Efficient Simulation of Structural Faults for the Reliability Evaluation at System-Level

Michael A. Kochte, Christian G. Zoellin, Rafal Baranowski,
Michael E. Imhof, Hans-Joachim Wunderlich
University of Stuttgart
Institute of Computer Architecture and Computer Engineering
Pfaffenwaldring 47
D-70569 Stuttgart, Germany

Nadereh Hatami, Stefano Di Carlo,
Paolo Prinetto
Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24
I-10129 Torino TO, Italy

***Abstract*—In recent technology nodes, reliability is considered a part of the standard design flow at all levels of embedded system design. While techniques that use only low-level models at gate- and register transfer-level offer high accuracy, they are too inefficient to consider the overall application of the embedded system. Multi-level models with high abstraction are essential to efficiently evaluate the impact of physical defects on the system. This paper provides a methodology that leverages state-of-the-art techniques for efficient fault simulation of structural faults together with transaction-level modeling. This way it is possible to accurately evaluate the impact of the faults on the entire hardware/software system. A case study of a system consisting of hardware and software for image compression and data encryption is presented and the method is compared to a standard gate/RT mixed-level approach.**

***Index Terms*—Fault simulation, multi-level, transaction-level modeling**

I. Introduction

Structural faults model the consequences of physical defects at the gate- and logic-level. The variability and defect mechanisms in nano-scale CMOS are complex [1] and require that structural faults of VLSI circuits are considered also during functional operation [2]. The impact of faults can even depend on the application scenario [3, 4], which occupy and utilize the hardware differently.

Only a subset of the errors observed at logic-level lead to failures at system-level [5], but those that do must be accurately analyzed. The analysis of this interaction at early design stages gives important feedback for reliable [6, 7] and secure systems [8]. Usually, it is not feasible to run gate-level simulation of a complex design either for its size or model availability. Instead, multi-level simulation techniques are used. These techniques use models at different abstraction levels: models with high accuracy for the fault injection and highly abstract models for the evaluation of the consequences [5]. Only errors observable at component boundaries are propagated at a high abstraction level, without loss of accuracy. This allows to retain the advantages of structural modeling at much higher simulation speed.

Numerous approaches for the different abstraction levels have been proposed, as for example:

- Multi-level simulation of switch-level and gate-level representations [9]
- Serial simulation of structural faults in mixed-level gate-level/RTL models with event-based simulators [10, 11]
- Mixed-level fault-simulation of gate-level and RTL using concurrent simulation [12, 13]
- Simulation of structural faults in mixed-level gate-level/architectural-level simulations with symbolic simulation in the architecture-level model [14, 15]
- Serial fault injections performed at RT-level with error propagation at system-level [5]
- Injection of structural faults into mixed gate-level/high-level SystemC models [16]
- Mutator-based injection of faults into RTL and transaction-level models [17]

The work presented here is the first approach that efficiently implements concurrent multi-level fault simulation across gate- and transaction-level in an integrated simulation environment. Our work is based on a structural fault model with an efficient concurrent fault simulator at gate-level. The effects of faults observable at gate-level boundaries are propagated concurrently at transaction-level, allowing to evaluate realistic faults at system-level. The used rollback mechanism is simple to use with existing models and transaction-level simulators.

The advantage of this approach is the combination of the precision of gate-level simulation with the high simulation speed of transaction-level models (TLM, [18]). By following our methodology it is possible to analyze the effect of complex applications on system reliability.

TLMs for design exploration are reused here for reliability evaluation. The presented methodology can be used for exploration early in the design flow since only gate-level models of individual cores or components are required. This is often the case in core-based design flows.

The remainder of the paper is structured as follows: Section II describes shortly the modeling at transaction-level. Section III presents the proposed methodology and the integration of the gate-level sequential fault simulator within TLM. Section IV discusses the concurrent transaction-level fault propagation approach and its implementation. Finally, section V presents two case studies and discusses the consequences of a system-level reliability evaluation with the presented method.

