GPU-Accelerated Simulation of Small Delay Faults

Schneider, Eric; Kochte, Michael A.; Holst, Stefan; Wen, Xiaoqing; Wunderlich, Hans-Joachim

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)
Vol. 36(5) May 2017

doi: http://dx.doi.org/10.1109/TCAD.2016.2598560

Abstract: Delay fault simulation is an essential task during test pattern generation and reliability assessment of electronic circuits. With the high sensitivity of current nano-scale designs towards even smallest delay deviations, the simulation of small gate delay faults has become extremely important. Since these faults have a subtle impact on the timing behavior, traditional fault simulation approaches based on abstract timing models are not sufficient. Furthermore, the detection of these faults is compromised by the ubiquitous variations in the manufacturing processes, which causes the actual fault coverage to vary from circuit instance to circuit instance, and makes the use of timing accurate methods mandatory. However, the application of timing accurate techniques quickly becomes infeasible for larger designs due to excessive computational requirements. In this work, we present a method for fast and waveformaccurate simulation of small delay faults on graphics processing units with exceptional computational performance. By exploiting multiple dimensions of parallelism from gates, faults, waveforms and circuit instances, the proposed approach allows for timing-accurate and exhaustive small delay fault simulation under process variation for designs with millions of gates.

Preprint

General Copyright Notice

This article may be used for research, teaching and private study purposes. Any substantial or systematic reproduction, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

This is the author's "personal copy" of the final, accepted version of the paper published by IEEE.[1]

[1] **IEEE COPYRIGHT NOTICE**
©2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

GPU-Accelerated Simulation of Small Delay Faults

Eric Schneider*, *Student Member, IEEE,* Michael A. Kochte*, *Member, IEEE,*
Stefan Holst†, *Member, IEEE,* Xiaoqing Wen†, *Fellow, IEEE,*
Hans-Joachim Wunderlich*, *Fellow, IEEE*

***Abstract*—Delay fault simulation is an essential task during test pattern generation and reliability assessment of electronic circuits. With the high sensitivity of current nano-scale designs towards even smallest delay deviations, the simulation of small gate delay faults has become extremely important. Since these faults have a subtle impact on the timing behavior, traditional fault simulation approaches based on abstract timing models are not sufficient. Furthermore, the detection of these faults is compromised by the ubiquitous variations in the manufacturing processes, which causes the actual fault coverage to vary from circuit instance to circuit instance, and makes the use of timing accurate methods mandatory. However, the application of timing accurate techniques quickly becomes infeasible for larger designs due to excessive computational requirements.**

In this work, we present a method for fast and waveform-accurate simulation of small delay faults on graphics processing units (GPUs) with exceptional computational performance. By exploiting multiple dimensions of parallelism from gates, faults, waveforms and circuit instances, the proposed approach allows for timing-accurate and exhaustive small delay fault simulation under process variation for designs with millions of gates.

***Index Terms*—small gate delay faults, fault simulation, parallel, GPU, timing-accurate, waveform, process variation**

I. Introduction

MODERN nano-scale circuit manufacturing processes involve many sources of random as well as systematic variations [1], [2]. With high performance demands and strict low-power requirements (i.e., near threshold), circuits are operated close to their physical limits and thus become highly sensitive to even slightest deviations in the physical shape of gates or interconnects [3]. These deviations can cause resistive open defects [4] and variations in the threshold voltages of transistors [5] that lead to changes in the timing behavior and result in so-called *gate delay faults* [6]. *Gate delay faults* increase the delay of a signal by slowing down transitions through the affected site by an additional amount of time. In contrast to traditional *gross delay* fault models, such as transition faults [7], the delay introduced by *small (gate) delay faults* is much smaller than the clock period, yet it may still cause a circuit to fail. Furthermore, their presence can be used as an indication of initial signs of early life failures [8], [9], which is why small delay faults cannot be ignored during test [10], [11], [12] or diagnosis [13], [14].

*E. Schneider, M. A. Kochte and H.-J. Wunderlich are with the Institute of Computer Architecture and Computer Engineering, University of Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany, Email: {schneiec,kochte}@iti.uni-stuttgart.de and wu@informatik.uni-stuttgart.de.

†S. Holst and X. Wen are with the Department of Creative Informatics, Kyushu Institute of Technology, 680-4 Kawazu, Iizuka 820-8502, Japan, Email: {holst,wen}@ci.kyutech.ac.jp.

Manuscript received March 7, 2016; revised July 15, 2016.

Delay fault simulation approaches can be categorized as logic-based, static analyses and probabilistic approaches, each of which uses a different abstraction of the timing behavior of the circuit. Logic-based *(zero-delay)* simulation approaches [7], [15] are based on multi-valued logic algebras [16] that allow to model the presence of signal transitions such as rising and falling transitions as well as static and dynamic hazards. Since the calculations of the logic-based approaches rely solely on Boolean functions, their evaluation is fast, but the accuracy suffers from both pessimistic and optimistic predictions, as no circuit timing is considered at all. Their use is limited to the evaluation of gross delay fault models [7], [15], which only focus on the presence of signal transitions at gates without considering *when* the transitions actually occur. The static analyses on the other hand allow to incorporate circuit timing data, such as individual gate delays, but they constrain themselves to corner cases that consider the calculation of the *earliest arrival* (EA) and *latest stabilization* (LS) times of signals [17], [6], [18]. While the use of intervals bounded by EA and LS times avoids the necessity of evaluating every internal signal switch, it may give a coarse estimate of the worst-case timing of a circuit suffering from a fault. Since the actual signal value within the interval is not known, uncertainty remains for faults with a particular amount of delay. As for probabilistic approaches [19], [20], the fault detection throughout the circuit is estimated using statistical delay fault distributions, which allows to determine, for example, the detection probability of a given fault size along a certain path. However, both static and probabilistic analyses are viable for small delay fault simulation only in case of robust fault propagation due to the lack of modeling all occurring switching events at a signal. Yet, not all structures can be tested robustly, as reconvergent paths can introduce glitches that cause the fault detection of tests to be invalidated [21], [22], [23], which further lowers the accuracy of these approaches.

For an accurate analysis of small delay faults, a more fine-grained evaluation in the time domain is required in order to ensure accurate signal timing and propagation, as well as to track all occurring glitches that influence the fault detection in the faulty and fault-free circuits. The authors of [24] proposed a waveform-based small delay fault simulator to evaluate the coverage of resistive open defects by simultaneous propagation of detection intervals in the time domain. Yet, the evaluation complexity during the calculation of the detection intervals drastically increases for high numbers of signal switches and fault propagation along reconvergent branches. Since all of these algorithms were designed for the sole use on regular CPUs, they entail very large runtimes even for small circuits

and quickly become inapplicable for larger problem sizes.

More importantly when considering smaller delay sizes, zero delay simulation and abstract fault modeling based on Boolean or higher-order logic [16], [15] are not sufficient to reproduce the resulting subtle delay effects [19], [21]. For example, a transition delay fault has more impact and much higher detection probability compared to a small delay fault, since the fault is assumed to be detected along all sensitized propagation paths regardless of the actual path delays. There are also cases as shown in Fig. 1, where a *slow-to-fall* small delay fault of finite size is detected by a glitch. However, a slow-to-fall transition fault at the same location would cause a constant *fault-free* (*high*) output signal in zero delay simulation and be *undetected*.

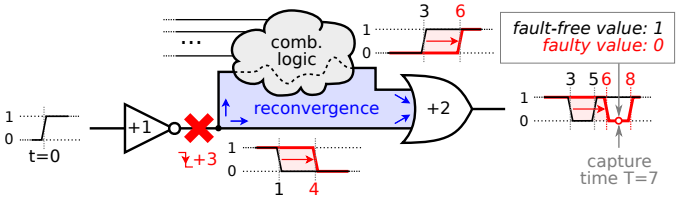


Fig. 1. Example of a *slow-to-fall* small delay fault at a reconvergent fanout gate delaying a signal by 3 time units (marked by the '✖').

Concerning random and systematic circuit variation [2], the detection of small delay faults is heavily compromised as the actual timing behavior of each gate in a circuit differs from circuit instance to circuit instance [10]. Hence, for circuits under variations the fault coverage of a given test set has to be expressed statistically [25], [26]. Considering variation in Monte-Carlo experiments causes the simulation complexity to rise quickly, as each fault has now to be evaluated not only for each pattern, but also for multiple circuit instances. Since the impact of variation on the delays is typically only a fraction of a gate delay, accurate timing simulation is mandatory for the evaluation, which poses a bottleneck due to its high runtime.

With the introduction of general purpose computing with *graphics processing units* (GPU), the paradigm of the *many-core processing* emerged [27]. GPUs contain many small multi-processing elements and are able to execute thousands to millions of *threads* of a program concurrently in a *single-instruction-multiple-data* (SIMD) fashion. This way GPU devices are capable of achieving very high throughput, which has been exploited in electronic design automation applications, e.g. for acceleration of logic simulation [28], [29] as well as parallel fault simulation [30], [31], [32], [33], [34]. However, all these algorithms focus solely on the acceleration of zero-delay logic simulation and do not provide the accuracy required for analyzing small delay faults.

