Low Area ANN Architecture with Stochastic Computing and a Simplified Sigmoid Function

Huy-Hung Ho*, Xuan-Thuan Nguyen, Van-Thuat Nguyen, Van-Dung Nguyen

SIS Laboratory, VNU University of Engineering and Technology (VNU-UET),

144 Xuan Thuy road, Cau Giay district, Hanoi, Vietnam.

(*) Email: 14020590@vnu.edu.vn

***Abstract*—Artificial Neural Networks (ANNs) is one of the hottest research trends in recent years. Developing ANNs plays as an important key to reach the future of automation. In this work, we study the ANNs and propose a low complexity hardware architecture of ANNs. The proposed architecture is built up thanks to a combination of our simplified Sigmoid function and the Stochastic Computing to reduce the complexity of the addition and multiplication operations. The simulation results on Xilinx FPGA technology show that the simplified Sigmoid reduces 21.18% of area cost compared to the 16-bits LUT methods. In the accuracy side, the mean square error of the simplified Sigmoid is 1.79×10^{-5} , and the 16-bit LUT is 3.12×10^{-7} . The implementing of the Stochastic Computing into ANNs at 2-3-2 model reduces 20.98 % of area cost compared to the LSI reference architecture.**

***Keywords*—Stochastic Computing, neural network, activation function, sigmoid function, ANN architecture.**

I. INTRODUCTION

Artificial Neural Networks (ANNs) [1] are massively parallel systems with large numbers of interconnected simple processors. It is widely used in analog and digital signal processing applications such as hand-written recognition, face detection and recognition, real-time surface discrimination, and so on. However, software implementation of ANNs hardly responses the demand of real-time applications. In this work, we design a hardware model of the ANNs with the parameterization of input coefficients. The size of the designed architecture can be easily modified for specific applications, and the design also enabling errors checking and verification.

Activation function is one of the most important modules in ANNs. Hyperbolic tangent and sigmoid are the most used activation functions. However, the activation function is very complex and hard to be implemented as a hardware architecture. The implementation accuracy of these activation functions in digital networks faces certain challenges. There are many different methods have been proposed to improve the approximation of hyperbolic tangent such as piecewise nonlinear approximation (PWL) [2], lookup table (LUT) [3], separating the input range into different regions [4], etc. But Sigmoid function is rarely mentioned. In this work, a new hybrid architecture, which is based on LUT with dividing the input range into different regions is proposed. It is a simple method to improve the accuracy of sigmoid function.

Besides the activation function, the ANNs consume a large amount of addition and multiplication operators. After investigating multiple optimization techniques to improve upon artificial neural network, we propose to replace the conventional adders and multipliers module by a Stochastic Computing module. The Stochastic Computing (SC) was proposed in the 1960s as a computational technique by Gaines [5], processes data in the form of digitized probabilities. The main motivation of using SC is the simplicity of the involved computational elements. SC has shown promising results for low-power area-efficient hardware implementations. Hence, it has been shown to be useful in image-processing and communication applications.

The remaining part of this paper is organized as follows. In Section II, we introduce the proposed sigmoid function architecture. Next, Stochastic Computing technique is applied in forward module is described in Section III. In Section IV, the implementation and simulation results using FPGA technology are presented and discussed. Finally, Section 5 are some conclusions and perspectives.

II. THE PROPOSED APPROXIMATION OF SIGMOID FUNCTION ARCHITECTURE

The Sigmoid function provides an output range [0; 1] and is defined as follows:

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

By examining the relationship between the output and the input of sigmoid function, we divide the sigmoid function into three different regions to reduce the complexity: linear region; nonlinear region; and constant region. These regions are shown in Fig. 1. Region I, in which the outputs are approximately equal to the linear combination of the input, is named the linear region. In the two far ends of the sigmoid function, because of the low variation of the output, we can represent the outputs as constants in region III, it is named constant region. Region II includes the rest of input ranges, named nonlinear region.

