Abstract: Current nanometer CMOS circuits show an increasing sensitivity to deviations in first-order parameters and suffer from process variations during manufacturing. To properly assess and support test validation of digital designs, low-level fault simulation approaches are utilized to accurately capture the behavior of CMOS cells under parametric faults and process variations as early as possible throughout the design phase. However, low-level simulation approaches exhibit a high computational complexity, especially when variation has to be taken into account. In this work a high-throughput parallel fault simulation at switch level is presented. First-order electrical parameters are utilized to capture CMOS-specific functional and timing behavior of complex cells allowing to model faults with transistor granularity and without the need of logic abstraction. Furthermore, variation modeling in cells and transistor devices enables broad and efficient variation analyses of faults over many circuit instances for the first time. The simulation approach utilizes massive parallelization on Graphics Processing Units (GPUs) by exploiting parallelism from cells, stimuli, faults and circuit instances. Despite the lower abstraction levels of the approach, it processes designs with millions of gates and outperforms conventional fault simulation at logic level in terms of speed and accuracy.

SWIFT: Switch-Level Fault Simulation on GPUs

Eric Schneider, *Student Member, IEEE,* Hans-Joachim Wunderlich, *Fellow, IEEE*

***Abstract*—Current nanometer CMOS circuits show an increasing sensitivity to deviations in first-order parameters and suffer from process variations during manufacturing. To properly assess and support test validation of digital designs, low-level fault simulation approaches are utilized to accurately capture the behavior of CMOS cells under parametric faults and process variations as early as possible throughout the design phase. However, low-level simulation approaches exhibit a high computational complexity, especially when variation has to be taken into account.**

In this work a high-throughput parallel fault simulation at switch level is presented. First-order electrical parameters are utilized to capture CMOS-specific functional and timing behavior of complex cells allowing to model faults with transistor granularity and without the need of logic abstraction. Furthermore, variation modeling in cells and transistor devices enables broad and efficient variation analyses of faults over many circuit instances for the first time. The simulation approach utilizes massive parallelization on Graphics Processing Units (GPUs) by exploiting parallelism from cells, stimuli, faults and circuit instances. Despite the lower abstraction levels of the approach, it processes designs with millions of gates and outperforms conventional fault simulation at logic level in terms of speed and accuracy.

***Keywords*—parallel simulation; fault simulation; switch level; parametric faults; complex gates; variation analysis; GPU**

I. Introduction

THE simulation of faults is an important task of test validation flows for current nanometer CMOS designs [1]. Parametric deviations within cells cause faults at transistor level [2], [3], such as *resistive opens*, *bridges*, *cross-wire opens* or *shorts*, as well as *parasitic capacitances*. Due to the continuity of the fault parameters, they can exhibit varying timing and functional behavior based on the type of parameter and the amount of the deviation. Small deviations in resistive or capacitive parameters of a CMOS cell might cause delay faults that violate the timing along certain paths in the design. Larger parameter deviations are able to impact the functional behavior and cause, for example, *transistor stuck open faults* [4], [5]. Both, *small* and *large* faults are hard to detect and often not screened properly in testing, due to complex activation and propagation conditions [6]. On top of that, the detection is tampered by hazards [7], [8] or pessimistic timing assumptions [9], [10] and has become subject of recent test and diagnosis research [1], [11], [12].

For a proper validation full timing- and glitch-aware simulation approaches have to be applied in order to ensure proper activation and propagation of faults and signal transitions. Logic level simulation-based approaches typically rely on data from low-level characterization, but expose severe inaccuracies due to abstraction and modeling limitations by simplified timing assumptions and defect mechanisms. As soon as complex CMOS-cells are involved, more refined defect and simulation models at lower abstraction levels are necessary, since the behavior of many parametric and parasitic faults cannot be expressed at logic level at all. Furthermore, effects, such as *multiple input switching* (MIS) [9], [10], tamper with the circuit timing and can severely impact the fault detection of small delay faults by resistive opens. Thus, for accurate validation, it is crucial to apply as little abstraction as possible in order to avoid loss in information or modeling capabilities. Low-level effects that impact either functional and timing behavior in CMOS cells should be considered as many as possible. Clearly, by using analog simulations (i.e., SPICE [13]), more realistic results can be produced, but this comes along with an increase in simulation time by several orders of magnitude. Extensive analog simulations take hours or even several days to finish, even for small designs and few stimuli [14]. In *cell-aware test* [1] low-level characterization of cells is utilized to derive so-called *user defined fault models* which is based on information down to cell layout by small-scale analog simulation. Still, in order to avoid this simulation overhead, sufficiently accurate, but less expensive switch level simulation and fault modeling was applied [15], [12]. Yet, despite the lower runtime complexity, switch level simulation is still performed only in small scale.

With shrinking circuit structures and near-threshold operating conditions of devices, process variations and physical manufacturing defects exhibit increasing impact on the behavior of cells and gain more significance [16], [17]. Small deviations in first-order parameters in the layout of CMOS cells are sufficient to compromise the reliability of a system [18]. Several logic-level approaches have been proposed [19], [20], [21], [22], [23] that employ statistical timing analyses and Monte Carlo simulations with randomized gate delays to determine the impact of delay variation in a design on the test coverage as early as possible. However, the timing description of the simulation models is too abstract and the resulting errors by neglecting either transition ramps or MIS effects can end up quickly in the range of both variations or faults. Hence, in order to investigate variations more properly extensive and comprehensive low-level simulations are required.

The complexity of a holistic and accurate fault simulation of CMOS circuits further raises a big scalability problem that has been tackled by exploiting the inherent parallelism from circuit and fault [24], [25]. With the recent introduction of general purpose computing on *graphics processing units* (GPUs) high-throughput acceleration got cheaper and more effective by pro-

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

Manuscript received July 10, 2017; revised MONTH DAY, YEAR

cessing thousands to millions of lightweight threads on a single die [26]. Many GPU-based simulation approaches have been published to pursue faster and more efficient simulations [27], [28], [29], [30], [31], [32], [33], [34]. While at higher levels simulation runtimes have shown a significant improvement, the speedup of accelerated analog simulation is still limited and the scalability is not sufficient to process designs with millions of cells. A first GPU-accelerated switch level simulation and switch level fault simulation have been presented in [32], [35] that allowed waveform accurate timing simulation with transistor granularity based on first-order parameters. The GPU acceleration and arithmetic throughput allowed for significant speedup even over conventional logic level timing simulation, despite employing a more accurate simulation model.

This work presents "SWIFT" (**SWI**tch level **F**aul**T** simulator), the first high-throughput simulation approach for fast and scalable variation-aware switch level fault simulation on data-parallel GPUs. The core contributions are the following:

- Explicit modeling of functional and timing-related parametric and parasitic faults of CMOS cells at switch level.
- Transparent and overhead-free fault injection scheme in order to maintain high simulation performance.
- Pre-processing of structurally independent parametric faults for parallel injection and evaluation in order to drastically reduce the simulation overhead.
- Modeling of first-order parametric variation of cells with transistor granularity supporting both random as well as systematic variability in the design.

Besides the cell and waveform parallelism [32], the presented novel simulation approach is able to utilize up to four dimensions of parallelism simultaneously (*cells*, *waveforms*, *faults* and circuit *instances* under variation). Efficient processing as well as marginal memory overhead and negligible synchronization overhead enable to fully occupy computing resources of single or multiple GPU devices. The simulator achieves high throughput performance that outperforms conventional time simulation approaches at logic level in terms of simulation speed and accuracy. Furthermore, the presented approach enables:

- In-situ generation of arbitrary circuit instances under variation during simulation for efficient investigations of first-order parameter deviations.
- Accurate syndrome evaluation allowing for comprehensive analysis of faults under parametric variation applicable to designs with millions of cells.

The remainder of this paper is structured as follows: The following section briefly summarizes characteristics of GPU-architectures and GPU-accelerated simulation approaches. Section III gives an overview of our novel parallel switch level fault simulation approach. The basic switch level model is briefly explained in section IV. The novel extension to device level variation analysis is presented in section V. In section VI, the low-level parametric and parasitic switch level fault modeling is explained, followed by the calculation and evaluation of syndromes in section VII. Section VIII summarizes the concepts of the high-throughput parallelization for data-parallel architectures. Finally, comprehensive experimental results are presented in section IX.

II. Background

Graphics processing units (GPUs) and their programming paradigm enable the vast acceleration of applications by massive computing throughput [26], [36], which have established in high performance computing. Yet, the architecture also has certain restrictions which need to be taken care of. The limited global memory on the GPU (usually 4–12GB) is shared among all running threads and the available local registers in the multi processing cores pose a limit to the number of active threads on each multi processor. The threads invoked by the parallelized programs (called *kernels*) should therefore work on compact data sets with as little registers as possible. The execution of threads is organized in a *single instruction multiple data* (SIMD) fashion, which demands for simple control flows and control flow uniformity of the kernels. Global synchronization between threads is expensive, any thread divergence will result in serialization and synchronization overhead and lower the performance. Furthermore, host to device memory transfers are costly in terms of runtime and should be minimized or avoided at all. Ideally, only small data packages are transferred between host CPU and GPU which are expanded prior to and packed after the computations on the device by parallel threads to reduce the bandwidth of the communication.

A. Circuit Simulation on GPUs

Due to the inherent parallelism available in circuit simulation, several circuit [28], [27], [31] and fault simulation approaches [37], [30], [29] have been developed for GPUs. The acceleration is achieved through structural independence from cells and faults, where the circuit netlist is partitioned into areas, each of which is handled by individual threads or groups of threads. Data-parallelism through simultaneous evaluation of input stimuli is also used for acceleration [29], [30] by exploiting word-level parallelism when calculating logic operations which is a common practice used in *zero-delay* simulation [38], [25].

In contrast to *zero-delay* approaches, timing simulation computes numbers (continuous time values) rather than plain logic values only. Timing descriptions for logic simulation are usually provided in *Standard Delay Format* (SDF) that describes the *pin-to-pin* delays for *rising* and *falling* signal transitions for each cell in the netlist. The authors of [39] proposed an GPU-accelerated *statistical static timing analysis* (SSTA) which implements *parallel pseudo random number generators* (PPRNG) to accelerate Monte-Carlo simulation by generating random numbers in parallel. A first timing-accurate and glitch-aware simulator on GPUs has been presented in [40], [33] that computes full switching histories (*waveforms*) at signals with support for individual pin-to-pin delays as well as fine-grained *small delay fault simulation* [34]. By exploiting both structural and data parallelism during time simulation, and the high floating point throughput of the GPUs, speedups in the order of three magnitudes were achieved with a throughput of up to several hundred million gate-evaluations per second.