In order to cope with the runtime complexity of the timing-accurate simulation, a first GPU-accelerated solution was presented in [35], which is capable of processing even industrial-sized circuits in the time-domain with full waveform histories. Although this simulator provides several orders of magnitude speedup over traditional gate-level simulation, it is still too slow to be practical for exhaustive small-delay fault simulation, especially when taking into account large numbers of different circuit instances [36]. The following core contributions presented here overcome this limitation:

- Fault-parallelism: An efficient fault-grouping method to simulate structurally independent faults in parallel in combination with structural fault collapsing allowing to reduce the simulation overhead drastically.
- An overhead-free fault injection method which maintains the high performance of the core simulator while being completely transparent during evaluation.
- Instance-parallelism: A novel approach for in-situ generation of circuit instances under variation, which reduces memory requirements and communication overhead drastically enabling efficient parallel simulation of arbitrary circuit instances.

In combination with gate-parallelism and stimuli-parallelism of the original simulator, the proposed approach provides four independent dimensions of parallelism (*gates*, *faults*, *waveforms* and *variation instances*) with marginal memory overhead and negligible communication overhead in the simulation core. This allows the simulator to fully occupy the GPU resources and to provide the highest possible performance for small delay fault simulation under variations. We confirm the practicality of our simulator by applying it to large benchmark circuits and show for the first time:

- While the transition delay fault coverage is generally higher than small delay fault coverage, hazard-based detection of small delay faults through non-robust activation and propagation is indeed quite common in large designs.
- A detailed analysis of the gains and losses in small delay fault coverage due to variation while considering non-robust detections.

The following section briefly summarizes the characteristics of current GPU-architectures and provides an overview of the proposed simulator. Section III outlines the concept of the proposed simulator for exploiting structural gate-parallelism. In Section IV the modeling of small delay faults, the injection mechanism as well as the partitioning into sets for parallel injection are presented. The concept and organization of the waveform-parallel evaluation is briefly explained in Section V, which will be extended for modeling circuit variation and parallel simulation of circuit instances with varying delays in Section VI. Experimental results reported in Section VII demonstrate that exploiting available dimensions of parallelism and efficient organization enables exhaustive small delay fault simulation even without fault dropping for multi-million gate designs, allowing for accurate computation of small delay fault coverage in presence of hazards, reconvergent fanouts and variation.

II. Background

A. GPU Architecture

With the data-parallel programming paradigm, GPU devices can massively improve the computational throughput for acceleration of high-performance computing applications [27], [37]. However, this capability also comes with certain restrictions which often pose major problems when mapping algorithms to code (so called *kernels*) for the parallel execution with many

threads. First of all, all threads have to share the same global device memory on the GPU device, which is typically limited to 4–12GB. The access to this memory is slow compared with the execution of bare arithmetic instructions. Furthermore, the amount of fast local memory that can be occupied by a single thread is scarce and has to be used efficiently in order to avoid frequent memory spilling. Care has to be taken such that each thread can run independently on its own working set, since information exchange between different threads can typically be only achieved through expensive global memory accesses and synchronization barriers. In accordance with the SIMD execution paradigm, threads are being processed in *batches* by the multi-processors, and all threads of a batch strongly need to follow the same control flow by executing the same instruction at any time. If a thread diverges from the batch, a branch in the execution is caused, which is treated serially by the thread scheduler until the control flow of the branches can be merged again. This increases the execution time. Also, any data transfers and communication between the host computer and the GPU device are serious performance bottlenecks and have to be minimized. Thus, efficient memory access and the uniformity of the kernel execution are of utmost importance for efficient parallelization [27].

B. Logic and Fault Simulation on GPUs

Recent GPU-accelerated approaches to simulate stuck-at faults [30], [31], [32], [33], [34] are based on the concurrent evaluation of independent structures in the netlist (*structural-parallelism*) for multiple inputs, such as patterns and faults (*data-parallelism*). As a general principle, each execution thread is assigned a certain structure (i.e., a single gate or a fanout-free region) and some input data to work on independently. The algorithms evaluate gates efficiently by using look-up tables and bit-level parallelism within single threads to increase data-parallelism. The authors of [28] presented a circuit simulation approach, where the circuit netlist is partitioned into clusters for an independent evaluation of output signals. By duplicating gates, each cluster can be simulated concurrently by a separate block of threads, which process the gates in parallel by exploiting structural independence within the clusters. A first event-driven solution based on a different circuit partitioning approach was presented in [29]. Here, connected gates of the netlist are partitioned into *macro-gates* whose computation by a thread block is invoked upon activation through input events at the respective structures. However, this approach also utilizes duplication to allow an independent parallel evaluation, which reduces the effective global memory usage on the GPU.

C. Timing Simulation on GPUs

So far, the above-mentioned algorithms follow similar concepts of exploiting parallelism in logic simulation, but without the consideration of the actual circuit timing. In [38], a *statistical static timing analysis* (SSTA) based on parallel evaluation of Monte-Carlo instances on GPUs was proposed. The approach accelerates delay computation by parallelized generation and evaluation of pseudo random circuit instances. However, SSTA performs a probabilistic analysis of the circuit timing and does not consider the actual propagation of signal transitions or hazards, which is not suitable for determining whether a given small delay fault is detected in a specific circuit instance. The accurate investigation of small delay faults requires *timing-accurate* evaluation methods that model all transitions and glitches of signals in the time domain. The authors in [35] proposed a GPU-accelerated time simulator for power estimation which utilizes a two-dimensional execution scheme to maximize simulation throughput by exploiting structural gate- as well as data-parallelism. All threads process different gates for different inputs concurrently, while modeling full switching histories for each signal in the circuit with floating-point timing accuracy as so-called *waveforms*.

Fig. 2 illustrates example waveforms a and b along with their vector representation [35] applied for two gates with unit delay. Each waveform is implemented as an array of times of switching events sorted in temporal order. By default, any waveform w has an initial value of $w(t) = 0$ for time $t = -\infty$. The simulation algorithm can process gate input events from earliest to latest using an efficient mergesort approach. This allows to evaluate a gate in a single pass. The resulting waveforms are stored in the global waveform memory on the device during the process. The evaluation algorithm itself utilizes an efficient data encoding and storage management in order to compute the waveforms with low memory-footprint and little synchronization overhead.

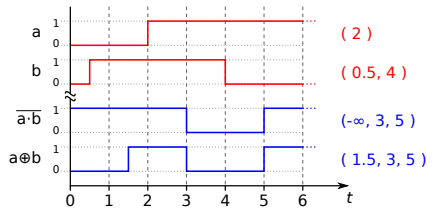


Fig. 2. Signal waveforms and their vector representation before and after passing through NAND and XOR gates with a delay of 1 time unit.

D. Overview of the Proposed Approach

The small delay fault simulation approach proposed in this work simultaneously exploits (a) *gate-parallelism*, (b) *fault-parallelism*, (c) *waveform-parallelism*, and (d) *instance-parallelism* as illustrated in Fig. 3. It adopts the two-dimensional scheme of [35] for combining gate- and pattern-parallelism, which enables fast *waveform-accurate* time simulation. Fault-parallelism is exploited by evaluating groups of structurally independent small delay faults in the same simulation instances [39]. Instance-parallelism is exploited through simultaneous processing of multiple circuit instances with varying gate delays that are generated during evaluation.

For the evaluation of a fault set under circuit variation, the *naïve serial simulation flow* of these four dimensions will be mapped to a simulation scheme as shown in Fig. 4. Given a set of circuit instances I with different delay parameters, the test responses of all stimuli provided by the test set T will be investigated for all faults of interest F, each involving the evaluation of all gates N in a circuit. Thus, the amount of

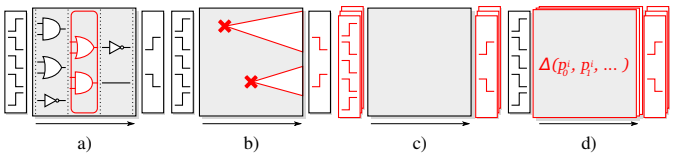


Fig. 3. Dimensions exploited for maximum throughput parallelization: a) gate-, b) fault-, c) waveform- and d) instance-parallelism.

simulation problems sums up to a total of $|N| \times |F| \times |T| \times |I|$ gate evaluations as indicated by the four nested loops. In this work we combine the dimensions of the structural problems, i.e., gates N and faults F, to exploit *structural parallelism* for simulation speedup. In addition, the evaluation of the data-specific aspects, i.e., the test-set stimuli T and the circuit instances I, will be combined as well to utilize *data-parallelism* in every step of the simulation.

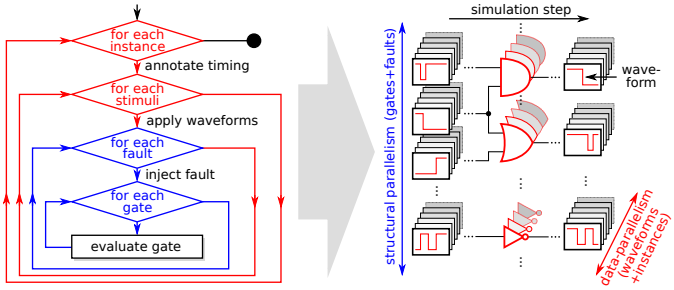


Fig. 4. Mapping of a *serial simulation* flow to the parallel evaluation scheme.

Fig. 5 shows an overview of the proposed simulation approach. The upper part (Step 1–4) consists of all necessary pre-processing steps for reading in (1) and preparation of (2) the netlist to initialize the simulator. This is followed by fault collapsing (3) and finding fault groups suitable for parallel simulation. The lower part (Step 5–8) contains the actual simulation process composed of the injection of parallel fault groups (5), the timing annotation (6) as well as waveform-accurate time simulation (7) which is followed by a fault detection kernel (8) to capture the output responses at given sample times. All shaded boxes denote parallel processings on the GPU.

