Automated di/dt Stressmark Generation for Microprocessor Power Distribution Networks

Youngtaek Kim and Lizy Kurian John
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX
young.kim@utexas.edu ljohn@ece.utexas.edu

***Abstract*— In this paper, we propose a method for automated di/dt stressmark generation to test maximum voltage droop in a microprocessor power distribution network. The di/dt stressmark is an instruction sequence which draws periodic high and low current pulses that maximize voltage fluctuations including voltage droops. In order to automate di/dt stressmark generation, we devise a code generator with the ability to control instruction sequencing, register assignments, and dependencies. Our framework uses a Genetic Algorithm in scheduling and optimizing candidate instruction sequences to create a maximum voltage droop. The simulation results show that our automatically generated di/dt stressmarks achieved more than 40% average increase in voltage droop compared to hand-coded di/dt stressmarks and typical benchmarks in experiments covering three micro-processor architectures and five power distribution network (PDN) models. Additionally, our method considers all the units in a microprocessor, as opposed to a previous ILP scheduling method that handles only execution units.**

Keywords - voltage droop; microprocessor power distribution network; di/dt stressmark; system-level power-aware design

I. INTRODUCTION

Reliability has become an important consideration in computer system design because of increases in complexity, decreases in supply and threshold voltages, and increases in frequency. Errors due to di/dt noise are an important reliability issues, caused by inductance in the power distribution network (PDN). Periodic, large current load variations may cause di/dt noise. Identifying di/dt noise effects on a microprocessor is very important in preventing voltage emergencies, which may cause timing violations and/or improper behavior of a component [5].

In a microprocessor, the supply voltage is provided through a PDN, which can be represented as a distributed RLC circuit with resonance frequencies (Fig. 1). Varying current (di/dt) can cause fluctuations of the supply voltage that are proportional to the inductance (L) of the circuit ($v = L \cdot di/dt$). Voltage droop is maximized if the periodic, large current variation occurs at the resonance frequency of the PDN. A resonance frequency in the mid-frequency (50~200MHz) range is the most significant [1]. Significant supply voltage droop may cause reliability problems in a microprocessor. Low voltage increases the delay of signals, which could affect the timing between two flip-flops in a microprocessor circuit. Also, insufficient voltage could fail to set bit-signals properly and lead to soft errors.

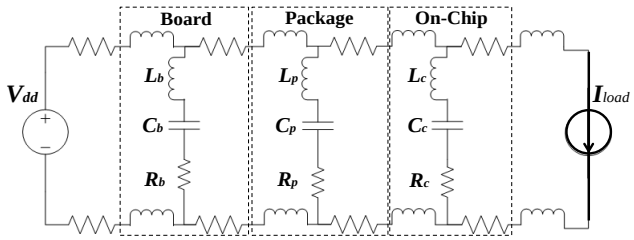

Figure 1. Generalized Lumped Power Distribution Network Model (can be extended to complex, distributed model [2])

To identify the maximum voltage droop caused by di/dt noise, designers can use typical benchmarks. However, most benchmarks, such as SPEC CPU2006, focus on high performance, so they may not generate periodic, high and low current draw under normal condition [5]. Moreover, typical benchmarks require a long simulation time for system-level designs. Therefore, there is a need for a stressmark that causes severe voltage droops in a short simulation time.

On the other hand, in many cases, designers manually generate a di/dt stressmark to test their processor/system. However, the manual generation of a di/dt stressmark is tedious and time-consuming. Designers need to recreate stressmarks whenever an architectural change occurs. In addition, the search space is extremely large, so it is not feasible for designers to manually generate and test every possible combination of parameters, configurations, and instructions to fully utilize a processor/system.

In this paper, we propose an automatic di/dt stressmark generation framework to produce significant voltage droops. We utilize a Genetic Algorithm to generate and optimize candidate di/dt stressmarks. To efficiently explore the large search space, we reduce the number of instructions, devise an instruction structure, and assign registers for scheduling. The results show that the automated di/dt stressmark always induces higher voltage droop than hand-coded di/dt stressmarks and typical benchmarks. Current waveform analysis also demonstrates the effectiveness of our di/dt generation method.

