Improving Evolutionary Algorithm Performance on Maximizing Functional Test Coverage of ASICs Using Adaptation of the Fitness Criteria

Burcin Aktan
Intel Corporation
Network Processor Division
Hudson, MA 01749
burcin.aktan@intel.com

Garrison Greenwood
Dept. of Elec. & Computer Engr.
Portland State University
Portland, OR 97207
greenwd@ee.pdx.edu

Molly H. Shor
Dept. of Elec. & Computer Engr.
Oregon State University
Corvallis, OR 97331
shor@ece.orst.edu

Abstract- Adaptation of the fitness criteria can be a very powerful tool enhancing the feedback scheme employed in standard evolutionary algorithms. When the problem the Evolutionary Algorithm (EA) is trying to solve is changing over time, the fitness criteria needs to change to adapt to the new problem. Significant performance improvements are possible with feedback based adaptation schemes. This work outlines the results of an adaptation scheme applied to the maximization of the functional test coverage problem.

I. INTRODUCTION

Conventional automatic test equipment setup for ASICs uses a test pattern generator (TPG) that creates input test patterns for the device-under-test (DUT), and circuitry that records and analyzes the DUT output responses. This analysis checks for either manufacturing correctness or functional verification. The goal is to have *coverage*, which means the test pattern set exercises the DUT in ways so that any errors can be detected.

One test pattern is insufficient to yield 100% coverage so a set of patterns is needed. This problem thus differs from many optimization problems because there is no globally optimal solution to find; instead one tries to find a *set* of solutions out of the problem space. Intuitively coverage should increase as test patterns are added to the set, but that may not always be true because newly added patterns may merely test what is already tested by previously added patterns. Plus large sets can lead to unacceptable test times. Hence, the objective is to find a minimum cardinality set that maximizes coverage—an objective that has proven to be both tedious and difficult to attain, which means a heuristic design approach is required.

Evolutionary algorithms (EAs) can provide a means for designing these minimum cardinality test pattern sets. Two EA-based protocols have been used: (1) a direct approach where the EA explicitly evolves the test patterns—i.e., the EA is the TPG, or (2) an indirect approach where an EA evolves the TPG itself. Intel uses the latter approach in a prototype test bench for ASICs (see Figure 1). The TPG runs on a workstation and an externally supplied set of control parameters tells the TPG what type of test patterns are required to conduct the test [1]. For instance, during some functional tests specific registers in the DUT must be configured in particular ways prior to running the test. These control parameters tell the TPG what those configurations should be so that an appropriate set of patterns can be constructed. In practice, a test writer creates a set of control parameter vectors that specify a particular test to conduct and also the expected result. Each distinct test has a distinct vector. Since it is not uncommon to find thousands of control parameters may be needed for adequate coverage of some VLSI chips, this can be a very labor intensive and costly process. We replaced the existing method with an EA and the resultant event coverage was significantly improved [1] [2].

In this previous work the goal was to find a set of individuals that would maximize the overall coverage obtained. (Each individual defined a set of parameters for a microcontroller that deterministically created a test pattern set.) An individual's fitness was proportional to the amount of total coverage it provided, but this could be further increased by its incremental coverage—i.e., any new tests not covered by any other individual in the population. Both parameters had a user-defined threshold. The reason for using thresholds—as opposed to using a "best" individual—is to compensate for the unknowns in the problem and to minimize any biases the problem setup might impose. However, we have observed that this tends to overemphasize the sub-problem of improving individual test coverage instead of improving total coverage. Since one can quickly run out of the possible new cases, especially when the algorithm used gives an initial fast coverage ramp, the problem rapidly degrades to improving individual test coverage. The probability of hitting new cases decreases as the feedback from areas that have already been visited dilutes the population effectiveness.

Here we propose a new method of adaptation which still uses the notion of thresholds but adapts these with respect to the changes in the problem landscape. As less number of cases remain in the total unexplored space, the threshold that rewards new cases is relaxed (or decreased) to increase the rewards for new exploration. Similarly, the total case threshold is increased to reflect the fact that newer tests are reaching high levels of coverage so the rewards of getting large coverage decreases.

