Efficient Mutation-Analysis Coverage for Constrained Random Verification

Tao Xie[1], Wolfgang Mueller[1], and Florian Letombe[2]

[1] University of Paderborn /C-LAB, Fürstenallee 11,
33098 Paderborn, Germany
{tao,wolfgang}@c-lab.de
[2] SpringSoft, 340 rue de l'Eygala
38430 Moirans, France
florian_letombe@springsoft.com

Abstract. Constrained random simulation based verification (CRV) becomes an important means of verifying the functional correctness of the increasingly complex hardware designs. Effective coverage metric still lacks for assessing the adequacy of these processes. In contrast to other coverage metrics, the syntax-based Mutation Analysis (MA) defines a *systematic correlation* between the coverage results and the test's ability to reveal design errors. However, it always suffers from extremely high computation cost. In this paper we present an efficient integration of mutation analysis into CRV flows, not only as a coverage gauge for simulation adequacy but also, a step further, to direct a dynamic adjustment of the test probability distribution. We consider the distinct cost model of this MA-based random simulation flow and try to optimize the coverage process. From the probabilistic analysis of the simulation cost, a heuristics for steering the test generation is derived. The automated flow is implemented by the SystemC Verification Library and by Certitude™ for mutation analysis. Results from the experiment with an IEEE floating point arithmetic design show the efficiency of our approach.

Keywords: Verification Coverage, Constrained Random Verification, Mutation Analysis.

1 Introduction

Simulation has still a dominant role in the verification of the functional correctness of electronic and embedded systems. Today, designs are increasingly complex, on the one hand, driven by the need to fill the Moore's Law driven capacity of integrated circuits and, on the other hand, thanks to our eager promotion of the design capability. Model-based, system-level methodologies and more extensive IP reuse are adopted. However, our verification ability lags behind. By [13], many current development projects have already a verification team sized over 2:1 to the design team.

To accommodate this growing complexity of designs, random simulation is applied to ease the labor cost of writing directed test vectors. It generates test input automatically and, therefore, reinforces the scalability of simulation-based approaches.

M. Hinchey et al. (Eds.): DIPES/BICC 2010, IFIP AICT 329, pp. 114–124, 2010.

Constraints and biases on the inputs domain can be imposed additionally on the randomness to overcome its shortage and exercise the design more extensively. By the nature of focusing on the boundary, the constrained random simulation based verification (CRV) processes need particularly an effective metric, or a suite of them, to assess their adequacy. On the whole, the effectiveness and reliability of such an adequacy guard should be judged by its ability to detect potential design errors.

Code coverage like *statement coverage* or *branch coverage* is the most intuitive metrics and long being used for both software testing and hardware design simulation. It is also supported by most Hardware Description Languages (HDLs) simulation tools. However, though a necessary step, high code coverage solely reflects the completion progress very limitedly. The *functional coverage* mechanism [20, 21] provided by the recently popular hardware verification languages like *SystemVerilog* requires the explicit definition of variable value ranges to hit, which is then recorded during simulation. A major drawback is the enforcement of verification engineers to thoroughly understand the design and extra effort to define the coverage points or applying libraries as a *subjective* metric.

Originally proposed for software testing, Mutation Analysis (MA) is a fault-based test data selection technique. A so-called *mutation* is a *single syntactic* change to the original program source code under test, such as replacing an *add* arithmetic operator with a *minus*, as an artificially injected bug. Such a program mutant is said to be *killed* by a test when it under the test produces a different output from that of the original program. When applied to generate or assess an adequate set of tests, MA creates a bunch of mutants from the original program, each by a single, different mutation. Then the percentage of the mutants killed by the testing process is measured as the adequacy of the coverage.

