High-Throughput Transistor-Level Fault Simulation on GPUs

Schneider, Eric; Wunderlich, Hans-Joachim

Proceedings of the 25th IEEE Asian Test Symposium (ATS'16) Hiroshima, Japan, 21-24 November 2016

doi: http://dx.doi.org/10.1109/ATS.2016.9

Abstract: Deviations in the first-order parameters of CMOS cells can lead to severe errors in the functional and time domain. With increasing sensitivity of these parameters to manufacturing defects and variation, parametric and parasitic-aware fault simulation is becoming crucial in order to support test pattern generation. Traditional approaches based on gate-level models are not sufficient to represent and capture the impact of deviations in these parameters in either an efficient or accurate manner. Evaluation at electrical level, on the other hand, severely lacks execution speed and quickly becomes inapplicable to larger designs due to high computational demands. This work presents a novel fault simulation approach considering first-order parameters in CMOS circuits to explicitly capture CMOS-specific behavior in the functional and time domain with transistor granularity. The approach utilizes massive parallelization in order to achieve high-throughput acceleration on Graphics Processing Units (GPUs) by exploiting parallelism of cells, stimuli and faults. Despite the more precise level of abstraction, the simulator is able to process designs with millions of gates and even outperforms conventional simulation at logic level in terms of modeling accuracy and simulation speed.

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**
©2016 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.

High-Throughput Transistor-Level Fault Simulation on GPUs

Eric Schneider and Hans-Joachim Wunderlich
University of Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany
schneiec@iti.uni-stuttgart.de, wu@informatik.uni-stuttgart.de

***Abstract*—Deviations in the first-order parameters of CMOS cells can lead to severe errors in the functional and time domain. With increasing sensitivity of these parameters to manufacturing defects and variation, parametric and parasitic-aware fault simulation is becoming crucial in order to support test pattern generation. Traditional approaches based on gate-level models are not sufficient to represent and capture the impact of deviations in these parameters in either an efficient or accurate manner. Evaluation at electrical level, on the other hand, severely lacks execution speed and quickly becomes inapplicable to larger designs due to high computational demands.**

This work presents a novel fault simulation approach considering first-order parameters in CMOS circuits to explicitly capture CMOS-specific behavior in the functional and time domain with transistor granularity. The approach utilizes massive parallelization in order to achieve high-throughput acceleration on Graphics Processing Units (GPUs) by exploiting parallelism of cells, stimuli and faults. Despite the more precise level of abstraction, the simulator is able to process designs with millions of gates and even outperforms conventional simulation at logic level in terms of modeling accuracy and simulation speed.

***Keywords*—fault simulation; transistor level; switch level; GPUs**

I. Introduction

As circuit structures are shrinking, physical defects express a more complex and severe impact on the switching behavior of CMOS cells [1, 2]. With the strict high-performance and low-power requirements of today's circuits and due to omnipresent variability of the manufacturing processes, even small deviations of first-order parameters in the layout of CMOS cell structures are sufficient to cause faults at transistor-level that lead to severe system failures. Transistor-level faults and parametric deviations at cell-level, such as *resistive bridges*, *cross-wire* and *resistive opens* [3], often cannot be detected by conventional tests due to complex activation and propagation mechanisms in the time domain [4]. Therefore, it is crucial to thoroughly validate test schemes and test generation for detecting faults at transistor-level, which have recently became focus of current test research [5].

The simulation of faults is an essential task for test set validation, test pattern generation and reliability assessment. Many of the defects found in CMOS cells have impact in both functional and timing aspects [4] and thus require consideration of timing with glitch-accuracy. For large circuits, the simulation itself is typically performed with gate-level abstraction due to the runtime complexity being several orders of magnitude smaller compared to lower-level approaches such as electrical simulation. The behavior of defects found at lower abstraction levels is then mapped to inputs for higher level simulation algorithms [6], and during the process often some of the modeling accuracy has to be sacrificed.

