An Efficient Hardwired Realization of Embedded Neural Controller on System-On-Programmable-Chip (SOPC)

Karan Goel, Uday Arun, A. K. Sinha

ABES Engineering College, Department of Computer Science & Engineering, Ghaziabad
Membership, IEEE Circuit and System Society and ACM Embedded System Computing Society
Membership, IEEE Computer Society

Abstract

In this paper, an efficient hardwired realization of Embedded Neural Controller (ENC) on Field Programmable Gate Array (FPGA/SOPC) is presented. This system architecture is described using Very High Speed Integrated Circuits Hardware Description Language (VHDL) and implemented on Xilinx FPGA/SOPC chip. A design is verified on a Xilinx FPGA/SOPC demo board. This paper discusses the issues involved in the implementation of a multi-input neuron with nonlinear activation function using FPGA/SOPC. The issues are parallel/sequential implementation of neurons, data representation and precision for the inputs, weights & activation function and the use of hybrid method (PLAN with RALUT approximation) for the approximation of a nonlinear, continuous, monotonically increasing, differentiable activation function. It is shown that fully parallel implementation of neurons is not feasible in hardware rather an alternative method called MAC circuit is used to overcome this problem. Moreover, fixed point (FXP) representation is proved to be better than floating point (FLP) representation. Finally, hardware realization of activation function is presented. The results are analysed in terms of low values of average error & maximum error, maximum working frequency, high speed of execution, low power consumption, less delay, less area, higher accuracy, low cost and usage percentage of chip resources such as number of slices (each slice is a logic unit which includes two 4-input look-up tables (LUT) & two flip-flops), number of Input/output blocks (IOB) & number of block RAMs. Here, the multiplier has a parallel, non-pipelined and combinational structure.

Keywords: *ANN, FPGA/SOPC, sigmoid activation function, tangent hyperbolic activation function, PlAN approximation, LUT approximation, RALUT approximation, hybrid approximation method.*

Introduction

Artificial neural networks (ANNs) have been used in many fields such as solving pattern classification & recognition problems, signal processing, control system etc.

Although, neural networks have been implemented mostly in software, hardware versions are gaining importance. Software versions have the advantage of being easy to implement, but with poor performance. Hardware versions are generally more difficult and time consuming to implement, but with better performance than software versions. Hardware version takes the advantage of neural network's inherent parallelism.

Neural networks can be implemented using analog or digital systems. The analog ones are more precise but difficult to implement and have problem with weight storage. The digital implementation is more popular as it has the advantage of higher accuracy, better repeatability, lower noise sensitivity, better testability, higher flexibility, compatibility with other types of processors & has no problem with weight storage.

The digital NN hardware implementation are further classified as DSP Embedded Development board, Microprocessor/Microcontroller based, ASIC-based, VLSI-based, FPGA- based implementations.

Digital Signal Processor (DSP) and microprocessor based implementation is sequential and hence does not preserve the parallel architecture of the neurons in a layer. Application Specific Integrated Circuit (ASIC) & Very Large Scale Integration (VLSI) based implementation do not offer re-configurability by the user, run only specific algorithms, expensive, time consuming to develop such chip and limitation on the size of the network.

IJERT

FPGA is a suitable hardware for neural network implementation as it preserves the parallel architecture of the neurons in a layer and offer flexibility in reconfiguration and maintains high gate density which is needed to utilize the parallel computation in an ANN. Parallelism, modularity and dynamic adaptation are three computational characteristics typically associated with ANNs. Therefore, FPGA/SOPC based implementation is best suited to implement ANNs.

FPGA/SOPC are a family of programmable device based on an array of configurable logic blocks (CLBs) which gives a great flexibility in prototyping, designing and development of complex hardware real time systems.

When implementing ANN on FPGA/SOPC, certain measures are to be taken to minimize the hardware. These measures are the data representation and the number of bits (precision) of inputs, weights and the activation function.

1. Generic Architecture of Multilayer Neural Network (MLP)

The Generic architecture of multilayer neural network shown in figure 1 consists of a layer of input nodes called input layer, one or more hidden layer(s) consisting of hidden nodes and one output layer consisting of output nodes.

