

Compilation for Delay Impact Minimization in VLIW Embedded Systems

José L. Ayala[†], David Atienza[‡], Praveen Raghavan^{*}, Marisa López-Vallejo[†], Francky Catthoor^{*}

[†]Dpto. de Ingeniería Electrónica/Universidad Politécnica de Madrid (Spain)

[‡]Dpto. de Arquitectura de Computadores y Automática/Universidad Complutense de Madrid (Spain)

^{*}IMEC, Leuven (Belgium)

Email: {jayala,marisa}@die.upm.es datienza@dacya.ucm.es {ragha,catthoor}@imec.be

Abstract

Tomorrow's embedded devices need to run high-resolution multimedia as well as need to support multi-standard wireless systems which require an enormous computational complexity with a very low energy consumption and very high performance constraints. In this context, the register file is one of the key sources of power consumption and performance bottleneck, and its inappropriate design and management can severely affect the performance of the system. In this paper, we present a new compilation approach to mitigate the performance implications of technology variation in the shared register file in upcoming embedded VLIW architectures with several processing units. The compilation approach is based on a redefined register assignment policy and a set of architectural modifications to this device. Experimental results show up to a 67% performance improvement with our technique.

1 Introduction

Recently, with the emerging market of new mobile wireless terminals that integrate multiple services such as multimedia and wireless network communications, the performance requirements can only be met using VLIW-like processors. Those forthcoming terminals will also include several heterogeneous processors as one of the most effective ways to tackle at the same time all the different multimedia services present in such systems. Some of these initial platforms start to be available today (e.g. ST Nomadik [22], Philips Nxpria [18], TI OMAP [25]). Unfortunately, the semiconductor industry is still facing several technological challenges to build these systems, specially due to their required low-power characteristics and the impact of process variations [8]. Moreover, very recently it has been found that, in new proposed embedded platforms with several processing elements, the shared register file plays a very important role as part of the memory subsystem because it can

heavily affect the cycle time and consumes a very significant portion of the total energy consumed by the memory hierarchy [25]). From the delay/performance point of view, the size achieved in the implementation of this device, as well as the technology variations introduced in deep sub-micron processes, cause different delays when accessing every register. Hence, it is crucial to reduce the energy spent on the register file as well as to reduce the performance implications of the existing delays.

With continued technology scaling, process variations have become a major factor that affects circuit performance and may lead to excessive yield loss. Variations come from various sources. Geometric process variations are most significant [7]. Effective channel length L_{eff} can vary more than 50% below 130nm technology node. The parameter variations of interconnects also increase beyond 25%. Supply voltage and temperature variations are also becoming more prominent in nanometer design [8]. These variations can lead to 30% or more die-to-die performance differences.

In this paper we introduce a new compilation approach to reduce the delay impact of technology variations in the shared register file of upcoming embedded architectures with several VLIW processors. In this work, we have used the compilation and simulation capabilities provided by the CRISP framework [13] to emulate a complex VLIW-based system with strong delay impact. The idea is to provide a mechanism to minimize the effect of technology variations in the delay of accessing different registers by the modification of the register assignment task.

Thus, our approach can either be used to increase the yield output of manufactured chips that do not fully reach the timing in the whole register file or can improve performance in general, since processors do not need to be bounded to a worst case, but on a case-per-case basis.

The structure of this paper is as follows: Section 2 presents the fundamentals and goals of our work, while the proposed approach is described in Section 3. The experimental work and results are covered in Section 4 and, finally, some conclusions are drawn.

2 Fundamentals and Goal

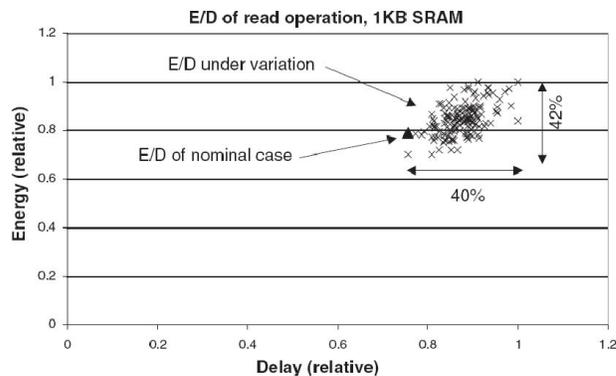
CMOS scaling trends are leading to a greater number of smaller and faster transistors in a single chip, but a relative increase in wire delay. Wire delays in memory structures are quite appreciable due to its size, and prevent them from being pipelined in a scaled manner [12]. The register file implemented in MPSoCs suffers this effect because of its large size and multi-ported architecture. Many times, the implemented register file in MPSoC systems is oversized in order to assure that the multi-threading code does not run out of registers.

