Efficient multi-level fault simulation of HW/SW systems for structural faults

BARANOWSKI Rafal[1][*], DI CARLO Stefano[2], HATAMI Nadereh[1,2],
IMHOF Michael E.[1], KOCHTE Michael A.[1], PRINETTO Paolo[2],
WUNDERLICH Hans-Joachim[1] & ZOELLIN Christian G.[1]

[1]*University of Stuttgart, Institute of Computer Architecture and Computer Engineering,*
Stuttgart D-70569, Germany
[2]*Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino TO I-10129, Italy*

Abstract In recent technology nodes, reliability is increasingly considered a part of the standard design flow to be taken into account at all levels of embedded systems design. While traditional fault simulation techniques based on low-level models at gate- and register transfer-level offer high accuracy, they are too inefficient to properly cope with the complexity of modern embedded systems. Moreover, they do not allow for early exploration of design alternatives when a detailed model of the whole system is not yet available, which is highly required to increase the efficiency and quality of the design flow. Multi-level models that combine the simulation efficiency of high abstraction models with the accuracy of low-level models are therefore essential to efficiently evaluate the impact of physical defects on the system. This paper proposes a methodology to efficiently implement concurrent multi-level fault simulation across gate- and transaction-level models in an integrated simulation environment. It leverages state-of-the-art techniques for efficient fault simulation of structural faults together with transaction-level modeling. This combination of different models allows to accurately evaluate the impact of faults on the entire hardware/software system while keeping the computational effort low. Moreover, since only selected portions of the system require low-level models, early exploration of different design alternatives is efficiently supported. Experimental results obtained from three case studies are presented to demonstrate the high accuracy of the proposed method when compared with a standard gate/RT mixed-level approach and the strong improvement of simulation time which is reduced by four orders of magnitude in average.

Keywords fault simulation, multi-level, transaction-level modeling

1 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 also depends on the application scenarios [3, 4], which occupy and utilize the hardware differently.

---

*Corresponding author (email: baranorl@iti.uni-stuttgart.de)

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]; and 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]). This allows 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 2 describes shortly the modeling at transaction-level. Section 3 presents the proposed methodology and the integration of the gate-level sequential fault simulator within TLMs. Section 4 discusses the concurrent transaction-level fault propagation approach and its implementation. Finally, section 5 presents two case studies and discusses the consequences of a system-level reliability evaluation with the presented method.

2   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.

As TLMs of hardware and software modules are often used in design space exploration, they can be reused for fault simulation.

3 Multi-level system model for fault simulation

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.

Figure 1(a) depicts the principle 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. The system is simulated at transaction-level until a transaction with the component subject to fault simulation is requested. Upon the request, fault simulation proceeds at gate-level. If a fault causes an observable error, this error is lifted to the transaction-level. For each lifted fault, functional error propagation is performed at transaction-level. The result of the error propagation is then evaluated at system-level and it is determined whether the fault eventually results in a system failure.

A wrapper is used to fill the abstraction gap between the gate- and transaction-level (Figure 1(b)). 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 lifts faults from the gate-level to the transaction-level.

The next section defines the terminology used here, followed by the description of the modeling at transaction-level. Finally, the wrappers that translate transactions between the different abstraction levels, including the gate-level fault simulator, are detailed.

3.1 Terminology

A fault is an abstraction of one or more physical defects affecting a circuit. A fault model is the formal abstraction of a class of defects. Fault models are essential to the tractability and automation of design for testability and design for reliability. An error is the consequence of a fault that has manifested in the circuit state captured in the memory elements. A failure is the inability of the system to accomplish its target mission when subject to a fault.

Fault simulation of a gate-level model determines which faults cause errors at the outputs of that model. Fault lifting is the mapping of gate-level errors due to structural faults to a transaction-level fault model. A mutation is an instrumentation for fault injection that becomes part of a (high-level) model and modifies its behavior according to a fault model [17]. Mutation-based testing is a common approach in functional verification of high-level hardware-software systems. In the approach presented here, the mutation is chosen to map well to structural faults and not to design bugs. The instrumentation is part of the gate-level wrapper.