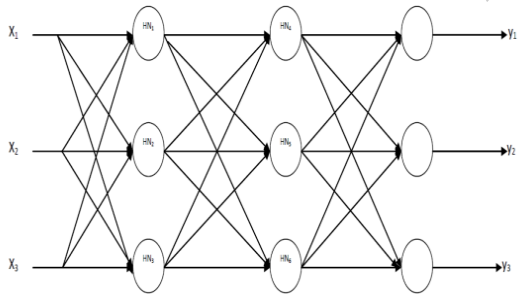


Figure 1. Generic architecture of MLP

Where

x1, x2 & x3 are the inputs.

HN1 to HN6 are the hidden nodes that constitutes a hidden layer

Y1, Y2, Y3 are the outputs of the network.

1.1. NN Implementation with only One Input for the Weights

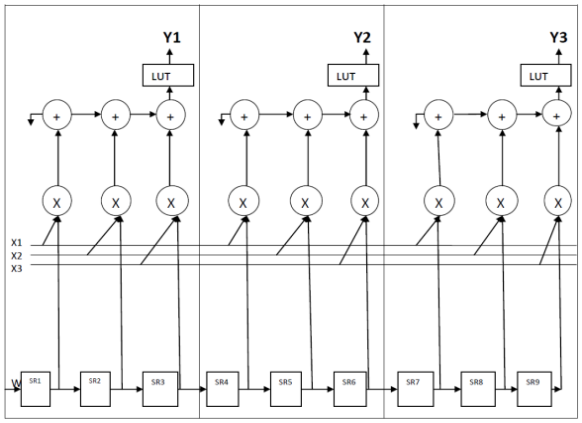
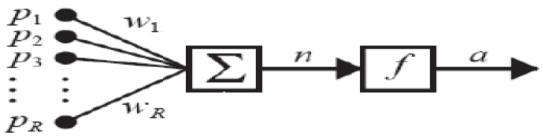


Figure 2. NN implementation with only one input for the weights

2. Mathematical Model of a Single Artificial Neuron

The common mathematical model of a single artificial neuron is shown as



The neuron output can be written as:

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

Where p_j is the input value and w_j is the corresponding weight value, a is the output of the neuron and the f(.) is a nonlinear activation function.

3. Data Precision & Representation

Before beginning a hardware implementation of an ANN, an arithmetic representation (floating point or fixed point) and the number of bits (precision) must be considered for the inputs, weights and activation function, which may range from 16 bit, 32 bit or 64 bit floating-point format (FLP) or fixed point format (FXP). Increasing the precision of the design elements significantly increases the resources used.

IJERT

FLP format is area hungry with improved precision whereas FXP format needs minimal resources at the expense of precision. The minimum allowable precision and minimum allowable range is to be decided to conserve silicon area and power without affecting the performance of ANN. Precision of fixed point data is independent of the number but dependent on the position of the radix point. Antony W.Savich et al. [9] had shown that the resource requirement of networks implemented with FXP arithmetic is appreciably two times lesser than the FLP arithmetic with similar precision and range. Also FXP computation produces better convergence in lesser clock cycles. For a normalized input and output in the range of [0,1] to realize the log-sigmoid activation function, Holt and Bakes [10] showed that 16-bit FXP (1-3-12) representation was the minimum allowable precision. Ligon III et al. [11] exhibited the density advantage of 32-bit FXP computation over FLP computation for Xilinx FPGA/SOPC.

3.1. Analogy of Fixed Point Format

The Fixed point format is defined as follows:

[s] a.b

Where the optional s denotes a sign bit with 0 for positive and 1 for negative numbers, a is the number of integer bits and b is the number of fractional bits.

To represent negative numbers, two's complement notation is used. The examples in Table 1 clarify the notation.

If the sign bit is used, the minimum x_{min} and maximum x_{max} numbers in a.b notation are

$$x_{min} = 2^{-a}$$
$$x_{max} = 2^{a} - 2^{-b}$$

If the sign bit is not used, the minimum x_{min} and maximum x_{max} numbers in a.b notation are

$$x_{min} = 0$$
$$x_{max} = 2^{a} - 2^{-b}$$

Table 1: Examples of binary notation

Format	Example binary number	Decimal number
*s*3.5	011001100	6.375
*s*2.5	11001001	−1.71875
0.8	11001001	0.78515625

4. Implementation of Fully Parallel Neural Network

A fully parallel network is fast but inflexible. In a fully parallel network, the number of multipliers per neuron must be equal to the number of connection to this neuron. Since all of the products must be summed, the number of full adders equals to the number of connections to the previous layer minus one. In 3-5-1 architecture, the output neuron must have 5 multipliers and 4 Full-Adders, while each hidden neuron must have 3 multipliers and 2 adders. So, different neuron architectures have to be designed for each layer. A second drawback of a fully parallel network is gate resource usage because multipliers are costly hardware. Krips et. Al. [1] proposed such architecture.

