardware design which can accommodate variations in network structure without hardware redesign and multiple FPGA reconfigurations. The dynamics of many real world applications require a more flexible network structure that can evolve and re-learn to accommodate changes in the input data patterns. A typical example of such a system could be individualized classifiers. To train a single network that performs well for every user is difficult. The classifier may need to be dynamically re-trained for each different individual. Also, training a neural network with a large global dataset would be very time-consuming and could result in much larger networks, assuming the training converges to a suitable solution.

2- The Cost and Delays of Hardware Neural Network

The cost of the NN is depend on the numbers of multipliers, adders and LUTs multiplied by there costs as in equation (1).

$$Cost\ of\ NN = multipliers\ No.*multiplier\ cost + adders\ No.*adder\ cost + LUTs\ No.*LUT\ cost \quad \text{---(1)}$$

Hence, the numbers of multipliers, adders and LUTs are fixed for a NN and depend on the numbers of nodes and number of connections as in equation (2).

$$Cost\ of\ NN = connections\ No.*multiplier\ cost + nodes\ No.*adder\ cost + nodes\ No.*LUT\ cost \quad \text{--(2)}$$

The number of nodes is the sum of nodes in the hidden layers and output layer (all nodes except the input layer nodes), while the number of connections is the number of weights in the NN. Example1 shows how the number of nodes and number of connections in the NN of Fig (1).

Example 1: a NN has four layers that have K, M, N and O nodes respectively as in Fig (1).

The number of nodes = M+N+O

The number of connections = K*M+ M*N+ N*O.

However, the key to reduce the NN cost is depending on the reduction of the costs of the NN components (multipliers, adders and LUTs).

The delay of NN is depend on the maximum delay in all its elements, therefore the designer must calculate all the delays of the NN then re-design it to reduce the maximum delay and reduce the cost of the small delay components (that will increase the delay, but its value must be remain less than the maximum delay in the final design) to achieve the optimal speed, cost NN.

3- The Cost and Delays of Neural Network Components

This section will discuss in details the calculations of cost and delay of all NN components. The NN components are adders, multipliers and LUTs. These calculations will depend on the term in equation (3) that represents a general node in NN [8].

$$o = f\left(w_0 + \sum_{i=1}^{K} w_i * x_i\right) \quad \text{----------(3)}$$

Where o is the output, f is the activation function, w is the weight, x is the input and K is the number of nods in previous layer.

3.1- The Cost and Delay of adder in NN

The adder part in NN represent the term ($+\sum_{i=1}^{K}$) in equation (3). This part will not implement as a serial adder but it will be implement as a parallel adder or practically it will be implemented as a binary tree of adders as in Fig (2).

If K is the number of inputs each one has n-bit, the tree adders will require to K/2 adders in layer 1 each one has n bit inputs and n+1 bit outputs.

Layer 2 has half number of adders in layer 1 (K/4) but each one has n+1 input and n+2 outputs, so on the final layer has one adder with n+t bit output, where t is the number of layers in the binary tree that calculate from equation (4).

