Deployment of Better Than Worst-Case Design: Solutions and Needs

Todd Austin **Valeria Bertacco**
The University of Michigan
1301 Beal Ave, Ann Arbor, MI 48109
{*austin,valeria*}*@umich.edu*

Abstract

The advent of nanometer feature sizes in silicon fabrication has triggered a number of new design challenges for computer designers. These challenges include design complexity and operation in the presence of environmental and device uncertainty. To make things worse, these new challenges add to the many challenges that designers already face in order to scale system performance while meeting power and reliability budgets. Current design objectives are being met by applying even more engineers and increasing overall design times, an unsustainable trend. This paper overviews a novel design strategy, called Better Than Worst-Case design, that addresses these challenges through a methodology based on separating the concerns of performance and reliability by coupling complex design components with simple reliable checker mechanisms. We present the key aspects of Better Than Worst-Case Design and cover some recently proposed solutions that deploy this technique in application domains ranging from microprocessors to digital signal processors. We then highlight a few aspects that need to be addressed to make this approach more practical in general contexts and suggest possible solutions.

1. Introduction

A critical aspect of any computer design is its reliability. Users expect a system to perform without fail when asked to compute a task. In reality, it is impossible to build a fully correct and reliable system, consequently, computer manufacturers target failure rates that are imperceivable small [22]. In most systems today, reliability targets are met by utilizing a fault-avoidance design methodology. However, the trend in semiconductor fabrication towards the nanometer scale devices has created many new design challenges that make these targets harder than ever: Design complexity, process variation, and single-event upsets all conspire to compromise system correctness and reliability. Unless these challenges are addressed, computer vendors can expect poor design quality and long times-to-market.

Design Complexity As the industry rides Moore's law, designers have available to them exponentially increasing transistor budgets, leading to growing design complexity. Consequently, significant design resources are dedicated to design verification. Recently, the ITRS (the International Technology Roadmap for Semiconductors) assessed that it takes thousands of engineer-years to develop top-end systems, yet processors still reach the market with hundreds of bugs [3]. Moreover, more than twice as many resources are spent on verification compared to design in the microprocessor arena, bringing the design-to-verification gap to crisis proportions.

Process Variation Another reliability challenge designers face is the design uncertainty that is created by increasing process variations. Process variations result from device dimension and doping concentration variations that occur during silicon fabrication. These variations are of particular concern because their effects on devices are amplified as device dimensions shrink [1], resulting in structurally weak and poor performing devices. Designers are forced to deal with these variations by assuming worst-case device characteristics (usually, a 3-sigma variation from typical conditions), which leads to overly conservative designs. The effect of this conservative design approach is quite evident by the extent to which hobbyists can overclock high-end microprocessors [20].

Single-Event Radiation (SER) There is a growing concern about providing protection from soft errors caused by charged particles (such as neutrons and alpha particles) that strike the bulk silicon portion of a die [28]. SER events are caused when high-energy particles strike the P-N junction of a MOSFET, which generates a current-pulse in the depletion region, resulting in a temporary loss of charge stored on the P-N junction. If the charge variation in the P-N junction is sufficient to alter the logic value corresponding to the voltage across the junction, a single-event upset (SEU) results. The final effect is a logic glitch that can potentially corrupt combinational logic computation or state bits. While a variety of studies have been performed that demonstrate the unlikeliness of such events [18, 25], con-

cern remains in the architecture and circuit communities. This concern is fueled by the trends of reduced supply voltage and increased transistor budgets, both of which exacerbate a design's vulnerability to SEU faults.

The combined effect of these challenges is that designers are forced to work harder just to keep up with system performance, power and reliability goals. The monumental task of meeting these goals with limited resource budgets and increasing time-to-market pressure has raised these design challenges to crisis proportion.

1.1. Better-than worst case design (BTWC)

To meet these challenges, we are advocating a novel design methodology, called Better Than Worst-Case design, that employs a design style which separates the concerns of correctness and robustness from the ones of performance and power [5]. The name Better Than Worst-Case (BTWC) design underscores the improvement that this approach provides over worst-case design techniques. Traditional worst-case design techniques strive to build complete systems which satisfy the guarantees of correctness and robust operation. The previously highlighted design challenges collude to make this an increasingly untenable design technique. Better Than Worst-Case designs take a markedly different approach, as illustrated in Figure 1.

