Analysis of Instruction-level Vulnerability to Dynamic Voltage and Temperature Variations

Abbas Rahimi[†], Luca Benini[‡], Rajesh K. Gupta[†]
[†]Department of Computer Science and Engineering, University of California, San Diego, USA
[‡]Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna, Bologna, Italy
abbas@cs.ucsd.edu, luca.benini@unibo.it, gupta@cs.ucsd.edu

***Abstract*—Variation in performance and power across manufactured parts and their operating conditions is an accepted reality in aggressive CMOS processes. This paper considers challenges and opportunities in identifying this variation and methods to combat it for improved computing systems. We introduce the notion of instruction-level vulnerability (ILV) to expose variation and its effects to the software stack for use in architectural/compiler optimizations. To compute ILV, we quantify the effect of voltage and temperature variations on the performance and power of a 32-bit, RISC, in-order processor in 65nm TSMC technology at the level of individual instructions. Results show 3.4ns (68FO4) delay variation and 26.7x power variation among instructions, and across extreme corners. Our analysis shows that ILV is not uniform across the instruction set. In fact, ILV data partitions instructions into three equivalence classes. Based on this classification, we show how a low-overhead robustness enhancement techniques can be used to enhance performance by a factor of 1.1x−5.5x.**

I. Introduction

Variability in transistor characteristics is a major design challenge in nanoscale CMOS technologies which causes performance and power uncertainty [1]. Both static and dynamic variations arise from different physical sources such as: (i) static inherent process parameter variations, e.g. channel length and threshold voltage variations due to random dopant fluctuations and sub-wavelength lithography; (ii) dynamic environmental variations in ambient condition such as temperature fluctuations and supply voltage droops [2]. Static process variations can sometimes be mitigated through binning or by post-silicon tuning, while dynamic variations change as a function of time and environment, and therefore cannot be compensated by static pre- and post-silicon tuning. Consequently accurate analysis coupled with efficient design techniques are required to overcome the variability challenge.

Examples of dynamic variation from environmental and workload changes include supply voltage droops and temperature changes. Voltage droops result from abrupt changes in the switching activity, inducing large current transients in the power delivery system (dI/dt problem), and contain high-frequency and low-frequency components which occur locally as well as globally across the die [3]. On the other hand, temperature variations occur at a relatively slow time scale with local hot spots on the die, depending on environmental, and workload conditions [4]. Designers commonly use conservative guard-bands into the operating frequency and voltage to handle these variations to ensure error-free operation within the presence of worse case dynamic variations over circuit lifetime [5][6] that leads to loss of operational efficiency.

An alternative is to use sensor circuits to detect dynamic variations coupled with an adaptive recovery methods for quick on-line error detection and compensation [7][8]. For instance, recent 45nm Intel resilient processor core [9] integrates two fast error detection mechanisms: it uses error-detection sequential (EDS) [10] circuits in critical paths to detect late timing transitions; and it also places a tunable replica circuit (TRC) [11] per pipeline stage to monitor worst-case delays. To ensure recovery, the processor supports dynamic frequency scaling as well as multiple-issue instruction replay that corrects errant instructions. In a similar vein, Razor [7] storage devices have been used in a 65nm ARM ISA processor [12], running at frequencies over 1GHz, where fast dynamic variations are significant. Less intrusive and low-overhead on-chip variability sensors using PLLs [2] and ring oscillators (RO) [13] have been proposed to detect process, supply voltage, and temperature variations.

Further progress in this area requires a careful analysis of the effect of variations on individual instructions. This paper makes three contributions. First, we analyze the effect of a full range of voltage and temperature variations on the performance and power of the 32-bit in-order RISC processor. Second, we introduce the notion of instruction-level vulnerability (ILV) to characterize dynamic variations. Our results show that ILV is not uniform across the instruction set. Third, using ILV data we show the effectiveness of a minimally intrusive and parsimonious technique to mitigate the dynamic variations that achieves up to 5.5x performance improvement in comparison to the traditional worst-case design.

