International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 19 (2018) pp. 14135-14141
© Research India Publications. http://www.ripublication.com

Optimized Mathematical Model of Digital Circuit using ANN on FPGA

Virendra V. Shete
Department of Electronics and Telecommunication Engineering,
MIT Collage of Engineering, Survey No. 124, MIT College Campus,
Ex-Serviceman Colony, Paud Road, Kothrud, Pune, Maharashtra, 411038, India.
Email id: virendra.shete@mitcoe.edu.in

Mangesh Islampurkar, Kishanprasad Gunale
(Co- Authors)
Department of Electronics and Telecommunication Engineering,
MIT Collage of Engineering, Pune, Maharashtra, 411038, India.
electronics.mangesh@gmail.com
kishanprasad.gunale@mitcoe.edu.in

Abstract

To calculate the response physical circuit for the unknown condition, it is necessary to understand the physical circuit using mathematical model. The mathematical model is useful in case of design the control system or fault detection system for the external circuit. The proposed system deals with implementation of the mathematical model of the physical circuit using artificial neural network algorithms on FPGA. In the proposed system, the circuit under test (CUT) is connected to FPGA. FPGA applied test pattern input to the physical circuit, generate the weights and calculate the activation function. As the testing inputs are applied to the FPGA after successful training, the outcome results of the system are logically similar with a comparison of the external test circuit. To reduce the Register transfer level (RTL) logic design Vedic multiplier and divider using subtract and increment method is implemented in proposed system.

Keywords: Mathematical Modelling, ANN, FPGA, Vedic Maths, VLSI

INTRODUCTION

The artificial neural network algorithms can be designed and synthesized using FPGA. There are various methods and techniques used to synthesize the mathematical equation of artificial neural network on FPGA.

Before describing a physical device using a mathematical model, it is necessary to analyse a mathematical model using laws of digital electronics. This mathematical model is able to describe every digital circuit with similar structure but different logical behaviour. Once the mathematical model of the system is derived, the unknown test inputs and various algorithms are applied to the mathematical model instead of the physical device [1]. Faults with various natures can be applied to a mathematical model instead of a physical device to avoid open or short circuit damage. The mathematical model is operating on FPGA operating frequency instead of physical device operating frequency. As a result, the speed of analysing the physical device is increased. The mathematical model is also helpful to design control system for the physical device. After designing a control system for the mathematical model, it is tested on the real system.

To derive any mathematical model using artificial neural network algorithms, it is necessary to synthesize the digital system using arithmetical and logical operation. Designing and optimize logical operation is simple compared to design the arithmetic operation. It is necessary to synthesize the multiplier in the process of mathematical modelling. During the synthesizing time, the multiplier model, the compiler is trying to synthesize the hardware multiplier. In case of microcontrollers, the multiply operation is done using micro programs at the same time Discrete Signal Processor (DSP) uses built-in hardware multiplier [2]. As a result, the compiler synthesizes the DSP module to perform multiplier operations. The numbers of DSP modules are limited in FPGA. By reason of this, it is necessary to synthesize the dedicated structure for the multiplication operation. There are various algorithms used to synthesize the hardware multiplier like the Series multiplier, Booth multiplier, Parallel multiplier, Vedic multiplier etc. Subtract and increment method is used in case of division operation. In this method, deviser is subtracted from dividend until deviser is smaller than the dividend. The quotient is incremented during every subtraction operation and the dividend is modified. When the dividend is less than the divisor, the modified dividend is considered a reminder of the operation.

LITERATURE REVIEW

Ravikant et. Al. [3] describes the multilayer artificial neural network. The system-on-chip design methodology is used in this paper. The author focuses on feed forward neural network. This feed-forward network employs the back-propagation algorithm for training. In this paper, the weight generation process uses FPGA architecture in system-on-chip design and activation functions are calculated using embedded architecture.

Several researchers have reported the implementation of ANNs in both software and hardware. Quick constructability, adaptively and testability for a wide range of applications are some of the attractive features of the software implementation of ANNs [4]. Corresponding implementations can be found in [5, 6]. However, the latter papers do not consider hardware implementation.