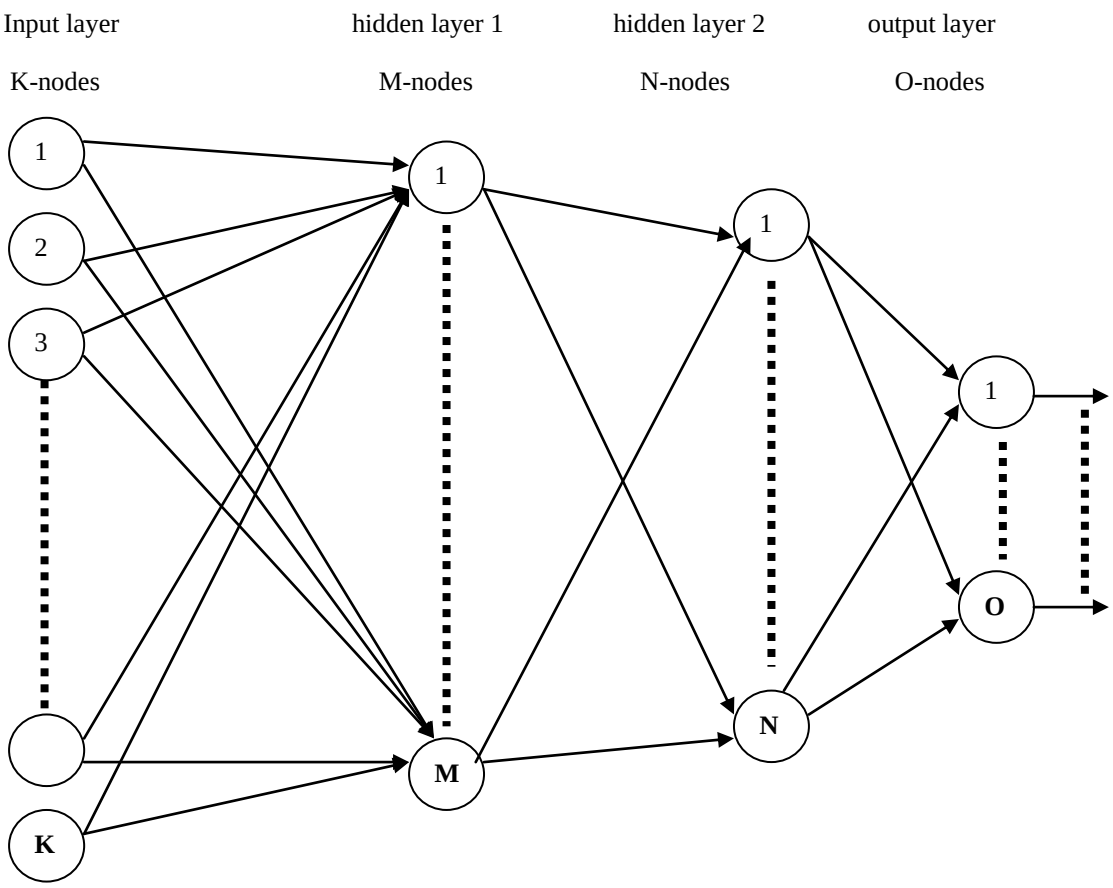


Fig (1): The block diagram of 6x5 inputs TDNN [8].

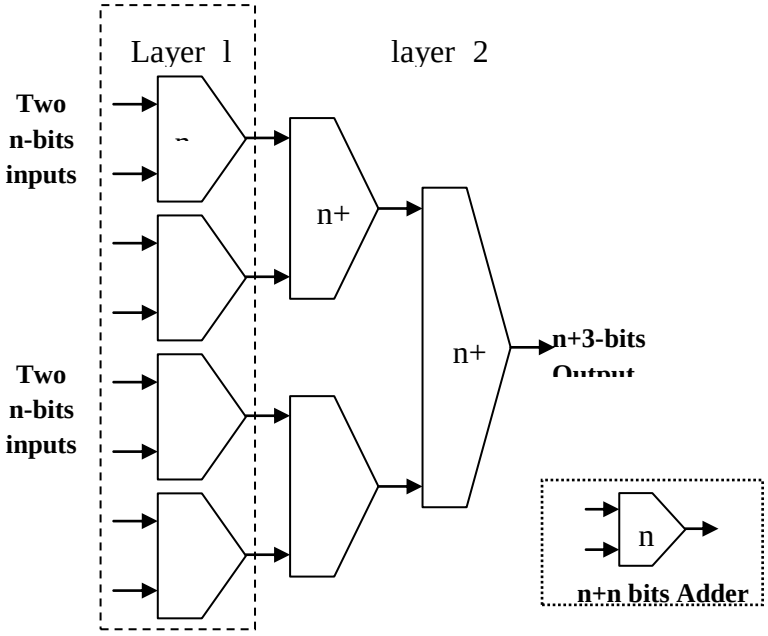


Fig (2): The block diagram of a 4 binary tree adder each one is n-bits.

$$t = \lceil \log_2^{\kappa} \rceil \text{ ----------(4)}$$

Where $\lceil \ \rceil$ is the nearest most integer.

Each n-bit adder in this figure refer to n-bit adder/subtractor as in Fig (3), that represent a three bits adder (adder/subtractor) as example of the adder. Fig (3) shows each 1-bit adder require 1 cell for the sum (si) and 1 cell for the carry (Ci) connected parallel or in result it requires 2 cells as cost with 1 cell as delay. That's mean each n-bit adder require to n cells cost with n cell delay.

$$si = (ai \oplus as) \oplus (bi \oplus bs) \oplus C_{i-1}$$
$$C_i = (ai \oplus as)(bi \oplus bs) + (ai \oplus as)C_{i-1} + (bi \oplus bs)C_{i-1}$$

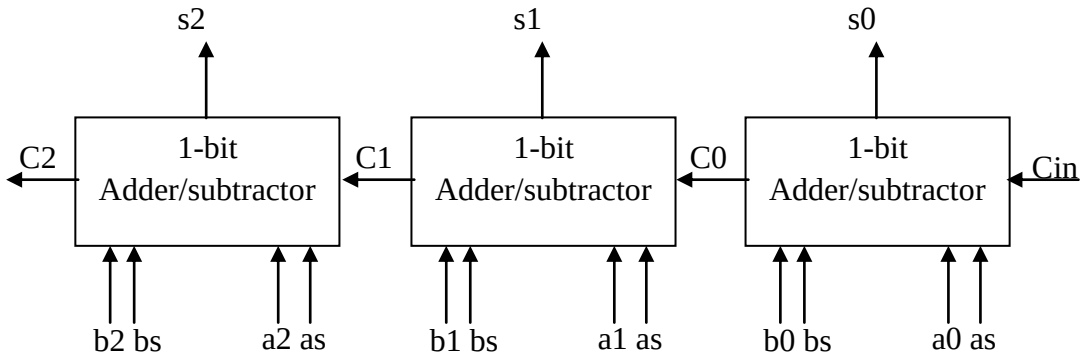


Fig (3): The block diagram of a 3-bit adder.

However, to simplify the calculations of the cost we will propose the number of the two input adders in the binary tree is ***K*** adders with average inputs is ***1.5n-bit*** in each one. While, the calculations of the delay will use the maximum delay that spent in the last adder in the binary adder tree or in simple word it is equal to the delay of adder with ***(n+t)- bits***.