The high arithmetic throughput of GPUs also leads to the acceleration of low-level analog simulation (SPICE) [28], [31]. While in [28] the computationally expensive calculations were moved to the GPU, which showed a speedup of 10×, but regarding the runtimes it is still infeasible for multi-million cell designs. In [31] the complete SPICE simulation flow is performed on the GPU achieving two orders of magnitude speedup. However, due to the high working set footprint the simulation scales only for netlists composed of a few transistors only (up to 30) with quickly diminishing speedup.

B. *Fast Switch-Level Simulation on GPUs*

A recently presented approach [32], [35] implemented switch level simulation on GPUs that considers first-order electrical parameters, such as resistances and capacitances, for describing the functional and timing behavior of CMOS cells. In contrast to logic level simulation, the simulation at switch level utilizes simplified assumptions to model the electrical behavior of CMOS cells as shown in Fig. 1. Instead of modeling instantaneous transitions of discrete signal values, the timing is expressed as a function of continuous voltages over time based on the RC properties of the cells. The simulator [32], [35] can process complex CMOS-cells and covers important CMOS related timing effects, such as glitch filtering, transition ramps and multiple input switching [9], [10]. Despite the more complex simulation model, it fits well into the GPU environment and outperforms traditional logic level timing simulation approaches by several orders of magnitude.

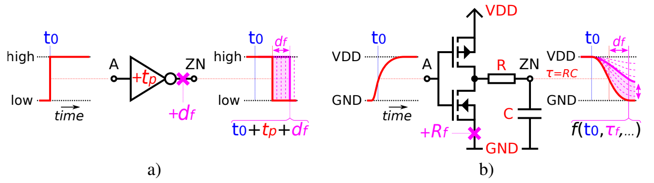


Fig. 1. Signal and timing abstraction in a) logic level and b) switch level simulation of a faulty (*slow-to-fall*) inverter cell.

III. Parallel Switch-Level Fault Simulation

This work utilizes the GPU-accelerated switch level simulator presented in [32], [35] and provides extensions to support high-throughput parametric and parasitic fault simulation with comprehensive syndrome analysis under systematic and random parameter variation. Fig. 2 illustrates the dimensions of parallelism that are simultaneously exploited by the implemented fault simulator during evaluation: a) *cell-parallelism*, b) *waveform-parallelism*, c) *fault-parallelism* and d) *instance-parallelism*. The structural parallelism from cells and data parallelism from stimuli have been adopted from [32]. These dimensions have been extended for cell-fault and instance parallelism which allows to simulate different instances of a circuit with varying parameters at the same time.

To achieve a high-throughput parallel simulation, the *naïve* serial simulation flow is mapped as shown in Fig. 3 to exploit the four dimensions of parallelism. Given a set I of instances of a circuit population, the naïve flow assigns the parameter specification of an instance to the netlist one after another. Input stimuli of the provided test set T are then assigned one

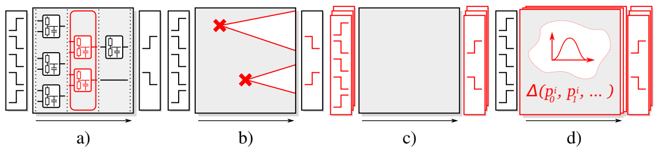


Fig. 2. Dimensions of parallelism exploited during simulation: a) cell-, b) fault-, c) waveform- and d) instance-parallelism.

by one, for each of the faults F to be investigated. In the worst case, the evaluation of a stimuli for a fault in a circuit instance involves the evaluation of all cells N in the circuit netlist. Hence, as indicated by the four nested loops, the total number of simulation problems can sum up to $|N| \times |T| \times |F| \times |I|$ cell evaluations.

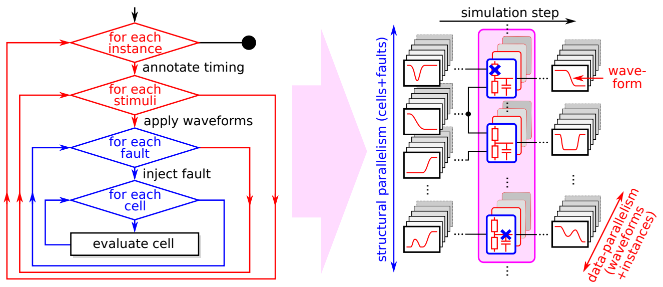


Fig. 3. *Serial simulation* flow mapped to the parallel evaluation scheme.

This work combines the structural problems (i.e., cells N and faults F) to utilize *structural parallelism* for accelerating switch level simulation of the circuit. All data-specific problems (i.e., stimuli T and instances I) are merged in addition in order to exploit data-parallelism during simulation, thus forming a multi-dimensional parallelization scheme.

The overall view of the presented fault simulation comprises two phases as shown in Fig. 4. During an initialization phase (Steps 1–3), the combinational netlist is extracted from the design and mapped to switch level primitives (1), so-called *Resistor-Resistor-Capacitor* (RRC-) cells [32], that consider first-order electrical parameters of CMOS cells. The required electrical parameters are extracted from *Detailed Standard Parasitics Format* (DSPF) files [41] obtained from layout synthesis. After mapping, the RRC-cells are partitioned into levels of topologically ordered cells (2). Then the provided fault set is collapsed in order to remove any equivalent faults and grouped into *fault groups* for parallel injection (3). In the simulation phase, the fault groups are processed one after another. First, the current fault group is injected into the circuit description (4). Then, the circuit instance parameters are assigned (5) which are used to modify the cell descriptions. In (6) a waveform-accurate switch level simulation is performed followed by a fault detection kernel (7) that captures the output responses of the circuit at given sample times. The shaded boxes denote parallel processes on the GPU.

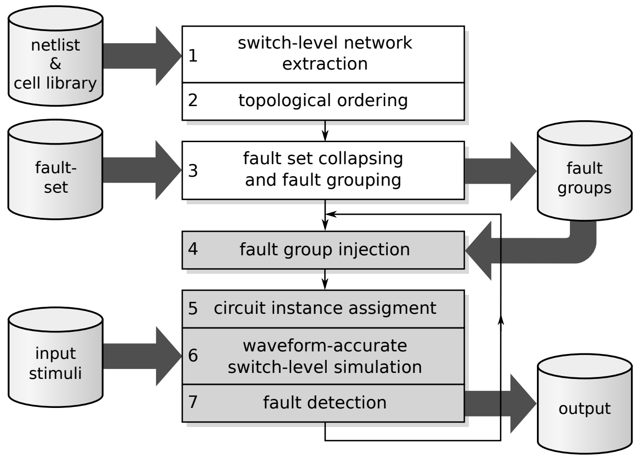

netlist & cell library
1 switch-level network extraction
2 topological ordering
fault-set
3 fault set collapsing and fault grouping
fault groups
4 fault group injection
input stimuli
5 circuit instance assigment
6 waveform-accurate switch-level simulation
7 fault detection
output

Fig. 4. Flow-chart of the variation-aware fault simulation algorithm.

IV. Simulation Model

In the following, the basic switch level circuit model of the *Resistor-Resistor-Capacitor* (RRC-)cell-based time simulation [32] and the signal representation will be briefly explained.

A. Circuit Model

For simulation of a CMOS circuit, the transistor netlist is partitioned into so-called *channel-connected components* [42], [12], which are sub-networks of PMOS and NMOS transistors that are connected via their drain and source terminals, such that current can flow freely in-between via their channels. Channel-connected components can be derived from most primitive cells (such as AND, NAND, ...) and complex CMOS-cells (AOI, XOR, ...) found in cell libraries. Fig. 5 shows an example of a highlighted channel-connected mesh extracted from the transistor netlist of a complex 10-transistor XOR-cell with some output load. The extracted mesh is controlled by pull-up and pull-down networks that drive an intermediate signal domain (Y), which is input to transistors of a next channel-connected component. Ideally, current flows only within the mesh and is not allowed to pass over transistor gate terminals to other meshes. Thus, each channel-connected component draws current independently from its associated power supply.

The signal voltage of the channel is controlled via the pull-up and pull-down networks of the mesh. Depending on the input voltages at the transistor gate terminals, pull-up and pull-down meshes behave like a voltage divider, that drive an output capacitance C_{load} via some wire resistance R_w.

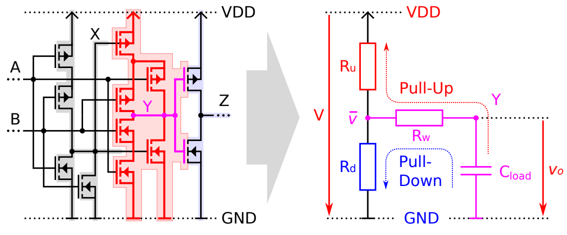


Fig. 5. Extraction of *channel-connected components* in a transistor netlist.

Resistor-Resistor-Capacitor (RRC-) cells provide a simple unidirectional model for representing the switching behavior within channel-connected components [32]. An RRC-cell models a channel-connected component, that is driven by PMOS and NMOS transistor networks as shown in Fig. 6.

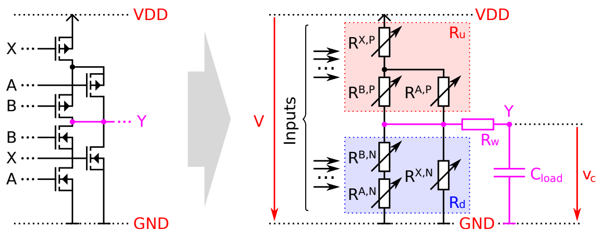


Fig. 6. Switch-level abstraction of cells modeling an input-controlled voltage-divider (R_u, R_d) driving an output load C_{load} as RRC-cell.

The transistors are described each by a 3-tuple $D = (V_{th}, \{R_{off}, R_{on}\})$ as part of the *RRC-cell description* $\mathcal{R}$, with V_{th} as threshold voltage and R_{off} (R_{on}) as *blocking* (*conducting*) drain-source resistance. Both resistances and thresholds of the transistors are obtained from characterization of the SPICE model cards, which needs to be done only once for each transistor type during a pre-processing. Each device is viewed as a voltage-controlled resistor $R_D(v)$, that models a threshold-based binary switch based on the applied gate voltage v:

$$R_D(v) = \begin{cases} R_{off} & \text{if } v < V_{th}, \\ R_{on} & \text{else.} \end{cases} \tag{1}$$

The resistances from pull-up net R_u and pull-down net R_d form a voltage divider (R_u, R_d) driving a lumped output capacitance C_{load} via an output resistance R_w. Upon transistor switches, the resistances R_u and R_d change as a consequence. After a change at time t_i, the output $v_c(t)$ will follow an exponential curve for $t \geq t_i$ with time constant $\tau = S \cdot R_u C_{load}$, due to the RC-property of the cell. Starting from $v_c(t_i)$ the curve aims for a *stationary voltage* $\overline{v} = S \cdot V + \text{GND}$, with $S = \frac{R_d}{R_u+R_d}$ as divider ratio and $V = \text{VDD} - \text{GND}$. Since RRC-cells derived from most standard-cell libraries show regular pull-up and pull-down nets, simple nodal analyses using Kirchhoff's laws are applied to obtain R_u and R_d [32].