1081-7735/10 $26.00 © 2010 IEEE
DOI 10.1109/ATS.2010.10

II. Transaction-Level Modeling

Increased system complexity requires the move to higher levels of abstraction in system modeling [19]. Transaction-level models (TLMs) are used to facilitate simulation-driven design space exploration and design verification [18] of large hardware/software systems with simulation speed-ups of multiple orders of magnitude over gate- and RTL modeling.

The speed-up is achieved by abstracting from signal-level communication to complex communication operations as atomic transactions. This reduces the number of events to be processed in event-driven simulators and the number of context switches between simulation processes [20]. The modularity and separation of communication and functionality in TLMs allow to quickly explore different implementation alternatives as is required in design exploration. Still, they provide enough detail to make important design decisions regarding performance, die area and power [21, 22].

In the TLM notion, functional units are modeled as *modules* with a set of concurrent processes that represent their behavior. These modules communicate by sending transactions through abstract communication *channels* with well-defined interfaces. The SystemC language and the TLM-2.0 standard [23] provide the simulation kernel, common data types and interfaces required for transaction-level modeling of bus-based System on Chip (SoC) platforms. Among others, the specified core interfaces comprise blocking and non-blocking transport interfaces which are used to send transactions between communication initiators, interconnect resources and targets.

Transaction-level modeling, e.g. with SystemC, allows to model timing behavior with different granularities, from cycle-accurate over approximately timed to untimed models [24]. The TLM-2.0 standard focuses on approximately timed models and distinguishes loosely and approximately timed models.

III. Multi-level System Model for Fault Simulation

This section describes how the models at the different abstractions levels are integrated into a single simulation environment. First, the impact of the TLM style on the accuracy of the fault injection campaign is discussed. The section then deals with the wrappers that translate transactions between the different abstraction levels and include gate-level fault simulators.

The multi-level fault simulation method proposed in this work combines the accuracy of gate-level fault simulation and the simulation speed of behavioral models. A transaction-level model of the system is augmented by precise gate-level models of components which are subject to fault injection. Fault simulation of a gate-level model determines which faults cause errors at the outputs of that model. For the subset of observable faults, functional error propagation is performed at transaction-level. The propagated error is then evaluated at system-level and it is determined whether the fault eventually results in a system failure, i.e., an error observable in the application. Here, the classification from [25] is used:

- Benign error or recoverable error
- Silent data corruption (SDC)
- Detected unrecoverable error

The fault injection used in this work is based on a mutation model for TLM [17]. Instead of random mutations, it accurately reflects the effects of structural faults on transactions issued to and from the the component subject to fault injection. The following mutations are used:

- Corruption of a parameter such as address, payload, transaction state or delay.
- A transaction falsely issued by the fault-simulated component.
- A transaction deadlock in the fault-simulated component.

Fig. 1 depicts the flow of the proposed approach. The system and the target application are modeled at transaction-level. For the hardware blocks and cores to be investigated, gate-level fault simulator instances are created using gate-level models.

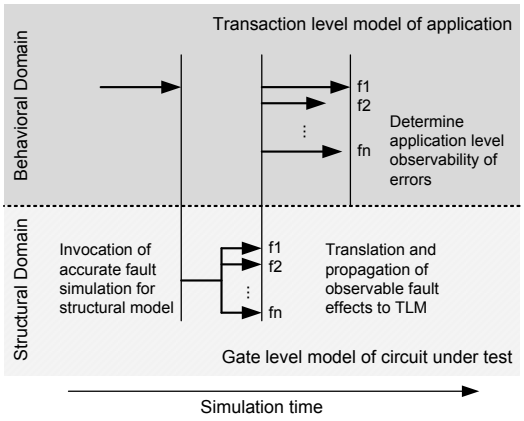

Fig. 1. Conceptual overview of the proposed multi-level fault simulation

A wrapper is used to fill the abstraction gap between the gate- and transaction-level (cf. figure 2). The wrapper encapsulates the gate-level model and translates transactions into the pin- and cycle-accurate protocol of the gate-level component. It also includes the gate-level fault simulator and requests fault propagation at transaction-level.