An early attempt of FPGA implementation of a Feed-Forward Neural Network (FFNN) is reported in [7]. Owing to the use of general purpose data path elements, the area trade-off in this implementation is not suitable for networks that have multiple hidden layers. The authors in [8] present a hardware implementation of multilayer FFNN using existing system generator library functions that are mapped on to the FPGA during synthesis. In this implementation, code from Simulink blocks has been mapped on to corresponding FPGA resources without structural optimization. To implement a multilayer FFNN, [9] used the pipeline-based architecture methodology. The speed of operation as reported in the paper is limited to 10MHz while up to 72% of available resources on Virtex XCV400 device were consumed.

Various implementations [10] reported in the literature either seek to implement and verify the network in MATLAB or LABVIEW and use a cross-compiler to map the code on to an FPGA resulting in sub-optimal hardware implementation. Part of implementations reported in [11] and [12] seek to leverage pulsed neural networks for achieving desired area trade-off for on-chip implementation. However, the limitations of such approaches have been outlined in [10]. Use of multiprocessor-on-chip design for implementation of neural networks has been reported in [13]. In such approaches use of existing soft-core processor, IPs do not provide the required trade-off for scalability both in terms of the number of neurons per layer and number of layers.

M. Akila et. Al. [14] explains a design and implementation of the Vedic multiplier. According to the paper, they aim at developing a multiplier using modified Adder which has been implemented “URDHVA TIRYAKBHYAM” sutra with improved speed of operation. In this research, they used an 8x8 multiplier and ripple carry adder to design 16-bit multiplier.

PROPOSED SYSTEM

The proposed system deals with design training ANN and activation function algorithms using FPGA. The weights of an artificial neuron are calculated according to the logical behaviour of the external training circuit. The block diagram of the proposed system is shown in figure 1.

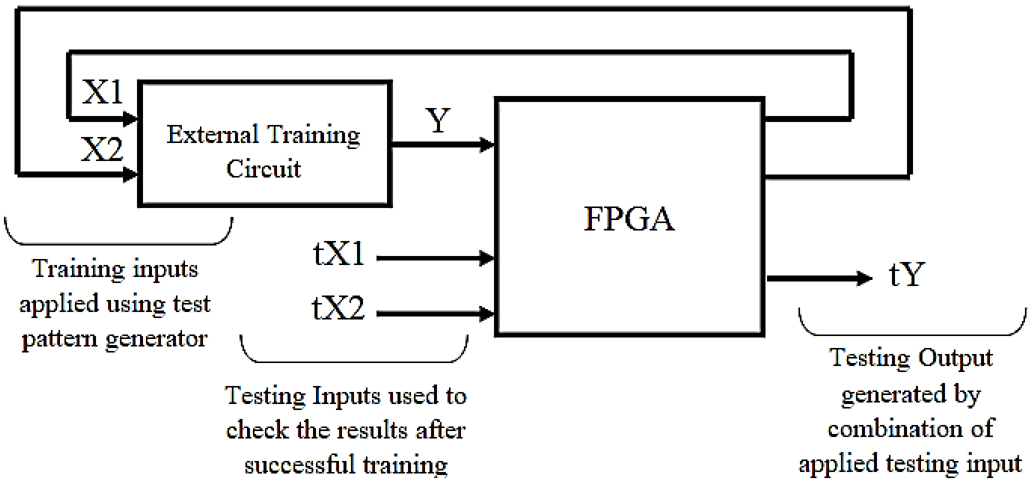
X1
X2
External Training Circuit
Y
FPGA
tX1
tX2
tY
Training inputs applied using test pattern generator
Testing Inputs used to check the results after successful training
Testing Output generated by combination of applied testing input

Figure 1. Block diagram of Proposed System

In this system, the external training circuit applied is two input one output gate. In the beginning, the FPGA generates test pattern signals X1 and X2 and applied to external training circuit. The output Y is a response generated by the external training circuit for a particular input test pattern. The Hebbian weight calculation algorithm applied to generate and calculate training weights of ANN. Based on the expected response of the output of the external circuit (Y), the activation function calculations are modified. After successful training of the Network, Testing inputs tX1 and tX2 are applied to check the response of ANN. The output tY is the resultant output generated by the weight calculated, activation function and the combination of applied testing inputs.

