Proc. Estonian Acad. Sci. Eng., 2006, **12**, 3-2, 323–335

Environment for FPGA-based fault emulation

Peeter Ellervee, Jaan Raik, Kalle Tammemäe and Raimund Ubar

Department of Computer Engineering, Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia; lrv@cc.ttu.ee, jaan@pld.ttu.ee, kalle@cc.ttu.ee, raiub@pld.ttu.ee

Received 15 May 2006

Abstract. This paper describes an environment to accelerate fault simulation by hardware emulation on FPGA. Fault simulation is an important subtask in test pattern generation and it is frequently used throughout the test generation process. The problems associated with fault simulation of digital circuits are explained. The proposed approach allows simulation speed-up of 40 to 500 times as compared to the state-of-the-art in software-based fault simulation. Based on the experiments, it can be concluded that it is beneficial to use emulation for circuits that require large numbers of test vectors while using simple but flexible algorithmic test vector generating circuits, e.g. built-in self-test.

Key words: fault simulation, acceleration, emulation, field-programmable logic.

1. INTRODUCTION

Test generation has become one of the most complicated and time-consuming problems in the field of digital design. As the size of circuits grows, so do the test costs [1]. This includes both time and resources spent to test a circuit, and time and resources spent to generate appropriate test vectors. Many techniques exist to perform the suitability analysis of a given set of test vectors – the most important subtask of any test generation approach. Efficient fault simulation algorithms for combinational circuits are known already for some time [2]. However, large sequential designs create the need for faster implementation, e.g. by hardware emulation. For instance, it may take years to simulate processor cores [3]. At the same time, reconfigurable hardware devices have been found useful as system modelling environments [4]. This has been made possible by the availability of multimillion-gate FPGAs. For academic purposes, cheaper devices with rather large capacity, e.g. the newest Spartan3 chips, can be used.

The availability of large FPGAs does not allow merely implementation of the circuit under test along with fault models but, additionally, to include test vector generation and output response analysis circuits into a single reconfigurable device. It is important to stress that current approach is not aiming at testing the FPGA itself neither simulating any defects in it: FPGA is assumed to be tested by the manufacturer. The purpose of the emulation environment, described in this paper, is to speed up fault simulation mainly for ASIC and System-on-Chip (SoC) projects using FPGA simply as a fast emulation environment.

A number of papers on fault emulation for combinational circuits has been published. They rely either on fault injection [5,6] or on implementing specific fault simulation algorithms in hardware [7]. Recently, acceleration of combinational circuit fault diagnosis using FPGAs has been proposed in [8]. In many of the approaches, faults are injected either by modifying the configuration bitstream while the latter is being loaded into the device [9,10] or by using partial reconfiguration [11–13]. This kind of approach is slow due to the run-time overhead required by multiple reconfigurations. Other options for fault injection are to model faults with additional circuitry either at netlist wires [10] (used also in this paper) or at the gates [14]. In addition to merely increasing the speed of fault simulation, the idea proposed in this paper can be used for selecting optimal built-in self-test (BIST) structures. In earlier papers [14,15], fault emulation methods to be used for evaluating the circular self-test path (CSTP) type BIST architectures have been presented.

In [16], we proposed an efficient FPGA-based fault emulation environment for sequential circuits. The main feature of the emulation approach was in implementing multiplexers and distributed decoders for fault injection, which, unlike the shift register based injection, allowed to insert faults in an arbitrary order. In addition, we used an on-chip input pattern generator as opposed to loading the simulation stimuli from the host computer. The fault emulation environment, presented in this paper, introduces hardware support for fault dropping that increases the simulation speed-up nearly by an order of magnitude. This is achieved without a significant penalty to the area of the accelerator FPGA. In addition, the improved emulation environment allows implementation of various fault-modelling scenarios.

The emulation approach is planned for use in cooperation with diagnostic software Turbo Tester, described in Section 2. The emulation environment is described in Section 3. In Section 4, the results of experiments are presented and Section 5 is dedicated to conclusions.

2. OVERVIEW OF THE TURBO TESTER

The emulation environment was designed to work together with Turbo Tester (TT), a diagnostic software package developed at the Department of Computer Engineering of the Tallinn University of Technology [17,18]. The TT software consists of the following test-related tools: test generation by different algorithms (deterministic, random and genetic), test program optimization, fault simulation for combinational and sequential circuits, and testability analysis and fault diagnosis. It includes test generators, logic and fault simulators, a test optimizer, a module for hazard analysis, linear feedback shift register (LFSR) simulators for BIST, design verification and design error diagnosis tools (Fig. 1). TT can read the schematic entries of various contemporary VLSI CADs that makes it independent of the existing design environment. Turbo Tester versions are available for MS Windows, Linux, and Solaris operating systems.

