Process Variation Aware Issue Queue Design *

Raghavendra K and Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology-Madras
Chennai - 600036, India
raghavendra83@gmail.com, madhu@cse.iitm.ac.in

Abstract

In sub-90nm process technology it becomes harder to control the fabrication process, which in turn causes variations between the design-time parameters and the fabricated parameters. Variations in the critical process parameters can result in significant fluctuations in the switching speed and leakage power consumption of different transistors in the same chip.

In this paper, we study the impact of process variation on issue queues. Due to process variation, issue queues can take variable access latency. In order to work with non-uniform access latency issue queues, by exploiting ready operands of instructions at dispatch time, we propose a process variation aware issue queue design. Experimental results reveal that, for a 64-entry issue queue with half of the entries affected by process variation, our technique recovers most of the lost performance due to process variation and incurs a performance penalty of less than 2% with respect to the performance of issue queues without process variation.

1 Introduction

In the advanced process technologies, it is very difficult to control the fabrication process. The fabricated parameters can be different from the design-time parameters. The major factors leading to process variation are wafer misalignment, random dopant fluctuations, and imperfections in planarizarion steps. Variations in the critical process parameters such as threshold voltage or effective channel length can result in significant fluctuations in the switching speed and leakage power consumption of different transistors in the same chip. For instance, parameter variations in 130nm technology cause a 30% variation in the maximum allowable frequency of operation and a fivefold increase in leakage power [5].

Parameter variations are classified as *process variation* due to manufacturing phenomena, *voltage variations* due to manufacturing and runtime phenomena, and *temperature variations* due to activity and power dissipation variations. While the voltage and temperature variations are runtime phenomena, process variation is static and manifest itself as *die-to-die* (DID) and *with-in-die* (WID) variations. WID variation can be further divided into *random* and *systematic* variations, where random variation is small change from transistor to transistor and systematic variation exhibits spatial correlations. WID variation can reduce the performance gains of deeper pipelines [22].

Process variation is severe in memory components as minimum sized transistors are used for density reasons [15]. Critical path delay in memory components is mainly dictated by sensing operations and the variable sense current produced by the minimum sized transistors causes a large timing variations of output data arrival [12]. The usual practise of designing for worst-case process margins may not be viable for future designs because of the degree of variability encountered in the advanced process technologies.

In this paper, we study process variation effects on *issue queues*, a performance-critical component of superscalar processors [14]. Here we consider CAM/RAM based issue queue and assume that the CAM cells take non-uniform search times due to process variation and hence the issue queue takes variable access latency. In order to work with non-uniform access latency issue queues, by exploiting ready operands of instructions at dispatch time, we propose a process variation aware issue queue design. We validate our techniques by running SPEC2000 CPU benchmark suite [3] on Simplescalar simulator [2] and show that, for a 64-entry issue queue with half of the entries affected by process variation, our technique recovers most of the lost performance due to process variation and incurs a performance penalty of less than 2% with respect to the performance of issue queues without process variation.

*This work was supported in part by grant from Department of Science and Technology(DST), India, Project No. SR/S3/EECE/80/2006.

978-3-9810801-3-1/DATE08 © 2008 EDAA

2 Related Work

As process variation can significantly impact performance, several circuit-level and architectural-level techniques have been proposed in the literature to minimize the performance loss. Circuit-level techniques such as *adaptive body biasing* [17] and *adaptive supply voltage* [26] are proposed to reduce the clock speed variability due to DID and WID variations. Though these techniques are effective, they add complexity to the manufacturing process, increase leakage power, and reduce reliability.

As process variation makes some of the pipeline stages slower and hence reduces the maximum frequency attainable by the pipeline, an architectural technique, called *ReCycle* [7], is proposed to transfer the time slack of the faster pipeline stage to the slower stages.

A variation-aware cache architecture [4] is proposed to avoid faults by adaptively resizing the cache and hence improve the yield. For process variation affected data caches, an architectural technique, called *variable-latency cache architecture* [23], is proposed in which extra buffers are placed before the functional units to support different load accesses with varying latencies.

By identifying wakeup/select logic path as a critical bottleneck, a dependence-based issue queue design is proposed in [24], which replaces the conventional issue queue with a set of FIFOs. As instruction issue takes place through the FIFO heads, the wakeup/select logic latency scales with the number of FIFOs rather than the number of issue slots. Dispatch logic makes sure that all dependent instructions are placed in the same FIFO.

By exploiting the fact that most instructions have one or two ready operands at dispatch time, an issue queue design is proposed in [10], which consists of three separate issue queues, i.e., without CAM logic for instructions whose operands are ready; CAM logic for instructions with one non-ready operand; CAM logic for instructions with both operands non-ready at dispatch time. By using these separate issue queues along with the *last-tag speculation*, a large fraction of tag comparisons are eliminated from the schedulers critical path. As some of the instructions may have both operands ready at the time of dispatch or require only one operand, a technique to disable wakeup logic for empty entries and ready operands is proposed in [11] to minimize dynamic power consumption.