To classify the system level failure modes, the classification from [25] is used: Benign errors do not lead to system failure, recoverable errors do not fail but may impact the performance, silent data corruption (SDC) is the most severe failure mode, detected unrecoverable errors (DUE) allow the system to reach a safe state upon failure.

3.2 Fault lifting to the TLM

For fault simulation with high accuracy, the mapping, binding and timing of the TLM must be taken into account. Mapping is the process of selecting an actual implementation for the functional system model and determines what hardware components are required to implement the system functionality. Binding
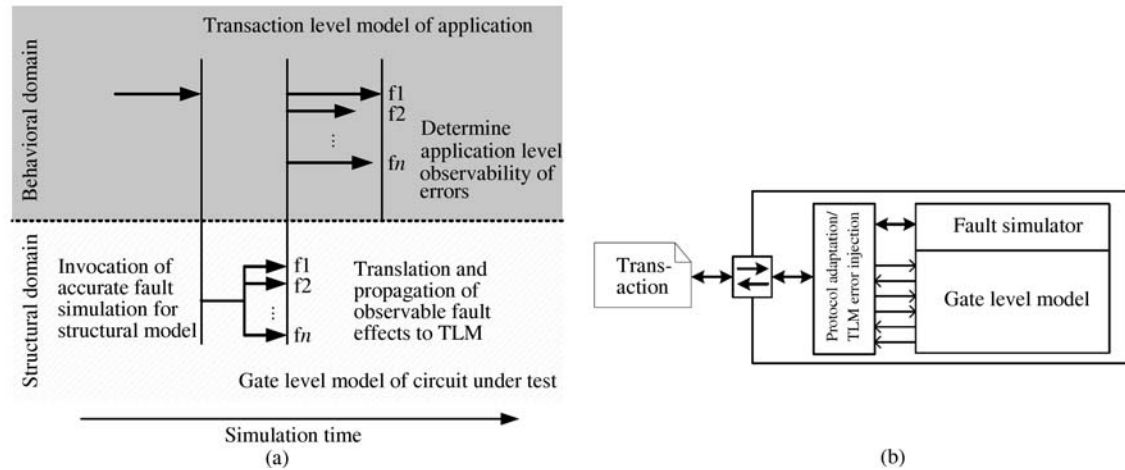
Figure 1 Conceptual overview of the proposed multi-level fault simulation. (a) Simulation sequence; (b) wrapper between TLM and gate-level.

involves scheduling the steps required to provide the system functionality and assigning them to the hardware components selected during mapping. Since TLMs are often used for design exploration, it is obvious that in early design steps not all of the components of the system may be mapped. However, error masking often does exist even in the function provided by the system itself or in the application running on the system. This inherent masking of operations in a data-flow graph is called transparency [26]. The method presented here takes full advantage of the transparency in the behavior.

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 [27] is used at least in the direct surrounding of the component subject to fault injection.

The lifting of the gate-level errors to the TLM used in this work uses the fault 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, transactions falsely issued by the fault-simulated component.

For this purpose, there must be a correspondence between the errors at gate-level and a mutation in the behavioral model. A structural fault within a gate-level module can be observable at the module's outputs. These errors can be classified into errors affecting data, address or control outputs of the gate-level module. Figure 2 depicts how these gate-level error classes relate to the mutations of the TLM.

For the fault lifting in Figure 2, we have taken advantage of the standard TLM 2.0 payload packet (i.e., TLM_GENERIC_PAYLOAD) to find this correspondence. An error at the gate-level can affect one of these parameters at TLM level. As an example, an error in the "read request" signal for the memory at gate-level model can result in the corruption of the TLM_READ_COMMAND attribute of a transaction being issued. On the other hand, a fault in the data-path that results in an erroneous data output may affect the m_data parameter of the TLM payload.