The possible syntactic changes, as *mutation operators*, obviously depend only on the description language of the objects. As testing is a requirement with many computer-aided artifacts, subsequent research work extends mutation analysis to other languages. Particularly, HDLs share similar syntaxes with programming languages and have alike execution means as software. As an industrial EDA tool, Certitude™ [7,14] from *SpringSoft* implements the mutation analysis mechanism on Verilog and VHDL. Mutation operators specifically for HDLs are defined like:

```
      sign <= opa(63)  xor  opb(63) ;
Δ     sign <= opa(63) xnor opb(63) ;
```

where Δ is by convention used to indicate the only mutated statement.

Design errors, at various levels of descriptions, are essentially any of its deviation from the specification. Different from other fault-based methods like [10,11,12] for test data selection, MA defines a *systematic correlation* between the coverage results and the test's ability to reveal design errors. This is done in two steps. First, a coverage point is defined directly as if a test exposes a potential mistake by the designer. Second, MA hypothesizes and experimentally establishes a *coupling-effect* [1,4], which states that a set of tests capable of killing those mutants with simple faults injected will also be effective at exposing other more complex errors. As such, MA serves as a reliable guard for testing or verification processes ensuring the detection of design errors.

However, though with extensive study, mutation analysis suffers from extremely high computation cost, which becomes the main challenge for any MA application. Considering a hardware design with L lines under the simulation which is guarded by a mutation analysis with M mutation operators, we will have a mutant set with approximately a size $(M * L)$ as the coverage metric. M as a constant and assuming the simulation cost linear to the design size, calculation of one test case against the metric will have a cost to $O(L * L)$ and the overall coverage evaluation a cost to $O(T * L^2)$ for T test cases. This is a high computation requirement with increasingly complex designs. With a CRV flow, the situation is even worse, since T will be enlarged as randomness is used to reach the adequacy. This cost efficiency issue should be addressed. Therefore, in this work, we experiment with the use of MA coverage for CRV, consider the accurate cost model of such a flow, and try to develop an efficient algorithm to tackle the coverage cost problem.

2 Related Work

MA is a fault-based verification technique. Analogously, the fault models at gate-level, e.g. the stuck-at, is used to guide the selection of product test data for exposing defects that may be introduced during the manufacturing processes. Manufacturing defects can be viewed as the deviation of a product circuit from the designed structure. Automatic test pattern generation (ATPG) algorithms like PODEM (Path-Oriented Decision Making) and FAN (FAN-out-oriented test generation algorithm) generate test vectors targeting the gate-level modeled faults. Although theoretically, when hardware designs are concerned, we can always translate higher level faults to gate-level and apply an ATPG there to generate test vectors that correspondingly expose the high-level faults. This mapping imposes high complexity and inefficiency, especially with complex designs. Successful application of ATPGs relies on *Design-for-Testability* techniques [8], with which ATPG algorithms can assume a small portion of the circuit as their input, and output effective tests for the *structural testing*. In contrast, simulation vector generation for functional verification, similar to *functional testing*, concerns the overall functionality of the design. They are supposed to take the whole design as the algorithm input.

Fault models for automatic test generation at higher levels, such as the behavioral level or RTL, has also been considered in [9,11], for instance. [9] also mentions the use of MA for hardware designs. The designs are transformed to FORTRAN programs and then fed as the standard input into software mutation analysis tool Mothra [2]. Faults analysis and tests generation are then the task of Mothra [2,5]. However, neither the language translation is efficient, nor does the Mothra system handle complex objects. [11] first transforms the original and faulty VHDL descriptions to Binary Decision Diagram (BDD) based representations, with a different BDD for each output bit. Then each pair of these bits is compared to extract the symbolic test vector. Here, scalability is the main challenge.

Other coverage metrics have been used to direct random test generation. Code coverage, more specifically branch coverage is considered in [15]. A Genetic Algorithm, with the branch coverage degree as a *fitness* measurement, is developed to guide simulation sequences generation and evaluated on some VHDL design. The method in

[17] begins with a test planning and the coverage is defined as the amount of pre-planned verification tasks that have been simulated, e.g., specific transactions from a CPU unit. Then an evolving Bayesian Network is constructed to model the correlation between test generation directives and the coverage. [16] employs a so-called *tag-coverage*. A *tag* is defined as some symbolic disturbance to a variable value assignment and is said to be covered if this disturbance is propagated to any observation point in the simulation. A Markov Chain derived from the hardware design is built and tuned according to this tag-coverage. Probability distribution of the random input is then optimized by the chain.

