AENSI Journals

Australian Journal of Basic and Applied Sciences

ISSN:1991-8178

Journal home page: www.ajbasweb.com

FPGA Implementation of Memory Optimized Distributive Arithmetic Neural Network Architecture for Image Compression

[1]Sridhar S, [2]Rajesh Kumar P, [3]Ramanaiah K.V

[1]*Professor, Department of ECE, Lendi Institute of Engineering and Technology, Vizianagaram, Andhra Pradesh, India.*
[2]*Professor, Department of ECE , Andhra University College of Engineering, Andhra University, Visakhapatnam, Andhra Pradesh, India.*
[3]*Professor, Department of ECE, YSR Engineering College, Yogi Vemana University, Proddutur, Andhra Pradesh, India.*

ARTICLE INFO

Article history:
Received 19 August 2014
Received in revised form
19 September 2014
Accepted 29 September 2014
Available online 3 December 2014

Keywords:
Distributive Arithmetic, Field Programmable Gate Array, Feed Forward Neural Network, Image Compression, Low Power Dissipation.

ABSTRACT

Compression and transmission of images over noisy channel impose challenges in reconstruction process as the presence of noise in channel greatly affects the image quality. Neural network approaches for image compression offer good quality images even in the presence of noise. Single hidden layer Feed forward neural network (FFNN) trained with Backpropagation algorithm is considered after designing optimum values of weights. The hardware complexity of FFNN architectures requiring large number of adders and multipliers is reduced with the proposed distributive arithmetic (DA) algorithm based FFNN architectures. The multiplier array is reduced to the look up table (LUT) logic, while the parallel adders and network functions are replaced with memory blocks. Proposed design is modeled in Verilog HDL and synthesized on Xilinx ISE Virtex 5 FPGA device. FFNN architecture on FPGA operates at maximum speed of 222 MHz and consumes power less than 1W occupying 20% of silicon area. Design is suitable for low power applications.

To Cite This Article: Sridhar S, Rajesh Kumar P, Ramanaiah K V., FPGA Implementation of Memory Optimized Distributive Arithmetic Neural Network Architecture for Image Compression. ***Aust. J. Basic & Appl. Sci.,*** *8(18): 445-454, 2014*

INTRODUCTION

Single hidden layer Feed Forward Neural Network (FFNN) with input layer and output layer is chosen for performing image compression in the hidden layer and decompression in the output layer, after the network is appropriately trained. Hardware implementation of neural networks for image compression applications impose many challenges stressing the requirement of large numbers of multipliers, adders and memory elements for computations adding complexity to the design. Hardware circuits designed to implement artificial neural network (ANN) architectures are referred to as hardware neural networks (HNN). HNNs implementation on FPGA platforms is however advantageous to promote reusability and reconfigurability, also FPGA's are cost effective and readily available offering flexibility for online reconfiguration and parallel processing. Krips, Lammert and Kummert (2002) presented FPGA implementation of neural networks for real time hand detection tracking system. Yang and Paindavoine (2003) presented FPGA based hardware implementation on embedded systems. Gadea, Cerda, Ballester and Micholi (2000) implemented systolic array based multi-layer perceptron on Xilinx Virtex XCV400 FPGA device. Girau and Huitzil (2007) implemented a Local Excitatory Global Inhibitory Oscillator Network spiking neural model for image segmentation. Himavathi, Anitha and Muthuramalingam (2007) used layer multiplexing technique for FFNN on Xilinx FPGA XCV400hq240. Rice, Taha and Vutsinas (2009) reported FPGA implementation of cognitive model to provide throughput gain of 75 times over software implementation. Szabo, Antoni, Horvath and Feher (2000) suggested distributed arithmetic methods to implement digital filters based on Canonic Signed Digit (CSD) algorithm.

Most of the works reported in literature considered multilayer FFNN architectures for various applications rather than image compression. Since multilayer architectures require massive number of parallel blocks for computations, the data movement among the layers needs to be pipelined for better throughput and latency values. Intermediate memory blocks are used for storing synaptic weights and data movement within the network layers. This work is therefore an attempt to realize and implement different parallel processing NN architectures on FPGA and ASIC platforms for optimizing the parameters like speed, power and area additional

Corresponding Author: S Sridhar, Department of ECE, LENDI Institute of Engineering and Technology, Vizianagaram, Andhra Pradesh, INDIA.
E-mail: Sridhar.vskp@gmail.com

to the regular objective fidelity metric measures- the peak signal to noise ratio (PSNR), mean square error (MSE) and the Percentage Compression etc.

Rest of the manuscript is organized as follows: different parallel processing neural network architectures are described in Section 2. Proposed distributive arithmetic FFNN architectures for image compression are elaborated in Section 3. HDL based FPGA implementation and result analysis is performed in section 4 followed by concluding remarks in section 5.