IJERT

5. Design and Modelling of Multiply and Accumulate (MAC) Circuit of a Neuron

In this work, we chose multiply and accumulate structure for neurons. In this structure, there is one multiplier and one accumulator per neuron. The inputs from previous layer neurons enter the neuron serially and are multiplied with their corresponding weights. Every neuron has its own weight storage ROM. Multiplied values are summed in an accumulator. The processes are synchronized to clock (global) signal. The number of clock cycles for a neuron to finish its work is equals to the number of connections from the previous layer. The accumulator has a load signal, so that the bias values are loaded to all neurons at start-up. This neuron architecture is shown in figure 3. In this design, the neuron architecture is fixed throughout the network and is not dependent on the number of connection.

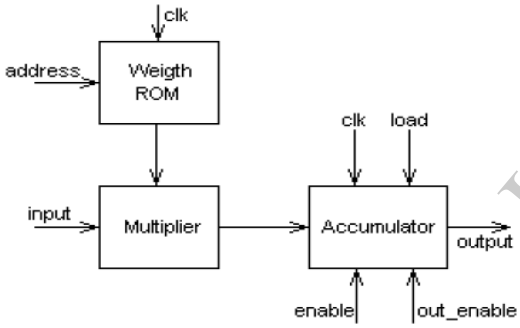


Figure 3. Block diagram of a single neuron

The precision of the weights and input values are both 8 bits. The multiplier is an 8-bit by 8-bit multiplier, which results in a 16-bit product, and the accumulator is 16-bits wide. The accumulator has also an enable signal, which enables the accumulating function and an out_enable signal for its three state outputs. Here the multiplier has a parallel, non-pipelined, combinational structure, generated by Xilinx Logicore Multiplier Generator V 5.0[4].

6. Analysis & Synthesis of Layered Architecture of Multi-Neurons

In our design, an ANN layer has one input which is connected to all neurons in this layer. But previous layer may have several outputs depending on the number of neurons it has. Each input to the layer coming from the previous layer is fed successively at each clock cycle. All of the neurons in the layer operate parallel. They take an input from their common input line, multiply it with the corresponding weight from their weight ROM and accumulate the product. If the previous layer has 3 neurons, present layer take and processes these inputs in 3 clock cycles. After these 3 clock cycles, every neuron in the layer has its net values ready. Then the layer starts to transfer these values to its output one by one for the next layer to take them successively by enabling corresponding neuron's three-state output. Since, only one neuron's output have to be present at the layer's output at a time, instead of implementing an activation function for each neuron, it is convenient to implement one activation function for each layer. The block diagram of a layer architecture including 3 neurons is shown in figure 4.

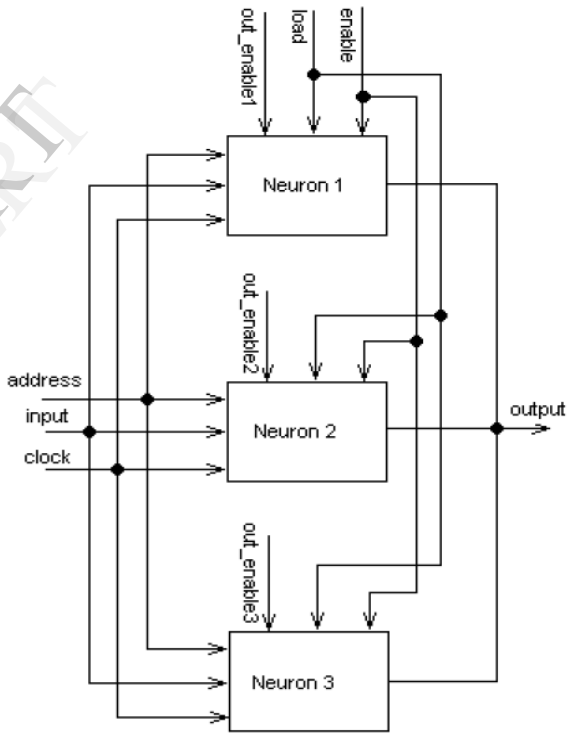


Figure 4. Block diagram of a layer consisting of 3 neurons

7. Design, Modelling, Synthesis of Neural Network Architecture

The control signals in the system are generated by a state machine. This state machine is responsible of controlling all of the operations in the network. First it activates the load signals of the neurons and