In a Better Than Worst-Case design, the core component of the design is coupled with a checker unit that validates the semantics of all core operations. The advantage of such designs is that the efforts with respect to correctness and robustness are concentrated on the checker component. The performance and power efficiency concerns of the design are relegated to the core component, and consequently they can be addressed independently of any correctness concerns. By removing the correctness concerns from the core component, many core design constraints are relaxed, making this approach much more amenable to addressing future physical design challenges. In addition, by delegating the responsibility of correctness and reliability to the checker, it becomes possible to quickly build provable correct designs that effectively address both performance and reliability constraints.

In a Better Than Worst-Case methodology, the designer needs to consider only typical design constraints in order to satisfy design closure in areas such as timing, noise, power, and verification. Consideration of corner cases, which take up to 90% of the design time, but occur only very rarely in practice, can be ignored and are handled seamlessly using embedded error detection and correction circuitry contained within the checker component.

The remainder of this paper is organized as follows. Section 2 highlights a Better Than Worst-Case design technology called the DIVA checker. The DIVA checker is a hardware component that provides a very high-level of design error coverage for a microprocessor design. Section 3 highlights a number of other Better Than Worst-Case design applications from the literature. In Section 4, we suggest future applications and needs, and finally in Section 5 we give conclusions.

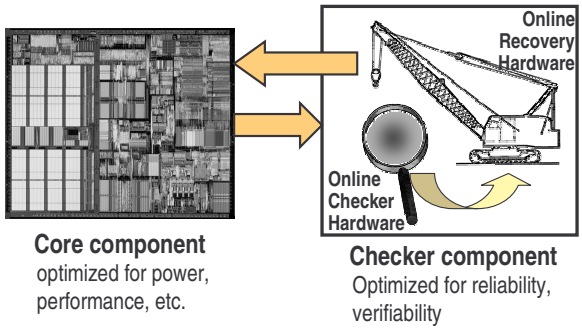

Figure 1. Better Than Worst-Case design approach

2. DIVA: BTWC design for microprocessors

In our research, we have been exploring ways to employ Better Than Worst-Case design techniques to ease the verification of complex microprocessors. The DIVA (Dynamic Implementation Verification Architecture) project has developed a microprocessor checker component that provides a near complete separation of concerns for performance and correctness [7, 9, 26]. The design, which is illustrated in Figure 2, employs two processors: a sophisticated core processor that quickly executes the program, and a checker processor that verifies the same program by re-executing all instructions in the wake of the complex core processor.

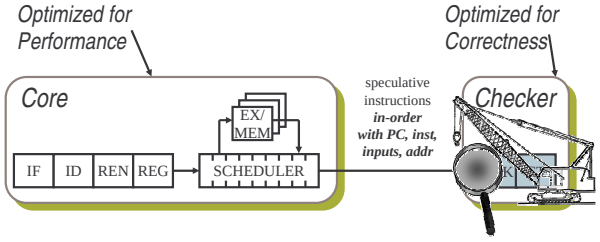

Figure 2. DIVA Architecture

The core processor is responsible for pre-executing the program to create the instruction prediction stream. This stream consists of all executed instructions (delivered in program order) with their input values and any memory addresses referenced. In a typical design, the core processor is identical in every way to a traditional complex microprocessor core, up to the retirement stage of the pipeline (where register and memory values are committed to state

resources). The checker executes in the wake of the core processor, verifying all computation by re-executing all program instructions. The high-quality stream of instruction predictions from the core processor simplifies the design of the checker processor and speeds up its processing. Pre-execution of the program on the complex core processor eliminates all of the checker's processing hazards (*e.g.*, branch mispredictions, cache misses and data dependencies) that slow simple processors and necessitate complex microarchitectures. Thus, it is possible to build a simple in-order checker pipeline without speculation that can match the retirement bandwidth of the core. In the event of the core generates a bad prediction value (*e.g.*, due to a core design error), the checker fixes the errant value, flushes all internal state from the core processor, and then restarts the core at the instruction following the computation error.

For dynamic verification to be viable, the checker processor must be simple and fast. It must be simple enough to reduce the overall design verification burden, and fast enough to not slow the core processor. A single-issue two-stage checker processor is illustrated in Figure 3a. The design presented assumes a single-wide checker, but scaling to wider or deeper designs is a straightforward task [9].

