The Test Vector Problem and Limitations to Evolving Digital Circuits

Kosuke Imamura James A. Foster Axel W. Krings
Computer Science Department
University of Idaho
Moscow, ID
83844-1010
{kosuke,foster,krings}@cs.uidaho.edu

Abstract

How do we know the correctness of an evolved circuit? While Evolutionary Hardware is exhibiting its effectiveness, we argue that it is very difficult to design a large-scale digital circuit by conventional evolutionary techniques alone, if we are using a subset of the entire truth table for fitness evaluation. The test vector generation problem for testing VLSI (Very Large Scale Integration) suggests that there is no efficient way to determine a training set which assures full correctness of an evolved circuit.

1 Introduction

Evolutionary computation has shown its effectiveness, particularly in the applications where the search space is huge and multi-modal. Many evolutionary applications have been implemented in software in the past. However, a recent development in evolutionary computation is its application to hardware design using programmable and reconfigurable electronic devices such as FPGA (Field Programmable Gate Array). While evolved hardware successes have been reported [NASA99, ICES98, GP98, GP97], it is important to investigate under what circumstances Evolutionary Hardware (EHW) would be successful, and unsuccessful. Our conclusion is that truth table driven EHW is likely to succeed in two cases: 1) there are large numbers of "don't care" bits involved, and 2) fitness evaluation is exhaustive. Circuit testing is an integral part of VLSI chip production, and the test vector generation problem, i.e., finding input test vectors that expose hardware faults at the output, has been well studied [AgrawalSeth87]. EHW performs circuit verification through fitness evaluation using selected training samples. The training samples are in fact test vectors for EHW. In the following, we present a scaled down problem, yet it describes our concerns. We emulate the evolution process using an AND-OR logic array, since every feed-forward gate network circuit can be transformed into sum-of-products. Suppose that we are evolving an OR gate. Assume that we chose input vectors (0,0), (0,1), and (1,0) as a fitness evaluation set. The desired outputs of this test input set should be 0, 1, and 1 respectively. Can we conclude that this test set is sufficient to verify the correctness of this circuit? We consider evolving the connection points on the AND and OR planes of **Figure 1**.

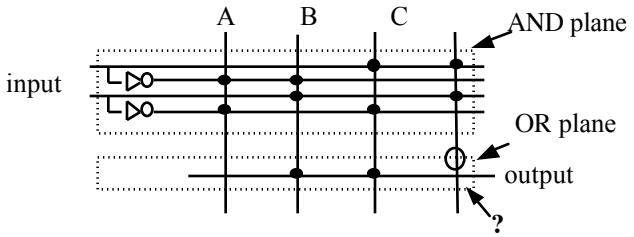

Figure 1: Emulated evolution of an OR gate on a logic array.

Every feasible input can be recognized by the four lines A, B, C, and D. Suppose that our evolved circuit produces correct output for input vectors (0,0), (0,1), and (1,0) which are emulated on A, B, and C. But what would be our reasonable expectation on line D, which did not receive any training? How do we know which test vectors must be included in the fitness evaluation? This raises the following fundamental question:

What fitness function and which input-output samples can guarantee that the evolved circuit is indeed what we want, if we are using only a subset of the truth table of the target function?

2 The test vector problem and fitness evaluation

A fault is a physical defect, imperfection, or flaw that occurs within some hardware or software component [Johnson96]. In this paper only hardware faults are considered. A test for faults is an input vector (a vector or a sequence of vectors) that will produce different outputs in the presence and absence of the fault [AgrawalSeth87]. It is difficult to generate test vectors that uncover as many faults as possible. To see why, consider random testing for the line stuck type fault in **Figure 2**. It passes all 2^{19} test vectors but one. The only test vector that detects the stuck-at-0 fault at X is ($i_1..i_{11}$=1, $i_{12}..i_{19}$=0). This example points out a serious problem with random VLSI fault diagnosis. The test vector generation problem addresses how the number of test vectors can be kept reasonable.