II. Instruction-level Vulnerability to Variations

A. Effect of Operating Conditions

We analyze the effect of a full range of operating conditions on the performance and power of the LEON-3 [14] processor compliant with the SPARC V8 architecture. Specifically, we used a temperature range of -40°C−125°C, and a voltage range of 0.72V−1.1V. Figure 1 shows how the critical path of the processor varies across corners. The higher voltage results in the shorter critical path, while the lower temperature leads to a higher delay in the low-voltage region (voltage ≤ 0.9V), since MOSFET drain current decreases when the temperature is decreased in the deep submicron technologies [15]. These operating condition (hence dynamic) variations cause the critical path delay to increase by a factor of 6.1x when the operating condition is varied from the one corner to the other. Consequently, a large conservative guard-band into the operating frequency is needed to ensure the error-free operation in presence of the dynamic variations.

978-3-9810801-8-6/DATE12/©2012 EDAA

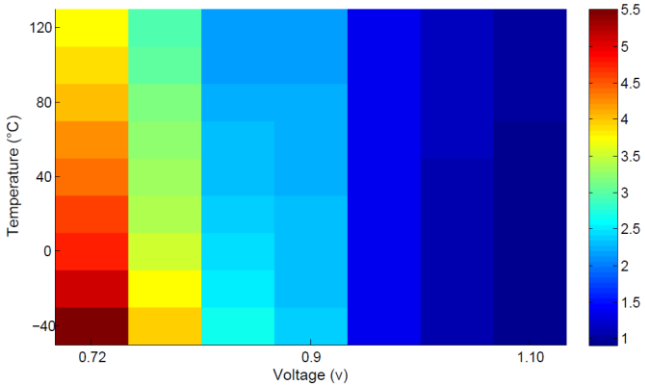

Figure 1. Effect of voltage and temperature variations on the critical path (ns).

B. *Variability among Pipeline Stages*

We evaluate the critical paths of each pipeline stage for a given cycle time, while changing the operating conditions. Figure 2 shows the number of failed paths with a negative slack for each parallel pipeline stages across three corners. The cycle time is set at 0.85ns, and voltage varies from 0.72V to 0.88V, and then to 1.10V at a constant temperature of 125°C. As shown in Figure 2, most of the failed paths lie in the *execute* and *memory* stages in all three operating voltages. On the other hand, each of the *fetch, decode*, and *register access* stages contains less than 40K failed paths. Furthermore, there is a relatively small fluctuation in their number of critical paths across voltage variations for these stages. Quantitatively, the *memory* stage at operating voltage of 0.72V has 1.3x, 1.8x, 3.8x more critical paths in comparison to the *execute*, *write back*, and *decode* stages, respectively. *Memory* stage at operating voltage of 1.10V also faces 1.4x, 1.9x more critical paths when the voltage drops to 0.88V, 0.72V, respectively.

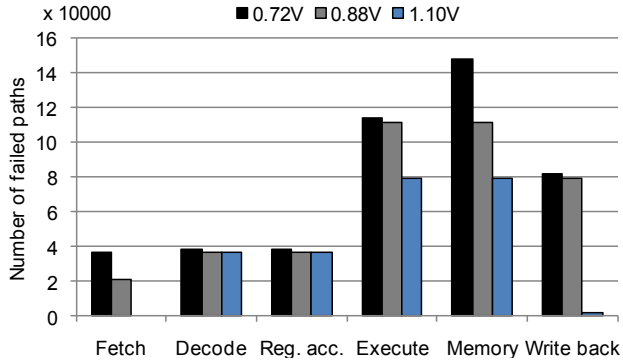

Figure 2. Effect of voltage variation on the pipeline stages at 125°C.