The cost of n-bit in Fig (2) is n full adders, each full adder require two cells one for sum and the other for the carry or in result the cost of n-bit adder is ***2n cells***. While the delay of n-bit in Fig (2) is the delay of n full adders, each full adder spent one unit delay or in result it will spent ***n gate delays***. Summery the cost and delay of adder part in equation (3) are as in equation (5).

$$Adder\ \cos t = K * 1.5n * 2 = 3nK$$
$$Adder\ delay = (n+t) * 1 = \left(n + \left\lceil \log_2^K \right\rceil\right) \qquad \text{----------(5)}$$

3.2- The Cost and Delay of Multiplier in NN

The multiplier part in NN represent the term ($w_i * x_i$) in equation (3). This part implement as in Fig (4).

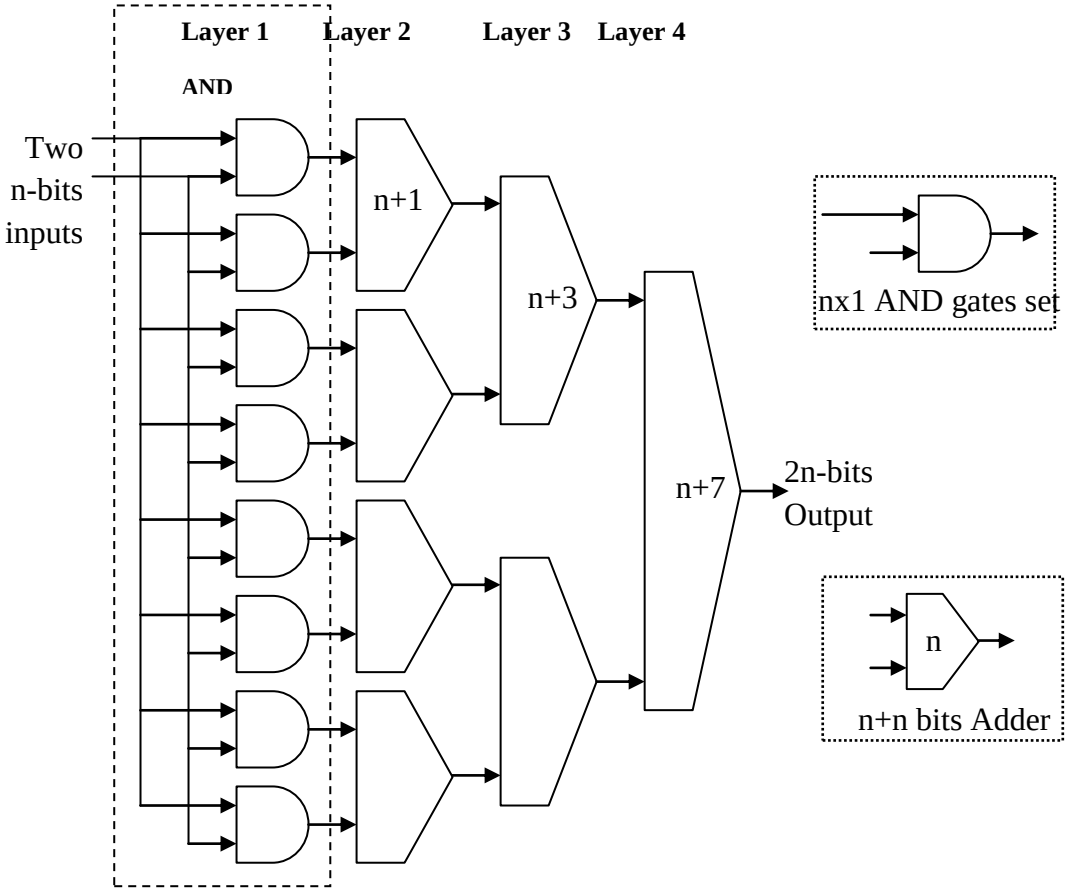


Fig (4): The block diagram of an 8x8-bits multiplier.

Let the inputs of the multiplier are m-bit and n-bit, this multiplier will implement in form of the tree adders as in adder implementation but with two differences first it has additional layer present an n*1-bit multiplier. The second difference between adder and the multiplier is the shift between adders in the tree layers that will represent an additional cost and delay.

The m*n-bits multiplier will require to m*n AND gates in layer 1 each one require 1 cell cost and 1 gate delay. Layer 2 has m/2 adders each one has n+1 input and n+2 outputs, layer 3 has m/4 adders each one has n+3 input and n+4 outputs so on the final layer has one adder with n+m bit output.

However, to simplify the calculations of cost and delay we will use the same rules in the adder cost and delay approximations. These approximations will lead to the m*n multiplier require to about m adders each one has average inputs is ***2n-bit*** with ***m*n gates***.

