Enhancing Speedups for FPGA Accelerated SPICE through Frequency Scaling and Precision Reduction

Lim Hui Hui, Nachiket Kapre
School of Computer Engineering,
Nanyang Technological University,
Singapore 639798
nachiket@ieee.org

***Abstract*—**
Frequency scaling and precision reduction optimization of an FPGA accelerated SPICE circuit simulator can enhance performance by 1.5× while lowering implementation cost by 15–20%. This is possible due the inherent fault tolerant capabilities of SPICE that can naturally drive simulator convergence even in presence of arithmetic errors due to frequency scaling and precision reduction. We quantify the impact of these transformations on SPICE by analyzing the resulting convergence residue and runtime. To explain the impact of our optimizations, we develop an empirical error model derived from in-situ frequency scaling experiments and build analytical models of rounding and truncation errors using Gappa-based numerical analysis. Across a range of benchmark SPICE circuits, we are able to tolerate to bit-level fault rates of 10^{-4} (frequency scaling) and manage up to 8-bit loss in least-significant digits (precision reduction) without compromising SPICE convergence quality while delivering speedups.

I. Introduction

The SPICE circuit simulator [1] is a well-known industrial tool for analysis and design of circuits. It is also notoriously hard to parallelize and is a challenge problem for computer architects and parallel programmers. A SPICE simulation is an iterative numerical computation that consists of a device model evaluation phase, a tricky matrix factorization phase and a sequential control phase that drives the numerical integration and linearization phases. We shown a high-level visualization of the SPICE simulator in Figure 1 and describe it in more detail later in Section II. Hardware-accelerated SPICE solvers using FPGAs [2], [3] and GPUs have been show to deliver 3–10× speedup over conventional CPU-based solvers by exploiting spatial parallelism of the FPGA fabric. While these speedups are promising, they are not easily scalable due to limited parallelism and cost of FPGA hardware. In this paper, we investigate alternative approaches for improving performance of the SPICE accelerator by frequency scaling (runtime) or precision adaptation (compile time).

Despite these limits of parallelism, we observe that iterative numerical computations such as SPICE are designed to tolerate the impact of finite precision arithmetic hardware on the accuracy of the solution of the computation [4]. These algorithms use non bit-exact formulations of convergence and correctness, and are *guided* towards the final solution through periodic measurements of error (degree of incorrectness of the solution)

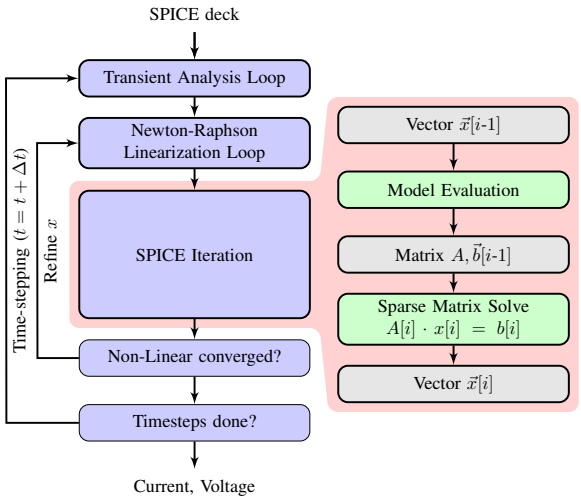


Fig. 1: SPICE Simulator Flow

and proportional corrective actions. For example, the Newton Raphson algorithm for finding the root of a polynomial, $f(x)$=0, will incrementally revise x in each iteration based on the error residue at any given iteration $f(x_{iter})$ until the convergence condition $f(x_{converge}) < \epsilon$ is met at a particular iteration. In the same spirit, the SPICE [1] circuit simulator contains two iterative loops (1) linearization of non-linear elements to construct Jacobian matrix, and (2) Backward-Euler numerical integration of time-varying elements. We exploit this observation to consider the effect of the following two optimizations to enhance speedups and lower the cost of FPGA accelerated SPICE solvers.