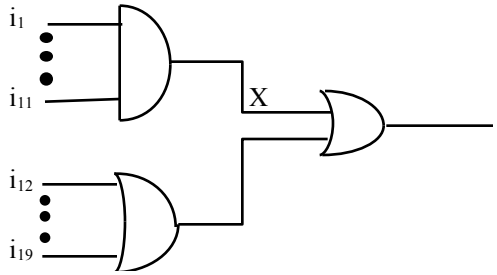

Figure 2: Line stuck example modified from [MazumderRudnick99]

A combinational or non-sequential circuit consists of interconnected gates with no feedback loops, while a sequential circuit includes feedback loops, which may be clocked or non-clocked. There are algorithms to test combinational circuits [Fujiwara85]. However, the problem of generating a test for a given fault has been proven to be NP-complete even for combinational circuits [ChengAgrawal89]. Testing a sequential circuit is a far more complex task than testing a combinational circuit and no satisfactory methods is known. For complex chips, *scan* design methods (by which flipflops can be initialized through a shift-register) can reduce the complexity of sequential circuit testing [AgrawalSeth87]. An approach to the test vector generation problem based on genetic algorithms is discussed in [RudnickHsiaoPatel99, MazumderRudnick99].

A feed-forward EHW gate network is a non-sequential circuit. There are known test vector generation algorithms for non-sequential circuits, such as the D-algorithm, PODEM, and the fan algorithm [Fujiwara85], but they assume that the circuit structure is known. These algorithms generate test patterns to detect assumed faults. What would be the assumed faults if the circuit structure were not known? A black box has only the input-output specification without the circuit diagram. What would be an efficient algorithm to generate test vectors for a black box? We define the term, "efficient algorithm" as follows:

Let **S** be the input-output specification and let **I** denote a set of input vectors. The corresponding set of output vectors, **O,** can be obtained by **O** = **S**(**I**). The most accurate testing is the exhaustive test. Let $\mathbf{I}_a$ be the set that contains all the feasible inputs. We are interested in the number of incorrect outputs. Therefore, we define **D** as

$$\mathbf{D}(I_i, O_i) = \begin{cases} 0 : \text{if input } I_i \text{ results in the specified} \\ 1 : \text{otherwise} \end{cases}$$

Thus, the perfectly functioning circuit or chip is expressed as $\mathbf{f}(\mathbf{I}_a) = \sum \mathbf{D}(I_i, O_i) = 0$ for all $I_i \in \mathbf{I}_a$. A circuit passes the test if $\mathbf{f}(\mathbf{I}_p) = 0$. We may prefer $\mathbf{I}_p \subset \mathbf{I}_a$ over $\mathbf{I}_q \subset \mathbf{I}_a$ if the reject rate is lower when the circuit is tested with $\mathbf{I}_p$ than with $\mathbf{I}_q$. The reject rate is the ratio of the number of defective chips over the number of chips passing the test. An efficient algorithm is one that generates $\mathbf{I}_p$ such that there exists correlation between $\mathbf{f}(\mathbf{I}_p)$ and $\mathbf{f}(\mathbf{I}_a)$, i.e., when $\mathbf{f}(\mathbf{I}_p) = 0$ it is likely that $\mathbf{f}(\mathbf{I}_a) = 0$.

Is there an efficient algorithm to generate test vectors for a black box? The answer is empirically "No". We now show that fitness evaluation is in fact one special case of the test vector generation problem, namely black box testing. The diagnostic resolution is the quantity of information on locations and types of fault. If a test detects only the presence of faults and no other information, then the diagnostic resolution is zero [Fujiwara85]. Since any test vector for a black box detects only the presence of faults, the diagnostic resolution is zero. The fitness evaluation of a typical EHW does not consider how the gates are connected. Assume that the fitness is computed based on deviation of the output, for instance, Hamming distance or binary distance. The training samples provide no other information than the existence of design flaws (faults). Thus, this type of fitness evaluation is equivalent to a fault detection test with zero diagnostic resolution. Hence, fitness evaluation is black box testing.