Artificial Neural Network

a. Weight Generation and Calculation

There are two stages of operation used to calculate net weight: weight generation and weight calculation. In the proposed system, Hebbian weight calculations are used to calculate net weight. Initial weight present in the neural network is considered as zero. As the test pattern applied to the external test circuit, output Y is generated. According to Hebbian weight generation method, output (Y) generated by external test circuit is multiplied with the test input (Xi). If the simple multiplier is used in the multiplication operation, it synthesizes DSP which leads increasing in RTL logic design.

To reduce the RTL logic design, a Vedic multiplier is designed and implemented using Urdhva-Tiryakbhyam Sutra.

The mathematical model for two data inputs and one bias input perceptron can be derived as shown in equation 1,

$$\text{tX2} = \text{tX1} * \frac{W1}{W2} + b * \frac{Wb}{W2} \quad (1)$$

After successful training, the weights W1, W2 and Wb generated for logical OR gate are 2, 2 and 2 respectively. Substituting the value of weights in equation 1, equation 2 can be written as,

$$\text{tX2} = \text{tX1} + b \quad (2)$$

b. Activation Function

In proposed system, linear activation function is used. The activation function consists of two models: Calculating equation and decision logic. Calculating equation combines the weights generated with the unknown input condition as shown in equation 1 and 2. Decision logic is used to decide the output condition based on calculated equation. The RTL design synthesized for activation function is shown in figure 2.

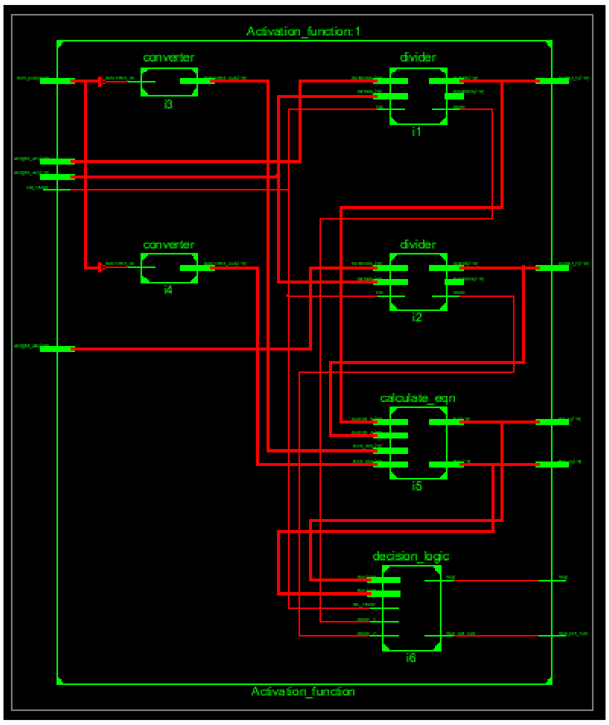

Figure 2. RTL Design for Activation Function

Vedic Multiplier

According to Urdhva-Tiryakbhyam Sutra, the multiplication can be possible with few steps and in less time. This method leads vertically and crosswise multiplication of the input numbers. Vedic Sutras are designed to multiply unsigned numbers. In proposed system, signed inputs are converted to unsigned format using 2's complement method before multiplication. After multiplication, the result converted to signed number. The N-bit Vedic multiplier is designed using interconnecting 2x2 Vedic multipliers. The logic diagram used to design 2x2 Vedic multiplier is shown in figure 3.