- **Precision Reduction**: SPICE (`spice3f5`) uses double-precision floating-point arithmetic but the voltage, current and conductance physical quantities involved cover a small dynamic range of possible values. For FPGA accelerators, we can implement numerical computations using fewer bits with custom precision implementations. This change in precision introduces a truncation error in each step of the computation. *We hypothesize that truncation errors may not matter if they accumulate to sufficiently small levels ($< \epsilon$) and the linearization and integration loops can self-correct to converge to the right value.*

- **Frequency Scaling**: Physical FPGA hardware is typically designed to operate with aggressive safety margins to guarantee correct behavior. However, it is possible to do *better-than-worst-case* by frequency scaling the circuit for performance (or undervolting the power supply for energy savings). However, if operated beyond a threshold, we start to introduce timing faults in the results due to critical path timing violations. *Again, we expect the SPICE iterative loops to naturally measure the degree of incorrectness* (e.g. *residue calculations in SPICE) and compensate for them to an extent.*

We believe the techniques presented in this paper are increasingly important as we scale to smaller transistor geometries that may result in chips that are defective at fabrication time, and faulty at runtime due to a combination of factors such as process variation, aging and other statistical physical effects. If we can show how to continue to use faulty silicon for meaningful computation, we can deliver both performance and cost reductions; both in terms of yield and operating energy.

We summarize the key contributions of this paper in the following list:

- Development of a framework for SPICE (and KLU sparse LU solver [5]) to model the impact of errors due to frequency scaling and precision reduction on SPICE convergence and iteration count.
- Characterization of the properties of double-precision floating-point operators in the presence of truncation errors (Gappa-based analysis) and timing violations (in-situ FPGA experiments).
- Quantification of speedups and resource savings of a hardware-accelerated SPICE simulator across a variety of circuit benchmarks obtained from academic projects and industrial packages.

II. Background

The SPICE [1] circuit simulator (`spice3f5`) belongs to a broad class of iterative numerical algorithms with a quantifiable (non bit-exact) measure of correctness. SPICE simulates the transient analog behavior of a circuit represented using non-linear differential equations. SPICE solves these circuit equations by computing small-signal linear operating-point approximations for the non-linear (*e.g.* diodes, transistors) and time-varying elements (*e.g.* capacitors, inductors) until convergence and termination (see Figure 1. The specific numerical algorithms used are Newton-Raphson (linearization inner loop) and Backward-Euler (numerical integration outer loop). The linearized system of equations is represented as a solution of $A\vec{x} = \vec{b}$ handled in the **Matrix-Solve** phase, where A is the matrix of circuit conductances, $\vec{b}$ is the vector of known currents and voltage quantities and $\vec{x}$ is the vector of unknown voltages and branch currents. The simulator repeatedly updates entries in A and $\vec{b}$ from the device equations that describe device transconductance (*e.g.*, Ohm's law for resistors, transistor I-V characteristics) in the **Model-Evaluation** phase. In each iteration, SPICE tracks the linear solver residue $(\vec{b} - A\vec{x})$ and Δ changes in $\vec{x}$ *w.r.t* previous iteration (`spice3f5`). For

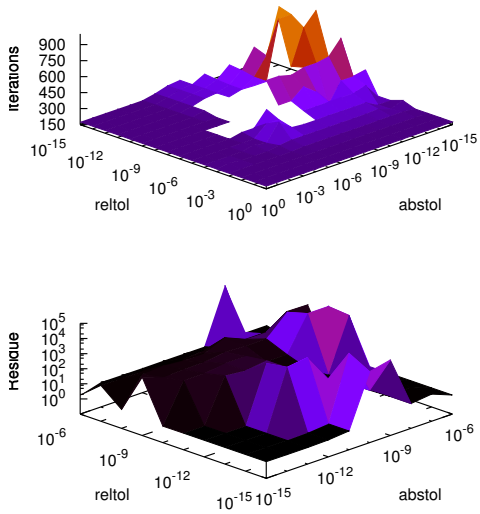


Fig. 2: Impact of changing `abstol` and `reltol` tolerances on iteration count and residue of a SPICE circuit simulation for a bsim3 `inverter`

SPICE netlists with a large number of non-linear devices, as much as 90% (avg. 55%) of runtime is devoted to the Model-Evaluation phase. This is because runtime we must evaluated each non-linear device individually and total time is linearly proportional to the number of non-linear devices in the design. For circuits with a large number of resistors and capacitors that are often encountered in parasitic simulations results in very large and very sparse matrices. In these circumstances, the Matrix-Solve phase accounts for as much as 70% (avg 38%) of SPICE simulation time. The Iteration controller is the smallest portion of total time (less than 10%).

A. Preliminary Error Analysis

We conducted preliminary experiments to understand the impact of varying `abstol` (absolute tolerance) and `reltol` (relative tolerance) parameters of SPICE which control convergence and termination of the simulator as shown in Figure 2. You would expect tighter tolerances to deliver more accurate results with a increased runtime trade-off but that does not always hold. Furthermore, the shape of the iteration surface peaks gradually providing us sufficient freedom to consider alternative implementations as long as the designer minimum expectation of accuracy is satisfied. It is clear that this affects two key metrics of the simulator: number of iterations (runtime) and residue (measure of accuracy). Instead of chasing bit-exact evaluations, we can control the tolerances to reflect frequency scaling and precision reduction in the design to deliver satisfactory simulation results with low convergence residue.