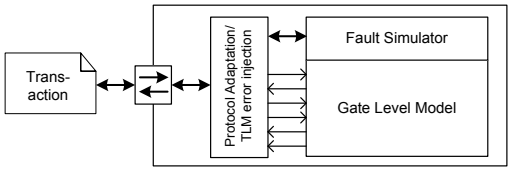

Fig. 2. Wrapper between system TLM and gate-level fault simulator

A. Transaction-Level Models

As TLMs of hardware and software modules are often used in design space exploration, they can be reused for fault simulation. In the proposed multi-level approach, only the component subject to fault injection is required to have a model at gate-level. Hence only a partial mapping of the entire system is required and binding may be limited to the components subject to fault injection. As a result, the approach can be used early to evaluate mapping and binding decisions and explore design alternatives w.r.t. reliability.

The timing accuracy of the transaction-level model can range across several orders of magnitude and the designer has great freedom in modeling the timing aspects. On the

other hand, RT- and gate-level models are usually at least cycle accurate. Obviously, there may be structural faults that can impact the temporal behavior of a sequential component and for example lead to longer completion times of certain operations. System failures due to such faults may be masked in a loosely timed TLM. In order to increase accuracy for this type of failure, adaptive timing accuracy [26] is used at least in the direct surrounding of the component subject to fault injection.

B. Wrapper for Gate-Level Models

The precision of gate-level models allows to model multiple aspects of a system that are usually not considered at transaction-level, as for example multi-valued logic, multiple clock phases and reset signals. In a gate-level component that is subject to structural fault injection, these modeling aspects may be visible at the component boundary: Some faults affect buses and cause conflicts that should be considered at transaction-level. Multiple clock phases that were previously in a known relationship become undetermined and lead to race conditions. And reset signals, due to their high fan-out, have structural faults that result in any combination of uninitialized latches or flip-flops that show up as unknowns at the gate-level/TLM boundary.

The wrapper that encapsulates the gate-level model and fault simulator is therefore responsible for both the protocol translation between transaction- and gate-level, as well as the aforementioned issues. The accurate protocol translation from transactions to the pin and cycle accurate protocol of the gate-level model is achieved by decomposing each transaction, mapping complex values to binary values, and providing additional control signals at gate-level which are not explicitly represented at transaction-level (e.g. reset or write-enable signals). The cycle and pin accurate values are processed by the synchronous fault simulation of the gate-level model, where in each simulation cycle a new data vector is passed to the simulator. The result of the simulation of each cycle is evaluated. If errors become observable at the gate-level model boundary, propagation is conducted at transaction-level as detailed in section IV, using a transaction derived from the result of the fault simulation. Since unknown values cannot easily be represented in a regular TLM, the wrapper replaces them by random values or by values for the worst or best case, depending on whether a pessimistic or optimistic bound of system reliability is to be evaluated. The exact strategy depends on the function of the given component.

During the gate-level fault simulation a large degree of parallelism can be exploited by efficient evaluation of faults, patterns and gates in parallel [27]. Here, the concurrent fault simulation algorithm [28] is used to achieve high efficiency by simulating several faults in parallel such that gains are obtained by common sensitization criteria amongst faults.

IV. Error Propagation and Evaluation in TLM

The gate-level fault simulator determines the observability of fault effects at the primary outputs of the gate-level model. To determine if a fault has any undesirable impact on system functionality, its effect (error) is propagated in the system and evaluated within the application context. This section introduces an efficient, parallel error propagation and evaluation method at transaction-level.

A. Error injection mechanism

An error that is observable at the boundary between gate- and transaction-level is injected in an atomic transaction and further propagated and evaluated in transaction-level simulation. The specific mutation of a transaction is determined by the wrapper of the gate-level model whenever the gate-level simulator requests fault propagation. To this end, an existing wrapper from functional validation (testbench) is reused and extended with means to determine mutations based on information provided by the gate-level fault simulator.

In order to keep the simulation effort low and classify faults quickly, initially just a subset of outputs at gate-level is evaluated to determine the type of mutation. For instance, if at a given time an output specifying data validity of the corresponding port is deasserted in both the fault-free and faulty machines, the data provided by the port does not need to be verified and no fault propagation at transaction-level follows. Fault propagation is also given up if the error is certain to be masked by the bus protocol. For example, error propagation is not requested if a fault affects only bus address bits that are masked out by the bus masking bits. Such faults are classified as benign already in the wrapper to avoid superfluous error propagations.

