ARPIA: a High-Level Evolutionary Test Signal Generator

F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero

Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24 I-10129, Torino, Italy
http://www.cad.polito.it/

Abstract. The integrated circuits design flow is rapidly moving towards higher description levels. However, test-related activities are lacking behind this trend, mainly since effective fault models and test signals generators are still missing. This paper proposes ARPIA, a new simulation-based evolutionary test generator. ARPIA adopts an innovative high-level fault model that enables efficient fault simulation and guarantees good correlation with gate-level results. The approach exploits an evolutionary algorithm to drive the search of effective patterns within the gigantic space of all possible signal sequences. ARPIA operates on register-transfer level VHDL descriptions and generates effective test patterns. Experimental results show that the achieved results are comparable or better than those obtained by high-level similar approaches or even by gate-level ones.

1 Background

In recent years the application specific integrated circuit (ASIC) design flow experienced radical changes. Deep sub-micron integrated circuit (IC) manufacturing technology is enabling designers to put millions of transistors on a single integrated circuit. Following Moore's law, design complexity is doubling every 12-18 months. In addition, there is an ever-increasing demand on reducing time to market. With complexity skyrocketing and such a competitive pressure, designing at high levels of abstraction has become more of a necessity than an option.

At the present time, exploiting design-partitioning techniques, register-transfer level (RT-level) automatic logic synthesis tools can be successfully adopted in many ASIC design flows. However, not all activities have already migrated from gate- to RT-level and are not yet mature enough to.

High-level design for testability, testable synthesis and test pattern generation are increasing their industrial relevance [1]. During the development of ASIC, designers would like to be able to foresee its testability before starting the logic synthesis process. Furthermore, RT-level automatic test signals generators are expected to exploit higher-level compact information about design structure and behavior, and to be able to produce more effective sequences within reduced CPU time. The test signals may possibly be completed after synthesis by ad-hoc gate-level tools.

However, despite the big effort of electronic design automation (EDA) industries, tackling test issues at high levels is still an unsolved problem. The lack of a common fault model is probably the hardest theoretical barrier. Over the years, the computer aided design (CAD) community agreed on some well-defined gate-level fault models. The most popular is the permanent single stuck-at fault, and all commercial products are able to use it. On the contrary, up to now it was unable to agree on any high-level fault model.

The most important technical barrier, on the other side, is the lack of efficient algorithms to generate effective test signals. As a matter of fact, at the present time even *simulating* RT-level test signals is a challenging task. Fault simulation algorithms for RT-level designs are known since more than a decade [2], but commercial tools usually don't include these capabilities. Furthermore, classical algorithms are difficult to integrate in simulators, mainly due to the complexity and to the several peculiarities of hardware description languages. Some prototypical fault simulators were proposed [3] [4], but until some fault model becomes widely accepted EDA industries have no good reason to invest and this situation is not likely to change.

Even so, researchers and pioneering design groups already need test signals on their RT-level designs. Many generators were proposed. Nevertheless, since any attempt of backward justification must take into account all structural, behavioral and timing specifications [5], traditional algorithms are almost unusable. Researchers sometimes achieved good results, but they were generally limited to specific classes of circuits. However, interesting successful results have been reported using evolutionary algorithms [6]. These approaches exploit natural evolution principles to drive the search of effective patterns within the gigantic space of all possible signal sequences. Evolutionary heuristics begin to appear a reasonable alternative to traditional techniques.

This paper presents ARPIA, a high-level evolutionary automatic test signals generator. Experimental results gathered using the prototypical implementation is remarkable. The effectiveness of the generated test signals is at least comparable with (and in several cases higher than) that of previously proposed approaches. Additionally, thanks to the evolutionary algorithm and to a fault dropping mechanism, computational requirements of the new system are lower.

Next Session details ARPIA. An analysis of the proposed approach is illustrated in Section 4. Section 5 draws some conclusions.

2 ARPIA

ARPIA is a simulation-based evolutionary test signals generator. Being an evolutionary algorithm, it evolves a population seeking fitter individuals. But, since individuals are test sequences for a digital circuit, the fitness measures the sequence ability to detect faults in the design. And it is computed by simulation. Given a fault model, the *fault coverage* is defined as the percentage of faults that the test sequence is able to detect. Thus, the goal of ARPIA can be rephrased as "generate a sequence of signals that attains maximum fault coverage."