Similarly, the temperature of processor is varied from -40°C to 125°C at a constant voltage of 1.1V. As a result, there are no failed paths in the *fetch* stage when the temperature is varied, and only a small number of failed paths are found in the *write back* stage at the highest temperature. On the other hand, similar to Figure 2, many paths fail within the *execute* and *memory* stages. The *execute* and *memory* parts of the processor are not only very sensitive to voltage and temperature variations, but also exhibit a large number of critical paths in comparison to the rest of processor. Therefore, we would anticipate that the instructions that significantly exercise the *execute* and *memory* stages are likely to be more vulnerable to voltage and temperature variations.

Let us now examine the situation of all paths through the processor under different operating condition and frequency. The Y-axis of Figure 3 shows the proportion of failed paths to non-failed paths for three corners. We observe that this proportion of failed paths suddenly drops below a certain threshold while the clock is finely scaled with a resolution of 0.01ns. For instance, the proportion falls below 0.5 with only 0.06ns clock scaling at (1.10V, 0°C); in the other words, the number of non-failed paths is twice as many as those which fail. Alternatively, the number of non-failed paths is doubled when the cycle time is increased for 0.3ns at (0.9V, 125°C). These provide an opportunity for an error-free running of some instructions that will not activate those failed paths.

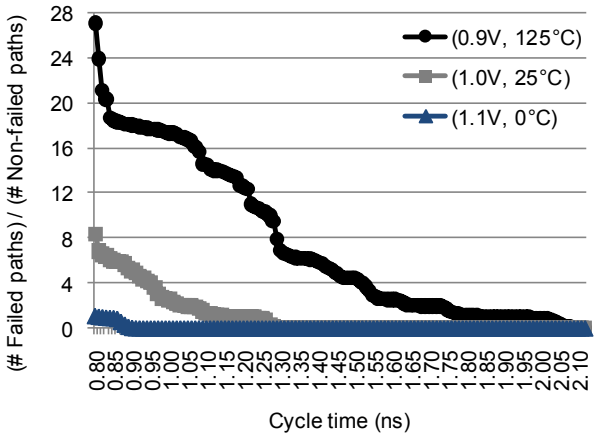

Figure 3. The proportion of failed paths to non-failed paths versus clock.

From the previous analysis, we get the intuition that instructions will have different levels of vulnerability to variations depending on the way they exercise the non-uniform critical paths across the various pipeline stages. To capture this phenomenon, we define the concept of instruction-level vulnerability to dynamic variations. The classification of instructions is a valuable mechanism to alleviate the guard-banding and improving performance: (i) within a fixed corner, by acquiring the knowledge about which class of instructions is running, the processor can adapt the guard-banding accordingly, without any need for the intrusive variability sensor/observer; (ii) across every corner, processor can adjust its guard-banding for all class of instructions by using a low-overhead variability observer, e.g. PLL[2], RO [13].

III. Methodology and Experimental Results

In this section, we describe the instruction-level characterization methodology and experimental results for performance and power of the integer pipeline of LEON-3 [14] with hardware multiplier/divider units as well as the instruction/data caches. First, we synthesized the open-source VHDL code of LEON-3 with the TSMC 65nm technology library, general purpose process. The sign off stage for accurate analysis of the operating conditions has been made with Synopsys PrimeTime, thanks to its voltage-temperature scaling features for the composite current source approach of modeling cell behavior. Mentor Graphics' ModelSim is also used for detail gate-level simulation.

A. Monte Carlo Gate-Level Simulation

In the gate-level simulation, for each individual instruction, we apply the Monte Carlo method to observe instruction behavior. To accurately exercise each instruction, we use a normal distribution for the sources, destination, and immediate operands. A large sample of the SPARC ISA is evaluated, including the logical/arithmetic instructions, memory access instructions (load/store), multiply/divide instructions. To quantify the ILV to voltage and temperature variations, we define the probability of failure (PoF) for each instruction$_i$ in (1), where N_i is the total number of clock cycles in Monte Carlo simulation which takes to execute instruction$_i$ with random operands; and Violation$_j$ indicates whether there is a violated stage at clock cycle$_j$ or not.