As memory begins to dominate chip area in high performance applications, SRAM has become the focus of technology scaling [14]. Traditionally, SRAM cell size has scaled in accordance with technology ground rules; however, with the growing importance of variability, it is feared that this may no longer be possible. Because minimum size (gate length and width) devices are used to minimize cell area, SRAM is most susceptible to both process-induced variations in device geometry as well as threshold voltage variability due to dopant fluctuations [16]. In addition, the impact of variability is most pronounced in SRAM because cell operation, which depends upon well-matched FETs, must be satisfactory for each individual cell (no averaging across multiple stages as in logic). The fundamental concern of cell stability [20], which determines minimum array operating voltage and yield, has thus become increasingly difficult to address [10]. Moreover, the appreciable effect is the different delays to access every register in the register file. This effect is even worse when the delay introduced by the long wires is taken into account.

The analysis of the impact that intra-die stochastic variability has on the performance of CMOS processor blocks, such as ALUs, is very recent [11, 27]. For on-chip memories, the focus has been in functional yield and reliability issues, where it has been studied the use of SRAM cell stability (e.g. signal to noise margin) and design rules to compensate for performance issues [5, 15]. However, HW/SW methods to address variability problems in register files at compiler level as we propose here have not been considered yet.

Regarding reliability in future technologies, Single-Error-Correcting Multiple-Error-Detecting (SEC-MED) codes are already integrated in on-chip memories and some processor components [2, 6]. Also, Bowen et al. [9] showed that program behavior patterns can be used to generate custom-error correction mechanisms for memory portions. Nevertheless, these techniques are not suitable for register files since they impose significant area and energy consumption overheads.

In addition, system-level fault tolerant management mechanisms have been proposed for soft errors [2]. Shiv-



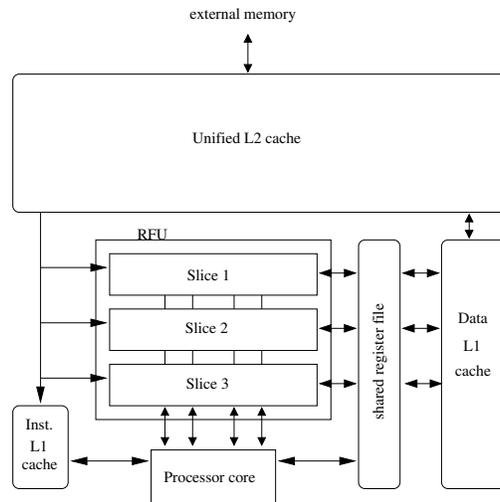
akumar2003 et al. [21] used redundant components and self-checking for embedded systems to increase resistance against hard errors and extend processor lifetime in case of process variation. Su et al. [24] presented self-repairing mechanisms via pre-existing processors. These works are complementary to our approach since they deal with functional variation in on-chip memories and processors, while we consider parametric variation in the register file.

Apart from anything else, the energy/delay cloud of the memory for read operation is shown in Figure 1 [28]. This figure shows how the delay tends to become bigger. This increase in delay mostly comes from longer decoder delay but, also from the cell array. This is the case for small size SRAMs, since the decoder contribution in the overall delay is significant [3]. As Figure 1 shows this is significantly affected by variability.

Taking these ideas into account, this paper presents a mechanism to minimize the effect of technology variations in the delay of accessing different registers. Again, this goal is accomplished with a hardware modification of the register file and the support of a power-aware compiler designed for this modified architecture. The information of technology variations is provided by the manufacturer and direct experimental characterization of the device.

3 Variability-Aware HW/SW Compilation Scheme

In this work, we have used the compilation and simulation capabilities provided by the CRISP framework [13]. It is a re-targetable compiler and simulator framework based on Trimaran [26], which is cycle accurate. The baseline architecture described by CRISP consists of a selectable number of processing elements as in VLIW processors, with a coarse-grained reconfigurable logic that can be adapted to



each desired DSP instruction set to be simulated. Furthermore, this framework also enables data and instruction clustering. The overall architecture of the type of VLIW processor that we use in this paper is shown in Figure 2.