A. Determining the boundaries for different regions

1) Region I: Linear Region

In the linear region, the output is approximated by a linear function. It is obtained as follows:

$$sigmoid(x) = 0.245x + 0.5 \text{ with } x \in [-1;1] \quad (2)$$

2) Region II: Nonlinear Region

In the nonlinear region, we use LUT with $x \in [-3.5;-1) \cup (1;3.5]$.

3) Region III: Constant Region

In the constant region, the sigmoid function reaches its maximum value (approaching 1) with $x \in [4.5;+\infty)$, minimum value (approaching 0) with $x \in (-\infty;-4.5]$. Besides, the sigmoid function reaches approximation value 0.984375 with $x \in [3.5;4.5]$, 0.015625 with $x \in [-4.5;-3.5]$. Two constant values are used in this region which are shown as follows:

$$sigmoid(x) = 0.984375 \text{ with } x \in (3.5;4.5] \quad (3)$$

$$sigmoid(x) = 0.015625 \text{ with } x \in [-4.5;-3.5) \quad (4)$$

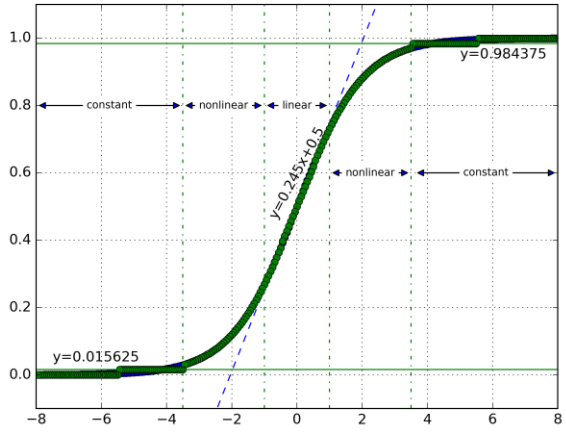

Fig. 1. Different regions of sigmoid function.

B. Approximation of sigmoid function architecture

Our proposed hardware architecture to calculate the sigmoid function is shown in Fig. 2. We only use LUT to calculate with $x \in [-3.5;-1) \cup (1;3.5]$. It only needs two sets of 5-bit LUT and two sets of 3-bit LUT to express 80 values of x in this range instead of 256 values of the basic LUT method. Moreover, the multiplication with 0.245 is approximately replaced by the 2-bit shift-right logic to reduce the hardware complexity.

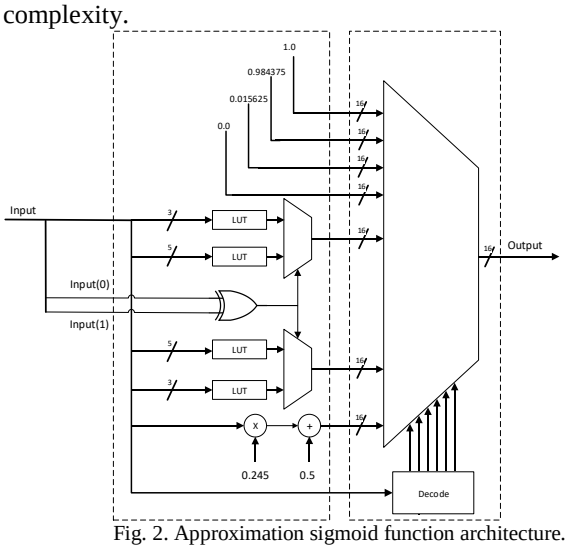

Fig. 2. Approximation sigmoid function architecture.

The weighted inputs pass through the decoder to create 6 selector signals for the multiplexer. The input range will be decoded to generate the selection signal for the multiplexer to choose different values for the sigmoid function. The operating principle of this mux is illustrated in TABLE I.