B. Signal Representation

The curve segment of a transient response at time t_i is expressed as exponential function for $t \geq t_i$ as follows:

$$v(t) := (v(t_i) - \overline{v}_i) \cdot e^{-\frac{\Delta t}{\tau_i}} + \overline{v}_i \tag{2}$$

where $v(t_i)$ is the signal value when the transition is initiated and $\Delta t = (t - t_i)$. Between two consecutive transistor switches at t_i and t_{i+1}, all resistances in the RRC-cell remain constant. As shown in Fig. 7, the curve in each switch interval $[t_i, t_{i+1}]$ can be described entirely by a tuple of three parameters $p_i = (t_i, \overline{v}_i, \tau_i)$, referred to as *pivot* [32], which are:

- t_i: The *time* of the switch that initiates the curve.
- $\overline{v}_i$: The *stationary voltage* targeted by the curve.
- τ_i: The *slope* of the curve given by the time constant.

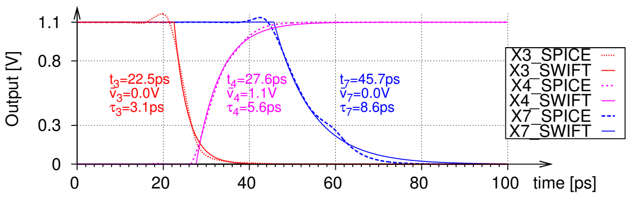


Fig. 7. Signals of cells with varying fanout in SPICE transient analysis (dotted) and approximation using the switch level pivot representation (bold).

The complete switching history or *waveform* w of a signal is described by a list of pivots $w = \{p_0, ..., p_k\}$. The pivots are ordered temporally from earliest to latest and allow to model arbitrary waveforms as shown in Fig. 8 using piecewise approximation. For the evaluation of a waveform $w(t)$ at time t Eq. (2) is applied iteratively along the pivot times t_i and eventually for t itself to obtain $w(t) = v(t)$. Hence, detailed signal information is sustained with very small memory overhead, allowing for an efficient time- and value-continuous evaluation without the need of sampling [32].

V. Variation Modeling

Parametric variation among circuit instances affects both functional and timing behavior of cells. Its source is distinguished as either *random* or *systematic* nature. *Random variations* have quantum mechanical origin and involve uncertainties which are typically modeled by *independent random variables* [16]. *Systematic variation* on the other hand takes into account spatial and parametric dependencies within dies, wafers or lots that affect the underlying cells simultaneously [43], [44]. Sources relate to material properties and limitations of fabrication processes (e.g., *lithography*, *polishing*) that sustain correlations between neighboring structures.

At logic level, variations are typically modeled by modifying or randomizing the delay of cells for each instance of a circuit population [39], [22], [34]. In this work, the variation modeling of [34] is mapped to switch level for application with transistor granularity. The modeling supports both independent random as well as correlated systematic variation and provides:

- *efficient generation* of arbitrary individual circuit instances during simulation, and
- modeling of *variability faults* that involve parametric deviations due to higher uncertainty.

A. Random Parameter Variation

Random variation is typically modeled using random variables RV from uniform or normal distributions [39], [22] which can be described by *mean* and *variance*. Given the nominal specification of a parameter $\pi_{nom} \in \mathcal{R}$, the expected value of the distributions is assumed to be $E(RV) = \pi_{nom}$. Thus, cell descriptions need to be extended only by a *variance* parameter. For the number generation during simulation, a *pseudo random number generator* (PRNG) is used, which uses initialization with a unique *seed* for each instance of a circuit population [34]. When a cell is simulated, additional entropy

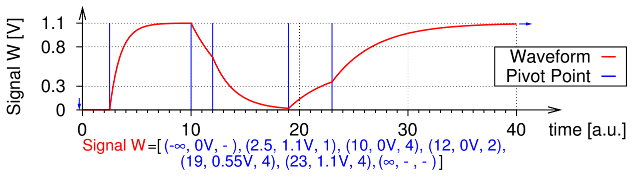


Fig. 8. Pivot representation and visualization of an arbitrary signal waveform.

from cell identifier, cell type and other parameters is added to obtain a unique seed within its respective instance.

The randomization is described by the function $\theta : P \mapsto \mathbb{R}$ which takes as input a vector P that represents a point in the parameter space that identifies a specific circuit instance of the population. The vector P is composed of the standard deviation σ and a set of parameters $\{p_0, p_1, \cdots, p_u\}$ containing the initial seed and additional entries for entropy. For each nominal cell parameter $\pi_{nom} \in \mathcal{R}$ the simulation procedure applies the variation once upon loading the cell description:

$$\pi_{res} := \pi_{nom} \cdot (1 + \theta(\sigma, p_0, p_1, \ldots, p_u)) \, . \tag{3}$$

Thus, compared to the cell evaluation itself, only little computational overhead is introduced while computing the variation during simulation. Furthermore, new instances of the circuit population can be generated on the fly without the need for storing and transferring descriptions of every instance.

B. Systematic Parameter Variation

For *systematic variation*, the previous modeling concept is generalized to consider parametric correlations between the input variables. The parameters and their variation impact are described on multiple levels using a hierarchical parameter space model [16], [43]. On each level of the hierarchy, a real function $\theta_v : P_v \mapsto \mathbb{R}$ is specified to calculate the parametric deviation with respect to a subset of input parameters P_v. A function can take a vector P_v of entries p_i as input that refer to coordinates of the cell at die-, wafer- or lot-level, or manufacturing-related and environmental parameters (i.e., ambient temperature), and map to an absolute parametric deviation in an RRC-cell. The individual systematic components θ_v are implemented as series of either polynomials or trigonometric functions over the parameters to approximate between process corners. To obtain the deviation of a specific (cell) instance parameter, the components of the hierarchy are added and weighted based on their individual impact α_v:

$$\pi_{res} := \pi_{nom} \cdot \left(1 + \sum_{v=0}^{n} (\alpha_v \cdot \theta_v(p_0, p_1, \ldots, p_{u_v}))\right) . \tag{4}$$

This way, the spatial and parametric correlations within and across levels are sustained for the population.

The evaluation of these functions can be efficiently processed on GPU devices due to their high floating point throughput. In case the θ_v are described by polynomials, Horner's method can be applied to reduce the number of arithmetic operations. Also, the use of architecture-specific instructions, such as *fused multiply-add* (FMA), is enforced which provides an increase in computation speed on current device generations [36].

VI. Fault Modeling

Low-level parametric faults in the circuit are typically modeled by utilizing first-order electrical parameters of CMOS cells. In *cell-aware test*, the identification of potential faults usually requires manual work and additional effort to obtain *user-defined fault models* [1]. Ways for layout-aware extraction of defects and their abstraction to relevant fault types in either electrical or logic domain have been already proposed in [45], [46]. This work utilizes the RRC-cell parameters $\mathcal{R}$ for fault modeling and injection, allowing to model a broad variety of cell-internal low-level parametric faults without the need of higher level abstraction [35]. In the following, the general fault modeling and the injection methods will be explained.

A. Parametric Faults

Each RRC-cell description $\mathcal{R}$ is organized as a set tuples containing device descriptions $D = (V_{th}, \{R_{off}, R_{on}\})$ with blocking and conducting resistances as well as threshold voltages of each PMOS and NMOS transistor D, along with the output load C_{load} and the voltage levels for VDD and GND. Additional static resistances can be considered to model cell-internal wire and via resistances in the cell descriptions [35].

1) Resistive Shorts and Opens: A *resistive fault* f of a cell is described by a tuple $f = (loc, \Delta R_f)$, which is composed of a location parameter loc as well as a *fault size* ΔR_f. The location loc refers to a resistive parameter $R^{loc} \in \mathcal{R}$ of the cell which is either the blocking or conducting resistance of a transistor D or a static resistance. The fault size describes the actual deviation of the selected parameter in Ohms, which is modified accordingly by $\widetilde{R}^{loc} := R^{loc} + \Delta R_f$. Depending on the sign of ΔR_f, either resistive opens ($\Delta R_f > 0$) or resistive shorts are modeled ($\Delta R_f < 0$). Transistor open faults can be obtained by increasing the conducting resistance R_{on}, while for shorted transistors the blocking resistance R_{off} is lowered. The static resistances in the RRC-cell model offer to include parametric faults related to the vias and wires in a cell, such as cross-wire opens and bridges [6].

2) Capacitive Faults: For modeling resistive and capacitive faults in interconnects, a lumped model is assumed [35]. The capacitive faults are described as tuple $f = (loc, \Delta C_f)$, introducing an additional capacitance ΔC_f to the RRC-cell output capacitor C_{load}. The resistive property of the interconnection fault is added to the wire component R_w of the driving cell.

3) Voltage-related Faults: The RRC-cell voltage parameters, such as device threshold, VDD and GND, offer opportunities for modeling power-related issues and aging phenomena, such as *Negative-Bias Temperature Instability* (NBTI) or *Hot-Carrier Injection* (HCI). NBTI and HCI cause an increase in the threshold voltage of devices over time, which delays the transistor switching process [47], [48]. In the RRC-cell descriptions the threshold voltage parameter V_{th} of a transistor allows to inject aging faults. A particular shift ΔV can be modeled by lowering (raising) the threshold of the targeted PMOS (NMOS) transistor respectively $\widetilde{V}_{th} := V_{th} \mp \Delta V$ [35]. Similarly, the VDD and GND voltage of a cell can be altered to reflect power-related issues, such as fluctuations in the power grid due to IR-drop and ground bounce [49].

B. Variability Faults

Severe shifts in standard deviation and other variability parameters can cause outliers due to *unstable* components that expose different fault behavior throughout a circuit population. These unstable components can cause yield problems, and need to be identified early. However, worst case analyses are too pessimistic to provide meaningful information.