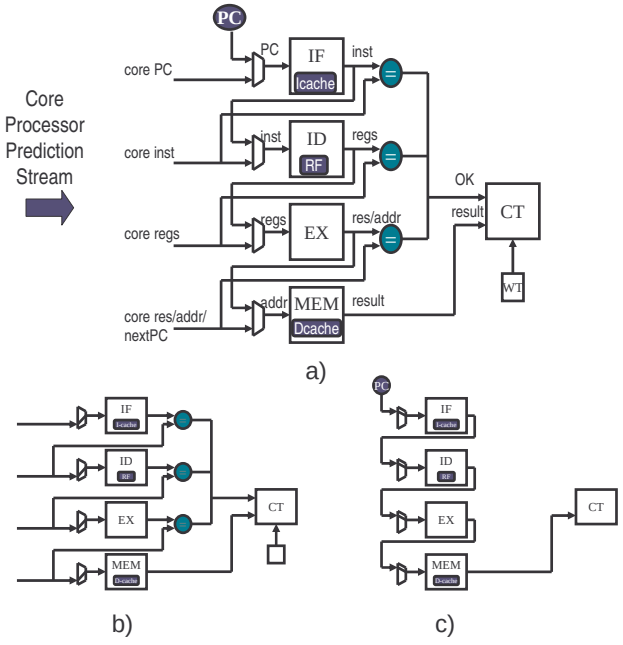

Figure 3. Operation of the checker processor

During normal checker operation (as shown in Figure 3b), the core processor sends instructions (with predictions) to the checker pipeline. These predictions include the next PC, instruction, instruction inputs, and addresses referenced (for loads and stores). The checker processor ensures the correctness of each instruction result through four parallel checker stages, each of which verifies a separate component of the prediction stream. If each prediction from the core processor is correct, the result of the current instruction (a register or memory value) as computed by the checker processor is allowed to retire to non-speculative storage in the commit (CT) stage of the checker processor.

In the event any core computation is found to be incorrect, the bad result is fixed, the core processor is flushed, and the core and checker processor pipelines are restarted after the errant instruction. In this event, as shown in Figure 3c, the routing MUXes are configured to form a recovery pipeline. The recovery pipeline is a serial pipeline, very similar to a classic five-stage processor pipeline. In recovery mode, all instruction computations are sent to the immediate next logical stage in the checker processor pipeline, rather than used to verify core predictions. Unlike the classic five-stage pipeline, only one instruction is allowed to enter the recovery pipeline at a time. As such, the recovery pipeline configuration does not require bypass datapaths or complex scheduling logic to detect hazards. Processing performance for a single instruction in recovery mode will be quite poor, but as long as faults are infrequent there will be no perceivable impact on program performance [7]. Once an errant instruction has retired, the checker processor re-enters normal processing mode and restarts the core processor after the errant instruction.

We have shown through cycle-accurate simulation and timing analysis of a physical checker design that our approach preserves system performance while keeping low area overheads and power demands [7]. Furthermore, analysis suggests that the checker is a simple state machine that can be formally verified [17], scaled in performance and reused [26].

The simple DIVA checker addresses the concerns highlighted in the introduction, in that it provides significant resistance to design and operational faults, and provides a convenient mechanism for efficient and inexpensive detection of core faults. Specifically, if any design errors remain in the core processor, they will be corrected (albeit inefficiently) by the checker processor. The impact of design parameter uncertainty is mitigated since the core processor frequency and voltage can be tuned to typical-case circuit evaluation latency. The DIVA approach uses the checker processor to detect energetic particle strikes in the core processor. As for the checker processor, we have developed a re-execute-on-error technique that allows the checker to check itself [26].

3. BTWC design in other application domains

A number of other Better Than Worst-Case designs exists, developed by the authors and others. To give the reader a broader perspective of how this technology can be applied, we present additional design examples from the literature.

Razor logic is an error-resilient circuit design methodology based on in-situ detection of circuit timing errors [6]. The approach utilizes Razor latches, which double sample combinational logic output values. The first sample is taken with an aggressively timed clock, and the result is forwarded on to later logic. The second sample, from the so-called shadow latch, is a reliable sample of the combinational logic output, which is compared to the earlier sample. If the two samples match, there was sufficient time/energy to complete the computation, otherwise, circuit timing error recovery is initiated. Timing error recovery is implemented with microarchitectural support. In this event, the pipeline is flushed and the correct result from the shadow latch is inserted back into the pipeline. A physical Razor processor design demonstrated that the approach could cut energy requirement nearly in half with little performance impact, by utilizing error rates to tune voltage to the minimum necessary level [10]. In addition, novel circuit-aware architectural simulation techniques were developed that provide fast simulation analysis of microarchitectural designs that make decisions based on circuit-level phenomena [15].