The main advantage of the system is that different methods and algorithms for various test problems have been implemented and can be investigated separately of each other or working together in different combinations.

Model synthesis. The component library of Turbo Tester consists of Binary Decision Diagram (BDD) representations for the library components of the circuits to be processed. The library is open and can be updated for new components.

Test generation. For automatic test pattern generation (ATPG), random, deterministic and genetic test pattern generators (TPG) have been implemented. Mixed TPG strategies, based on different methods, can also be investigated. Tests can be generated both for combinational and sequential circuits.

Test pattern analysis. Single fault simulation, parallel fault simulation and critical path tracing fault analysis methods are implemented in the system. These competing approaches can be investigated and compared for circuits of different complexities and structures. For this part the emulation approach was proposed. So far we have experimented only with "Fault Simulation" part (black in Fig. 1). In the future, also the other simulation-related parts might be considered (gray in Fig. 1).

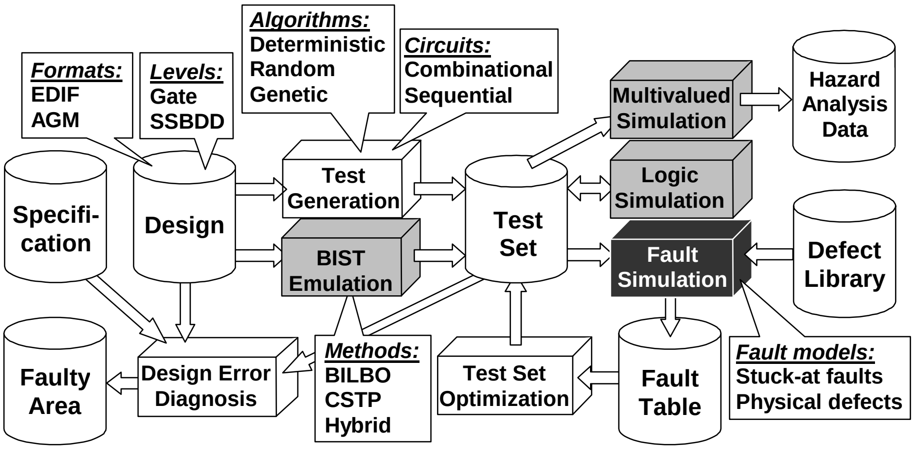

Fig. 1. Turbo Tester environment.

Test set optimization. The tool minimizes the number of test patterns in the test set by means of static compaction. The technique implements effective representation of fault matrices by weighted bipartite graphs.

Multivalued simulation. In Turbo Tester, multivalued simulation is applied to model possible hazards that can occur in logic circuits. The dynamic behaviour of a logic network during one single transition period can be described by a representative waveform on the output or simply by a corresponding logic value.

Design error diagnosis. After a digital system has been designed according to specifications, it might go through a refinement process in order to be consistent with certain design requirements, e.g., timing specifications. The changes introduced by this may lead to undesired functional inconsistencies compared to the original design. Such design errors should be identified by verification.

Testability analysis. The real cost of a digital product is expressed as

$$Cost(Design + Test) < Cost(Design) + Cost(Test).$$

It follows from the fact that looking at the design and test as one integral activity rather than two separate unrelated activities, designer can minimize the total cost. The tools can be used for, e.g. enumerating untestable faults, for estimating the controllability, observability and testability characteristics of the design.

Evaluation of built-in self-test (BIST) quality. The BIST approach is represented by applications for built-in logic block observer (BILBO) and circular self-test path (CSTP) emulation. Different BIST architectures can be simulated and the self-test quality of these architectures can be evaluated.

3. EMULATION ENVIRONMENT

The initial emulation environment was created keeping in mind that the main purpose was to evaluate the feasibility of replacing fault simulation with emulation. Based on that, the main focus was put on how to implement circuits to be tested on FPGAs. Less attention was paid how to organize data exchange between hardware and TT. For the first series of experiments, we looked at combinational circuits only. Results of experiments with these circuits were presented in [6].

For sequential circuits, most of the solutions used for combinational circuits could be exploited. The main modification was an extra loop in the controller because sequential circuits require not a single input combination but a sequence consisting of tens or even hundreds of input combinations. Also, instead of hard-coded test sequence generators and output analysers, loadable modules were introduced. The authors of this paper carried out their first experiments with sequential circuits in [16].