While, the calculations of the delay will use the maximum delay that spent in the last adder in the binary adder tree or in simple word it is equal to the delay of adder with ***(n+m)-bits***. Summery the cost and delay of n*m multiplier part in equation (3) are as in equation (6).

$$\begin{aligned} Multiplier\ \ \text{cost} &= m * 2n * 2 + m*n = 5nm \\ Multiplier\ \ delay &= (n+m) * 1 = n+m \end{aligned} \qquad \text{----------(6)}$$

3.3- The Cost and Delay of LUT in NN

The LUT part in NN represent the term $f(\ )$ in equation (3). The cost of n-bit inputs and m-bit output LUT is as in equation (7).[6]

$$\begin{aligned} LUT\ \ \text{cost} &= \begin{cases} m & for\ n \le 4 \\ m*2^{n-4} & for\ n > 4 \end{cases} \\ LUT\ \ delay &= 1 \end{aligned} \qquad \text{----------(7)}$$

4- Practical Example for Neural Network

This paper will take the Ten English (Arabic) Digits NN (TDNN) with 6x5 input matrix as in Fig (5) as example to NN.

The TDNN has 30 input nodes, 10 output nodes with two hidden layers first hidden layer has 17 node and second hidden layer has 13 nodes.

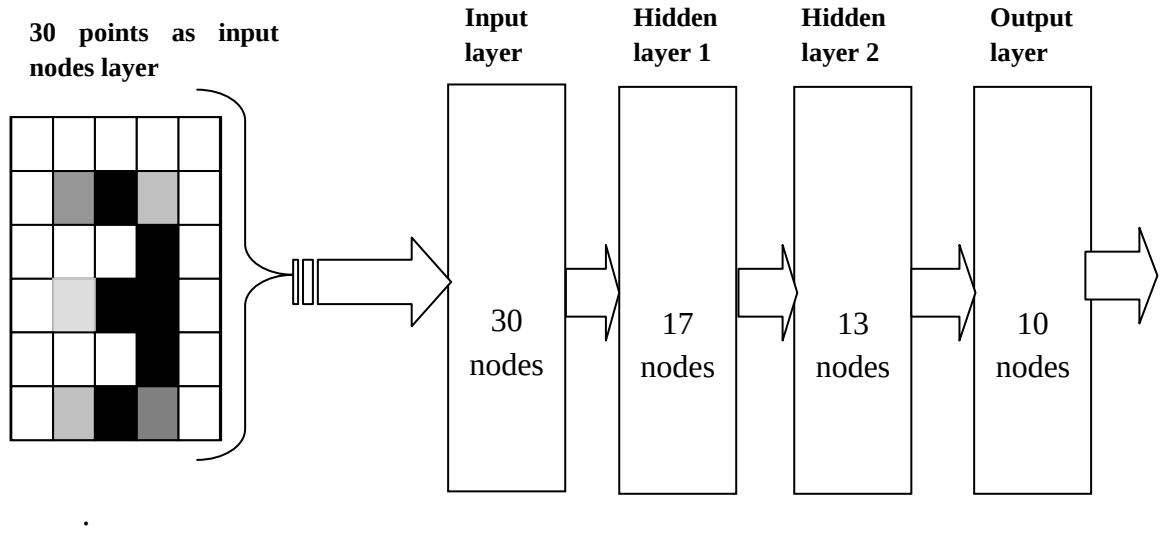


Fig (5): The block diagram of 6x5 inputs TDNN.

The TDNN has 40 nodes and 861 connections, its require as in equation (2) to 510 1xn-bit multiplier between input and hidden 1 layers and 351 n x n-bit multiplier between hidden 1, hidden 2 and output layers, 861 n-bit adders and 40 LUTs. The other factor in the calculations of the TDNN is the data-bus of it. The suitable data-bus for NN is 16-bit that will give a low error.

The approximated costs and delays of TDNN will calculate as in equations (5, 6 and 7) as in table (1). While the real costs and delays of the TDNN components are given by the hardware simulation using ISE 4.1i as in the last two columns in table (1).

Table (1) shows the TDNN required -for 16-bit data-bus hardware implementation- to about 3,200,000 cells with maximum time delay is about 73 nsec while the capacity of Xilinx FPGA as example for XCE5VLX330 (largest capacity chip in Xilinx Virtex-5 Family FPGAs) is 331,776 cells [R1].

This problem solve in the classic methods [4, 5] by using a small general NN (SGNN) and divided the NN to a multi sub-NN each one covers a section from the NN then use the SGNN multi-times to execute the total job of NN.

This method has three disadvantages first its slow because it use the single SGNN serially, second it's require a very complex control circuit with large size of memory and third its need a special partition for each NN.

Table (1): The costs and delays of the TDNN components.

Component	Approximated cost/ cell	Approximated delay/ cell delay	Real cost / cell for XCE5VLX330	real delay/ nsec for XCE5VLX330*
16*16-bit multipliers	1280x351=449280	30	795x351 =279045	73
1*16-bit multipliers	16x510=8160	1	16x510=8160	3
16-bit LUTs	65536x40 =2621440	1	65536x40 =2621440	3
16-bit adders in hidden layer 1	1400x17 =23800	20	965x17 =16405	50
16-bit adders in hidden layer 2	695x13 =9035	19	590x13 =7670	4^8
16-bit adders in output layer	460x10 =4600	19	424x10 =4240	4^6
Total adders	37435	20	28315^+	50
Total system	3117520	30	2936960^+	73^+

*Each cell delay in XCE5VLX330 is about 2.4 nsec.
$^+$ Estimated values.