In order to tackle this issue, the fault modeling is extended by *variability faults*. For random variation, faults are injected by modifying the standard deviation parameter σ of the cell description $f = (loc, \Delta\sigma_f)$ by some size $\Delta\sigma_f$. By increasing the standard deviation $\widetilde{\sigma} := \sigma + \Delta\sigma_f$, unstable properties can be triggered within a cell to exhibit different impact over a circuit instance and the entire circuit population. As for systematic variation, the injection of an *offset vector* ΔP with $f = (loc, \Delta P)$ allows to *move* cells within the parameter space. Thus, by modification of the cell parameter vector $\widetilde{P} := P + \Delta P$, changes in multiple dimensions can be performed simultaneously that maintain all spatial or parametric correlations within the parameter space.

C. Fault Collapsing

For a particular fault size, the number of fault locations can be reduced by structural collapsing of the fault list. Compared to transition faults [38], collapsing becomes more restrictive if actual timing has to be taken into account [34]. For this, *classes* of *timing equivalent faults* need to be identified. Timing equivalent faults show identical waveform behavior in the switch level model. Thus, simulation of only one *representative* is necessary to evaluate all faults of a class. At switch level the scope of fault collapsing is even more restricted than at logic level, since not only the switching times, but also the shape of waveforms needs to be considered. For faults in different cells, both affect the switching of their successors differently.

For fault locations within a RRC-cell, this work applies a simple rule to collapse resistive opens and shorts. Let $\pi_0, \pi_1, ..., \pi_n \in \mathcal{R}$ be resistances or devices in either pull-up or pull-down net of a cell. Two fault locations π_i and π_j in a net N are *timing equivalent* iff there is a path $p \in N$ from VDD or GND to the cell output with $\pi_i, \pi_j \in p$, such that $\forall p' \in N : p' \neq p$ either $\pi_i, \pi_j \in p'$ or $\pi_i, \pi_j \notin p'$. In the example of Fig. 6, resistors associated with the NMOS devices of signals A and B are considered equivalent, since they form a series $(R^{A,N}, R^{B,N})$ where current needs to flow through both resistors. Thus, for any resistive fault f of size ΔR_f the injection in $R^{A,N}$ or $R^{B,N}$ delivers identical results: $(R^{A,N} + \Delta R_f) + R^{B,N} = R^{A,N} + (R^{B,N} + \Delta R_f)$.

D. Fault Injection

Any RRC-fault is injected into the circuit prior to its actual simulation by manipulating the associated cell descriptions, and marking the fault sites as *faulty*. During simulation, the presence of faults is transparent to the kernels. Multiple faults can be injected simultaneously to model multi-faults across the circuit or within RRC-cells, as well as for exploiting *fault parallelism* (cf. Sec. VIII-B). After the simulation has been completed, the descriptions of all fault sites marked as *faulty* are restored to their original specification and the simulator is ready for a new fault simulation run. Since the fault descriptions are compact, only a few small memory transactions are necessary during the injection process.

VII. Syndrome Evaluation

After the simulation of the circuit, the waveforms of all output pins in the output cone of a fault site are captured at a user-specified signal sample time point. Given the sample time t_{samp}, each output waveform is traced pivot by pivot until the latest curve segment $p_i = (t_i, \overline{v}_i, \tau_i)$ with $t_{i+1} > t_{samp}$ is reached. The signal voltage is computed iteratively along the pivot boundaries according to Eq. (2). Eventually, the final signal value at $t_{samp} \in [t_i, t_{i+1})$ in the last segment p_i is computed for $\Delta t = (t_{samp} - t_i)$ time units respectively [35].

A. Signal Interpretation

Eventually, the obtained voltage values are compared against the reference values of the fault-free simulation to determine *right* from *wrong*. The logical interpretation of the continuous signals is done by applying a threshold-based characterization of the voltage level. The obtained voltage values are interpreted as either *high* (1), *low* (0) or *unknown* (X). For this, a *threshold interval* $(V_{thL}, V_{thH}) \in [\text{GND}, \text{VDD}]$ is defined, which is bounded by a *low* threshold V_{thL} and *high* threshold V_{thH}. Given an arbitrary voltage value $v \in \mathbb{R}$ sampled from a signal waveform, the mapping to the logic symbols $\{0, 1, X\}$ is described as follows [35]:

$$val : \mathbb{R} \to \{0,1,X\}, val(v) := \begin{cases} 0 & \text{if } v \leq V_{thL}, \\ 1 & \text{elif } v \geq V_{thH}, \\ X & \text{else.} \end{cases} \quad (5)$$

Signal values v within $[\text{GND}, V_{thL}]$ ($[V_{thH}, \text{VDD}]$) are considered as *low* (*high*), since the voltage levels are likely to be amplified in CMOS technology. Values in (V_{thL}, V_{thH}) are considered pessimistically as *undefined* and *possibly* erroneous, since succeeding cells might interpret voltages differently.

B. Discrete Syndrome Computation

To determine the presence of a *faulty* value at a given output and time t, the output waveform $w(t)$ is compared against the *fault-free* value $w(\infty)$ for $t \to \infty$. It is assumed that the *fault-free* responses of a circuit are *stable* and have clear *high* or *low* signals. A *syndrome waveform* $syn(t)$ maps the voltage differences of the output waveform $w(t)$ and the *fault-free* values according to Eq. (5) to discrete logic symbols [35]:

$$syn(t) := \begin{cases} val(v(t)) & \text{if } v(\infty) \leq (\frac{\text{VDD}+\text{GND}}{2}), \\ val(\text{VDD} - v(t) + \text{GND}) & \text{else} \end{cases}$$

allowing to distinguish the three discrete logic cases (0, 1, X). Therefore, $syn(t) = 1$ ($syn(t) = 0$) iff the cell produces a *faulty* (*fault-free*) signal at time t. In case $w(t)$ is *undefined* the syndrome is *unknown* and the output is pessimistically considered as *possibly erroneous* (X).

C. Setup-Hold Time Violations

To consider violations in setup and hold times of storage elements, margins for setup t_{setup} and a hold times t_{hold} are utilized to check the stability of an output signal in some interval $[t_S, t_H]$ with $t_S := t_{samp} - t_{setup}$ and $t_H := t_{samp} + t_{hold}$. Any violations can be obtained by traversing the syndrome waveform. First, the value $syn(t_S)$ is captured for reference. Since the curve segments in w are monotonously increasing, further comparisons of the signal values only need to be done at pivot boundaries t_i until time t_{samp} is reached. A setup-violation S is raised iff

$$S \Leftrightarrow (\exists t \in [t_S, t_{samp}] : syn(t) \neq syn(t_{samp})) \,.$$

Similarly, for hold-violations the value at t_{samp} is used as reference and the traversal is continued until t_H. A hold violation H is issued iff

$$H \Leftrightarrow (\exists t \in [t_{samp}, t_H] : syn(t) \neq syn(t_{samp})) \,.$$

This way, the syndrome capturing, as well as the checks for setup and hold time violations can be performed in the same process during output evaluation.

Regarding the impact on the fault detection, it is assumed that any setup or hold time violation at an output causes additional uncertainty in the captured signal. For simplification, the captured output will be considered pessimistically as *unknown* and thus *possibly erroneous*, if either $S = 1$ or $H = 1$ holds.

D. Fault Detection

The detection of a fault is classified into *detected*, *undetected* and *possibly detected*. The classification is performed by looking up the syndromes $syn_o(t)$ of all the outputs o in the corresponding output-cone O of the fault. Given the captured syndromes of the outputs, a fault is:

- *detected* (DT) iff *any* output signal in the output-cone shows a *faulty* syndrome ($\exists o \in O : syn_o(t_{samp}) = 1$),
- *undetected* (UD) iff *all* outputs in the output-cone show a *fault-free* syndrome ($\forall o \in O : syn_o(t_{samp}) = 0$),
- *possibly detected* (PD) iff a *non-empty subset* of outputs in the output-cone exhibits an *unknown* syndrome ($\exists o \in O : (syn_o(t_{samp}) = X) \vee (S_o \vee H_o)$), while the others do not show a *faulty* syndrome ($\forall o \in O : syn_o(t_{samp}) \neq 1$).

After a simulation pass, the output waveforms remain untouched during the evaluation and stay present in the memory. They can be sampled quickly in succession at further points in time. Individual capture times can be considered for each output, thus allowing to model skew in the clock distribution. Also, since the output responses are available for all stimuli, space- as well as time-compaction can be applied.

VIII. Parallelization

The whole simulation of the circuit is implemented as a sequence of smaller tasks, each of which handles a different aspect during evaluation (e.g., stimuli conversion, cell evaluation, fault detection, data extraction, etc.). Each task is executed by a multi-dimensional simulation kernel that invokes many threads arranged as an array or *grid* on the GPU device. A single thread within the grid performs its respective task for a distinct cell (and fault) under a particular stimuli and circuit instance parameters. In the following, the implementation of each dimension will be briefly explained.

A. Cell-Parallelism

The parallel simulation of RRC-cells is based on the parallel processing of mutually data-independent nodes [32], [33]. If two cells are neither in their input- nor output-cones of each other, the order of their evaluation can thus be chosen freely and as well be scheduled for parallel execution, as opposed to data-dependent cells, where the output of one cell needs to be computed first to be provided as input for another.

Groups of mutually data-dependent and independent cells are obtained after topological ordering of the netlist in the levelization pre-processing. During levelization, the cells are partially ordered and partitioned into so called *levels* based on their topological distance (i.e., from primary and pseudo-primary inputs). The levels have to be simulated sequentially from inputs to outputs in order to satisfy the data dependencies of all the cells. The evaluation is performed by invoking the simulation kernel for each level. Cells within each level are mutually data-independent and the parallelization of their evaluation is arranged by starting threads for each of the cells upon invoking the simulation kernel of the level. All threads then simultaneously process the previously computed input signals for their cells.

In order to keep the control flow of the simulation kernels simple and uniform, the output waveforms of the cells have to be stored in fixed positions in the memory. Since the number of resulting signal switches in a waveform is not known a priori, an *overflow* detection and memory *calibration* mechanism is applied [33]. In case the assigned memory of a signal waveform is not sufficient to store all transitions, an overflow is reported. The simulation of the level is then repeated with increased storage limitation of the culprit waveforms and adjusted memory allocation until all overflows are resolved.

B. Fault-Parallelism

The parallel simulation of faults is organized by injecting sets of output-independent faults into a single simulation instance. The RRC-cell faults originate at the cell outputs and eventually propagate along succeeding cells towards the outputs of the circuit. Therefore, their impact is limited to the output cone. For a parallel simulation of RRC-cell faults it must be ensured that injected faults do not interfere by adding or masking fault effects during propagation. For this, fault sets are partitioned into *fault groups* of mutually output-independent faults [24] for simultaneous injection by ensuring mutual output-independence of their reachable outputs. The underlying graph coloring problem to solve this problem *optimally* is NP-hard [50]. Instead, the heuristic of [35], [34] is used which is especially suitable for exhaustive fault sets. The heuristic processes faults in reverse topological order from circuit outputs towards inputs in a broad fashion. Each fault is assigned an initial fault group index based on previously processed topological successors. Upon identifying an output-independent fault group the index is propagated to its predecessor nodes. This way, grouping attempts and comparisons can be reduced and suitable fault groups are found quickly. Once all fault groups have been determined, the simulator processes them in consecutive simulation runs one after another.