$$\text{PoF}_i = \frac{1}{N_i}\sum_{j=1}^{N_i}\text{Violation}_j$$
$$\text{Violation}_j = \begin{cases} 1 & \text{If any stage violates at cycle}_j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

In other words, PoF$_i$ defines as the total number of violated cycles over the total simulated cycles for the instruction$_i$. If any of the analyzed stages have one or more violated flip-flop at clock cycle$_j$, we consider that stage as a violated stage at cycle$_j$. Intuitively, if instruction$_i$ runs without any violated path, PoF$_i$ is *0*; on the other hand, PoF$_i$ is *1* if instruction$_i$ faces at least one violated path in any stage, in every cycle.

B. Instruction-Level Delay Variability

The following tables (Table 1-2) summarize the PoF of each evaluated instruction across various corners. We finely change the clock cycle to observe the paths failure for every exercised instruction, and then consequently evaluate its PoF. As shown, instructions exhibit a very wide range of delay under different operating conditions ranges from 0.76ns to 4.16ns. More precisely, the PoF values shown in tables evidence two important facts. First, for their vulnerability to variations, instructions are partitioned into three main classes: (i) the logical/arithmetic instructions, (ii) the memory instructions, and (iii) the multiply/divide instructions. The 1st class shows an abrupt behavior when the clock cycle is slightly varied. Its PoF switches from *1* to *0* with a slight increase in the cycle time (0.02ns) for every corner, mainly because the path distribution of the exercised part by this class is such that most of the paths have the same length, then we have a all-or-nothing effect, which implies that either all instructions within this class fail or all make it. The 2nd class, the memory instructions, needs much more relaxed cycle time to be able to survive across conditions. For instance, as shown in Table 2, only 0.04ns more guard-banding on the cycle time of the 1st class instruction can guarantee the error-free execution of the memory instructions while they are experiencing 40°C temperature fluctuation. The 3rd class is the multiply/divide instructions which need higher guard-banding in comparison to the 1st class instruction, ranges from 0.02ns at (1.1V, -40°C) to 0.30ns at (0.72V, 125°C). Since this class highly exercises the *execution* unit[1], it has a higher PoF in comparison with the rest of classes in the same clock cycle, for every corner.

TABLE 1. PROBABILITY OF FAILURE OF ISA AT VOLTAGES 1.1V AND 1.0V, WHILE VARYING TEMPERATURE AND FREQUENCY.

Corners		(1.1V, -40°C)			(1.1V, 0°C)			(1.1V, 125°C)						(1.0V, 25°C)				
Cycle time (ns)		0.74	0.76	0.78	0.74	0.76	0.78	0.80	0.82	0.84	0.86	0.88	0.90	1.08	1.10	1.12	1.14	1.22
Logical & Arithmetic	add	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*
	and	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*
	or	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*
	sll	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*
	sra	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*
	srl	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*
	sub	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*
	xnor	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*
	xor	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*
Mem	load	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**	0.786	*0*	*0*	*0*
	store	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**	0.814	*0*	*0*	*0*
Mul. &Div	mul	**1**	*0*	*0*	**1**	0.967	*0*	**1**	0.042	0.015	0.012	0.002	*0*	**1**	0.998	0.976	0.074	*0*
	div	**1**	0.837	*0*	**1**	0.948	*0*	**1**	0.991	0.991	0.984	0.984	*0*	**1**	0.964	0.993	0.990	*0*

TABLE 2. PROBABILITY OF FAILURE OF ISA AT CONSTANT VOLTAGE 0.72V, WHILE VARYING TEMPERATURE AND FREQUENCY.