This work uses a *pin-to-pin delay* model that considers individual delay annotations for each input of a cell. Furthermore it distinguishes between *rising* and *falling* transition polarities. The timing data is obtained from *standard delay format* (SDF) files [40] generated from synthesis, which are provided as input to the simulator. In the following, the applied methods of parallelization for the accelerated small delay fault simulation on GPUs will be explained in more detail.

III. Gate-Parallel Simulation

If two gates are neither in the input-cone nor in the output-cone of each other, they are considered to be mutually *data-independent* as the inputs of any of the gates do not depend on the output values of the others and vice versa. For data-independent gates, the order of evaluation does not matter and by using parallel architectures the evaluation of these gates

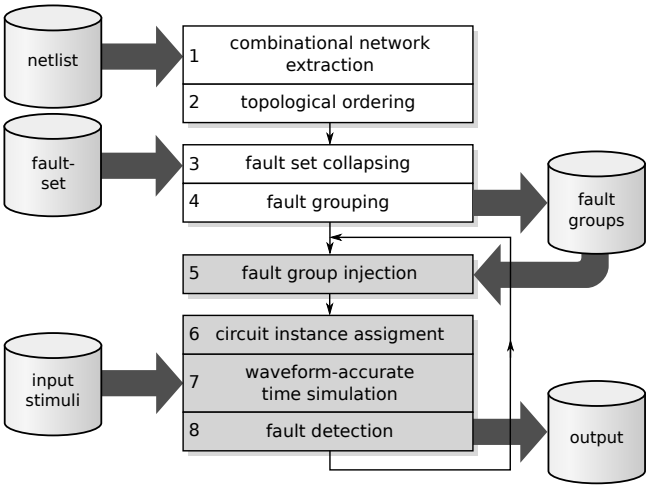


Fig. 5. Flow-chart of the overall simulation algorithm.

will be performed concurrently. For data-dependent gates on the other hand, a partially ordered evaluation sequence has to be defined first to ensure that all necessary input signals are provided before the time of their evaluation.

Fig. 6 depicts the scheme for the parallel evaluation of a topologically ordered netlist. The topological ordering partitions the combinational netlist, which is treated as a directed acyclic graph, into partitions called *levels* based on their topological distances to either circuit inputs or circuit outputs. Typically *as-soon-as-possible* (ASAP) schedules are used for the ordering which schedules nodes as soon as their predecessors have been levelized. Hence, all gates in such a partition are mutually data-independent. The amount of parallelism that can be exploited per evaluation is limited by the number of gates on each level of the circuit, which typically decreases towards the outputs in ordinary ASAP-scheduled netlists. The runtime of the simulation depends on the circuit depth (and hence number of levels), since all levels have to be evaluated in sequential order by individual invocations of the evaluation kernels.

Although the full circuit description resides in the device memory on the GPU, the working set of the threads processing a level only contains its respective gates and input waveforms. For each level, the kernel from [35] is invoked, which spawns individual threads for each gate to be evaluated. All threads simultaneously compute the output waveforms of their corresponding gates by processing the toggle events of input waveforms. Since all switching events are sorted in temporal

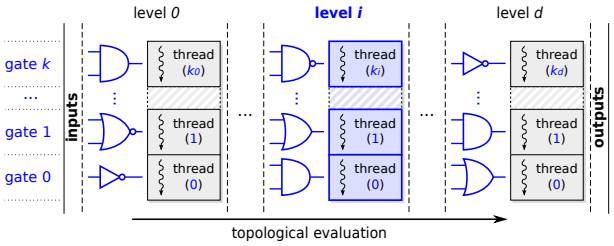


Fig. 6. Parallel evaluation sequence of data-independent gates in a topologically ordered netlist.

order, the simulation algorithm can process gate input events from earliest to latest using an efficient mergesort approach which allows for evaluation within a single pass. The resulting waveforms are stored in the global waveform memory on the device during the process. If at some point during the simulation a signal is not the input of any of the remaining gates still scheduled for evaluation, its associated waveform is de-allocated in order to free memory [35]. Therefore, the required waveform memory is bounded by the maximum number of signals that are direct input to gates on the deeper levels in the remaining simulation process.

However, for keeping the control flow of the evaluation algorithm simple, all waveforms are stored at predefined positions in the waveform memory. In order to prevent overwriting a waveform with another, each is assigned only limited storage for storing its switching information. Since the switching activity of a signal is not known a priori, *overflows* might occur during evaluation, when there is insufficient space for storing all of its toggles. Thus, if an overflow has been reported by a kernel after a first simulation, the simulation run is repeated with additional checks after the processing of each level in order to identify the culprit waveforms. This information is utilized by the host system to compute a new waveform allocation. The storage limitation of the overflown waveforms is then increased and reallocated, which is stored in the circuit description, before proceeding with the remaining evaluation on the GPU [35].

IV. Fault-Parallel Simulation

In order to reduce the number of fault locations in advance, a structural collapsing of the fault list is performed by collapsing the provided fault set into *equivalence classes* on the host system. All faults of an equivalence class have an identical behavior at the outputs of the circuit and thus require evaluation at a single *representative* fault location only. For small delay faults that affect both transition polarities (*rising* and *falling*) by the same delay amount, the equivalence rules of *transition faults* [7] are applied as follows:

- If a gate has a single input, then the fault locations at input and output pins of the gate are *equivalent*.
- If a gate has a single fanout, then the location at the output pin and the corresponding input pin of the succeeding gate are *equivalent*.

These rules obey transitivity and can further be extended to support collapsing of faults with different polarities as well.

For simulating small delay faults we employ a parallelization scheme based on groups of independent faults [39]. If the output cones of two faults share no common output logic, they are referred to as *output-independent* as they propagate to different parts in the circuit. Since these faults have no mutual influence, they can be injected in the same simulation instance for parallel evaluation. In the following, such sets of output-independent faults will be referred to as *fault groups*. To achieve a maximum simulation speedup it is favorable to process the target faults in the least number of fault groups, or equivalently by maximizing the average number of faults simulated per group. An optimal selection and scheduling can be viewed as a *minimum graph coloring* problem (also *chromatic number problem*), where each node is assigned a color such that no edge connects two nodes of the same color with the additional requirement that the number of used colors is minimal. For this the mutual output-dependence of each fault pair is mapped to an *output-dependency graph*. The nodes in this graph represent the individual faults, while two nodes are connected by an edge iff the associated faults share any output logic. Hence, an edge indicates that the associated fault pair is ineligible for parallel injection. Since after coloring all vertices of the same color in a graph are not directly connected, faults corresponding to the nodes of the same color form a valid fault group. As the graph is minimally colored (minimum number of different colors) the number of fault groups is minimal as well. However, the minimum graph coloring problem is NP-complete [41] and its optimization problem is even NP-hard, thus rendering it inapplicable to multi-million gate designs.

To enable the efficient computation of fault groups for large problem sizes, we employ a *greedy* heuristic approach that is outlined in Fig. 7. Given a particular fault set as input for the algorithm, the spatial information is derived in a first step to form an initial set of *fault locations*, such as the pins of a gate. These fault locations are then sorted in (reversed) topological order from outputs to inputs (1), which are then processed in an *as-late-as-possible* (ALAP) manner. Starting from a given fault location f, a list of reachable outputs is first determined (2) by traversing the netlist towards the circuit outputs. This list is then compared with the reachable outputs of a group in order to determine any shared logic. A map is stored for each group G that holds the reachable outputs of all faults contained. Initially, the map of a group is empty. If the outputs of a group G and a fault f are disjoint, f is inserted into G (5) and the reachable outputs of f are added to the output map of the group (6). If a shared output has been detected, the comparison process is repeated for the next group in the list (4a). If none of the currently existing groups is disjoint with f, a *new* group will be created for the fault f (4b).

Once a fault has been inserted into a group, the index of the group is assigned to the fault location and propagated towards the inputs of the circuit by annotating the nodes in the input cone of the fault (7). This back-traversal annotation is part of the heuristic as the index will be used as the starting group when trying to find a valid group assignment for subsequent scheduling of faults. The start index avoids unnecessary group comparisons when processing these faults due to the transitive structural dependency of succeeding gates, since a fault f cannot be scheduled for concurrent evaluation with the faults in its output cone. Initially, the starting group of every node is initialized with 0. For a particular fault location f, the starting group $start(f)$ is then computed as:

$$start(f) := \max\{start(g) \mid g \in \{f \cup fanout(f)\}\} + 1$$

whose value is propagated to all the nodes that share the same outputs as the fault location. This is arranged by forward-propagation towards the primary outputs followed by a back-propagation to the primary inputs. Propagation is terminated at nodes that were already assigned a higher group index. Since

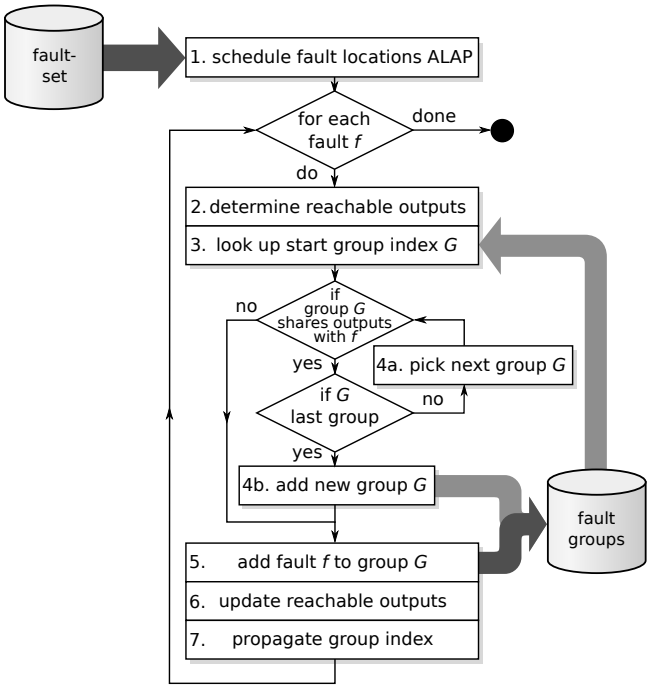


Fig. 7. Flow-chart of the greedy fault grouping heuristic.

all faults are processed in ALAP-order, the starting group of each fault allows to quickly skip all comparisons of nodes in the output cones. In general, fault groups and runtime vary depending on the order in which the faults are being processed.

