Accelerating Test Generation by VLSI Hardware Emulation

Morteza Fayyazi and Zainalabedin Navabi
Department of Electrical and Computer Engineering
Northeastern University
409 Dana Research Center
Boston, MA 02115, USA
Tel : (617)-373-7780; Fax : (617)-373-8970
mfayyazi@ece.neu.edu

navabi@ece.neu.edu

Abstract:

A hardware emulation system based on a programmable VLSI array is used for test pattern generation for combinational circuits. The real-time simulation capability of the hardware emulator significantly improves time required for test generation time. The VLSI hardware emulator implements a parallel algorithm for test pattern generation based on neural networks. Test generation is achieved by mapping a circuit into its equivalent VLSI array test generation model emulating this test generation algorithm. The programmed array serves as a hardware accelerator for automatic test pattern generation

Impact:

The main impact of this method is in test generation of combinational circuits. Combining our VLSI programmable cells with shift register cells for register modeling and scan insertion, a complete digital system can be modeled for test generation in a VLSI emulator. Another potential application of this work is in digital implementation of a limited form of Hopfield neural networks.

Keywords: Test Generation, Hardware Emulation, Acceleration, Neural Networks.

Accelerating Test Generation by VLSI Hardware Emulation

Abstract

A hardware emulation system based on a programmable VLSI array is used for test pattern generation for combinational circuits. The real-time simulation capability of the hardware emulator significantly improves time required for test generation time. The VLSI hardware emulator implements a parallel algorithm for test pattern generation based on neural networks. Test generation is achieved by mapping a circuit into its equivalent VLSI array test generation model emulating this test generation algorithm. The programmed array serves as a hardware accelerator for automatic test pattern generation.

1. Introduction

We have used parallelism in neural networks to develop a parallel algorithm for test pattern generation. The hardware implementing those properties of the neural network algorithm that are used for test generation has been implemented on a programmable VLSI array. The programmed array becomes an emulator for the developed algorithm and an accelerator for test pattern generation.

A VHDL model reflecting the hardware of programmable VLSI array cells has been developed to simulate the algorithm in order to study hardware limitations and final real-time results. The performance estimation indicates that this approach could be several orders of magnitude faster than the existing software approaches for large designs.

In the sections that follow we will describe our test generation method its corresponding neural network modeling base on [2]. Next, in Section 3, utilizing the neural models for test generation is discussed. In Section 4, a hardware realizable in a programmable array is presented. This array format serves as the hardware that can be programmed for modeling and eventual test generation of the combinational circuit that it is representing. Our design of the VLSI array and its applicability to test generation is verified by modeling array cells in VHDL and performing VHDL simulation. This work along with comparisons with other test generation methods is presented in Section 5. Section 6 presents conclusions and future work that may utilize our technique for developing a complete programmable test array.

2. Test Generation

We have based our test generation acceleration on the ATPG algorithm that we presented in Reference 1. According to this algorithm, the netlist of a gate level circuit will be mapped into a Hopfield neural network for justification, and into a fault injectable circuit for fault injection and propagation. Faults are sequentially injected into the circuit. If a test vector can be found for an injected fault, it will appear on the circuit primary inputs. A test controller monitoring fault injection process and observing the primary inputs will report the generated test. Structure of a good circuit, faulty circuit and the output interface are shown in Fig. 1.

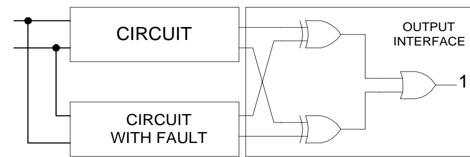


Fig. 1 Test generation for a 2-output circuit

According to the algorithm, the output interface uses models of bidirectional XOR gates and an OR gate. Forcing the output of OR gate to a ‘1’ value, causes at least one of the two outputs to have different values. Forcing the output of all gates to a certain value is possible when we use bi-directional gate models [2]. The following paragraphs discuss models used in Fig. 1.