In order to deal with faults that effect the time behavior (e.g., transaction delay), the TLM model must be timing aware. Differences in transaction duration caused by faults can impact system performance or cause timeouts to be triggered. For blocking TLM calls with b_transport the transaction delay is passed
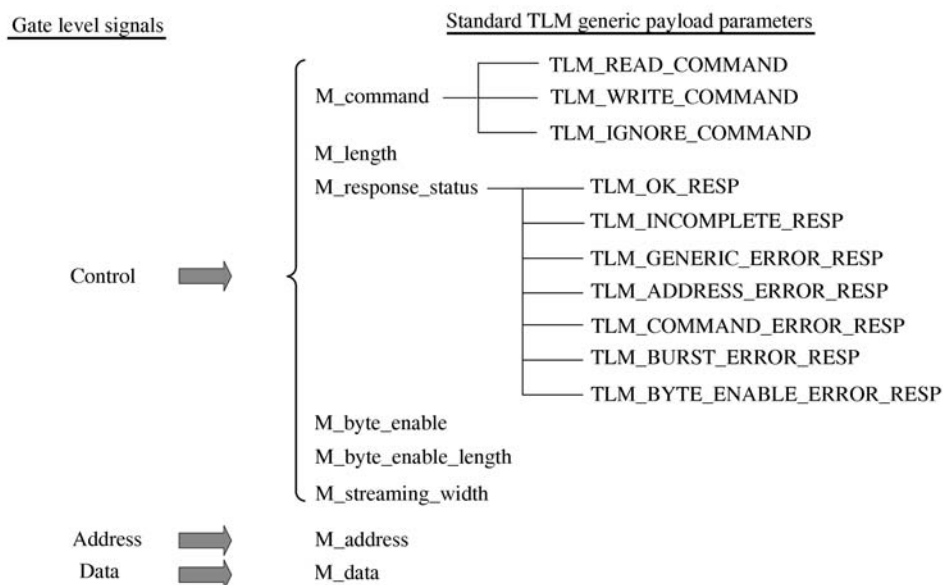
Figure 2   Gate-level to TLM fault lifting.

along with the transaction. Errors on the control lines that affect the delay are translated to the TLM delay parameter. Timeouts are reported to the originator through the TLM_INCOMPLETE_RESPONSE. For non-blocking calls with nb_transport, the actual simulator time is advanced before the response is communicated.

3.3   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 fault-free system, multi-valued logic is easily translated (e.g., for a well-behaved bus). Multiple clock phases are deterministic if their relationship is known. And reset signals will flush any unknown values from the gate-level model. 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 fault lifting. 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 4, using a transaction derived from the result of the fault simulation.

Faults can lead to transaction properties that are not part of the fault-free specification of the communication protocol. For example, faults can cause transaction types such as burst transfers that are not part of the fault-free communication. Hence, the wrapper must model more communication aspects for the fault simulation case than for the fault-free case.

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. Low-level simulators introduce a third fault class besides detected and undetected called possibly detected. Hence, a similar probabilistic consideration of unknowns must be done in the wrapper. In case of the worst case analysis, if unknown values appear at the gate-level boundary, they are propagated at the transaction-level several times: by replacing them with zeroes, ones, random values, and values that are the inverse of the good value simulation. If any propagation results in failure of the application, the fault that resulted in the unknowns is classified as possibly detected.

A special consideration in the fault simulation wrapper is timing deviation and uncertainty as outlined in the previous subsection. The wrapper must keep track of the fault simulation time, at which errors occur. To detect faults that cause the gate-level module to exceed the response time of the good simulation, the wrapper must advance the fault simulator time beyond the simulator time of the good simulation. The upper bound for this is the timeout specified by the bus model. Again, the fault simulation wrapper must model details that go beyond the specification of the wrapped model and model the allowed specification of the communication mechanism and its correspondence to errors. For example, transaction ordering errors can occur subject to faults even if the good-simulation assumes some specific ordering. Furthermore, since faults can lead to unknowns on control signals, the exact duration of a transaction can even become undetermined.