As a novel feature, the emulation environment incorporates hardware support for fault dropping, i.e. if a fault has been detected by a test vector then the remaining test sequence will be cancelled and the process will continue with the next fault. Comparison against the golden device is used for implementing the fault dropping. The proposed approach allows simulation speed-up of 40 to 500 times as compared to the state-of-the-art in fault simulation. Another improvement is the use of three-valued logic instead of the two-valued one. This allows to eliminate the problem of undefined register values at power-up. It should be noted that the size of the circuits is more or less doubled because of the need to use two wires instead of one to encode a bit.

Fault insertion. The main problem here was how to represent non-logic features, faults, in such a way that they can be synthesized using standard logic synthesis tools. Since most of the analysis is done using stuck-at-one and stuck-at-zero fault models, the use of multiplexers at fault points was the most obvious one. Also, since typically a single fault is analysed at a time, decoders were introduced to activate faults (Fig. 2a). The extra multiplexers will increase the gate count (approximately 3–4 times) and will make the circuit slower (typically 5–10 times). It is not a problem for smaller circuits but may be too prohibitive for larger designs – the circuit may not fit into the target FPGA. A solution is to insert faults selectively. Selection algorithm, essentially fault set partitioning, is a subject of future research. Compared against shift-register based fault injection approaches [10] (Fig. 2b), the use of multiplexers has both advantages and

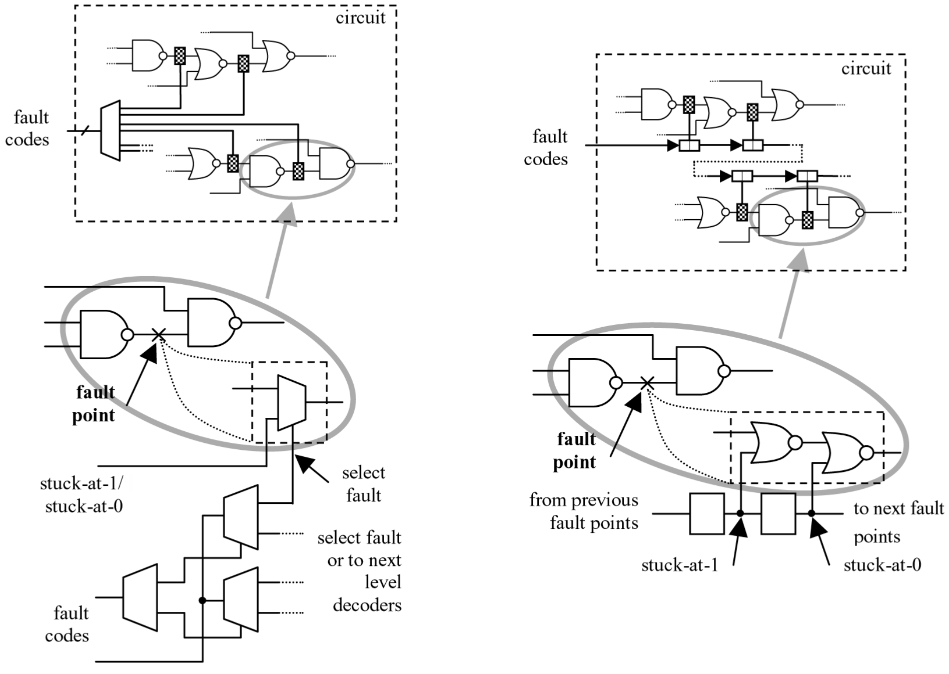

Fig. 2. Fault point insertion with multiplexer (a) and with shift-register (b).

disadvantages. The main disadvantage is small increase both in area and delay of the circuit. Although the delay increase is only few percents, execution time may increase significantly for long test cycles. The main advantage is that any fault can be selected in a single clock cycle, i.e. there is no need to shift the code of a fault into the corresponding register. This difference comes from the way a fault point is activated – multiplexers receive activation through distributed decoders (shown simplified in Fig. 2a) but when using shift-registers, the activation is shifted from one flip-flop to another (Fig. 2b). It should be noted that also the order of fault points in the shift-register is important because this affects the length of the wires between different flip-flops. The decoders, on the other hand, require more control lines between the circuit under test and controller. Combining both approaches may be the best solution and a part of our future work is planned in that direction.