ARPIA shares the same philosophy with [6]. They are both simulation-based approaches, and individuals are evaluated resorting to an RT-level fault simulator. However, the two methodologies exploit different fault models, different fault simulation techniques and different evolutionary algorithms. Next Sections detail these three key points.

2.1 Fault Model

Many fault models have been proposed in the past, mainly by borrowing from software-testing metrics [7]. While software derived metrics are well known and quite standardized, and they are already implemented in some commercial tools, their usefulness for hardware testing is quite low. In particular, software metrics neglect observability issues and the effects of logic synthesis, and thus are not an accurate indicator of circuit testability.

One successful proposal of a hardware-related fault model is *Observability-Enhanced Statement Coverage* [8] [4]: it introduces the concept of *tag* as the possibility that an incorrect value is computed at a given location, thus approximating the effects of fault propagation. Since this fault model does not assume any specific fault effect, its generality prevents explicit fault simulation.

An extension of observability-enhanced statement coverage was first proposed in [9] and then refined in [10], where explicit *RT-level assignment single-bit* stuck-at's are used instead of generic tags. An RT-level assignment single-bit stuck-at fault is defined as a single-bit stuck-at in the effect of an RT-level assignment operation: when a fault is present, the affected object (signal or variable target of an assignment statement) loads the correct value, except for one bit that is forced to 0 or 1.

Figure 1 shows the example of a stuck-at fault. The fault affects the third leftmost bit of the assignment operation, and modifies the result of the expression, after it has been computed and before it is assigned to the target signal. The faulty signal is updated as usual, according to propagation rules, but with a faulty value. Other assignments of the same signal are assumed to be fault-free, since stuck-at faults on the same signal but on different statements are considered different. More details about the fault model can be found in [9].

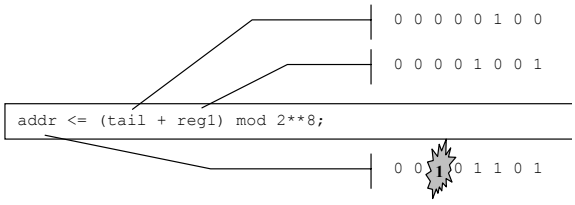

Figure 1: ***Stuck-At Fault Example***

The current fault model exploits several heuristics to filter out unrepresentative RT-level stuck-at. This elimination enables the RT-level assignment single-bit stuck-at fault coverage to be highly correlated with gate-level fault coverage: the correlation

coefficient R, computed over a set of 10 different circuits by simulating the test sequences generated in three different ways, reaches R=0.77 [10].

2.2 Fault Simulation Technique

Fault simulation is made possible by creating routines that *change* the target signal/variable bit value during simulation, using the simulator scripting language (Tcl), when a given target assignment instruction is executed. The fault injection procedures, presented in [9], must face various issues derived from the fault model, from VHDL semantics and from the simulator itself.

This fault simulation environment allows us to compute fault coverage figures at the RT-level with a minimal CPU time overhead, since the VHDL model of faulty circuits is simulated at the same speed as the fault-free model. The main time penalty is at fault activation time instants, where some breakpoints are set and TCL commands to modify values are executed.

2.3 Algorithm

A fault can be marked as *tested* only when it is both *excited* and *observed*. Given a fault, the target of the whole process is first to force the corresponding bit to a value that make the fault visible (excitation), then to propagate the fault effects to some primary output (observation).

```
current_sequence = random_sequence(L);
steady_state_factor = 0;
do
{
   new_sequence = mutate(current_sequence);
   if(excited_faults(new_sequence) > excited_faults(current_sequence)) {
      current_sequence = new_sequence;
      steady_state_factor = 0;
   } else {
      increase(steady_state_factor);
   }
} while(excited_faults(current_sequence) == 0 &&
      steady_state_factor < steady_state_limit)
```

Figure 2: Phase One Pseudo Code

The two problems are tackled separately, with two different strategies. Indeed, test generation is performed in three phases. The first is aimed at exciting faults, the second tackle observation, while the third dynamically optimize the fault list.