During the gate-level fault simulation a large degree of parallelism can be exploited by efficient evaluation of faults, patterns and gates in parallel [28]. Here, the concurrent fault simulation algorithm [29] is used to achieve high efficiency by simulating several faults in parallel such that gains are obtained by common sensitization criteria amongst faults.

4   Optimizations at the abstraction boundaries and in the TLM

The gate-level fault simulator determines the observability of fault effects at the primary outputs of the gate-level model. The gate-level fault simulator can take full advantage of a plethora of techniques that significantly improve the computational efficiency such as concurrent simulation of faults, special data structures and algorithmic optimizations [30]. 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.

4.1   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.

4.2 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 including also 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 [31], this mechanism is used for error detection and fault classification by checking the output for non-codewords. Such errors are communicated with an appropriate transaction response and lead to an early abort of error propagation. 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. Assertions used for functional verification are reused to detect invalid behavior earlier than the checksum mechanism.

4.3 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 [32] or directly with the TLM simulation kernel [33]. 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. The evaluation of system failure or success can be done entirely in the faulty machine. Only for the fault classification mentioned before, communication between the good and faulty machine processes must be established. However, the classification is easily enumerated and it can be communicated cheaply using the process return value upon termination of the faulty machine process.

4.4 Implementation

The multi-level fault simulation algorithm has been implemented based on the sequential gate-level fault simulator Hope [30] 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 [34].
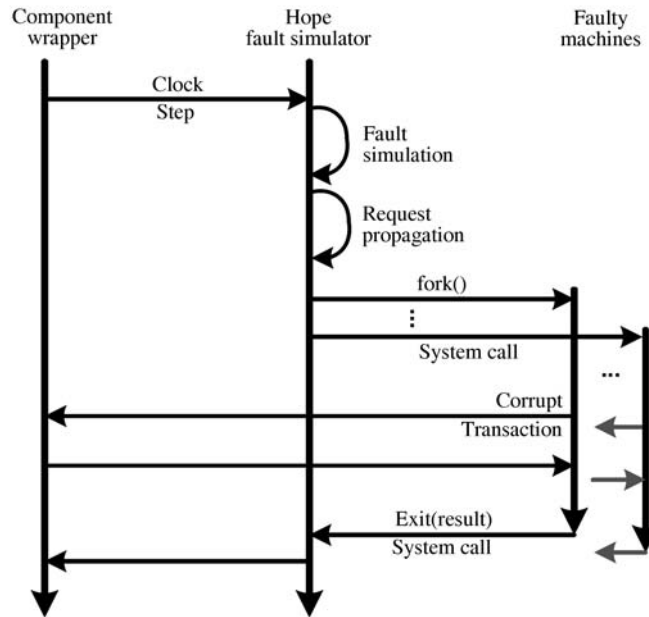
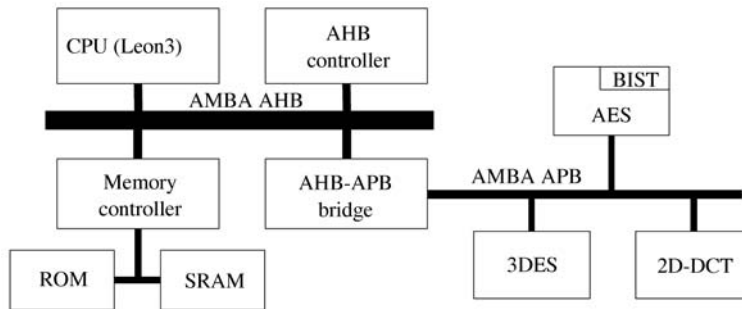**Figure 3**   Steps in multi-level fault simulation.

Figure 4   SoC with triple-DES, AES and DCT accelerators.

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.