B. Evaluation of system failure conditions

The functionality of a system is checked against its specification in functional verification. A system failure is defined as a deviation of the system operation from its specification. The expected behavior included in test scenarios from functional verification is reused in our fault simulation approach to construct gate-level wrappers and to evaluate overall system behavior. In a holistic model, which also includes the environment (e.g. a stability controller within a vehicle), certain system properties can be verified under faults.

If the component subject to fault simulation is self-testing or self-checking [29], this mechanism is used for error detection and fault classification. Similarly, assertions from functional verification, which usually compose built-in model instrumentation, are also reused. Assertions implement sanity checks to find faulty states and control flow violations. At system-level they check for instance out-of-bounds exceptions.

To speed up fault simulation, the transaction-level fault propagation is halted as soon as there is enough information for fault classification. In case of signal processing applications, a checksum is calculated from the output data stream. The checksum is then evaluated and compared at intermediate checkpoints.

C. Concurrent error propagation

In order to efficiently propagate a large number of errors it is important to have an effective means of reverting to the good machine state and undoing the changes made by the propagation. In gate-level fault simulators, this is achieved by keeping track of the changes on a stack or by using tags or group IDs to identify data that differs from the good machine state. However, this is not feasible in TLM simulation since the models consist mostly of functional abstractions in the form of host-compiled code. Besides code modification, existing error injection approaches for TLM work with instrumentation of the compiled simulation binary [30] or directly with the

TLM simulation kernel [31]. But with all these methods, one simulator session can only be used for a single injection.

The error injection method proposed here is based on the concept of concurrent fault simulation with one fault-free machine and several faulty machines evaluated in parallel. The fault-free machine is running as the main process. Faulty machines are created quickly as sub-processes using operating system facilities. Since processes are protected from each other, the cost of a rollback amounts to terminating the child process that executes the faulty machine. Besides its low cost the approach is by principle also truly concurrent on host computers with multiple cores.

The approach is easily implemented on top of any existing transaction-level model. No changes to the simulation kernel are necessary and intellectual property can be used as is. The evaluation of system failure or success can be done entirely in the faulty machine. Only for the fault classification mentioned before, communication must be done between the good and faulty machine processes. However, the classification is easily enumerated and it can be communicated cheaply using the process return value upon termination of the faulty machine process.

D. Implementation

The multi-level fault simulation algorithm has been implemented based on the sequential gate-level fault simulator Hope [32] and the OSCI SystemC 2.2 and TLM-2.0 libraries.

To allow for the integration into the object oriented SystemC simulation environment, a C++ wrapper is implemented for the Hope fault simulator. In the Hope wrapper, relevant data structures and methods were exposed to obtain fault detection information and methods were added to initiate error propagation for faults visible at the gate-level boundary. Separate instances of the Hope fault simulator are dynamically created for the considered gate-level models. While the algorithmic optimizations in Hope target the stuck-at fault model, they can be extended to other structural fault models using the concept of conditional stuck-at faults [33].

Figure 3 shows the interaction between the core wrapper, the Hope fault simulator instance and the faulty machine at transaction-level. The gate-level fault simulator is part of the good machine and faulty TLM machines are created as necessary using the POSIX $fork()$ command. This allows to quickly create a faulty machine since Unix implements process forks with copy on write. Consequently, fault-free and faulty machine share the same memory regions until a memory page is modified in the faulty machine. Overall, the mechanism is transparent for many system models, but some care must be taken for file handles opened for writing in the simulation environment and the file handles should be closed in faulty machines.

V. Evaluation

The evaluation of the proposed multi-level fault simulation method concentrates on the fault classification accuracy and performance for two applications executed on an AMBA based SoC with a LEON3 processor. The SoC contains hardware accelerator cores for Triple-DES (Data Encryption Standard) as well as for two-dimensional discrete cosine transformation (2D-DCT) (cf fig. 4).