However, the functional and timing behavior of CMOS cells is not only influenced by intrinsic parameters, but other factors as well: Simultaneous switching at cell input terminals [7] can cause different transition ramps and speed-up the actual switching process of a cell. This effect cannot be considered efficiently and accurately using gate-level approaches. By ignoring these effects, small errors can occur at various stages during fault simulation which, especially for faults with impact in the time domain, quickly accumulate to and exceed the amount of the targeted fault sizes. Therefore, these detections are likely to result in false positives by overestimating the actual fault coverage.

For accurate validation of test patterns targeting timing-related faults in complex CMOS cells, it is necessary to model faults with little or no abstraction at all. The simulation of these faults then requires algorithms with sufficiently high time-resolution and accuracy. Since higher accuracy involves more computational complexity and more data to process, runtimes quickly rise, which causes conventional algorithms to become inapplicable to larger designs. In order to tackle this complexity issue, algorithms are being parallelized for the execution on data-parallel architectures, such as *Graphics Processing Units* (GPUs), which have well established in high-performance computing, since they are known to provide vast computational throughput [8].

In this work we present the first approach to utilize the computing capabilities of GPUs to enable accelerated fault simulation at transistor-level. It combines massive-throughput parallelization concepts of circuit timing simulation and parallel fault modeling at CMOS-level to enable fast parallel fault simulation with transistor-granularity. By carefully selecting available dimensions of parallelism and efficient organization of the data-structures, the proposed approach allows for:

- Efficient and explicit modeling of functional and timing-related parametric and parasitic faults at transistor-level, achieving higher modeling capability and accuracy compared to logic level,
- Overhead-free fault injection and fault evaluation method for fast and transparent parallel fault simulation on GPUs outperforming conventional logic level approaches,
- Accurate and comprehensive fault analysis applicable to larger designs with millions of gates.

The next section summarizes simulation algorithms for timing validation and introduces the basic concepts for capturing functional and time behavior of CMOS circuits. Section III presents the novel simulation scheme for efficient transistor-level fault simulation by explaining the fault modeling and the fault injection mechanism for parallel fault simulation on GPUs. Section IV presents the calculation of fault syndromes for evaluation of the fault detection. In section V, experimental results are discussed, which prove that the new GPU-based transistor-level fault simulator outperforms even commercial timing simulators at logic-level that only support timing validation without consideration of any faults.

II. Related Work

Timing simulation is usually done at gate-level, where the delay of a gate or cell is expressed as a function of events at the inputs (e.g. rising and falling transition at a specific pin at a given time point). All input events are then typically processed in an event-driven manner from earliest to latest signal transition in order to calculate the response at the output. Most approaches utilize the so-called *pin-to-pin delay*. With pin-to-pin delay, individual rising and falling delay times can be assigned to each physical pin of a cell, which are typically provided in *Standard Delay Format* (SDF) by the synthesis tools. In addition to the complexity of the time domain, the timing-accurate evaluation of circuits typically involves calculations with floating-point numbers, which is much more expensive than *untimed* (*zero-delay*) logic simulation.

With the introduction of the general purpose computing on *Graphics Processing Units* (GPUs) many *untimed* logic and fault simulation approaches have spawned. In [9–12] algorithms were presented that exploit both *structural-* as well as *data-parallelism* to solve multiple problems simultaneously for achieving high speedup over a serial execution.

First accelerated approaches for statistical timing simulation [13] or waveform-accurate simulation [14] at logic-level have been proposed, which allow to exploit the floating-point throughput capabilities of the GPUs. Although the simulators are able to achieve high speedups, they lack the essential modeling capabilities for further increasing the accuracy of the simulation efficiently, such as the ability to consider CMOS-related pattern-dependent delays [7] and transition ramp times.

At circuit-level, methods for accelerating SPICE simulations [15] have been proposed as well [16, 17]. Yet, these approaches offer either too little speedup to be applied on a large scale or are even limited to designs composed of few transistors only. In [18] a *High-Throughput Oriented Parallel Switch-Level Simulator* (HiTOPS) was proposed, a first timing simulator for execution on GPUs for fast simulation of circuits at CMOS level. It models delays at transistor granularity under consideration of first-order parameters of standard cells while achieving high simulation throughput even for designs with millions of gates. The simulation model is based on *Resistor-Resistor-Capacitor* (RRC-) cells, which are extracted from identification of current connected meshes in the transistor netlist of primitive or complex cells as shown in Fig. 1.