Other works such as two-level issue queue design [20], half-price architecture [13] and sub-banking issue queue design [25] are proposed in literature.

3 Issue Queue

In order to support out-of-order execution, modern day superscalar datapaths include a number of components.

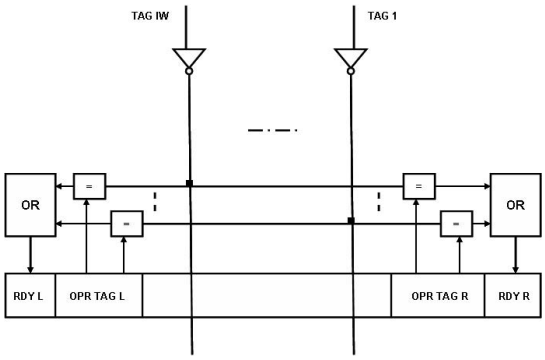

Figure 1. Issue queue entry [14].

One such component is *issue queue*. The issue queue stores the instructions waiting for execution, identifies whether operands are ready through *wakeup logic*, and selects instructions for execution whose operands are ready and whose required resources are available through *select logic*.

After an instruction is fetched, it is decoded and dispatched to the issue queue irrespective of the availability of its input register operands. An instruction waits in the issue queue until its input operands are ready. When an instruction is executed, the result tag of the instruction is broadcasted to all the instructions waiting in the issue queue. The wakeup logic compares the result tag with the input operand tags of each instruction in the issue queue and if it finds a match, the corresponding ready bit is *set*. When both operands of an instruction are ready, a request is made to the select logic for execution. Based on the scheduler policy, the select logic chooses the next instructions to execute from all ready instructions. The selected instructions upon receiving a grant signal from the select logic are sent forward to later stages in the pipeline.

A common way to implement issue queues is based on CAM/RAM structures [24]. The RAM cells store operations, destination operands, and *ready bits* indicating whether source operands are ready, while the CAM cells store source operand *tags*. As all source operand tags in the issue queue are searched in a fully associative manner to know which operands become ready for all the instructions in the issue queue [14, 24], the CAM cells are used to store the source operand *tags*. The main idea behind using CAM to store the source operand tags is that it implements fully associative search in a single clock cycle using dedicated comparison circuitry [18].

Figure 1 shows the structure of an issue queue entry based on the CAM/RAM structure. Each entry can accommodate two operands by providing a tag location and a ready bit for each operand. Every result tag is connected to all the operand tags in the issue queue as shown in Figure 1. In each cycle, a maximum of issue width (IW) number of result tags are compared with the operand tags in the issue

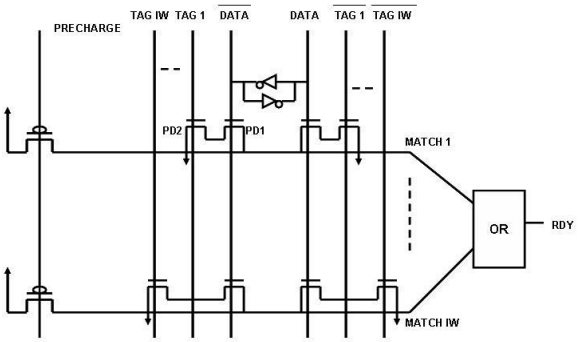

Figure 2. CAM cell wakeup logic [24].

queue. CAM cell wakeup logic is shown in Figure 2. Each cycle, all the match lines are precharged and then they are conditionally discharged based on the tag values stored in the CAM cell. For detailed hardware implementation of the issue queue, one can refer to [24].

4 Process Variation Aware Issue Queue

In this paper, we consider an issue queue based on CAM/RAM structure and assume that the wakeup logic of the issue queue is affected by process variation. We assume that the threshold voltage (V_{th}) variation is the major source of WID variation as the effect of other parameter variations can be translated as effective variations in V_{th} [4]. Note that CAM is compared every cycle and RAM containing the payload area of the issue queue is accessed only during the read operation when the instruction is dispatched on to the functional unit. Hence we take into consideration the CAM logic being affected by process variation when proposing our techniques as it plays a very crucial role in the performance of the issue queue and the system as a whole.

As wakeup logic is implemented using CAM cells, we discuss the impact of process variation on CAM cells. A CAM cell is mainly classified as either NOR-type or NAND-type CAM cell [18] and it has a bit storage and bit comparison circuitry. The CAM cells use SRAM cells for bit storage. Process variation results in the mismatch in the strengths of the different transistors in a CAM cell. Due to such device mismatches, different type of failures can occur in the CAM cell, which include *search time failure*, *match failure*, and *SRAM bit failure* [4, 6]. The SRAM bit failures are further classified as *read failure*, *write failure*, and *access time failure* [4].