IJERT

the neurons load their bias values. Then hidden layer is enabled for 3 clock cycles, and then the output layer consisting of a single neuron is enabled for 5 clock cycles. Out_enable signals are also activated by this state machine. The state machine generates weight ROM addresses in a priory determined sequence so that the same address lines can be used by all of the neurons in the system. Input RAM also uses this address line. Once the input RAM is loaded by input values using the switches on the board, the propagation phase starts and the output of the network is displayed. The block diagram of the network is shown in figure 5. The resource usage of the selected network architecture is shown in table 2.

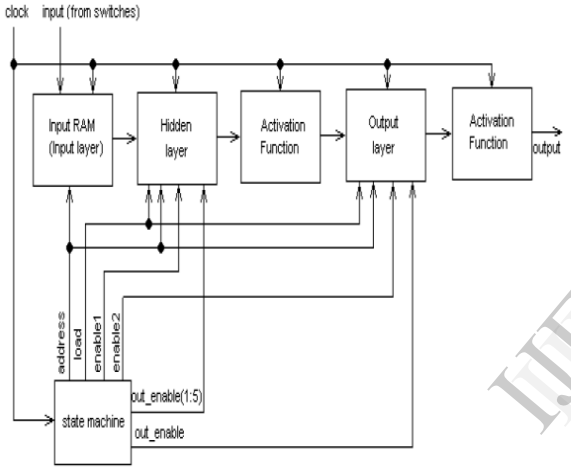


Figure 5. Block diagram of the complete neural network

8. Hardware Realization of Activation Function

One of the important components in designing ANNs in hardware is the nonlinear activation function. The most common used function is sigmoid activation function and tangent hyperbolic activation function. These activation functions is a monotonically increasing, continuous, differential and nonlinear exponential function that limits the output of a neuron within the certain range i.e. [0,1] & [-1, 1] respectively. These functions are reported to be a better activation function for ANN. It is defined as:

$$Sigmoid(x) = \frac{1}{1+e^{-x}}$$

$$Tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

and is shown by figure 6 .

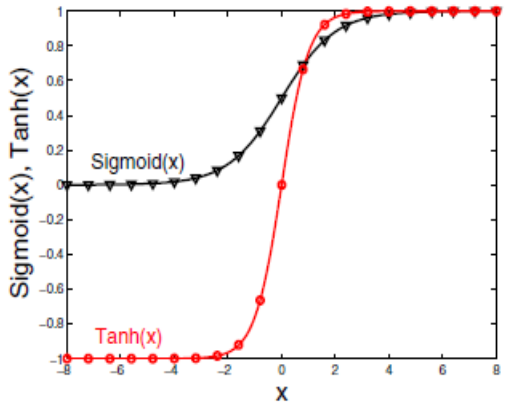


Figure 6. Log-sigmoid and hyperbolic tangent AF

Straightforward implementation of these functions in hardware is not practical due to their exponential nature. Several different approaches exist for the hardware approximation of activation function, including piecewise linear approximation, look-up tables (LUT), Range addressable look-up tables and Hybrid Methods [2],[3],[4],[5],[6],[7],[8]. The efficiency criteria for a successful approximation are the achieved accuracy, speed and area resources.