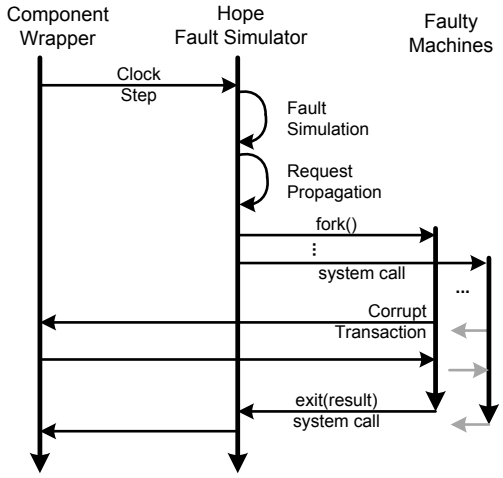

Fig. 3. Steps in Multi-Level Fault Simulation

Except for the validation, the experiments were run on a multiprocessor system with 8 Intel Xeon CPUs (2.67 GHz) and 48 GB of RAM. The memory usage did not exceed 250 MB in any of the experiments.

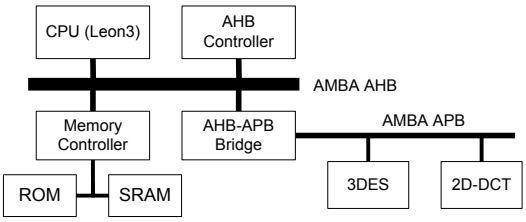

Fig. 4. SoC with Triple-DES and DCT accelerators

A. Validation

The proposed approach is validated in a traditional fault injection environment based on a state-of-the-art commercial simulator. The SoC is modeled at RT-level, except for the core subject to fault injection which is modeled at gate-level.

In each simulation run a single stuck-at fault is evaluated. The simulation is run until a result of the application is produced. A time-out is set in order to detect faults that lead to deadlocks and unacceptable delays. The simulation outcome is evaluated by the fault injection environment and the fault is classified accordingly. Due to the high computational cost, a random sample of 3000 faults per core is investigated this way.

Each fault is classified according to the categories from section IV-B. In validation experiments the following cases are discerned:

- Covered: The classification from the proposed method agrees with the validation experiments,
- False corrupt: The fault causes an SDC in the proposed method, but is benign in the validation experiments,
- False benign: The fault is benign in the proposed method, but causes an SDC in the validation experiments.

As there is no error detection mechanism in the SoC, detected unrecoverable errors do not occur (cf. section IV-B).

Validation experiments of the proposed method and the reference simulator were conducted on a farm of workstations equipped with AMD Athlon 64 Dual Core Processors (2.4 GHz) and 4 GB of RAM.

B. Triple-DES Encryption Application

The first case study is based on an encryption application utilizing the Triple-DES core in the SoC from figure 4. It encrypts a string of 64-bit words using a 64-bit key. The software part of the application is responsible of the block-wise transfer of data to the core and the read-back of results. This application is chosen as an example that exhibits almost no inherent masking.

The Triple-DES dedicated core has been obtained from OpenCores[1] and synthesized for the LSI10k generic library. It contains 19,917 logic cells and 53,010 stuck-at faults. In the following, we present the results for the system-level effects of faults in the Triple-DES core obtained by the proposed multi-level approach, and then we discuss its performance and accuracy.

Table I presents system-level fault masking in four scenarios. The first column specifies the type and length of the input data set that is encrypted. The encryption keys were chosen randomly for each scenario. In the second column, we give the number of sensitized faults, i.e., faults that produce an observable change on the core boundaries but do not necessarily lead to errors at system-level. The third and fourth column provide the number of SDCs and benign errors, respectively. In all scenarios more than 99% of faults that were sensitized, led to an error on the system-level (SDC). This is explained by the fact that the results from the core are directly transferred to the system output, so no data error masking takes place. The remaining 193 faults cause errors during inactivity of the "data ready" signal and hence they are benign.