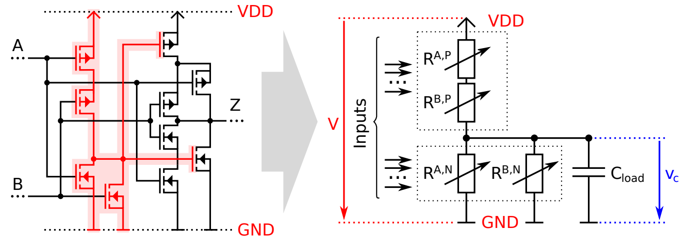


Fig. 1. RRC-cell extracted from a complex 10T XOR-cell with input-controlled voltage divider (PMOS/NMOS nets) and capacitor C_{load}.

Each transistor of an RRC-cell is viewed as a voltage-controlled resistor that behaves like a threshold-based binary switch. Depending on the applied gate voltage v, each transistor assumes a *blocking* or *conducting* state which are modeled by high and low resistance parameters, respectively. During the simulation, the transistors switch states every time the associated input signals undergo transitions crossing their threshold voltage V_{th}:

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

After each switch, the resistances of the transistor-nets (either the PMOS- or the NMOS-net) change and the RRC-cell output level aims for a new stationary voltage. All changes in the stationary voltage cause the output load capacitor to (dis-)charge via the voltage divider over time with the transient response being modeled by exponential curves to better reflect the electrical behavior.

III. Transistor-Level Fault Simulation

In this work, efficient modeling and simulation of faults at transistor-level is realized by introducing a simulation scheme that allows to obtain a high degree of parallelism, simulation speedup and accuracy during execution. The proposed simulation scheme exploits three dimensions of parallelism, namely: a) *fault parallelism*, b) *cell-parallelism* and c) *waveform-parallelism*, utilizing both structural as well as data-independence of RRC-cells as depicted in Fig. 2. First, interaction of parametric and parasitic faults and their locations are identified in order to form *fault groups* [19] for parallel injection. Secondly, the topological dependencies within a circuit are used to simulate data-independent RRC-cells in parallel. Finally, input stimuli, either single test vectors or test vector sequences (i.e., delay test) are considered independent and hence will be evaluated concurrently as well. By careful consideration of all three dimensions, the proposed execution scheme enables faster and more accurate simulation of faults modeled at transistor-level than compared to using logic-level simulation.

In the following, the fault modeling as well as the fault injection scheme are discussed along with the efficient grouping of faults for parallel injection into the simulation instances.

A. Fault Modeling

The abstraction level of the underlying fault model corresponds to that of the RRC-cells. Any deviation of the cell description can be used, which may require either manual work

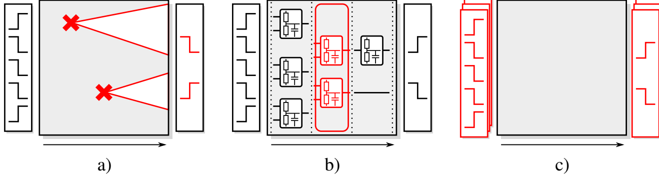


Fig. 2. Dimensions of parallelism exploited during simulation: a) fault-parallelism, b) cell-parallelism, c) waveform-parallelism.

on a complex cell library or additional effort for automatically using a *user-defined fault model* as used in cell-aware test [5]. Methods for mapping defects and extracting relevant faults on either electrical or logic domain based on the layout have been proposed in [20]. In this work we focus on RRC-cell-based fault modeling using the available cell parameters, i.e., of the transistor in the *pull-up* and *pull-down* networks. This allows to model many of the *resistive* faults in the with finite (*resistive-open*, *shorts*) or infinite (*stuck-open*) resistance and threshold variations, as well as deviations in the output capacitance. This way, low-level defect information can be directly employed and does not need to be mapped to a logical behavior, which avoids the common modeling restrictions faced on logic level and allows to act beyond their modeling capabilities. The scheme can be applied to support more fault models, such as cell-internal bridging faults, by assuming, for example, additional resistors in the transistor networks of the RRC-cell descriptions.