Even if we did account for the structure of evolved circuits in the fitness function, training would have to be based on input similar to those derived from standard test vector generation algorithms, e.g., PODEM or the fan algorithm. To illustrate this observation, we evolve a circuit that performs (A **AND** B). Assume that the evolved circuit is (A **AND** (B **OR** ~B)). This case is equivalent to a stuck-at-1 fault, (A **AND** 1), and fitness evaluation becomes exactly the same as the test vector generation.

[YaoHiguchi96] points out the following difficulties of EHW relating to fitness evaluation:

1. If all the combinations of input are used to evaluate fitness, then we do not have problems. However, it is not economically feasible.
2. A fitness function, which guarantees the correctness of the circuit, is very difficult to create.
3. It is difficult to know for EHW when a correct circuit has evolved.

It is our observation that these difficulties are a manifestation of the fact that there is no efficient algorithm to generate test vectors for a black box. Although the observation is simple, the following can be said about an evolved feed-forward gate network:

1. The exhaustive fitness evaluation ensures the correctness of the circuit. However, if a truth table based EHW requires the exhaustive fitness evaluation, then why do we need an EHW? A straightforward (simplified) sum-of-product can implement the truth table.
2. If $\mathbf{I}_p$ is used for the fitness evaluation, then the high correlation must be observed between $\mathbf{f}(\mathbf{I}_p)$ and $\mathbf{f}(\mathbf{I}_a)$. But, it seems that there is no known mechanism that forces the high correlation between them.

Numerous EHW applications have been tried successfully in the past. In the next section, we will investigate why EHW does and does not seem to work.

3 Current evolutionary approaches

[DamianiLiberaliTettamanzi98][DamianiTettamanziLiberali99] applied EHW to a 16-bit to 8-bit hashing function generation. The fitness is measured in terms of even distribution of the address space mapping. Hashing does not require state information. As a result, the hashing function generation circuit is implemented by a combinational circuit. We should notice that this can be implemented by specifying any 8 bits of input as "don't cares", which guarantees the perfect even distribution. This example shows that EHW can discover "don't care" components as specified.

[MillerThomsonICES98] carried out EHW experimentation evolving combinational circuits for 2-bit and 3-bit multipliers. They conclude that it is very difficult to evolve correct circuits using only a subset of the entire truth table, even for a 2-bit multiplier. This is an unfortunate conclusion because the truth table grows exponentially as the number of input bits increases. Why is this difficult? The circuit used for this experiment is a feed-forward structure of gates. If only a subset of the truth table is used during evolution, the circuit may output correctly if the input is included in the subset. But nothing can be said about the correctness of output resulting from input that is not in the training set. We show later that correct generalization is fundamentally limited by the number of sample input-output patterns.

Evolutionary Algorithms perform well to obtain approximate optima. Given that exact correctness of a circuit needed by a multiplier is difficult, perhaps EHW can approximate real-valued functions. This question was raised by Miller and Thomson [MillerThomsonGP98] in the context of a square root function, indicating the limitation of a feed-forward structure. Their best result is an approximation by a stair case function (**Figure 3**), suggesting that the least significant bits (low-order bits) became "don't cares".

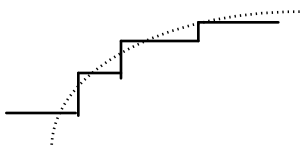

Figure 3: Approximation by a stair case function

If low-order bits have more influence over the approximation than the most significant bits, then fitness will be low. The interval between 0.0 and 0.1 is not well approximated. Miller and Thomson speculate this is because of the rapid change in the gradient of square root (x). Indeed, if the gradient is steep, we need to have fewer low-order "don't care" bits in input numbers. On the other hand, when the tangent line becomes flat, we can have more low-order "don't cares" bits in the input. This implies that it is a necessary (but it is not sufficient) condition that we sample more often in (0.0-0.1). If we have too many "don't cares" in this intervals, more samples will not increase the accuracy of the approximation. Therefore, the number of "don't cares" must vary from one interval to another. It is also possible to design a more precise circuit by conventional techniques. For example, one can divide the domain into several intervals. The truth table for each interval is implemented on a different logic array, using some portion of the input bits (low-order bits in this case). The remaining input bits are then used to select the appropriate logic array. Essentially, this is address decoding.