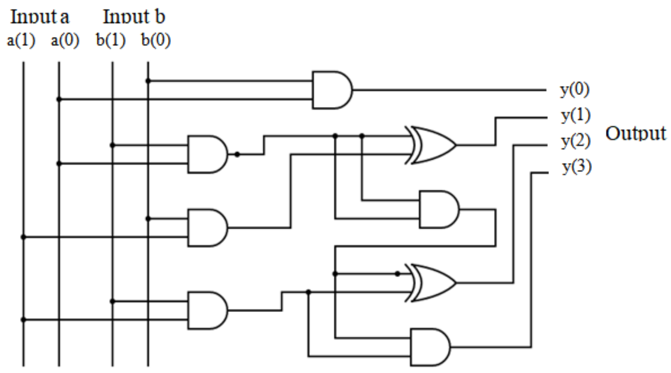

Figure 3. Logical design of 2x2 Vedic Multiplier

The 4x4 Vedic multiplier is designed using commination of 2x2 Vedic multiplier and 4 bit ripple carry adder as shown in figure 4,

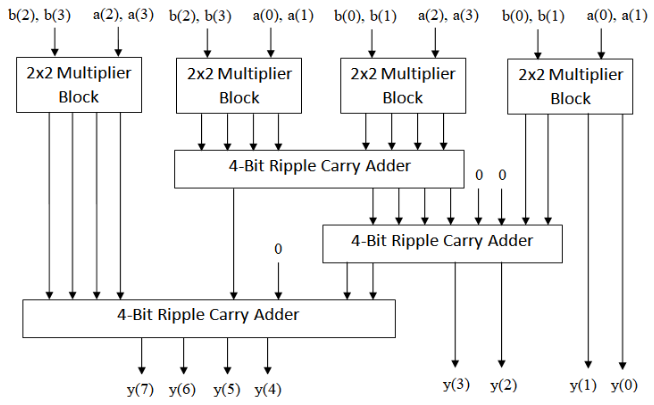

Figure 4. Logical design of 4x4 Vedic Multiplier

Similarly, the 8x8 Vedic multiplier can be designed using combination of 4x4 Vedic multiplier and 8 bit ripple carry adder as shown in figure 5,

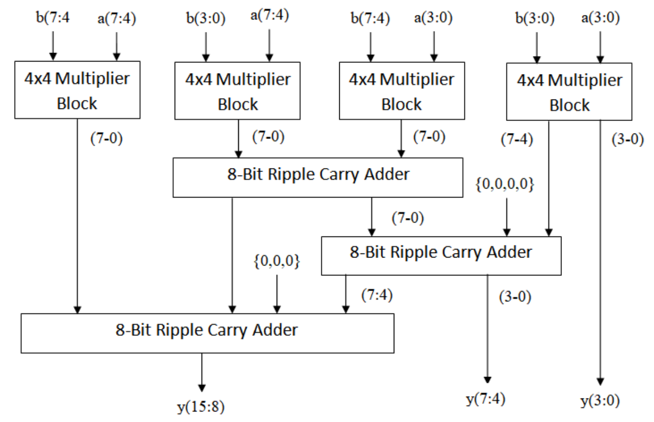

Figure 5. Logical design of 8x8 Vedic Multiplier

The comparison of device utilization between 8-bit signed simple multiplier and Vedic multiplier is shown in table 1.

Table 1. Design summery of Vedic multiplier

Logic Utilization	Using Simple Multiplier	Using Vedic Multiplier	Available
Number of Slice LUTs	8	67	5720
Number of fully used LUT-FF pairs	0	0	137
Number of bonded IOBs	25	25	102
Number of DSP48A1s	**1**	**0**	**16**

Subtract and Increment Divider

In case of divider operation, the traditional divider module uses 4266 LUTs to calculate quotient and reminder for 32 bit signed number. This LUT utilization consumes almost 74% LUT available in Spartan 6 FPGA. To reduce the utilization of LUT and share the other resources available in FPGA (i.e. Slice registers, LUT-FF pair), It is necessary to design the customized divider module. To share the other resources present in FPGA, subtract and increment method is used. The algorithm used for divide signed number is shown below,