In the basic simulation model each RRC-cell is assigned a description of its internal first-order parameters composed of output capacitance and transistor properties. The latter is organized as set $\mathcal{R}$ containing tuples which each describes the functional behavior of a transistor (PMOS as well as NMOS types) by the threshold voltage levels and pairs of individual resistances for conducting and blocking states respectively:

$$\mathcal{R} = \{(V_{th}^0, \{R_{off}^0, R_{on}^0\}), (V_{th}^1, \{R_{off}^1, R_{on}^1\}), ...\}.$$

A *resistive* transistor fault f in a cell is represented by a tuple $f = (loc, \Delta R_f)$ composed of the culprit transistor as *fault location* and resistance values as the *fault size*. The injection of the fault into the circuit is done by modifying the RRC-cell data tuple $(V_{th}^{loc}, \{R_{off}^{loc}, R_{on}^{loc}\})$ of the associated transistor in the circuit description, such that $\widetilde{R}_i^{loc} := R_i^{loc} + \Delta R_f$ with $i \in \{off, on\}$. For example, in order to model open transistors, the conducting resistance is increased ($\Delta R_f > 0$) and, vice versa, for shorted transistors the blocking resistance is reduced ($\Delta R_f < 0$).

Resistive or capacitive faults in the interconnects are modeled assuming a lumped wire resistance and capacitance [18], which incorporates the resistive property of the fault by distribution over the transistor descriptions of the driving RRC-cell. As for capacitive faults $f = (loc, \Delta C_f)$, that introduce an additional parasitic capacitance ΔC_f, such as wire loads or gate capacitances, the parameter will be added to the RRC-cell internal output load capacitance value $\widetilde{C}_{load} := C_{load} + \Delta C_f$. In a similar manner, threshold-related faults in individual transistors, i.e., caused by aging effects like *Negative-Bias Temperature Instability* (NBTI) and *Hot-Carrier Injection* (HCI), can be expressed. In order to model a particular shift ΔV_{th} in the threshold voltage parameter, the threshold of the targeted PMOS or NMOS transistor is raised or lowered respectively.

B. Fault Injection

Each RRC-fault is injected into the circuit by modifying the parameters in the cell description of the associated RRC-cell according to the fault model prior to the actual simulation process. During this process the fault location is marked as *faulty* for later removal. To the simulation kernel, the presence of faults in the circuit is completely transparent, since the circuit description is always read during the simulation process like in the fault-free case. Additional faults can be injected at the same time in order to represent multi-faults located across the circuit or even within single cells. However, for the sake of simplicity, this work considers the single fault assumption.

Once the fault simulation is completed, the parameters of the original descriptions of all cells currently being marked as *faulty* are restored to their original *nominal* description. Since the fault descriptions are compact, only few small memory transactions are involved during the injection process.

C. Fault Grouping

All faults modeled in RRC-cells manifest as either delayed output transitions and deviations in the signal voltages at the associated cell output $v_c(t)$ which can propagate towards the primary circuit outputs in the sensitized case. Hence, each RRC-fault can only affect the signals in its own output-cone, and it must be ensured that there is no other fault injected at the same time that shares common output logic, or otherwise fault effects might influence each other by masking or adding (single fault assumption). For parallel fault simulation a partitioning of the given fault-set into groups of *mutually independent* faults for parallel injection can be performed [19]. These *fault groups* are obtained by checking the mutual *output-independence* of the respective reachable outputs. Since identifying largest fault groups is a minimum graph coloring problem for which finding an optimal solution is NP-hard [21], we use a heuristic that partitions the faults into suitable groups for injection in reverse topological order starting from the primary outputs towards the inputs [22].