8.1. Analysis of Piece-wise Linear Approximation of a Nonlinear Function (PLAN)

Piecewise linear (PWL) approximations uses a series of linear segments to approximate the activation function [2]. Piecewise linear approximation is a method to obtain low values for both maximum and average error with low computational complexity. The number and locations of these segments are chosen such that error, processing time and area utilization are minimized. The number of pieces required in the PWL approximation of the activation function depends upon the complexity of the problem to be solved [12]. Using power of two co-efficient to represent the slope of the linear segments allows right shift operations instead of multiplication [12] to reduce the hardware as multipliers are expensive hardware components in terms of area and delay. Piecewise linear approximations are slow, but consume less area. For hardware implementation, it is convenient that the number of data points be a power of two, we will assume that the interval I is divided into 2^k intervals:

IJERT

[L, L + (U-L)/ 2^k), [L + (U-L)/ 2^k, L + 2(U-L)/2^k) ....., [L + 2^{k-1}(U-L)/2^k)

Where U and L are the upper and lower limit of inputs applied to the activation function respectively. Then, given an argument x, the interval into which it falls can readily be located by using, as an address, the k most significant bits of the binary representation of x. A piecewise linear approximation of the hyperbolic tangent function with five segments is shown in figure 7.

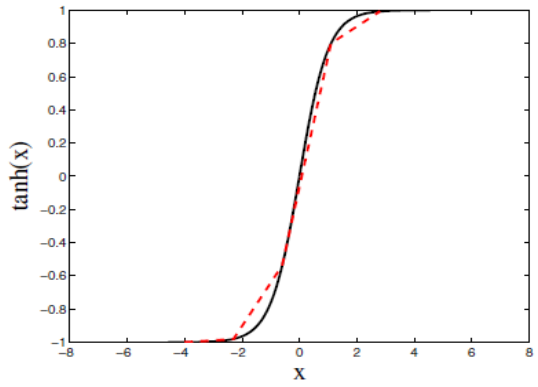


Figure 7. Piecewise linear approximation of Tanh(x) with five segments

The representation of five segments is shown as:

$$f(x) = \begin{cases} 0, & \text{(region 1) if } x \le -8 \\ \frac{8-|x|}{64}, & \text{(region 2) if } -8 < x \le -1.6 \\ \frac{x}{4} + 0.5, & \text{(region 3) if } |x| < 1.6 \\ 1 - \frac{8-|x|}{64}, & \text{(region 4) if } 1.6 \le x < 8 \\ 1, & \text{(region 5) if } x > 8. \end{cases}$$

The computational modules of the activation function is as shown in figure 8.

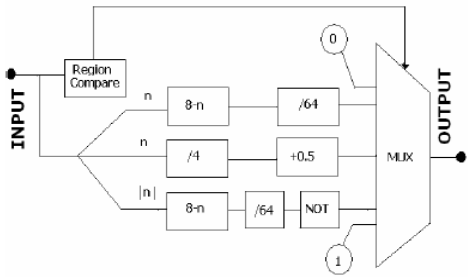


Figure 8. Computational modules of the activation function

Individual sub blocks are realized using FXP formats.

8-n : Inverter module
/n : constant bit shifts
Mod n : Subtraction
+ 0.5 : Constant adder

Simulated results of PWL based approach is shown in table 4.

8.2. Analysis of LUT Approximation

With this approach, the function is approximated by a limited number of uniformly distributed points [5], [7]. This methods results in high performance design, however it will require a large amount of memory to store the look up table (LUT) in hardware. The look-up table's input space is designed so that it covers a range between -8 and +8. In our data representation, this range corresponds to s4.5 format ie.. 10 bits. So, a 1024*8 ROM is needed to implement the activation function's look-up table. The LUT design does not optimize well under FLP format and hence fixed point format is used. But still on-chip realization of log-sigmoid function increases the size of hardware considerably. To optimize the area to some extent, the inbuilt RAM available in FPGA/SOPC is used to realize LUT based activation function. It reduces area and improves speed. If the precision is to improve, then PWL approximation approach of activation function is to be utilized along with the look-up table (Hybrid Method). The look-up table approximation of the sigmoid function with sixteen points is shown in figure 9.

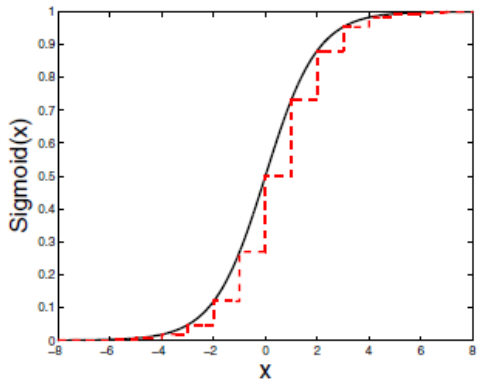
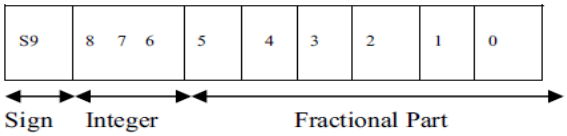