TABLE I THE OPERATION OF THE DECODER AND NEW SIGMOID FUNCTION MODEL

Weighted input range	Selector	Output
$(4.5;+\infty)$	000000	0.999023
$(3.5;4.5]$	100000	0.984375
$(1;3.5]$	110000	LUT
$(-1;1]$	111000	$a \mid a \in y = 0.245x + 0.5$
$(-3.5;-1]$	111100	LUT
$(-4.5;3.5]$	111110	0.015625
$(-\infty;4.5]$	111111	0.0

III. The Proposed Neural Network using Stochastic Computing for Forward Module

A. Stochastic Computing and its computational elements

In SC, numbers are encoded as bit streams that are interpreted as probabilities. In SC's basic unipolar format, a bit-stream X of length N has the value p_x which is shown as follows:

$$p_X = N_1 / N = x \quad (5)$$

In bipolar format, a bit-stream X of length N has the value p_x which is shown as follows:

$$p_X = (N_1 - N_0) / N = (x+1)/2 \quad (6)$$

where N_1 is the number of 1's in X, N_0 is the number of 0's in X. The positions of the 1's are not prescribed, so many different bit-streams can have the same p_x, and any real number can be represented in one of these two formats by scaling it down to fit within the appropriate interval. A stochastic stream of a real value x is usually generated by a Linear Feedback Shift Register (LFSR) and a comparator.

1) Stochastic number generator (SNG)

Circuits that convert binary numbers to stochastic number (SN), and vice versa, are fundamental elements of SC. Fig. 3 illustrates a widely used 8 bit binary-to-stochastic conversion circuit, which we will refer to as a stochastic number generator. It includes a Linear-Feedback Shift Register (LFSR) and an 8-bit comparator.

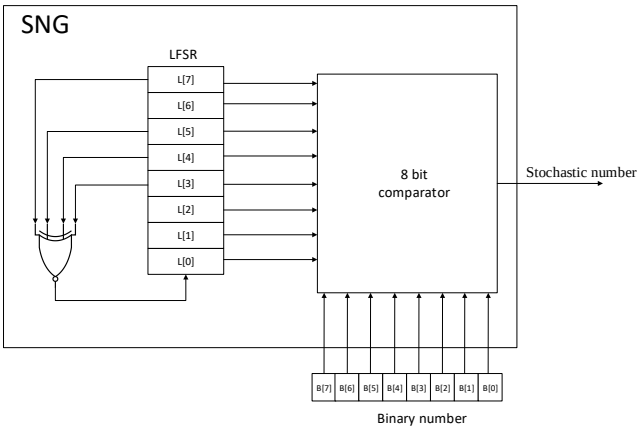

Fig. 3. Stochastic number generator.

The conversion process involves generating a k bit random binary number in each clock cycle by means of a random or, more likely, a pseudo-random number generator, and comparing it to the k bit input binary number B. The comparator produces '1' if the random number is less than B, and '0' otherwise. Assuming that the random numbers are uniformly distributed over the interval [0, 1] and the probability of '1' appearing at the output of the comparator at each clock cycle is equal to $B / 2^k$ [6].

To reduce inaccuracies in calculation, SNGs are needed which produce SNs that are sufficiently random and uncorrelated. Using LFSRs is a suitable choice because they can generate multiple uncorrelated bit-streams. Sequences generated by LFSRs pass many randomness tests, and so are very suitable for SC.

2) Multiplication in SC

Multiplication of two stochastic bit-streams is performed using AND and XNOR gates in unipolar and bipolar encoding formats respectively, as illustrated in Fig. 4 and Fig. 5.

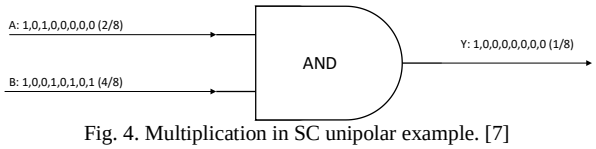

Fig. 4. Multiplication in SC unipolar example. [7]

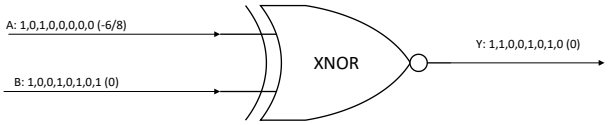

Fig. 5. Multiplication in SC bipolar example. [7]

In unipolar format, the multiplication of two inputs stochastic bit-stream of A and B is computed as:

$$Y = A \text{ AND } B \tag{7}$$

If the input sequences are uncorrelated, we have:

$$y = p_X = a \times b \tag{8}$$

Multiplications in bipolar format can be performed as:

$$Y = A \text{ XNOR } B \tag{9}$$

If the input sequences are uncorrelated, we have:

$$y = 2p_Y - 1 = (2p_A - 1) \times (2p_B - 1) \tag{10}$$

3) Addition in SC

The additions in SC are usually performed by using either scaled adders or OR gates.