In a first attempt, all faults are assigned to an initial fault group based on their fault location and by looking up their topological successors. By utilizing knowledge about the general circuit structure and by distributing group information from previously processed faults along the netlist, the number of comparisons and grouping attempts can be reduced such that suitable groups are found quickly. Once all fault groups have been determined, the simulator performs repeated simulation runs processing all groups one after another. In each simulation run all the faults contained in a group are injected into the circuit description prior to the actual simulation.

IV. Fault Detection and Evaluation

The detection of a fault is determined by observing the values of the waveforms of all output signals in its output cone at a given user-specified sample time t_{samp}. Each waveform

is stored in the global waveform memory on the GPU device as an ordered list of *pivots* $(p_0, p_1, ..., p_k)$. Each pivot of a waveform expresses a signal switch of an RRC-cell output (caused by a change in the resistances of the transistor-nets) at a certain point in time. A pivot p is a tuple composed of the time of the associated signal switch t_p, the stationary voltage $\overline{v}_p$ as well as the time constant τ_p which indicates the slope. Given the sample time t_{samp}, the pivot lists of all the output signals are traced until the corresponding curve segments $p_i = (t_i, \overline{v}_i, \tau_i)$ with $t_{i+1} > t_{samp}$ are reached in each waveform. In the meanwhile, the change of the signal voltage that takes place in each curve interval $[t_i, t_{i+1}]$ is determined for the time span $\Delta t = (t_{i+1} - t_i)$ and the previous output value $w(t_i)$ as given by:

$$w(t_{i+1}) := (w(t_i) - \overline{v}_i) \cdot e^{-\frac{\Delta t}{\tau_i}} + \overline{v}_i. \tag{1}$$

The final signal value $w(t_{samp})$ at time $t_{samp} \in [t_i, t_{i+1})$ is computed in the same way by evaluating the latest curve segment starting at t_i for $\Delta t = (t_{samp} - t_i)$ time units, respectively. Since the value domain of RRC-cell outputs is continuous, the value $w(t_{samp}) \in [\text{GND}, \text{VDD}]$ obtained does not have to correspond to a proper logic value that allows to distinguish *right* from *wrong*. Thus, a mapping to discrete logic values along with instructions to derive a comprehensive *syndrome* is required to distinguish between *fault-free* and *faulty* responses. In order to put a sampled output signal in comparison to the *good* simulation, we characterize the sampled value through boundaries.

A. Signal Evaluation

We define a *signal threshold interval* as $(V_{thL}, V_{thH}) \subset [\text{GND}, \text{VDD}]$ with $V_{thL} < V_{thH}$ to distinguish between *high*, *low* and *undefined* signals [23]. Signal values that are within $[\text{GND}, V_{thL}]$ (or $[V_{thH}, \text{VDD}]$) are considered as *low* (resp. *high*) due to the amplification of succeeding cells in CMOS technology. These boundaries can be obtained by characterizing the input/output relationship of succeeding cells and storage elements by determining appropriate transfer functions based on the target technology. Within the range (V_{thL}, V_{thH}) succeeding cells might interpret the output signal differently and cause uncertainties during the propagation. Hence, in this interval the signal is assumed to *undefined* (symbol 'X') and considered pessimistically as *possibly* erroneous due to its weakness. The mapping from a sample $w \in \mathbb{R}$ of a time-continuous signal $w(t)$ to a discrete logic value is then described by

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

B. Discrete Syndrome Calculation

The *syndrome* $syn(t)$ of a cell output $w(t)$ will be used to continuously determine the presence of a *faulty* value at any given time t. It is acquired directly from the waveform by comparing the output value $w(t)$ of the cell with its *fault-free* stable value $w(\infty)$ for $t \to \infty$. By default it is assumed that the *fault-free* responses of a circuit are stable and always have clear *high* or *low* signal values. As usual, the difference of the output value with respect to the *fault-free* value potential then determines the syndrome:

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

Therefore, $syn(t) = 1$ ($syn(t) = 0$) iff the cell produces a definite *faulty* (*fault-free*) signal at time t, or *unknown* in case $w(t)$ is *undefined*.