C. Stimuli-Parallelism

So far only structural aspects have been tackled by the parallelization of cells and faults. In addition data-centric aspects will be included as well in order to increase the simulation throughput, by processing circuits not only for single stimuli, but multiple stimuli at the same time. The organization of the simulation kernels is therefore extended as shown in Fig. 9, resulting in a two-dimensional array or *grid* of threads [32], [33]. Each thread in the vertical dimension simulates a different cell for a given stimuli. The set of threads in this dimension will be referred to as a *slot*. All threads in the horizontal dimension, on the other hand, simulate the same cell of the circuit, but each with different stimuli.

The threads of the kernel grid are scheduled in fixed batches for simultaneous execution on the GPU multi-processors by the thread scheduler. The threads of all batches all handle the same cell, but for different stimuli. Hence, the threads compute the same function, but for different data, which is compliant with the SIMD execution scheme of the GPU processors. In addition, the waveforms are stored in the global memory in a way, such that the accesses of the threads in a batch are coalesced to connected address ranges. This way, the utilization of each memory transaction is maximized and the overall amount of transactions by the thread blocks is reduced.

The two-dimensional scheme is applied throughout the different simulation kernels, from test vector assignment, through cell evaluation to fault detection. The amount of stimuli that can be processed in parallel depends on the amount of global memory on the GPU as well as the amount of memory required for processing a single simulation instance. If more stimuli are provided than able to fit on the GPU memory, the stimuli set is split into chunks that are processed either serially on a single GPU, or in *parallel* on multiple GPU devices on the host system. Thus, larger memories and more GPUs allow for a higher degree of parallelism. In contrast to the structural parallelism from cells, the effective parallelism from data remains constant throughout the simulations.

D. Instance-Parallelism

For parallel simulation of instances under variation, the concept of *slots* is generalized for utilization with instances similar to [34]. The extension is illustrated in Fig. 10 and shows

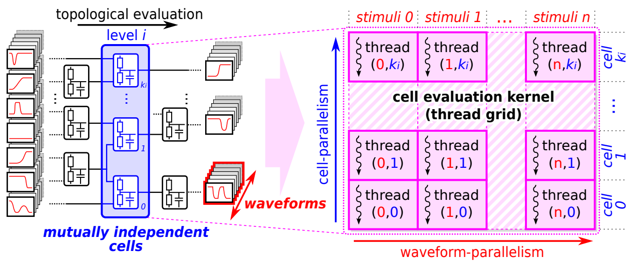


Fig. 9. Two-dimensional parallel evaluation of multiple data-independent cells and input stimuli in a topologically ordered netlist.

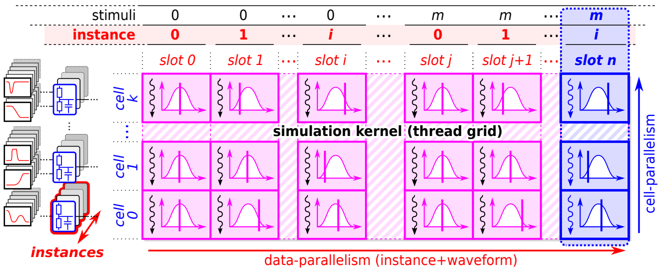

stimuli 0 0 0 m m m
instance 0 1 i 0 1 i
slot 0 slot 1 slot i slot j slot j+1 slot n
cell k
cell 1
cell 0
simulation kernel (thread grid)
instances
cell-parallelism
data-parallelism (instance+waveform)

Fig. 10. Thread-grid organization for the evaluation of multiple circuit instances in parallel. Each *slot* covers one instance for one stimuli [34].

the basic multi-dimensional thread-organization of the GPU kernels. Each thread of a row computes the same cell, while within each column, the threads now compute a given stimuli for a specific instance, whose arrangement can be arbitrarily chosen. The thread organization maintains the control flow uniformity of the underlying kernels, and also allows to fully occupy the GPU memory resources.

The calculation of the instance specific parameters is scheduled at the beginning of each cell evaluation. The threads access the instance information of their respective slot and manipulate the cell parameters upon loading the cell description. An efficient implementation of a *parallel pseudo random number generator* (PPRNG) is utilized in order to generate the random numbers on the GPU [51], [39]. For incorporating systematic variation, the parameter space spanned by given process corners is implemented as real functions. For both, random and systematic variation, the calculation of the instance-specific parameters only needs to be done once at the beginning of the evaluation of a cell and therefore causes negligible overhead [34]. Again, the parallelization scheme allows to be distributed among multiple GPU devices to avoid serialization due to insufficient device memory.

IX. Experimental Results

All experiments (except where otherwise mentioned) were conducted on a host system connected to a NVIDIA® GeForce™ GTX® 1080 Ti device with 3584 cores clocked at 1.6MHz and access to 11GB of global device memory. The host system was equipped with two Intel® Xeon® E5-2687W v2 processors clocked at 3.4GHz and 256GB of RAM. As for the circuit description of our largest circuit p3881k, the maximum memory occupied on the GPU for storing the entire circuit with 3.7 million nodes was roughly 280MB, which is less than 2.5% of the available global device memory.

The experimental section is split into four parts. The first part is concerned about the general circuit statistics and the fault grouping. In the second part, the runtime performance of the simulation is investigated. The third part focuses on the fault simulation experiments and the evaluation. In the last part, parameter variation is investigated.

A. Overview

Table I summarizes the statistics of the circuits and their fault grouping results. As sample circuits, synthetic benchmarks from ISCAS'89, ITC'99 and industrial designs provided by NXP have been synthesized using a 45nm standard cell library. For the provided fault set to be grouped a high resistive open was considered at each transistor in the circuit in addition to a capacitive fault at each cell output. The node counts (inputs, outputs and cells) of the respective circuits range from 18 thousand to over 3 million (Col. 2). The logic depth of each circuit is shown in column 3. Column 4 contains the number of pattern pairs that can be simulated concurrently in a single pass on the GPU device when fully utilizing the global device memory. Columns 5–9 provide the results from the fault grouping given the initial number of faults (Col. 5). The remaining faults after collapsing cell-internal faults is shown in column 6, followed by the number of obtained fault groups (Col. 7), the *grouping efficiency* (*eff.*, Col. 8) and the runtime required for the grouping process (Col. 9). The grouping efficiency is defined as the fraction of initial faults divided by the number of obtained fault groups and hence represents the average number of faults simulated per simulation run. The time for the fault grouping can be considered as negligible, as the grouping provides a substantial reduction in the simulation effort compared to the naïve serial simulation. For example, even in case of p469k with a low group efficiency of 1.3, the achieved saving in simulation runs is about 25 percent.

Fig. 11 shows the distribution of the fault group sizes as well as the cumulative fault count over all fault groups obtained from the grouping in Table I. As shown, the leftmost groups contain the most faults which are located directly or close to the outputs of the circuit and thus have a high probability of mutual output independence (e.g., each output itself is output-independent of others). Roughly 90 percent of the total faults in each circuit are processed in average after simulation of the first 25 percent of the fault groups. As the grouping progresses the groups get smaller at varying speed depending on the circuit. The fault groups sustain a size of at least ten faults or larger until 50 percent have been processed. After that, the group sizes drop for most of the circuits to a few faults only. These *late* faults usually lie on the structurally longest paths and therefore have a high probability of mutually sharing common outputs which prohibits from being grouped.

B. Runtime Performance

For the performance evaluation of SWIFT, the runtimes are compared against time simulation at logic level in Table II. A commercial ATPG tool was used to generate n-detect

TABLE I. BASIC CIRCUIT AND FAULT GROUPING STATISTICS.

Circuit(1)	Nodes(2)	Depth(3)	Parallel Pattern-Pairs(4)	Fault Grouping				
				Faults		Groups(7)	*eff.*(8)	Time(9)
				init.(5)	*coll.*(6)			
s38417	19.0k	48	12.2k	68.7k	59.5k	2084	33.0	940ms
s38584	23.1k	70	10.2k	88.0k	75.4k	2610	33.7	1.35s
b17	42.8k	120	13.7k	178.3k	150.3k	18.5k	9.7	6.09s
b18	125.3k	195	4224	529.3k	445.9k	36.9k	14.3	38.22s
b19	250.2k	203	2208	1.06M	891.1k	39.6k	26.7	43.98s
b22	27.8k	88	14.4k	118.6k	99.5k	11.8k	10.1	3.61s
p77k	70.5k	466	5568	287.5k	242.7k	72.3k	4.0	2:02m
p141k	178.1k	79	1856	695.0k	592.8k	55.8k	12.4	3:15m
p267k	218.4k	55	1376	846.2k	716.8k	14.2k	59.8	31.50s
p330k	286.9k	61	1280	1.15M	969.2k	85.4k	13.4	5:10m
p418k	440.3k	174	768	1.68M	1.44M	22.7k	74.3	36.54s
p469k	104.4k	239	5664	451.1k	381.1k	337.2k	1.3	42.17s
p500k	527.0k	179	736	2.05M	1.76M	24.3k	84.6	44.09s
p533k	676.6k	112	512	2.77M	2.33M	10.7k	258.2	45.73s
p951k	1.09M	153	224	3.92M	3.40M	20.2k	194.1	1:12m
p1522k	1.09M	508	320	4.32M	3.65M	77.8k	55.5	24:47m
p2927k	1.67M	388	160	6.49M	5.56M	37.0k	175.2	5:10m
p3188k	2.85M	618	128	11.48M	9.72M	524.5k	21.9	6:05h
p3726k	3.56M	438	96	14.31M	12.19M	201.8k	70.9	1:11h
p3847k	2.96M	913	96	11.60M	9.88M	83.9k	138.2	1:26h
p3881k	3.69M	178	64	14.04M	12.02M	111.2k	126.3	24:41m

TABLE II. FAULT-FREE SIMULATION RUNTIME.