Algorithm 1 Subtract and Increment Divider

```
1: Start,
2: Load inputs vectors,
3: if dividend and divisor have same sign then
4:     if dividend and divisor are negative then
5:         Convert dividend and divisor to positive no.
           using 2's complement method,
6:             set sign flag,
7:         else
8:             reset sign flag,
9:     end if
10:    Substitute value of dividend into reminder,
11:    while reminder > divisor do
12:        reminder = reminder – divisor,
13:        quotient = quotient + 1,
14:    end while
15: else
16     reset sign bit,
17:    Substitute value of dividend into reminder,
18:    while reminder and divisor have different sign do
19:        reminder = reminder + divisor,
20:        quotient = quotient – 1,
21:    end while
22: end if
23: if sign bit is set then
24:     Convert reminder into negative no. using 2's
        complement method,
25: end if
26: Update value of quotient and reminder,
27: Stop.
```

To design system on FPGA, it is necessary to use customized number of bits operation. The data bits can be vary from 4-bit up to 32-bit. The utilization of resources can be increased tremendously as number of bits increases. Figure 6, 7 and 8 shows the graphical representation of comparison between traditional method utilization and proposed method utilization. Figure 6 shows the utilization of Slices LUTs. Figure 7 shows the utilization of Slices Registers. Figure 8 shows the utilization of fully used LUTs-FF pairs.

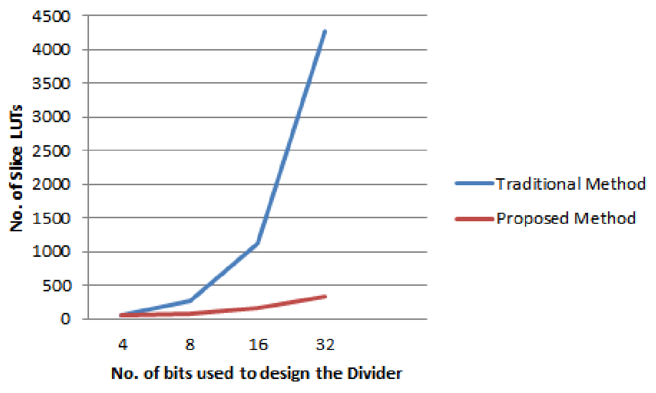

Figure 6. Comparison between Slice LUTs utilization of traditional method and proposed method

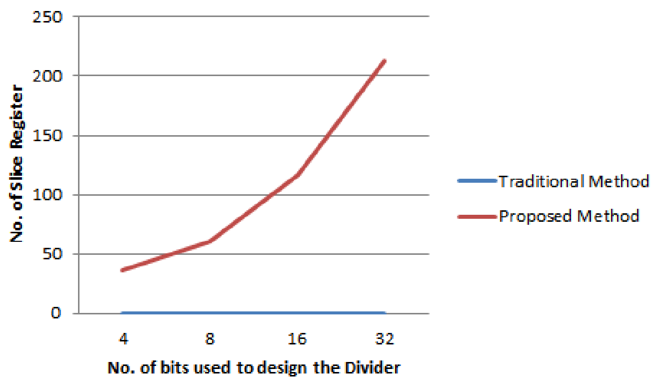

Figure 7. Comparison between Slice Registers utilization of traditional method and proposed method

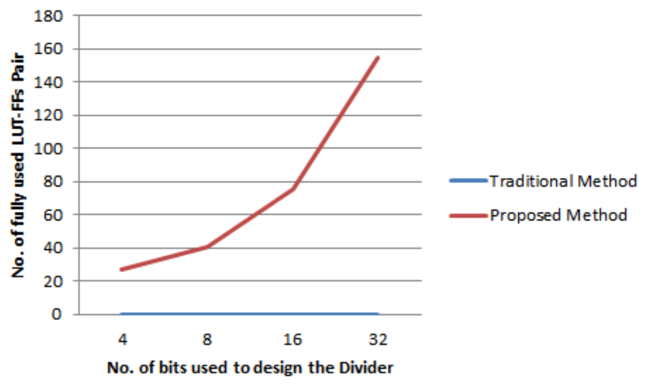

Figure 8. Comparison between LUT-FF pairs utilization of traditional method and proposed method

The comparison of device utilization between 8-bit signed simple divider used in proposed system is shown in table 2.