We consider the distinct cost model of a MA-based random simulation flow and try to optimize the coverage by dynamically adjusting the probability distribution of the random test generation.

3 Mutation-Analysis Directed Constrained Random Simulation

Our CRV flow is built with three components. First, the SystemC Verification Library (SCV) [18] presents a standard constrained-random test generation (CRTG) facility, with a handy interface for defining input constraints associated with weighted ranges. Second, the ModelSim™ simulator is employed due to its ability to simulate mixed SystemC/VHDL/Verilog designs. Third, as a key enabling factor, the Certitude™ defines a comprehensive model of design errors on VHDL and Verilog for mutation analysis.

Originally, the identification of mutants is defined by observation and comparison at the boundary of the object under test. Another concept *weak mutation* is developed in [3] by allowing this observation at any intermediate points between the mutation point and the design output, e.g. immediately after the execution of the mutated expression, or statement. In contrast, the classical MA with the mutant identification at the output, can be denoted as *strong mutation analysis*. In Certitude™, the option for distinguishing mutants' behaviors ranges from directly after the mutation line, to any subcomponent ports, and to the top design output ports. Further, Certitude™ applies another so-called *schema-based mutation* technique [6], which encodes all independent mutants into a single design copy. Compilation of mutants becomes a one-time job and, at the same time, the statement-based weak mutation analysis for all mutants, i.e. whether a mutant produces a locally different behavior, requires only *a single simulation* of this instrumented design by in-time comparison with the execution of the original statement.[1]

3.1 The Simulation Flow and Its Cost

Figure 1 depicts the general design flow. The three bold arrows represent simulations and behavior monitoring, either on the original DUV (Design Under Verification) or the mutants. We start with some initial test constraints for the DUV and a CRTG. At the beginning and any time the DUV is changed the design files are copied and instrumented by the mutation operators. This process is determinate and the product is

[1] Certitude introduces a layer called *functional qualification* [19], which gives the test bench a good credit when its monitor is vigilant enough and flags a failure when a mutant does produce a distinguishable behaviour at the observation point.

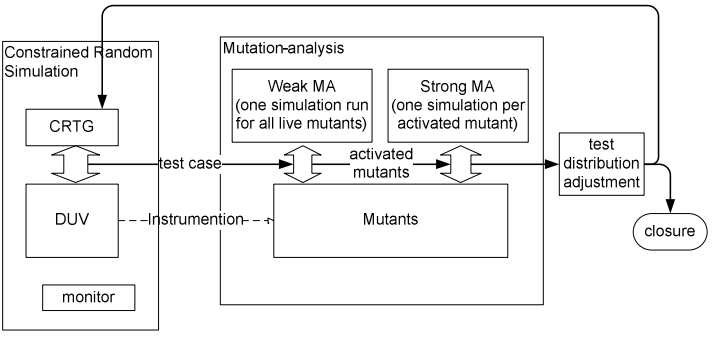

Fig. 1. The CRV loop and closure directed by the coverage of mutation analysis

the mutants. We should also note that some mutants remain *functionally unchanged,* which always produce the same output as in the original program. These so-called *equivalent mutants* are eliminated from the mutants box. Each time the CRTG generates a test case and the monitor flags a pass to the DUV simulation result, this test case is sent to be assessed by the mutation analysis.

In this work, we use *strong mutation analysis results as the final coverage measurement*. However, putting a weak mutation phase at the front saves simulation effort definitely and significantly, as herewith we only need to simulate the *locally already exposed* mutants, though each one against the test case. As previously described, this requires only *one* extra simulation under Certitude™.