Figure 9. LUT approximation of sigmoid function with sixteen points

The log sigmoid activation function is realized by representing input x in 10 bits and output f(x) in 8 bits. 10-bit input x has the following FXP format to represent the address of the memory location and output f(x) is represented as 8-bit fractional part only, to indicate the respective content of memory location. The data precision of the above two approaches (PWL and LUT approximation) of AF

IJERT

is shown in table 3. Simulated results of LUT based approach is shown in table 6. Comparison of Synthesis report for LUT and PWL approximation using Xilinx is shown in table 5.



The smallest possible value that can be represented as output is 2^{-n}, where n is the number of bits in the stream. This value should be treated as zero, since f(x)=0 when x=minus infinity, which may never occur. Hence an error 2^{-n} is introduced. Similarly the largest possible value of f(x) is $1-2^{-n}$ to limit the output to be within (0, 1).

Therefore, the minimum value of AF,

$$f(x1) = 1 / (1+e^{-x1}) = 2^{-n}$$

which decides the lower limit of input,

$$x1 = -ln (2^n - 1)$$

Similarly, the maximum vale of AF,

$$f(x2) = 1 / (1 + e\text{-}x2) = 1\text{-}2^{-n}$$

which decides the upper limit of input,

$$x2 = ln (2n - 1)$$

Hence, the step size or increment of input is found to be

$$X = ln [(0.5 + 2^{-n}) / (0.5 - 2^{-n})]$$

The minimum number of LUT values is given by

$$min\ number = (x2\text{-}x1) / X$$

min number for the x1 and x2 respectively is found to be 710 for 8-bit precision. Hence 1K RAM with 10-bit address lines is chosen that gives 1024 locations with step size X = 0.015625. For a 10 bit input x, the minimum value x1 = - 5.5413 and the maximum value is x2 = 5.5371. The output RAM has 1K memory in which 710 locations are used leaving 374 locations unused. Wastage of memory is a draw back and the second method (PWL) overcomes it.

8.3. RALUT approximation

It shares many aspects with the classic LUT with a few notable differences. In LUTs, every data point stored in the table corresponds to a unique address. In RALUTs, every data point corresponds to a range of addresses. This alternate addressing allows for a large reduction in data points in situations where the output remains constant over a range. Examples are the hyperbolic tangent and the sigmoid functions, where the output changes only slightly outside the range of (-2, 2). A RALUT approximation of the hyperbolic tangent function with seven points is shown in figure 10.

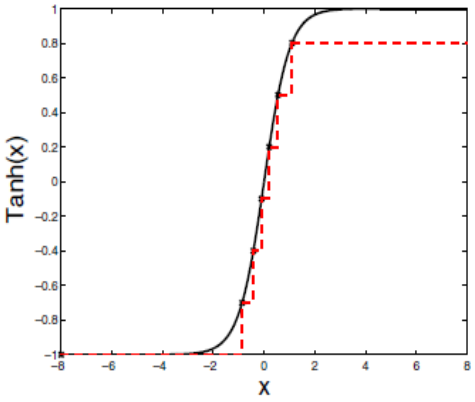


Figure 10. RALUT approximation of hyperbolic tangent function

8.4. Analysis of Hybrid Method Approximation

Hybrid methods usually combine two or more of the previously mentioned methods to achieve better performance. This method makes use of piecewise linear approximation along with the use of RALUT [8]. It initially makes use of a piecewise linear approximation with five segments to approximate the activation function.

Afterwards, the difference between the actual function's value and the approximated value will be adjusted to achieve a better approximation. The adjustment values are calculated offline and stored in a lookup table. These adjustments consists of subtracting the values stored in the lookup table from the initial piecewise linear approximation. This can be achieved with a simple combinational subtracter circuit.

This method has two main advantages. First, all the five segments were chosen to avoid the use of a multiplier in the design. Second, the lookup table used in our design stores the difference between the piecewise linear approximation and the actual function. This greatly minimizes the range of the output variance, which reduces the size of the lookup table. To further optimize the design, RALUT was used instead of classic LUT. The block diagram of the hybrid system is shown in figure 11. Complexity comparison of different implementation is shown in table 7.