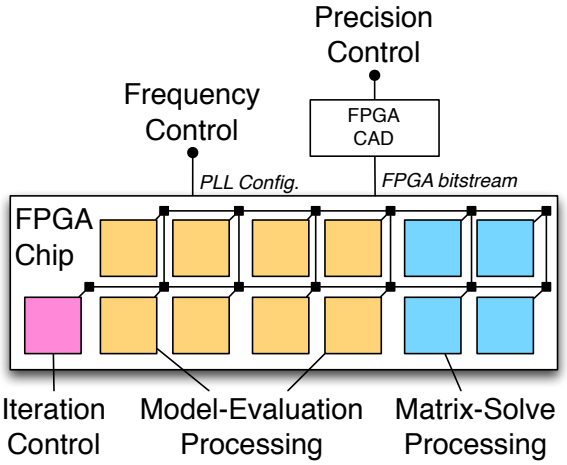


Fig. 3: High-Level Design of an FPGA Accelerator for SPICE Solver

B. Accelerator

In Figure 3, we sketch a high-level diagram of the FPGA SPICE accelerator based on an earlier design [2]. We use a design with 8 floating-point operators for the `bsim4` Model Evaluation, 4 floating-point operators for the Sparse Matrix-Solve and a generic state-machine for the Iteration Controller phases of SPICE. This design can accelerate SPICE by 3–10× (depending on benchmark) when compared to a sequential Intel CPU. As bulk of compute time is spent on Model Evaluation, we devote a larger fraction of the chip for a custom VLIW accelerator [6] for this phase. Matrix Solve step is the second most dominant phase and is also allocated a significant portion of the chip to implement a Token dataflow design [7]. The sequential controller runs on a VLIW soft processor [8] with low area footprint to conserve resources. The FPGA is managed from a host CPU that programs the FPGA with the accelerator bitstream and loads the per-circuit SPICE parameters to the onchip memory at runtime for different simulations. For our experiments, we can control frequency with the on-board PLL at runtime and modify the operator precision using a VHDL code generator that requires resynthesizing the FPGA bitstream at compile time.

III. Frequency Scaling and Precision Reduction Framework

As discussed in the previous section, the SPICE simulation is an iterative algorithm. We can analyze the impact of frequency scaling and precision reduction by studying a single iteration in greater detail and understanding how it composes with other iterations. We observe that Model-Evaluation and Sparse Matrix-Solve phases are responsible for contributing to both runtime (≈90% as indicated earlier in Section II) and numerical stability (*e.g.* convergence). Furthermore, the aggregate memory usage and bandwidth of the program is

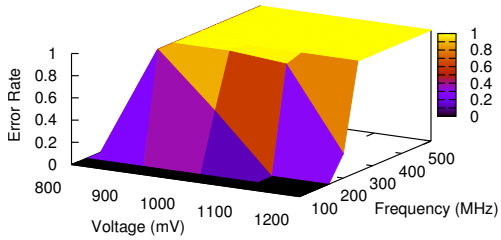


Fig. 4: Voltage-Frequency scaling of a Double-Precision Floating-Point Multiplier on a DE0 Nano FPGA board rated at 1.1V and 100MHz

dominated by the core data-structures required to hold matrix A and the factorized matrices L and U. We would expect, timing errors in the **Model Evaluation** phase will propagate and corrupt values in matrix A while the those in the **Sparse Matrix Solve** phase will corrupt data stored in matrices L and U.

A. Frequency Scaling

It has been shown that we can operate digital circuits at beyond worst-case specifications of clock frequency and voltage up to a limited extent without any loss in correctness [9]. We conduct an experiment on the Terasic DE0 Nano FPGA board operating at a nominal voltage of 1.2V and 100MHz for a double-precision floating-point multiplier with gaussian-distributed input vectors. The VHDL for this multiplier is generated using Flopoco library and wrapped in a suitable test harness to facilitate this experiment. In Figure 4, we show the impact of observing one-or-more bit errors in the output of this multiplier. The multiplier works correctly across a range of voltage-frequency combinations going as low as 800mV and as high as 180MHz without error. Strategic use of frequency scaling (or voltage scaling) can allow the circuit to operate faster (or lower energy) with a low-enough error rate.