In this paper, we make the following key contributions:

- We propose an automatic framework to generate an effective di/dt stressmark without comprehensive knowledge of a microprocessor system.
- We utilize a Genetic Algorithm to generate a benchmark that creates a maximum voltage droop in a given microprocessor and PDN.
- Our automated framework reduces designers' time to generate a hand-coded di/dt stressmark and/or to simulate typical benchmarks that are possibly irrelevant to inducing maximum voltage droop.
- Our di/dt stressmark generation method can target an individual unit, the whole processor, or even more complex architecture such as multiple processors and GPUs.
- Our di/dt stressmark can be applied to PDN analysis and circuit marginality tests.

We review the related works in Section II and explain our di/dt stressmark generation framework in Section III. We compare our method to a previous method in Section IV and conclude in Section V.

II. Related Work

A significant number of studies on di/dt problems have been conducted. However, most of these studies focused on prevention/avoidance [5][7], mitigation [10], or recovery from the di/dt effect [9]. In contrast, there are three previous works on creation of di/dt stressmarks to maximize voltage droop.

Joseph, Brooks, and Martonosi presented a hand-coded di/dt stressmark in [5]. However, their di/dt stressmark was manually crafted for the given architecture and focused only on memory intensive behavior, such as loads and stores to increase current draw by accessing L1 and L2 data caches.

Ketkar and Chiprout proposed a di/dt stressmark generation methodology using integer linear program (ILP) scheduling [6]. However, they focused only on execution portions of the processor, so it is difficult to apply their technique to the entire system.

Joshi et al. [11] mentioned that high-power and low-power instruction sequences from two different power optimizations can be attached to generate a di/dt stressmark. However, Joshi et al. gave only a suggestion without implementation details or results.

III. Generation of di/dt Stressmark

In this section, we present our di/dt stressmark generation framework. A di/dt stressmark is an instruction sequence which consists of a certain number of instructions that induces extremely high and low current draws in order to make the voltage fluctuate as much as possible within a very short period. To generate and test such a di/dt stressmark, we utilize the following simulation method and optimization algorithm.

A. Current-Voltage Simulation

In here, we describe our basic current-voltage simulation framework. First, an instruction sequence is created as C or assembly code with parameters and constraints such as stressmark size and instruction types. Next, we compile the program code and run it on a system simulator to estimate current draw per cycle in a microprocessor. During the system simulation, all the activities are counted every cycle and converted as power consumption per cycle. To get instantaneous current values, the obtained cycle power numbers are divided by a DC supply voltage. Then, the current trace from the system simulator is fed to the circuit simulator to simulate voltage fluctuation. After collecting the voltage trace, it is analyzed to identify a maximum voltage droop. The instruction set architecture (ISA) in this paper is based on Alpha because it is convenient to use the well-known system and power simulator, SimpleScalar/Wattch [3]. However, our method is not limited to a specific ISA or simulator, and it can be applied to other ISAs such as x86 and SPARC. In our framework, we use the HSPICE simulator, instead of the convolution of an impulse response, because it is more accurate and it is convenient to change RLC values to model a power distribution network efficiently.

B. Reduction of Search Space

In order to explore the instruction scheduling space efficiently, we need to reduce the number of instructions to be considered. The search space is almost impossible to be enumerated with all the different types of opcodes and register combinations. Therefore, this step is necessary before searching the instruction scheduling space to eliminate redundant combinations of instructions and to reduce search time significantly. Each instruction can be categorized into one of a few groups: data type, arithmetic, logic, load/store, bit-level, conditional move, and branch/jump. For data type, we use both integer and floating-point types to utilize the execution units maximally, but only the quad-word (64-bit) type for integer and the double precision type for floating-point are selected to draw large current due to multiple-bit changes. For example, instructions such as add-bytes, add-words, and add-double-words are not used in an instruction sequence. The arithmetic and load/store instructions use different execution units with different latencies, so they are considered individually. For logic, bit-level, conditional move, and branch/jump groups, one instruction can represent other instructions if they use the same execution unit with the same latency such as **cmple** (compare less than or equal) and **cmpeq** (compare equal).

C. Genetic Algorithm for di/dt Stressmark

We applied a Genetic Algorithm (GA) for generating and developing a di/dt stressmark. The Genetic Algorithm is known to be very efficient in solving an optimization problem because it controls the simulation to find a best fitness value for the problem by killing inferior candidates and promoting superior ones. In our framework, initially, random instruction sequences are created, and they are forced to reproduce, mutate, and compete for maximizing voltage droop as the algorithm proceeds.