As Figure 2 depicts, it includes a main processor core and coarse-grained reconfigurable logic, which is divided into slices. The main processor core can be any type of processor and it is included in the CRISP template architecture to be able to schedule instructions that in real-life VLIW systems cannot be executed efficiently (e.g. control and non-parallel operations). In our simulated architecture the main processor core is a simple RISC (Reduced Instruction Set Computer) core, but it is not relevant in our simulations since our case studies only include a very insignificant proportion of operations compared to DSP-like or loop operations, as we indicate in our experimental results (Section 4). Then, the slices for the configurable VLIW processing part are mapped onto the CRISP's Reconfigurable Functional Unit (RFU) part [13]. The RFU allows extensive customization of the VLIW processing units for the desired instruction set to be used in the simulated architecture within CRISP. In the RFU an operation can be issued every clock cycle. Regarding interconnections, the RFU reads/writes data from/to the main shared register file. Then, the additional RISC processor core and the RFU read their instructions from the level 1 instruction cache and move the data from the level 1 data cache to the shared register file. Finally, both types of caches are connected to a unified level 2 cache, which is in turn connected to an external memory.

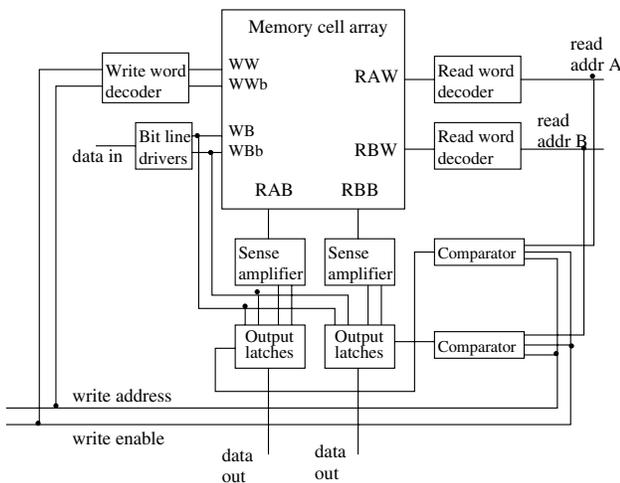
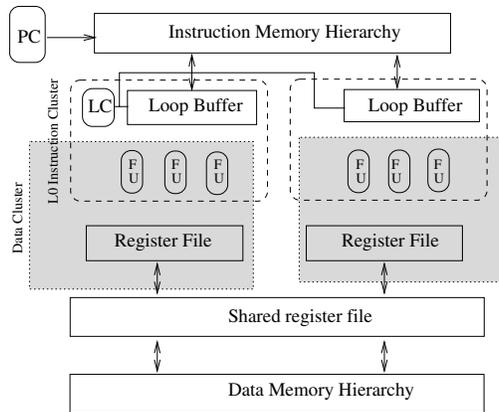
As we have previously mentioned, the RFU part of the emulated architecture is divided into reconfigurable slices (e.g. in Figure 2 three slices are depicted). Figure 3 details the internal structure of each slice. This figure shows that each slice contains several coarse-grained Functional Units

(FUs), which can be an ALU, shifter, multiplier or memory unit. As a matter of fact, such complex processing elements are better suited than the traditional logic blocks based on Look Up Tables (LUTs) for the execution of the operations typically found in multimedia applications, which are word-oriented and not bit-oriented. These complex FUs allow the reconfigurable logic to operate at higher frequencies with lower power consumption when compared to other possible embedded multi-processor architectures using traditional FPGAs [4].

In addition, each slice typically includes two instructions and data clusters. A data cluster includes a distributed register file, which can be used across multiple slots of functional units. Next, an instruction cluster is formed by using a distributed instruction buffer (or also called loop buffer [13]) across multiple issue slots of the VLIW processor. In fact, all the units of one data cluster can access the data register file in that cluster. However, accessing data from another data cluster (i.e. another local register file) is relatively expensive since it requires an explicit inter-cluster copy operation to transfer the data from the source data cluster to the destination data cluster. In contrast, all the functional units of a cluster can access any data present in that register file. Therefore, in order to provide enough bandwidth for all these potential concurrent accesses, our baseline architecture includes two read and one write port of the register file, which are allocated to each slice of the VLIW processor. All functional units in one slice are connected to these three ports via a full crossbar.