An example of the fault grouping is illustrated in Fig. 8. The faults of a given fault set are sorted in topological order and processed one by one starting from outputs to inputs (a). All mutual output dependencies are illustrated in the output-dependence graph (b). Fault a is assigned the first group index $g_a = 1$, then back-propagation is performed. In the next step, fault b is processed. Since node b already knows the group of a from back-propagation of its index, it will skip comparison for g_a and start looking for output dependencies in group $g_a + 1 = 2$ instead. As the group is empty, fault b will be assigned the index $g_b = 2$. Regarding fault c, no output dependent faults have been processed yet, thus it will be assigned index 1, sharing the same group with a. The back-propagation from node c is terminated for all nodes within the red highlighted cone, as they have been already assigned a higher group index from b. Faults d and e have both received the group information from b and thus start looking in group $g_b + 1 = 3$ for dependencies. For the faults d and e, the range of the reachable outputs, denoted as O_d and O_e, are shown respectively. As both faults share no common outputs ($O_d \cap O_e = \emptyset$), they remain in the same group $g_d = g_e = 3$.

A. Fault-Injection

For each gate in a circuit, the data structure of its timing annotation is organized as a set D of tuples with pin-delay values, with each tuple representing the delays for rising and

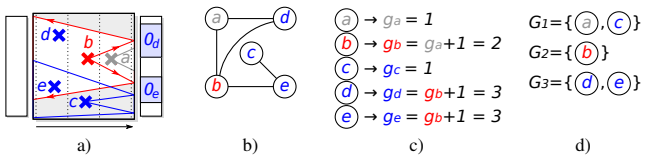


Fig. 8. Fault grouping: a) Fault set $\{a, b, c, d, e\}$ in reversed topological order, b) output-dependence graph, c) index assignment and d) resulting groups.

falling transitions at a certain input-pin of the associated gate:

$$D = \{\{d^0_{rise}, d^0_{fall}\}, \{d^1_{rise}, d^1_{fall}\}, \cdots\}.$$

The underlying simulation algorithm annotates pin-to-pin delays at the gate inputs and processes transitions at gate inputs as well with respect to their polarity. Each small delay fault f is represented by a tuple $f = (loc, \{\delta_{rise}, \delta_{fall}\})$ consisting of a particular gate pin (loc) as *fault location* and a set of delay values for the rising (δ_{rise}) and falling (δ_{fall}) transition polarity as *fault size*. The injection of a fault into the circuit is conducted by modifying the gate timing descriptions prior to the simulation run as follows:

- For a fault at an input the delay values of the fault size are added to the rising and falling delays of the associated pin timing descriptions of the affected gate.
- Faults at a gate output are modeled by injecting the delay values of the fault size into the delay descriptions of *every* input pin of the affected gate.

In addition, all gates currently subject to fault injection are marked. Yet, the presence of the injected faults is completely transparent to the evaluation kernel and thus causes no additional control flow divergence during thread execution. Since the simulator processes fault groups as a whole, all faults contained in a group are injected into the same simulation instance. After the simulation of the fault group is completed, all injected small delay faults are removed by restoring the nominal delay specification of all cells in the circuit that are currently marked. This way, the injection scheme causes only few memory operations and thus keeps data communication and synchronization at a minimum.

V. Waveform-Parallel Simulation

The time simulation algorithm (Section III) adopts a two-dimensional parallelization concept in which all gates are processed for different stimuli concurrently. In each simulation step, the evaluation kernels invoke a two-dimensional *grid* of execution threads as shown in Fig. 9. Threads in the vertical direction process the different gates at one topological level in parallel. These threads form a *slot*, which processes simulation of the circuit for a particular input stimuli combination (i.e., test vector pair). In the horizontal direction, each of the threads evaluates the same gate, yet operates on different input stimuli and hence simulates different slots. The threads are scheduled in *batches* by the thread scheduler for simultaneous execution in the multi-processing cores. During the execution, each batch evaluates the same gate for multiple stimuli. In global memory, the necessary waveform data is aligned such that the memory accesses of the threads within a batch exploit

fully utilized memory transactions. This efficient coalescing of the waveform memory access along with the caching within thread blocks reduces the overall amount of global memory transactions and maximizes the computational throughput.

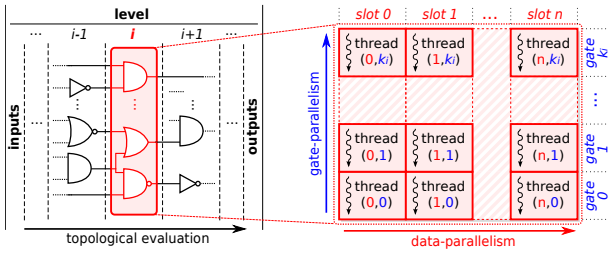


Fig. 9. Two-dimensional organization of concurrently executed threads.

The number of input stimuli that are processed in parallel depends on the available global device memory and the memory required for processing a single slot (cf. Section III). In contrast to the level-dependent gate-parallelism, the parallelism obtained from waveform stimuli remains constant throughout the simulation. If more input stimuli are provided than the memory can store at a time, the simulation run is split into a sequence of executions each processing a different bunch of stimuli. Therefore, larger memories allow for a higher degree of parallelization.

A. Response Evaluation

For the evaluation of the circuit responses, the signal values of all output waveforms are captured at a given sample time T. Again, a kernel with a two-dimensional thread-grid is used. Each thread traverses the toggle list in its associated output waveform $w(t)$ until time T is reached. The waveform value $w(T)$ then represents the captured value.

In order to determine whether a fault has been detected or not, the *syndrome* $syn(T)$ of the waveform is computed during the evaluation process. Since small delay faults are of finite size, the stabilized *good* value at each output can be acquired directly from the waveforms at $t = \infty$ without the need of an explicit *good value* reference simulation The syndrome waveform $syn(t)$ of a signal $w(t)$ is '1' iff the value $w(t)$ is the opposite of its final stable value $w(\infty)$:

$$syn(t) := w(t) \oplus w(\infty).$$

Therefore, if a fault has been detected at an output, the syndrome value of the output waveform is '1'; otherwise it is '0'. The detection of a given fault is then determined by looking up all computed syndromes of the corresponding reachable outputs. A kernel compresses the syndrome information of all outputs as a sequence of bits to allow fast access to the fault detection of each stimuli. Regarding the evaluation, the output sampling itself is not limited to a single capture time. As the computed output waveforms remain untouched during the capture process, multiple captures at different sample times can be evaluated quickly in succession. Furthermore, individual capture times can be provided for each output to model skew in the clock distribution tree.

VI. Instance-Parallel Simulation

In the following, the aforementioned *slots* (cf. Section V) will be used for modeling and evaluating circuit instances under delay variation with respect to a given *parameter space* that is spanned by devices with different delay characteristics and process corners. Variation is typically distinguished into *random* and *systematic* variation: Random variation is caused by statistical processes during production, which influence, for example, the interconnect delay by *line-edge roughness* or the transistor threshold due to *random dopant-fluctuation* [1]. Since these processes are of quantum mechanical nature involving numerous uncertainties, each gate delay is viewed as an *independent* random variable [2]. For systematic variation *spatial correlations* and dependencies are assumed between the gates (i.e., inter-die, wafer-to-wafer, lot-to-lot). These variations relate to material properties or fabrication related limitations during *lithography* or *polishing* that impact the behavior of neighboring gates in a similar way [42]. Fig. 10 illustrates the extension of the basic thread-organization of the simulation kernels for the parallel evaluation of individual circuit instances. While all the threads in the horizontal dimension still process the same gates, the threads of each slot will now process the assigned stimuli with the delays of a given circuit instance, thus maintaining control flow uniformity. This way the available GPU memory can be fully occupied.

In general, rather than keeping the raw timing data of all considered circuit instances in memory, the proposed simulation algorithm calculates the local delay deviation of the gates during runtime. For this it is assumed that the delays of each design are subject to a nominal specification that describes the mean or expected delay of each gate (i.e., the SDF-file generated from synthesis) to which the variation will be applied. When the evaluation kernel of the simulation starts, all threads of the grid compute the gate delay specification of their gate based on its nominal values and the delay distribution of the variation source. Hence, for each pin i of a gate, the resulting pin delay d_{res} is calculated by:

$$d^i_{res} := d^i_{nom} + \Delta(p^i_0, p^i_1, \ldots, p^i_n)$$

where d_{nom} is the nominal pin delay and Δ represents a variation distribution function that maps to a scalar delay offset. The latter is expressed as a function based on a set of different local parameters p_j (for $j = 0, 1, \ldots, n$) of the parameter space. A set of parameters is assigned to each slot prior to the execution of the simulation in order to link the respective slots to specific circuit instances.