Algorithmic noise tolerance is another Better Than Worst-Case design technique that takes advantage of special properties of signal processing applications to reduce processing energy requirements [12]. The technique couples a high-precision digital signal processor with a reduced-precision version (*e.g.*, utilizing fewer bits of precision). In the event there is insufficient voltage to correctly compute the high-precision result, it will be overridden by the less accurate (but less energy demanding) low-precision result. Using the combination of high and low-precision processors, it is possible to significantly reduce energy requirements (via aggressive voltage scaling), while only introducing a tolerable amount of noise into the computation (in the form of occasional low-precision computations).

Lu's work on approximate circuits recognized that many computational circuits, *e.g.*, adders, typically produce their final result long before the worst-case latency [16]. To leverage this property, they defined modified microprocessor functional units which could, for most computations, quickly compute a result, but in the event that the computation was more heavy weight (*e.g.*, an adder operation with a long carry-chain) an additional computation cycle was provided by the microprocessor scheduler. Lu also demonstrated that redesigning functional units for optimized typical-case latency produced additional gains. One example of this design style was an adder with circuits optimized for highly likely short carry propagations.

Worm introduced the use of Better Than Worst-Case design for on-chip busses [27]. In their design, they used communication-based coding techniques to protect transmissions over on-chip busses. In the event that a transmission does not reach the other end of the bus without errors, a retransmission (possibly with increased voltage) is requested. By monitoring the error rate of the on-chip bus, they found that they could reduce voltage until small error rates were encountered, thereby eliminating on-chip bus voltage margins and reducing overall energy requirements.

The techniques presented in this section comprise only a subset of the Better Than Worst-Case designs we have found in the literature. The interested reader will also find clever applications in the work of Kehl [14], Anghel [4], and Uht [24].

4. Future applications and needs

Looking ahead, we see a number of promising applications for Better Than Worst-Case design methodologies, plus challenges in their deployment. In this section we highlight one of our current efforts to apply this methodology to the design of a defect-tolerant chip multiprocessor. In addition, we highlight one of the greatest deployment challenges, that is, how do we reduce the human effort required to incorporate this methodology into existing and ongoing designs. We briefly introduce our effort to implement turnkey Better Than Worst-Case design methodologies.

4.1. BTWC for defect tolerance

As silicon technologies move into the nanometer regime, there are a number of failure factors that have risen in importance. Many technology experts agree that transistor reliability is bound to wane as devices become subject to extreme process variation, particle-induced transient errors, and transistor wearout. Unless these challenges are addressed, manufacturers can expect low yields and short mean-times-to-failure. There are a number of possible causes for device failures; in the following subsections we highlight the most relevant ones.

Manufacturing Defects. Deep sub-micron technologies are increasingly vulnerable to several fabrication-related failure mechanisms. Optical proximity effects at the sub-micron level, airborne impurities, and processing material defects can all lead to faulty transistors and interconnect [21]. In addition, manufacturing defects arise from a range of processing problems that manifest during fabrication. For example, step coverage problems that occur during the metalization process may cause open circuits. Post-manufacturing test [19] and built-in self-test (BIST) [2] are two techniques to impress test vectors onto circuits to identify manufacturing defects. A more global approach to detect defects is taken by IDDQ testing, which uses on-board current monitoring to spot short-circuits. During IDDQ testing, any abnormally high current spike found during functional testing is indicative of short-circuit defects [8].

Gate Oxide Wearout. Technology scaling has adverse effects on the lifetime of transistor devices, due to time-dependent wearout. There are three major failure modes for time-dependent wearout: electromigration, hot carrier degradation (HCD), and time-dependent oxide breakdown. Electro-migration results from the mass transport of metal atoms in chip interconnects. The trends of higher current density in future technologies increases the severity of electromigration, leading to a higher probability of observing open and short-circuit nodes over time [11]. HCD is the result of carriers being heated by strong electrical fields and subsequently being injected into the gate oxide. The trapped carriers cause the threshold voltage to shift, eventually leading to device failure. HCD is predicted to worsen for thinner oxide and shorter channel lengths [13]. Time-dependent oxide breakdown is due to the extensive use of ultra-thin oxide for high performance. The rate of defect generation in the oxide is proportional to the current density flowing through it, and therefore it is increasing dramatically due to relentless down-scaling [23].