Optimized Parallel FFNN Architecture:

After appropriate training, the weights and bias values of network are identified for further hardware modelling the desired neural network architectures. FFNN of 16-4-16 dimensions is considered for analysis, having 16 neurons in the input and output layers and 4 neurons in the hidden layer. The 2-D input image of size K×K is rearranged into 1-D data by segmenting it into M number of blocks, each of size N×N pixels. Input data (X) matrix of size 16×1 is compressed into 4×1 together with the 4 x 16 weight matrix (Wf) and the 4 bias values of the hidden layer, as defined in Equation 1.

$$[Wf * X] + Bias = C \quad . \tag{1}$$

The 4×1 compressed matrix (C) is further decompressed to 16×1 (C') in the output layer using the 16 x 4 layer weight matrix (WS), as defined in Equation 2.

$$[Ws = 16\times4] * [C' = 4\times1] + bias = [C' = 16\times1] \tag{2}$$

16 elements input data(X) at input layer is broadcast to the four hidden layer neurons, such that each input is multiplied with the corresponding weight, as defined in Equation 3.

$$\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} & \cdots & W_{1,16} \\ W_{2,1} & W_{2,2} & W_{2,3} & \cdots & W_{2,16} \\ W_{3,1} & W_{3,2} & W_{3,3} & \cdots & W_{3,16} \\ W_{4,1} & W_{4,2} & W_{4,3} & \cdots & W_{4,16} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ . \\ . \\ . \\ . \\ . \\ . \\ X_{16} \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix} \tag{3}$$

Further, outputs of the four neurons at hidden layer are added with the respective bias values as defined in Equation 4.

$$a1 = C1 + b1 \tag{4}$$

Subsequently, the hidden layer outputs a1 to a4 are processed with activation function for obtaining the final compressed outputs Y1, Y2, Y3 and Y4 as defined in Equations 5 to 8.

$$Y1 = Tansig(a1) \tag{5}$$
$$Y2 = Tansig(a2) \tag{6}$$
$$Y3 = Tansig(a3) \tag{7}$$
$$Y4 = Tansig(a4) \tag{8}$$

Proposed FFNN architecture for compression is shown in Figure 1. Here input memory stores the two level decomposed discrete wavelet transform (DWT) sub band coefficients for compression with neural network architecture. Hidden layer neurons process sub band components by taking 16 inputs every time. Four intermediate parallel memories are used to store the 16 inputs for multiplication with network weights in the four multiplier arrays. Products are accumulated in adder array with bias values and processed in translinear (tansig) activation function for storing the compressed data in the output memory. As the number of arithmetic operations is large, control logic is used to control the data processing operations with control enable (CE) signals for pipelined processing, as shown in Figure 1.

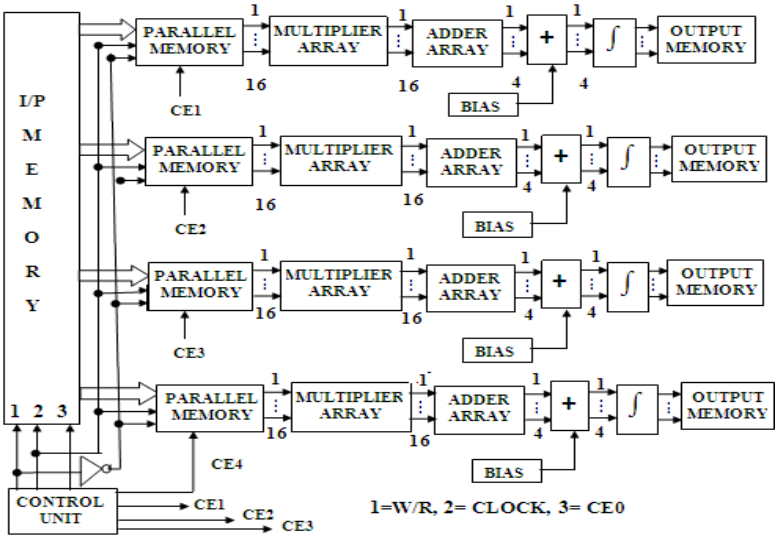

Fig. 1: Data Synchronization for Compression Unit

Distributive Arithmetic (DA) FFNN Architectures:

After re-organizing the 2D input image into 1D data, the weights and bias values are appropriately scaled to integers by multiplying them with 256 and represented in 9-bit fixed point format for hardware implementation. Scaled weights and input data are stored in memory for data processing. Proposed novel architecture performs the translation of 16 x 1 input data into 4 x 1 data. Equation 3 can now be expressed as under in Equation 9.

