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 problem from Very Large Scale Integration (VLSI) suggests that there is no efficient way to determine a training set which assures full correctness of an evolved circuit.

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 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 a truth table driven EHW is likely to succeed in two cases: 1) There are large number 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 problem is well studied. EHW performs circuit verification through fitness evaluation generation after generation 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? With this problem, we are evolving is the connection points on the AND and OR planes. Every feasible input can be recognized by the four lines, A, B, C, and D. Suppose that our evolved circuit produces correct output for (0,0), (0,1), and (1,0) which are emulated on A, B, and C in Fig 1. 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?

In the following section, we introduce the test vector problem and define an efficient test set generation algorithm.
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, a fault is one that occurs strictly within hardware. A test for fault is an input vector (a vector or a sequence of vectors) that will produce different outputs in the presence and absence of the fault. [AgrawalSethTut87] It is difficult to generate test vectors that uncover as many faults as possible. To see why, consider the line stuck type fault in Fig. 2. It passes all $2^{19}$ test vectors but one. The only test vector that detects 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 VLSI fault diagnosis.

![Fig.2 Line stuck example modified from [MazumderRudnick99]](image)

A combinational (non-sequential) circuit is interconnected-gates with no feedback loops, while a sequential circuit is with feedback loops which may be clocked or non-clocked. There are algorithms to test non-sequential circuits. [Fujiwara85] 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 there is no known satisfactory methods. For complex chips, scan design methods (by which flipflops can be initialized through a shift-register) should be the rule for testing. [AgrawalSethTut87] To overcome this complexity, [RudnickHsiaoPate99, MazumderRudnick99] discusses application of genetic algorithms to generate test vectors.

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, the 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 the input-output specification be $S$ and $I$ be a set of input vectors. Then, $O$, the corresponding set of the output vectors, can be obtained by $O = S(I)$. The most accurate testing is the exhaustive test. Let $I_a$ be the set that contains all the feasible inputs. Let $D(li, Oi) = 0$ if input $li$ outputs the specified output $Oi$ else $D(li, Oi) = 1$. The perfectly functioning chip is expressed as $f(I_a) = \sum D(li, Oi) = 0$ for all $li \in I_a$. A chip passes the test if $f(I_p) = 0$. We may prefer $I_p \subset I_a$ over $I_q \subset I_a$ if the reject rate is lower when the chip is tested with $I_p$ than with $I_q$. The reject rate is the ratio of the number of the defective chips over the number of the chips which passed the test. An efficient algorithm is one that generates $I_p$ such that there exists correlation between $f(I_p)$ and $f(I_a)$, i.e., when $f(I_p) = 0$, it is likely that $f(I_a) = 0$. 
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 problem, namely black box testing and that, therefore, there is no efficient algorithm to determine a set of training sample that likely verify the full correctness of an evolved circuit. 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. Let $D$ be Hamming distance for bitwise positional evaluation. ([Masner99, Kalganova99] used a distance function opposite of Hamming distance, which is total number of output bits – Hamming distance.) Assume the fitness is decided by $D$. The training samples provide no other information than existence of design flaws (faults), so this type of fitness evaluation is equivalent to a fault detection test with zero diagnostic resolution. Hence, fitness evaluation is black box testing, so there is no efficient algorithm that determines $I_p$.

Even if we did account for the structure of evolved circuits in the fitness function, there would be no general-purpose algorithm to generate $I_p$. It must be noted, though, that this does not mean that there is no preferred test set. We only know it after EHW is exposed to the exhaustive testing.

To illustrate this observation, we evolve a circuit which performs $(A \text{ AND } B)$. Assume that the evolved circuit is $(A \text{ AND } (B \text{ OR } \overline{B}))$. We have an equivalent case to a stuck-at-1 fault, $(A \text{ AND } 1)$, and the 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. Fitness function, which guarantees the circuit correctness, is very difficult to create.
3. It is difficult to know for EHW when correct circuit is 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. But 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 $I_p$ is used for the fitness evaluation, then the high correlation must be observed between $fp$ and $f$. 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.

