Multiple Stuck At Fault Diagnosis System For Digital Circuit On FPGA Using Vedic Multiplier and ANN

Mangesh Islampurkar, Kishanprasad Gunale, Sunil Somani, Nikhil Bagade,
Dr. Vishwanath Karad MIT World Peace University,
Survey No, 124, Paud Rd, Kothrud, Pune, Maharashtra, 411038
India

Received: September 16, 2021. Revised: March 7, 2022. Accepted: April 12, 2022. Published: May 30, 2022.

Abstract- In an electronics circuit, the presence of a Fault leads to undesired or unexpected results. The output of many nodes on the circuit is changed due to the presence of the Fault at one node. So, it is necessary to detect the nature of the Fault present in a particular faulty node. To detect the fault present in the digital circuit, it is necessary to understand logical behavior using mathematical modeling. After the successful modeling, parameters are extracted, and the database is generated. The mathematical model uses Hebbian Artificial Neural Network algorithms [1] [2]. The database generated is used by the fault detection system to find the masked and multiple faults. A fault detection system monitors the faults present in the test circuit and finds the origin and nature of the Fault [3] [4]. The database generated for single stuck-at faults is used to find the multiple faults present in the faulty circuit. In this paper, Modified Vedic Multiplication [5] [4] method is used to optimize the utilization of the proposed system. In this proposed design multiplier of {N x N} bit input and {N} bit output is used, due to which device utilization is decreased, which is the expected outcome from the design. This system is designed using ISE Design Suite and implemented on Spartan-6 FPGA [6] [7].

Keywords- Index Terms Fault Detection, Vedic Multiplication, Artificial Neural Network, Verilog HDL, Hebbian ANN [8].

I. Introduction

Electronics system uses theoretical concepts, mathematical calculations, and assumptions. Unexpected changes in the behavior of the system lead to Fault. However, the system is not working correctly in multiple cases as many things are assumed constant in nature, i.e., temperature, source voltage, etc.

Various faults can occur in the digital system. Before diagnosing the Fault, knowing the original testing circuit is necessary. To understand the behavior of the testing circuit, a mathematical model is derived instead of referring to the truth table. This mathematical model is reflected as the logical expression of the testing circuit.

Initially, the mathematical model is derived using the basic principles of Artificial Neural networks (ANN) [1].

As the mathematical model is derived, Field Programmable Gate Array (FPGA) has a logical expression of the testing circuit. The mathematical model operates on the FPGA operating frequency instead of the physical device. Hence, the speed of analysis of the physical device is increased. The mathematical model is also helpful in designing control systems for the physical device. After testing a control system for a mathematical model, it is tested on the natural system. To derive any mathematical model using ANN algorithms on FPGA, it is necessary to synthesize the multiplier module.

While synthesizing the multiplier module, the compiler is trying to synthesize the hardware multiplier. Digital Signal Processors (DSPs) have built-in hardware multiplier units. Thus, the compiler synthesizes the DSP module to perform multiplier operations. The number of DSP modules is limited in FPGA. Therefore, it is necessary to synthesize the dedicated structure for the multiplication operation. Many algorithms synthesize the multiplier like the Series Multiplier, Booths Multiplier, Parallel Multiplier, Vedic Multiplier [9], etc. Employing these algorithms, the compiler synthesizes multiplier using Look-Up Tables (LUTs) instead of the DSP module.

II. LITERATURE SURVEY

The experts present various analytics tools that can effectively detect the Fault in the system. Faster and more efficient multipliers should be utilized to increase system speed. A Vedic multiplier is one of the most effective ways to execute multiplications faster by omitting stages that arent required in the standard multiplication process [10] [9].

The Urdhva Tiryagbhyat Sutra, derived from Vedic mathematics, was used to design a high-speed 16 x 16

Vedic multiplier to cut down power consumption, a big issue in digital design. It is an unparalleled combination for performing any complex multiplication operation where speed is critical. Using Vedic Multiplication in Elliptic Curve Cryptographic (ECC) reduces the processing time. In [4] author presented the multiplier circuit using fault-tolerant adder circuits. In this system, the Fault and increase in the reliability of the multiplier circuits are reduced. The multiplier is coded in Verilog HDL ISE Sim 11 and used to perform simulations, and usage of the area and power consumption were obtained using the Xilinx ISE 11.