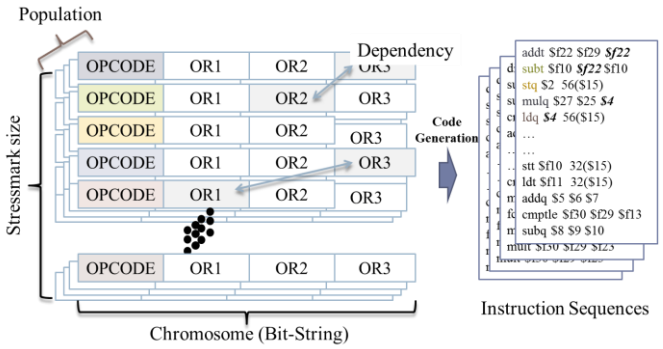

Figure 2. Instruction Sequence Generation for Genetic Algorithm

Fig. 2 depicts instructions, stressmark size, and candidates for the di/dt stressmark in the Genetic Algorithm. An instruction consists of an opcode (OPCODE), operands (OR), and dependencies and is represented as a bit-string for the chromosome. A certain number of *chromosomes* are placed in an *individual* (stressmark size) that becomes an instruction sequence and a possible di/dt stressmark. *Population* is a collection of *individuals* and corresponds to one *generation*.

The Genetic Algorithm guides our simulations as shown in Fig. 3, and generates a di/dt stressmark as an output. With a control parameter setting, initial instruction sequences are generated and consist of a population in the first generation. All the individuals in the population are evaluated for the objective function – maximum voltage droop - with multiple simulations. Then, two of the high ranked individuals in the population are selected for reproduction, and they exchange a certain number of instructions with each other. The rate of reproduction is called *crossover rate* and it affects the overall optimization results because crossover rate determines the speed of convergence of the algorithm. After crossover, the characteristic of each individual can be changed by *mutation* that converts one or multiple bits of an individual instruction. Such GA operations repeat for a given number of generations, and a maximum voltage droop is determined at the end of the last generation.

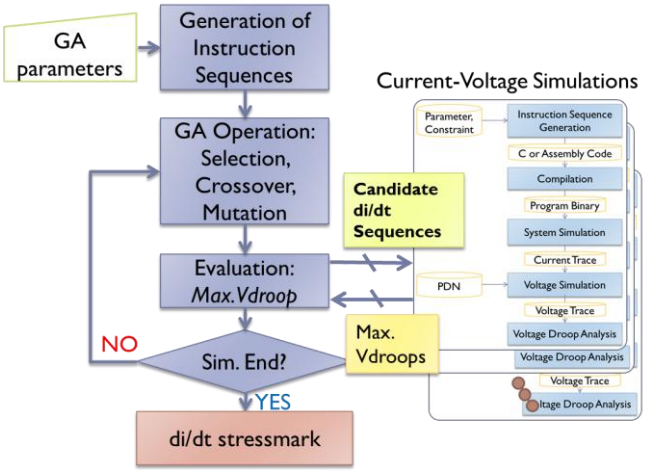

Figure 3. di/dt Stressmark Generation Framework using Genetic Algorithm

D. *Dependency Control and Register Assignment*

One of the knobs in the automatic framework is dependencies between instructions. Data dependencies cause a pipeline stall in a processor until it is resolved. In [5], dependencies are used to cause low current draw during part of a resonant period, and the same register is assigned to a target register of an instruction and a source register of a following instruction. Prior research [5] chose a floating-point divider instruction, ***divt***, as the only stalling instruction, but we do not impose this limitation. Any instruction is able to have a dependency with the previous instructions, and its operand registers are assigned according to the dependency.

IV. Experimental Results

In this section, we describe how we constructed our framework and show the simulation results. The simulators and their configurations and power distribution networks are carefully selected from the previous studies.

For the power (current) simulator, we select the combination of SimpleScalar and Wattch [3] to estimate current load variations per cycle in a microprocessor. We modify the original simulator to generate a current trace per cycle by dividing the power per cycle by the supply voltage. For circuit simulation, HSPICE is used to simulate the current trace and to measure voltage droop.

To apply the Genetic Algorithm to our simulation environments, we use GAUL [4] which provides an open source utility library for Genetic Algorithms including population creation, evolution, and evaluation.