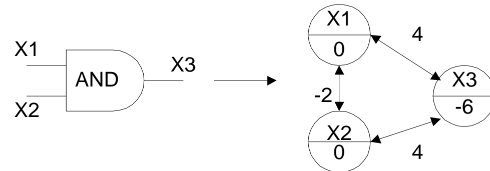


Fig. 2 AND gate and the corresponding model

For a gate level circuit, a neural network model can be obtained. For this purpose, each net (signal) of the circuit is represented by a neuron and the value on the net is the activation value (0 or 1) of the neuron (Xi). Each gate is independently mapped into a Hopfield neural network. Interconnections between the gates are used to combine the individual gate neural networks into a neural network representing the circuit. Neural network for 2-input AND, OR, NAND, NOR, XOR, XNOR, NOT gates constitute the basic set. Gates with more than two inputs are constructed from this basic set.

Figure 2 shows a 2-input AND gate and its corresponding neural network.

The advantage of this model is its forward and backward propagation capability. If the input neuron activations are forced on a state, the output neuron activations in the stable condition of the neural network will be consistent with the logic circuit functionality. In the opposite direction, by forcing the output neurons to a specific state, the input neuron activations will be justified by the neural network. The neural network for a digital circuit is characterized by an energy function E that has global minima only at the neuron states which are consistent with the function of all gates in the circuit. All other neuron states have higher energy. The summation of the energy functions of the individual gates yields the energy function E of the logic circuit. Since the individual gate energy functions can only assume non-negative values, E is non-negative. As an illustration, consider the logic circuit shown in Figure 3.

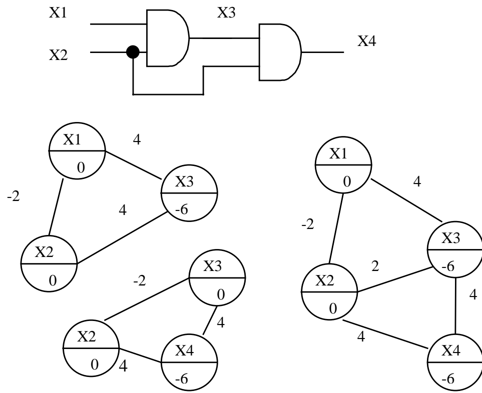


Fig. 3 Merging of two AND gates

The individual neural networks of the two input AND gates are merged together to result the neural network of the logic circuit as described below. Neurons with identical labels are replaced by a single neuron with a threshold equal to the sum of the original neuron thresholds. Similarly, identical edges are merged into a single edge with edge-weight equal to the sum of the original edge-weights. The energy of the neural network is calculated by the following expression:

$$E = (-1/2)\sum\sum T_{ij}V_iV_j - \sum I_iV_i + K;$$

where the range of Σ is the number of neurons in the neural network, T_{ij} is the weight of the edge between neurons i and j, V_i is the activation value of neuron i, I_i is the threshold of neuron i, and K is a constant. Obviously T_{ii} is equal to zero. The difference between energy of a neural network with its Kth neuron off (V_k = 0) and its energy with that neuron on (V_k = 1) is derived from the following expression:

$$\Delta E_k = E(V_k = 0) - E(V_k = 1) = I_k + \sum T_{jk}V_j$$

As it can be seen from the above expression, the following update rule reduces the total energy of the neural network:

$$V_k = (1, \text{ if } \Delta E_k>0 \text{ and } 0, \text{ if } \Delta E_k<0 \text{ and } V_k, \text{ otherwise}).$$

This updating rule is described as the gradient descent technique. The gradient descent algorithm terminates at a local minimum, due to the fact that it is a greedy algorithm. Greedy algorithms only accept moves towards reducing the energy of the neural network. To achieve a global minimum, probabilistic algorithms are devised that can accept moves towards increasing the neural network energy.

We consider stuck-at-fault model for fault simulation in a gate level circuit. This model consists of an acceptable coverage of physical errors. A test pattern for a given stuck-at-fault is a case of primary inputs that is able to control the faulty line to its inverse value, and make that fault observable at the primary outputs of the circuit. Therefore by assigning a test pattern to the primary inputs of the circuit under test, outputs of the faulty and the fault-free circuit will be different.

There are two blocks referred as the good circuit and the faulty circuits in our model (Fig. 1.) The fault list is produced by fault collapsing process, faults from this list are sequentially injected to the faulty circuit. The faulty circuit performs fault simulation, and the good circuit performs justification of input values for any injected fault in the test generation process. There is a direct connection between primary inputs of two blocks. An interface is needed between the primary outputs of two blocks. If there is only one primary output for the circuit, the interface can be a NOT gate; otherwise the interface should make sure that there is at least one different bit between two primary outputs. If the model settles down into a valid state, the values at the PIs are the generated test pattern. However if the PI values change, the above process should continue with the new primary input values.