5- Reduction the Cost of the TDNN

This paper proposes a new approach to re-design the components of the NN to become more suitable with hardware and in result implement a complete NN in a single chip FPGA. First we will calculate the ratios of the costs in the NN components, and then reduce these costs. The cost of TDNN in table (1) divided as 89% for LUTs, 10% for multipliers and 1% for adders.

The reduction of the NN cost start with the highest cost (LUTs), then the next reduces the cost of the multiplier.

The reduction of the multiplier cost must reduce the delay because this component has the maximum delay in the NN. The reduction of the costs and the delays will be satisfied in the three combinational methods.

5.1- Method 1

The main cost of TDNN fails in the area of activation function (LUTs). However, there are a large number of methods to reduce the cost but most these methods depends on the reduction of the data bus and in result it will reduce the resolution of the NN. The proposed method is to implement the low cost NN activation function (Sigmund, tanh….) is use the piecewise approach.

The piecewise approach is depends on the division of the activation function to two parts (functions) as in Fig (6) and in equation (8).

$$f(x) = f(\alpha + \beta) = g1(\alpha) + \beta g2(\alpha) \qquad \text{----------(8)}$$

Where α is the 8 MSBs of x, β is the 8 LSBs of x, g1(α) is the initial value of line (shift), g2(α) is the slope of line.

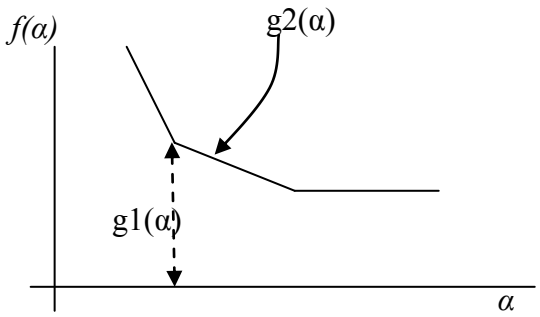


Fig (6): The general form piecewise function.

The block diagram of the LUT using equation (8) is in Fig (7). The re-design of the LUT using this method will result to implement the 16x16 bit LUT using two LUTs. First LUT for shift that use the 8 MSBs inputs with 8-bit outputs that will generate the cores value of the final output. The second LUT use the 8 MSBs inputs (α) with 8-bit outputs that will generate the slope value, it will be multiplied with the 8 LSBs (β) to generate the fine value of the final output. The summation of the cores and the fine values is the final output for LUT.

The cost of each 8x8 LUT (shift or slope) is 128 cells as in equation (7). That is mean the total cost of the 16x16 LUT will become 486 cells. Hence, this approach will require to 71 cell delay. This delay will capture the system; the solution for this problem is the pipeline approach. This approach add an internally memory as in Fig (7). That will increase the cost and reduction the delay. Summary the total new cost of the 16x16 LUT is 510 and the maximum delay is 15 cell delay.

That is mean the cost of all TDNN LUTs will reduced to 20400 cells, ie about 0.78 % from the original cost of LUTs with maximum delay is less than the maximum delay of the system in table (1) (32 cell delay).

The total estimated cost for TDNN using this method is 523625 cells, that represent about 17 % from the original cost of the TDNN, but its a large value in the FPGA techniques.

5.2- Method 2

Method 1 reduced the cost but the total cost is large also because it required to 2 FPGA chips (XCE5VLX330) at least. The second step is reducing the cost of multipliers. This method depend on the re-modification of the learning (weight generation) NN algorithm (back propagation, genetic…) to satisfy two conditions. The first condition is to capture the values of data in 16-bit (normalized with 16-bit) and the values of weights in 8-bits. This is a novel method that will reduce the cost of each multiplier to 640 cells (equation (6)) with maximum delay is 12 cell delay, the cost of the TDNN multipliers in hidden layer 2 and output layer will reduced to 640*351 = 224640 cells that represent about 50% from the original cost of these multipliers.