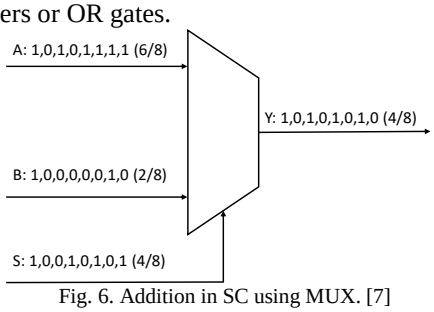

Fig. 6. Addition in SC using MUX. [7]

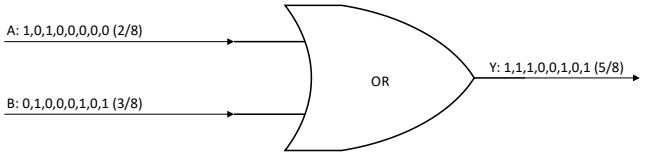

Fig. 7. Addition in SC using OR gate. [7]

The scaled adder uses a MUX to perform addition, as illustrated in Fig. 6. The output of a MUX Y is:

$$Y = A \times S + B \times (1 - S) \tag{11}$$

The expected value of p_Y would be $(p_A + p_B)/2$ when the select signal $S = 1/2$. The select signal $S = 1/2$ applied to the MUX's select input ensures that p_Y lies in the unit interval [0; 1] (with unipolar format) or [-1; 1] (with bipolar format) and so can be treated as a probability. If the input sequences are uncorrelated, we have:

$$y = 2p_Y \tag{12}$$

OR gates can also be used as approximate adders as shown in Fig. 7. OR gates function as adders only if p_{AB} is close to 0. This type of adders still requires long bit-streams to overcome a precision loss incurred by the scaling factor as mentioned in [7].

Besides, the additions in SC are also performed by using a new stochastic adder that is more accurate and does not require additional random inputs such as scaled adders. This adder is proposed by Vincent T. Lee [8] which is illustrated in Fig. 8.

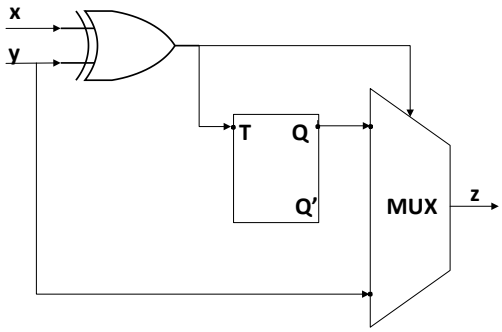

Fig. 8. TFF-based stochastic adder.

The area cost of a T flip-flop (TFF) is no more than a random number generator that is required for generating 1/2. More importantly, the bit-stream generated by the TFF is always uncorrelated with its input bit-stream. This means that there are no constraints on the auto-correlation of the input bit-stream, unlike common sequential SC circuits that do not function as intended if the input is auto-correlated. So, in this work, we use it instead of the original stochastic adder.

B. Architecture of Stochastic Computing for forward module

Compared to the reference architecture of LSI Contest illustrated in Fig. 9, we propose to transmit directly the bias, weight values, and remove the weight module, the bias module, controller module in our forward module. Fig. 10 shows our proposed architecture using SC in the forward module. Next, we replace adders and multipliers by Stochastic computing adders and Stochastic computing multipliers in the weighted input module. Fig. 11 shows the conventional weighted input module of hidden layer while the new weight input module is illustrated in Fig. 12. Similarly, Fig. 13 and Fig. 14 present the replacement of the multipliers in the weighted input module of output layer.