Transistor Infant Mortality. Scaling has had adverse effects on the early failures of transistor devices. Traditionally, early transistor failures have been reduced through the use of burn-in. The burn-in process utilizes high voltage and temperature to accelerate the failure of weak devices, thereby ensuring that the parts that survive burn-in only possess robust transistors. Unfortunately, burn-in is becoming less effective in the nanometer regime, as deep sub-micron devices are subject to thermal run-away effects, where increased temperature leads to increased leakage current and increased leakage current leads to yet higher temperatures. The end results is that aggressive burn-in will destroy even robust transistors. Consequently, vendors may soon have to relax the burn-in process which will ultimately lead to more early-failures for transistors in the field.

The first design point we are tackling is *BulletProof*, a defect-tolerant chip-multiprocessor, capable of tolerating medium to high levels of defects. Our design approach is a Better Than Worst-Case design based on system-level error checking and recovery. Additionally, due to the permanent nature of silicon defects, we must enhance the methodology to include repair mechanisms. Through the utilization of on-line testing diagnostics and run-time reconfiguration, it should be possible to repair silicon defects and restore normal operation to the extent that spare components are available. Reconfiguration is accomplished by providing a pre-selected set of redundant components for the most vulnerable portions of the system. After redundant components are exhausted, further faults result in a loss of functionality, thereby gracefully degrading the system performance. In our preliminary experiments, we have applied these techniques to a CMP router switch, and we found that the system is capable of tolerating a number of defects before any performance impact. Moreover, through the use of system-level error checking and recovery, it is possible to provide defect tolerance with much lower cost than traditional fault tolerance techniques, such as triple-modular redundancy (TMR).

4.2. Turnkey solutions

In the previous section we overviewed a range of BTWC design solutions. All these solutions are ad-hoc techniques that can be applied only to a specific domain. To the best of our knowledge no computer-aided design support is yet available to support the adoption of a BTWC design methodology. While generalized support is not yet foreseeable, some specialized CAD tools are easy to devise. In this section, we highlight a number of "turnkey" BTWC design technologies that we are currently pursuing. For instance, we can envision a software that can be used to "razorize" any transaction-based ASIC design. The software could insert the checker mechanism by substituting all the latches with razor-latches. The recovery could be very simple, by just inserting a voltage controller and a mechanism that upon the detection of a failure, flushes or flags as faulty the entire transaction.

A light-weight general DIVA solution could also be deployed as part of a design methodology that develops an optimized system through subsequent refinements of a high-level reference description. Because of the tight requirements for performance of the final systems, the optimizations cannot be generated automatically and are usually ad-hoc solutions put in place by the design team. Consequently, these high-level descriptions are commonly only used to validate the output of the optimized design. In this context, a DIVA checker can be obtained by synthesizing the high-level model to a simple non-optimized design component which has the same functionality of the core design, but lacks all of its optimizations. Since the checker's only requirement is correctness, the synthesis process can be in this case automatic. In order to overcome the performance limitations of the checker, this design can be pipelined so that each stage is small enough to execute as fast as the core system. The design team should still intervene to connect the pipeline stages of the checker to the proper internal signals of the core.

5. Conclusions

In this paper, we presented the concept of Better Than Worst-Case design, a novel design methodology that utilizes checker components to effectively separate the concerns of performance and correctness in complex designs. We highlighted our own efforts in this design space as well

as the efforts of others from the literature. Finally, we presented future applications we are pursuing, including technologies to support the deployment of Better Than Worst-Case design methodologies.

Acknowledgments. This work is supported by grants from NSF and the Gigascale Systems Research Center.

References

[1] A. Agarwal, V. Zolotov, and D. Blaauw. Statistical clock skew analysis considering intra-die process variations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(8):1231–1242, 2004.

[2] H. Al-Asaad and J. P. Hayes. Logic design validation via simulation and automatic test pattern generation. *Journal of Electronic Testing*, 16(6):575–589, 2000.

[3] A. Allan, D. Edenfeld, J. W. H. Joyner, and A. B. Kahng. 2001 technology roadmap for semiconductors. *IEEE Computer*, Jan. 2002.