Test vector generation and output data analysis. Here we relied on a well-known solution for BIST – Linear Feedback Shift Register (LFSR) is used both for input vector generation and output correctness analysis [19]. LFSRs structures are thoroughly studied and their implementation in hardware is very simple. This simplifies data exchange with the software part – only seed and feedback polynomial vectors are needed to get the desired behaviour. Hardware for the analysis of the output correctness needs first to store the expected output signature and then to report to the software part whether the modelled fault was detected or not. Figure 3a illustrates a stage of a LFSR, implemented in the current approach. Figures 3b and 3c illustrate the generator and the analyser, respectively. The

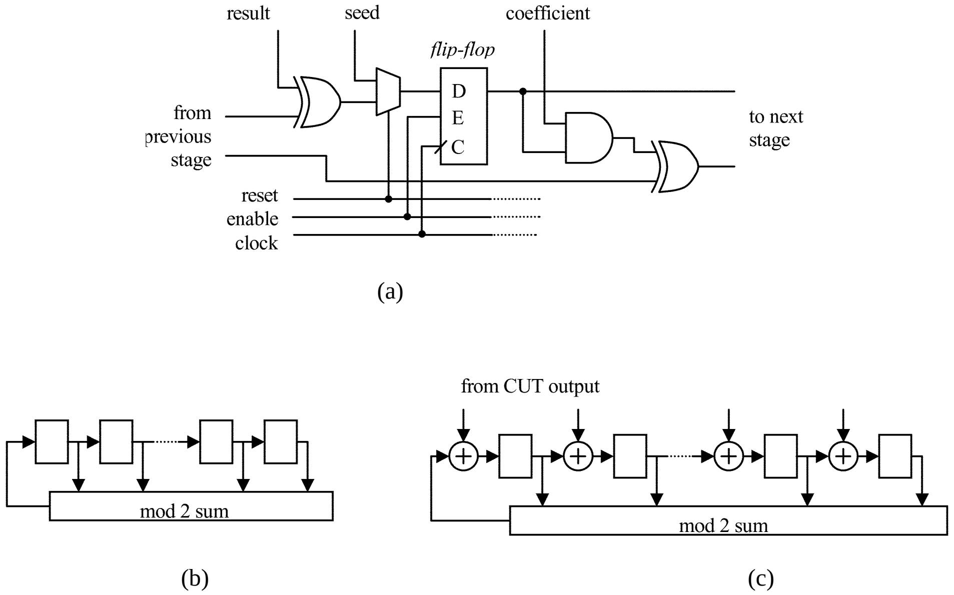

Fig. 3. LFSR-based generator and analyser: (a) single stage of the LFSR; (b) generator of the input vector; (c) analyser of the output vector.

input “coefficient” is used for feedback polynomial. The input “result” is used only for the analysis of the result and is connected to zero for input vector generation, thus masking out XOR-gates at the stage’s input. AND- and XOR-gates at the stage's output converge into a single stage of “mod 2 sum” block (Figs. 3b and 3c).

Automation of the generation of the emulation environment was rather easy because of the modular structure of the hardware part. All commonly used modules are written in VHDL that allows to parameterize the design units (Fig. 4).

- CUT – circuit under test, generated by the fault insertion program.
- CUT-pkg and CUT-top – parameters of CUT and wrapper for CUT to interface with the generic test environment, generated by the wrapper program.
- Two LFSRs – to generate test vectors and to calculate the output signature.
- Three counters – one to count test vectors, one to count test sequences and one to count modelled faults (generic VHDL module).
- Test bench with controller (FSM) to connect all submodules, to initialize LFSRs and counters, and to organize data exchange with the external interface; a generic VHDL module; algorithms implemented by the FSM are depicted in Fig. 5.
- Interface to organize data exchange between the test bench (FPGA) and the software part (Host PC).

The abstraction level of VHDL modules corresponds to register-transfer level, thus allowing the use of basically any FPGA mapping tool. The interface is currently only partly implemented because further studies are needed to define data exchange protocols between hardware and software parts. In future, any suitable FPGA board can be used, assuming that supporting interfaces have been developed to organize data exchange between the FPGA and the host PC.