The model described above is appropriate for the good circuit since it is capable of input justification. The faulty circuit operates in the forward propagation mode and a unidirectional model with ability of fault injection will function properly. For this circuit, it is desirable to be able to inject faults easily and be able to propagate faults with minimum simulation time. Meanwhile, the structure of the circuit model must not be changed for each fault injection. The good circuit model will not be suitable for this purpose, since there is not a simple way

for fault injection and the structure of network should be changed in order to inject a fault. The simulation time will also be long, due to two direction edges that are not required for the faulty circuit. Unidirectional simulation insures the faster simulation and simplicity of fault injection.

3. Test Controller

It is assumed that first, a C program receives the gate description of the original circuit and computes weights and thresholds of the Hopfield neural network. Then it outputs the good circuit component. The faulty circuit component is generated adding fault injection multiplexers to the original circuit signals. Primary outputs of two components are connected to each other by an interface and the primary inputs of two components need to have similar values. The circuit starts with zero values assigned to the primary inputs and fault is injected to the faulty circuit. Primary outputs of the faulty circuit will become available after several delta times.

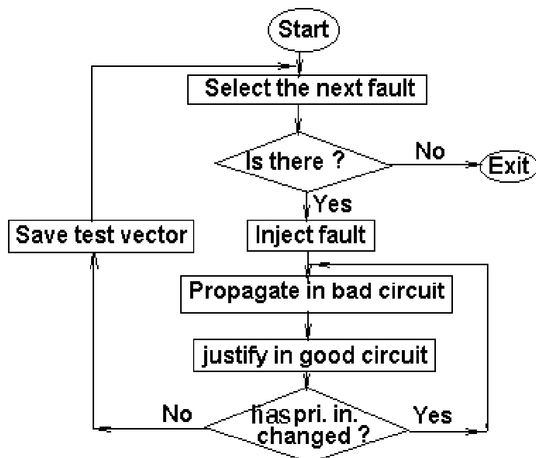


Fig.4. Test pattern generation system

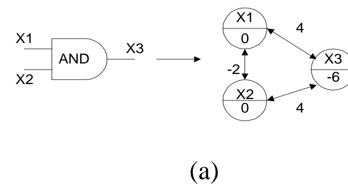
The interface circuit transmits values of the primary outputs of faulty circuit with at least one toggled bit to the good circuit primary outputs. The values of good circuit will be justified until the network reaches to a stable condition. If in this case the new primary inputs are the same as before, they will be the test vector for injected fault; otherwise the above process will be repeated with the new primary inputs. When the test vector is obtained, other faults will sequentially be injected. Fig. 4. Shows the test pattern generation process.

Unlike simulation, a hardware emulator performs gate evaluation in parallel, which can provide real-time logic operation and fast design verification, and hence greatly reduce the design turnaround time.

4. Hardware Implementation

Because of complexity involved in the implementation of a Hopfield neural network in digital logic (as they are implemented by analog circuits), in the design of our FPGA-based emulator we have only considered a subset of this type of a network. This subset covers propagation and justification that are necessary for a test generation algorithm. For instance, since neural network parameters are calculated according to the constant values of primitive gate models, neural network learning algorithms have not been implemented in the emulator design. The simple two-valued logic of neuron outputs in the ATPG algorithm has made digital implementation of the algorithm more efficient and practical than analog approach.

The novel idea of digital implementation of a Hopfield neural network has been resulted from a new way of looking at these networks. We have considered a complex Hopfield neural network as the collection of primitive neural networks each of which has been separately designed using a state machine. We can define some primitive elements that contain two or three neurons. These elements combine to construct a network. As an example, the neural model of an AND gate and its equivalent hardware are shown in Fig.5.