We configure three different architectures to see the effectiveness and the architecture dependency of our di/dt stressmark generation method (Table 1). The base architecture configuration, **Arch1** shown in Table 1, is an 8-wide microprocessor with 3 GHz clock speed, based on the Pentium 4, similar to the configuration in [5]. For the second architecture, **Arch2**, we decrease the number of resources memory ports from 4 to 2 in order to reduce memory accesses, and other parameters were also adjusted to a 4-wide microprocessor. The last configuration, **Arch3**, is nearly the same as **Arch1**, but we increase the latency of a key component, *fdiv* unit from 12 to 18 cycles to see the architecture dependency of our di/dt generation method.

Table 1. Base Architecture Configuration for SimpleScalar

Parameter	Arch1	Arch2	Arch3
CPU Clock	3 GHz	3 GHz	(1) Latency of *fdiv* is changed from 12 to 18. (2) Other parameters are the same as Arch1
Fetch/Decode/Issue	8 / 8 / 8 instr.	4 / 4 / 4 instr.	
EXU	8 alu, 2 mul/div, 4 falu, 2 fmul/*fdiv*, 4 mem-port	4 alu, 2 mul/div, 2 falu, 2 fmul/*fdiv*, 2 mem-port	
RUU / LSQ	128 / 64	128 / 64	
Branch Predictor	Combined, 64Kb	Combined, 32Kb	
BTB	1K entries	512K entries	
L1 I/D-Cache	64KB, 2-way	32KB, 2-way	
L2 Cache	2MB, 8-way	1MB, 8-way	

Then, we take the five different power distribution network (PDN) circuits from the previous studies (Table 2). The first PDN from [7] is simple, but shows mid-frequency behavior which dominates the PDN's characteristic. The second PDN from [2] is an implementation of the Pentium 4's PDN. The third PDN used in [8] is also for Pentium 4, but has different resonant frequency, current swing, and number of RLC stages from **PDN2**. **PDN4-A** and **PDN4-B** [12] are the same circuits with different decoupling capacitance values.

TABLE 2. FIVE DIFFERENT PDNS FOR CIRCUIT SIMULATION

	PDN1 [7]	**PDN2 [2]**	**PDN3 [8]**	**PDN4-A [12]**	**PDN4-B [12]**
Resonant Frequency	100MHz	100MHz	68MHz	150MHz	200MHz
Current Swing	6-50A	3-20A	2-12A	5-16A	5-16A
#of RLC Stages	1	4	5	2	2

To compare the effectiveness of our di/dt stressmark to that of other methods, we run the SPEC CPU2006 suite with 100 million instructions, and program the hand-coded assembly code in [5]. The hand-coded di/dt stressmark consists of two parts; one is for low current draw, and the other is for high current draw. The low current draw part is implemented with the divider instruction, **divt**, which has a fixed, long latency. The high current draw part use a store instruction, **stq**, which store data to main memory through L1 and L2 caches. We find the best maximum voltage droop by increasing the number of the **stq** instruction from 0 to 200 under the given architecture and PDN configurations. Effort is made to create the best possible hand-coded baseline stressmark for comparison.

Table 3 compares the maximum voltage droop in milli-Volts of SPEC CPU2006, the hand-coded stressmarks, and our di/dt stressmarks. The larger number means the larger maximum voltage droop, and only the worst voltage droop is shown among the 22 SPEC benchmarks. Overall, our di/dt stressmarks always invokes larger maximum voltage droops than the other two methods. For **Arch1**, compared to SPEC CPU2006 and the hand-coded stressmark, 35.7% and 15.7% average increases in voltage droop are achieved by our automated di/dt stressmark for the different PDNs, respectively. In **Arch2**, architecture difference between **Arch1** and **Arch2** affects the performance of the di/dt stressmark, but our di/dt stressmark is less architecture-dependent because the hand-coded di/dt stressmark heavily depends on the number of memory ports due to the store instruction. Considering **Arch1** and **Arch3**, it is shown that the hand-coded di/dt stressmark significantly depended on the specific instruction, **divt**, executed in the *fdiv* unit whose latency is changed from 12 to 18 cycles. In contrast, our automated di/dt generation and SPEC benchmarks for **Arch3** make a similar range of voltage droops to **Arch1**'s regardless of the execution cycle change of the divider unit. This can also reveal that our automated di/dt stressmark generation technique is architecture-independent.