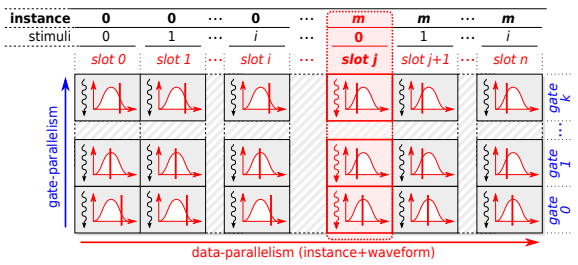


Fig. 10. Example of a thread-grid organization for the evaluation of multiple circuit instances in parallel. Each *slot* is assigned an instance and a stimuli.

In the following, the applied approaches for modeling both random as well as systematic variation for efficient computation on GPUs will be explained in more detail.

A. Random Variation

For modeling *random variation*, each slot is assigned an *instance number* in order to link the circuit instance simulated in the slot to a specific instance of the parameter space. Furthermore, the circuit timing data structure is extended to incorporate additional information about the variation into the gate annotations. During the execution of a thread, *pseudo random number generators* (PRNG) generate the random numbers for the delay calculation. These generators use a unique *seed* as a starting point for the number generation. In this work, the seeds are computed using a hash function which uses as inputs the information about the circuit instance as well as some *unique* information of the gate (e.g. layout coordinates, gate numbers or names) to be processed to introduce an independent random variable for each gate in a circuit instance. However, the spatial information can also be specified with lower granularity to form clusters of gates with the same behavior.

Since the randomness typically subjects to a given distribution (e.g. uniform or normal distribution), *mean* and *variance* of the distribution have to be specified to describe the pool of random values RV. Without any loss of generality, this work focuses on normally distributed random numbers. For the sake of simplicity, it is assumed that the expected value is $E(RV) = d_{nom}$, which is already contained in the timing data. Hence, only a variance value has to be added to the timing annotation of each gate, introducing only a negligible memory overhead.

For the computation, a parallel kernel function derives the delay deviation at the start of each gate evaluation. This function Δ requires two parameters: The gate variance as well as the aforementioned seed that is based on the slot's instance number and the current gate in order to generate the random number. Similar to [38], the parallelism of the GPU computation hides the latency of the random number generation. The normally distributed random numbers are computed by an algorithm that uses a uniform pseudo random number generator along with the well-known Box-Muller method [43]. It is assumed that the random variation affects all pins within a gate in the same manner, hence the calculation of the deviation Δ has to be performed only once for each gate, thus minimizing the computational overhead. If two slots are assigned identical instance numbers, the threads will always generate the same delays for the gates, which allows for full compliance with the general waveform-parallel approach (cf. Section V).

B. Systematic Variation

As for the *systematic variation*, the calculation of the delay deviation Δ at a specific gate is based on the evaluation of given *distribution functions*. These functions shall describe the deviation from the mean propagation delay within a parameter space of multiple dimensions as scalar values. Input to these functions are the Cartesian coordinates of the gates $(p_0, p_1, \ldots, p_n)$ in the parameter space, where each parameter p_j $(j = 0, 1, \ldots, n)$ represents a value of either spatial or parametric property. These properties can, for example, represent x- and y-coordinates of the gate in the chip or refer to specific wafers, lots, measurements or process parameters. Each parameter spans over a dimension of the parameter space that is considered to have distinct impact on the delay behavior of the gates.

The distribution functions over the parameter space spanned by all the parameters allow to model (spatial) dependencies across the different dimensions. Each *point* $(p_0, p_1, \ldots, p_n)$ or coordinate in the parameter space delivers the expected deviation of a gate as a scalar value representing an absolute or relative amount. Since systematic variation is typically spatially correlated depending on the type of process, knowledge from layout synthesis or manufacturing itself is required. These correlations can be learned from post-silicon characterization and test sweeps to extract process corners [44].

Once the relation between parameters and the delay impact has been determined, distribution functions are built in order to approximate and describe the delay behavior as a function of the parameters. This work employs a *hierarchical* approach similar to [2], [45] to model spatial correlations. The deviation is obtained from individual components, each of which is modeled at a level of hierarchy, such as inter-chip, inter-wafer or lots. The components are modeled by distinct distributions Δ_k that describe the influence of the variation on the gate delays at the given level based on the parametric properties as obtained from characterization. Each distribution function Δ_k is approximated using real functions and shall compute the delay deviation of the respective component k as a scalar value, given the coordinates p_j of a gate. When considering all components together, the resulting variation is modeled as sum of the individual components:

$$\Delta(p_0^i, p_1^i \ldots, p_n^i) := \sum_{k=0}^{n} \Delta_k(p_0^i, p_1^i, \ldots, p_{n_k}^i).$$

The functions Δ_k are approximated by polynomials or sine and cosine primitives, which allow to model even more complex distributions by approximation as series. Due to the continuity of these functions, the spatial correlations and dependencies are sustained (i.e., between neighboring gates on a chip or across regions on wafers of the same lot). The evaluation of polynomials can be processed efficiently on the GPUs when transformed according to the Horner-scheme, since it allows for extensive use of efficient *fused multiply-add* operations on recent architectures [37].

Similar to the random variation, the systematic variation is calculated and incorporated into the nominal gate delay values once at the beginning prior to the waveform processing and thus causes negligible overhead, which is hidden by the GPU parallelization. Note that the scheme of computing systematic variation can also be applied in combination to random variation. Further, it can be used to manipulate mean and standard deviation as well as to model spatially dependent random distributions.

VII. Experimental Results

The proposed simulation approach has been evaluated in a series of experiments and compared against a state-of-the-art commercial event-based logic-level time simulator. The set of benchmark circuits investigated contain the largest designs from ISCAS'89 and ITC'99 as well as industrial designs provided by NXP. All designs have been synthesized using the NanGate 45nm Open Cell Library [46]. During this process, all state elements were removed, thus leaving only the combinational circuit structure. For the evaluation of each benchmark circuit, n-detect pattern sets composed of stimuli pairs for the detection of transition faults ($n = 10$) have been generated using a commercial *automated test pattern generation* tool with pattern compaction. The patterns sets were then applied to the circuits during simulation to determine the fault coverage. A spatially-exhaustive fault set is used that considers two small delay faults affecting either rising or falling transitions for each pin of a gate (inputs and outputs).

All experiments were executed on a NVIDIA® Tesla® K80 dual-GPU-accelerator card with 2×2496 cores clocked at 824MHz, each of which has exclusive access to 12GB of global memory. However, only a single GPU was used in the experiments. Regarding the memory consumption of the circuit netlists stored on the GPU, the timing-annotated netlist description of our largest circuit (p3881k, 3.15 million gates) occupied only 202MB, which is roughly 1.7% of the total memory available on the device. Note that although the experiments have been conducted on the NVIDIA® CUDA® architecture, the general concepts and programming paradigms are applicable to other current general purpose many-core architectures. The host system contains eight Intel® Xeon® processors clocked at 3.0GHz and 128GB of RAM, though the peak memory consumption of the host never exceeded 32GB.

A. Fault Grouping

Table I contains information about the designs of the circuits and their corresponding fault sets that were grouped on the host system prior to the actual fault simulation. The name, size and logic depth of each circuit are given in the first three columns. Note that the number of gates divided by the logic depth of the circuit (levels) gives an estimate of the average *gate-parallelism* available. Columns 4 (*"Faults"*) and 5 (*"Collapsed"*) show the number of faults before and after collapsing equivalent faults. We compare the number of faults with the number of fault groups after performing the grouping. For this, we define the *effectiveness* as the number of faults *before* divided by the number of groups *after* the grouping process, which delivers the approximate speedup of the proposed approach over a naive simulation. As shown in columns 6 (*"Groups"*) and 7 (*"eff."*), the group pre-processing was able to partition the fault set into groups with an effectiveness (*eff.*) ranging from 1.2 to 773.2, which also indicates the degree of available *fault-parallelism*. For circuits p35k and p469k, the grouping is less effective, due a large number of reconvergent fanouts in the circuits and a high number of overlapping output cones of the faults. However, even in the case of p469k, an average group size of 1.2 saves almost 15 percent (≈ 45,000) of the simulation runs compared to serial execution. On the other hand, p378k has fewer reconvergent fanouts and more mutually data-independent nodes, which allows to reduce the number of simulation runs by roughly 99.9 percent given the effectiveness of over 773. For all circuits, the runtime of the grouping heuristic including the collapsing phase ranges from less than a second to few minutes as shown in the last column. This is a negligible amount of time spent for pre-processing, since the overall simulation time for processing the faults is dominated by the total number of simulation runs.

TABLE I
Circuit and fault grouping statistics.