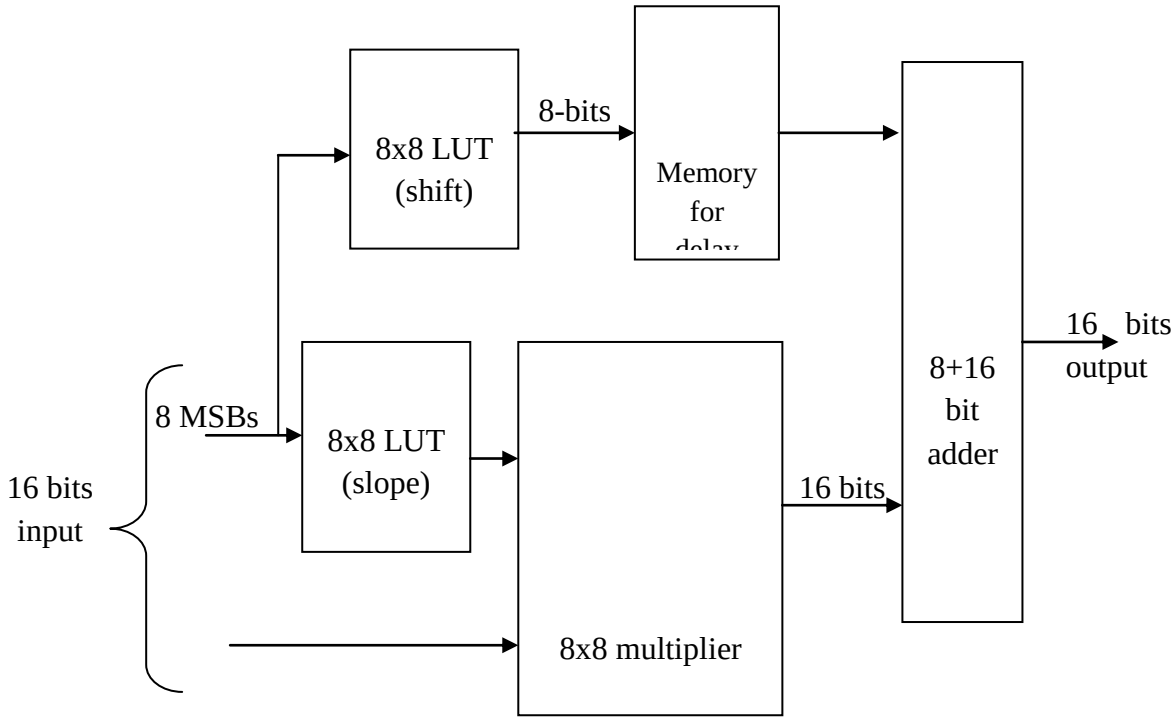


Fig (7): The block diagram of a piecewise implementation of the NN activation function.

The second condition is a special condition in TDNN but its not general for all NN. This condition depends on replacing the values of the input matrix from (0, 1) to (-1, 1). That will enhancement the learning algorithm and in the same time we will replace each 1x16-bit multiplier by 1x1-bit multiplier (sign multiplier). This condition will reduce the cost of the multiplier in hidden layer 1 from 8160 to 1x510= 510 cell.

Hence, the total cost after use method 1 and method 2 is 282985 cells, that is mean the new cost is 9 % from the original cost. These points realize the target and the TDNN can be implementing in a single chip FPGA (XCE5VLX330).

5.3- Method 3

Methods 1&2 reduced the cost but the total cost now is fit the TDNN in a single FPGA chips (XCE5VLX330). The FPGA chip has some free space; therefore we will use this space to re-design the slowest component (20 cell delay in the adder) in a faster component. The target of this step is reducing the delay of adders. Method 3 depends on the redesign of the adder using pipeline technique as in Fig (8), that will increase the cost of each 16-bit adder to 53 cells but it will reduce the maximum delay to 8 cell delay.

Now applied this method on the slowest component in the TDNN, and then re-calculate the cost of the TDNN after the partitioning of the adders. Repeat this method while the total cost of the TDNN is less than the maximum capacity of the FPGA chip.

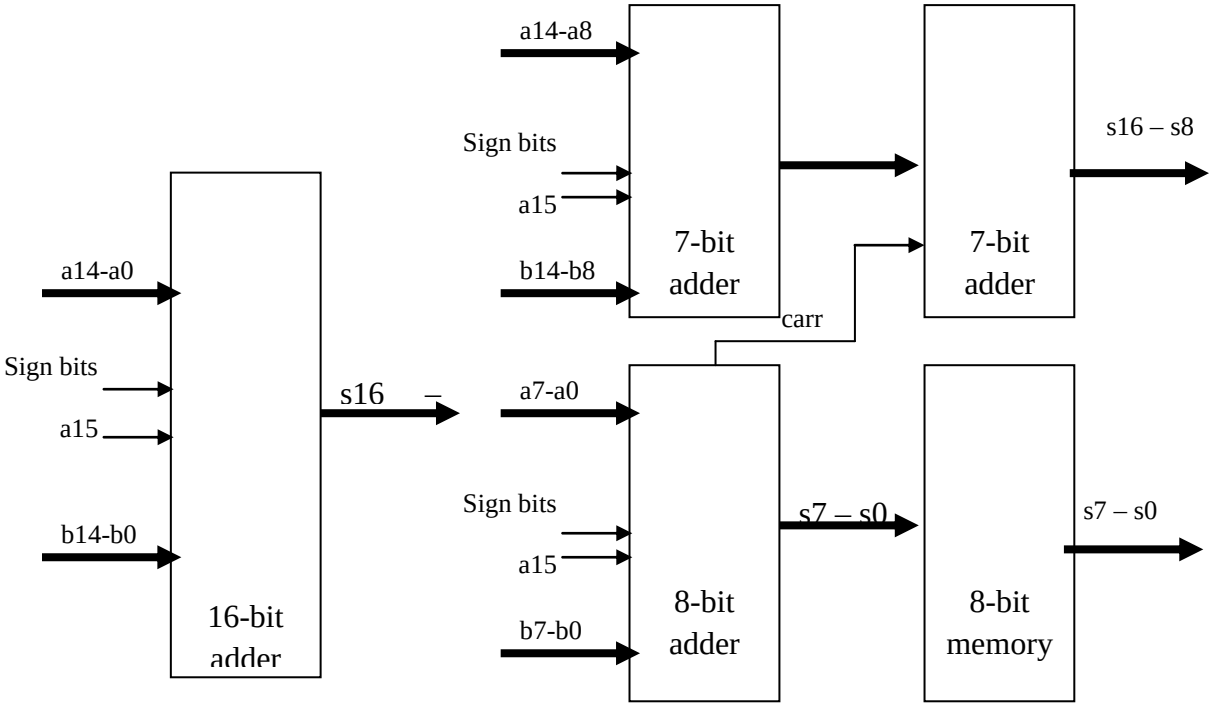
Practical note: the NN capacity must allow some space in the FPGA chip for routing. That is because the full of the FPGA chip will capture the routing process and in result that will increase the maximum delay time over the detected delay.