Scenario	Faults sensitized	Silent data corruptions	Benign errors
English 3.5 KB	32916	32723	193
Italian 21 KB	33247	33054	193
Italian 20 KB	32901	32708	193
Random 8 KB	32953	32760	193

TABLE I
FAULT MASKING IN TRIPLE-DES APPLICATION

Table II gives an insight into the performance of our approach on the 8-core machine. Column "Num. sim. contexts" gives the number of fault propagations performed using *fork*. The third and fourth column provide the CPU-time spent for the concurrent fault simulator and for the execution of the TLM model, respectively. The Hope CPU-time is on average about four times longer than the execution of the TLM model. The last column provides the overall run-time of our approach, including the system CPU time for child process creation and termination. As in our experiments the fault propagation effort is low compared to the overhead of child-process creation and termination, the contribution of the system CPU time in the overall run-time was about half on average. This is still more favorable than an explicit state roll-back mechanism in a sequential fault simulator.

[1]http://www.opencores.org

Scenario	Num. sim. contexts	Gate-level Hope CPU-time	TLM CPU-time	Overall run-time
English 3.5 KB	32253	1m 03s	0m 15s	4m 51s
Italian 21 KB	32584	4m 51s	1m 08s	9m 32s
Italian 20 KB	32238	4m 36s	1m 04s	9m 13s
Random 8 KB	32290	2m 01s	0m 29s	6m 06s

TABLE II
RUN-TIME RESULTS FOR TRIPLE-DES APPLICATION (IN MIN/SEC)

The validation was performed on a random sample of 3000 faults from the full set of 53,010 faults. A string of 3,576 ASCII characters was encoded using various keys. According to the classification from section V-A, all the sampled faults were categorized as "covered" by the proposed multi-level method, i.e., no fault was mispredicted. The run-times are summarized in table III. The first column lists the type of the key used for encryption, and the subsequent columns provide the comparison between the CPU time of the validation experiments (RTL/gate) and the proposed approach (TLM/Hope), both performed on the same Athlon machine. We achieved a perfect match under an average speed-up of about 16,500x.

Scenario	CPU-time RTL/gate	CPU-time TLM/Hope
All "0"	233h	47.5s
All "1"	243h	73.2s
Sequence	234h	40.1s
Random	242h	46.1s

TABLE III
VALIDATION RESULTS FOR TRIPLE-DES APPLICATION
(RANDOM SAMPLE OF 3,000 FAULTS)

C. JPEG Encoder Application

In case of the JPEG encoding application we study the strong impact of error masking. The baseline JPEG encoding algorithm can be decomposed into four steps: (1) color transformation, (2) two-dimensional discrete cosine transform (2D-DCT), (3) quantization, and (4) lossless compression. It is performed by the SoC architecture from fig. 4. As the 2D-DCT is the most computationally expensive operation, it is accelerated by the hardware core. All other operations are performed by the LEON3 processor. The 2D-DCT core has been obtained from OpenCores and synthesized for the LSI10K library. It contains 28,001 logic cells and 78,914 stuck-at faults. In the following, we study the performance and accuracy of our approach for several case studies with various images.

The run-time results for our approach running on the previously mentioned 8 core machine are gathered in table IV. The first column specifies the type and dimensions of the image that is encoded. The remaining columns are analogous to table II. The number of simulation contexts depends on the image size, as for each 8x8 pixel block the effects of all sensitized faults that were not yet classified as SDC have to be analyzed. Due to the masking property of JPEG, a large number of error propagation occurs before the associated fault is classified as SDC and can be dropped. Due to the fault dropping, the run-time is not linear with the image size. For the image composed of 48 pixel blocks, the run-time increases just 7 times compared to the scenario with a single block.