Now, what is the chance for the output to be correct if the input is not in the training set? Assuming the output for untrained input data is a uniformly distributed random variable, we have certain success rates for producing 100% correct circuits, using a subset of the truth table for the training set. Some definitions are needed:

N: number of total feasible input vectors.
T: number of input/output vector samples in the training set, ($T \leq N$).
O: number of correct output bits that a trained EHW needs to produce.

If the evolved circuit is 100% correct using the training set, then the probability of being 100% correct on all the feasible inputs is $(2^{-O})^{(N-T)}$. The term (2^{-O}) is the probability of having a correct output and there are (N-T) input-output pairs outside of the training set. If we consider the OR gate evolution example of **Figure 1**, the circuit will produce 1 or 0 for input vector (1,1), which is not in the training set. N = |{(0,0), (0,1),(1,0),(1,1)}| = 4, T = |{(0,0), (0,1),(1,0)}| = 3, and O = 1. Thus, we have $(2^{-1})^{(4-3)}$. So, the evolved OR gate has a 0.5 probability of being an OR gate and a 0.5 probability of being an exclusive OR gate.

In the case of a 2-bit multiplier, there are 16 possible multiplier-multiplicand combinations. The output is 4

bits. Assume the training set has 10 samples out of 16 combinations. After successful training, the evolved circuit has a $(2^{-4})^{(16-10)} = 5.96*10^{-8}$ probability of being 100% functional. Now, assume that we set aside P bits as "don't cares" so that O-P is the number of effective bits. The probability of obtaining a satisfactory circuit after training is now expressed by $(2^{-(O-P)})^{(N-T)}$. This implies that if the target function has a large number of "don't cares" and the test set is large enough, then the evolved circuit is likely to generalize.

With respect to the function approximation of a square root presented in [MillerThomsonGP98], a larger number of low-order output bits defined as don't cares (higher tolerance in approximation), i.e., a larger P value, results in a smaller O-P, which leads to a higher success rate. If the higher order bits can be mapped correctly, then the difference of output produced by low order bits is relatively small. However, this difference depends on the position of two related points on the function, e.g. more high order bits are needed for the function segment with steeper tangent lines than for flat segments. When the tangent line is near horizontal, we need only a few high order bits of input data to approximate the function outputs. For example, if we have two samples of input-output pairs, (10100, 11000) and (10111, 11001), and if evolution successfully maps 101xx to 1100*x*, we will have a perfect approximation for any input between 10100 and 10111 so long as both 11000 and 11001 are acceptable outputs.

To summarize the above feed-forward based EHW, generalization is only possible with discovery of "don't cares". A hash function evolution specifies "don't care" bits of inputs. In the case of 2-bit/3-bit multipliers, there are no "don't cares", so the only viable method is the exhaustive fitness evaluation. Discovery of "don't cares" not only in inputs but also outputs plays a key role for real-valued approximation.

4 Conclusion

We observe that the difficulties with EHW are rooted in the fact that there is no efficient algorithm to test a black box. EHW will be successful if we have a large test set and/or there are a large number of "don't care" bits. However, classical techniques may be better suited for these cases.

Yet there are advantages of EHW. If the full truth table is not available, EHW can be built as an adaptive system, adding more training samples over an extended period while online. If a target function is poorly understood so that traditional design techniques do not apply, and if it happens to have many "don't cares", and if it is difficult to identify them, then EHW may be a viable approach. Such applications may include feature extraction, data mining, and detecting signals in noisy data.