$$Y_k = \sum_{n=1}^{16} W_{k,n}\, \chi_n \qquad \text{Where k= [1, 2, 3, 4]} \tag{9}$$

Expanding Equation 9, for k=1. Equation 10 is obtained as under:

$$Y1 = W_{1,1} X_1 + W_{1,2} X_2 + W_{1,3} X_3 + \ldots\ldots + W_{1,16} X_{16} \tag{10}$$

Computation of equation 10 requires 16 multipliers and 15 adders. Therefore, computations of the four hidden layer neuron (compressed outputs) outputs: Y1, Y2, Y3 and Y4 require 64 multipliers and 60 adders. Since the four outputs are computed simultaneously the computational complexity of the architecture occupies more area. In order to reduce the number of multipliers and increase the frequency of operation novel FFNN architectures for image compression based on distributive arithmetic (DA) algorithm are designed and implemented.

Proposed DA Neural Network Architecture:

By expanding input X in terms of 8 bit binary numbers in Equation 10, Equation 11 can be obtained as under:

$$\begin{aligned} Y1 = {} & X_1^9 W_{1,1} + X_1^8 W_{1,1} + X_1^7 W_{1,1} + \ldots\ldots\ldots\ldots\ldots\ldots + X_1^2 W_{1,1} + X_1^1 W_{1,1} \\ & + X_2^9 W_{1,2} + X_2^8 W_{1,2} + X_2^7 W_{1,2} + \ldots\ldots\ldots\ldots\ldots\ldots + X_2^2 W_{1,2} + X_2^1 W_{1,2} \\ & + \ldots\ldots\ldots\ldots\ldots\ldots \\ & \ldots\ldots\ldots\ldots\ldots\ldots \\ & + X_{16}^9 W_{1,16} + X_{16}^8 W_{1,16} + X_{16}^7 W_{1,16} + \ldots\ldots\ldots\ldots + X_{16}^2 W_{1,16} + X_{16}^1 W_{1,16} \end{aligned} \tag{11}$$

Where, $X_{1,1}^9 \ldots\ldots\ldots X_{1,16}^9$ Are binary numbers with values '0' or '1' $X_{1,1}^9$ Represents the MSB bit and $X_{1,1}^8$ Represent the MSB-1 bit. Now, by representing the binary weights at various bit positions, Equation 12 is obtained as under.

$$\begin{aligned} Y1= & (X_1^9)2^8\ W_{1,1}+(X_1^8)2^7\ W_{1,1}+\ldots\ldots\ldots\ldots\ldots\ldots+(X_1^1)2^0\ W_{1,1} \\ & +(X_2^9)2^8\ W_{1,2}+(X_2^8)2^7\ W_{1,2}+\ldots\ldots\ldots\ldots\ldots\ldots+(X_2^1)2^0\ W_{1,2} \\ & +\ldots\ldots\ldots\ldots\ldots\ldots \\ & \ldots\ldots\ldots\ldots\ldots\ldots \\ & +(X_{16}^9)2^8\ W_{1,16}+(X_{16}^8)2^7\ W_{1,16}+\ldots\ldots\ldots\ldots\ldots\ldots+(X_{16}^1)2^0\ W_{1,16} \end{aligned} \quad (12)$$

Further, by grouping the product elements vertically in terms of binary weights, Equation 12 is rearranged to generate Equation 13.

$$Y1 = \begin{bmatrix} (X_1^9)W_{1,1} \\ + \\ (X_2^9)W_{1,2} \\ + \\ (X_3^9)W_{1,3} \\ . \\ . \\ . \\ . \\ +(X_{16}^9)W_{1,16} \end{bmatrix} 2^9 + \begin{bmatrix} (X_1^8)W_{1,1} \\ + \\ (X_2^8)W_{1,2} \\ + \\ (X_3^8)W_{1,3} \\ . \\ . \\ . \\ . \\ +(X_{16}^8)W_{1,16} \end{bmatrix} 2^8 + \ldots\ldots\ldots\ldots \begin{bmatrix} (X_1^1)W_{1,1} \\ + \\ (X_2^1)W_{1,2} \\ + \\ (X_3^1)W_{1,3} \\ . \\ . \\ . \\ . \\ +(X_{16}^1)W_{1,16} \end{bmatrix} 2^0 \quad (13)$$