Scenario	Num. Sim. contexts	Gate-level Hope CPU-time	TLM CPU-time	Overall run-time
White 8x8	21813	1m 00s	0m 12s	2m 04s
Black 8x8	19728	1m 02s	0m 12s	2m 06s
Noise 8x8	44739	1m 07s	0m 16s	4m 55s
Fruits 64x48	275887	8m 17s	2m 03s	34m 49s

TABLE IV
RUN-TIME RESULTS FOR JPEG APPLICATION

The validation experiments were conducted in a setting identical to the one used for Triple-DES. Due to high computational effort, validation was run for scenarios with a single 8x8 pixel image. The results are summarized in table V. It is analogous to table III except for the two additional columns that give the number of "benign faults" and the number of faults categorized as "false corrupt" (cf. section V-A) among the 3,000 faults in the sample. From 52% up to 82% of sampled faults were found benign, what is attributed to the error masking property of the JPEG quantization step. The effects of 2 to 7 faults per scenario were mispredicted and classified as "false corrupt", which is pessimistic. They were found to either result in a period of an unknown value on the "data ready" signal while the signal should be inactive, or generate additional active pulses on this signal after the data becomes invalid. In the validation experiments, these faults were classified as benign only due to the short length of the application and favorable synchronization. Under unfavorable circumstances they could in fact cause SDCs. However, even if we assume the validation experiments to be the golden reference, we achieve a match for 99.8% of faults under an average speed-up of 15,400x.

Scenario	Faults benign	False corrupt	CPU-time RTL/gate	CPU-time TLM/Hope
White 8x8	2377	6	115h	25.7s
Black 8x8	2463	7	117h	24.4s
Sequence 8x8	2067	2	119h	32.8s
Noise 8x8	1580	2	148h	33.8s

TABLE V
VALIDATION RESULTS FOR JPEG APPLICATION
(RANDOM SAMPLE OF 3,000 FAULTS)

VI. CONCLUSIONS

The presented fault simulation methodology allows to consider structural faults in a multi-level simulation at gate-level and transaction-level. Simulation time is improved by four orders of magnitude by using an efficient concurrent fault simulator at gate-level and concurrent error propagation at transaction-level. The methodology and error propagation mechanism allow to reuse TLM models from design space exploration. The accuracy of precise gate-level simulations is achieved.

ACKNOWLEDGMENT

This work has been supported by a Vigoni grant of the German Academic Exchange Service (DAAD).