The Hebbian weight is based on the Hebbian learning rule. The winner output neuron obtained from the forward calculation part is adjusted. The whole circuit does not require the participation of CPU, FPGA, or other microcontrollers, providing the possibility to realize computing in memory and parallel computing. The advanced ECC high-speed architecture is designed using the FPGA technique for elliptic curve-based multi-level key exchange, age, and encryption mechanism (ECM-KEEM) [5] is implemented and evaluated with Virtex-7 and Kintex7 platforms. It is designed for enhanced performance to verify the improved application efficiency with Vedic multiplier-based design [9]. It solves fault detection problems and achieves objectives with the Hebbian theory's help [8], one of the neuroscience field's learning methods [11].

III. PROPOSED DESIGN

Figure (1) shows the block diagram representation of the designed system. Here, two input one output digital logic circuits are considered as test circuits.

Weights for a testing circuit are generated using ANN algorithms [11]. Weights are generated using the Hebbian algorithm. The suggested systems testing circuits are linearly separable. As a result, linear activation function techniques are employed. After successful mathematical modeling of the physical circuit, the parameters are extracted as controllability of every node and the effect of individual artificial faults on an individual node. Following the extraction of parameters from the mathematical module, a database comprises information on faulty and fault-free behavior and the controllability of individual nodes. This database is used by Fault Detection System (FDS) to detect the multiple and masked faults present in the Circuit Under Test (CUT) [3].

When an external event occurs, the fault detection procedure begins. The flags are set, and test patterns are applied based on the errors found inside the system. The test patterns responses are utilized to determine the origin and type of the Fault.

The proposed system has a 100 % accuracy for a single stuck-at fault; if an input or output is stuck at {0,1}, the proposed system can recognise the type of the fault. The proposed system has a 100% accuracy for a single stuck-at fault; if an input or output is stuck at {0,1}, the proposed system can recognize the type of the Fault. Multiple masked stuck at faults at the input stage can be detected with 100% accuracy, while multiple masked faults at the input and output stages can be recognized with 33.33% accuracy. CUT is a logical AND gate with two inputs in the proposed design, and a single output is used. If one of the inputs is stuck at 0 and the output is also stuck at 0, the suggested approach cannot identify a stuck at 0 faults at the output stage. However, if another input is stuck at {0,1}, the system can detect a fault in both inputs.

In the proposed system number of input tests pattern are generated to detect the Fault as minimum as possible, due to which it is easy to detect the Fault as quickly as possible. Let us consider CUT as a logical AND gate with two inputs and one output. If the CUT response output is stuck at 1, then the test pattern generated to test the Fault is {0,0} at both the inputs.

As ANN is already trained to detect all the possible combinations of faults, the number of test patterns needed to generate to find the possible Fault is decreased.

In this proposed design, a Vedic multiplier with {N x N}bit input and {N} bit output is required. As a result, a modified Vedic Multiplier using the straight and cross approach is required. In comparison to {N,N} bit input and {2N} bit output, these data show a decrease in device utilization.

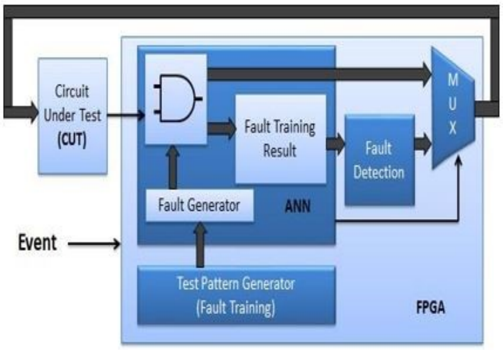


Fig. 1: Block diagram of the designed system

IV. MATHEMATICAL MODELLING

The mathematical model of the CUT is derived using algorithms of neural networks. The test circuit used in the designed system is two inputs and one digital output system. The inputs of the systems are linearly separable. Due to this, the line equation is synthesized inside the FPGA [11].The net weight equation designed for two testing inputs and one biased input is described per the generalized neural network as in (1).