Circuit(1)	Pattern-Pairs(2)	Logic-Level		Switch-Level					
		Event-	GPU	Worst (GPU)			Best (GPU)		
		Driven(3)	[34](4)	Time(5)	MEPS(6)	X(7)	Time(8)	MEPS(9)	X(10)
s38417	348	5.00s	351ms	552ms	12.0	9	385ms	17.2	13
s38584	563	12.71s	458ms	742ms	17.5	17	455ms	28.5	28
b17	2135	1:03m	478ms	3.11s	29.3	20	1.19s	76.6	52
b18	3174	8:27m	1.12s	16.41s	24.2	31	4.95s	80.3	102
b19	4651	27:13m	2.50s	49.70s	23.4	33	13.77s	84.5	119
b22	1190	22.84s	403ms	2.16s	15.3	11	779ms	42.5	29
p77k	1979	3:27m	1.04s	13.28s	10.5	16	3.13s	44.6	66
p141k	2043	5:56m	1.38s	14.47s	25.1	25	4.85s	75.1	73
p267k	3181	13:32m	1.76s	19.51s	35.6	42	6.84s	101.6	119
p330k	5928	1:04h	3.90s	48.04s	35.4	80	15.63s	108.8	245
p418k	3676	29:09m	6.96s	58.97s	27.4	30	14.59s	110.9	120
p469k	347	3:01m	723ms	6.95s	5.2	26	2.13s	17.0	84
p500k	5012	1:12h	13.47s	1:59m	22.3	36	26.82s	98.5	160
p533k	3417	1:04h	7.62s	1:49m	21.3	35	26.12s	88.5	146
p951k	7063	3:12h	23.86s	9:12m	14.0	21	1:06m	118.2	176
p1522k	17980	7:42h	42.09s	28:55m	11.3	16	3:01m	108.3	153
p2927k	22107	19:12h	1:18m	1:15h	8.2	15	5:17m	116.9	218
p3188k	26502	41:16h	3:30m	3:45h	5.6	11	12:27m	101.2	199
p3726k	15512	43:09h	2:25m	2:36h	5.9	17	11:09m	82.7	232
p3847k	31653	49:11h	4:38m	7:06h	3.7	7	16:22m	95.4	180
p3881k	12092	22:57h	2:28m	2:47h	4.5	8	10:56m	68.1	126

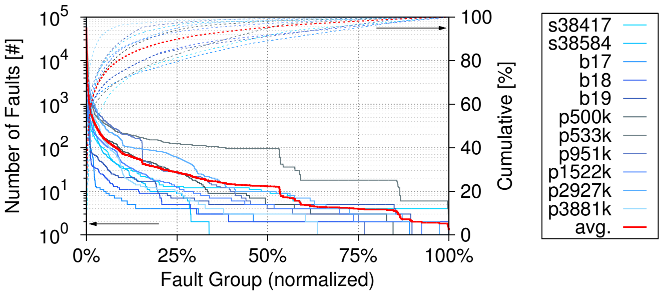


Fig. 11. Sizes of obtained fault groups (left axis, log-scale) and cumulative amount of processed faults (dotted, right axis) after simulation.

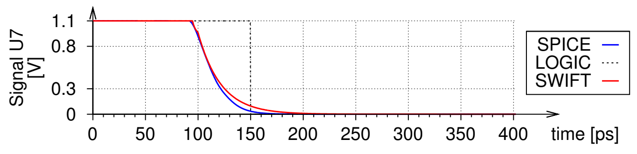
transition fault test pattern sets ($n = 10$) for the circuits with a test coverage of over 98.7% in average. The circuit and the number of pattern pairs obtained from ATPG are given in columns 1 and 2. The runtimes of a timing simulation at logic-level using a commercial event-driven simulator and the GPU-accelerated simulator of [34] are given in columns 3 and 4 respectively for comparison. For the simulation with SWIFT, thread-grid dimensions have been adjusted to the test pattern set (cf. Table I, Col. 5). Column 5 to 7 show the *worst case* runtime of SWIFT, the throughput performance in *million node evaluations per second* (MEPS) and the respective speedup (Col. 7) compared to the event-driven approach. The *worst case* runtimes refer to simulation runs without pre-initialized waveform capacities, which can trigger an *overflow mechanism* in the simulation [33]. The *best case* runtime performance after initial calibration is given in column 8 to 10.

As shown, the speedups of the worst case simulation range from 7 to 80× when compared to the unparallelized logic level event-driven simulation. While smaller to medium circuits are still processed within seconds or few minutes, the larger circuits take up to a few hours, as the overflow calibration procedure is performed by the host system itself. In the best case, on the other hand, the simulation runs at full speed since no calibration is necessary and the simulation speedup increases up to 245× (p330k). Circuits of medium size require only a few seconds, and for the million cell designs runtimes dropped to a few minutes. The achieved speedups are typically higher for the larger circuits, since the relative initialization and synchronization overhead between the host diminishes, leaving a better utilization and occupancy of the GPU computing resources with a throughput performance of up to 118 MEPS (p951k). Thus, the parallel switch level simulation allows to outperform conventional logic level timing simulation despite the more detailed abstraction level.

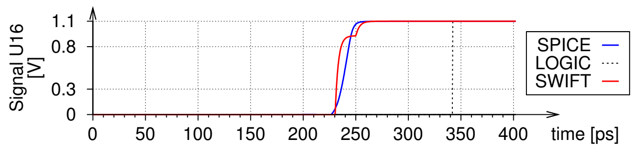
Fig. 12 compares signal waveforms extracted from SPICE, logic level and switch level simulation [32]. For this experiment a chain of two-input NAND cells has been synthesized that enforces a *multiple input switching* effect (MIS) at all even stages throughout the chain. A rising transition was used as input stimuli, thus, at all even stages the cell output is charged via parallel PMOS transistors, hence causing a faster switching time for rising transitions. While the switch level simulation is able to reflect this effect, it is completely ignored in logic simulation, which introduces small errors at all stages. These errors accumulate throughout the design up to a delay deviation of roughly 30 percent of the reference delay. Again, this effect is more severe for cells with three or four inputs.

C. Fault Simulation

For the fault set generation, the nominal clock period of each circuit was extracted from the latest transition time and an additional safety margin of 10 percent was added. All outputs with a slack of less than 25 percent were selected and traced back to the circuit inputs. The RRC cells in the traced input cones were selected as possible fault locations. As fault models, both resistive open transistor faults in the PMOS and NMOS networks of the RRC-cells, as well as capacitive faults at the cell outputs are considered. To observe the impact of



a) Signal waveforms at stage 7: Falling transition without MIS.



b) Signal waveforms at stage 16: Rising transition with MIS.

Fig. 12. Waveforms from electrical, logic and switch level simulation of a signal transition propagating through a chain of two-input NAND cells.

each fault parameter on the fault detection, the fault locations were investigated for multiple fault sizes.

Table III summarizes the simulation results for the resistive open transistor faults. Out of the possible fault location candidates, a maximum of 1000 random RRC-cells were picked for simulation assuming one fault at each transistor. The three chosen fault sizes range between the conducting and blocking resistances of the transistors in the cells, which have been obtained from SPICE simulations of the transistor models [52], [53]. The total number of faults simulated for each fault size is shown in the second column. Column 3 shows the total simulation for processing the three sets of faults for all patterns and column 4 provides the average simulation time per fault group. The number of *detected* (DT) and *possibly detected* (PD) faults with a size of 10kΩ is given in column 5 and 6. The remaining faults are *undetected* (UD) by the pattern set. Similarly, the simulation results for 100kΩ and 1MΩ are given in columns 7 to 8 and 9 to 10 respectively.

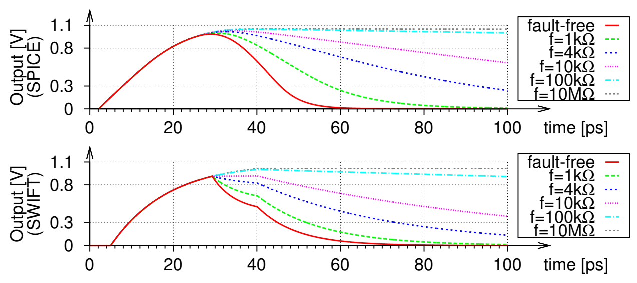
As shown, with a fault size of 10kΩ only few faults were detectable, as the effect is typically in the range of a few cell delays, which is often covered by the clock margin but many faults are *possibly detected* which indicates timing violations. After shifting to 100kΩ, many possibly detected faults turn detectable and further faults become visible. As expected, for the 1MΩ fault set the detection ratio is the highest, since within the clock interval the behavior of many faults is similar to those of transition faults. Although in this case, the timing behavior of the cells is severely impacted, the functional behavior is still correct. The faults behave like weak *stuck-open transistor faults* hindering charge from moving normally. Similar to a transition fault, signal transitions are suspended within the constrained clock interval. Hence, the high detection ratio. The high spatial correlation of the faults unfortunately prohibits the fault grouper from effectively grouping the fault set (efficiency avg. 3.8, median 2.7). The runtime of the grouping itself took less than a second for all cases and is negligible. While the calibration of the waveform memory saturates quickly after the first groups, the average runtimes per group gets close to the *best case* simulation times. Sometimes, the fault simulation is able to avoid the memory initialization overhead (transferring stimuli, circuit data), since data it is already present in the memory. This allows to amortize some calibration overhead from fault injection and also can result in lower runtimes compared to those reported in the full-speed runs of Table II.

TABLE III. RESISTIVE OPEN FAULT SIMULATION (MAX. 1000 CELLS).