In Figure 5, we show the impact of frequency scaling on bit-wise absolute error at the output of a double-precision floating-point multiplier. The safe operating frequency is 100MHz but we can see that different bits (ordered by significance) start failing gracefully at different frequencies. As expected, the MSB-side bits (most significant bits) fail first (*e.g.* bit 60 fails at 160MHz) while the LSB-side bits (least significant bits) fail later (bit 24 fails only at 250MHz). We expect the multiplier bits to exhibit varying bit-specific failure profiles depending on the path lengths of the logic circuit terminating at those respective bits. The exact nature of failure will depend on a combination of overclock frequency as well as the actual values of data being processed and activating specific paths within the circuit.

B. Precision Reduction

Cheap (low area, low energy) implementations of hardware are possible through the use of lower-precision arithmetic

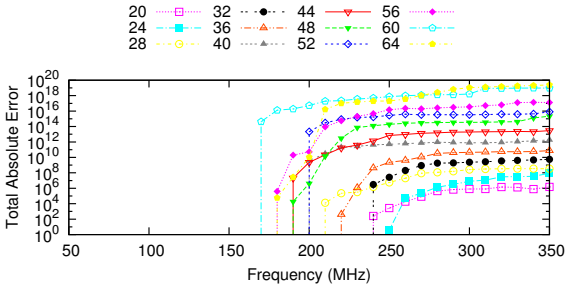


Fig. 5: Bit-wise absolute error due to frequency scaling of a Double-Precision Floating-Point Multiplier on a DE0 Nano FPGA board rated at 100MHz

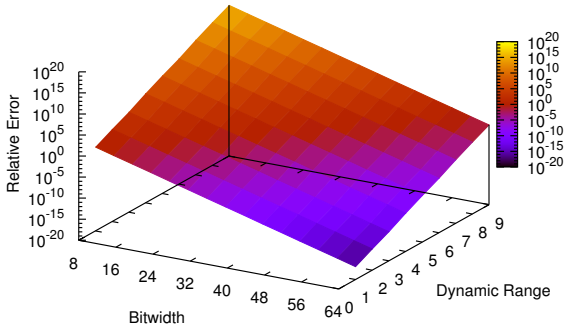


Fig. 6: Absolute error of a Custom-precision Multiplier vs. arbitrary precision baseline using Gappa

(*e.g.* single-precision multiplier is smaller than a double-precision multiplier). If the dynamic range of the inputs to the arithmetic operators is far smaller than the full range made possible by double-precision encoding, we may be able to tune the number of bits to be just right for the numerical computation. For example, the voltage quantities being handled is likely to be in the μV–mV range and currents may be in the pA–μA range (as appropriate for the kind of circuit being simulated). We use Gappa [10] to analyze the relative error properties of the multiplier under different dynamic input range and precision combinations. Gappa is a formal analysis tool that evaluates error properties of numerical computations backed by proofs. Increasing precision reduces the error in the computation while this error will increase if the dynamic range of the input is larger. Limited operating dynamic range of numerical variables in SPICE simulations can allow safe reduction of operating precision without compromising numerical stability and quality of the solution. This reduces fault susceptibility by increasing the maximum operating frequency and providing a smaller circuit (fewer transistors) that can fail.

In Figure 6, we use Gappa to analyze the relative error properties of the multiplier under different dynamic input range and precision combinations. Increasing precision reduces the error in the computation while this error will increase if the dynamic range of the input is larger.

C. Understanding Fault Injection

To understand the impact of frequency-scaling and precision reduction on the SPICE error properties, we develop a simple software-based modeling framework that modifies the key data structures in SPICE *e.g.* A, L, U based on our analysis parameters.

- We model timing failures due to frequency scaling using probabilistic error injection based on parameters such as error injection probability ($10^{-15} \leq p_f \leq 10^{0}$) and the number of bits suffering from timing fault ($1 \leq b_{oc} \leq 3$). These are derived from real-world measurements as indicated in Section III-A. We use an error model based on bit-flipping which is implemented as 64-bit wide XOR masks in the top $b_{msb} \leq 20$ bits away from the most significant bit (see Figure 5 and accompanying explanation). A similar idea has been previously explored in [11]. Timing fault detection and correction techniques have been discussed in [12] and we use these techniques to quantify our fault rates.
- For precision reduction, we simply truncate the $b_{tr} \leq 10$ bits from the least significant. To implement precision reduction, we use a stuck-at-zero error model to blank out relevant bits in *every* item of data.