The goal of the first phase is to produce a set of signals able to excite an untested fault. The first phase implements a simple first-improvement hill climber (Figure 2). The local search procedure starts with a random sequence of given length *L*. In each step, a new sequence is generated by randomly mutating the current one. If the new sequence excites a larger number of untested faults, it becomes the current one.

Otherwise it is discarded. The process ends whenever the current sequence is able to excite at least k fault, or after a predefined amount of useless steps. It is worth noting that the number of excited faults is computed over the untested faults only.

Three mutations are currently implemented by ARPIA: *add*, *delete* and *change*. The first two respectively add and delete a vector of input signals from the test sequence. The last one randomly changes a vector of input signals in the sequence.

When a test sequence able to excite a sufficient number of faults is found, it is transferred to the second phase together with the excited faults. The goal of the second phase is to observe each single fault in the excited set. This stage exploits an evolutionary algorithm similar to an evolution strategy [11].

First, a target fault f_t is selected from the excited set. Then, a population of P sequences is created by mutating the original sequence and evolved using a (P+P) strategy. In every evolution step, P new individuals are generated by mutating the P original ones (each sequence generates exactly one new sequence). The P fitter individuals are selected for survival among the 2P. The same three mutation operators of the first phase are adopted.

In the second phase, given a target fault f_t, the fitness measures how far a sequence is able to propagate f_t effects. More precisely, it is the maximum number of differences caused by the fault during the application of a single vector of signals of the test sequence (1).

$$evaluation\_two(S, f_t) = \underset{v \in S}{MAX} \sum_{objects} different\_bits(object) \tag{1}$$

The evolution is halted whenever f_t is detected or after a certain amount of generations. The pseudo code is shown in Figure 3.

The second phase is iterated until all faults in the excited set have been tested or aborted.

The evolution-strategy approach was chosen because of the complexity of the encoding. Individuals are sequence of vectors. Each vector of signals is simulated in a clock cycle. Circuits contain memory elements, thus the behavior in a clock cycle depends both on current input signals and previous ones. The effect of a traditional recombination operator, like the uniform crossover, can be very similar to a complete random mutation at phenotypic level. We shun any risk and exploit an algorithm that "omits recombination since its philosophy relies on species as evolving entities" [12].

After each successful second phase, an optimization mechanism called *fault dropping* is activated. All still untested faults are simulated with the new sequence, seeking if any additional fault is detected by it. This is more of a possibility than an expectation, since the starting sequence found in the first phase is required to be able to excite more than one fault. The fault dropping mechanism greatly enhances overall algorithm performance.

```
for t = 1 to P {
   population[t] = mutate(starting_sequence);
}
generations = 0;
success = 0;

do
{
   generations = generations + 1;
   for t = 1 to P {
      population[P + t] = mutate(population[t]);
      if(tested(f_t, pupulation[t])
         success = 1;
   }
   sort(population);
} while(not success && generations < generations_limit);
```

Figure 3: Phase Two Pseudo Code

3 Experimental Analysis

In order to practically evaluate the effectiveness of the proposed approach, we implemented a prototype. The generator is composed of about 1,500 lines in ANSI C and interacts with V-System 5.3 VHDL simulator by Model Technology. Special techniques are adopted to speed-up fault simulation [9]. During experiments we adopt the following parameter values:

- first-phase initial sequence length of 50 clock cycles ($L = 50$)
- first phase sequence required to excite 5 different faults ($k = 5$)
- second phase population of 10 individuals ($P = 10$).

Circuit	CPU time	RT Faults			Gate Faults		
	[s]	Tot	Det	FC%	Tot	Det	FC%
b01	78.80	81	81	100.00%	258	258	100.00%
b02	39.19	43	39	90.70%	150	149	99.33%
b03	1,089.96	213	145	68.08%	822	615	74.82%
b04	1,627.86	424	353	83.25%	3,356	3,035	90.44%
b05	1,932.27	778	244	31.36%	5,552	1,856	33.43%
b06	200.31	110	82	74.55%	5,552	5,387	97.02%
b07	9,297.26	289	146	50.52%	2,404	1,401	58.28%
b08	2,832.38	154	118	76.62%	918	839	91.39%
b09	4,970.53	240	196	81.67%	900	768	85.33%
b10	778.35	172	127	73.84%	1,054	961	91.18%
b11	34,837.11	381	263	69.03%	2,868	2,614	91.14%
b12	7,890.20	870	115	13.22%	5,280	1,105	20.92%
b13	2,801.85	284	224	78.87%	1,818	1,501	82.56%
b14	473,741.90	10,493	9,114	86.86%	28,990	23,708	81.78%
b15	590,611.31	4,900	2,026	41.35%	55,568	18,060	32.50%