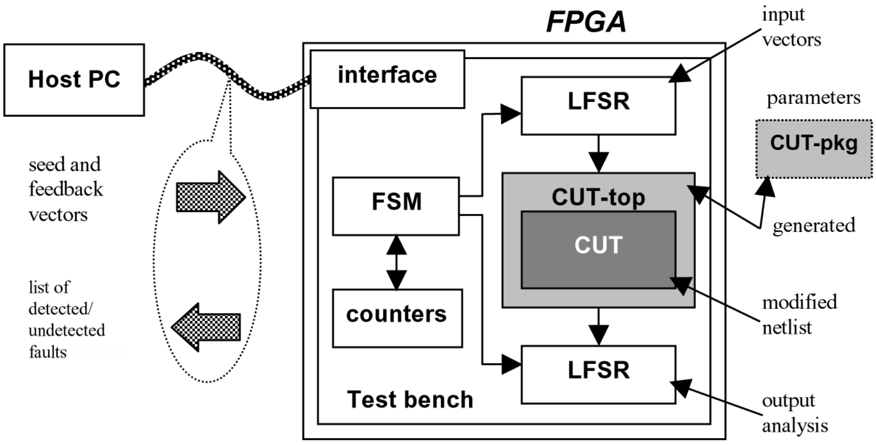

Fig. 4. Structure of the emulation environment.

```
(Combinational circuits)
reset_LFSRs;
for every test_vector
  emulate_CUT;
store_signature;
for every fault {
  reset_LFSRs;
  for every test_vector
    emulate_CUT;
  compare_signature;
  send_report;
}
```

```
(Sequential circuits)
reset_LFSRs;
for every test_sequence {
  reset_CUT;
  for every test_vector
    emulate_CUT;
}
store_signature;
for every fault {
  reset_LFSRs;
  for every test_sequence {
    reset_CUT;
    for every test_vector
      emulate_CUT;
  }
  compare_signature;
  send_report;
}
```

Fig. 5. Algorithms, implemented by the state machine.

Hardware support for fault dropping. Compared against the initial emulation environment, only minimal modifications were needed to introduce hardware support for fault dropping. First, the reference unit, the golden device, had to be inserted. For that purpose we used the original netlist of the CUT – the one without fault models. Few alternative solutions, e.g. context switching like approach with one combinational part and two sets of registers, were also considered but the used solution appeared to be the cheapest (in terms of resources and performance). Second, the circuitry for on-the-fly comparison was needed to get the full benefit of the fault dropping approach. This was implemented using a simple parallel comparator and modified emulation algorithm. Moreover, the emulation environment was also simplified because the output signature analyser was not needed any more. The modified structure (Fig. 6) makes use of the new CUT-top module along with the golden device (GOLD). The LFSR for output analysis has been replaced by the comparator (CMP). In a similar manner, the emulation algorithm (Fig. 7) has been modified. There is no need to build the fault-free signature as a comparison is performed at the end of every emulation step.

Encoded don’t-care value approach. This requires introduction of a new data type, three-valued logic represented by two-bit signals, and redefinition of all logic functions. These new definitions affect only the units under test. The rest of the environment can be used as it is, except LFSR for output analysis that must have the double amount of bits: two bits per each output in order to achieve three-valued encoding. When using fault-dropping approach, the comparator must use three-valued logic too.

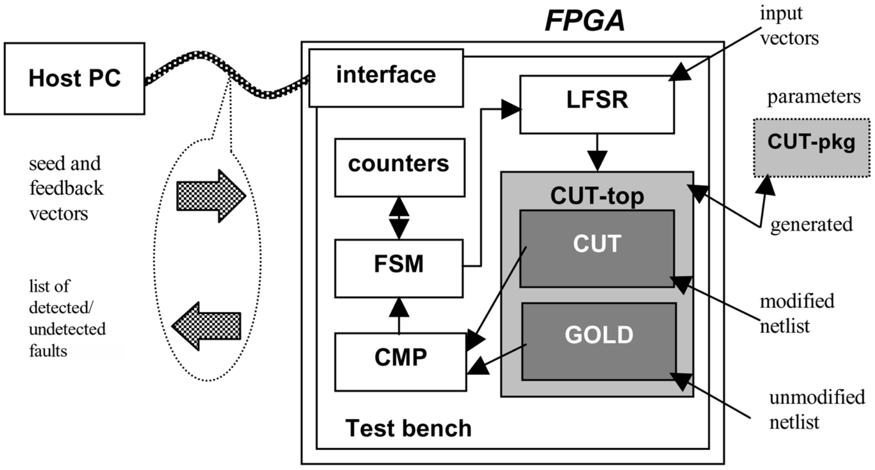

Fig. 6. Modified structure of the emulation environment.

```
(Sequential circuits & fault dropping)
for every fault {
  reset_LFSR;
  for every test_sequence {
    reset_CUT; reset_GOLD;
    for every test_vector {
      emulate_CUT;
      emulate_GOLD;
      if (outputs_differ)  break test_sequence;
    }
  }
  send_report;
}
```

Fig. 7. FSM algorithm for fault dropping.