Killed mutants are removed from the mutants box. The accumulated results from mutation analysis are used for a runtime calculation of some reasonable adjustment to the test distribution, and if any, to be fed back to the CRTG. At the same time, a certain percentage of dead mutants is used to break the loop and end the flow. This does not apply a 100% killing of mutants, which can rarely be the case for complex designs. Without loss of generality, we consider here the cost model for an 100% mutant-killing coverage.

Although constraint solving, source code instrumentation for creating mutants, and equivalent mutants identification all consume computation resources, most time of the flow will be spent on simulation. Basically this is due to the nature of mutation analysis of feeding each test case to individual mutants for simulation. Since the single simulation cost scales linearly with the design size, the complexity of the proposed flow is decided by the number of simulation runs. In the following, we make a detailed analysis on the required simulations. Consider D the design in the flow under verification and $M(D) = \{M_1, M_2, \ldots, M_N\}$ as the set of non-equivalent mutants generated. At any time in the CRV loop we have a probability distribution φ over the input variable value domain $I(D)$. With MW_i the random variable for the weak mutation analysis outcome on M_i and MS_i for the strong analysis, for any φ we can define $pw_{\varphi,i} \stackrel{\text{def}}{=} \Pr(MW_i = 1)$ and $ps_{\varphi,i} \stackrel{\text{def}}{=} \Pr(MS_i = 1)$.

Then the simulation runs on M_i in the strong mutation phase can be represented by a random number X_i as the times of $(MW_i = 1)$ happening until the first success of $(MS_i = 1)$. Noting that for $\forall i\ (MS_i = 1) \subseteq (MW_i = 1)$, we can derive by geometric distribution the expected value of X_i as

$$E_{\varphi}(X_i) = \frac{1}{\Pr(MS_i = 1 | MW_i = 1)} = \frac{1}{\frac{ps_{\varphi,i}}{pw_{\varphi,i}}} = \frac{pw_{\varphi,i}}{ps_{\varphi,i}}$$

denoted by $Cost_{\varphi,i,strong}$. Further, as the weak MA phase costs one simulation for all remained mutants, $Cost_{\varphi,weak}$ as the total weak mutation runs, i.e., the flow loop count until the last live mutants being killed, can be simply estimated by $\max_i(1/ps_{\varphi,i})$. Therefore, we calculate $Cost_{\varphi}$ as the total simulation effort needed to kill all mutants under distribution φ, expectedly, as

$$Cost_{\varphi} = Cost_{\varphi,weak} + \sum_{1 \le i \le N} Cost_{\varphi,i,strong} = \max_{1 \le i \le N}(1/ps_{\varphi,i}) + \sum_{1 \le i \le N} \frac{pw_{\varphi,i}}{ps_{\varphi,i}} \quad (1)$$

I other words, a high activation rate of mutants with low propagation probability leads to high simulation costs. For instance, assuming a set of 100 design mutants and under a certain test distribution φ_1 each of $pw_{\varphi_1,i}$ having a same value of 0.01, and all $ps_{\varphi_1,i}$ a value of 0.005, we can calculate then $Cost_{\varphi_1}$ as a cost estimation of 400 simulation runs. For another φ_2 with all $pw_{\varphi_2,i}$ having a value 0.5 and $ps_{\varphi_2,i}$ 0.01, *though the mutants have a higher probability to be exposed*, they give more costs with a total of 5100 simulation runs, expectedly.

As another example, under selective mutation operators an RTL FFT design module with 29811 lines derives already $M(D)$ of 26758 non-equivalent mutants. $Cost_{\varphi}$ becomes extremely high with growing design sizes. Symbolic methods traditionally used for mutation-based test generation [5, 2] assume at most time $(MW_i = 1) \rightarrow (MS_i = 1)$, i.e., a mutant when activated then propagates to the output. This is not the case if we apply mutation analysis to the simulation of large designs. This cost problem can be addressed and, next, we present a heuristics as our first effort towards an efficient mutant-killing coverage for the CRV flow.