5   Evaluation

The evaluation of the proposed multi-level fault simulation method concentrates on the fault classification accuracy and performance. We target the higher bound of fault detectability, and thus follow the approach of multiple propagation for unknown values (subsection 3.3). We perform experiments on three classes of applications executed on an AMBA based SoC with a LEON3 processor. The SoC contains hardware accelerator cores for triple-DES (Data Encryption Standard), AES (Advanced Encryption Standard), as well as for two-dimensional discrete cosine transformation (2D-DCT) (Figure 4). The AES core is equipped with built-in self-test (BIST) facilities.

Except for the validation, the experiments were run on a multiprocessor system with 8 Intel Xeon CPUs (2.8 GHz). The memory usage did not exceed 250 MB in any of the experiments.

5.1 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, so that the per-application validation effort does not exceed two weeks.

Each fault is classified according to the categories from subsection 4.2. In validation experiments the following cases are discerned: (i) covered—the classification from the proposed method agrees with the validation experiment; (ii) false corrupt—the fault causes a silent data corruption (SDC) in the proposed method, but is benign or causes a detected and unrecoverable error (DUE) in the validation experiment; (iii) false benign—the fault is benign in the proposed method, but causes an SDC or DUE in the validation experiment; (iv) false detected—the fault results in an DUE in the proposed method, but is benign or causes an SDC in the validation experiment.

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.

5.2 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 for 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 19917 logic cells and 53010 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 1 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. As there is no error detection mechanism, detected and unrecoverable errors do not occur (subsection 4.2). 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 only during inactivity of the "data ready" signal and hence they are benign.

The last four columns of Table 1 give 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 sixth and seventh column provide the CPU-time spent for the concurrent fault simulator and for the execution of the TLM model, respectively. The time needed for the child process creation and termination is included in the latter. The last column provides the overall run-time of our approach. The Hope CPU-time proved to be one fifth to one half of the total execution time.

The validation was performed on a random sample of 3000 faults from the full set of 53010 faults. A string of 3576 ASCII characters was encoded using various keys. According to the classification from subsection 5.1, 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 2. The first column lists the type of the key used for encryption, and the subsequent columns provide the comparison between the CPU

---

1) http://www.opencores.org

Table 1 Fault masking and run-time results for triple-DES application

Scenario (1)	Faults sensitized (2)	Silent data corruptions (3)	Benign errors (4)	Num. sim. contexts (5)	Gate-level Hope CPU-time (6)	TLM+Sys CPU-time (7)	Overall run-time (8)
English 3.5 KB	32916	32723	193	37983	1 m 16 s	4 m 18 s	5 m 34 s
Italian 21 KB	33247	33054	193	37396	5 m 56 s	5 m 10 s	11 m 6 s
Latin 20 KB	32901	32708	193	37809	5 m 41 s	5 m 20 s	11 m 1 s
Random 8 KB	32953	32760	193	37777	2 m 21 s	4 m 23 s	6 m 44 s

Table 2 Validation results for triple-DES application (random sample of 3000 faults)

Scenario	CPU-time RTL/gate	CPU-time TLM/Hope
All "0"	233 h	67.1 s
All "1"	243 h	88.6 s
Sequence	234 h	51.4 s
Random	242 h	53.0 s

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 13200×.

5.3 AES encryption application

The second case study is based on the self-testable AES core within the SoC from Figure 4. The core is able to encrypt a string of 64-bit words using a 64-bit key. Its BIST functionality provides a 64-bit signature that is unique for the core—any deviation in the signature indicates that the core is faulty. The software part of the application is responsible for the block-wise transfer of data to the core and the read-back of results. Similarly to the triple-DES application, AES exhibits little inherent masking.

The self-testable AES core has been obtained from the authors of [35]. It has been synthesized for LSI10k library and contains 22985 logic cells and 53850 stuck-at faults.