REFERENCES

[1] K. Roy, T. Mak, and K. Cheng, "Test consideration for nanometer-scale CMOS circuits," *IEEE Design & Test of Computers*, vol. 23, no. 2, pp. 128–136, 2006.
[2] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE MICRO*, pp. 10–16, 2005.
[3] N. Wattanapongsakorn and S. P. Levitan, "Reliability optimization models for embedded systems with multiple applications," *IEEE Transactions on Reliability*, vol. 53, no. 3, pp. 406–416, 2004.
[4] J. Cano and D. Rios, "Reliability forecasting in complex hardware/software systems," in *Proc. of the The First International Conference on Availability, Reliability and Security (ARES)*, 2006, pp. 300–304.
[5] R. Leveugle, D. Cimonnet, and A. Ammari, "System-level dependability analysis with RT-level fault injection accuracy," in *19th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT)*, 2004, pp. 451–458.
[6] R. Leveugle and K. Hadjiat, "Multi-level fault injections in VHDL descriptions: Alternative approaches and experiments," *J. Electronic Testing*, vol. 19, no. 5, pp. 559–575, 2003.
[7] A. Jhumka, S. Klaus, and S. A. Huss, "A dependability-driven system-level design approach for embedded systems," in *Design, Automation and Test in Europe (DATE)*, 2005, pp. 372–377.
[8] K. Rothbart, U. Neffe, C. Steger, R. Weiss, E. Rieger, and A. Mühlberger, "High level fault injection for attack simulation in smart cards," in *13th Asian Test Symposium (ATS)*, 2004, pp. 118–121.
[9] W. Meyer and R. Camposano, "Active timing multilevel fault-simulation with switch-level accuracy," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 14, no. 10, pp. 1241–1256, 1995.
[10] M. B. Santos and J. P. Teixeira, "Defect-oriented mixed-level fault simulation of digital systems-on-a-chip using HDL," in *Design, Automation and Test in Europe (DATE)*, 1999, p. 549.
[11] Z. Navabi, S. Mirkhani, M. Lavasani, and F. Lombardi, "Using RT level component descriptions for single stuck-at hierarchical fault simulation," *J. Electronic Testing*, vol. 20, no. 6, pp. 575–589, 2004.
[12] S. Gai, P. L. Montessoro, and F. Somenzi, "MOZART: a concurrent multilevel simulator," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 7, no. 9, pp. 1005–1016, 1988.
[13] K. P. Lentz and J. B. Homer, "Handling behavioral components in multi-level concurrent fault simulation," in *Proc. 33th Annual Simulation Symposium (SS 2000)*, 2000, pp. 149–156.
[14] M. S. Hsiao and J. H. Patel, "A new architectural-level fault simulation using propagation prediction of grouped fault-effects," in *International Conference on Computer Design (ICCD)*, 1995, pp. 628–.
[15] O. Sinanoglu and A. Orailoglu, "RT-level fault simulation based on symbolic propagation," in *19th IEEE VLSI Test Symposium (VTS)*, 2001, pp. 240–245.
[16] S. Misera, H. T. Vierhaus, and A. Sieber, "Simulated fault injections and their acceleration in SystemC," *Microprocessors and Microsystems - Embedded Hardware Design*, vol. 32, no. 5-6, pp. 270–278, 2008.
[17] G. Beltrame, C. Bolchini, and A. Miele, "Multi-level fault modeling for transaction-level specifications," in *Proc. of the 19th ACM Great Lakes Symposium on VLSI*, 2009, pp. 87–92.
[18] F. Ghenassia, Ed., *Transaction-Level Modeling with SystemC - TLM Concepts and Applications for Embedded Systems*. Springer, 2005.
[19] A. Gerstlauer, C. Haubelt, A. Pimentel, T. Stefanov, D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1517 –1530, oct. 2009.
[20] M. Radetzki, "Object-oriented transaction level modelling," in *Advances in Design and Specification Languages for Embedded Systems*, S. Huss, Ed. Springer, 2007.
[21] Y. Hwang, S. Abdi, and D. Gajski, "Cycle-approximate retargetable performance estimation at the transaction level," in *Proc. Design, Automation and Test in Europe (DATE)*, 2008, pp. 3–8.
[22] M. Cheema and O. Hammami, "Introducing Energy and Area Estimation in HW/SW Design Flow Based on Transaction Level Modeling," in *Proc. International Conference on Microelectronics (ICM)*, 2006, pp. 182–185.
[23] Open SystemC Initiative (OSCI) TLM Working Group, "Transaction level modeling standard 2 (OSCI TLM 2)," June 2008, www.systemc.org.
[24] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *Proc. International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2003, pp. 19–24.
[25] S. S. Mukherjee, J. S. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *11th International Conference on High-Performance Computer Architecture (HPCA-11 2005), 12-16 February 2005, San Francisco, CA, USA*, 2005, pp. 243–247.
[26] M. Radetzki and R. S. Khaligh, "Accuracy-adaptive simulation of transaction level models," in *Design, Automation and Test in Europe (DATE)*, 2008, pp. 788–791.
[27] M. A. Kochte, M. Schaal, H.-J. Wunderlich, and C. G. Zoellin, "Efficient fault simulation on many-core processors," in *Proc. ACM/IEEE Design Automation Conference (DAC)*, 2010.
[28] E. Ulrich and T. Baker, "The concurrent simulation of nearly identical digital networks," in *Proc. 10th Workshop on Design Automation*, 1973, pp. 145–150.
[29] P. Lala, *Self-checking and fault-tolerant digital design*. Morgan Kaufmann, 2001.
[30] A. da Silva Farina and S. Prieto, "On the use of dynamic binary instrumentation to perform faults injection in transaction level models," in *4th International Conference on Dependability of Computer Systems*, 2009, pp. 237 –244.
[31] J. Na, "A novel simulation fault injection using electronic systems level simulation models," *IEEE Design Test of Computers*, no. Early Access Article, 2009.
[32] H. K. Lee and D. S. Ha, "Hope: An efficient parallel fault simulator for synchronous sequential circuits," in *Proc. ACM/IEEE Design Automation Conference (DAC)*, 1992, pp. 336–340.
[33] H. Wunderlich and S. Holst, "Generalized fault modeling for logic diagnosis," in *Models in Hardware Testing*. Springer, 2009, pp. 133–155.