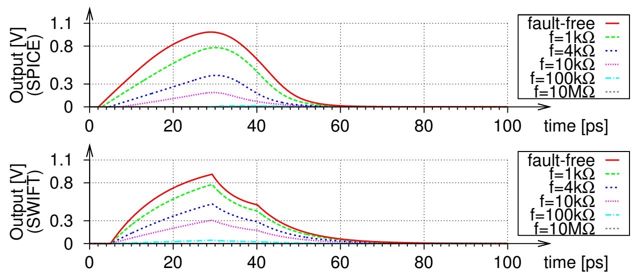
Circuit(1)	Faults *(coll.)*(2)	Runtime		Fault Size (ΔR_f)					
				10kΩ		100kΩ		1MΩ	
		Total(3)	Group(4)	DT(5)	PD(6)	DT(7)	PD(8)	DT(9)	PD(10)
b17	2543×3	56:55m	964ms	0 (0.0%)	1 (0.0%)	1020 (40.1%)	77 (3.0%)	2312 (90.9%)	0 (0.0%)
b18	2616×3	6:33h	7.14s	0 (0.0%)	0 (0.0%)	972 (37.2%)	50 (1.9%)	2497 (95.5%)	0 (0.0%)
p77k	2685×3	5:14h	2.58s	0 (0.0%)	0 (0.0%)	453 (16.9%)	25 (0.9%)	1678 (62.5%)	6 (0.2%)
p141k	2313×3	2:37h	4.67s	0 (0.0%)	0 (0.0%)	1403 (60.7%)	72 (3.1%)	2202 (95.2%)	0 (0.0%)
p267k	2275×3	2:24h	8.04s	6 (0.3%)	4 (0.2%)	1492 (65.6%)	83 (3.6%)	2198 (96.6%)	0 (0.0%)
p330k	2573×3	9:01h	15.28s	9 (0.3%)	8 (0.3%)	2038 (79.2%)	125 (4.9%)	2560 (99.5%)	0 (0.0%)
p418k	2460×3	13:05h	13.42s	0 (0.0%)	1 (0.0%)	1361 (55.3%)	43 (1.7%)	2378 (96.7%)	0 (0.0%)
p533k	2615×3	8:16h	57.01s	0 (0.0%)	0 (0.0%)	1692 (64.7%)	35 (1.3%)	2548 (97.4%)	0 (0.0%)

To investigate the fault modeling of the SWIFT simulation approach, signal changes after injecting and simulating faults in RRC cells has been observed. Fig. 13 shows a visualization of the effects of *resistive open transistor* faults in a two-input NOR-cell. The fault-free transient response as well as the simulation of discrete fault sizes have also been performed in SPICE for comparison. Initially, the two inputs are set to (*high*, *low*) causing an initial 0V output signal, before switching to (*low*, *low*). The input change triggers a charging process at the output load towards 1.1V. Eventually, the inputs switch to (*low*, *high*), after which the output load starts to discharge again. In the first case (a), resistive faults have been injected into the parallel pull-down net, thus causing a slow falling transition after the second input switch, as expected. For higher ohmic resistances (10MΩ) the drain current of the affected transistor is too small to discharge the load, thus the output level sustains at a *high* level. Whereas in the second case (b), the faults were injected into the serial pull-up net, that strongly affects the rising transition of the output hazard. With increasing fault size, the pull-up net becomes incapable of charging the output load in time, before the second input transition arrives. As shown, the behavior of the faults simulated in the SWIFT simulator shows a fairly high similarity compared to SPICE.

For the capacitive fault simulation, 1000 fault locations were investigated for three different fault sizes in each circuit. The sizes have been chosen as multiples of the typical average net load (i.e., 10fF for the circuits used) which were injected into the output load capacitor of the cells. As opposed to the resistive faults, no structural collapsing was performed. The results are summarized in Table IV similar to Table III. As shown, the fault detection gradually increases with the fault size. In case of 1pF, most of the faults were either *detected* (DT) or *undetected*. However, a few locations in b18 and p77k still showed possible *detections* (PD), due to the comparably higher depth of the circuits and larger slacks at the respective fault locations. For any (finite) fault size, capacitive faults do not interfere with the functional behavior of cells, thus eventually providing correct output values after some time.

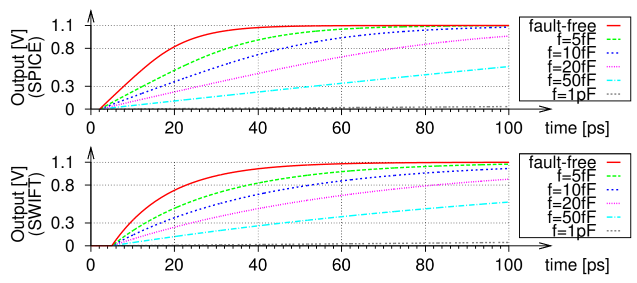


a) Injection at an NMOS-transistor in the *parallel* pull-down net.

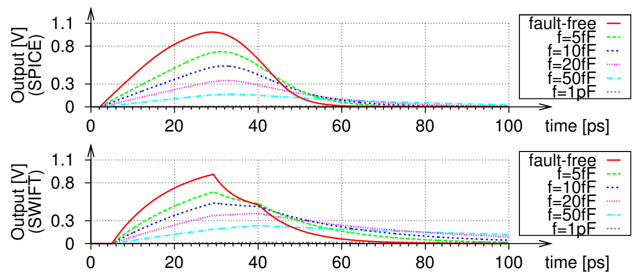


b) Injection at a PMOS-transistor in the *serial* pull-up net.

Fig. 13. Behavior of a resistive-open transistor fault in a) NMOS- and b) PMOS-transistors of a NOR-cell in presence of an input hazard.



a) Rising signal transition affected by capacitive faults.



b) Signal hazard affected by a capacitive fault.

Fig. 14. Behavior of capacitive faults at the NOR-cell output affecting a) a single rising transition, b) a hazard.

TABLE IV. CAPACITIVE FAULT SIMULATION (MAX. 1000 CELLS).

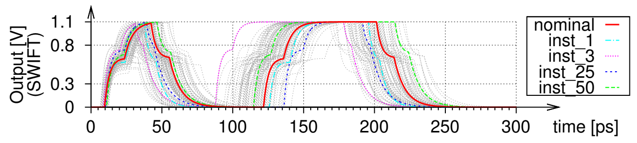
Circuit(1)	Faults(2)	Runtime		Fault Size (ΔC_f)					
				50fF		100fF		1pF	
		Total(3)	Group(4)	DT(5)	PD(6)	DT(7)	PD(8)	DT(9)	PD(10)
b17	1000×3	26:52m	1.13s	17 (1.7%)	6 (0.6%)	82 (8.2%)	15 (1.5%)	939 (93.9%)	0 (0.0%)
b18	1000×3	2:55h	8.39s	9 (0.9%)	4 (0.4%)	95 (9.5%)	18 (1.8%)	969 (96.9%)	3 (0.3%)
p77k	1000×3	2:07h	2.81s	0 (0.0%)	0 (0.0%)	18 (1.8%)	8 (0.8%)	650 (65.0%)	3 (0.3%)
p141k	1000×3	1:19h	5.40s	46 (4.6%)	16 (1.6%)	223 (22.3%)	40 (4.0%)	880 (88.0%)	0 (0.0%)
p267k	1000×3	1:11h	9.26s	89 (8.9%)	61 (6.1%)	314 (31.4%)	68 (6.8%)	865 (86.5%)	0 (0.0%)
p330k	1000×3	3:50h	17.45s	105 (10.5%)	45 (4.5%)	300 (30.0%)	37 (3.7%)	954 (95.4%)	0 (0.0%)
p418k	1000×3	6:11h	15.07s	10 (1.0%)	2 (0.2%)	91 (9.1%)	26 (2.6%)	941 (94.1%)	0 (0.0%)
p533k	1000×3	3:58h	1:10m	8 (0.8%)	9 (0.9%)	192 (19.2%)	22 (2.2%)	984 (98.4%)	0 (0.0%)

The behavior of capacitive faults at the cell output are shown in Fig. 14. In contrast to their resistive counterparts, capacitive faults do not impact the functional behavior (stationary voltage) of the cell, but only affect its timing. Eventually, the capacitor gets charged and the cell delivers the correct output level. The capacitive faults both impact rising and falling transitions at the same time (Fig. 14b), similar to small delay faults at logic level with rising and falling impact. Again, this behavior was also observed in SWIFT. However, although the capacitive fault has a discrete and fixed value, the delay impact of the faults vary in presence of MIS. This cannot be captured by logic level time simulation based on SDF descriptions appropriately and thus needs evaluation at lower levels.

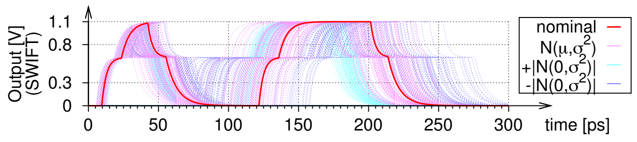
D. Variation Analysis

To demonstrate the effect of parametric variation at switch level, Monte-Carlo experiments have been conducted for a population of 100 random circuit instances. All resistances in the circuits have been altered using a Gaussian normal distribution $\mathcal{N}(\mu, \sigma^2)$ with μ as the nominal parameters and $\sigma = 0.2 \cdot \mu$ as standard deviation. Fig. 15a) shows the signal waveforms obtained at an output in the nominal instance and the entire population. As shown, the delay is strongly altered which can result in faster and slower instances (*inst_3* vs. *inst_50*). Also, the later transitions show a wider spread throughout the population, due to accumulation of the randomness over long paths, which may eventually affect the detection of faults.

In Fig. 15b) the simulations were repeated with altered transistor threshold voltages. Again, a Gaussian normal distribution with 20% standard deviation is assumed for each ΔV causing a typical spread output behavior with faster and slower instances. In a second scenario $(+|\mathcal{N}(0, \sigma^2)|)$, a folded normal distribution was assumed that increases threshold voltages of the transistors relative to the bulk potential (VDD or GND). While expecting a general increase in the circuit delay, many circuit instances turned out to be faster. For higher threshold voltages in PMOS transistors ($\Delta V > 0$), for example, the cell output is delayed for *falling* input transitions at the gate terminal. At the same time, the PMOS transistors switch off earlier for *rising* input transitions, which is consistent with simulation in SPICE (cf. Fig. 16). In a third case $(-|\mathcal{N}(0, \sigma^2)|)$, the folded distribution was applied to lower the threshold voltages only ($\Delta V < 0$). In contrast to the high threshold scenario, the reduction resulted mainly in slower circuit instances.



a) Random variation in resistances ($\mathcal{N}(\mu, \sigma^2)$ with $\sigma = 0.2\mu$).



b) Random variation in transistor threshold voltages.

Fig. 15. Signal waveforms (at output FD1_915_D) after simulation of 100 random instances of circuit s38417 for pattern #3.

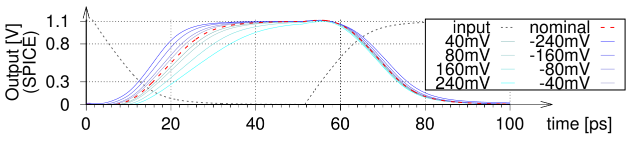


Fig. 16. Propagation of an input hazard at an inverter cell with the threshold voltage of the PMOS transistor altered (ΔV).

X. Conclusions

This work presented SWIFT, an approach for fast and accurate switch level fault simulation on data-parallel GPU architectures. The fault simulation utilizes first-order electrical parameters found in CMOS technology to model functional and timing behavior of primitive or complex-cells, as well as for fault modeling. Parametric and parasitic faults are modeled and injected without the need of logical abstraction, thus avoiding the general limitations faced in simulation at logic level. During simulation the full switching history of signals is computed, which allows for timing-accurate evaluation even in the presence of signal slopes, hazards and glitches. Systematic and random variation at device level is efficiently applied during the evaluation of cells by modification of first-order parameters in circuit instances during runtime. Due to the high runtime complexity of general timing-accurate simulation, it is crucial to exploit parallelism during fault simulation as much as possible. In SWIFT, multiple dimensions of parallelism from cells, waveforms, faults and circuit instances are exploited to achieve high simulation throughput for acceleration with over 116 million cell evaluations per second with speedups of up to 245× compared to conventional timing simulation at logic level, even for designs with millions of cells.