Table 3 presents system-level fault masking and is analogous to Table 1. The first four applications are equivalent to those discussed in the previous case study. The following four applications are similar to the first four, except that they begin with a BIST run. If a fault is detected by the BIST, it is classified as DUE. If a fault was not classified as a DUE and results in failure of the application, it is considered as SDC. If a fault is not detected by BIST and does not influence the application, the fault is benign. In each scenario more than 96% of faults that were sensitized, were either detected by BIST (DUE) or resulted in SDC. BIST was proven to detect 95% faults from the full set of 53850 stuck-ats. It failed to detect at least 1411 faults (2.6%) that led to a failure in the sixth application. The undetected faults were found to reside either at the primary inputs or at the gates in the direct neighborhood thereof. As the BIST runs with a constant seed that is applied internally, the faults at data and key inputs do not propagate and thus are not detected.

The last four columns of Table 3 provide performance details for the previously discussed scenarios. As a single BIST-run takes 512 cycles and faults can be dropped only after the run is complete, the last four scenarios require a considerable effort at the low-level, which approaches half of the overall run-time. However, as a BIST run results in massive fault dropping, the overall run-time is little affected by the length of the application itself. For this reason, even though the applications differ in the length of the text subject to encryption, they exhibit similar run-times.

Like in the previous case study, validation is performed by encoding a string of 3576 ASCII characters with various keys. A random sample of 3000 faults was chosen from the full set of 53850 faults. For all the sampled faults our approach provided the correct classification and all faults were categorized as "covered" w.r.t. subsection 5.1. The execution times for the validation experiments are summarized in Table 4, which is analogous to the previously discussed Table 2. We achieved a perfect match under an average speed-up of about 6400×.

Table 3   Fault masking and run-time results for AES application

Scenario	Faults sensitized	Silent data corruptions	Detected unrec. error	Benign errors	Num. sim. contexts	Gate-level Hope CPU-time	TLM+Sys CPU-time	Overall run-time
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
English 3.5 KB	45836	44274	–	1562	45670	24 s	1 m 59 s	2 m 23 s
Italian 21 KB	46213	44601	–	1612	45676	48 s	2 m 7 s	2 m 55 s
Latin 20 KB	46057	44478	–	1579	45646	47 s	2 m 8 s	2 m 55 s
Random 8 KB	46071	44487	–	1584	45562	31 s	2 m 0 s	2 m 31 s
BIST, Eng. 3.5 KB	52973	1334	51146	493	54639	10 m 12 s	10 m 3 s	20 m 15 s
BIST, Ita. 21 KB	53076	1411	51146	519	54716	10 m 18 s	9 m 4 s	19 m 22 s
BIST, Lat. 20 KB	52986	1328	51146	512	54633	10 m 13 s	8 m 30 s	18 m 43 s
BIST, Rnd. 8 KB	53053	1398	51146	509	54703	10 m 13 s	8 m 40 s	18 m 53 s

Table 4   Validation results for AES application (random sample of 3000 faults)

Scenario	CPU-time RTL/gate	CPU-time TLM/Hope
All "0"	278 h	21.9 s
All "1"	281 h	22.2 s
Sequence	288 h	22.1 s
Random	278 h	22.2 s
BIST, All "0"	298 h	4 m 59 s
BIST, All "1"	306 h	4 m 58 s
BIST, Sequence	290 h	5 m 6 s
BIST, Random	295 h	5 m 9 s

5.4   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: (i) color transformation, (ii) two-dimensional discrete cosine transform (2D-DCT), (iii) quantization, and (iv) lossless compression. It is performed by the SoC architecture from Figure 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 28001 logic cells and 78914 stuck-at faults. In the following, we study the performance and accuracy of our approach for several case studies with various images.

Table 5 describes the effects of faults in the 2D-DCT core. The first column specifies the type and pixel dimensions of the image that is encoded in each scenario. Subsequent columns are analogous to the previously discussed Table 1. Compared to the triple-DES and AES applications, there is a much larger proportion of sensitized faults that lead to benign errors, i.e., faults that are masked by the application although their effect is observable on the core boundaries. Among the sensitized faults, from 16% up to 36% of faults are benign. This is due to the error masking property of the quantization step.

The run-time results for our approach running on the previously mentioned 8-core machine are gathered in the last four columns of Table 5. The number of simulation contexts clearly depends on the image size, as for each 8×8 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.