Table 2. Design summery of subtract and increment divider

Logic Utilization	Using Traditional Divider	Using Proposed Divider	Available
Number of Slice Registers	0	52	11440
Number of Slice LUTs	276	67	5720
Number of fully used LUT-FF pairs	0	48	137
Number of bonded IOBs	32	32	102

RESULTS AND DISCUSSION

Software Simulation

To Implement and synthesis the artificial neuron, XILINX ISE Design Suit 14.5 is used. The RTL design of the structure of artificial neuron connected to external logical OR gate is shown in figure 9.

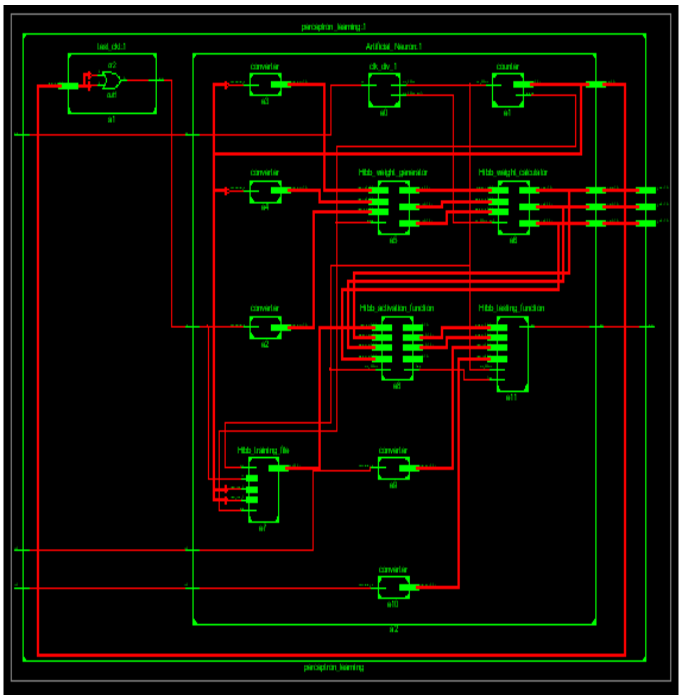

Figure 9. RTL design for Artificial Neuron

The simulation diagram for weight generation is shown in figure 10. The weight calculated for logical OR gate is represented by weight_w1, weight_w2 and weight_wb.

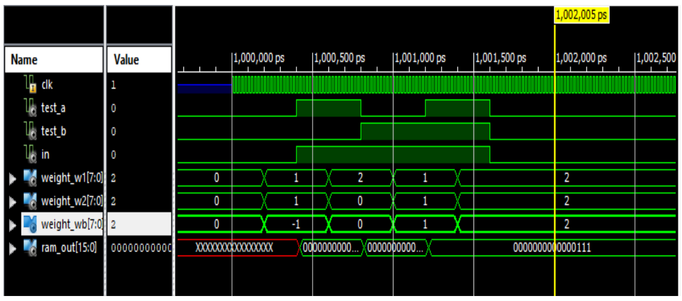

Figure 10. Simulation results of weight generation

Hardware Implementation

The proposed system is implemented using Spartan 6 FPGA developed by Xilinx family. To synthesis the artificial neuron on hardware, Spartan 6 family, Xilinx X-SP6-X9 development board is used. This system is operating on 50MHz clock frequency. The hardware implementation of ANN using FPGA is shown in figure 11.