In Figure 7, we show the impact of frequency scaling and precision reduction on the error residue ($\vec{b} - A\vec{x}$) for different timesteps (iterations). Typically we want to see error residue to be as low as possible for stable convergent operation. For the case without any error injection, the residue of each timestep (iteration) is low enough for correct convergence. The value of residue changes per timestep as the input matrix A will vary due to changes in SPICE simulation conditions. This explains the increase in residue between 40–120 iterations while still delivering the expected final result. As we can see, truncating the least significant bits extends simulation time by requiring the simulator to work harder (longer) to achieve the same result. This results in a "dilation" effect on the convergence which requires additional timesteps (iterations) to converge to the final correct result. On the other hand, frequency scaling affects the most significant bits and introduces sudden and abrupt changes in the error residue. However, the inherent recovery mechanisms in the iterative SPICE simulator are capable of tolerating a limited number of such events and still provide the same solution while requiring additional iterations. As mentioned earlier, this recovery is possible due to the fact that the residue values are measured in each timestep (iteration) and used to *guide* the next simulation step towards convergence. A large value of residue will result in a suitably proportional compensation operation in the algorithm. In both cases, the resulting implementation cost (hardware required to run SPICE) and runtime trade-off (more iterations at over-clocked frequency) will determine efficacy of this approach.

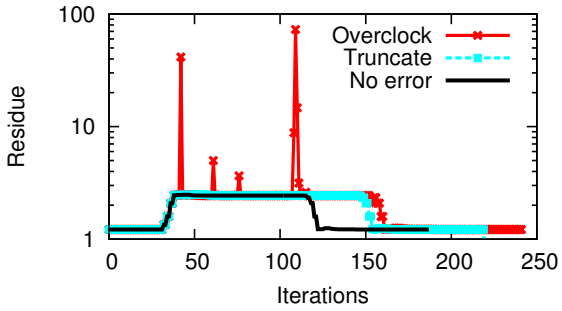


Fig. 7: Comparing frequency scaling (1 bit-error near MSB, $p_f = 1e^{-2}$) and precision reduction (eight LSBs) on the `inverter`

Circuit	Matrix Size	Sparsity (%)	Ops.
Simucad [15]			
mux8	42	15	626
ringosc	104	6.5	1.6K
dac	654	1.6	23.6K
ram2k	4875	0.3	1.0M
Clocktrees [13]			
r4k	39948	0.01	515.5K
Wave-pipelined Interconnect [14]			
20stages	11225	0.06	219.2K
40stages	22405	0.03	395.7K

TABLE I: Circuit Simulation Benchmark Matrices

IV. Methodology

As shown earlier in Figure 3, we use an FPGA implementation that spatially implements all three phases of SPICE in hardware. For our experimental purposes we use the `spice3f5` open-source package but replace the Sparse 1.3 package with the improved KLU matrix solver that is optimized for circuit simulation. We use a diverse set of benchmark circuit-simulation matrices, listed in Table I, including RAM netlists (Simucad), clocktrees [13] (University of Michigan), and wave-pipelined circuits [14] (UBC). Our framework sweep of the design space across all benchmark circuits and model parameters to measure the number of SPICE iterations, error residue as well as extraction of the raw voltage or current waveforms for inspection and analysis. In our experiments, we analyze two key metrics of the SPICE circuit simulator in presence of faults: **iteration count** growth and **error residue** variation. All experiments are conducted at the default/nominal `abstol`=10^{-12} and a `reltol`=10^{-3} so solution quality is same across all cases.

Fault-injection experiments used to generate results in Figure 4 are based on double-precision multiplier and adder circuits generated using Flopoco [16] v2.5.0. We use Altera Quartus 12.0 toolflow running Ubuntu 12.04.3 64-bit Linux to generate bitstreams. We generate random input distributions for the hardware using the GNU Scientific Library (`libgsl` v1.16). All measurements are run on the Terasic DE0 Nano FPGA board supported by a TTi PL303QMD-P PSU unit

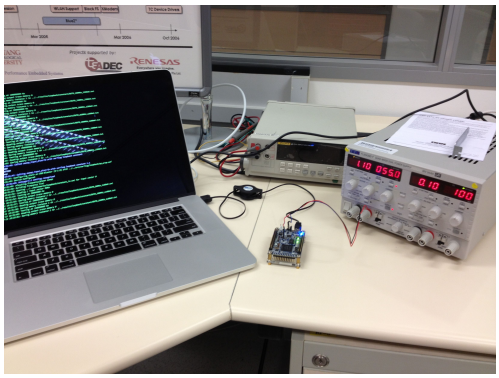


Fig. 8: Experimental Platform with the Terasic DE0 Nano FPGA Board and TTi PL303QMD-P PSU controlled using a an Ubuntu 12.04 64-bit VM