Since there are only four neurons in hidden layer, the outputs of remaining 3 hidden layer neurons (Y2, Y3 and Y4) are obtained by replacing the value of K by K= 2, 3 and 4 in Equation 9 and repeating the above procedure of calculating Y1. Therefore, Equation 14 for Y2 is obtained as under by putting K=2:

$$Y2= \begin{bmatrix} (X_1^9)W_{2,1} \\ + \\ (X_2^9)W_{2,2} \\ + \\ (X_3^9)W_{2,3} \\ . \\ . \\ . \\ . \\ +(X_{16}^9)W_{2,16} \end{bmatrix} 2^9 + \begin{bmatrix} (X_1^8)W_{2,1} \\ + \\ (X_2^8)W_{2,2} \\ + \\ (X_3^8)W_{2,3} \\ . \\ . \\ . \\ . \\ +(X_{16}^8)W_{2,16} \end{bmatrix} 2^8 + \ldots\ldots\ldots\ldots + \begin{bmatrix} (X_1^1)W_{2,1} \\ + \\ (X_2^1)W_{2,2} \\ + \\ (X_3^1)W_{2,3} \\ . \\ . \\ . \\ . \\ +(X_{16}^1)W_{2,16} \end{bmatrix} 2^0 \quad (14)$$

Similarly, substituting K=3 for Y3, Equation 15 is obtained

$$Y3= \begin{bmatrix} (X_1^9)W_{3,1} \\ + \\ (X_2^9)W_{3,2} \\ + \\ (X_3^9)W_{3,3} \\ . \\ . \\ . \\ . \\ +(X_{16}^9)W_{3,16} \end{bmatrix} 2^9 + \begin{bmatrix} (X_1^8)W_{3,1} \\ + \\ (X_2^8)W_{3,2} \\ + \\ (X_3^8)W_{3,3} \\ . \\ . \\ . \\ . \\ +(X_{16}^8)W_{3,16} \end{bmatrix} 2^8 + \ldots\ldots\ldots\ldots\ldots + \begin{bmatrix} (X_1^1)W_{3,1} \\ + \\ (X_2^1)W_{3,2} \\ + \\ (X_3^1)W_{3,3} \\ . \\ . \\ . \\ . \\ +(X_{16}^1)W_{3,16} \end{bmatrix} 2^0 \quad (15)$$

Substituting K=4 for Y4, Equation 16 is obtained:

$$Y4=\begin{bmatrix}(X_1^9)W_{4,1}\\+\\(X_2^9)W_{4,2}\\+\\(X_3^9)W_{4,3}\\.\\.\\.\\.\\+(X_{16}^9)W_{4,16}\end{bmatrix}2^9+\begin{bmatrix}(X_1^8)W_{4,1}\\+\\(X_2^8)W_{4,2}\\+\\(X_3^8)W_{4,3}\\.\\.\\.\\.\\+(X_{16}^8)W_{4,16}\end{bmatrix}2^8+\ldots\ldots\ldots+\begin{bmatrix}(X_1^1)W_{4,1}\\+\\(X_2^1)W_{4,2}\\+\\(X_3^1)W_{4,3}\\.\\.\\.\\.\\+(X_{16}^1)W_{4,16}\end{bmatrix}2^0 \quad (16)$$

If the weights W1, W2…..W16 are constant and Xkn1, n2 represent the binary values at k positions, by regrouping the weights vertically-binary numbers at each position are multiplied with input to accumulate the products. The first term in Equations 13, 14, 15 and 16 represent the accumulated product of the MSB (Most Significant Bit) of input X and corresponding neuron weight. Similarly, the second term in Equations 13, 14, 15 and 16represent the product of MSB-1 bit position of inputs and weights. Computations are carried out for all bit positions by scaling the output of second term by 2 and adding the product obtained from the first term. Since, the binary bits at each term consist of 16 MSB bits there are 216 possible partial products of the input matrix. Further the precomputed partial products are stored in a ROM; the inputs are loaded into a shift register and used as address bits to read the corresponding partial products from ROM. Each of the partial products is right shifted and accumulated with successive partial products to compute the final output.