Acknowledgment

This work has been funded by the German Research Foundation (DFG) under grant WU 245/16-1 and WU 245/17-1.

References

[1] F. Hapke, W. Redemund, A. Glowatz, J. Rajski, M. Reese, M. Hustava, M. Keim, J. Schloeffel, and A. Fast, “Cell-Aware Test,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 9, pp. 1396–1409, Sep. 2014.

[2] R. L. Wadsack, “Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits,” *The Bell System Technical Journal*, vol. 57, no. 5, pp. 1449–1474, May 1978.

[3] A. D. Singh, “Cell Aware and Stuck-Open Tests,” in *Proc. IEEE 21st European Test Symp. (ETS)*, May 2016, pp. 1–6, Paper 15.1.

[4] H. Cox and J. Rajski, “Stuck-Open and Transition Fault Testing in CMOS Complex Gates,” in *Proc. IEEE Int’l Test Conf. (ITC)*, Sep. 1988, pp. 688–694.

[5] J. C. M. Li, C.-W. Tseng, and E. J. McCluskey, “Testing for Resistive Opens and Stuck Opens,” in *Proc. Int’l Test Conf. (ITC)*, Nov. 2001, pp. 1049–1058.

[6] C. Han and A. D. Singh, “Testing cross wire opens within complex gates,” in *Proc. IEEE 33rd VLSI Test Symp. (VTS)*, Apr. 2015, pp. 1–6.

[7] 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.

[8] 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.

[9] E. Melcher, W. Röthig, and M. Dana, “Multiple input transitions in CMOS gates,” *Microprocessing and Microprogramming*, vol. 35, no. 1–5, pp. 683–690, 1992.

[10] L.-C. Chen, S. K. Gupta, and M. A. Breuer, “A New Gate Delay Model for Simultaneous Switching and Its Applications,” in *Proc. ACM/IEEE 38th Design Automation Conf. (DAC)*, Jun. 2001, pp. 289–294.

[11] X. Lin, W. T. Cheng, and J. Rajski, “On Improving Transition Test Set Quality to Detect CMOS Transistor Stuck-Open Faults,” in *Proc. IEEE 24th Asian Test Symp. (ATS)*, Nov. 2015, pp. 97–102.

[12] H. H. Chen, S. Y. H. Chen, P. Y. Chuang, and C. W. Wu, “Efficient Cell-Aware Fault Modeling by Switch-Level Test Generation,” in *Proc. IEEE 25th Asian Test Symp. (ATS)*, Nov. 2016, pp. 197–202.

[13] L. W. Nagel and D. O. Pederson, “SPICE (Simulation Program with Integrated Circuit Emphasis),” EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M382, Apr. 1973.

[14] K. Peng, Y. Huang, P. Mallick, W. T. Cheng, and M. Tehranipoor, “Full-Circuit SPICE Simulation Based Validation of Dynamic Delay Estimation,” in *Proc. IEEE European Test Symp. (ETS)*, May 2010, pp. 1010–106.

[15] R. E. Bryant, “A Switch-Level Model and Simulator for MOS Digital Systems,” *IEEE Transactions on Computers*, vol. C–33, no. 2, pp. 160–177, Feb. 1984.

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

[17] S. Borkar, “Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation,” *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov. 2005.

[18] B. Becker, S. Hellebrand, I. Polian, B. Straube, W. Vermeiren, and H.-J. Wunderlich, “Massive Statistical Process Variations: A Grand Challenge for Testing Nanoelectronic Circuits,” in *Proc. Int’l Conf. on Dependable Systems and Networks Workshops (DSN-W)*, Jun. 2010, pp. 95–100.

[19] 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.

[20] M. Wagner and H. J. Wunderlich, “Efficient Variation-Aware Statistical Dynamic Timing Analysis for Delay Test Applications,” in *Proc. Conf. on Design, Automation Test in Europe (DATE)*, Mar. 2013, pp. 276–281.

[21] M. Fujita, “Variation-Aware Analysis and Test Pattern Generation Based on Functional Faults,” in *Proc. IEEE Computer Society Annual Symp. on VLSI (ISVLSI)*, Jul. 2014, pp. 273–277.

[22] 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. 19th European Test Symp. (ETS)*, May 2014, pp. 1–6.

[23] M. S. Golanbari, S. Kiamehr, M. Ebrahimi, and M. B. Tahoori, "Variation-Aware Near Threshold Circuit Synthesis," in *Proc. Conf. on Design, Automation Test in Europe (DATE)*, Mar. 2016, pp. 1237–1242.

[24] 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.

[25] M. Abramovici, B. Krishnamurthy, R. Mathews, B. Rogers, M. Schulz, S. Seth, and J. Waicukauski, "What is the Path to Fast Fault Simulation?" in *Proc. Int'l Test Conf. (ITC)*, Sep. 1988, pp. 183–192.

[26] 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.

[27] D. Chatterjee, A. DeOrio, and V. Bertacco, "Event-Driven Gate-Level Simulation with GP-GPUs," in *Proc. ACM/IEEE 46th Design Automation Conf. (DAC)*, Jul. 2009, pp. 557–562.

[28] K. Gulati, J. F. Croix, S. P. Khatri, and R. Shastry, "Fast Circuit Simulation on Graphics Processing Units," in *Proc. 14th Asia and South Pacific Design Automation Conf. (ASP-DAC)*, Jan. 2009, pp. 403–408.

[29] 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.

[30] M. Li and M. S. Hsiao, "3-D Parallel Fault Simulation With GPGPU," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 10, pp. 1545–1555, Oct. 2011.

[31] L. Han, X. Zhao, and Z. Feng, "TinySPICE: A Parallel SPICE Simulator on GPU forMassively Repeated Small Circuit Simulations," in *Proc. 50th Design Automation Conf. (DAC)*, May 2013, pp. 1–8.

[32] E. Schneider, S. Holst, X. Wen, and H.-J. Wunderlich, "Data-Parallel Simulation for Fast and Accurate Timing Validation of CMOS Circuits," in *Proc. IEEE/ACM 33rd Int'l Conf. on Computer-Aided Design (ICCAD)*, Nov. 2014, pp. 17–23.

[33] 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.

[34] E. Schneider, M. A. Kochte, S. Holst, X. Wen, and H. J. Wunderlich, "GPU-Accelerated Simulation of Small Delay Faults," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 5, pp. 829–841, May 2017.

[35] E. Schneider and H.-J. Wunderlich, "High-Throughput Transistor-Level Fault Simulation on GPUs," in *Proc. IEEE 25th Asian Test Symp. (ATS)*, Nov. 2016, pp. 151–156.

[36] NVIDIA Corporation. (2017) High Performance Computing (HPC) and Supercomputing — NVIDIA Tesla — NVIDIA. [Online]. Available: http://www.nvidia.com/object/tesla-supercomputing-solutions.html [Accessed: Mar. 2, 2017]

[37] 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.

[38] 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.

[39] 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.

[40] S. Holst, E. Schneider, and H. Wunderlich, "Scan Test Power Simulation on GPGPUs," in *Proc. IEEE 21st Asian Test Symp. (ATS)*, Nov. 2012, pp. 155–160.

[41] IEEE Computer Society, "IEEE Standard for Integrated Circuit (IC) Open Library Architecture (OLA)," *IEEE Std 1481-2009*, pp. c1–658, Mar. 2010.

[42] A. E. Ruehli and G. S. Ditlow, "Circuit Analysis, Logic Simulation, and Design Verification for VLSI," *Proceedings of the IEEE*, vol. 71, no. 1, pp. 34–48, Jan. 1983.

[43] 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.

[44] 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.

[45] C. Sebeke, J. P. Teixeira, and M. J. Ohletz, "Automatic Fault Extraction and Simulation of Layout Realistic Faults for Integrated Analogue Circuits," in *Proc. European Conf. on Design and Test (EDTC)*, Mar. 1995, pp. 464–468.

[46] F. J. Ferguson and J. P. Shen, "A CMOS Fault Extractor for Inductive Fault Analysis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 11, pp. 1181–1194, Nov. 1988.

[47] K. U. Giering, C. Sohrmann, G. Rzepa, L. Hei, T. Grasser, and R. Jancke, "NBTI modeling in analog circuits and its application to long-term aging simulations," in *Proc. IEEE Int'l Integrated Reliability Workshop Final Report (IIRW)*, Oct. 2014, pp. 29–34.

[48] D. Lorenz, G. Georgakos, and U. Schlichtmann, "Aging Analysis of Circuit Timing Considering NBTI and HCI," in *Proc. 15th IEEE Int'l On-Line Testing Symp. (IOLTS)*, Jun. 2009, pp. 3–8.

[49] M. Shao, Y. Gao, L.-P. Yuan, and M. D. R. Wong, "IR drop and Ground Bounce Awareness Timing Model," in *Proc. IEEE Computer Society Annual Symposium on VLSI: New Frontiers in VLSI Design (ISVLSI)*, May 2005, pp. 226–231.

[50] R. M. Karp, "Reducibility Among Combinatorial Problems," in *Proc. Symp. on Complexity of Computer Computations*, Mar. 1972, pp. 85–103.

[51] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Publishing, 1997.

[52] W. Zhao and Y. Cao, "New Generation of Predictive Technology Model for Sub-45 nm Early Design Exploration," *IEEE Trans. on Electron Devices*, vol. 53, no. 11, pp. 2816–2823, Nov. 2006.

[53] Nanoscale Integration and Modeling (NIMO) Group. Predictive Technology Model (PTM). [Online]. Available: http://ptm.asu.edu/ [Accessed: Feb. 2, 2016]

Eric Schneider (S'14) received the Diploma (Dipl.-Inf.) degree in computer science from the University of Stuttgart, Stuttgart, Germany, in 2012, where he is currently pursuing the Ph.D. degree with the Institute of Computer Architecture and Computer Engineering.

His research interests include circuit simulation, test and diagnosis, as well as parallel programming on graphics processing units (GPUs) to accelerate and aid design and test validation tasks.

Hans-Joachim Wunderlich (M'85–F'09) 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 associated 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 280 reviewed scientific papers in journals and conferences. His research interests include test, reliability, fault tolerance and design automation of microelectronic systems.