The paper is organized as follows. This work extends work

previously reported in [2], but for continuity the Intel test bench is described again in Section II. Section III describes the coverage based validation problem. The following section outlines the modifications made to the testbench and the EA description. Section IV details the threshold adaptation scheme based on EA performance. Preliminary results are presented in Section V. Finally, Section VI describes future research efforts.

II. Test Bench Description

The functional tests generated by this system are designed to exercise very complex interactions of multiple design features. For instance, a DUT may incorporate error correction logic. An example of such a test would be to perform large data block transfers (with high external error rates) to the DUT to stress the error correction logic. Coverage monitors would then be used to track the events at different microarchitecture levels. This data reflects the effectiveness of the test patterns.

Figure 1 shows the test bench setup used by Intel. Its major components are:

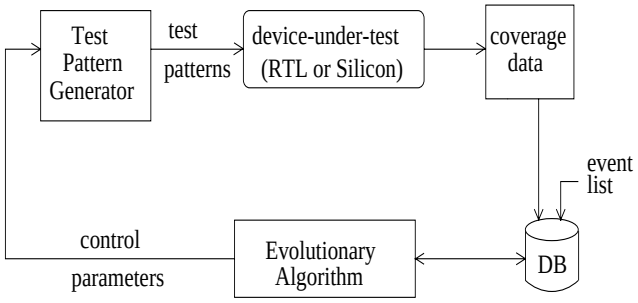

Fig. 1. The test bench configuration.

TPG:
This software uses input control parameters to create a set of test patterns for the DUT. The TPG is also responsible for generation of all interface signals needed to input these test patterns. It is implemented in C/C++ and runs on a workstation.

DUT:
The DUT receives input test patterns and outputs the corresponding responses. In the current configuration the DUT exists only as an RTL description—i.e., a VHDL source file. However, the actual silicon device may be substituted without changing the test bench. (See Section 5 of [2].)

Coverage Data:
The subsystem collects the DUT responses and determines which functional tests were evaluated by the current input test pattern. Essentially coverage is a metric for how well a design has been exercised. This subsystem performs the analysis function. We note that obtaining coverage and detecting errors is somewhat orthogonal—i.e., coverage is a goal only if there are no failing patterns.

Database (DB):
The database is a collection of all past test coverage. It takes two inputs: the set of all tests covered by the current test pattern set, and the list of all events (functional tests) that must be covered before the DUT is considered validated. The first input is provided by the coverage data subsystem while the second input is provided by a validation engineer.
The database takes the input from the coverage data subsystem and constructs the intersection with the set of all functional tests covered so far. Any uncovered tests that are not in the intersection are deemed the *incremental coverage set*. The database computes two numbers: the percent of all tests covered by the current test pattern set only, and the total percent of tests now covered between all past test patterns and the current test pattern set. These two percentages are the phenotype provided to the EA. However, although the database is updated with the phenotype values, these updates may later be removed if the EA indicates the genotype did not survive the selection process (see Section 3 of [2]).

EA:
The EA is responsible for evolving the set of control parameters needed by the TPG to create the test patterns. (Specific details on the EA can be found in Section 3 of [2]). This software runs on a workstation.

Note that the interconnection between the EA and the database is bi-directional. The database furnishes the phenotype information based on all past test patterns plus the current test pattern. However, the EA may decide not to keep the genotype that ultimately produced the current test pattern set. In that case the database would need to purge the current test pattern set so that the total coverage percent value accurately reflects the coverage provided by the genotypes present in the population.

The TPG and EA may or may not run on the same workstation. If they do not run on the same workstation, the exchange of information—i.e., control parameter set and phenotype values—is accomplished via message-passing.

It should be emphasized that the legality of test sequences is a very important factor for obtaining valid user-defined event coverage. Thus this system yields itself well to user defined and predefined coverage methods. While the indirect method gives much more flexibility and degrees of freedom to the EA it also might increase the problem difficulty since the control is moving to a higher level. TPG design and the control parameters need to be carefully considered while deploying such a system. The main assumption we make is that the events targeted are not only controllable through the TPG, but they are

also observable through event coverage monitors.

III. COVERAGE

The need for user-defined coverage thresholds—and the benefit derived by their self-adaptation—requires a firm understanding of the difficulties involved in achieving coverage. This section describes those difficulties.