Circuit[(1)]	Gates[(2)]	Depth[(3)]	Faults[(4)]	Collapsed[(5)]	Groups[(6)]	*eff.*[(7)]	Runtime[(8)]
s38417	15.6k	48	80.9k	46.1k	1744	26.5	460ms
s38584	19.9k	70	104.6k	59.5k	2010	29.6	565ms
b17	39.8k	120	215.1k	127.5k	15.6k	8.2	3.17s
b19	236.9k	203	1.28M	770.1k	36.5k	21.1	25.61s
p35k	42.9k	74	221.3k	113.9k	59.7k	1.9	27.64s
p45k	37.8k	57	203.1k	118.6k	9272	12.8	1.91s
p77k	63.6k	466	344.3k	205.5k	64.9k	3.2	15.33s
p78k	73.3k	50	412.4k	276.5k	1788	154.6	2.78s
p89k	88.4k	85	471.1k	266.6k	13.7k	19.4	4.66s
p100k	84.4k	108	456.1k	275.9k	9272	29.8	4.84s
p267k	184.5k	55	996.8k	591.9k	9710	61.0	10.78s
p330k	251.4k	61	1.36M	842.7k	64.1k	13.1	1:34m
p378k	366.3k	50	2.06M	1.38M	1788	773.2	17.96s
p418k	380.0k	174	2.00M	1.12M	18.6k	60.2	14.33s
p469k	103.4k	239	553.5k	308.5k	263.9k	1.2	12:46m
p500k	465.4k	179	2.46M	1.41M	21.3k	66.0	23.38s
p533k	610.6k	112	3.32M	2.03M	9248	219.8	29.29s
p951k	893.7k	153	4.62M	2.53M	13.5k	187.9	30.55s
p1522k	949.0k	508	5.13M	2.94M	65.7k	44.8	2:45m
p2927k	1.48M	388	7.77M	4.46M	32.3k	137.9	2:13m
p3881k	3.15M	178	16.66M	9.79M	82.1k	119.3	4:01m

While the grouping of exhaustive fault sets in general represents the worst case in terms of effort, the proposed heuristic has also been investigated for sparse sets. The corresponding results are summarized in Table II. In the first two columns the circuit name and the number of faults in the initial collapsed fault set are shown. The fault grouping heuristic was then applied to fault sets with different sparsity in column 3 (*"Set Sparsity"*) with each fault set containing the specified fraction of faults at random locations. For each percentage, 100 distinct random fault sets have been generated and evaluated to observe the impact on both the effectiveness (*eff.*) of the grouping (Col. 4–6 respectively for *mininum*, *average* and *maximum*) as well as the average runtime of the algorithm in the last column. As shown in Table II, when compared to the spatially-exhaustive set, the effectiveness of the fault groups obtained can vary, ranging from a 33 percent decrease to a 25 percent increase in some of the cases. This is due to the random selection of faults, as their mutual output dependencies impact the grouping result. However, in average (*avg.*) the grouping effectiveness sustains while the runtime scales linearly with the sparsity of the fault set and hence the number of faults. Therefore, in all cases the grouping for fault parallelism allows for significant reduction in the number of simulation runs at negligible runtime overhead.

TABLE II
GROUPING OF RANDOM FAULT SETS WITH DIFFERENT SPARSITY (100 REPETITIONS).

Circuit[1]	Collapsed Faults[2]	Set Sparsity[3]	Effectiveness (*eff.*) *min.*[4]	*avg.*[5]	*max.*[6]	Runtime[7]
s38584	59530	100%	–	29.6	–	565ms
		75%	28	29.6	30	388ms
		50%	28	29.6	31	346ms
		25%	27	29.7	32	250ms
		10%	25	29.5	35	195ms
		5%	22	29.4	37	172ms
p500k	1405130	100%	–	66.0	–	23.88s
		75%	65	66.0	66	17.61s
		50%	65	66.1	67	12.88s
		25%	64	66.1	67	6.98s
		10%	62	66.2	69	3.41s
		5%	59	66.2	75	2.45s
p951k	2534274	100%	–	187.9	–	30.55s
		75%	186	187.9	190	24.16s
		50%	183	187.7	192	15.03s
		25%	181	187.4	193	9.77s
		10%	178	185.6	198	6.41s
		5%	171	183.6	196	5.18s

TABLE III
RUNTIME COMPARISON.

Circuit[1]	Pattern-Pairs *10det.*[2]	Pairs *max.*[3]	Fault-Free Simulation Event-Based[4]	Worst (GPU) Time[5]	Worst (GPU) X[6]	Best (GPU) Time[7]	Best (GPU) X[8]
s38417	348	29504	6.92s	227ms	30	67ms	103
s38584	563	24864	1:09m	314ms	217	79ms	864
b17	2135	33120	1:25m	1.21s	69	221ms	380
b19	4651	5408	0:48h	12.79s	224	1.95s	1469
p35k	4096	11744	3:40m	1.17s	187	321ms	683
p45k	2417	14048	1:53m	848ms	132	225ms	500
p77k	1979	13536	6:27m	5.91s	65	422ms	916
p78k	150	11744	40.34s	827ms	48	135ms	298
p89k	2460	8480	4:11m	2.06s	121	405ms	618
p100k	2809	8288	8:40m	2.50s	207	457ms	1135
p267k	3181	3360	0:17h	3.74s	260	1.03s	946
p330k	5928	3104	1:15h	11.65s	381	2.40s	1853
p378k	200	2400	5:24m	3.56s	90	595ms	543
p418k	3676	1888	0:39h	21.07s	109	2.54s	910
p469k	347	17056	0:19h	2.75s	397	373ms	2926
p500k	5012	1792	1:40h	39.08s	153	4.54s	1317
p533k	3417	1312	1:44h	38.15s	162	4.51s	1378
p951k	7063	544	3:45h	3:30m	64	20.04s	672
p1522k	17980	800	11:55h	9:31m	75	32.68s	1312
p2927k	22107	416	27:29h	0:31h	53	1:23m	1194
p3881k	12091	224	30:15h	2:18h	13	6:21m	286

B. Runtime

Table III compares the runtime of the proposed simulation approach to the commercial event-based time-simulator running on the same host system as a single thread. The first three columns list the circuit names, the number of stimuli pairs in the 10-detect pattern set (*10det.*) and the *maximum* number of stimuli pairs that can be processed simultaneously on the GPU (*waveform-parallelism*) with the current memory configuration of 12GB of global memory on the GPU device. If the generated pattern set is smaller than the maximum number of patterns able to fit on the GPU, the thread-grid dimensions have been reduced accordingly. This avoids spawning and management of unnecessary threads that would otherwise be scheduled for evaluation. These unused slots can later be utilized for instance parallel simulation. Since the commercial solution is in general too slow, only a fault-free simulation run has been performed in order to enable a comparison. The fourth column (*"Event-Based"*) shows the average runtime of the event-based commercial simulator for processing all input stimuli. The runtimes of the proposed approach are split into *worst-case* and *best-case*. In the first run, the GPU-accelerated time simulator achieved runtimes 13–397× times faster than the event-based solution (Col. 5–6, *"Worst (GPU)"*). If the simulation is repeated, the overhead for the memory management of the waveform reallocation is reduced and eventually the simulator is able to run at full speed, resulting in a higher speedup. This yields an additional improvement of one order of magnitude on average (Col. 7–8, *"Best (GPU)"*). Note that the speedup of the simulation run is independent of the faults as the presence of faults is completely transparent to the evaluation kernel which allows for efficient exhaustive small delay fault simulation.

C. Small Delay Fault Detection

Table IV compares the fault simulation coverage results of both transition and small delay faults for the applied 10-detect transition fault test patterns. In column 2 (*"Faults"*) the number of fault locations is reported. Again, a spatially exhaustive fault set is used. Given the nominal clock frequency obtained from static timing analysis, the size of each small delay fault has been set halfway between the slack of the longest and shortest path through each fault location. The third column (*"Runtime"*) contains the runtime of the small delay fault simulation *without* fault dropping. Hence, the detection information of every pattern is obtained for each fault. Since the fault simulation is the repeated execution of simulation runs with the same pattern set for different injected fault groups, the memory calibration quickly converges, which results almost exclusively in full-speed runs. Columns 4 and 5 (*"Transition (TD)"*) contain the number of detected transition faults ("*TD det.*") and the portion of small delay faults that have not been detected at the same location ("$\supseteq$ *SD und.*"). As shown in Table IV, a fair amount of small delay faults could not be detected although the corresponding transition faults were detected, confirming the well-known fact that transition faults overestimate the small delay fault coverage.

The last two columns (*"Small Delay (SD)"*) summarize the detections for the small delay faults respectively. As expected, the coverage of the small delay faults is generally much lower than for transition faults. However, there are also numerous cases where small delay faults are detected, but their corresponding transition faults are not detected. Here, the small delay faults were propagated along reconvergent fanouts causing glitches to appear at the circuit outputs while the output signals were being captured as previously depicted in Fig. 1. These faults were detected only for smaller (*finite*)

TABLE IV
FAULT DETECTION OF TRANSITION DELAY (TD) AND SMALL DELAY (SD) FAULTS AT SAME LOCATIONS.

Circuit[1]	Faults[2]	Runtime[3]	Transition (TD)		Small Delay (SD)	
			TD *det.*[4] ⊇ SD *und.*[5]		SD *det.*[6] ⊇ TD *und.*[7]	
s38417	46138	32.11s	45965	12198	33777	10
s38584	59530	40.18s	57838	20242	37734	138
b17	127490	0:38h	124979	60557	64443	21
b19	770082	27:21h	764557	216023	549854	1320
p35k	113946	3:49h	111820	39172	72702	54
p45k	118608	0:26h	118287	34138	84242	93
p77k	205478	7:12h	188221	78187	110500	466
p78k	276486	4:59m	276486	37757	238729	0
p89k	266576	1:15h	264314	95266	169049	1
p100k	275948	1:40h	274753	66758	208199	204

fault sizes that cannot be represented by transition faults. Although these cases seem rare, they are especially important for diagnosis and failure analysis, thus emphasizing the *necessity* of fast and accurate small delay fault simulation.