The calculation and evaluation of the syndrome itself at some sample time t_{samp} is performed by a two-dimensional kernel that invokes threads as shown in Fig. 3. Each thread traverses the pivot list of the computed output waveform $w(t)$ of a specific output pin and given stimuli concurrently until t_{samp} is reached and the waveform value is extracted. For each output, the acquired syndrome information is then further encoded by two bits as symbol to distinguish the three discrete logic cases, merged as bit-strings and finally stored in a separate memory on the GPU device for all input stimuli respectively. The detection of a fault is then determined by looking up the captured syndromes of all the outputs o in its corresponding output-cone O:

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

Since the output waveforms remain untouched during the evaluation, multiple capture time points can be evaluated quickly in succession. Furthermore, individual capture times can be applied for each output in order to model skew in the clock distribution tree.

V. Experimental Results

The presented simulation approach was evaluated for a set of selected benchmark circuits from ISCAS'89, ITC'99 and industrial designs provided by NXP. All circuits have been synthesized using a 45nm digital standard-cell library. During this process all state elements have been removed, thus leaving only the combinational logic structure. Internal

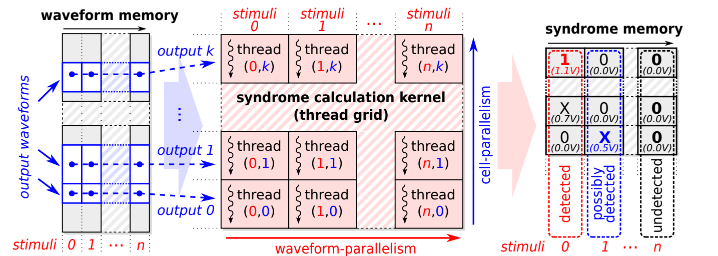


Fig. 3. Parallel syndrome calculation at outputs using a two-dimensional grid of threads with separate memory containing the encoded syndrome symbols.

parameters were then obtained from the layout of the cells as well as extraction from SPICE simulations of the transistor models. For the evaluation of each circuit, n-detect pattern sets composed of stimuli pairs for detecting transition delay faults ($n = 10$) have been generated using a commercial ATPG tool with pattern compaction. The experiments were executed on a host machine equipped with NVIDIA® Tesla® K80 dual-GPU-accelerator cards each of which has 2×2496 cores clocked at 824MHz with access to 12GB of global device memory. However, only a single GPU device was used in the experiments. The host system contains eight Intel® Xeon® processors clocked at 3.0GHz and 128GB of RAM. The peak memory consumption of the host never exceeded 32GB.