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 8×8 pixel image. The results are summarized in Table 6. It is analogous to Table 2 except for the two additional columns that give the number of "benign faults" and the number of faults categorized as "false corrupt" (subsection 5.1) among the 3000 faults in the sample. From 52% up to 82% of sampled faults were found benign,

Table 5 Fault masking and run-time results for JPEG application

Scenario	Faults sensitized	Silent data corruptions	Benign errors	Num. Sim. contexts	Gate-level Hope CPU-time	TLM CPU-time	Overall run-time
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
White 8×8	25297	16295	9002	25927	56 s	1 m 15 s	2 m 11 s
Black 8×8	22729	14420	8309	25399	57 s	1 m 16 s	2 m 13 s
Noise 8×8	48794	34064	14730	47892	1 m 14 s	3 m 48 s	5 m 02 s
Fruits 64×48	64797	54141	10656	279563	9 m 11 s	21 m 5 s	30 m 15 s

Table 6 Validation results for JPEG application (random sample of 3000 faults)

Scenario	Faults benign	False corrupt	CPU-time RTL/gate	CPU-time TLM/Hope
White 8×8	2377	6	115 h	29.9 s
Black 8×8	2463	7	117 h	31.3 s
Sequence 8×8	2067	2	119 h	36.9 s
Noise 8×8	1580	2	148 h	42.8 s

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 12700×.

6 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.

References

1 Roy K, Mak T, Cheng K. Test consideration for nanometer-scale CMOS circuits. IEEE Des Test Comput, 2006, 23: 128–136

2 Borkar S. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. IEEE Micro, 2005, 25: 10–16

3 Wattanapongsakorn N, Levitan S P. Reliability optimization models for embedded systems with multiple applications. IEEE Trans Reliab, 2004, 53: 406–416

4 Cano J, Rios D. Reliability forecasting in complex hardware/software systems. In: Proceedings of the First International Conference on Availability, Reliability and Security, Vienna, Austria, 2006. 300–304

5 Leveugle R, Cimonnet D, Ammari A. System-level dependability analysis with RT-level fault injection accuracy. In: Proceedings of the 19th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, Cannes, France, 2004. 451–458

6 Leveugle R, Hadjiat K. Multi-level fault injections in VHDL descriptions: Alternative approaches and experiments. J Electron Test, 2003, 19: 559–575

7 Jhumka A, Klaus S, Huss S A. A dependability-driven system-level design approach for embedded systems. In: Proceedings of the Conference on Design, Automation and Test in Europe, Munich, Germany, 2005. 372–377

8 Rothbart K, Neffe U, Steger C, et al. High level fault injection for attack simulation in smart cards. In: Proceedings of the 13th Asian Test Symposium, Kenting, Taiwan, China, 2004. 118–121

9 Meyer W, Camposano R. Active timing multilevel fault-simulation with switch-level accuracy. IEEE Trans CAD Integr Circ Syst, 1995, 14: 1241–1256

10 Santos M B, Teixeira J P. Defect-oriented mixed-level fault simulation of digital systems-on-a-chip using HDL. In: Proceedings of the Conference on Design, Automation and Test in Europe, Munich, Germany, 1999. 549

11 Navabi Z, Mirkhani S, Lavasani M, et al. Using RT level component descriptions for single stuck-at hierarchical fault simulation. J Electron Test, 2004, 20: 575–589

12 Gai S, Montessoro P L, Somenzi F. MOZART: a concurrent multilevel simulator. IEEE Trans CAD Integr Circ Syst, 1988, 7: 1005–1016

13 Lentz K P, Homer J B. Handling behavioral components in multi-level concurrent fault simulation. In: Proceedings of the 33th Annual Simulation Symposium, Washington DC, USA, 2000. 149–156

14 Hsiao M S, Patel J H. A new architectural-level fault simulation using propagation prediction of grouped fault-effects. In: Proceedings of International Conference on Computer Design, Austin, TX, USA, 1995. 628