Table 1: ARPIA Result

Table 1 reports the experiments performed on the ITC99 RT-level benchmarks. These benchmarks are representative of typical circuits, or circuit parts, that can be automatically synthesized as a whole with current tools and are described in [13]. Experiments have been run on a Sun Enterprise 250 running at 400 MHz and equipped with 2 Gbytes of RAM.

The first column of Table 1 reports the name of the benchmark, while the CPU time required to generate the test signal sequence is shown in the second column. RT-level fault figures are reported in the next column block in terms of: total number of faults [Tot], number of detected faults [Det] and percent fault coverage [FC%]. The next column block shows gate-level figures: total number of gate-level faults [Tot], number of detected [Det] and respective fault coverage [FC%].

Results show that ARPIA is able to generate test sequences that are highly effective both at RT-level and at gate-level, within an acceptable CPU time. However, to better analyze the tool performance, we need to compare it with different approach (Table 2).

Circuit	ARPIA (no ES)		ARTIST		ARPIA	
	RT	Gate	RT	Gate	RT	Gate
b01	100.00%	99.61%	100.00%	100.00%	100.00%	100.00%
b02	90.70%	99.33%	90.70%	99.33%	90.70%	99.33%
b03	56.81%	69.10%	68.08%	74.82%	68.08%	74.82%
b04	69.34%	69.19%	83.79%	91.03%	83.25%	90.44%
b05	11.31%	5.42%	31.36%	33.50%	31.36%	33.43%
b06	70.00%	93.38%	74.80%	97.35%	74.55%	97.02%
b07	47.75%	56.49%	50.52%	58.28%	50.52%	58.28%
b08	52.60%	28.00%	60.10%	71.68%	76.62%	91.39%
b09	62.50%	48.89%	77.84%	81.33%	81.67%	85.33%
b10	46.51%	65.84%	73.69%	90.99%	73.84%	91.18%
b11	43.57%	58.79%	68.64%	90.62%	69.03%	91.14%
b12	2.76%	4.36%	29.06%	45.99%	13.22%	20.92%
b13	31.69%	31.19%	n/a	n/a	78.87%	82.56%
b14	11.57%	37.91%	n/a	n/a	86.86%	81.78%
b15	14.69%	12.75%	n/a	n/a	41.35%	32.50%

Table 2: Comparison with ARTIST and ARPIA without Evolutionary Algorithm

The first column block of Table 2 reports data for the first prototype of ARPIA, where a simple hill-climber was exploited instead of the evolution strategy. The gap between the two RT-level figures shows the fundamental role played by the evolutionary algorithm. The gap between gate-level fault coverage statistics is a mere consequence.

In the second column group of Table 2 we reported results attained by ARTIST [6], a highly-optimized tool exploiting a genetic algorithm. The difference between these two tools can be explained resorting to both the fault model and the new evolutionary mechanism. A deeper comparison between the results of ARPIA and of ARTIST (complete data can not be reported here for lack of space) shows that the former is characterized by a higher efficiency (i.e., it requires a lower CPU time), thanks to fault dropping and a higher compactness of the generated sequences.

Indeed, ARTIST was not able to tackle some of the benchmarks due its lower efficiency (marked with "n/a" in the table).

Circuit	RT-Level Faults				Phase2
	Tot	Exc	Det	Err	Efficacy
b01	81	81	81	0	100.00%
b02	43	43	39	4	100.00%
b03	213	148	145	3	100.00%
b04	424	356	353	3	100.00%
b05	778	340	244	19	76.01%
b06	110	87	82	5	100.00%
b07	289	240	146	20	66.36%
b08	154	118	118	0	100.00%
b09	240	196	196	0	100.00%
b10	172	139	127	12	100.00%
b11	381	289	263	26	100.00%
b12	870	130	115	1	89.15%
b13	284	266	224	16	89.60%
b14	10,493	10,165	9,114	0	89.66%
b15	4,900	2,244	2,026	24	91.26%

Table 3: Phase Two Effectiveness

The effectiveness of the evolutionary algorithm can also be seen in Table 3, where the number of excited faults is shown in column [Exc] together with the number of detected faults [Det]. The effectiveness of the evolutionary algorithm can be defined as its ability to observe (i.e., to detect) excited faults, not considering faults that are certainly unobservable due to incorrect design ([Err] column).