$Net\,Weight = (X_1 \times W_1) + (X_2 \times W_2) + (\mathrm{b} \times W_\mathrm{b})$ (1)

Considering the example of Logical OR gate connected as an external test circuit to the Articial Neuron [12], after successful weight generation, the values for weights W_1 , W_2 and W_b are modified as per the weights calculated are as shown in Table I,

Table I: Hebbian Weight Calculation for logical OR operation [8]

Input			Output	Weight Generated			Weight Calculator		
X1	X2	b	Y	W1	W2	Wb	W1	W2	Wb
Initial							0	0	0
1	1	1	1	1	1	1	1	1	1
1	-1	1	1	1	-1	1	2	0	2
-1	1	1	1	-1	1	1	1	1	3
-1	-1	1	-1	1	1	-1	2	2	2

If the calculated weights in Table I are applied to (1), the resultant modied as in (2), (3), (4), and (5),

$$1 * 2 + X1 * 2 + X2 * 2 = 0 \qquad (2)$$

$$X2 * 2 = -(X1 * 2) - 2 \qquad (3)$$

$$X2 = -\left(\frac{X1}{2}\right) - \frac{2}{2} \qquad (4)$$

$$X2 = -X1 - 1 \qquad (5)$$

The equation used for boundary line condition for linear function with a negative slope is represented as (6),

$$y = mX + c \qquad (6)$$

The Equation (5), matches the equation of boundary condition shown in Equation (6). The line equation represented by the Equation (5) is shown in Fig. 2

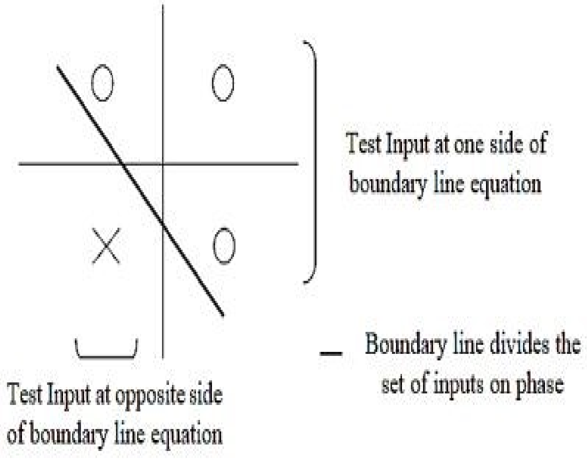

Test Input at one side of boundary line equation
Boundary line divides the set of inputs on phase
Test Input at opposite side of boundary line equation

Fig. 2: Line Plane representation of Equation 5

Here, Equation (5) substitutes on the plane. However, the boundary side described by logical HIGH and logical LOW is unknown logic. Input training database stored in RAM describes the logical HIGH state and logical LOW state of the boundary condition. The test pattern inputs are X1 = 1 and X2 = 1. Applying these values in (5),

$$1 = -1 - 1 \qquad (7)$$

$$1 = -2 \qquad (8)$$

The condition of equality does not satisfy by the (7) and (8). The right-hand side of the (8) is smaller than the left-hand side. It results in the upper side of the plane being described by logical HIGH and the lower side of the plane being described by logical LOW. In other cases, if the right-hand side of the Equation is greater than the left-hand side, the upper side of the plane is described by logical LOW, and the lower side of the plane is described by logical HIGH. Due to this, the upper side of the plane is described by logical LOW, and the lower side of the plane is described by logical HIGH.

A. Artificial-neural-network

The basic architecture of two inputs articial neuron[11], used to design the proposed neural network in this system is shown in Fig. 3

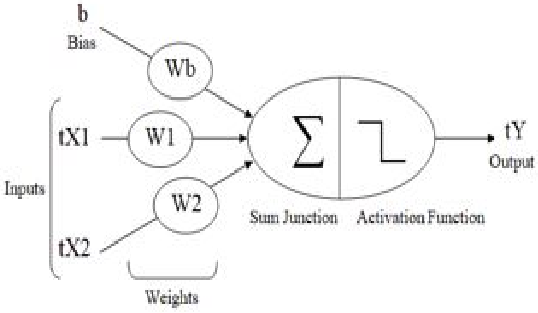


Fig. 3: Structure of Single Artificial Neuron[2]

In the proposed system, Bilinear classifier logical gates are considered as a test circuit i.e. Logical OR gate, and experimentation is implemented to verify the operation of logical gates on FPGA [13]. Hebbian weight generation algorithm [8] is used to calculate the weights. Simulation results using the ISE Design suite for weight generation [14] are shown in Fig. 4.