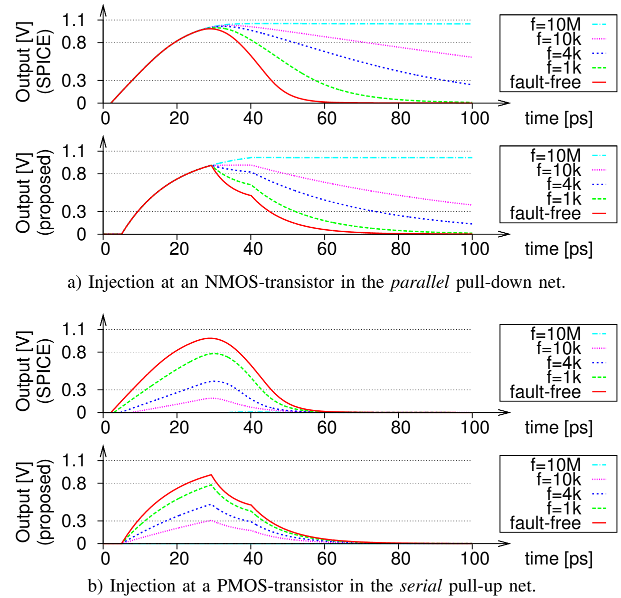
A. Transistor-Level Fault Behavior

In order to evaluate the fault modeling of the proposed approach, the behavior of affected cells has been compared with simulations at electrical level first. Fig. 4 shows the output waveforms of a circuit under the effect of *resistive open transistor* faults in the presence of an input hazard. It depicts the fault-free transient response as well as simulation of different fault sizes in SPICE against the visualized output of the proposed approach. Each of the open faults is modeled and injected as an additional ohmic resistance in a transistor's conducting state. In the first case (a), faults in an NMOS transistor of the pull-down net of a NOR-cell have been investigated for varying sizes. As expected, with increasing fault size, the falling transition at the output gets significantly delayed. For higher ohmic resistances (10MΩ), the resulting drain current is too small to discharge the load within the required time-frame and the output level sustains. This behavior was observed in both SPICE and our simulations as well. In the second case (b), the fault is now located in a PMOS transistor that strongly affects the rising edge in the output signal. Due to the high resistance in the pull-up net, the cell is not able to charge the output load in time before the off-path signal arrives. Hence, the larger the fault size, the flatter the output level remains. Again, this impact could be observed in both of the simulations.

Similarly, the behavior of capacitive faults at the output load have been investigated for varying sizes as depicted in Fig. 5. These faults affect the delay of both rising and falling transitions at the same time, but in contrast to the resistive faults they show no influence on the stationary voltage calculated during simulation. Thus, for larger simulation intervals, the outputs achieve full signal strength. However, in logic simulation similar capacitive faults need to be expressed as small delay faults that affect both rising and falling edges of the pulse simultaneously [24]. Hence this will sustain the pulse, as both the ramping behavior and the corresponding pulse filtering effect cannot be reflected at logic level appropriately.

B. Runtime

For evaluation of the performance, simulation runtimes are measured and compared. In the experiments we constrained the fault sets of each circuit to 10,000 transistor locations that have been chosen randomly for fault injection, since out of



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. 4. Behavior of a resistive-open transistor fault in a) NMOS- and b) PMOS-transistors of a NOR-cell compared to electrical level simulation.

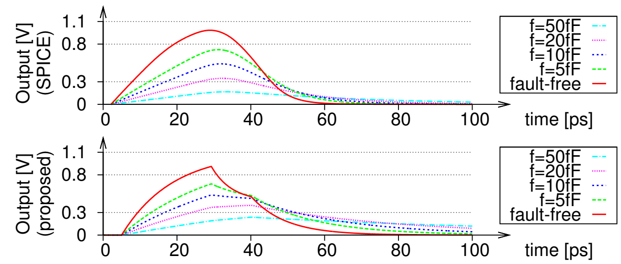


Fig. 5. Behavior of a capacitive fault at the NOR-cell output in comparison.

all possible defects in a circuit, a large portion can in general be handled at logic level [20]. However, in order to prove the scalability, we avoid further comparisons with SPICE due to the drastic computational complexity of its simulation. Instead, a commercial event-based logic-level timing simulator is used as a reference.

Table I provides an overview of the investigated circuits and simulation results. The first three columns contain the names of the circuit, their number of cells and the number of stimuli pairs generated by the ATPG tool. Column 4 shows the number of remaining fault groups after grouping the uncollapsed fault set. We define the *grouping efficiency* (*eff.*) as the initial number of faults divided by the number of obtained fault groups for parallel injection. Hence, the efficiency provides the estimated speedup over a naïve serial simulation of the faults, which is reported in column 5. In all cases, the grouping was finished after a few seconds which is a negligible effort compared to a serial simulation. As shown, the use of fault groups allows for significant reduction of the simulation overhead. Even in case of p469k with the small efficiency, almost 15% of the initial total fault simulation runs are saved.

The last three columns present the runtime performance of the proposed fault simulation. For an unbiased evaluation, the simulation runtimes are compared only for a single fault-free simulation run (Col. 6–7). As shown, the proposed simulator reduces the runtimes from several hours to few minutes despite the more precise abstraction. Finally, the last column contains the time for evaluating all 10,000 faults by using injection of fault groups. Compared to the fault-free simulation run, slight improvements in the average runtime per fault group are achieved, since the simulation of each group is the repeated processing of the same stimuli set that is cached on the GPU. Under consideration of the fault grouping efficiency, the cumulative speedup of the simulation exceeds indeed three orders of magnitude over naïve serial logic level simulation.