D. Variation Impact

The generated 10-detect pattern set has further been evaluated under random variation for a population of different circuit instances each of which has distinct gate delays. For the distribution function Δ of the pin delays, a Gaussian normal distribution $\mathcal{N}(\mu, \sigma^2)$ has been chosen with mean $\mu = d_{nom}$ and standard deviation of $\sigma = 0.2 \cdot \mu$. In order to avoid excessive occurrences of timing failures due to the variation, the sample time T for all test cases (nominal and instances under variations) has been set to 1.5× of the longest path delay (nominal). Again, the fault sizes of the small delay faults have been chosen halfway between minimum and maximum slack at the fault sites with respect to the nominal instance. During simulation, the aforementioned thread-grid organization for parallel instance evaluation (cf. Fig. 10) has been applied.

Table V depicts the impact of the variation on the detection of small delay faults for a population of 100 random circuit instances. For each circuit the fault detections in nominal and variation instances are compared and further classified as *gains* or *losses* based on the difference as shown in column 2 (*"Detection Class"*). A fault detection is considered a *gain*, if a fault was detected in the instance under variation, while it was undetected in the nominal case. Analogously, a fault detection is called a *loss*, if the fault was detected in the nominal instance, but not in the instance under variation. As shown in columns 3–8 (*"Affected Instances"*), gains and losses due to variation occur throughout the circuit population with different impact.

Note that the instances stated in each column are supersets of the instances stated in the columns to their right. For example in p35k, 3752 faults were undetected in the nominal instance, but have been detected in at least one random circuit instance. Out of these, 1392 faults have been observed in even more than 20% of the circuit population. In fact, in all of the investigated circuits almost 50% of the fault detections *influenced by variation* showed impact in more than 20% of the instances, while losses occurred more frequently than gains. In p35k, ten previously undetected faults even showed gains in *all* of the variation instances due to captured glitches in the nominal case. Altogether, the average impact of the variation on the detection has been observed in almost 10% of the faults in the fault universe of each circuit, ranging up to 16% in case of s38584. Thus, variation should be taken into account to reason about the robustness of the small delay fault detection in presence of circuit variation [26]. For this, the proposed simulation approach can be efficiently applied, since it exploits the dimensions of parallelism from gates, faults, waveforms and variation in order to cope with the computational complexity.

TABLE V
OCCURRENCES OF FAULT DETECTIONS INFLUENCED BY VARIATION IN 100 RANDOM CIRCUIT INSTANCES.

Circuit[1]	Detection Class[2]	Affected Instances					
		>0%[3]	>20%[4]	>40%[5]	>60%[6]	>80%[7]	all[8]
s38417	gain	1802	872	514	232	50	0
	loss	3246	930	80	0	0	0
s38584	gain	3322	1124	444	108	32	0
	loss	6392	2720	710	2	0	0
b17	gain	5254	2344	1240	404	76	2
	loss	6806	2180	564	52	0	0
p35k	gain	3752	1392	682	258	58	10
	loss	4274	1098	448	134	14	0
p45k	gain	3068	1452	718	158	18	2
	loss	7404	1514	264	2	0	0
p78k	gain	3470	1968	1134	480	92	0
	loss	8138	1380	162	0	0	0
p89k	gain	11532	4486	2040	506	136	6
	loss	16716	4458	798	14	0	0
p100k	gain	8838	4134	2130	510	102	24
	loss	10904	3132	660	16	0	0

VIII. CONCLUSIONS

In this work an approach for enabling fast and accurate simulation of small delay faults on data-parallel GPU architectures is presented. The fault simulation is waveform-accurate and considers individual rising and falling pin-to-pin delay annotations as well as glitch filtering. It maintains full information about the switching activity and allows to determine the coverage of small delays in the presence of hazards and reconvergences. Furthermore, random as well as systematic variation can be incorporated during the evaluation by modifying gate delays during runtime in order to investigate the robustness of fault detection. Rather than focusing on a latency-optimized evaluation, the proposed method utilizes many dimensions of parallelism that can be found in circuit simulation (gates, faults, waveforms, instances) along with careful memory management to attain maximum simulation throughput and speedup. Runtime results of the approach have shown speedups of up to three orders of magnitude compared to conventional logic-level timing simulators. With this significant simulation speedup, the proposed approach enables for the first time a waveform-accurate and exhaustive simulation

of small delay faults under variation for large industrial designs with more than a million gates. Future work will include the application of the waveform-accurate simulation algorithm for accurate power and IR-drop estimation to investigate the impact on circuit timing and fault detection in the presence of variation for multi-million gate designs.

ACKNOWLEDGMENT

This work has been funded by the German Research Foundation (DFG) within the project PARSIVAL under contract WU 245/16-1 as well as the Japan Society for the Promotion of Science (JSPS) under JSPS Grant-in-Aid for Scientific Research (B) #25280016, JSPS Grant-in-Aid for Scientific Research on Innovative Areas #15K12003, JSPS Grant-in-Aid for the Promotions of Bilateral Joint Research Projects (Japan-Germany) and the German Academic Exchange Service (DAAD) as part of the exchange project #57155440 in collaboration with the JSPS.

REFERENCES

[1] S. R. Nassif, "Modeling and Analysis of Manufacturing Variations," in *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, May 2001, pp. 223–228.

[2] A. Srivastava, D. Sylvester, and D. Blaauw, *Statistical Analysis and Optimization for VLSI: Timing and Power*. Springer, 2005.

[3] ITRS. (2013) The International Technology Roadmap for Semiconductors: 2013. [Online]. Available: http://www.itrs2.net/2013-itrs.html [Accessed: Feb. 2, 2016]

[4] R. Rodríguez Montañés, J. P. de Gyvez, and P. Volf, "Resistance Characterization for Weak Open Defects," *IEEE Design Test of Computers*, vol. 19, no. 5, pp. 18–26, Sep. 2002.

[5] Y. Taur, D. A. Buchanan, W. Chen, D. J. Frank, K. E. Ismail, S.-H. Lo, G. A. Sai-Halasz, R. G. Viswanathan, H.-J. C. Wann, S. J. Wind, and H. S. Wong, "CMOS Scaling into the Nanometer Regime," *Proceedings of the IEEE*, vol. 85, no. 4, pp. 486–504, Apr. 1997.

[6] A. K. Pramanick and S. M. Reddy, "On the Fault Coverage of Gate Delay Fault Detecting Tests," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 1, pp. 78–94, Jan. 1997.

[7] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar, "Transition Fault Simulation," *IEEE Design Test of Computers*, vol. 4, no. 2, pp. 32–38, Apr. 1987.

[8] Y. M. Kim, Y. Kameda, H. Kim, M. Mizuno, and S. Mitra, "Low-Cost Gate-Oxide Early-Life Failure Detection in Robust Systems," in *Proc. IEEE Symp. on VLSI Circuits (VLSIC)*, Jun. 2010, pp. 125–126.

[9] S. Hellebrand, T. Indlekofer, M. Kampmann, M. A. Kochte, C. Liu, and H.-J. Wunderlich, "FAST-BIST: Faster-than-At-Speed BIST Targeting Hidden Delay Defects," in *Proc. IEEE Int'l Test Conf. (ITC)*, Oct. 2014, pp. 1–8, Paper 29.3.

[10] I. Polian, B. Becker, S. Hellebrand, H. Wunderlich, and P. Maxwell, "Towards Variation-Aware Test Methods," in *Proc. IEEE 16th European Test Symp. (ETS)*, May 2011, pp. 219–225.

[11] M. Sauer, A. Czutro, I. Polian, and B. Becker, "Small-Delay-Fault ATPG with Waveform Accuracy," in *Proc. IEEE/ACM Int'l Conf. on Computer-Aided Design (ICCAD)*, Nov. 2012, pp. 30–36.

[12] S. Eggersglüß and R. Drechsler, *High Quality Test Pattern Generation and Boolean Satisfiability*. Springer New York, 2012.

[13] X. Qian and A. D. Singh, "Distinguishing Resistive Small Delay Defects from Random Parameter Variations," in *Proc. IEEE 19th Asian Test Symp. (ATS)*, Dec. 2010, pp. 325–330.

[14] M. Tehranipoor, K. Peng, and K. Chakrabarty, *Test and Diagnosis for Small-Delay Defects*. Springer New York, 2011.

[15] S. M. Lin, C. J.and Reddy, "On Delay Fault Testing in Logic Circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 5, pp. 694–703, Sep. 1987.

[16] J. P. Hayes, "Digital Simulation with Multiple Logic Values," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 5, no. 2, pp. 274–283, Apr. 1986.

[17] V. S. Iyengar, B. K. Rosen, and J. A. Waicukauski, "On Computing the Sizes of Detected Delay Faults," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 3, pp. 299–312, Mar. 1990.

[18] S. Bose, H. Grimes, and V. D. Agrawal, "Delay Fault Simulation with Bounded Gate Delay Model," in *Proc. Int'l Test Conf. (ITC)*, Oct. 2007, pp. 1–10, Paper 26.3.

[19] E. S. Park, M. R. Mercer, and T. W. Williams, "Statistical Delay Fault Coverage and Defect Level for Delay Faults," in *Proc. Int'l Test Conf. (ITC)*, Sep. 1988, pp. 492–499, Paper 25.3.

[20] Y. Sato, S. Hamada, T. Maeda, A. Takatori, Y. Nozuyama, and S. Kajihara, "Invisible Delay Quality – SDQM Model Lights Up What Could Not Be Seen," in *Proc. IEEE Int'l Test Conf. (ITC)*, Nov. 2005, pp. 1–9, Paper 47.1.