6- Total Results of the Three Methods

The result in table (2) shows how the good use for the mathematics and logic methods with the digital circuits and systems can reduce cost and delay which is required to implement a useful digital system. The most reduction in the cost comes from method 1 because it deals with the higher cost part (LUT). At this point the digital methods become passive because all the components in TDNN are classic components. Method 2 depends on the modification of the NN to execute its job but in lower cost. This method will increase the final error in TDNN but in acceptable amount, that effect will be discussed later in section 7.

Table (2): The effect of the three proposed methods on the costs and delays of the TDNN.

component	Original		Method 1		Method 2		Method 3	
	cost/cell	delay	cost/cell	delay	cost/cell	delay	cost/cell	delay
adders	37435	20	37435	20	37435	20	**47755**	**14**
multipliers	457440	30	457440	30	**225150**	**12**	225150	12
LUTs	2621440	1	**20400**	**15**	20400	15	20400	15
Total system	3117520	30	523625	30	282985	20	**293305**	**15**



(a) (b)

Fig (8): The block diagram of 16-bit adder a) direct, b) using the pipeline.

The comparation between the initial (original) costs and delays are summarize in table (3). These results are the proposed values therefore they not give the real results but they are indicator to the real results. The real results are found using the hardware simulation of ISE 4.1i software. Table (4) shows the final report results of the hardware simulation of ISE 4.1i software. Some values in this table is scored by the symbol "$^{+}$" because it not available direct in ISE 4.1i, such example the original cost of the TDNN is larger than the XCE5VLX330 chip capacity, therefore it can not be calculate in ISE 4.1i, in this case we will implement the components as multi-units and sum the total cost for all units.

Table (3): The original and the proposed costs and delays of the TDNN.

Component	Original cost / cell	Original delay / nsec	Proposed cost / cell	Proposed delay / nsec
adders	28315	50	35515	34
multipliers	287205	73	218481	30
LUTs	2621440	3	19920	37
Total TDNN	2936960	73	**273916**	**38**

Other important factor in table (4) is the **latency** that is the time spent between first input and first output. This factor is the result of multiply the latency cycles by the time of the cycle (maximum time delay in the system). This factor is very important for the systems that have feedback but it is a secondary factor for the direct output system such as the TDNN.

Table (4): The final report of the TDNN hardware simulation using ISE 4.1i software.

System performance	Measured unit	Original TDNN	Final TDNN
Cost	Value / cell	2936960[+]	273916
	ratio %	100%	9.33%
	XCE5VLX330 chips	**8.86**[+]	**0.826**
Delay	Value/ nsec	73	38
	ratio %	100%	52%
Latency	In cycles	25	29
	In msec	2[+]	1.2[+]

[+] calculated value because it not available directly in hardware simulation.

However, the total cost reduced to about 275,000 cells and the maximum total delay is 38 nsec. That means the new cost is 9.33 % from the original cost with speed up to about two times over the original speed of the TDNN, so it in result the cost of TDNN is suitable for XCE5VLX330 or less Xilinx FPGA.

7- Additional Error Processing

This section will discus the sources of error in the hardware and proposes a practical solution for them. The calculation of the error hasn't rules or equations but it's estimated using a practical example (or examples) with tests. This paper will use the simulation of the TDNN under C++ to find the value of *error* (error in this section mean the approximated normalized value for the accumulated MSE in learning data of the TDNN) for all cases.

The software simulation results in fig (9) with float number has a final error reach to 0.1% while the direct truncation to 16-bit without normalize destroy the result (error up to 70%), while the direct truncation to 16-bit after the normalize increase the error to 25%. The capture of the data and weights in the learning time to 16-bit such as in method 1 gives an error less than 3%. While the capture of the data to 16-bit and weights to 8-bit in the learning time gives an error less than 4% while the method 2 will increase the error to about 4.2%.

Fig (9) shows the output error vs learning cycles of TDNN for software learning (64-bit), 16-bit and 8-bit.