It should be noted that phase two effectiveness is quite high also for b12, a problematic circuit where ARPIA only manages to get 13.22% RT-level fault coverage. Thus, primarily the first phase can be hold responsible for the low performance.

4 Conclusions and Future Works

Due to the wide adoption of logic synthesis tools, high-level techniques are increasingly necessary in order to shift test-related activities towards the description level adopted by designers. A crucial point for developing effective high-level test signal generator lies in the identification of a suitable fault model. Another crucial point is the availability of a suitable algorithm for test generation.

In this paper we exploit a recent fault model to propose a methodology for generating high quality test signals. The approach is based on an evolution strategy. Experimental data show the potential of the proposed approach with regard to similar techniques, the effectiveness of the fault model, and the convenience of the evolutionary algorithm.

We are currently improving ARPIA in two directions. First, we are trying to replace the hill-climber in the first phase with sharper algorithms, solving the issues stemming from the lack of information concerning the system behavior before fault

excitation. We are also considering the adoption of an evolutionary core similar to the one exploited in [14]. Secondly, we are tuning the evolution strategy adopted in the second phase, experimenting more complex selection schemes. We are pondering to change the algorithm, adding back recombination in some constrained form.

5 References

[1] "High Time for High-Level Test Generation," Panel at the *IEEE International Test Conference*, 1999, pp. 1112-1119

[2] M. Abramovici, M. A. Breuer, A. D. Friedman, *Digital systems testing and testable design*, Computer Science Press, 1990

[3] A. Fin, F. Fummi, "A VHDL Error Simulator for Functional Test Generation," *IEEE European Design, Automation and Test Conference*, 2000, pp. 390-395

[4] F. Fallah, S. Devadas, K. Keutzer, "OCCOM: Efficient Computation of Observability-Based Code Coverage Metrics for Functional Verification," *DAC98: 34th Design Automation Conference*, 1998

[5] F. Ferrandi, F. Fummi, L. Gerli, D. Sciuto, "Symbolic Functional Vector Generation for VHDL Specifications," *DAC99: 35th Design Automation Conference*, 1999, pp. 442-446

[6] F. Corno, M. Sonza Reorda, G. Squillero, "High-Level Observability for Effective High-Level ATPG," *VTS2000: 18th IEEE VLSI Test Symposium*, May 2000, pp. 411-416

[7] B. Beizer, *Software Testing Techniques* (2nd ed.), Van Nostrand Rheinold, 1990

[8] F. Fallah, P. Ashar, S. Devadas, "Simulation Vector Generation from HDL Descriptions for Observability-Enhanced Statement Coverage," *DAC99: 35th Design Automation Conference*, 1999, pp. 666-671

[9] F. Corno, G. Cumani, M. Sonza Reorda, Giovanni Squillero, "RT-level Fault Simulation Techniques based on Simulation Command Scripts," *DCIS 2000: XV Conference on Design of Circuits and Integrated Systems*, November 21-24, 2000

[10] F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero, "An RT-level Fault Model with High Gate Level Correlation," *IEEE International High Level Design Validation and Test Workshop*, November 8-10, 2000

[11] T. Bäck, F. Hoffmeister, and H.-P. Schwefel, "A survey of evolution strategies." *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, pp. 2-9

[12] H.-P. Schwefel, F. Kursawe, *On Natural Life's Tricks to Survive and Evolve*, Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, 1998, pp. 1-8

[13] F. Corno, M. Sonza Reorda, G. Squillero, "RT-Level ITC 99 Benchmarks and First ATPG Results," *IEEE Design & Test, Special issue on Benchmarking for Design and Test*, July-August 2000, pp. 44-53

[14] F. Corno, M. Sonza Reorda, G. Squillero, "Automatic Validation of Protocol Interfaces Described in VHDL," *EvoTel2000: European Workshops on Telecommunications*, Edinburgh (UK), May 2000, pp. 205-213