IJERT

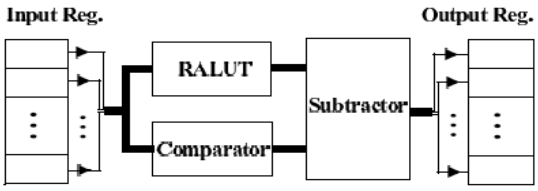


Figure 11. System block for a hybrid method

9. Experimental results

Table 2. Resource usage of the network architecture

Number of Slices	91
Number of Slice Flip-Flops	151
Number of 4-input LUTs	161
Number of Block RAMs	4

Table 3. Data precision of LUT and PWL Activation function

	LUT based AF		PWL approximation AF	
Data	FXP format	No. of bits	FXP format	No. of bits
Input	1 - 3 - 6	10	1 - 4 - 8	13
output	0 - 0 - 8	8	0 - 4 - 8	12

Table 4. Simulated results of PWL based approach

Input x value	Region	Binary Encoding Of input values	Output f(x) value	Binary Encoding Of the output values
-8.123	1	1100000011111	0	00000000000
-7 to -1.6	2	1011100000000	0.015625	00000000100
		to	to	to
		1000110011001	0.1	00000011001
-1.5 to 1.5	3	1000110000000	0.125	00000100000
		to	to	to
		0000110000000	0.875	00001100000
1.6 to 7	4	0000110011001	0.9	00001100111
		to	to	to
		0011100000000	0.984375	00001111100
8.123	5	0100000011111	1	00100000000

Table 5. Comparison of Synthesis report for LUT and PWL approximation using Xilinx

Components/modules	Used	Used	Available
Type	LUT-Approach	PWL-Approximation	
Number of 4-input LUTs	187	108	12288
Number of slices	102	58	6144
Number of Bounded IOBs	19	26	320
Maximum Delay(ns) (Critical path)	1.934	1.834	-----
Total Gate Count for design	1348	1029	-----

Table 5. Simulated results of LUT based approach

Input Range	Binary Address corresponding to input Range	Output Range	Binary code of Output
-5.5413 to -4.3069	1101100010 (866) 1100010011 (787)	0.0039063 to 0.013297	00000001 00000011
-4.2912 to -4.01	1100010010 (786) 1100000000 (768)	0.013503 to 0.017811	00000011 00000100
-3.9944 to -3.2287	1011111111 (767) 1011001110 (718)	0.018086 to 0.038099	00000100 00001001
-3.2131 to -2.009	1011001101 (717) 1010000000 (640)	0.038676 to 0.11816	00001001 00011110
-1.9943 to 1.9901	1001111111 (639) 0001111111 (127)	0.1198 to 0.87976	00011110 11100001
2.0058 to 3.2089	0010000000 (128) 0011001101 (205)	0.8814 to 0.96117	11100001 11110110
3.2245 to 3.9902	0011001110 (206) 0011111111 (255)	0.96175 to 0.98184	11110110 11111011
4.0058 to 4.2871	0100000000 (256) 0100010010 (274)	0.98212 to 0.98644	11111011 11111100
4.3027 to 5.5371	0100010011 (275) 0101100010 (354)	0.98665 to 0.99688	11111100 11111111

Table 7. Complexity comparison of different approximation method of activation function in terms of area and delay

Architectures	Function	Max-Error	AVG-Error	Area	Delay	Area × Delay
LUT	Tanh	0.5%	0.050%	37647.4 μm^2	2.61 ns	98.26×10^{-12}
RALUT			0.171%	27194.7 μm^2	2.26 ns	61.46×10^{-12}
Hybrid			0.203%	12400.1 μm^2	3.03 ns	37.57×10^{-12}
LUT		2%	0.200%	9045.9 μm^2	2.15 ns	73.44×10^{-12}
RALUT			0.446%	7090.4 μm^2	1.85 ns	13.11×10^{-12}
Hybrid			1.230%	3646.8 μm^2	2.31 ns	8.42×10^{-12}
LUT	Sigmoid	0.5%	0.098%	34159.1 μm^2	2.43 ns	83.01×10^{-12}
RALUT			0.485%	24385.5 μm^2	2.26 ns	55.11×10^{-12}
Hybrid			0.260%	9273.6 μm^2	3.01 ns	27.91×10^{-12}
LUT		2%	0.392%	8911.8 μm^2	2.15 ns	19.16×10^{-12}
RALUT			1.846%	4480.3 μm^2	2.12 ns	9.498×10^{-12}
Hybrid			1.060%	3004.5 μm^2	2.36 ns	7.091×10^{-12}