4. RESULTS OF EXPERIMENTS

For experiments, a powerful RC1000-PPE board with VirtexE chip XCV2000E (19200 slices) was used. All the test patterns were generated by a simulation-based ATPG SBGEN from the Turbo Tester system [17,18]. In all the experiments, a fault simulator, based on parallel sequential fault simulation algorithm similar to [20], belonging to Turbo Tester, was implemented. Test circuits were selected from ISCAS'89 and HLSynt'92 benchmark sets to evaluate the speedup when replacing fault simulation with emulation on FPGA. Results of some benchmarks are presented in the paper to illustrate the gains and losses of our approach (Tables 1 and 2). Experiments with few combinational circuits – c2670, c3540, c5315, and c6288 – from ISCAS’85 benchmark are presented for comparison in Table 1. Columns labelled “# of faults” illustrate the complexity of the test circuits. The columns “# of vectors” illustrate the complexity of tests. In Table 1, it is divided into two columns: “# of seq.” shows the number of sequences and “Seq. len.” shows lengths of these sequences. In Table 2, the column “# of vectors” is divided also into to two columns, “Total” and “Actual”. The column “Total” refers to the number of clock-cycles to be simulated by the initial emulation environment without considering fault-dropping while the second column “Actual” illustrates the number of simulated cycles with the improved (fault-dropping) environment. The column “SW” gives the fault simulation time based on a parallel algorithm and “Emul.” emulation time for the same set of test vectors. Synthesis times have been added for comparison (“Synt.”). The synthesis times include both source netlist modification (fault injection) and mapping onto FPGA. Columns “Slices” give the size of the emulation environment in FPGA.

Table 1. Results of experiments without fault dropping

Circuit	# of faults	# of vectors		SW simul.	HW				Speed-up
		# of seq.	Seq. len.		Slices	MHz	Synt.	Emul.	
c2670	824	20k	1	34.8″	1783	20	5.1′	1.65″	21.1
c3540	1 036	10k	1	20.9″	1874	15	7.8′	1.39″	15.0
c5315	2 076	1000	1	5.63″	3412	12.5	15′	0.33″	17.1
c6288	3 559	1000	1	18.3″	6423	5	61′	1.43″	12.8
s5378	5 150	80	100	26.8″	3311	35	11.8′	1.18″	22.7
s15850	12 314	200	200	15.6′	9939	15	79′	32.80″	28.5
GCD (16)	1 634	80	50	5.28″	1094	40	2.8′	0.16″	33.0
GCD (32)	3 734	10	400	22.6″	2227	25	7.9′	0.60″	37.7
prefetch (16)	1 042	40	100	1.34″	776	75	1.2′	0.06″	22.3
prefetch (32)	2 252	40	400	9.46″	1529	50	3.4′	0.72″	13.1
diff-eq (16)	10 008	20	200	87.9″	7469	10	80′	4.00″	22.0
TLC	468	40	100	2.69″	391	60	41″	0.03"	89.7

Table 2. Results of experiments with fault dropping

Circuit	# of faults	# of vectors		SW simul.	HW				Speed-up
		Total	Actual		Slices	MHz	Synt.	Emul.	
s5378	5 150	8 000	2 896	26.8″	3 573	30	14.8′	0.50″	53.6
s15850	12 314	40 000	25 521	15.6′	10 131	15	85′	21.0″	44.6
GCD (16)	1 634	4 000	510	5.28″	1 152	40	2.9′	0.02″	264
GCD (32)	3 734	4 000	558	22.6″	2 456	25	9.0′	0.08″	283
prefetch (16)	1 042	4 000	181	1.34″	701	75	1.3′	3 ms	447
prefetch (32)	2 252	16 000	1 904	9.46″	1 448	50	3.6′	0.09″	105
diff-eq (16)	10 008	4 000	175	87.9″	7 710	10	82′	0.17″	517
TLC	468	4 000	925	2.69″	409	60	41″	0.01″	269

For different benchmarks, the initial hardware emulation implementing signature analysis was on average 33 (ranging from 13 to 85) times faster than the software fault simulation. For the improved environment that supported fault-dropping, the corresponding speed-up was on average 250 (from 44 to 517). It should be noted that when considering also the time of synthesis, it might not be useful to replace simulation with emulation, especially for smaller designs. Nevertheless, taking into account that sequential circuits, as opposed to combinational ones, have much longer test sequences, the use of emulation will pay off.