3.2 Dynamic Distribution Adjustment for More Efficient Coverage

We note that Equation (1) can be simply applied to a subset of $M(D)$. At some point during the CRV flow in Figure 1, $M(D)$ is reduced by dead mutants and only those hard-to-kill under the current test distribution are left. Then if we adjust φ, a newly estimated computation cost is defined in the same manner. This adjustment should be based on the cost estimation in Equation (1), so as to *reach more quickly a high mutant-killing coverage*. For this, a heuristics as described in Figure 2 is developed.

The algorithm for the heuristics assumes that the test input domain can be segmented into some discrete ranges. Then, basically, it *utilizes the past analysis information to estimate the effectiveness of those ranges and re-distributes the probability*.

Mutation analysis results $\sum_{1\le i\le N} ms_i\,(tc)/\sum_{1\le i\le N} mw_i\,(tc)$ as given in Line 3 are used to represent the $\sum ps_i/pw_i$ under the current distribution. The effectiveness is then measured relatively to a value $effective\_^{ps}/_{pm}$ through Lines 6 to 11 and used to flag a range as effective by adding it to an *effective_distrib* array, if its n_{ms}/n_{mw} surpasses $effective\_^{ps}/_{pm}$. If no mutant is killed, we add it to an *ineffective_distrib* array. Initially $effective\_^{ps}/_{pm}$ is assigned a parameter value $initial\_effective\_^{ps}/_{pm}$. This relative measure always relaxes in Line 19 as live mutants decrease and the remaining ones become harder to kill.

Heuristics *#for the distribution adjustment box in Figure 1.*

Parameters: $starting\_heuristic$, $initial\_good\_^{ps}/_{pm}$, $adjustment\_threshold$

#Assume the input value domain can be segmented as a set of ranges $I(D):\{I_1, I_2, \dots, I_H \mid \cup_i I_i = I\}$. *For each mutant* M_i, $mw_i, ms_i : I \to \{0,1\}$ *as the weak and strong mutation analysis result, respectively.*

(1) $effective\_^{ps}/_{pm} := initial\_effective\_^{ps}/_{pm}$
(2) $mark := 0$
(3) $\left(tc \in I_k, n_{mw} := \sum_{1\le i\le N} mw_i\,(tc), n_{ms} := \sum_{1\le i\le N} ms_i\,(tc)\right)$
as received from CRTG and mutation analysis
(4) Enter the following loop if the previous total happening of event ($n_{ms} = 0$)
already reaches $starting\_heuristic$.
(5) **Loop** until the killed mutants reach a certain ratio predefined,
or the verification cost budget is reached
(6) **If** $\left(n_{ms}/n_{mw} \ge effective\_^{ps}/_{pm}\right)$
(7) Add pair $(k,\ n_{ms})$ into an array $effective\_distrib$
(8) **Elseif** $(n_{ms} = 0)$
(9) Add k into another array $ineffective\_distrib$
(10) Increase $mark$ by 1
(11) **End if**
(12) **If** $(mark \ge adjustment\_threshold)$
(13) **If** $effective\_distrib$ is not empty, set test distribution as:
(14) For each (k', n'_{ms}) in $effective\_distrib$, set
$\Pr(I_{k'}) := n'_{ms}/(sum\ of\ all\ \ n_{ms}\ in\ effective\_distrib\ )$
(15) **Else**, set the distribution as
(16) Uniformly distributed on $(\cup I_{k'} \mid k' \notin ineffective\_distrib\ )$
(17) **End if**
(18) Empty arrays $effective\_distrib$, $ineffective\_distrib$, set $mark := 0$
(19) Lower $effective\_^{ps}/_{pm} := effective\_^{ps}/_{pm}\ / 2$
(20) **End if**
(21) **End loop**

Fig. 2. Heuristics for mutant-killing by utilizing past analysis information

This establishes a *macro* relation between the test input domain and the overall mutant-killing. Our hypothesis is that if an input range is assessed to be effective at killing mutants, we expect it to be further capable of killing mutants and adjust the test distribution towards it. Otherwise, the distribution is steered away. Lines 13 to 17 are for this purpose. After this adjustment the arrays are emptied.