[21] H. Konuk, "On Invalidation Mechanisms for Non-Robust Delay Tests," in *Proc. Int'l Test Conf. (ITC)*, Oct. 2000, pp. 393–399, Paper 14.3.

[22] I. Pomeranz and S. M. Reddy, "Hazard-Based Detection Conditions for Improved Transition Fault Coverage of Scan-Based Tests," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 2, pp. 333–337, Feb. 2010.

[23] C. Han and A. D. Singh, "Improving CMOS Open Defect Coverage Using Hazard Activated Tests," in *Proc. IEEE 32nd VLSI Test Symp. (VTS)*, Apr. 2014, pp. 1–6.

[24] A. Czutro, N. Houarche, P. Engelke, I. Polian, M. Comte, M. Renovell, and B. Becker, "A Simulator of Small-Delay Faults Caused by Resistive-Open Defects," in *Proc. 13th European Test Symp. (ETS)*, May 2008, pp. 113–118.

[25] A. Czutro, M. E. Imhof, J. Jiang, A. Mumtaz, M. Sauer, B. Becker, I. Polian, and H.-J. Wunderlich, "Variation-Aware Fault Grading," in *Proc. IEEE 21st Asian Test Symp. (ATS)*, Nov. 2012, pp. 344–349.

[26] M. Sauer, I. Polian, M. E. Imhof, A. Mumtaz, E. Schneider, A. Czutro, H.-J. Wunderlich, and B. Becker, "Variation-Aware Deterministic ATPG," in *Proc. IEEE 19th European Test Symp. (ETS)*, May 2014, pp. 1–6.

[27] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[28] D. Chatterjee, A. DeOrio, and V. Bertacco, "GCS: High-Performance Gate-Level Simulation with GP-GPUs," in *Proc. Design, Automation Test in Europe (DATE)*, Apr. 2009, pp. 1332–1337.

[29] ——, "Event-Driven Gate-Level Simulation with GP-GPUs," in *Proc. ACM/IEEE 46th Design Automation Conf. (DAC)*, Jul. 2009, pp. 557–562.

[30] K. Gulati and S. P. Khatri, "Towards Acceleration of Fault Simulation using Graphics Processing Units," in *Proc. ACM/IEEE 45th Design Automation Conf. (DAC)*, Jun. 2008, pp. 822–827, Paper 45.1.

[31] M. A. Kochte, M. Schaal, H. Wunderlich, and C. G. Zoellin, "Efficient Fault Simulation on Many-Core Processors," in *Proc. ACM/IEEE 47th Design Automation Conf. (DAC)*, Jun. 2010, pp. 380–385, Paper 23.4.

[32] M. Li and M. S. Hsiao, "FSimGP^2: An Efficient Fault Simulator with GPGPU," in *Proc. IEEE 19th Asian Test Symp. (ATS)*, Dec. 2010, pp. 15–20.

[33] H. Li, D. Xu, Y. Han, K. T. Cheng, and X. Li, "*nGFSIM* : A GPU-Based Fault Simulator for 1-to-n Detection and its Applications," in *Proc. IEEE Int'l Test Conf. (ITC)*, Nov. 2010, pp. 1–10, Paper 12.1.

[34] J. G. Tong, M. Boule, and Z. Zilic, "Efficient Data Encoding for Improving Fault Simulation Performance on GPUs," in *Proc. Int'l Symp. on Electronic System Design (ISED)*, Dec. 2013, pp. 138–142.

[35] S. Holst, M. E. Imhof, and H.-J. Wunderlich, "High-Throughput Logic Timing Simulation on GPGPUs," *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, vol. 20, no. 3, pp. 1–22, Jun. 2015.

[36] E. Schneider, S. Holst, M. A. Kochte, X. Wen, and H.-J. Wunderlich, "GPU-Accelerated Small Delay Fault Simulation," in *Proc. ACM/IEEE Conf. on Design, Automation Test in Europe (DATE)*, Mar. 2015, pp. 1174–1179.

[37] NVIDIA Corporation. (2016) High Performance Computing (HPC) and Supercomputing — NVIDIA Tesla — NVIDIA. [Online]. Available: http://www.nvidia.com/object/tesla-supercomputing-solutions.html [Accessed: Feb. 2, 2016]

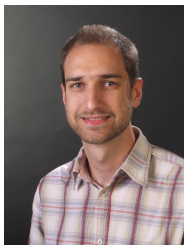
[38] K. Gulati and S. P. Khatri, "Accelerating Statistical Static Timing Analysis Using Graphics Processing Units," in *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC)*, Jan. 2009, pp. 260–265, Paper 3B–1.

[39] V. S. Iyengar and D. T. Tang, "On simulating faults in parallel," in *Proc. 18th Int'l Symp. on Fault-Tolerant Computing (FTCS)*, Jun. 1988, pp. 110–115.

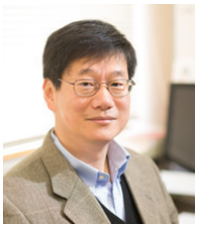
[40] IEEE Computer Society, "IEEE Standard for Standard Delay Format (SDF) for the Electronic Design Process," *IEEE Std 1497-2001*, Dec. 2001.
[41] R. M. Karp, "Reducibility Among Combinatorial Problems," in *Proc. Symp. on Complexity of Computer Computations*, Mar. 1972, pp. 85–103.
[42] P. S. Zuchowski, P. A. Habitz, J. D. Hayes, and J. H. Oppold, "Process and Environmental Variation Impacts on ASIC Timing," in *Proc. IEEE/ACM Int'l Conf. on Computer Aided Design (ICCAD)*, Nov. 2004, pp. 336–342.
[43] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Publishing Co., Inc., 1997.
[44] M. Sauer, A. Czutro, B. Becker, and I. Polian, "On the Quality of Test Vectors for Post-Silicon Characterization," in *Proc. IEEE 17th European Test Symp. (ETS)*, May 2012, pp. 158–163.
[45] A. Agarwal, D. Blaauw, and V. Zolotov, "Statistical Timing Analysis for Intra-Die Process Variations with Spatial Correlations," in *Proc. Int'l Conf. on Computer Aided Design (ICCAD)*, Nov. 2003, pp. 900–907.
[46] Nangate Inc. (2016) NanGate 45nm Open Cell Library. [Online]. Available: http://www.nangate.com/ [Accessed: Feb. 2, 2016]

Eric Schneider received the diploma degree in computer science (Dipl.-Inf.) from the University of Stuttgart, Germany, in 2012. There he joined the Institute of Computer Architecture and Computer Engineering (ITI), where he is currently working towards his Ph.D. His research interests include accelerated circuit simulation on Graphics Processing Units (GPUs), circuit test and diagnosis. He is a student member of the IEEE.

Michael A. Kochte received a diploma in computer science in 2007 and a Dr. rer. nat. (Ph.D.) from the University of Stuttgart, Germany, in 2014. He is currently with the Institute for Computer Architecture and Computer Engineering of the University of Stuttgart and leads the research group for Dependable Hardware. Dr. Kochte received a best dissertation award and two best paper awards (DELTA'08, JETTA'14). In addition to reconfigurable computing, his research interests include hardware reliability and hardware security.



Stefan Holst received his Ph.D. in the field of computer science from the University of Stuttgart, Germany, in 2012. He now serves as assistant professor in the Department of Creative Informatics at Kyushu Institute of Technology, Japan. His research interests include GPU-accelerated simulation and diagnosis of logic circuits, power-aware test and design-for-test. Furthermore, he is interested in rapid prototyping and device development in areas like rehabilitation and healthcare. He is a member of IEEE.



Xiaoqing Wen received the B.E. degree from Tsinghua University, China, in 1986, the M.E. degree from Hiroshima University, Japan, in 1990, and the Ph.D. degree from Osaka University, Japan, in 1993. From 1993 to 1997, he was an Assistant Professor at Akita University, Japan. He was a Visiting Researcher at University of Wisconsin, Madison, USA, from Oct. 1995 to Mar. 1996. He joined SynTest Technologies, Inc., USA, in 1998, and served as its Chief Technology Officer until 2003. In 2004, he joined Kyushu Institute of Technology, Japan, where he is currently a Professor and the Director of Dependable Integrated Systems Research Center. His research interests include VLSI test, diagnosis, and testable design. He co-authored and co-edited two books: VLSI Test Principles and Architectures: Design for Testability (Morgan Kaufmann, 2006) and Power-Aware Testing and Test Strategies for Low Power Devices (Springer, 2009). He holds 41 U.S. Patents and 14 Japan Patents on VLSI testing. He received the 2008 IEICE-ISS Best Paper Award for his pioneering work on X-filling-based low-capture-power test generation. He is a Fellow of the IEEE, a member of the IEICE, the IPSJ, and the REAJ. He is serving as associate editors for IEEE Transactions on Computer-Aided Design, IEEE Transactions on VLSI, and the Journal of Electronic Testing: Theory and Applications.

Hans-Joachim Wunderlich received the diploma degree in mathematics from the University of Freiburg, Germany, in 1981 and the Dr. rer. nat. (Ph.D. degree) from the University of Karlsruhe in 1986. Since 1991, he has been a full professor and since 2002 he has been the director of the Institute of Computer Architecture and Computer Engineering at the University of Stuttgart, Germany. He is editor of various international journals and program committee member of a variety of IEEE conferences on design and test of electronic systems. He has published 11 books and book chapters and more than 250 reviewed scientific papers in journals and conferences. His research interests include test, reliability, and fault tolerance of microelectronic systems. He is a fellow of the IEEE.