10. Conclusion

This work has presented the analysis, modeling, synthesis, verification, prototyping and implementation of embedded neural controller for high intelligent control systems on embedded SOPC & SPARTAN6 FPGA device. The proposed network architecture is modular, being possible to easily increase or decrease the number of neurons as well as layers. SOPC can be used for portable, modular, and reconfigurable hardwired solutions

IJERT

for neural networks, which have been mostly used to be realized on computers until now. With the internal layer parallelism we used, it takes lesser clock cycles for the applied network to calculate its output as compared to microprocessor based implementation. Because neural networks are inherently parallel structures, parallel architectures always result faster than serial ones. Moreover, hardware implementation of an activation function that is used to realize an ANN is tried in this work. A two-fold approach to minimize the hardware is proposed here, aiming at the SOPC realization. Firstly, FXP arithmetic is followed against FLP arithmetic to reduce area, by compromising slightly on accuracy. Secondly, PLAN approximation with five linear segments along with RALUT is proved to be hardware friendly.

11. Future work

Future research work includes the analysis, modeling, synthesis, verification, prototyping and implementation of embedded neural controller for high intelligent control systems on embedded SOPC & SPARTAN6 FPGA device, analysis & synthesis of hybrid approximation of activation function, pipelined based hybrid multiplier and memory elements for embedded neural controller.

IJERT

12. References

1. M. Krips, T. Lammert, and Anton Kummert, "FPGA Implementation of a Neural Network for a Real-Time Hand Tracking System", Proceedings of the first IEEE International Workshop on Electronic Design, Test and Applications, 2002.
2. K. Basterretxea and J.M. Tarela and I. Del Campo, "Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons", IEE Proceedings - Circuits, Devices and Systems, vol. 151, no. 1, pp. 18-24, February 2004.
3. K. Basterretxea and J.M. Tarela, "Approximation of sigmoid function and the derivative for artificial neurons", advances in neural networks and applications, WSES Press, Athens, pp.397-401, 2001.
4. M.T. Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic", IEE Proceedings on Computers and Digital Techniques, vol. 150, no. 6, pp.403 - 411, Nov. 2003.
5. K. Leboeuf, A.H. Namin, R. Muscedere, H. Wu, and M. Ahmadi "Efficient Hardware Implementation of the Hyperbolic Tangent Sigmoid Function", Third International Conference on Convergence and hybrid Information Technology, accepted, November 2008.
6. H.K. Kwan, "Simple sigmoid-like activation function suitable for digital hardware implementation" Electronic Letters,vol. 28, no. 15, pp. 1379-1380, July 1992 .
7. F. Piazza, A. Uncini, and M. Zenobi, "Neural networks with digital LUT activation functions", Proceedings of 1993 International Joint Conference on Neural Networks, IJCNN, pp. 1401-1404,October 1993.
8. A.H. Namin, K. Leboeuf, R. Muscedere, H. Wu, and M. Ahmadi, "HighSpeed VLSI Implementation of the Hyperbolic Tangent Sigmoid Function ", IEEE International Symposium on Circuits and Systems (ISCAS), accepted, May 2009.
9. Antony W. Savich, Medhat Moussa, and Shawki Areibi, "The Impact of Arithmetic Representation on Implementing MLP-BP on FPGAs: A Study," IEEE Trans. Neural Networks, vol.18, no.1, Jan.2003.
10. J. Holt and T. Baker, "Back propagation simulations using limited precision calculations," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN-91)*, Seattle, WA, Jul. 1991, vol. 2, pp. 121–126.
11. W. Ligon, III, S. McMillan, G. Monn, K. Schoonover, F. Stivers, andK. Underwood, "A re-evaluation of the practicality of floating point operations on FPGAs," in *Proc. IEEE Symp. FPGAs Custom Comput. Mach.*, K. L. Pocek and J. Arnold, Eds., Los Alamitos, CA, 1998, pp.206-215.