Furthermore, a threshold parameter $adjustment\_threshold$ is defined to trigger an adjustment procedure in Line 12, when the loop iteration killing none of the mutants, recorded by a variable $mark$, reaches this amount. We have not considered an optimal setting for this parameter. It could be set initially to a value of 1 and also loosens while the remaining mutants become more stubborn.

Since at this level 100 percent killing of the mutants could be infeasible under some time restriction, the whole flow should also be controlled by a *simulation cost budget* which terminates at a reasonably high *certain ratio* of killed mutants.

The presence of a $starting\_heuristic$ parameter is the last to notice. The dynamic distribution adjustment is not necessary at the beginning phase of the CRV flow, when many of the easy-to-kill mutants are still alive. This trigger is controlled by parameter $starting\_heuristic$.

4 Results

We have chosen a VHDL implementation of the IEEE binary double-precision floating point arithmetic unit from *opencores.org* for our experiments in our MA-directed CRV flow. Figure 3 shows the architecture of that example.

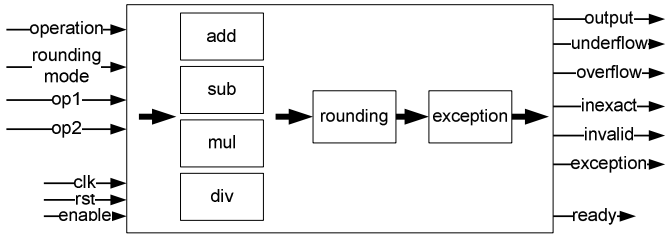

Fig. 3. The floating point arithmetic under CRV

The test domain of the DUV is composed of its major input ports including the arithmetic operator, rounding mode, and two operands. To execute the heuristics, this domain is segmented by the number classification of the operands, *norm*, *infinity*, *denormal*, etc. For strong mutation analysis, the mutant-distinguishing point is set at the output ports of the core including the arithmetic output and exception signals. Further, though not the focus of this experiment, a software implementation of the floating point standard is used in the simulation as an oracle, to compare and assess the correctness of the DUV output.

Figure 4 gives a summary of the experimental results. The design with a total number of 2492 lines-of-code derives 2257 mutants, which have the mutation points scattered over all the major sub-components. 58 of them are detected by the tool as

equivalent mutants. We then executed the flow in Figure 1 with two setups, one fixed with a uniform input distribution, another *also starting with a uniform distribution* but self-tuning directed by the heuristics. Each setup is executed twice for 200 loop iterations, i.e., 200 test cases as shown in the figure to provide more evident data. The adjustment threshold parameter of the heuristics is set to 1, and $initial\_good\_^{ps}/_{pm}$ set to 0.01. Our studies also compared the simulation time with and without memory utilization and found no significant difference. To conduct the two experiments with uniform distribution, it took us 89460 and 101681 simulation runs for about 85 and 96 hours, respectively, which killed 1301 and 1289 mutants. The other two experiments with the heuristics took 78460 and 78849 simulations for around 75 and 77 hours with a mutant killing coverage of 1679 and 1668, respectively. The original test bench delivered with the arithmetic core, simulating all the operations, rounding modes, and corner cases, is also exercised with the mutation analysis. It killed 1440 mutants.

In summary, experiments gave a clear improvement by the heuristics against the single uniform distribution, in terms of a higher total mutant-killing coverage and less simulation effort. This means that our heuristics significantly advances the current state of mutation-based verification automation. Although the deterministic test bench exposes a certain amount of mutants more rapidly, it is the advantage of the CRV to avoid the manual, labour intensive writing and improving of test cases.