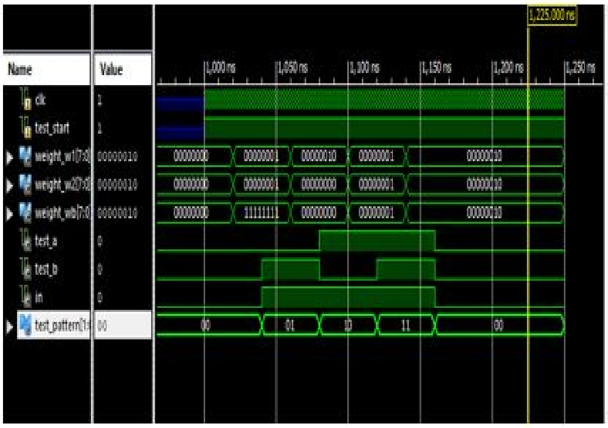


Fig. 4: Simulation results for weight generation using ISE Design Suite

Due to Bilinear classification, a linear activation function is applied, and the activation function is made up of two models: calculation equation and decision logic. Calculating the Equation combines the weights generated with the unknown input condition as shown in (1) and (2). The decision logic is utilized to determine the output condition based on a determined equation.

B. *Controllability*

Controllability of the system can be defined as the state condition of control input used to force the system to get known controlled output. To test an individual node for individual Fault, it is necessary to know the controllability of every node for logical LOW and logical HIGH [15]. The proposed system deals with generating a database of controllability for every node. The CUT has two inputs. So, for every node, two-bit data is generated for controllability of logical LOW and logical HIGH. There are two input nodes and one output node. Hence, a 12-bit database generated by the system represents the controllability of every node for logic LOW and logic HIGH.

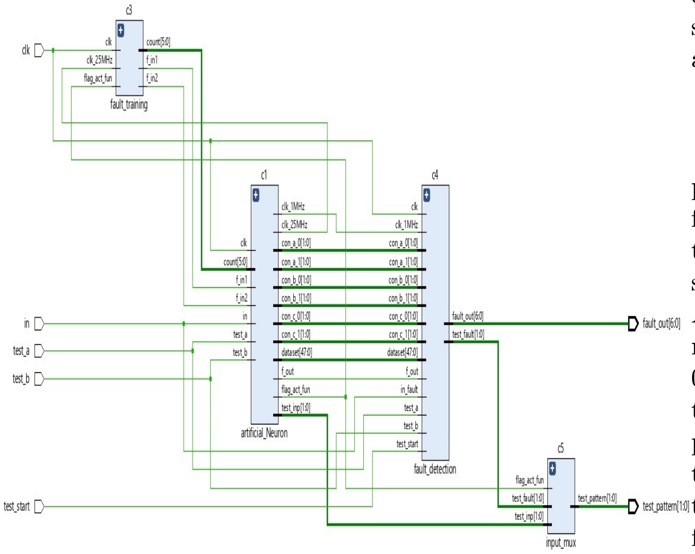


Fig. 5: RTL design of activation function with supervisory method using ISE Design Suite

C. *Artificial Faults Training*

After understanding the logical behavior of the system and the controllability of every node, it is necessary to understand the systems response to an individual fault.Once the mathematical model for the system is derived, it is essential to apply the faulty inputs to the ANN and generate the database for all single stuck-at faults. As shown in Fig. 5, a test pattern generator generates various test inputs with 6-bit flags. A fault generator block is used to generate every individual Fault in test input based on the flag value. This faulty test pattern is applied to the ANN, generating the fault database. The database generator is used to observe the nature of the ANN for fault-free response and faulty response [14]. The dataset generated for the faulty response of the logical OR gate for every single stuck-at Fault is shown in Table I. As shown in Table I, every fault response is compared with a fault- free response, and a mismatch between the faulty response and the fault-free response is stored in a database [15].

In the proposed system, a 48-bit database is generated. The database structure for two input logical OR gates used as a test circuit is shown in Table II.

Table III indicates some tristate conditions represented as zzz where faults result in a match with ideal results. The values are also considered data if the database contains logic HIGH or LOW values in such states. Thus, these exceptional conditions are represented by tristate logic, and these conditions change according to the test circuit connected to FPGA changes.