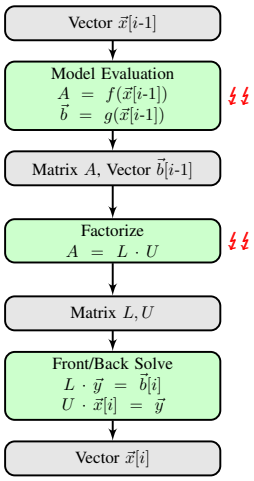


Fig. 9: Error Injection Flow

that supports programmable voltage control. We show our experimental setup in Figure 8. The input data supplied to the hardware are extracted from the SPICE simulation runs.

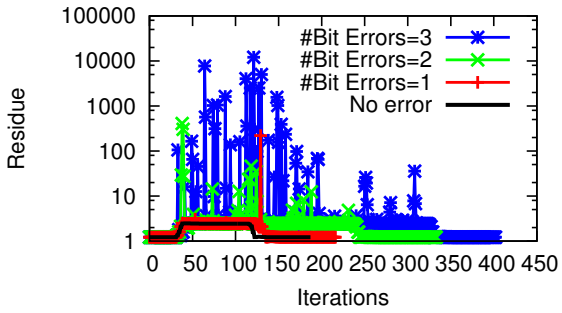
The fault injection framework conducts a sweep of the design space across all benchmark circuits, under the two error models for overclocking and truncation. For each error model we further do a sweep of the model parameter space and measure the number of SPICE iterations, error residue as well as extraction of the raw voltage or current waveforms for inspection and analysis. As shown in Figure 9, both these models focus on (1) the Model Evaluation phase through faults introduced in matrix A and vector b as well as (2) the Sparse Matrix Solve phase through faults injected in factored matrices L and U.

V. Evaluation

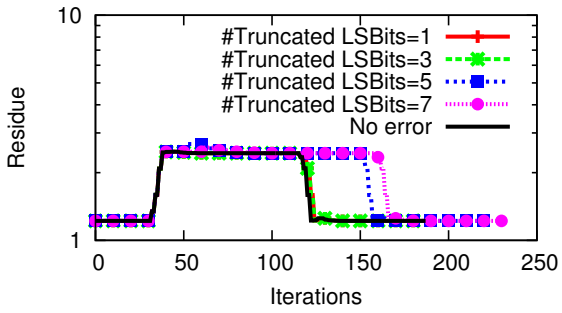
We now discuss the results of our fault analysis and quantify overall speedups and improvements possible with our framework.

A. Understanding Impact of Frequency Scaling and Precision Reduction on Convergence

In Figure 10(a), we show the impact of overclocking (random bit-flipping in b_{oc} positions with probability p_f) the SPICE simulation of a simple `inverter` circuit. As expected, we observe large transient increases in residue in certain iterations. However, the iterative numerical algorithms used in SPICE allow a graceful recovery driving down the residue to nominal levels at the expense of increased iterations. Furthermore, for the benchmark set we considered, we could only tolerate timing faults as high as 3 bit errors as MSB-side errors can be critical. We observe a correlated increase in recovery period as we inject more bit errors into the simulation.



(a) Multi-bit timing failures in the `inverter` circuit for a p_f=10^{-2}

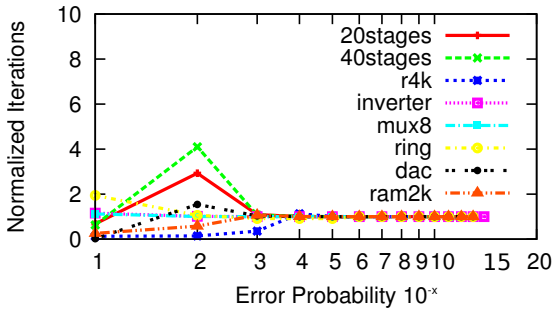


(b) LSB truncation in the `inverter` circuit

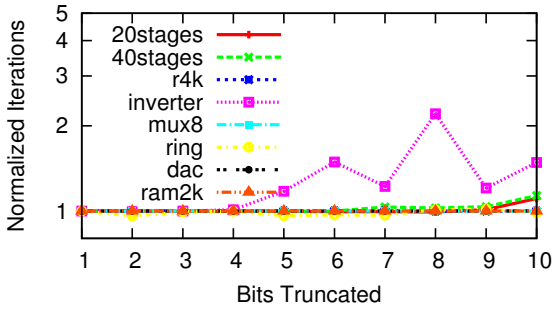
Fig. 10: Overclocking and truncation of `inverter`

In Figure 10(b), we show the effect of varying truncation width (b_{tr}) on residue and iteration count of an `inverter` simulation. Unlike the overclocking scenario, truncation simply trades off runtime (iterations) to achieve same result. Different benchmarks allow truncation to different extent before suffering from simulation failure. As we drop more bits, the simulation grows increasingly slower in a mostly monotonic fashion (exception `inverter` behavior in Figure 11(b)).