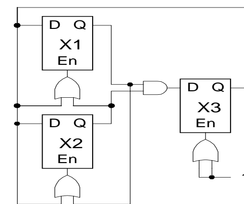
(a)

X1	X2	X3
0	0	0
0	1	0
1	0	0
1	1	1

X1	X3	X2
0	0	X2
0	1	1
1	0	0
1	1	1

X2	X3	X1
0	0	X1
0	1	1
1	0	0
1	1	1

(b)



(c)

Fig. 5 a) Neural model. b) Truth table of three signals, c) AND gate equivalent hardware

To extract a hardware model for a gate that is compatible with its neural model, state tables for each gate terminal in terms of other terminals are generated, as shown in Fig. 5(b). For an AND gate, the output signal has the truth table of AND functionality, while the input signals retain their values for “00” combination on output and the other input. Saving of input values shown in the first rows of X2 and X1 tables in Fig. 5(b), is implemented by D-type flip-flops attached to these input signals. For the output to become compatible with inputs of the same gate a same flip-flop as that used with input signals is placed on the output as a buffer. The circuit shown in Fig. 5(c) have the same stable states as the neural model of AND gate, in Fig. 5(a).

The difficult part of this approach is the way primitive models of logic gates are linked to form a complete circuit. In order to reduce the complexity of this work, at this time we are limiting ourselves to NAND gates only. Therefore, the state machine hardware implementation for NAND gates is all that we need in our primitive neural network library. Furthermore, the mechanism for linking NAND gates is easier than having to link gates of different types.

For a more efficient design, we are proposing a transmission-gate structure for a primitive NAND gate corresponding to the hardware of Fig. 5(c). This hardware, shown in Fig. 6, has been designed with the ability of fault injection and ease of connection to other gates.

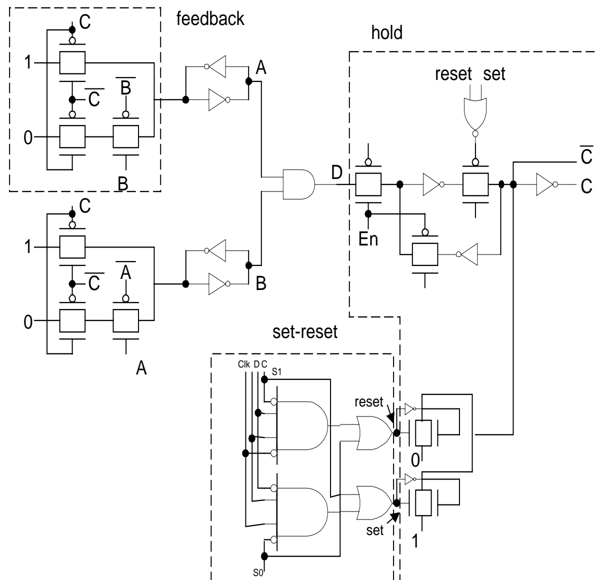


Fig. 6. Transmission-gate structure for a primitive NAND gate

The structure involves three main parts consisting of a feedback, hold circuitry and set-reset. The two feedback parts and their multiplexer logic input implement input signal flip-flops and their enabling logic. The hold circuitry on the output is the buffer output signal in addition to circuitry for forcing it into a given state of fault, preset or reset. The set-reset circuitry has stuck-at-1 and stuck-at-0 inputs for forcing a faulty state into a gate output. This part also provides circuitry for checking the stable state of a gate. A gate is in stable state when input and output of its hold circuitry are the same.

Transistor circuit of Figure 6 facilitates gate input-output interconnections. Figure 7 shows two NAND gates with one input connected. Fa and Fb are feedback inputs from gate with c output, while Fd and Fe are feedback inputs from gate with f output. Interconnection of b and d inputs requires Fb and Fd to be multiplexed into a common register implementing both interconnected inputs.

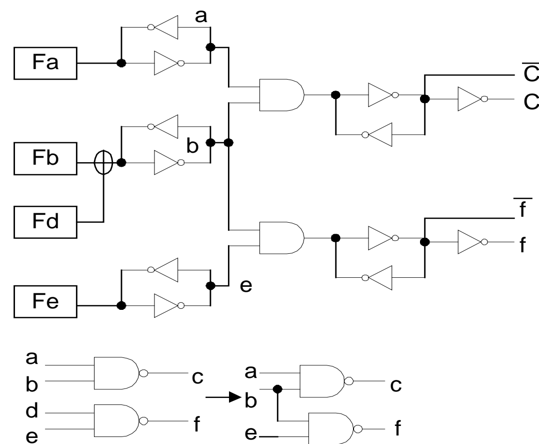


Fig. 7. Connection of two 2-input NAND gates

5. Hardware verification

For verifying our hardware components and comparing our model to other ATPGs, we have used VHDL modeling and simulation. According to the References 3, 4 and 9 we conclude that timing results in the case of hardware implementation would be several orders of magnitude faster than the software simulation. Some of the simulation results we obtained are reported in Table 1.