The goal of ASIC validation is to verify the output and the states of a certain design behave in a certain desired way given the input to the device under test. The verification result is correct if the device performs within the specifications for all possible (legal or allowable) input and state combinations. Naturally it is not feasible to dynamically test all of these combinations for large ASICs. Coverage is a test progress indicator reflecting the percentage of the total space verified with the process given above. The goal of testing therefore turns into maximizing the amount of space verified or covered. The art of validation is in specifying a subset of the total coverage space that is representative of the various intricacies of the device under test. In most cases coverage is restricted to static program-based coverage events such as code coverage metrics (statement, branch,etc.). In other cases functional, typically user-defined coverage is used [3]. These are sets of events that are deemed important by the validator to trigger for effective verification of the DUT and can be static or temporal.

Given the uncertainties involved in the coverage monitor definition process, whether defined automatically (code metrics) or by a user, it may not be possible to get full (100%) coverage of the defined events during the testing of an ASIC. Especially if the coverage event space includes temporal descriptions, the reachability of all the events are not well-defined. Most of the time these events are described at lower levels of the design blocks and may not be fully controllable from the periphery of the device under test. The problem arises from the fact that there is not always a clear mapping from the externals to the internals of the design especially for highly complex ASICs. There is no clear way for a designer to identify whether a defined temporal coverage event is reachable given the external protocol. The boundary conditions of the available control which essentially is the protocol legality rules may not allow for the described functional coverage events to be reached. Although from a subblock-only perspective the described events are clearly reachable.

Another reason for the limited reachability is the interactions between various design subblocks. A subblock may never drive a certain set of inputs to a given design block regardless of the fact that the block may accept that input. These situations are encountered frequently when reuse of design blocks is practiced.

Under some circumstances the coverage space description might be bloated. Given the simulation speed of a given design it may not be possible to reach all events in a reasonable time, such as testing correct behavior of a 64 bit floating point unit. Feedback based techniques might still do a better job here than other simulation based approaches like directed or random testing; however, it is best that these blocks be verified by formal methods.

Coverage is a simulation completeness indicator best used for control or datapath logic and should not be used to detect the effectiveness of simulation based methods for coverage sets derived from combinations of all possible input and internal states.

It is also likely that the coverage event descriptions themselves are specified erroneously which is common whenever human factor is involved. As a matter of fact it is typical for a designer to specify a cross product of two sets and not exclude illegal or impossible combinations from the resultant set.

Given all these ambiguities it is not likely that a collection of coverage events which is specified as the target set for a coverage-driven simulation-based test system be complete. The maximum possible coverage (as a percent of total set) is unknown. This poses yet another problem for specifying the feedback algorithm—i.e., coverage thresholds—for the EA-based test generation system.

IV. EA DYNAMICS AND THE ADAPTATION SCHEME

The EA is used to generate the control parameters that drive the TPG. Every distinct control parameter set (a test) generates a distinct set of test vectors to drive an RTL description of the device under test. If we consider each individual control parameter to be a gene, then the genome is an encoding of an entire control parameter set. The genome size depends on the exact number of control variables needed by the TPG. (In practice, the number of control variables is specified by the validation engineer.) The objective is to generate a set of tests—i.e., a collection of control vectors—that maximizes the functional test coverage. The EA is implemented as follows:

1. Let μ_t be the number of parents in generation t. Randomly generate an initial population μ_0 genotypes. Initialize the incremental and total coverage thresholds (C_I, C_T).
2. Randomly select two parents and create a single offspring using stochastic reproduction operators. Repeat this process until k_t offspring have been created. Send all offspring to TPG to create test patterns.
3. Receive phenotype values from database. If phenotype values exceed total coverage and/or incremental coverage thresholds, add the corresponding genotype to the population and increment μ_t. Otherwise, discard that genotype and leave μ_t unchanged. Inform database of the decision. Repeat this process for all k_t offspring.
4. Modify the thresholds based on final state of the database and the population at item 3.

5. Let $t = t + 1$. If termination criteria not met, go to step 2. Otherwise, exit.

Each genotype contains M specific control parameters stored as 32-bit integers, where M depends on the DUT. During generation t, μ_t parents generate a set of k_t offspring. The initial population has $\mu_0 = 2$ randomly generated parents, but μ_t grows as t increases. (This aspect will be explained in more detail shortly.) Offspring are created from a randomly chosen pair of parents using uniform crossover and each bit is mutated with a 5% probability. This rather high mutation probability was chosen because the initial population is so small—any smaller probability would make mutation ineffective. The probability is gradually reduced as the search progresses because the population size grows.