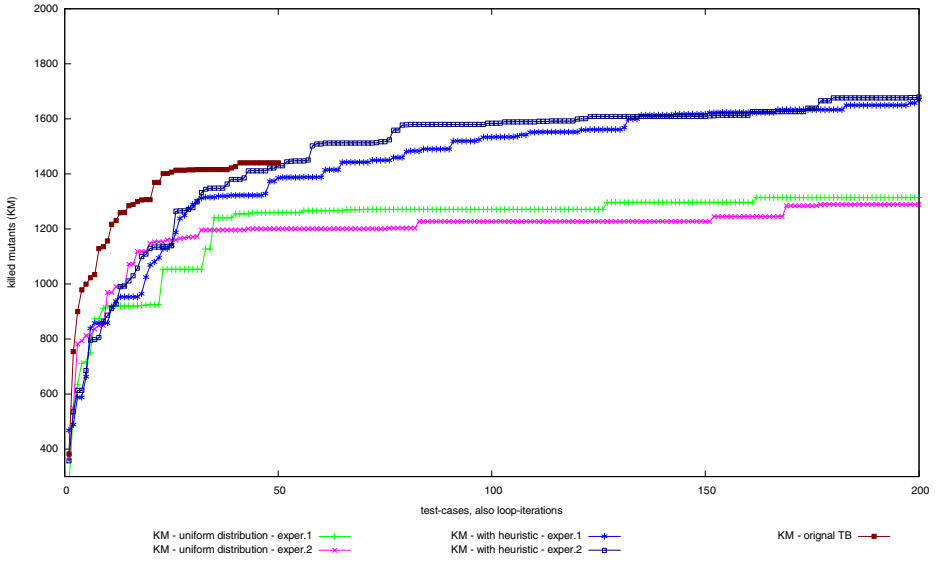

Fig. 4. Experimental results

5 Conclusion

We primarily considered the cost model when applying mutation analysis as the coverage metric to measure the completeness of a CRV flow. Basically, the simulation

effort is $O(T * L^2)$, where T stands for the number of test cases and L the size of the design. L representing the design complexity grows rapidly along with the Moore's Law driven capacity of integrated circuits. In CRV, T is further enlarged as the amount of test cases required to reach an adequacy is based on random generation. More accurately, with some probability analyses $(\max_i 1/ps_i + \sum_i pw_i/ps_i)$ is found to be the expected simulation runs. Based on this, a heuristics is developed that collects the past analysis information to estimate the effectiveness of test domain ranges and re-distributes the probability.

The CRV flow equipped with the dynamic distribution adjustment heuristics has been implemented and experimented with Certitude™ and a VHDL floating point arithmetic unit. The results are encouraging and show the efficiency improvement in terms of *reaching more rapidly a higher mutant-killing coverage*. With more, yet automated simulation effort, it also surpasses the manual test bench that is carefully composed by the author of the arithmetic core.

In future work, we will investigate different architecture and their impact on the heuristics with a focus on control-oriented circuits like microprocessors. Since the verification flow is based on simulation, it also scales well to large designs.

In contrast to other fault-based test generation approaches, MA systematically correlates the mutant-killing and the test's capability of revealing design errors. This can be key technology for solving the verification bottleneck today. The work presented in this paper is established on the macro relation between the test input domain and the overall mutant killing. It promotes coverage efficiency but specific, even-harder-to-expose mutants may remain. Here, future work will also consider the automatic, deterministic test generation for exposing an individual mutant. Existing solutions rely on symbolic execution and constraint solving with the assumption that mutant behaviors propagate to the output if activated. This is a limitation when the algorithms face complex SW/HW/system designs. More efficient, light-weight solutions have to be developed to enable a practical deployment. Nevertheless, the MA directed CRV will remain a necessary step to obtain a first mutant killing coverage, since it sieves out the easy-to-kill mutants, which the deterministic test generation algorithms can hardly do.

Acknowledgements. The work described herein is funded by the FP7 COCONUT project (FP7-ICT-3217069), the BMBF ITEA2 project VERDE (VERDE (01S09012H), and the DFG Sonderforschungsbereich SFB 614 (Self-Optimizing Systems for Mechanical Engineering).