The second approach, encoded don't-care values, does not improve the speed-up. However, it increases the size of the emulation hardware by 50 to 100%. This is caused by the need to use two wires for every original wire to represent four different values. The logic is not always doubled because inputs and resettable/settable registers have two effective values and therefore the logic could be simplified. In Table 3, parameters in the terms of FPGA resources are compared for sequential circuits. Column "# of gates" presents the size of circuits in the terms of equivalent gates (for ASICs) and illustrates complexity of the circuits. Parameters of three implementations are compared in the remaining columns. The two most noteworthy observations that can be made from the table are the following.

- The use of fault-dropping will increase the size of the implementation but much less than expected. The explanation is rather simple – the size of the added reference circuit is comparable to the size of the removed LFSR-based output analyser.
- The increase in the size when using three-valued logic may increase the time of synthesis when the FPGA utilization is more than 50%. The reason is that when mapping random logic netlist onto FPGA, it is hard to route all interconnects. This results in significant increase of synthesis times, especially of place-and-route times.

Table 3. Parameters of emulation environments on FPGA

Circuit	# of gates	Without fault dropping			With fault dropping			Three-valued logic		
		Slices	MHz	Synt.	Slices	MHz	Synt.	Slices	MHz	Synt.
s5378	4 933	3311	35	11.8′	3 573	30	14.8′	6 036	30	86′
s15850	17 081	9939	15	79′	10 131	15	85′	14 112	15	86′
GCD (16)	926	1094	40	2.8′	1 152	40	2.9′	1 689	30	2.6′
GCD (32)	2 061	2227	25	7.9′	2 456	25	9.0′	4 148	20	5.9′
prefetch (16)	796	776	75	1.2′	701	75	1.3′	1 328	75	1.4′
prefetch (32)	1 698	1529	50	3.4′	1 448	50	3.6′	2 846	50	4.0′
diff-eq (16)	4 562	7469	10	80′	7 710	10	82′	11 859	10	221′
TLC	290	391	60	41″	409	60	41″	668	60	44″

5. CONCLUSIONS

The paper presents an FPGA-based emulation environment for fault emulation of combinational and synchronous sequential circuits. Trade-offs in terms of required FPGA resources and accuracy of test quality assessment for fault emulation have been investigated. Experiments carried out with HLSynth'92 and ISCAS'89 benchmarks showed that the proposed approach allows simulation speed-up of 40 to 500 times as compared to the state-of-the-art in software-based fault simulation.

The experiments showed that for circuits that require large numbers of test vectors, e.g. sequential circuits, it is beneficial to replace simulation with emulation. Based on that, it can be concluded that the most useful application would be to explore test generation and analysis architectures based on easily reprogrammed structures, e.g. LFSRs. This makes fault emulation very useful to select the best generator/analyser structures for BIST. The need to simulate the same circuit many times with different seed and feedback vectors reduces also the impact of the main drawback of the approach – rather large times of the synthesis. Another useful application of fault emulation would be genetic algorithms of test pattern generation where also large numbers of test vectors are analysed. Future work will include development of more advanced on-chip test vector generators and analysers.

ACKNOWLEDGEMENTS

This work was supported partly by Enterprise Estonia funded development centre ELIKO and by Estonian Science Foundation (grants Nos. 5601, 5637, 5910 and 6717). The authors also thank Valentin Tihhomirov for implementing the fault injection tool.

REFERENCES

1. International technology roadmap for semiconductors, 2006. http://www.itrs.net/
2. McGeer, P., McMillan, K., Saldanha, A., Sangiovanni-Vincetelli, A. and Scaglia, P. Fast discrete function evaluation using decision diagrams. In *Proc. IEEE/ACM International Conference on Computer-Aided Design* (Rudell, R. and Rutenbar, R. A., eds.). San Jose, 1995, 402–407.
3. Gelsinger, P. Design and Test of the 80386. *IEEE Des. Test Comput.*, 1987, **4**, 64–71.
4. Axis systems uses world's largest FPGAs from xilinx to deliver most efficient verification system in the industry. *Xilinx Press Release #0273*, 2003. http://www.xilinx.com/
5. Sedaghat-Maman, R. and Barke, E. A new approach to fault emulation. In *Proc. 8th IEEE International Workshop on Rapid System Prototyping* (Carothers, J. D., ed.). Chapel Hill, 1997, 173–179.
6. Ellervee, P., Raik, J. and Tihhomirov, V. Fault emulation on FPGA: a feasibility study. In *Proc. 21st Norchip Conference* (Nielsen, I. R., ed.). Riga, 2003, 92–95.
7. Abramovici, M. and Menon, P. Fault simulation on reconfigurable hardware. In *Proc. 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (Arnold, J. and Pocek, K., eds.). Napa Valley, 1997, 182–190.