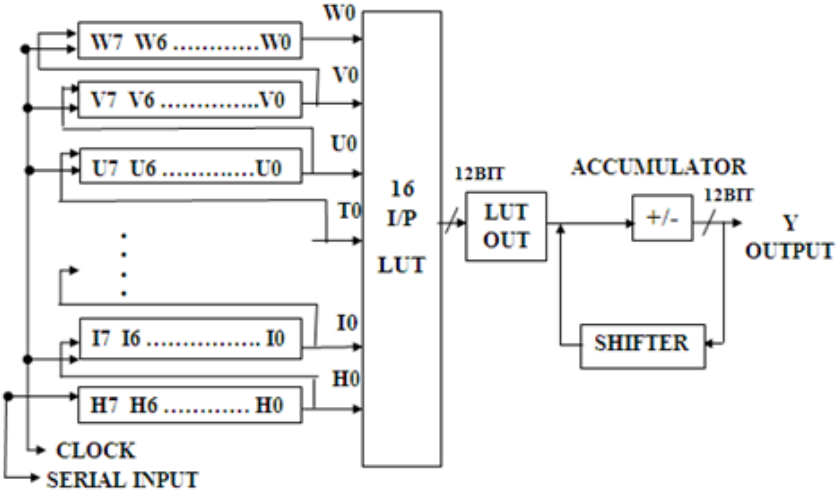

Fig. 2: Proposed DA Architecture for NN

Proposed DA architecture for realizing equation 13 (as shown in Figure 2) include 16 eight bit registers that are serially loaded in synchronization to the clock input. The first output Y1 is obtained after (16*8 + 8) clock cycles, the samples Y2, Y3, and Y4 are computed at the end of first (16*8 + 8) clock cycles. For computing the Y2, Y3 and Y4 samples, serial in serial out (SISO) shift register contents are used to address another three look up tables for storing different sets of precomputed partial products based on weights. Figure 3 shows the proposed DA FFNN compression architecture to compute four output samples. Shift registers are common to all the four memory contents to access the precomputed addresses based on weights.

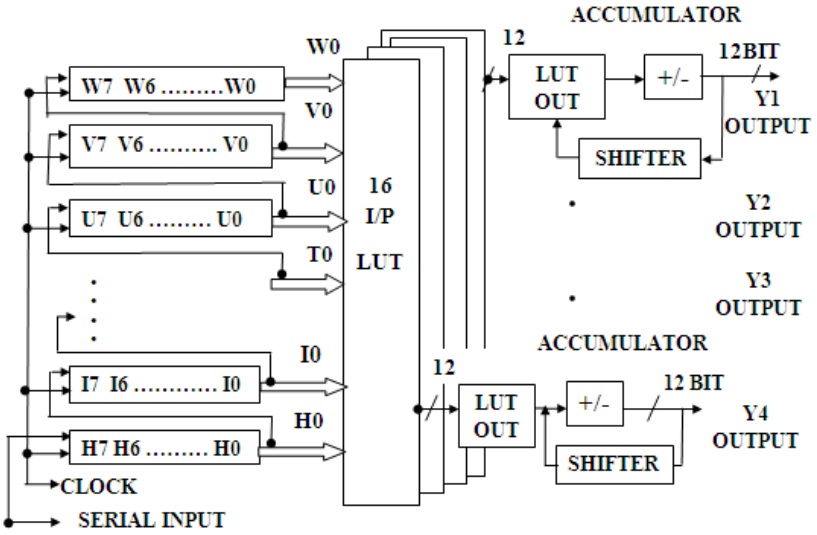

Fig. 3: DA Based Parallel Processing FFNN Architecture

Modified DA FFNN Architecture:

In the distributive arithmetic FFNN architecture for image compression, multiplier and adder stages are eliminated to reduce the circuit complexity. Bias values are added to the computed outputs of all four hidden layer neurons. Computed outputs are further used as addresses to access the ROM contents for completing the tansig operation. Modified architecture is shown in Figure 4.

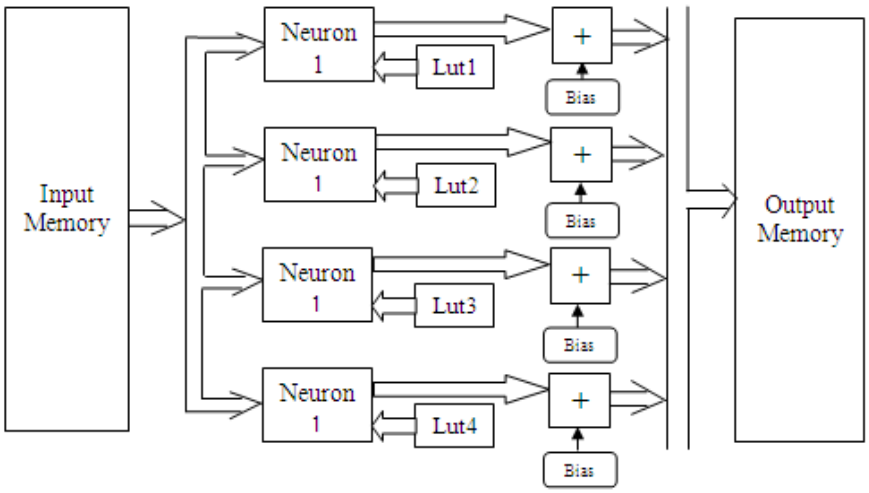

Fig. 4: Modified Compression units with DA Algorithm

Memory reduced DA FFNN architecture:

The main limitation of DA FFNN architecture for image compression is that the numbers of inputs are 16 and size of LUT required is 216. In the proposed memory reduced architecture, size of LUT is reduced by splitting the input sample into two halves. Such that, First 8 Serial in serial out (SISO) registers access the top LUT and bottom 8 SISO registers access the bottom LUT. Accumulated data is further added to compute the final output. In the split DA architecture of Figure 5, size of LUT is reduced from 64K to 512 for compression unit and 24 to 2*22 for decompression unit. Number of adders is increased from 1 to 3 compared with basic modified DA FFNN architecture discussed above. Proposed memory reduced architecture saves memory by more than 99%, thus it is more suitable for FPGA implementation as it occupies more look up tables for implementation.