3.1 Architectural Modifications

In a typical configuration, the register file is an array of N words by M bits. Any of the N words can be simultaneously



accessed by two read ports and a write port. Figure 4 shows that the register file contains seven distinct types of functional blocks [23], namely, the memory cell array, the read word decoders, the write word decoder, the bit line drivers, the sense amplifiers, the output latches, and the comparators. In this design, the memory cell array stores the data bits, and is arranged in a grid of N rows by M columns of memory cells. When any of the ports accesses the array, the read/write operation is performed simultaneously on every memory cell in the selected row.

Technology variations during the device manufacturing can create delays in any component shown in Figure 4. We characterize the delay incurred by every register in the register file as those meeting the timing requirement of the whole pipeline (the register can be accessed in one clock cycle), and those that require more than one clock cycle to be accessed.

In our approach, the overall write/read delay of each reg-

ister in the register file with respect to the original specification is coded into the HW architecture using some extra bits per register (in our notation, label bits). In case 1-bit label is used, a '0' would indicate the register satisfies the timing requirement and '1' would assume that it does not (therefore we would have to assume a worst case of 4 cycles delay). In case a 2-bit label is used, then a finer grain labelling can be done. A '00' would assume that the register satisfies 1 cycle delay, '01' would mean that the read would take 2-cycle latency and so on. This classification is done in an initial characterization phase after production and requires the area overhead of extra timing circuitry and storage of the label bits. It can be assumed that the overhead due to the timing circuitry would not be large and can be neglected. Furthermore, this circuitry is activated only once after fabrication time; thus, both its dynamic and leakage power overhead can be neglected because it can be completely gated from the power source (V_{dd}) during the normal processor execution. Then, the storage overhead for the label bits has been calculated using a UMC *90nm* technology [1] and our results indicate that it is very limited with respect to the total area of the baseline register file (see Section 4). It is assumed that there is no process variation that happens on these extra bits. This is because of two assumptions: the extra bits required are small, hence probability of any variation on this structure is low; the small register file can be overdesigned so that it is more tolerant to process variation without much overhead.

Finally, note that the granularity used to classify the registers (i.e. number of sets) enables a trade-off between the accuracy desired in the classification and the number of extra label bits needed. This trade-off is illustrated in Section 4 with several real-life multimedia benchmarks.

3.2 Compiler Modifications

To make profit of the HW architecture including label bits, we have modified the Trimaran compiler of the CRISP framework. Our new compiler includes a different register assignment algorithm for the underlying VLIW processor. The register assignment phase of compilers determines which register/s will be used for each program value selected during the register allocation phase, which is the phase that determines which values are placed in registers. In fact, our new compiler is aimed at in-order processors, since out-of-order execution could destroy this first assignment by using HW mechanisms to avoid hazards (e.g. register renaming [26]).

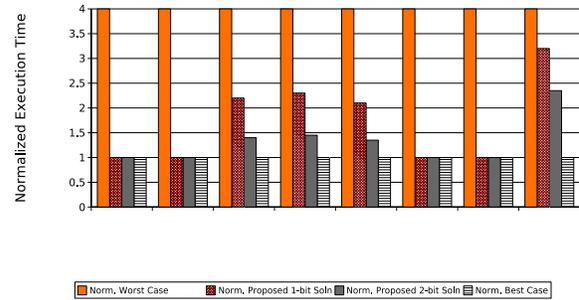
Traditionally, register assignment algorithms choose registers from the whole set of free registers without any constraint. In the case of Trimaran, as many other compilers, it retrieves the first register from a First In First Out (FIFO) list of free registers. In fact, the order of registers inside the list is not representative and only depends on the underlying HW architecture. Hence, since no restriction on selecting the registers exists, the assigned registers in the original register assignment can easily belong to different *delay sets*.

Our register assignment policy modifies this original register assignment by first selecting the registers labelled as faster registers. The new compiler initializes internally FIFO lists for each delay set in the register file by a first reading phase of the stored label bits, and performs a *variability-aware* assignment giving preference to registers of the fast sets to be used for the loops of multimedia applications. These applications are usually loop-dominated [17] and the execution of loops seriously affect the overall processor performance.