15 Sinanoglu O, Orailoglu A. RT-level fault simulation based on symbolic propagation. In: Proceedings of the 19th IEEE VLSI Test Symposium, Marina Del Rey, CA, USA, 2001. 240–245

16 Misera S, Vierhaus H T, Sieber A. Simulated fault injections and their acceleration in SystemC. Microproc Microsyst Embed Hardware Des, 2008, 32: 270–278

17 Beltrame G, Bolchini C, Miele A. Multi-level fault modeling for transaction-level specifications. In: Proceedings of the 19th ACM Great Lakes Symposium on VLSI, Boston Area, MA, USA, 2009. 87–92

18 Ghenassia F, ed. Transaction-Level Modeling with SystemC-TLM Concepts and Applications for Embedded Systems. NewYork: Springer, 2005

19 Gerstlauer A, Haubelt C, Pimentel A, et al. Electronic system-level synthesis methodologies. IEEE Trans CAD Integr Circ Syst, 2009, 28: 1517–1530

20 Radetzki M. Object-oriented transaction level modelling. In: Huss S, ed. Advances in Design and Specification Languages for Embedded Systems. New York: Springer, 2007

21 Hwang Y, Abdi S, Gajski D. Cycle-approximate retargetable performance estimation at the transaction level. In: Proceedings of Design, Automation and Test in Europe, Munich, Germany, 2008. 3–8

22 Cheema M, Hammami O. Introducing energy and area estimation in HW/SW design flow based on transaction level modeling. In: Proceedings of International Conference on Microelectronics, Dhahran, Saudi Arabia, 2006. 182–185

23 Open SystemC Initiative (OSCI) TLM Working Group. Transaction level modeling standard 2 (OSCI TLM 2), June 2008. www.systemc.org

24 Cai L, Gajski D. Transaction level modeling: an overview. In: Proceedings of International Conference on Hardware/Software Codesign and System Synthesis, Newport Beach, CA, USA, 2003. 19–24

25 Mukherjee S S, Emer J S, Reinhardt S K. The soft error problem: An architectural perspective. In: Proceedings of the 11th International Conference on High-Performance Computer Architecture, San Francisco, CA, USA, 2005. 243–247

26 Hiller M, Jhumka A, Suri N. EPIC: Profiling the propagation and effect of data errors in software. IEEE Trans Comput, 2004, 53: 512–530

27 Radetzki M, Khaligh R S. Accuracy-adaptive simulation of transaction level models. In: Proceedings of the Conference on Design, Automation and Test in Europe, Munich, Germany, 2008. 788–791

28 Kochte M A, Schaal M, Wunderlich H J, et al. Efficient fault simulation on many-core processors. In: Proceedings of ACM/IEEE Design Automation Conference, Anaheim, CA, USA, 2010

29 Ulrich E, Baker T. The concurrent simulation of nearly identical digital networks. In: Proceedings of the 10th Workshop on Design Automation, New Delhi, India, 1973. 145–150

30 Lee H K, Ha D S. Hope: An efficient parallel fault simulator for synchronous sequential circuits. In: Proceedings of ACM/IEEE Design Automation Conference, Anaheim, CA, USA, 1992. 336–340

31 Lala P. Self-checking and Fault-tolerant Digital Design. San Fransisco: Morgan Kaufmann, 2001

32 da Silva Farina A, Prieto S. On the use of dynamic binary instrumentation to perform faults injection in transaction level models. In: Proceedings of the 4th International Conference on Dependability of Computer Systems, Brunów, Poland, 2009. 237–244

33 Lee D, Na J. A novel simulation fault injection method for dependability analysis. IEEE Des Test Comput, 2009, 26: 50–61

34 Wunderlich H, Holst S. Generalized fault modeling for logic diagnosis. In: Models in Hardware Testing. New York: Springer, 2009. 133–155

35 Di Natale G, Doulcier M, Flottes M L, et al. Self-test techniques for crypto-devices. IEEE Trans VLSI Syst, 2010, 18: 329–333