In Figure 11(a), we show the effect of increasing timing failure probability of single-bit errors on increase in SPICE iteration count. As we increase fault probability, we observe no change in the resulting iteration count up to an error rate



(a) Impact of Frequency Scaling



(b) Impact of Precision Reduction

Fig. 11: Impact of Optimizations on SPICE iterations

of $p_f \approx 10^{-4}$. Above this error rate ($10^{-4} \leq p_f \leq 10^{-2}$) we start noticing a initial increase in iteration count as SPICE strives to converge until it eventual abandons attempts and quits at larger error rates.

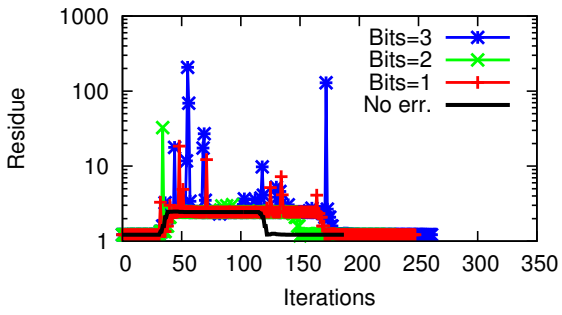
In Figure 11(b), we plot the effect of varying the number of truncated bits on the number of iterations required for convergence across a range of SPICE circuit simulations. As we see, there is no perceptible change in iteration if we drop $\leq$ 4 bits. Above 4 bits, the `inverter` iteration cost grows $\approx$2$\times$ the nominal but in most other larger benchmarks, the growth in iteration count is merely 10–20%. All simulations completed with correct results below 10 bits truncation.

B. Adaptive Overclocking

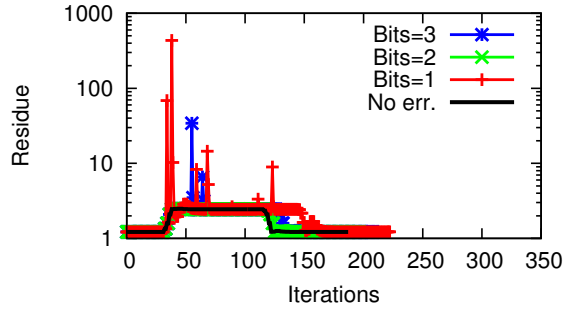
In Figure 12(a) and Figure 12(b), we consider the possibility of isolating the impact of fault injection on the Model Evaluation and Sparse Matrix Solve phases independently. We observe faster error recovery in the Model Evaluation phase than the Sparse Matrix Solve phase *e.g.* for inverter, with a p_f=10^{-2}, we observe a larger increase in overall iteration count for injecting errors in the Sparse Matrix Solve phase ($\approx$260 iterations) compared to Model Evaluation ($\approx$230 iterations).

C. Overall Acceleration

From the timing failure models shown in Figure 4 and borrowing the area models developed in [2], we can esti-



(a) Model-Evaluation Timing failures



(b) Sparse Matrix Solve Timing failures

Fig. 12: Adaptive Overclocking, `inverter`, p_f=10^{-2}

mate overall speedup of SPICE simulations. The overclocking margins possible on an FPGA are quite large 2× (≈80MHz for multiply) when compared against CAD timing analysis. However, if we compare with the board-specific real-world measurements the overhead gets reduced to ≈50MHz. As a result, the increase in runtime due to more iterations breaks even with overclocking at around p_f=10^{-2} (vs. CAD margins) and at around p_f=10^{-4} (vs. board-specific margins). *e.g.* p_f=10^{-6} enables speedup of 1.9× (vs. CAD margins) or 1.1× (vs. board-specific margins).

When considering precision reduction, we observe near-linear decrease in hardware area and a proportional increase in compute density of the FPGA SPICE accelerator. For our benchmarks, we achieve a 1.2× improvement for a 8-bit reduction in precision (some cases tolerate as many as 10 bits). This can be translated into either (1) use of cheaper FPGAs, or (2) re-allocate unused logic to the Model Evaluation and Matrix Solve phases on the same FPGA.