Acknowledgments

One of us (Foster) was supported on NIH grant F33 GM20122-01. Krings is supported on INEEL. We would like to thank Janet Holmberg for editing the manuscript.

References

[AgrawalSeth87] "Tutorial: Test generation for VLSI chips", Vishwani D. Agrawal, Sharad C. Seth, Manuscript, 1987.

[ChengAgrawal89] "Unified Methods for VLSI Simulation and Test Generation", Kwan-Ting Cheng, Vishwani D. Agrawal, luwer Academic Publishers, 1989.

[DamianiLiberaliTettamanzi98] "Evolutionary Design of Hashing Function Circuits Using an FPGA", Ernesto Damiani, Valentina Liberali, Andrea G. B. Tettamanzi, Proceedings, "Evolvable Systems: From Biology to Hardware", Second International Conference, ICES 98, pp.36-46, Lausanne, Switzerland, September 23-25, 1998 Lecture, Notes in Computer Science #1478, Springer, 1998.

[DamianiTettamanziLiberali99] "On-line Evolution of FPGA-based Circuits: A case Study on Hash Function", Ernesto Damiani, Andrea G. B. Tettamanzi, Valentina Liberali, Proceedings of the First NASA/DoD Workshop on Evolvable Hardware, pp.26-33, Pasadina, Calfornia, July 19-21, 1999.

[Fujiwara85] "Logic Testing and Design for Testability", Hideo Fujiwara, MIT Press, 1985.

[GP98] Genetic Programming 1998, Proceedings of the 3rd Annual Conference, Morgan Kaufmann, 1998.

[GP97] Genetic Programming 1997, Proceedings of the 2nd Annual Conference, Morgan Kaufmann, 1997.

[ICES98] Proceedings, "Evolvable Systems: From Biology to Hardware", Second International Conference, ICES 98 Lausanne, Switzerland, September 23-25, 1998 Lecture Notes in Computer Science #1478 Springer, 1998.

[Johnson96] "An Introduction to the Design and Analysis of Fault-Tolerant Systems", Barry W. Johnson, in Dhiraj K. Pradhan, "Fault-Tolerant Computer System Design", Chapter 1, p1, Prentice Hall PTR, 1996.

[MazumderRudnick99] "Genetic Algorithms for VLSI Design, Layout, & Test Automation", Pinaki Mazumder, Elizabath M. Rudnick, p16, Fig.1.12, Prentice Hall PTR, 1999.

[MillerThomsonGP98] "Evolving Digital Electronic Circuit for Real-Valued Function Generation using a Genetic Algorithm", Julian Miller, Perter Tomson, Proceedings of the 3rd Annual Conference, pp863-868, Morgan Kaufmann, 1998.

[MillerThomsonICES98] "Aspects of Digital Evolution:Geometry and Learning", Julian F. Miller, Perter Thomson, Proceedings, in Moshe Sipper, Daniel Mange, Andres Perez (Eds.) "Evolvable Systems: From Biology to Hardware", Second International Conference, ICES 98, pp.25-35, Lausanne, Switzerland, September 23-25, 1998 Lecture Notes in Computer Science #1478 Springer, 1998.

[NASA99] Proceedings of the First NASA/DoD Workshop on Evolvable Hardware, Pasadina, Calfornia, July 19-21, 1999.

[RudnickHsiaoPatel99] "Automatic Test Generation", E. M. Rudnick, M. S. Hsiao, Patel, in Pinaki Mazumder, Elizabeth M. Rudnick, "Genetic Algorithms for VLSI Design, Layout, & Test Automation", Chapter 6, pp.158-226, Prentice Hall PTR, 1999.

[YaoHiguchi96] "Promises and Challenges of Evolvable Hardware", Xin Yao, Tetsuya Higuchi, "Evolvable Systems: From Biology to Hardware", First International Conference, ICES 96, pp.55-78, Tsukuba, Japan, October 7-8, 1996 Lecture Notes in Computer Science #1259, Springer-Verlag, 1997.