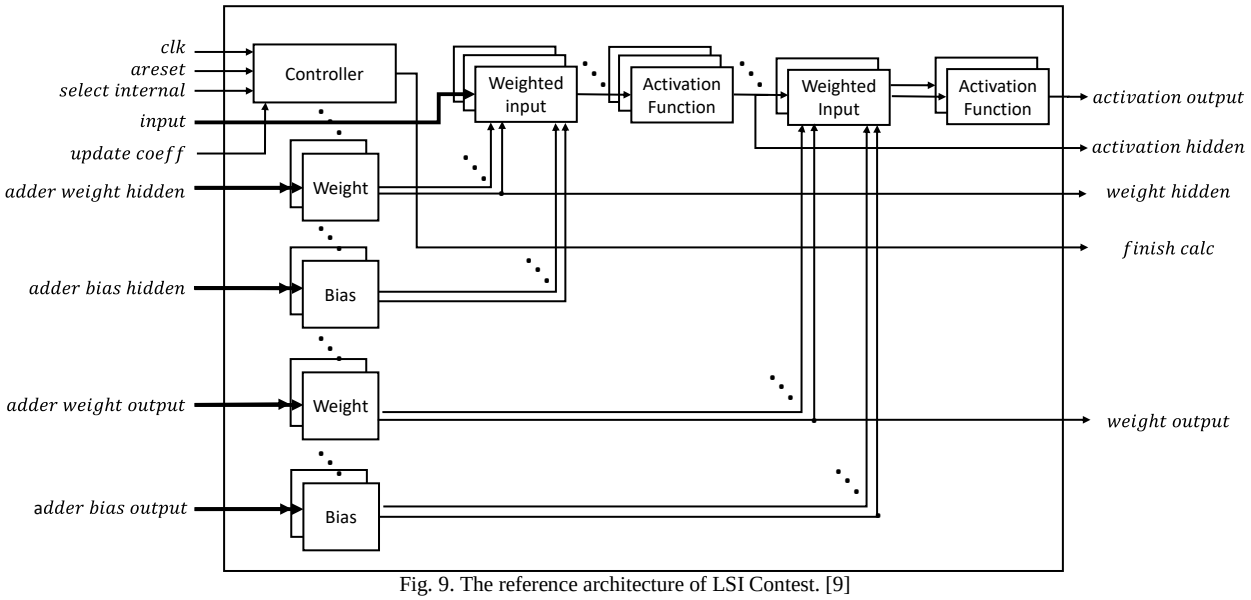

Fig. 9. The reference architecture of LSI Contest. [9]

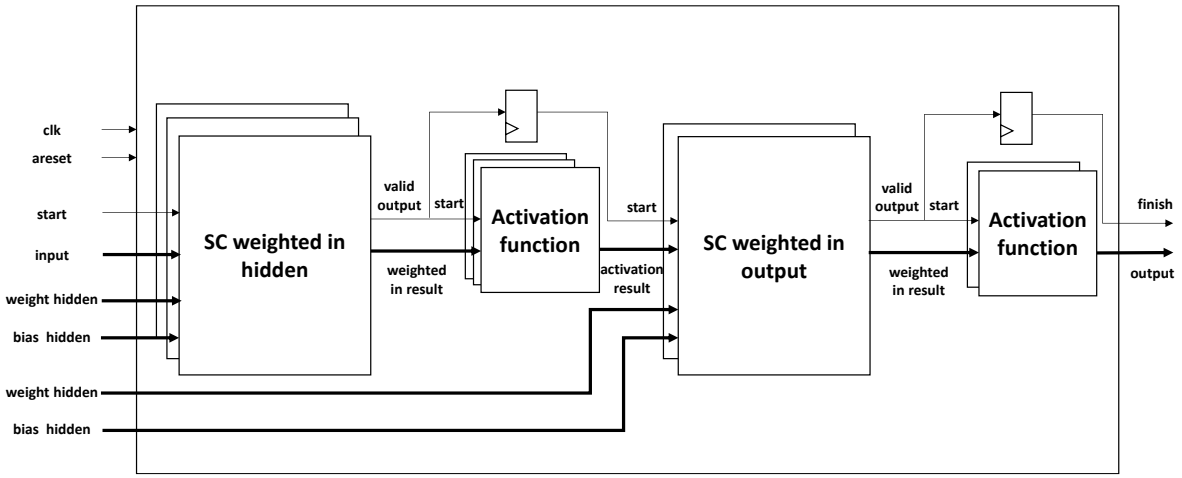

Fig. 10. Our proposed architecture using SC.