TABLE 3. MAXIMUM VOLTAGE DROOPS OF SPEC CPU2006, HAND-CODED [5], AND AUTOMATIC DI/DT STRESSMARKS

Arch. Config.	**PDN**	**SPEC CPU2006 (Worst) (mV)**	**Hand-Coded Droop (mV)**	**Auto-Stressed Droop (mV)**	**Improvement (Auto. vs. SPEC/ Auto. vs. Hand.)**
Arch1	PDN1	65.3	75.8	78.8	20.7% / 4.0%
	PDN2	111.9	112.9	121.5	8.6% / 7.6%
	PDN3	69.2	123.9	134.8	94.8% / 8.8%
	PDN4-A	101.1	107.6	137.5	36.0% / 27.8%
	PDN4-B	140.4	151.2	189.6	35.0% / 25.4%
Arch2	PDN1	26.4	29.0	34.9	32.2% / 20.3%
	PDN2	53.8	55.8	82.2	52.8% / 47.3%
	PDN3	41.8	45.8	60.5	44.7% / 32.1%
	PDN4-A	44.0	39.1	56.8	29.1% / 45.3%
	PDN4-B	53.7	46.5	82.4	53.4% / 77.2%
Arch3	PDN1	65.3	39.3	73.9	13% / 88%
	PDN2	110.9	62.8	130.2	17% / 107%
	PDN3	69.2	113.9	143.5	107% / 26%
	PDN4-A	101.7	80.5	153.0	50% / 90%
	PDN4-B	139.8	75.4	191.6	37% / 154%
Average (Overall)		**79.6**	**77.3**	**111.4**	**40% / 44%**

V. CONCLUSION

In this paper, we automatically generate a di/dt stressmark to test the maximum voltage droop in a microprocessor power distribution network. Our method removes the need for manual, tedium of designers to test di/dt effects in a given architecture and power distribution network. From the experimental results, our di/dt stressmarks effectively induce a maximum voltage droop, meeting the resonant frequency of a given PDN.

REFERENCES

[1] S. Pant and E. Chiprout, "Power Grid Physics and Implications for CAD," *Proceedings of the 43rd annual Design Automation Conference*, 2006.

[2] M. S. Gupta, J. L. Oatley, R. Joseph, G.-Y. Wei, and D. M. Brooks, "Understanding voltage variations in chip multiprocessors using a distributed power-delivery network," *Design, Automation & Test in Europe Conference & Exhibition*, 2007.

[3] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a Framework for Architectural-level Power Analysis and Optimizations," *27th Annual International Symposium on Computer Architecture*, 2000.

[4] Genetic Algorithm Utility Libray (GAUL), http://gaul.sourceforge.net/

[5] R. Joseph, D. Brooks, and M. Martonosi, "Control Techniques to Eliminate Voltage Emergencies in High Performance Processors," *Int'l Symposium on High-Performance Computer Architecture*, 2003.

[6] M. Ketkar and E. Chiprout, "A Microarchitecture-based Framework for Pre- and Post-silicon Power Delivery Analysis," *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2009.

[7] M. Powell and T. N. Vijaykumar, "Exploiting Resonant Behavior to Reduce Inductive Noise," *Int'l Symp. on Computer Architecture*, 2004.

[8] W. El-Essawy and D. Albonesi, "Mitigating Inductive Noise in SMT Processors," *International Symposium on Low Power Electronics and Design*, 2004.

[9] M. S. Gupta, K. Rangan, M. D. Smith, G.-Y. Wei, and D. M. Brooks, "DeCoR: A Delayed Commit and Rollback Mechanism for Handling Inductive Noise in Processors," *Int'l Symp. on High-Perf Computer Architecture*, 2008.

[10] K. Hazelwood and D. Brooks, "Eliminating Voltage Emergencies via Microarchitectural Voltage Control Feedback and Dynamic Optimization," *International Symposium on Low-Power Electronics and Design*, 2004.

[11] A. M. Joshi, L. Eeckhout, L. K. John, and C. Isen, "Automated microprocessor stressmark generation," *International Symposium on High Performance Computer Architecture*, 2008, pp. 229–239.

[12] Stephen Kosonocky, AMD, personal communication.