V. Fault Detection Unit

FDS is used to find the multiple and masked faults present in the test circuit. As shown in Fig.1, the database generated is applied to the FDS. The FDS starts to operate when an external trigger event triggers at a negative edge.

A. *Faults Identification*

According to the nature of the faults, if the Fault is present in the input node, the nature of the Fault is reflected in the output node. So, initially, it is necessary to find the number of faults present in the circuit. Considering the example of the logical OR gate, if the input A is stuck at logic LOW, the response of the system is mismatched with the ideal system for the condition {1, 0, 1}. According to Table II {0, 0, 0} condition indicates two faults input Sa0 and Output Sa0, respectively. The principal operation of the fault identification module is to find several faults present in the test circuit and set the flags based on faults detected. The simulation results for flag indication is shown in Fig. 6

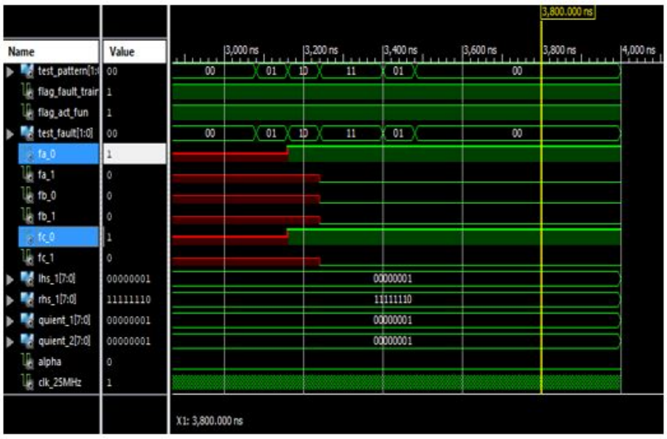


Fig. 6: Simulation results for flag indication using ISE Design Suite

Table II : Database Generate by apply artificial faults to ANN.

A	B	Out	a/0	a/1	b/0	b/1	Out/0	Out/1
0	0	0	0	1	0	1	0	1
0	1	1	1	1	0	1	0	1
1	0	1	0	1	1	1	0	1
1	1	1	1	1	1	1	0	1
Database			100	001	010	001	010	001
Generated							100	

B. *Multiple Fault Detection*

According to the nature of the Fault, the multiple faults are masked. The results generated from one Fault are masked or eliminated due to other faults. In this case, the Fault which is masked remains hidden or not detected. These hidden faults can be detected using a database generated for single stuck faults for every node and flag indicators. The algorithm used to detect the Fault at every node is described below,

- Case 1: No flag is Set: Test pattern is applied to control the node logical LOW. If the node is logical LOW, no hidden Sa0 fault is present. If no hidden Sa0 fault is present, the test pattern is applied to control the node logical HIGH. If the node is logical HIGH, no hidden Sa1 fault is present.

- Case 2: Sa0 flag is Set: Test pattern is applied to control the node logical HIGH. If the node is logical HIGH, no Sa0 fault is present.

- Case 3: Sa1 flag is Set: Test pattern is applied to control the node logical LOW. If the node is logical LOW, no Sa1 fault is present.

Table III Structure of database generated inside the RAM

Col 1	Col 2
Location 1 (No fault for input {0,0}	000
Location 2 (No fault for input {0,1}	011
Location 3 (No fault for input {1,0}	101
Location 4 (No fault for input {1,1}	111
Location 5 (Input a stuck_at_0)	100
Location 6 (Input a stuck_at_1)	001
Location 7 (Input b stuck_at_0)	010
Location 8 (Input b stuck_at_1)	001
Location 9 (Output stuck_at_0 for input {0,0}	zzz
Location 11 (Output stuck_at_0 for input {1,0}	100
Location 12 (Output stuck_at_0 for input {1,1}	110
Location 13 (Output stuck_at_1 for input {0,0}	001
Location 14 (Output stuck_at_1 for input {0,1}	zzz
Location 16 (Output stuck_at_1 for input {1,1}	zzz