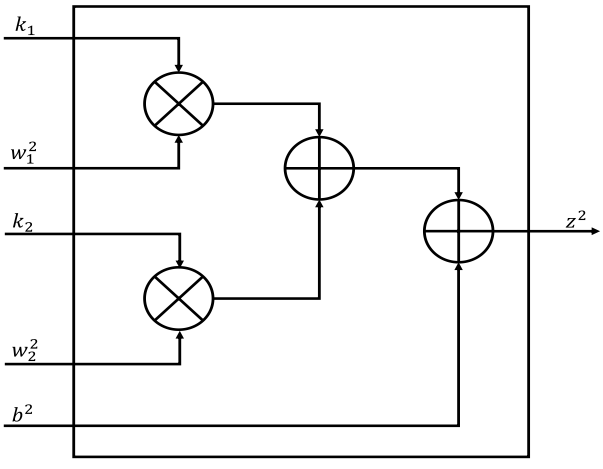

Fig. 11. The reference weighted input module of hidden layer. [9]

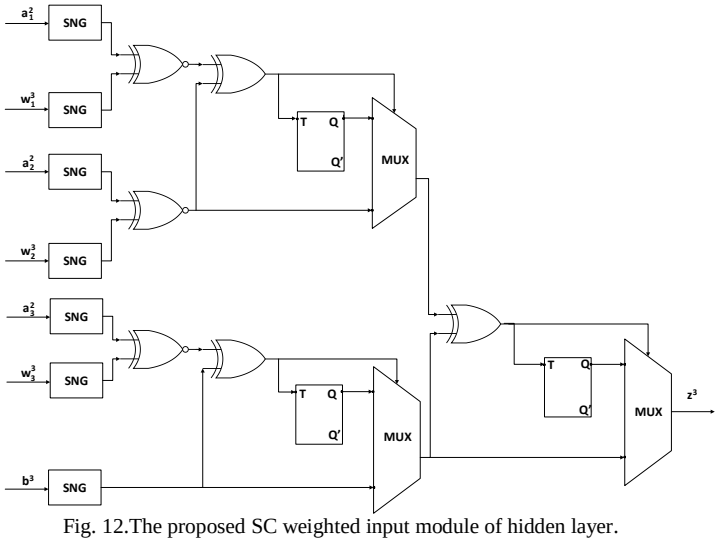

Fig. 12.The proposed SC weighted input module of hidden layer.

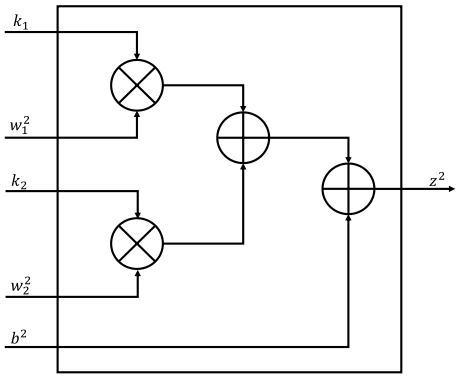

Fig. 13. The reference weighted input module of output layer. [9]

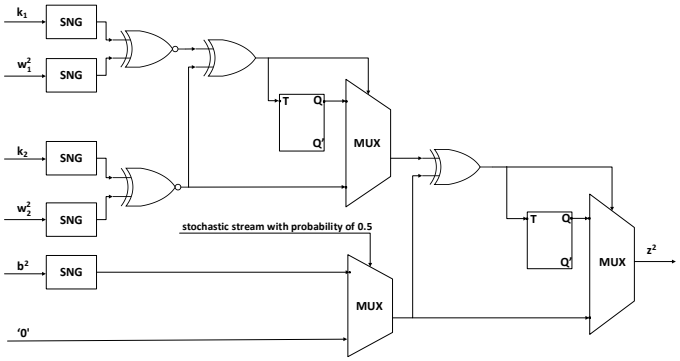

Fig. 14. The proposed SC weighted input module of output layer.