The compiler-based technique we have presented does not have a negative impact in the area, performance or power consumption of in-order processors and manages effectively the speed variation found in the register file of these architectures (see section 4). However, as every static approach, it requires the modification of the sources for every target processor since the profile of access delays varies between each instance of the final target architecture. This limitation makes difficult its extended use. Therefore, we have also designed two additional run-time HW techniques to solve these limitations at the potential cost of reducing system performance [19] (depending on the performance gains achieved by the effective register renaming). These run-time techniques, when compared with the compilation approach presented here, do not reach so optimal results.

4 Case Studies and Experimental Results

We have applied the proposed approach to several applications of the MediaBench suite [17]. They are the



following:

- *adpcm_decode*: audio decoder that uses adaptive differential pulse code demodulation, suited for embedded systems.

- *g721_decode*: an implementation of the G.721 voice decompression standard.

- *mesa_texgen*: an implementation of the Mesa 3D library (OpenGL clone) that generates a texture-mapped Utah teapot.

- *aes*: an implementation of the Advanced Encryption Standard (AES), a worldwide encryption standard since 2001.

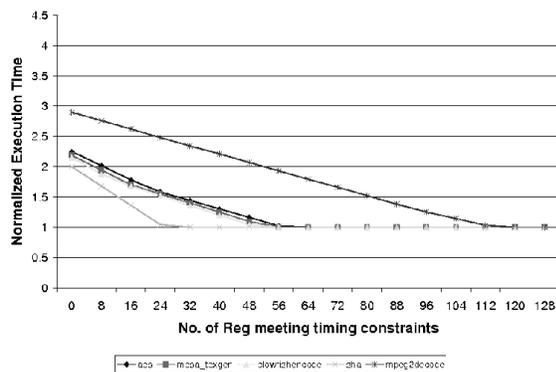
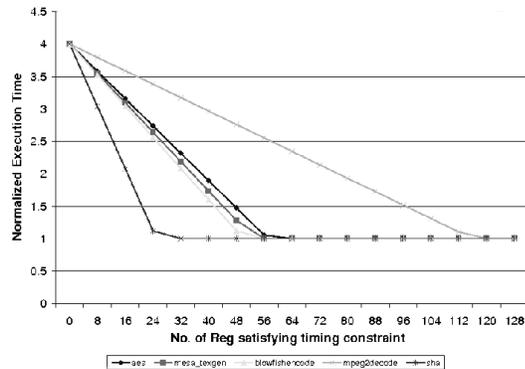
- *blowfishencode*: an implementation of the popular 64-bit blocks de/encryption algorithm.

- *epic*: an image compression utility that uses a dyadic wavelet decomposition and a run-length/Huffman entropy coder.

- *sha*: implementation of the Secure Hash Algorithm (SHA-1 Hash) used in wireless embedded terminals.

- *mpeg2decode*: implements the decoder of the MPEG2 standard for digital video transmission.

The CRISP framework was used to simulate a 32-bit, 4-issue VLIW processor, including 12 ports (8 read and 4 write) 128 entries deep register file. The area of the original register file was $3.04 \times 10^5 \mu\text{m}^2$ using the TSMC 90nm standard cell synthesis process. We assumed that process variations of access time delays in the whole pole of registers follows a homogeneous distribution, as suggested by [28].



The register file was annotated to inject such a distribution. Therefore, when the benchmark compilation process has finished in CRISP, the percentage of utilization of the register file can be obtained. Then, after the execution of the benchmarks, CRISP provides us the number of registers that meet the performance constraints and overall performance results with the injected timing variations (based on the register allocation).

4.1 Experiment 1

In a first set of simulations we have studied the normalized execution time when 75% of the registers do not meet the timing constraint of 1 cycle of operation, namely in this case running at 300 MHz, they do not respond in less than 3ns. The execution time has been normalized to this best

case, where all registers can react in one cycle. We have assumed that 32 out of 128 registers meet the performance constraints, namely, they can be read/written in one cycle, while 32 registers require 2 cycles, 32 require 3 cycles, and the last 32 require 4 cycles. No further process variation is considered. Any register that requires more than 4 cycles can be marked un-usable.

Figure 5 shows the normalized performance results in terms of average number of clock cycles to access the register file using one bit or two bits for the label bits. Also, this figure shows the results for the worst case simulation (i.e. all registers belong to the slowest set with 4 cycles of latency) and the best case simulation. These results indicate that our compiler based approach achieves the optimal point of the best case without variability in four of the benchmarks. Furthermore, the results are 57% better than the worst case for the other benchmarks on average with the 1-bit labelling and 67% better in case of the 2-bit labelling. Thus, our approach can either be used to increase the yield output of manufactured chips that do not fully reach the timing in the whole register file or can improve performance in general since processors do not need to be bounded to a worst case, but on a case-per-case basis.