The control parameter vector derived from the decoded offspring is then input to the TPG. The test run data is collected and analyzed for coverage, thereby determining the usefulness of the test sequences generated by the TPG. The extent of the test sequence coverage is the basis for assigning a fitness value.

The phenotype consists of two parameters: the total test coverage (C_T^i) and the incremental test coverage (C_I^i) produced from genotype i. C_T^i indicates what percent of the total tests are covered by genotype i; C_I^i indicates the number of new tests not covered by any other genotype in the current population. There is a user-defined threshold for both parameters. Genotypes with a phenotype that exceeds these thresholds are considered highly fit. In fact, these are the only genotypes that are allowed to survive.

Each generation $k_t = 20$ offspring are created independent of the population size (μ_t). Fit genotypes are selected deterministically for survival, which means the population grows by $\mu_{t+1} = \mu_t + \text{fit}(k_t)$ where fit(k_t) is the number of genotypes exceeding the C_T and C_I thresholds. In order to make sure that the EA has accurate information about the remaining solution space, it is important to maintain a consistent database state that reflects only the overall coverage of genotypes currently in the population. Therefore the phenotypes of non-surviving genotypes are deleted from the database.

The EA terminates after an acceptable coverage level is reached or a prescribed number of offspring have been processed. In the former case, a power set of μ_t is formed and the minimum cardinality subset with maximal coverage is retained as a test suite; all other genotypes are discarded. In the latter case all genotypes are kept to be used as regression suite for future operations.

Given the basic algorithm described above (see [2] for more details), a feedback scheme has been developed using heuristic observations and adaptive gain scheduling. Both the incremental and the total thresholds are adapted at the same time. The evaluation is done at every generation.

$$C_{total\_new} = C_{DB} - C_{T0}e^{(-\alpha * PSZ * C_{TB}/C_{DB})}$$
$$C_{inc\_new} = C_{I0}e^{(-\beta * PSZ * C_{TB}/C_{DB})} + 1$$

C_{DB} : cardinality of the total target space of coverage events
C_{TB} : cardinality of the space already covered by tests
$C_{DB} - C_{T0}$: initial total coverage threshold
$C_{I0} + 1$: initial incremental coverage threshold
PSZ : current population size
α : fixed positive sizing constant
β : fixed positive sizing constant

Total threshold imposes a selection pressure on the EA that improves the overall performance of an individual. Naturally increasing the threshold as total coverage and the population size increases puts pressure on the EA on the overall to improve the performance, not only by finding better individuals but also by trying to keep the total population low. The adaptation form also strives to achieve this by penalizing the population size as well as the total coverage measured. On the other hand the problem difficulty for finding new coverage is directly proportional to the incremental coverage. As a matter of fact, if this value is set high there may not be an individual in the search space that might satisfy the requirements. Especially there is not much point in setting the threshold value higher than the total available coverage remaining (difference between the maximum attainable value and the current database value). This keeps an artificially high mark for the EA to target than what is really necessary to solve the problem.

We have chosen a scheme which rewards the EA for finding new solutions by adapting the incremental coverage. It can be viewed that the policies outlined here for both coverage platforms are contradictory. While the algorithm is being "penalized" for population size and coverage improvements in the total threshold adaptation, it is being rewarded for the same factors in the incremental threshold case. However, considering the dynamics of the problem, this is exactly what is intended.

There are really multiple objectives in this problem. If the objective was to reach a single test with the maximum possible coverage, redefining the problem as a functional optimization problem, there would be no need for a thresholding scheme. In fact, elitist selection would be able to demonstrate suitable results and global convergence [4]. However, due to the nature of the solution set a thresholding scheme is deemed appropriate. The best individual in this sense may not be able to hit all available coverage. As a matter of fact, in the case where the available coverage has reduced to a few events, the only individual solving the problem might be an individual that is not fit with respect to the original fitness definition. This is the reason fitness (thus the thresholds) need to vary with the EA progress.