In the case of a proposed system, the time required to detect the Fault depends on the stages present in the digital circuit. For the input stage, if the test circuit has n number of inputs and the test circuit is operating on frequency m, the maximum time required to detect a fault at the input stage is shown in (9),

$$\textit{Input stage fault time} = (n\text{- } 1)\ /\ m \qquad (9)$$

Similarly, to detect the Fault at stage 2, which is the output stage, the maximum time required is shown in equation (10),

$$\textit{Output stage fault time} = (n + 1)\ /\ m \qquad (10)$$

VI. Vedic Multiplier

[5] In the process of weight generation, multiplier operation is synthesized as given in (1) and (2) to synthesize this multiplier, the Vedic multiplication method is used. In the designed system, the structure of The Vedic multiplier is divided into a straight multiplier and a cross multiplier [10] [16]. With the combination of straight multiplier and cross multiplier, the {N x N} input and {N} output multiplier are designed [4].

A. *Straight Multiplier*

In this research work, a Straight Multiplier and Cross multiplier are designed to reduce the FPGA resources as compared to a generalized Vedic Multiplier, as shown in Fig. 7

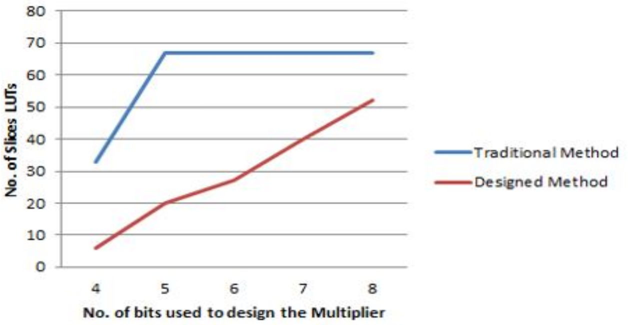
A straight Multiplier is used to multiply the single-digit input with carrying input and generates the single-digit output with carryout. The structure of the straight multiplier [17] is (as shown in Figs.7 a and 7 b)

B. *Cross Multiplier*

Cross Multiplier [18] is used to multiply the two-digit input with carrying input and generates the single-digit output with carryout. The structure of the cross multiplier is shown in (as shown in Figs. 8a and 8b).

The Vedic multiplier is described using a 4 × 4 Vedic multiplier and ripple carry adder to synthesize the multiplier. The device utilization of the existing Vedic multiplier and designed Vedic multiplier is shown in Table III.

According to Table III, the designed method uses fewer Look-up Tables (LUTs) due to the customized design. The FPGA performs bitwise operations. Due to this, no specified length data type is used. The length of the data type is custom or user-defined. According to the Vedic multiplier designed in [16], the length of the data type needs to be in 2^n format. This disadvantage is addressed in the proposed system. The comparison between device utilization for different length data types is shown in Table III.



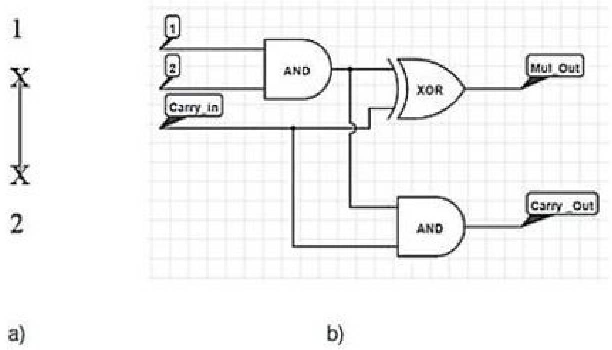


Fig. 7: Straight multiplier logic design (a) Straight multiplier structure (b) Multiplier logic design

Table III Device utilization of 8 × 8 Vedic Multiplier.

Devices technology	Calculate values	Existing	Design	Available
Slice LUTs	8	67	52	0
LUT FF pairs	0	0	0	0
Bonded IOBs	25	25	25	0
DSP48A1s	1	0	0	16

VII. Hardware Implementation

Fig. 9 indicates the hardware implementation of a fault diagnosis system [6]. IC D4071BC represents the Logical OR gate used as a testing circuit. Here, cap switches are used to insert the Fault in the testing circuit without physical damage. As shown in Fig. 11, if no fault is present in the test circuit, the output LED indicates the result of no fault detection on **Spartan 6 - X-XP6-X9** [7]