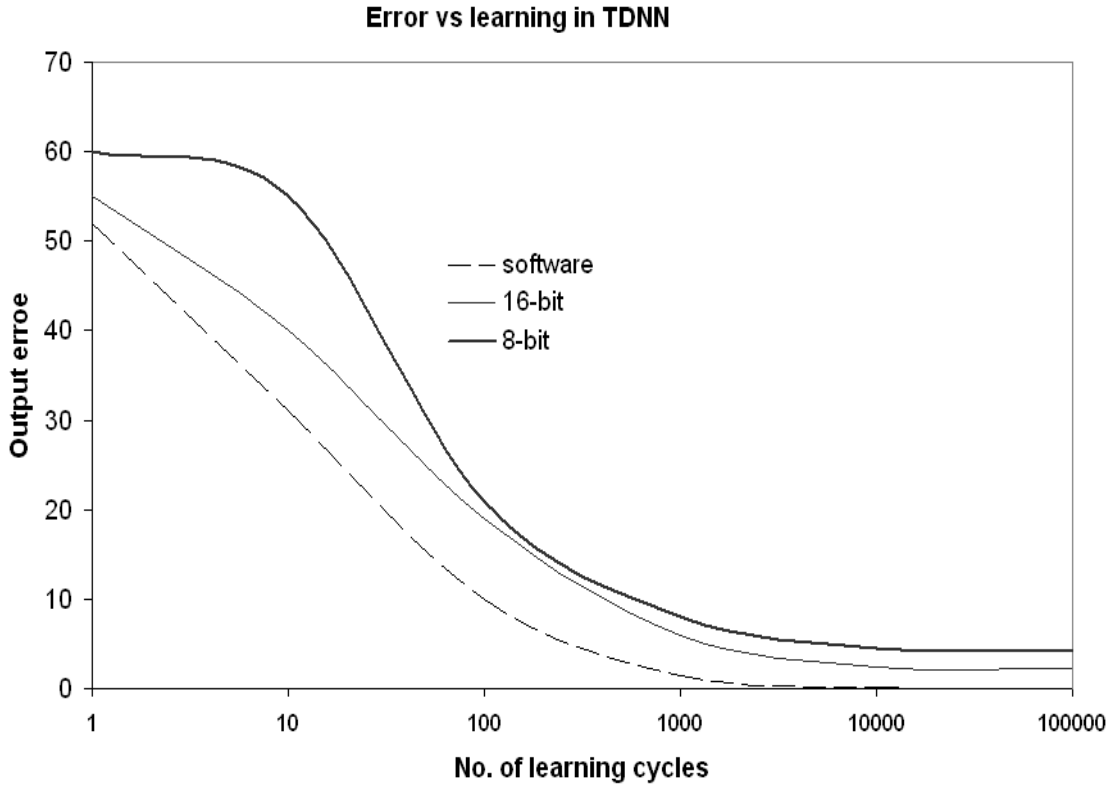


Fig (9): The output error vs learning cycles of TDNN for software learning (64-bit), 16-bit and 8-bit.

8- Discussion

The large cost is the most bottlenecks in the single chip implementation of NN and the activation functions take the most shares in the basic design of hardware NN. However, there are large methods to reduce the cost of the activation functions but most these methods reduce the accuracy of the output. The proposed method (method 1) is a novel method that will reduce the cost of LUTs to about 2% from its original cost.

The multiplications cost is the harder bottleneck in the single chip implementation of NN and there are a few low efficiency methods to reduce this cost. The proposed method (method 2) is also a novel method that will reduce the cost of multipliers to about 12% from its original cost and increase its speed up to 13 times with a small additional error. The problem in this method is that the reduction of the data bus after learning from 64-bits (flout variable) to 16-bit for hardware will increase (jamb) the error in the output results from 0.1% to about 70%. The process of this problem is that reduce the data bus before learning to 16-bit for data and 8-bit for NN weights. This solution increases the error in the output to about 4% (acceptable output error in the NNs).

The combination of the proposed methods in this paper will reduces the cost to about 10 % from the classic implementation methods and increase the total speed up to 2 times for the TDNN. The good point in this paper is that the combination of the proposed methods can be applied for any NN to reduce its cost and increase its speed to implement it as a fast single chip NN.

References

[1] Saumil Merchant, Gregory D. Peterson, Sang Ki Park, and Seong G. Kong, "FPGA Implementation of Evolvable Block-based Neural Networks", Department of Electrical and Computer Engineering, The University of Tennessee, Knoxville, TN 37996-2100, U.S.A. E-mail: {smerchant, gdp, spark14, skong}@utk.edu, 2006 IEEE Congress on Evolutionary Computation Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada pp 3129, July 16-21, 2006.

[2] J. Zhu, G. J. Milne, and B. K. Gunther, "Towards an FPGA based reconfigurable computing environment for neural network implementations," *Proceedings of Ninth International Conference on Artificial Neural Networks (ICANN'99)*, 1999.

[3] Important: Verify all data in this document with the device data sheets found at *www.xilinx.com/virtex5*.

[4] Randall KWeinstein1 and Robert H Lee, "Architectures for high-performance FPGA implementations of neural models", institute of physics publishing journal of neural engineering j. Neural Eng. 3 (2006) 21–34 doi:10.1088/1741-2560/3/1/003.

[5] Gilberto Contreras, Patricia Nava, "Design, Implementation and Testing of an FPGA-based Neuro-Coprocessor", Dept. of Electrical and Computer Engineering, The University of Texas at El Paso 500 W. University Ave. El Paso, TX 79968 pnava@ece.utep.edu, 2002.

[6] Dhafer R. Zaghar "Low cost and high speed look-up table implementation of XILINX FPGA", Engineering and development journal/ P 16, Vol. 9, No. 2, June, 2005.

[7] Dhafer R. Zaghar "Design and implementation of general digital down converter using field programmable gate array"/ Baghdad University / College of engineering / Electrical Dep. Phd thesis 2002.

[8] Howard Demuth and Mark Beale, "Neural Network Toolbox For Use with MATLAB®", Copyright by the MathWorks, Inc, 1992 – 2002.