Corners		(0.72V, -40°C)				(0.72V, 0°C)					(0.72V, 125°C)						
Cycle time (ns)		4.10	4.12	4.14	4.16	3.58	3.60	3.62	3.64	3.66	2.88	2.90	2.92	2.94	2.98	3.00	3.20
Logical & Arithmetic	add	**1**	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	*0*
	and	**1**	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	*0*
	or	**1**	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	*0*
	sll	**1**	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	*0*
	sra	**1**	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	*0*
	srl	**1**	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	*0*
	sub	**1**	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	*0*
	xnor	**1**	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	*0*
	xor	**1**	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	*0*
Mem	load	**1**	0.823	0.823	*0*	**1**	0.823	0.823	*0*	*0*	**1**	0.823	0.823	0.823	0.796	0.796	*0*
	store	**1**	0.847	0.847	*0*	**1**	0.847	0.847	*0*	*0*	**1**	0.847	0.847	0.847	0.823	0.823	*0*
Mul. &Div	mul	**1**	0.995	0.995	*0*	**1**	0.996	0.994	0	*0*	**1**	0.998	0.997	0.996	0.996	0.996	*0*
	div	**1**	0.995	0.995	*0*	**1**	0.995	0.995	0.812	*0*	**1**	0.994	0.994	0.993	0.991	0.991	*0*

[1] Moreover, 64%−82% (depends on the corner) of the failed paths in the *execution* stage lie in the hardware multiplier and divider units.

Further, based on these results, we can define an adaptive clock cycle for each class of instructions to mitigate the conservative guard-banding, not only within a fix process corner, but also across corners. All instruction classes act similarly across the wide range of operating conditions: as the cycle time increases gradually, the PoF becomes *0*, firstly for the 1st class, then for the 2nd class, and finally for the 3rd class. A processor can benefit from this by adapting its guard-banding for each class of instruction by acquiring the knowledge about which class of instructions is/will be running.

C. *Evaluating Effectiveness of Less Intrusive Variation-Tolerant Technique*

All intrusive techniques [7][9][12] try to avoid timing failure for instructions that activate the critical paths by dynamically switching to two-cycle operation. These expensive, instruction by instruction timing adjustment techniques do not expose opportunity for further software-level optimizations especially for sequences and classes of instructions. Therefore, we could have an advanced dynamic clock speed adaptation technique, possibly compiler driven, which can quickly decide on the clock speed of the processor at a very fine grain[16], just looking at the fetched instructions and keeping track of their entry into the stages, and at the same time monitoring the current corner with a low-overhead monitoring hardware [2][13]. This technique not only provides great performance enhancement for processor, but also is a step forward to the less intrusive and parsimonious robust design.

Table 3 shows how a program consisting of various classes of instructions can benefit by this technique under different operating conditions: the performance improvement when processor runs a program only consists of specific classes, in comparison to the traditional worst-case design. For instance, at the typical operating condition (1.0V, 25°C) processor can decrease the cycle time form 4.16ns (Table 2) to 1.22ns (Table 1), and consequently achieves 3.4x speed improvement, when its running program consists of all three classes. It can further reduce the cycle time to 1.12ns (3.7x speedup) when only the 1st, and 2nd classes of instructions are used in its program. As shown, the proposed solution can greatly achieve 1.1x-5.5x performance improvement depends on the type of instruction and the operating condition.

TABLE 3. PERFORMANCE IMPROVEMENT FOR CLASSES OF INSTRUCTIONS.

Vol. (V)	Temp. (°C)	1st and 2nd class	1st, 2nd, 3rd class
1.10	-40	5.5x	5.3x
1.10	0	5.5x	5.3x
1.10	125	5.1x	4.6x
1.00	25	3.7x	3.4x
0.88	-40	3.9x	3.7x
0.88	0	3.9x	3.7x
0.88	125	3.9x	3.5x
0.72	0	1.1x	1.1x
0.72	125	1.3x	1.3x

From the previous sections, we show how the delay of instructions varies intra- and inter-corner. Let us now examine the power variability of the instruction classes when the cycle time is adjusted for each class accordingly, i.e. the best frequency for each class is applied. As a result, all three classes of instructions experience a wide range of total power variability (0.1mW−2.6mW), 1.15x intra-corner power variation (across the three classes) due to exercising various parts of processor, and 26.7x inter-corner power variation, at maximum. This implies that ILV could potentially expose opportunity for further software-level optimizations for both performance and power.