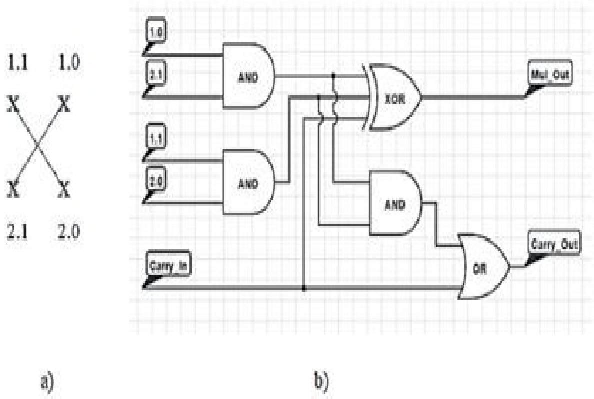


Fig. 8: Cross multiplier logic design a: Cross multiplier structure b: Multiplier logic Design

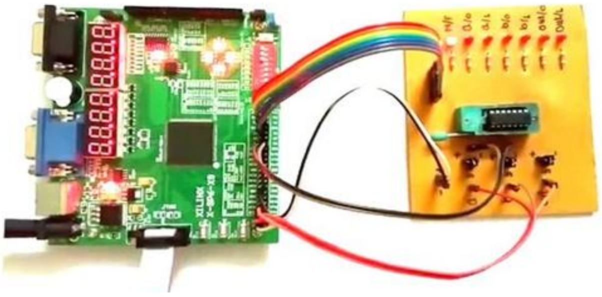


Fig. 9 Hardware Implementation Results for no fault detection in the designed system using Spartan 6- X-XP6-X9 [7]

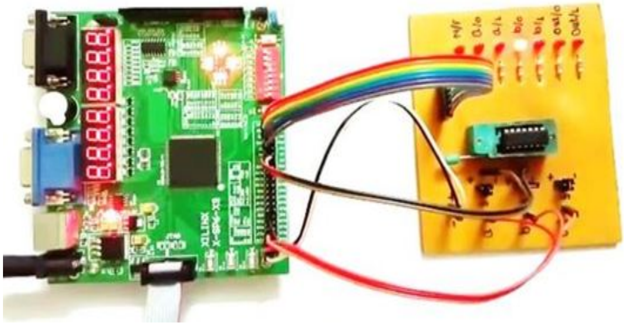


Fig. 10: Hardware Implementation Results for input b stuck at 0 fault detection in the designed system Spartan 6- X-XP6-X9 [7]

Conclusion A systematic technique is employed to discover multiple faults in the digital circuit. The suggested system is based on neural network architecture due to fault characteristics' high similarity and error overlap. The neural networks output creates a database used to determine the probabilities associated with the test flag. It is used to describe a specific type of Fault. The circuit in the proposed system optimizes the output processing so that the training data required for the circuit under test is minimal.

After successful training of the neural network using the Hebbian weight generation algorithm, the system is capable of robust fault diagnosis. The accuracy for a single stuck-at fault present at input and output in the proposed system is 100%. Multiple masked stuck at Fault present at the input stage were detected with 100% accuracy, and multiple masked faults present t at the input and out stage was detected with 33.33% accuracy. In the proposed system number of input tests pattern are generated to detect the Fault as minimum as possible, due to which it is easy to detect the Fault as early as possible. Since the ANN is already trained to detect all possible combination faults, the number of test patterns required to find the possible Fault is minimal.

FPGA [19] utilization of the system is optimized using Vedic Multiplier. In this proposed design, it is required to use a Vedic multiplier of {N x N} bit input and {N} bit output. As a result, a modified Vedic Multiplier using the straight and cross approach is required. In comparison to the {N, N}bit input and {2N} bit output multiplier, these findings show a reduction in device utilization.

The performance of a system for multiple fault diagnosis increases with increasing independent nodes, which is possible in large digital circuits. Multistage circuit fault detection can be possible using the self-organized map for future enhancement. The existing methods to detect faults are designed for a specific CUT or set of CUT. In the proposed method, the logical behavior of CUT is entirely unknown. The logical behavior of the system is analyzed and replicated by mathematical equations. The hardware utilization of mathematical equations is static concerning CUT. Hence, the proposed method can be used for more complex digital circuits or analog circuit fault detection with respective changes in the ANN algorithm.