References

[1] DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on Test Data Selection: Help for the Practicing Programmer. IEEE Computer 11(4) (April 1978)
[2] DeMillo, R.A., Offutt, A.J.: Constraint-Based Automatic Test Data Generation. IEEE Transactions on Software Engineering 17(9) (September 1991)
[3] Howden, W.E.: Weak Mutation Testing and Completeness of Test Sets. IEEE Transactions on Software Engineering 8(4) (July 1982)
[4] Offutt, A.J.: The coupling effect: fact or fiction. ACM SIGSOFT Software Engineering Notes 14(8) (December 1989)

[5] Offutt, A.J., Seaman, E.J.: Using Symbolic Execution to Aid Automatic Test Data Generation. In: Proceedings of the fifth Annual Conference on Computer Assurance, COMPASS 1990, Gaithersburg, MD, USA (June 1990)
[6] Untch, R.H., Offutt, A.J., Harrold, M.J.: Mutation Analysis Using Mutant Schemata. ACM SIGSOFT Software Engineering Notes 18(3) (July 1993)
[7] Hampton, M., Petithomme, S.: Leveraging a Commercial Mutation Analysis Tool For Research. In: Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques- MUTATION. TAICPART-MUTATION 2007 (September 2007)
[8] Sengupta, S.: Defect-based Tests: A Key Enabler for Successful Migration to Structural Test. Intel Technology Journal (1999)
[9] Al Hayek, G., Robach, C.: From Specification Validation to Hardware Testing: A Unified Method. In: Proceedings of the IEEE International Test Conference on Test and Design Validity, ITC 1996, Washington, DC, USA (October 1996)
[10] Ghosh, S., Chakraborty, T.J.: On Behavior Fault Modeling for Digital Designs. Journal of Electronic Testing: Theory and Applications 2(2) (June 1991)
[11] Ferrandi, F., Fummi, F., Sciuto, D.: Implicit Test Generation for Behavioral VHDL Models. In: Proceedings of the 1998 IEEE International Test Conference, ITC 1998, Washington, DC, USA (October 1998)
[12] Fin, A., Fummi, F.: A VHDL Error Simulator for Functional Test Generation. In: Proceedings of the 2000 conference on Design, Automation and Test in Europe, DATE 2000, Paris, France (March 2000)
[13] International Technology Roadmap for Semiconductors (ITRS). ITRS 2009 Edition, http://www.itrs.net/Links/2009ITRS/Home2009.htm
[14] SpringSoft. Functional Qualification Tool Certitude™, http://www.springsoft.com/products/functional-qualification/certitude
[15] Corno, F., Sonza Reorda, M., Squillero, G., Manzone, A., Pincetti, A.: Automatic Test Bench Generation for Validation of RT-Level Descriptions: An Industrial Experience. In: Proceedings of the 2000 Conference on Design, Automation and Test in Europe, DATE 2000, Paris, France (March 2000)
[16] Tasiran, S., Fallah, F., Chinnery, D., Weber, S., Keutzer, K.: Coverage-Directed Generation of Biased Random Inputs for Functional Validation of Sequential Circuits. In: Proceedings of the International Workshop on Logic and Synthesis (June 2001)
[17] Fine, S., Ziv, A.: Coverage directed test generation for functional verification using bayesian networks. In: Proceedings of the 40th annual Design Automation Conference, DAC 2003, Anaheim, CA, USA (2003)
[18] Open SystemC Initiative Verification Working Group, SystemC Verification Library Standard, release 1.0p2 (2006), http://www.systemc.org/downloads/standards
[19] Bombieri, N., Fummi, F., Pravadelli, G., Hampton, M., Letombe, F.: Functional Qualification of TLM Verification. In: Proc. of the 2009 ACM/IEEE Design, Automation and Test in Europe, DATE 2009, Nice, France (April 2009)
[20] Lachish, O., Marcus, E., Ur, S., Ziv, A.: Hole Analysis for Functional Coverage Data. In: Proceedings of the 39th Conference on Design Automation, pp. 807–812 (June 2002)
[21] Asaf, S., Marcus, E., Ziv, A.: Defining coverage views to improve functional coverage analysis. In: Proceedings of the 41st Conference on Design Automation, pp. 41–44 (June 2004)