V. RESULTS

The trial runs were conducted on two sample coverage problems. One of these problems is maximizing coverage on a control logic block consisting of a complex state machine and its various interactions. There are total of 745 cases monitored for coverage on this block. The second monitor targets various cases around the datapath of the component under test and is composed of 256 total cases. The controllability of these cases have been assessed as relatively low due to the difficulty in reaching high coverage numbers using conventional methods. The performance results given here of the EA are of typical runs averaged over both problems. The following figures demonstrate a dramatic improvement in the number of offspring evaluated and the resultant population size. Adaptive thresholds (solid lines) allowed 100% coverage to be reached in 50 generations and 1000 offspring evaluations with a final population size of 43 individuals. In contrast fixed thresholds (dashed lines) stagnated at a maximum (average) coverage level of 95.4% reached in 42 generations and 840 offspring evaluations with a population size of 112. The fixed thresholding scheme had not been able to improve the maximum coverage when the experiment was stopped at 12000 offspring evaluations, 600 generations and population size of 133. Just as a comparison, in the case of these monitors classical random testing methods have been able to reach about 78% maximum coverage with roughly about 12000 test runs.

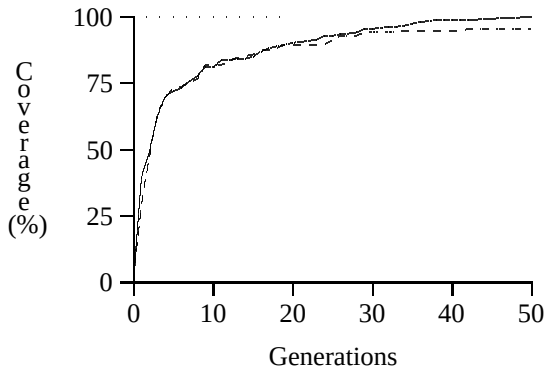

Fig. 2. Coverage provided by entire population versus number of generations.

VI. CONCLUSION

It should be noted that the adaptation scheme provided here is very rudimentary and cannot be considered a full control law that would apply to all coverage problems unilaterally. It was intended as a demonstration of the concept of adapting the fitness rule to help the EA performance on changing surfaces. The effectiveness can be improved for general coverage problems if a feedback based control law is derived based on the dynamics of the search process. After these initial results possible future directions include derivation of this control law. Note that the adaptation scheme used can also be considered a feedback control law but so far in its current form requires significant amount of tuning based on the specific coverage space defined.

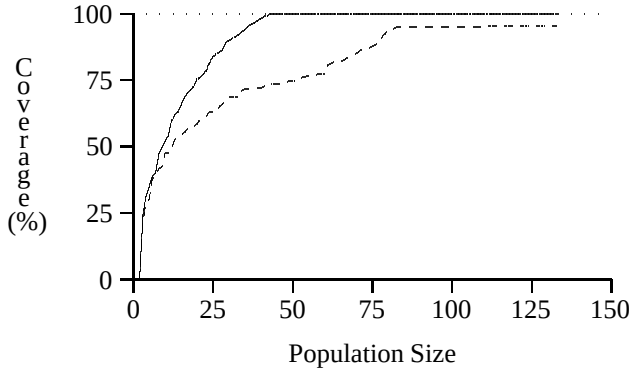

Fig. 3. Population size vs. coverage during the same run depicted in Figure 2.

Nevertheless, improving the coverage collection and ramp process on ASIC validation remains to be a big challenge and a topic for continuing research.

References

[1] B. Aktan and G. Greenwood. Darwin: A genetic algorithm based parameterized VLSI testing system. Tech. Report No. 2000/01, ECE Department, Portland State Univ., Portland, OR 97207, 2000.

[2] B. Aktan, G. Greenwood, M. Shor, and P. Doyle. Maximizing functional test coverage in ASICs using evolutionary algorithms. *Proc. CEC2001*, 2001.

[3] R. Grinwald, E. Harel, M. Orgad, S. Ur, and A. Ziv. User defined coverage - a tool supported methodology for design verification. *Proc. DAC98*, 1998.

[4] G. Rudolph. Finite markov chain results in evolutionary computation: A tour d'horizon. *Fundamenta Informaticae*, 35(1-4):67–89, 1998.