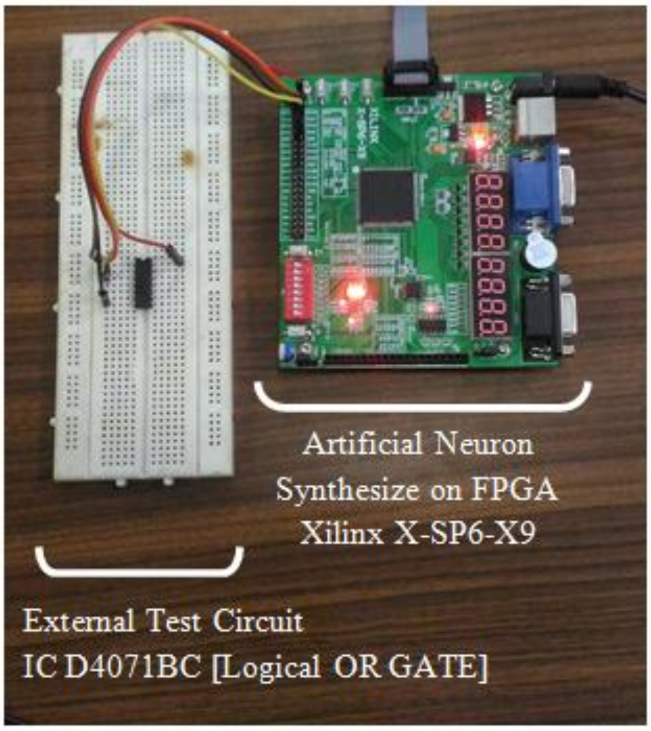

Figure 11. Hardware Implementation of Proposed System

Here, external test circuit applied to the Artificial Neuron synthesized in FPGA is IC D4071BC represents Logical OR GATE in CMOS Family. As testing input pattern applied to the system, the output of the Artificial Neuron is similar to the logic behaviour of External Test Circuit. The output of Artificial Neuron connected to IC D4071BC which represents Logical OR gate is shown in figure 12.

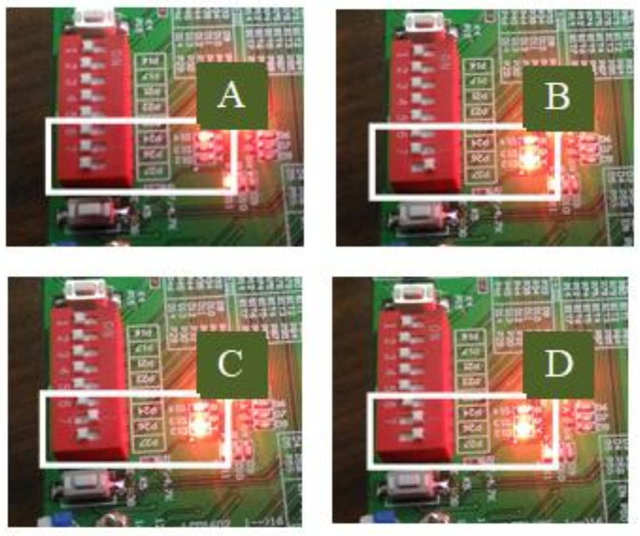

Figure 12. Output of artificial neuron connected to the external test circuit which is Logical OR GATE. Push Button 7 and 8 represents testing input and LED D12 represents the output of system. [A] Input 'LOW,LOW' Output 'LOW',[B] Input 'LOW,HIGH' Output 'HIGH',[C] Input 'HIGH,LOW' Output 'HIGH',[D] Input 'HIGH,HIGH' Output 'HIGH'

Optimization Results

The objective of the optimizing techniques is to reduce the RTL logic design of the proposed system. Table 3 shows the utilization summery of proposed system without optimization and using optimized multiplier and divider.

Table 3. Design Summery of Proposed system

Logic Utilization	**Without Optimization**	**Using Optimization**	**Available**
Number of Slice Registers	83	127	11440
Number of Slice LUTs	**546**	**261**	**5720**
Number of fully used LUT-FF pairs	40	102	137
Number of bonded IOBs	7	7	102
Number of DSP48A1s	**2**	**0**	**16**

The existing system described in [3] uses SoC architecture. In existing system, weight generation is performed by FPGA and activation function is calculated using ARM controller. The device utilization of existing weight generation unit and proposed system weight generation unit is shown in table 4. To calculate activation function, proposed system uses different method and architecture compare to existing system. Due to this, the results cannot be compared.