8. Lu, S.-K., Chen, J.-L., Wu, C.-W., Chang, W.-F. and Huang, S.-Y. Combinational circuit fault diagnosis using logic emulation. In *Proc. IEEE International Symposium on Circuits and Systems*. Bangkok, 2003, vol. 5, 549–552.
9. Alderighi, M., D'Angelo, S., Mancini, M. and Sechi, G. R. A fault injection tool for SRAM-based FPGAs. In *Proc. 9th IEEE International On-Line Testing Symposium*. Kos, 2003, 129–133.
10. Hwang, S.-A., Hong, J.-H. and Wu, C.-W. Sequential circuit fault simulation using logic emulation. *IEEE Trans. CAD Integr. Circ. Syst.*, 1998, **17**, 724–736.
11. Wieler, R., Zhang, Z. and McLeod, R. D. Simulating static and dynamic faults in BIST structures with a FPGA based emulator. In *Proc. 4th International Workshop on Field-Programmable Logic and Applications* (Hartenstein, R. W. and Servít, M., eds.). Springer-Verlag, Prague, 1994, 240–250.
12. Cheng, K.-T., Huang, S.-Y. and Dai, W.-J. Fault emulation: a new approach to fault grading. In *Proc. IEEE/ACM International Conference on CAD* (Rudell, R. and Rutenbar, R. A., eds.). San Jose, 1995, 681–686.
13. Burgun, L., Reblewski, F., Fenelon, G., Barbier, J. and Lepape, O. Serial fault emulation. In *Proc. 33rd IEEE Design Automation Conference*. Las Vegas, 1996, 801–806.
14. Wieler, R., Zhang, Z. and McLeod, R. D. Emulating static faults using a xilinx based emulator. In *Proc. 3rd Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (Athanas, P. and Pocek, K. L., eds.). Napa Valley, 1995, 110–115.
15. Parreira, A., Teixeira, J. P. and Santos, M. B. Built-in self-test preparation in FPGAs. In *Proc. IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*. Tatranska Lomnica, 2004, 109–112.
16. Ellervee, P., Raik, J., Tihhomirov, V. and Tammemäe, K. Evaluating fault emulation on FPGA. In *Proc. 14th International Conference on Field-Programmable Logic and Applications* (Becker, J., Platzner, M. and Vernalde, S., eds.). Springer-Verlag, Antwerpen, 2004, 354–363.
17. Jervan, G., Markus, A., Paomets, P., Raik, J. and Ubar, R. Turbo Tester: a CAD system for teaching digital test. In *Microelectronics Education*. Kluwer Academic Publishers, 1998, 287–290.
18. "Turbo Tester" home page, 2006. http://www.pld.ttu.ee/tt
19. Pradhan, D. K., Liu, C. and Chakrabarty, K. EBIST: A novel test generator with built in fault detection capability. In *Proc. Conference on Design, Automation and Test in Europe*. Munich, 2003, 224–229.
20. Niermann, T. M., Cheng, W.-T. and Patel, J. H. PROOFS: a fast, memory efficient sequential circuit fault simulator. In *Proc. 27th IEEE Design Automation Conference*. Orlando, 1990, 535–540.

FPGA-põhine rikete emuleerimise keskkond

Peeter Ellervee, Jaan Raik, Kalle Tammemäe ja Raimund Ubar

On kirjeldatud rikete simuleerimise kiirendamise keskkonda, kasutades riistvara emuleerimist FPGA-l. Rikete simuleerimine on testimustrite genereerimise oluline alamülesanne, mida kasutatakse korduvalt testi genereerimise käigus. On antud selgitus digitaalskeemide rikete simuleerimisega kaasnevatele probleemidele. Uusimate tarkvaraliste rikete simuleerimisega võrreldes lubab artiklis esitatud lähenemine simuleerimise kiirendamist 40 kuni 500 korda. Eksperimentide tulemustest järeldub, et emuleerimist tasub kasutada skeemide puhul, mis vajavad suurt hulka testvektoreid, kuid mis kasutavad vektorite genereerimiseks paindlikke algoritmilisi lahendusi. Üheks selliseks näiteks on sisseehitatud omatest.