References

[1] M. Li, Q. Hong, and X. Wang, "Memristor-based circuit implementation of competitive neural network based on online unsupervised hebbian learning rule for pattern recognition," *Neural Computing and Applications*, vol. 34, no. 1, pp. 319–331, 2022.

[2] S. Sivanandam and S. Deepa, *Introduction to neural networks using Matlab 6.0.* Tata McGraw-Hill Education, 2006.

[3] R. Isermann, *Fault-diagnosis systems: an introduction from fault detection to fault tolerance.* Springer Science & Business Media, 2005.

[4] G. G. Kumar and V. Charishma, "Design of high speed vedic multiplier using vedic mathematics techniques," *International Journal of Scientific and Research Publications*, vol. 2, no. 3, p. 1, 2012.

[5] C. Poomagal, G. Sathish Kumar, and D. Mehta, "Revisiting the ecm-keem protocol with vedic multiplier for enhanced speed on fpga platforms," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–11, 2021.

[6] F. de Souza Campos, M. M. Da Silva, M. E. Bordon, and J. W. Swart, "A logarithmic cmos image sensor with wide output voltage swing range," in *2017 International Caribbean Conference on Devices, Circuits and Systems (ICCDCS)*, pp. 69–72, Ieee, 2017.

[7] V. Bobin and D. Radhakrishnan, "A vlsi residue arithmetic multiplier with fault detection capability," in *Proceedings 1989 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 348–349, IEEE Computer Society, 1989.

[8] M. Ramos, V. Muñoz-Jiménez, and F. F. Ramos, "Learning clasiffier systems with hebbian learning for autonomus behaviors," in *Mexican Conference on Pattern Recognition*, pp. 328–339, Springer, 2020.

[9] S. Karthikeyan and M. Jagadeeswari, "Performance improvement of elliptic curve cryptography system using low power, high speed 16× 16 vedic multiplier based on reversible logic," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 3, pp. 4161–4170, 2021.

[10] S. Akhter and S. Chaturvedi, "Modified binary multiplier circuit based on vedic mathematics," in *2019 6th International Conference on Signal Processing and Integrated Networks (SPIN)*, pp. 234–237, IEEE, 2019.

[11] H. F. Restrepo, R. Hoffmann, A. Perez-Uribe, C. Teuscher, and E. Sanchez, "A networked fpga-based hardware implementation of a neural network application," in *Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (Cat. No. PR00871)*, pp. 337–338, IEEE, 2000.

[12] Q. Liu, T. Liang, Z. Huang, and V. Dinavahi, "Real-time fpga-based hardware neural network for fault detection and isolation in more electric aircraft," *IEEE Access*, vol. 7, pp. 159831–159841, 2019.

[13] F. Asghar, M. Talha, and S. H. Kim, "Neural network based fault detection and diagnosis system for three-phase inverter in variable speed drive with induction motor," *Journal of Control Science and Engineering*, vol. 2016, 2016.

[14] J. Ruiz-Rosero, G. Ramirez-Gonzalez, and R. Khanna, "Field programmable gate array applicationsa scientometric review," *Computation*, vol. 7, no. 4, p. 63, 2019.

[15] P. Palsodkar, P. Palsodkar, and R. Giri, "Multiple error self checking-repairing fault tolerant adder-multiplier," in *2018 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pp. 1–4, IEEE, 2018.

[16] M. Akila, C. Gowribala, and S. M. Shaby, "Implementation of high speed vedic multiplier using modified adder," in *2016 International Conference*

on Communication and Signal Processing (ICCSP), pp. 2244–2248, IEEE, 2016.

[17] K. D. Rao, C. Gangadhar, and P. K. Korrai, “Fpga implementation of complex multiplier using minimum delay vedic real multiplier architecture,” in *2016 IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON)*, pp. 580–584, IEEE, 2016.

[18] K. D. Rao, P. Muralikrishna, and C. Gangadhar, “Fpga implementation of 32 bit complex floating point multiplier using vedic real multipliers with minimum path delay,” in *2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, pp. 1–6, IEEE, 2018.

[19] A. Pathan, T. D. Memon, S. Keerio, and I. H. Kalwar, “Fpga based performance analysis of multiplier policies for fir filter,” in *2016 International Conference on Advances in Electrical, Electronic and Systems Engineering (ICAEES)*, pp. 17–20, IEEE, 2016.