[4] L. Anghel and M. Nicolaidis. Cost reduction and evaluation of temporary faults detecting technique. In *Proc. Design Automation and Test in Europe Conference (DATE)*, pages 591–598, 2000.

[5] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge. Opportunities and challenges for better than worst-case design. In *Proc. Asia South-Pacific Design Automation Conference (ASP-DAC)*, pages 12–17, Jan. 2005.

[6] T. Austin, D. Blaauw, T. Mudge, and K. Flautner. Making typical silicon matter with Razor. *IEEE Computer*, 37(3):57–65, 2004.

[7] T. M. Austin. DIVA: A reliable substrate for deep submicron microarchitecture design. In *Proc. International Symposium on Microarchitecture (MICRO)*, Dec. 1999.

[8] E. Bohl, T. Lindenkreuz, and R. Stephan. The fail-stop controller AE11. In *Proc. International Test Conference (ITC)*, pages 567–577, 1997.

[9] S. Chatterjee, C. Weaver, and T. Austin. Efficient checker processor design. In *Proc. International Symposium on Microarchitecture (MICRO)*, Dec. 2000.

[10] S. Das, S. Pant, D. Roberts, S. Lee, D. Blaauw, T. Austin, T. Mudge, and K. Flautner. A self-tuning dvs processor using delay-error detection and correction. In *Proc. Symposia on VLSI Technology and Circuits (VLSI)*, June 2005.

[11] P. Gupta and A. B. Kahng. Manufacturing-aware physical design. In *Proc. of the International Conference on Computer Aided Design (ICCAD)*, page 681, 2003.

[12] R. Hegde and N. R. Shanbhag. Energy-efficient signal processing via algorithmic noise-tolerance. In *Proc. of the International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 30–35, 1999.

[13] A. M. Ionescu, M. J. Declercq, S. Mahapatra, and K. B. a nd Jacques Gautier. Few electron devices: Towards hybrid CMOS-SET integrated circuits. In *Proc. Design Automation Conference (DAC)*, pages 88–93, 2002.

[14] T. Kehl. Hardware self-tuning and circuit performance monitoring. In *Proc. of the International Conference on Computer Design (ICCD)*, Oct. 1993.

[15] S. Lee, S. Das, V. Bertacco, T. Austin, D. Blaauw, and T. Mudge. Circuit-aware architectural simulation. In *Proc. Design Automation Conference (DAC)*, pages 305–310, 2004.

[16] T. Liu and S.-L. Lu. Performance improvement with circuit-level speculation. In *Proc. International Symposium on Microarchitecture (MICRO)*, pages 348–355, 2000.

[17] M. Mneimneh, F. Aloul, C. Weaver, S. Chatterjee, K. Sakallah, and T. Austin. Scalable hybrid verification of complex microprocessors. In *Proc. Design Automation Conference (DAC)*, pages 41–46, 2001.

[18] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proc. International Symposium on Microarchitecture (MICRO)*, pages 29–42, 2003.

[19] B. T. Murray and J. P. Hayes. Testing ICs: Getting to the core of the problem. *IEEE Computer*, 29(11):32–38, 1996.

[20] Overclockers Forum. Overclockers.com website.

[21] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice-Hall, 2003.

[22] D. P. Siewiorek and R. S. Swarz. *Reliable Computer Systems: Design and Evaluation, 3rd Edition*. A.K. Peters, Ltd. Publisher, 1998.

[23] J. H. Stathis. Reliability limits for the gate insulator in CMOS technology. *IBM Journal of Research and Development*, 46(2/3):265–286, 2002.

[24] A. Uht. Achieving typical delays in synchronous systems via timing error toleration. Technical Report TR-032000-0100, University of Rhode Island, Mar. 2000.

[25] N. J. Wang, J. Quek, T. M. Rafacz, , and S. J. Patel. Characterizing the effects of transient faults on a high-performance processor pipeline. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, pages 61–70, 2004.

[26] C. Weaver and T. Austin. A fault tolerant approach to microprocessor design. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, pages 411–420, 2001.

[27] F. Worm, P. Ienne, P. Thiran, and G. D. Micheli. An adaptive low-power transmission scheme for on-chip networks. In *Proc. of the International Symposium on System Synthesis (ISSS)*, pages 92–100, 2002.

[28] J. F. Ziegler. Terrestrial cosmic rays. *IBM Journal of Research and Development*, 40(1), 1996.