IV. CONCLUSION

The concept of instruction-level vulnerability to dynamic voltage and temperature variations is defined. Based on that, all exercised instruction in the integer pipeline of LEON-3 are partitioned into three classes for the full range of operating condition: (i) the logical and arithmetic instructions, (ii) the memory instructions, and (iii) the multiply and divide instructions. Leveraging this classification in conjunction with less intrusive variability observers provides us a great opportunity to enhance processor performance by 1.1x−5.5x, in TSMC 65nm technology. It is also a step forward to the low-overhead, efficient, and parsimonious robust design. Our ongoing work is focused on the characterization of a sequence of instruction including the control instructions.

V. ACKNOWLEDGMENTS

This research was supported by NSF Variability Expeditions Award CCF-1029783, and JTI SMECY (ARTEMIS-2009-1-100230).

REFERENCES

[1] S. Ghosh, et al., "Parameter Variation Tolerance and Error Resiliency: New Design Paradigm for the Nanoscale Era," Proc. of the IEEE , Vol.98, No.10, pp.1718-1751, Oct. 2010.

[2] K. Kang, et al., "On-chip variability sensor using phase-locked loop for detecting and correcting parametric timing failures," IEEE Tran. on VLSI Systems, Vol. 18, No. 2, pp. 270-280, 2010.

[3] K. Bowman, et al., "Dynamic Variation Monitor for Measuring the Impact of Voltage Droops on Microprocessor Clock Frequency," Proc. *CICC*, pp. 1-4, 2010.

[4] S. Murali, et al., "Temperature Control of High-Performance Multi-core Platforms Using Convex Optimization," Proc. *DATE*, pp.110-115, 2008.

[5] J. Tschanz, et al., "Adaptive circuit techniques to minimize variation impact on microprocessor performance and power," Proc. *ISCAS*, pp. 9-12, 2005.

[6] K. Bowman, et al., "Circuit techniques for dynamic variation tolerance," Proc. *DAC*, pp. 4–7, 2009.

[7] D. Ernst et al., "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," Proc. *MICRO*, pp. 7-18, 2003.

[8] N. Shah, et al., "Built-In Proactive Tuning System for Circuit Aging Resilience," Proc. *DFT*, pp. 96-104, 2009.

[9] K. Bowman, et al. "A 45 nm Resilient Microprocessor Core for Dynamic Variation Tolerance," IEEE J. of Solid-State Circuits, Vol.46, No.1, pp.194-208, Jan. 2011.

[10] K. Bowman, et al., "Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance," IEEE J. of Solid-State Circuits, Vol.44, No.1, pp.49-63, Jan. 2009.

[11] J. Tschanz, et al., "Tunable Replica Circuits and Adaptive Voltage-Frequency Techniques for Dynamic Voltage, Temperature, and Aging Variation Tolerance," IEEE Symp. VLSI Circuits Dig. Tech. Papers, pp. 112-113, 2009.

[12] D. Bull, et al., "A power-efficient 32b ARM ISA processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation," Proc. *ISSCC*, pp. 284-285, 2010.

[13] M. Bhushan, et al., "Ring oscillators for CMOS process tuning and variability control," IEEE Tran. on Semiconductor Manufacturing, Vol.19, No.1, pp. 10- 18, Feb. 2006.

[14] LEON-3 [Online]. Available: http://www.gaisler.com/cms/index.php ?option=com_content&task=view&id=13&Itemid=53

[15] R. Kumar, et al., "Reversed Temperature-Dependent Propagation Delay Characteristics in Nanometer CMOS Circuits," IEEE Trans. on Circuits and Systems, Vol.53, No.10, pp.1078-1082, Oct. 2006.

[16] J. Tschanz, et al., "Adaptive Frequency and Biasing Techniques for Tolerance to Dynamic Temperature-Voltage Variations and Aging," Proc. ISSCC, pp.292-604, 2007.