IV. Simulation and Implementation Results

A. Evaluation of the proposed Sigmoid function

TABLE 2. AREA COST AND ACURRACY EVALUATION (XILINX FPGA VC707 KIT)

	Conventional sigmoid model	**Proposed sigmoid model**
Area (nbr of cell)	85	67
MSE	3.12×10^{-7}	1.79×10^{-5}

B. Evaluation of the Stochastic computing

TABLE III. COMPARATION BETWEEN OUR MULTIPLICATION AND ADDTION METHOD AND PREVIOUS WORKS

Model	Vincent Lee [8]	**Our proposed**	
Type	8-bits unipolar	8-bits unipolar	8-bits bipolar
Multiplication	2.57×10^{-4}	7.43×10^{-6}	1.98×10^{-4}
Addition	1.91×10^{-6}	1.89×10^{-6}	1.32×10^{-5}

TABLE IV. THE ACCURACY OF SC FORWARD MODULE (WITH 1000 RANDOM TEST CASES)

	8-bits	10-bits
Timing	2^9+2 clocks	$2^{11}+2$ clocks
MSE	2.030733×10^{-4}	7.697317×10^{-5}

Table V. SYNTHESIS RESULT OF STOCHASTIC COMPUTING 2-3-2 ANN MODEL (XILINX FPGA VC707 KIT)

	LSI Contest [9]	**Our proposed**
Frequency	182MHz	**357MHz**
Slice LUTs	814	738
Slice Registers	528	450
Mux	60	75
DSP	12	**0**
IO	550	214
Area (cell)	2059	**1627**
Timing (clock)	4	$2\times2^{N(bit)}+2$

V. Conclusion

In this paper, we presented our proposed ANNs hardware architecture which employs new hybrid activation function architecture and Stochastic Computing in the forward module.

The implementation results of the proposed architecture on Xilinx FPGA technology show that the simplified Sigmoid reduces 21.18% area cost compared to the 16-bits LUT methods. In terms of the accuracy, the mean square error of the simplified Sigmoid is 1.79×10^{-5} and the 16-bit LUT is 3.12×10^{-7} . The implementation of the Stochastic Computing into ANNs at 2-3-2 model increases 1.96 times of the maximum frequency and reduces 20.98% as the LSI reference architecture.

Acknowledgment

This work has been done at VNU Key Laboratory for Smart Integrated Systems (SISLAB), the University of Engineering and Technology – a member of Vietnam National University Hanoi. We would like to thank the members of SISLAB for their helpful and continuous encourage.

References

[1] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: a tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, Mar. 1996.

[2] M. Zhang, S. Vassiliadis, and J. G. Delgado-Frias, "Sigmoid generators for neural computing using piecewise approximations," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 1045–1049, Sep. 1996.

[3] K. Leboeuf, A. H. Namin, R. Muscedere, H. Wu, and M. Ahmadi, "High Speed VLSI Implementation of the Hyperbolic Tangent Sigmoid Function," in *2008 Third International Conference on Convergence and Hybrid Information Technology*, 2008, vol. 1, pp. 1070–1073.

[4] B. Zamanlooy and M. Mirhassani, "Efficient VLSI Implementation of Neural Networks with Hyperbolic Tangent Activation Function," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 22, no. 1, pp. 39–48, Jan. 2014.

[5] B. R. Gaines, "Stochastic Computing Systems," in *Advances in Information Systems Science*, Springer, Boston, MA, 1969, pp. 37–172.

[6] “The Logic of Random Pulses: Stochastic Computing - ProQuest.” [Online]. Available: https://search.proquest.com/openview/4746644e3a0aa2ef4583641c69532ac5/1?pq-origsite=gscholar&cbl=18750&diss=y. [Accessed: 01-Mar-2018].

[7] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, “VLSI implementation of deep neural networks using integral stochastic computing,” in *2016 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, 2016, pp. 216–220.

[8] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze, “Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 13–18.

[9] “LSI Design Contest.” [Online]. Available: http://lsi-contest.com/shiyou_4e.html. [Accessed: 01-Mar-2018].