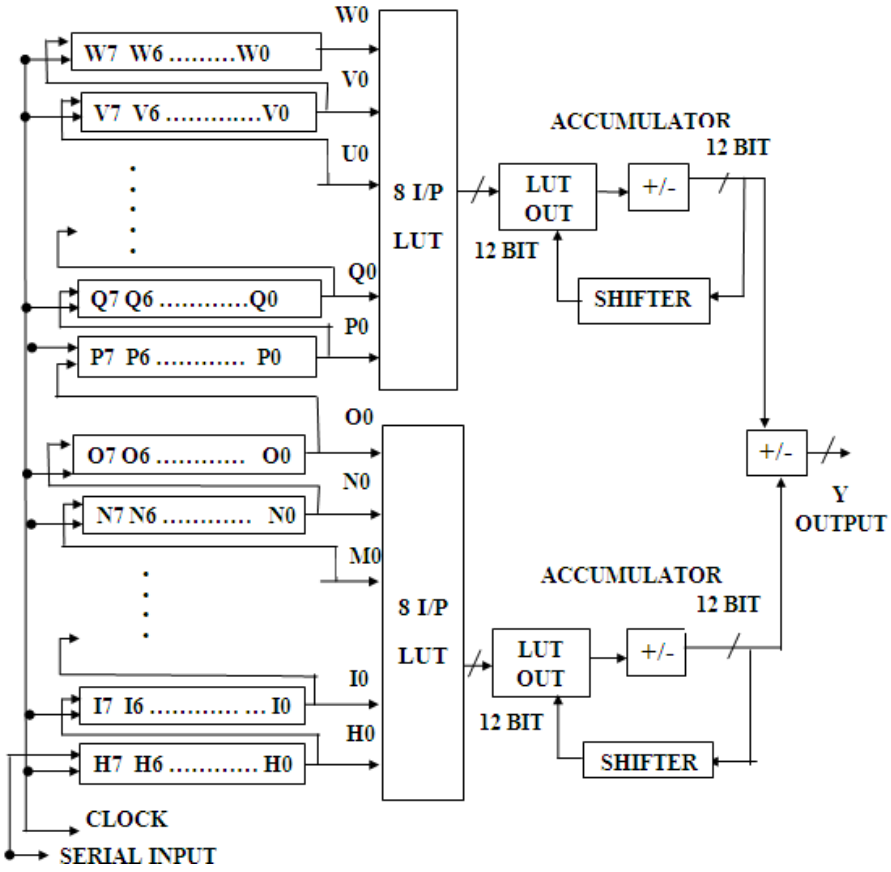

Fig. 5: Memory reduced DA FFNN architecture for compression

Table 1shows a comparative performance analysis of DA FFNN architectures. In case of the direct DA architecture, LUT size is reduced from 64K to 512 for the compression unit, and for decompression unit the size of LUT is reduced from 2^4 to $2*2^2$. However, the number of adders is increased.

Table. 1: Comparison of computational complexities in DA architecture

	Modified DA FFNN	Memory Reduced DA FFNN
LUT Size	2^{16}	$2^8 + 2^8$
Latency	8*16+8	8*16+9
Throughput	8	8
Adders	1	3
SISO	Required	Required
PISO	Not required	Not required

RESULTS AND DISCUSSIONS

Proposed architecture is modelled in Verilog HDL and functional verification is done using suitable test cases. A test environment is developed to test the logic correctness of proposed DA logic. From the simulation results obtained in ModelSim, the developed HDL model is found to produce correct results for all possible test vectors. The largest available Xilinx Virtex FPGA device, XCV300 is used as target device for performance verification of proposed architecture. Figure 6 shows the simulation results of the compression unit with known input test vectors.