In addition, the results shown in Figure 5 for large benchmarks with strong register pressure, like *aes* and *blowfishencode*, show that they always need to use registers that do not meet the timing constraints. Therefore, the solutions using two bits for the label bits (i.e. classifying the registers into four delay sets) achieve much better results than using only one label bit classification (more than twice worse than the best case). This occurs because in the case of 1-bit classification, the worst case delay has to be used for all the registers that do not meet the original time requirements as only two sets exist for the types of registers, while with two bits the registers are graded with a finer granularity and hence less performance penalties are applied. The only benchmark that does not benefit to the same extent with any of these configurations of label bits is *mpeg2dec*, since it demands a large amount of registers during the whole execution. Therefore, for this benchmark in particular we have performed additional experiments to study the potential benefits of increasing the number of bits of the label bits. Our experiments indicated that it was necessary to use 5 or more bits to obtain results significantly better than the case of 2 bits. Due to the dramatic area overhead incurred by the use of 5 or more label bits, these results outline the existence of trade-offs between area and system performance in case of very tight system requirements.

4.2 Experiment 2

In a second set of experiments we have evaluated possible trade-offs between the area overhead of label bits and

Technique	Area (in μm^2)	Power (in μW)	Avg. Performance (w.r.t Worst Case)	Compile Time
Compiler-based (2-bit)	1.16×10^3	0	67%	Compile needed for every chip

achievable performance if the number of registers that fail the timing constraints varies (Figure 6 and Figure 7). Our results indicate that the optimality in the number of label bits (1-bit or 2 bits) is enormously determined by the variability in the register file and the desired performance. Also, another important trend that can be observed is the much steeper degradation of the execution time in the case of 1-bit label registers comparing to the two-bit solution. This effect occurs due to the more fine-grained exploitation of the register delay with 2 bits, similarly as the performance effect explained before for the largest multimedia benchmarks. Moreover, these results indicate that using 2-bit for the label bits already achieves the best case performance bound in almost all tested embedded multimedia applications, assuming a variability of up to 39% (i.e. 50 registers responding in 1 cycle). It was seen that when more than 50% of the registers satisfy the register timing requirement, most benchmarks provide the required performance. But for benchmarks with high register pressure, the best performance can only be reached with all the registers satisfying the timing requirement.

The area overhead, power overhead and performance gains of the proposed technique are summarized in Table 4.1. It can be seen how the approach does not present any power overhead, while the area of the extra logic is very reduced. The approach requires the compilation for every target chip, but the savings obtained exceed expectations.

5 Conclusions

New consumer applications have recently increased in complexity and demand a very high-level of performance in the next generation of low-power embedded devices. Therefore, new techniques and mechanisms that can provide solutions for an efficient mapping of these complex applications in such platforms are in great need. One of the most important factors of power consumption and performance penalty is the shared register file between all processing units. In this paper we have presented and shown with realistic examples the applicability of a new hardware/software approach that combines a set of architectural extensions to achieve important reductions in the energy of the shared register file in upcoming embedded architectures with several VLIW processors. Also, the delays included by technology vari-

ations are characterized and their effects are minimized by the accurate management performed by the compiler. Our results indicate that this new integral approach enables a reduction of the energy consumed in the register file, as well as an optimization of its size and performance, of such forthcoming embedded architectures without a great impact in area or logic complexity.

Acknowledgements

This work is partially supported by the Spanish Government Research Grants TIC2003-07036 and TIN2005-05619.