Table 1. Comparing simulation results of the Accelerator to several other methods

	Circuit:	C5315	C3540	C2670	C1908
Circuit Properties	Gates	2307	1669	1193	880
	Inputs	178	50	233	33
	Faults	5350	3428	2747	1879
	Testable faults	5291	3291	2630	1870
	Test patterns	123	162	118	124
Run Times of test gen methods (Sec)	SOCR, Ref. [8]	14.2	23.6	12.9	12.6
	Nemesis	74.8	264.7	371.2	69.8
	TRAN, Ref. [5]	32.1	23.9	92.9	12.6
	TIP, Ref. [7]	6.7	11.5	6.7	4.6
	Genetic, Ref. [6]	21.4m	17.7m	16.1m	4.57m
	Accelerated	156.2	95.7	67.3	35.3

	Circuit:	C1355	C880	C499	C432
Circuit Properties	Gates	546	383	202	160
	Inputs	41	60	41	36
	Faults	1574	942	758	524
	Testable faults	1566	942	750	520
	Test patterns	87	57	56	54
Run times of test gen methods (Sec)	SOCR, Ref. [8]	6.1	1.6	2.9	1.4
	Nemesis	22.0	37.5	3.9	8.5
	TRAN, Ref. [5]	6.6	2.9	1.8	0.8
	TIP, Ref. [7]	1.6	0.4	0.7	0.6
	Genetic, Ref. [6]	1.97m	1.24m	38.4	43.4
	Accelerated	28.4	20.3	8.9	5.5

6. Conclusions

This paper presented layout of a programmable VLSI array for accelerating test generation of combinational circuits. The complete implementation of this work requires availability of such a VLSI array. However, an FPGA with programmable MOS switches may also be used for this purpose. The Cross-point CP20K FPGA does provide this capability.

The main impact of this method is in test generation of combinational circuits. Combining our VLSI programmable cells with shift register cells for register modeling and scan insertion, a complete digital system can be modeled for test generation in a VLSI emulator. Another potential application of this work is in digital implementation of a limited form of Hopfield neural networks.

References

[1] M. Fayyazi, Z. Navabi "Using VHDL Model for Automatic Test Generation" Spain IEEE Symposium on Hardware Modeling, April 1997.

[2] S. T. Chakradhar, M. L. Bushnell and V. D. Agrawal "Automatic Test Pattern Generation Using Neural Networks." In IEEE Proceedings of International Conference on CAD, pages 416-419, November 1988.

[3] Jin-Hua Hong, Shih-Arn Hwang, and Cheng-Wen Wu "An FPGA-based Hardware Emulator for Fast Fault Emulation," 1997 IEEE.

[4] Kwang-Ting Cheng, Shi-Yu Huang, and Wei-Jin Dai "Fault Emulation: A New Approach to Fault Grading," 1995 IEEE.

[5] S.T.Chakradhar, V.D.Agrawal, and S. G. Rothweiler "A Transitive Closure Algorithm for Test Generation." IEEE Transaction on CAD of Integrated Circuits, pages 1015-1028, 1993

[6] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann "A Genetic Algorithm Framework for Test Generation." In IEEE Transaction on CAD of Integrated Circuits and Systems. PP 1034-1044, September 1997.

[7] M. Henftling, K. J. Antreich, H. Wittmann "A Formal Non-Heuristic ATPG Approach." In Proceedings European Design Automation Conference, pages 248-253, 1995

[8] M. Schulz, E. Trischler, and T. M. Sarfert "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System." In Proceedings IEEE International Test Conference, pages 1016-1025, September 1987

[9] D. Jones and D. Lewis "A Time-Multiplexed FPGA Architecture for Logic Emulation." In Proceedings of the IEEE 1995 Custom Integrated Circuits Conference, pages 495-498. IEEE, May 1995