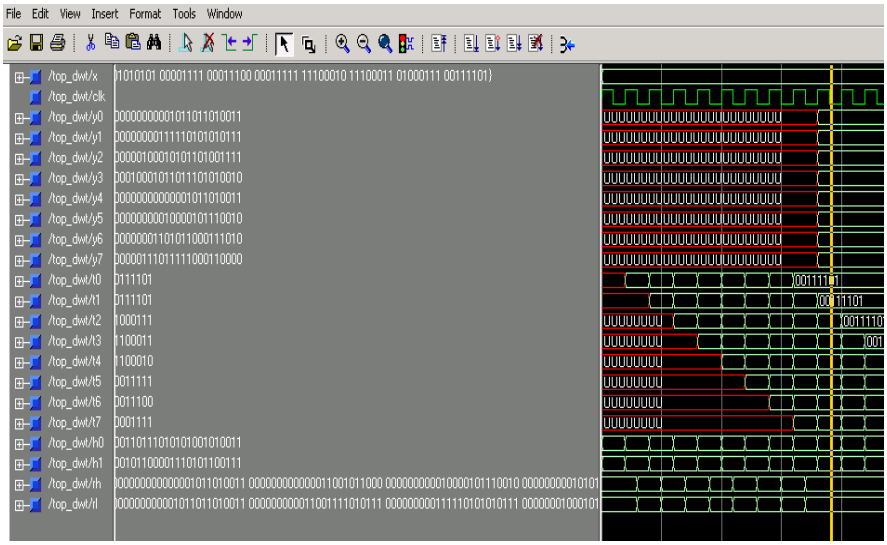

Fig. 6: Simulation Results of Compression Unit

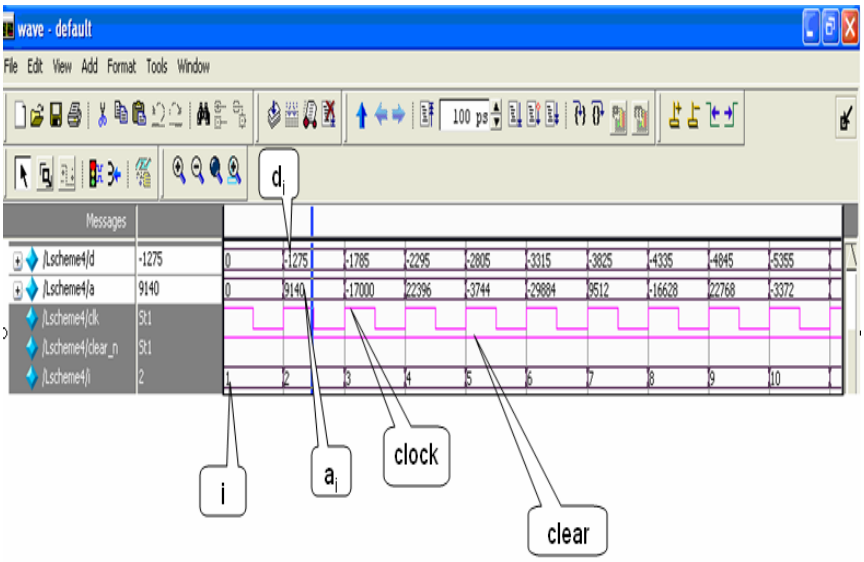

Fig. 7: Simulation Results of Decompression Unit

Simulation results of decompression unit shown in Figure 7 are found to match with the theoretical results on comparison.

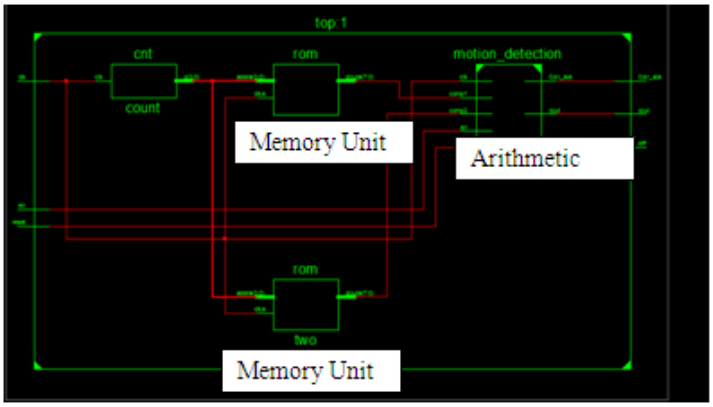

Fig. 8: Synthesis Results of Proposed Architecture

Table.2 shown below clearly illustrates the timing summary. Where, performance is normally measured with respect to two evaluation metrics; the throughput (or sample rate) which is denoted in terms of clock speed and the device utilization which is defined in terms of number of Virtex logic slices used for implementation.