References

- [1] UMC. <http://www.umc.com>.
- [2] V. Agarwal, S. Keckler, and D. Burger. The effect of technology scaling on microarchitectural structures. Technical report, Technical Report TR2000-02, University of Texas at Austin, USA, 2002.
- [3] B. S. Amrutur and M. A. Horowitz. Speed and power scaling of SRAM's. *IEEE Trans. Solid-State Circuits*, 35(2), February 2000.
- [4] D. Atienza, P. G. D. Valle, G. Paci, F. Poletti, L. Benini, G. D. Micheli, and J. M. Mendias. A Fast HW/SW FPGA-Based Thermal Emulation Framework for MultiProcessor System-on-Chip. In *Design Automation Conference*, 2006.
- [5] A. J. Bhavnagarwala, X. Tang, and J. D. Meindl. The impact of intrinsic device fluctuations on CMOS SRAM cell stability. *IEEE J. Solid-State Circuits*, 36(2):18–31, 2001.
- [6] M. Blaum, R. Goodman, and R. McEliece. The reliability of single-error protected computer memories. *IEEE Trans. Comput.*, 37(1):114–119, 1988.
- [7] D. Boning and S. Nassif. *Design of High-Performance Microprocessor Circuits*, chapter Models of Process Variations in Device and Interconnect. Wiley-IEEE Press, 2000.
- [8] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *Design Automation Conference*, 2003.
- [9] N. S. Bowen and D. K. Pradhan. The effect of program behavior on fault observability. *IEEE Trans. Comput.*, 45(8):868–880, 1996.
- [10] L. Chang et al. Stable SRAM cell design for the 32 nm node and beyond. In *Symposium on VLSI Technology*, 2005.

- [11] T. W. Chen and J. Gregg. A Low Cost Individual-Well Adaptive Body Bias (IWABB) Scheme for Leakage Power Reduction and Performance Enhancement in the Presence of Intra-Die Variations. In *Proceedings of DATE*, pages 240–245, 2004.
- [12] Z. Chishti and T. N. Vijaykumar. Wire delay is not a problem for SMT (in the near future). In *International Symposium on Computer Architecture*, 2004.
- [13] P. O. de Beeck, F. Barat, M. Jayapala, and R. Lauwereins. CRISP: A Template for Reconfigurable Instruction Set Processors. In *FPL*, pages 296–305, 2001.
- [14] D. M. Fried et al. Aggressively scaled ($0.14\mu\text{m}^2$) 6T-SRAM Cell for the 32 nm node and beyond. In *International Electron Devices Meeting*, 2004.
- [15] R. Heald. Managing variability in SRAM designs. In *In Proceedings International Solid-State Circuits Conference (ISSCC) uP Forum*, February 2004.
- [16] R. W. Keyes. Effect of randomness in the distribution of impurity ions on FET thresholds in integrated electronics. *IEEE Journal of Solid-State Circuits*, 10(4):245–247, August 1975.
- [17] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of International Symposium on Microarchitecture (MICRO)*, pages 330–335, 1997.
- [18] Philips Nexperia - Highly integrated programmable system-on-chip (MPSoC), 2004. <http://www.semiconductors.philips.com/products/nexperia/>.
- [19] P. Raghavan, J. L. Ayala, D. Atienza, F. Catthoor, M. López-Vallejo, and G. D. Micheli. Reduction of Register File Delay Due to Process Variability in VLIW Embedded Processors. Technical report, IMEC, 2006.
- [20] E. Seevinck, F. J. List, and J. Lohstroh. Static-noise margin analysis of MOS SRAM cells. *IEEE Journal of Solid-State Circuits*, 22(5):748–754, October 1987.
- [21] P. Shivakumar, S. W. Keckler, C. R. Moore, and D. Burger. Exploiting microarchitectural redundancy for defect tolerance. In *ICCD '03: Proceedings of the 21st International Conference on Computer Design*, page 481, Washington, DC, USA, 2003. IEEE Computer Society.
- [22] ST Nomadik Multimedia Processor, 2004. <http://www.st.com/stonline/prodpres/dedicate/proc/proc.htm>.
- [23] S. A. Steidl. *A 32-Word by 32-Bit Three-Port Bipolar Register File Implemented Using a SiGe HBT BiCMOS Technology*. PhD thesis, Rensselaer Polytechnic Institute, 2001.
- [24] C.-L. Su, R.-F. Huang, and C.-W. Wu. A processor-based built-in self-repair design for embedded memories. In *Proceedings of the 12th Asian Test Symposium (ATS)*, 2003.
- [25] OMAP Platform, 2004. <http://focus.ti.com/omap/docs/>.
- [26] Trimedia Technologies Inc. Trimaran: An infrastructure for research in instruction-level parallelism, 1999. <http://www.trimaran.org>.
- [27] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. In *Proceedings of DAC*, pages 331–336, 2004.
- [28] H. Wang, M. Miranda, W. Dehaene, F. Catthoor, and K. Maex. Systematic analysis of energy and delay impact of very deep submicron process variability effects in embedded SRAM modules. In *Design Automation and Test in Europe*, 2005.