**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, so 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 with evolving combinational circuit for 2-bit and 3-bit multipliers. Their conclusion is 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 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 for 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. Even if exact correctness of a
circuit needed by a multiplier is difficult, perhaps EHW can approximate real-valued functions. This is
Miller and Thomson’s question. [MillerThomsonGP98] Their results still indicate the limitation of a feed-
forward structure. Their best result is an approximation by a stair case function (Fig. 4), suggesting that the
least significant bits (low-order bits) became “don’t cares”.

![Fig.4 Approximation by a stair case function.]

If low-order bits have influence over the approximation more 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 gradient of square root (x). Indeed, if the gradient is changing rapidly, we
need to have fewer low-order “don’t care” bits in input numbers. When the tangent line becomes flat, we
need more low-order “don’t cares” bits in input. This implies that it is a necessary condition that we have to
sample more often in (0.0-0.1), but it is not sufficient. Because, if we have too many “don’t cares” in this
intervals, more samples will not increase the accuracy of 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, divide the domain into some intervals. Implement the truth table of
each interval on a different logic array, using some portion of input bits (low-order bits in this case). Use
the rest of the input bits to choose output of an appropriate logic array. Essentially, this is address decoding.
We can apply a circuit minimization technique, if needed.

Now, what is the chance for output to be correct when input is not in the training set? Assuming the
output for untrained input data is an uniformly distributed random variable, we have the following success
rate for producing 100% correct circuits, using a subset of the truth table for the training set:

\[
N: \text{number of total feasible inputs} \\
T: \text{number of input/output samples in the training set (} T \leq N) \\
O: \text{number of correct output bits that a trained EHW needs to produce}
\]

If the evolved circuit is 100% correct on 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 being correct output and there are
\((N-T)\) input-output pairs outside the training set. For example, the OR gate evolution example in the
introduction section (Fig. 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 \( O=1 \). We have \( (2^{-1})^{(4-3)} \).
So, the evolved OR gate has probability of 0.5 of being OR gate and 0.5 of being exclusive OR gate.

In case of 2-bit multiplier, there are 16 possible multiplier-multiplicand combinations. The output is 4
bits. If the training set has 10 samples out of 16 combinations, after all the training is done correctly, the
evolved circuit has \( (2^{-4})^{(16-10)} = 5.96 \times 10^{-8} \) probability of being 100% functional. 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 \( (2^{-O-P})^{(N-T)} \). This implies that if the target function has a large number of “don’t
cares” and a test set is large enough, then evolved circuit is likely to generalize. But in this case, it would
probably be more practical to design the actual circuit using classical techniques. In case of
[MillerThomsonGP98]’s function approximation, the more low-order bits of output are “don’t cares”
(higher tolerance in approximation), the larger \( P \) value, making \( O-P \) smaller which leads to a higher success rate. An upper bit pattern (high-order bits) remains the same for numbers between some two numbers. So, if this pattern is mapped to upper bit pattern of output values, output values will be relatively close. Especially, when the tangent line is near horizontal, we need only a few upper 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 \(1100x\), we will have a perfect approximation for any input between \(10100\) and \(10111\) so long as both \(11000\) and \(11001\) are acceptable outputs. Since untrained data needs to produce only one correct bit for \(x\) but this bit is a “don’t care”, we have \( O=P=1 \). This is not the case if the gradient is changing rapidly or the function is not monotonic. In short, the generalization ability for combinational circuits still depends on the distribution of “don’t cares” in this case in the output bits.

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 is 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.

**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. But 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 those, then EHW may be a viable approach. Such applications may include feature extraction, data mining, and detecting signals in noisy data.

**Acknowledgement:**

We would like to thank Dr. Axel Krings for valuable comments on fitness evaluation and test vector problems. We also would like to thank Janet Holmberg for editing the manuscript.

**References:**


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


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