TABLE I
SIMULATION OF 10,000 FAULTS AT RANDOM LOCATIONS.

Circuit(1)	Cells(2)	Pattern-Pairs(3)	Fault-Groups(4)	*eff.*(5)	Fault-Free Simulation Logic-Level(6)	Fault-Free Simulation Proposed(7)	Fault-Sim.(8)
s38417	15.6k	348	390	25.6	4.94s	383ms	50.42s
s38584	19.9k	563	328	30.5	9.90s	484ms	1:17m
b17	39.8k	2135	1166	8.6	1:22m	2.40s	40:07m
b19	236.9k	4651	485	20.6	31:04m	34.98s	4:32h
p35k	42.9k	4096	5325	1.9	2:20m	4.74s	6:05h
p89k	88.4k	2460	490	20.4	2:41m	5.03s	36:27m
p141k	156.3k	2043	930	10.8	6:04m	9.55s	2:10h
p378k	366.3k	200	26	384.6	4:15m	4.28s	1:13m
p469k	103.4k	347	8589	1.2	2:50m	4.59s	9:52h
p951k	893.7k	7063	55	181.8	2:43h	2:46m	2:33h
p1522k	949.0k	17980	222	45.0	7:57h	7:22m	27:16h
p2927k	1.48M	22107	67	149.3	17:37h	14:24m	17:03h
p3881k	3.15M	12092	102	98.0	23:41h	32:15m	54:16h

VI. CONCLUSION

This work presented a novel approach for fast and accurate transistor-level fault simulation on data-parallel GPU architectures. The fault simulator allows for modeling and evaluation of parametric and parasitic faults in complex standard cells and supports all major timing-related effects found in CMOS technology, such as pattern-dependent delays, individual transition ramps and pulse filtering. Dimensions of parallelism from cells, stimuli and faults are utilized in order to attain high simulation-throughput for acceleration. Runtime results have shown cumulative speedups of more than three orders of magnitude compared to naïve serial simulation at logic-level, hence outperforming higher-level approaches. This significant speedup as well as the additional gain in modeling detail thus enable, for the first time, accurate simulation of faults at transistor-level even for million-gate designs.

ACKNOWLEDGMENT

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

REFERENCES

[1] 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.

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

[3] J. C. Li, C.-W. Tseng, and E. J. McCluskey, "Testing for Resistive Opens and Stuck Opens," in *Proc. Int'l Test Conf. (ITC)*, Oct. 2001, pp. 1049–1058, Paper 38.2.

[4] 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.

[5] F. Hapke, W. Redemund, A. Glowatz *et al.*, "Cell-Aware Test," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 9, pp. 1396–1409, Sep. 2014.

[6] A. Czutro, N. Houarche, P. Engelke *et al.*, "A Simulator of Small-Delay Faults Caused by Resistive-Open Defects," in *Proc. 13th European Test Symp. (ETS)*, May 2008, pp. 113–118.

[7] 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, Paper 19.2.

[8] J. D. Owens, M. Houston, D. Luebke *et al.*, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] S. Holst, M. E. Imhof, and H.-J. Wunderlich, "High-Throughput Logic Timing Simulation on GPGPUs," *ACM Trans. on Design Automation of Electronic Systems*, vol. 20, no. 3, pp. 1–22, Article 37, Jun. 2015.

[15] 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.

[16] 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, Paper 4C–6s.

[17] L. Han, X. Zhao, and Z. Feng, "TinySPICE: A Parallel SPICE Simulator on GPU for Massively Repeated Small Circuit Simulations," in *Proc. ACM/EDAC/IEEE 50th Design Automation Conf. (DAC)*, May 2013, pp. 1–8, Article 89.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] S. Hillebrecht, I. Polian, P. Engelke *et al.*, "Extraction, Simulation and Test Generation for Interconnect Open Defects Based on Enhanced Aggressor-Victim Model," in *Proc. IEEE Int'l Test Conf. (ITC)*, Oct. 2008, pp. 1–10, Paper 33.3.

[24] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *Journal of Applied Physics*, vol. 19, no. 1, pp. 55–63, Jan. 1948.