Table 4. Design Summery of weight generator

Logic Utilization	**Using existing system describe in [3]**	**Using proposed system**	**Available**
Number of Slice LUTs	24	137	5720
Number of fully used LUT-FF pairs	0	0	137
Number of bonded IOBs	49	49	102
Number of DSP48A1s	**2**	**0**	**16**

CONCLUSION

This paper represents the optimized mathematical modelling of digital circuit using Artificial Neural Network on FPGA. Single layer perceptron is designed and implemented using a supervised training method. Hebbian weight calculation method is used to generate the weights. Linear plane and line equation are used for precision and activation function. The process of optimization aims to reduce the RTL logic design required to synthesize the neural network algorithms and eliminate the unwanted use of DSP48A1S synthesize by the compiler and share the available resources. Further advancement may be carried out to utilize the mathematical model on the application level. The application case for the proposed system can be extended to design control system for external circuit and fault diagnosis of the external analog and digital circuits.

ACKNOWLEDGEMENT

This work was partially supported by Department of Electronics and Telecommunication Engineering in MITCOE, Pune.

REFERENCES

[1] Ravikant G. Biradar, Abhishek Chatterjee, Prabhakar Mishra, Koshy George, " FPGA Implementation of a Multilayer Artificial Neural Network using System-on-Chip Design Methodology", lEEE, 2015.

[2] A. Gomperts, A. Ukil and F. Zurfluh, "Development and implementation of parameterized FPGA-based general purpose neural networks for online applications," IEEE Trans. Ind. Inf., vol. 7, no. 1, pp. 78–89, Feb. 2011.

[3] K. Subramanian, S. Suresh, and N. Sundararajan, "A metacognitive neuro– fuzzy inference system (McFIS) for sequential classification problems," IEEE Trans. Fuzzy Syst., vol. 21, no. 6, pp. 1080–1095, Dec. 2013.

[4] G. S. Babu and S. Suresh, "Sequential projection-based metacognitive learning in a radial basis function

network for classification problems," IEEE Transactions on Neural Networks and Learning Systems, vol. 24, no. 2, pp. 194 –206, 2013.

[5] G. Alizadeh, J. Frounchi, M. B. Nia, M. H. Zariff, and S. Asgarifar, "An FPGA implementation of an artificial neural network for prediction of cetane number," in Proc. Int. Conf. Comp. Comm. Eng., 2008, pp. 605–608.

[6] A. Tisan, M. Cirstea, S. Oniga, and A. Buchman, "Artificial olfaction system with hardware on-chip learning neural networks," in Proc. IEEE Int. Conf. Optimization of Electrical and Electronic Equipment (OPTIM 2010), Brasov, Romania, May 20–22, 2010, pp. 884–889.

[7] R. Gadea, J. Cerda, F. Ballester, and A. Mocholi, "Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation," in Proc. 13th Int. Symp. Syst. Synthesis, 2000, pp. 225–230.

[8] T. Orlowska-Kowalska and M. Kaminski, "FPGA implementation of the multilayer neural network for the speed estimation of the twomass drive system," IEEE Trans. Ind. Electron., vol. 7, no. 3, pp. 436–445, Aug. 2011.

[9] H. Hikawa, "Frequency-based multilayer neural network with on-chip learning and enhanced neuron characteristics," IEEE Trans. Neural Netw., vol. 10, no. 3, pp. 545–553, May 1999.

[10] T. Lehmann, "Classical conditioning with pulsed integrated neural networks: Circuits and systems," IEEE Trans. Circuits Syst. II, vol. 45, pp. 720–728, June 1998.

[11] R.J Aliaga, R Gadea, R.J. Colom, J.M. Monzo, C.W. Lerche, J.D. Martinez, A. Sebastia, F. Mateo, "Multiprocessor SoC implementation of neural network training on FPGA," in Proc. IEEE Int. Conf. on Advances in Electronics and Micro-electronics, pp.14, Sept. 29 2008-Oct. 4 2008.

[12] M. Akila, C. Gowribala, S. Maflin Shaby, " Implementation of High-Speed Vedic Multiplier using Modified Adder," International Conference on Communication and Signal Processing, April 6-8, 2016, India.