VI. Conclusions

Iterative numerical computation such as those used in the SPICE circuit simulator are naturally resilient to faults. In this paper, we show how to accelerate SPICE by an additional 1.5× on top of the 3–10× speedup already possible over modern CPUs. We develop a fault injection framework that allows us to model the effect of timing faults due to overclocking and truncation errors due to precision reduction of the numerical types used in SPICE. We observe that SPICE is resistant to single-bit overclocking errors up to $p_f \leq 10^{-4}$ with a graceful degradation in runtime at higher error rates. Across all our chosen benchmarks, we notice that SPICE can only safely and reliably recover from three bit timing errors. The effect of truncation on SPICE suggested a near-lossless behavior under identical tolerance condition even when throwing away as many as eight least significant bits.

In the future, we intend to include modeling for different bit-specific timing failure probabilities as well as input data-driven correlations to timing faults in our fault injection framework. Additionally we will consider opportunity for dynamic adaptive precision selection that could help reduce operating energy margins through clock gating bit-sliced portions of the datapath. We also seek to expand the scope of applications that can benefit from such improvements.

References

[1] L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," Ph.D. dissertation, University of California Berkeley, 1975.

[2] N. Kapre and A. DeHon, "SPICE2: Spatial Processors Interconnected for Concurrent Execution for Accelerating the SPICE Circuit Simulator Using an FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 1, pp. 9–22, 2012.

[3] N. Kapre, "SPICE^2–A Spatial Parallel Architecture for Accelerating the SPICE Circuit Simulator," Ph.D. dissertation, California Institute of Technology, Pasadena, Pasadena, Sep. 2010.

[4] G. H. Golub and C. F. Van Loan, *Matrix Computations*, ser. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press.

[5] T. A. Davis and E. Palamadai Natarajan, "Algorithm 907: KLU, A Direct Sparse Solver for Circuit Simulation Problems," *ACM Transactions on Mathematical Software*, vol. 37, no. 3, pp. 36:1–36:17, 2010.

[6] N. Kapre and A. DeHon, "Accelerating SPICE Model-Evaluation using FPGAs," in *FCCM '09: Proceedings of the 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*. IEEE Computer Society, Mar. 2009, pp. 1–8.

[7] ——, "Parallelizing Sparse Matrix Solve for SPICE circuit simulation using FPGAs," in *Field-Programmable Technology*, Jan. 2010.

[8] ——, "VLIW-SCORE: Beyond C for Sequential Control of SPICE FPGA Acceleration," in *Field-Programmable Technology (FPT), 2011 International Conference on*, Dec. 2011, pp. 1–9.

[9] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *36th International Symposium on Microarchitecture*. IEEE Comput. Soc, 2003, pp. 7–18.

[10] S. Boldo, J.-C. Filliâtre, and G. Melquiond, "Combining Coq and Gappa for Certifying Floating-Point Programs." Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 59–74.

[11] L. Zussa, J.-M. Dutertre, J. Clédière, B. Robisson, A. Tria, and Others, "Investigation of timing constraints violation as a fault injection means," *27th Conference on Design of Circuits and Integrated Systems*, 2012.

[12] E. Stott, J. M. Levine, P. Y. K. Cheung, and N. Kapre, "Timing Fault Detection in FPGA-based Circuits," in *FCCM '14: Proceedings of the 2014 22nd IEEE Symposium on Field Programmable Custom Computing Machines*, Mar. 2014, pp. 1–4.

[13] C. N. Sze, P. Restle, G. J. Nam, and C. Alpert, "ISPD2009 clock network synthesis contest," in *International Symposium on Physical Design*. New York, New York, USA: ACM Press, 2009, p. 149.

[14] P. Teehan, G. G. F. Lemieux, and M. R. Greenstreet, "Towards reliable 5Gbps wave-pipelined and 3Gbps surfing interconnect in 65nm FPGAs," in *International Symposium on Field Programmable Gate Arrays*. ACM, 2009, pp. 43–52.

[15] S. Silvaco, "BSIM4, BSIM4 and PSP benchmarks from Simucad," 4701 Patrick Henry Drive, Bldg 2 Santa Clara, CA 95054. USA. www.simucad.com, Tech. Rep., 2007.

[16] F. de Dinechin, C. Klein, and B. Pasca, "Generating high-performance custom floating-point pipelines," in *International Conference on Field Programmable Logic and Applications*. IEEE, 2009, pp. 59–64.