Table. 2: Timing Summary

Timing Summary	
Speed Grade :	**-5**
Minimum Period	**4.498ns**
Maximum Frequency	**222.334 MHz**
Minimum input arrival time before clock	**6.446 ns**
Maximum output required time after clock	**9.147 ns**
Maximum combinational path delay	**No Path Found**
Timing Detail : All values displayed in nanoseconds (ns)	

Table 3 depicts the performance analysis of proposed and implemented DA FFNN architectures with those of existing hardware neural network (HNN) architectures with respect to parameters like area, speed and power etc. Direct FFNN image compression architecture seem to consume 1.29W power on FPGA and 42362 slices. While, the DA-FFNN architecture is able to exploit the FPGA resources and reduce the redundancy by replacing multipliers with look up tables thereby reducing the number of slices (by 34%) to 33412 and power to 1.01W from the 1.29W. However, the memory optimized DA-FFNN architecture has the least power dissipation of 0.91W. Power reports are generated from the XIlinx ISE and the dynamic power is computed with all possible switching vectors based on the synthesized netlist. Experimental analysis clearly implies that power reduction can be achieved with intelligent modifications in the designs at the architectural level.

Table 3: Comparative analysis of Relative HNN Architectures

Parameters	Direct FFNN	Modified FFNN	Memory-Reduced FFNN	Suhap Sahin et al	Pinjare et al	
Maximum Frequency	102MHz	201 MHz	**245 MHz**	50 MHz	--	--
Maximum Power	1.29W	1.01W	0.91W	--	--	--
No. of Slices	42362	33412	22471	2352 slices	--	
No. of Gates	4596K	3517K	1478K	--	81% Area	13% Area
Design type	FFNN Architecture	Modified Architecture	Modified Architecture	2-3-2 ANNs	NN Architectu re	NN Architectur
Platform	Virtex5 XCV300	Virtex5 XCV300	Virtex5 XCV300	Spartan II FPGA	Spartan3 FPGA	Virtex-2 Pro FPGA

Architectures of Suhip Sahin et al (2006) and Pinjare et al (2006) mainly focused upon addressing the issues involved in FPGA implementation of neural network and implementation of trainable Artificial Neural Network (ANN) chips for basic gates, demanding the usage of large neural networks involving more area and delay in relation to the proposed architectures. Better fidelity metric values are obtained in terms of the Peak Signal to Noise Ratio (PSNR) and Mean Square Error (MSE). PSNR values obtained are in the ranges of 30-35 for medium detail images and even of the order of 41-48 for low detail images, while the MSE values are less in figures. Compression percentage obtained is around 75 for the proposed network dimensions.

Conclusion:

In this work, a new MLNN architecture for image compression and decompression is proposed, designed, modelled, verified and implemented on hardware. MLNN architectures performance in terms of MSE and PSNR for image compression and decompression is identified and compared. MLNN architecture consisting of Tansig and Purelin functions in the hidden layer and output layer is implemented on FPGA. The FFNN architecture consists of multipliers and adders that reduce the propagation delay, in order to reduce the delay and area occupied to improve the operating frequency, modified DA algorithm is designed and implemented on FPGA. Further the modified DA architecture is improvised for memory utilization based on memory split algorithm. Proposed architectures are compared for performances targeting Virtex 5 FPGA devices. FFNN architecture on FPGA operates at a maximum speed of 222 MHz consuming power of less than 1 W occupying 20% of the resources. The design implemented on FPGA can be interfaced with camera and used for real-time signals processing.

ACKNOWLEDGEMENT

The authors express sincere thanks and gratitude to the management of LIET affiliated to Jawaharlal Nehru Technological University for the support and experimental facilities provided for carrying out the proposed work.

REFERENCES

1. Krips, M., T. Lammert and A. Kummert, 2002. Proceedings of First IEEE International Workshop on Electronic Design Test and Applications, pp: 313-317.
2. Yang F and Paindavoine, 2003. IEEE Transaction on Neural Networks, 14(5): 1162-1175.
3. Gadea, R., J. Cerda, F. Ballester and A. Micholi, 2000. Proceedings of 13th International Symposium on System Synthesis, pp: 225-230.
4. Huitzil, T and B. Girau, 2007. Massively distributed digital implementation of an integrate-and-fire legion network for visual scene segmentation, Neurocomputing, 70(7–9): 1186-1197.
5. Himavathi, S., D. Anitha and A. Muthuramalingam, 2007. IEEE Transactions on Neural Networks, 18(3): 880-888.
6. Rice, K., T. Taha and C. Vutsinas, 2009. Scaling analysis of a neocortex inspired cognitive model on the Cray XD1, The Journal of Supercomputing, 47(1): 21-43.
7. Szabo, T., L. Antoni, G. Horvath and B. Feher, 2000. INNS–ENNS International Joint Conference on Neural Networks, 2: 2049.
8. Sahin, S., Y. Becerikli and S. Yazici, 2006.ICONIP, Part III, pp: 1105-1112.
9. Pinjare, S.L. and M. Arun Kumar, 2012. International Journal of Computer Applications, 52: 6.