The CAM cell search time is defined as the time taken for a pre-specified voltage drop on the matchline. Due to process variation, the CAM cell search time can vary significantly [6]. By assuming that the deviation in V_{th} follows a normal distribution with 20% standard variation, we evaluated the NOR-type CAM design (implemented in $65nm$ process technology) using Monte Carlo simulations in HSPICE and obtained a 10.5% variation in the search time of NOR-type CAM cell. Based on this analysis we assume that the access latency of process variation affected entry is 2 cycles and that of non-process variation affected entry is 1 cycle. In the base case we consider an issue queue which is not affected by process variation.

In order to work with variable latency issue queues, one can consider the following worst-case design scenarios:

- **CycleTimeIncrease (CTI):** The clock cycle time is increased based on the worst-case access time. Here, we consider 10.5% increase in the clock cycle.
- **MultipleCycles (MC):** The original clock cycle time is maintained but the issue queue is accessed in multiple clock cycles. Here, all entries of the issue queue take an access latency of *two* cycles.
- **DisablingAffectedSpace (DAS):** Disable an entry of the issue queue permanently if it contains at least one process variation affected tag locations so that the issue queue becomes a queue without process variation at the cost of reduced number of entries.

Increasing the clock cycle time has negative impact on the overall pipeline performance as it increases the execution time, while the *MC* scheme has a problem of *skipping tag comparisons*. As results of executed instructions in the pipeline are produced in every cycle, the result tags have to be compared with the source operand tags of instructions present in the issue queue. If the access latency of the issue queue is *two* cycles (in the case of *MC* scheme), some of the produced results may not be compared with the operand tags. In other words, if the issue queue is accessed in every *odd* cycles (because of two cycle latency), all the results produced in *even* cycles cannot be compared with the operand tags of instructions present in the issue queue. In order to facilitate tag comparison in every cycle, we need to split the issue queue into two sub-queues of equal entries such that both these queues store same data and one queue facilitates tag comparison in *even* cycle while the other queue facilitates tag comparison in *odd* cycle. Whenever we dispatch an instruction to one sub-queue, the instruction is also dispatched to the other sub-queue. Though the access latency of the issue queue takes *two* cycles, because of tag comparison in every cycle, the throughput remains same as that of the issue queue without process variation. As splitting the issue queue into two parts results in 50% reduction in the number of entries, the MC technique incurs significant performance penalty especially when only few entries of the issue queue are affected by process variation. The MC technique performs better when the number of entries affected by process variation is more than 50% of the issue queue size, while the DAS technique performs better when the number of entries affected by process variation is less than 50% of the issue queue size.

We now propose a process variation aware design technique to minimize the performance loss. In order to sup-

Parameter	Value
RUU size	64 instructions
LSQ size	32 instructions
Machine width	8 wide fetch, 8 wide issue, 8 wide commit
Functional Units	8 Integer ALUs, 4 Integer multiply/divide, 8 FP add, 4 FP multiply, 4 FP divide/sqrt
L1 Data cache	64K, 4-way (LRU), 64B blocks, 2 cycle latency
L1 Inst. cache	64K, 4-way (LRU), 32B blocks, 1 cycle latency
L2 cache	Unified, 512KB, 8-way (LRU), 128B blocks, 12-cycle latency
Memory	160 cycles
ITLB	16-entry, 4KB block, 4-way, 30-cycle miss penalty
DTLB	32-entry, 4KB block, 4-way, 30-cycle miss penalty
Branch predictor	Combined, Bimodal 2K table, 2-Level 1K table, 8 bit history, 4K choser
BTB	1K-entry, 4-way
Return-address stack	8
Mispredict penalty	18 cycles

Table 1. Our default configuration parameters

port our technique, we consider a modified version of the wakeup logic by considering a *PV* bit for each operand tag location and adding a 2-input AND gate to the precharge signal. If the *PV* bit is *reset* for an operand tag location, the operand tag location is said to be affected by process variation. We initialize the *PV* bits using the March test [19]. The March test is employed during the functional testing of memory components and it can distinguish the low and high latency components. It applies a sequence of operations that read and write values of 0s and 1s to different memory locations. The March test can also be used to identify the process variation effects such as destructive readout failures [21]. We characterize issue queue operand tag locations as either low or high latency locations through such test that captures access time failures before the operational phase of a microprocessor and initialize the corresponding *PV* bits. The inputs to the AND gate are the original precharge signal and the *PV* bit, which is associated with every operand tag location. The output of the AND gate will drive the precharge transistors of each match line. If the *PV* bit is *reset* for an operand, the precharge operation for all the match lines of the ready operand is disabled, the corresponding *ready* bit is *set*. Note that each entry of the issue queue has two extra bits PV_1 and PV_2, one for each tag location.

- **ProcessVariationAwareIssueQueue (PVAIQ):** Instructions with one or two ready input operands are stored in the process variation affected entries of the issue queue. If no process variation affected entries are found, the instructions are steered onto the non process variation affected space.

Configuration	Number of entries		
	Partially affected	Fully affected	Clean
Config1	8	8	48
Config2	16	16	32
Config3	8	40	16

Table 2. Different configurations of partially and fully affected entries of the issue queue.

At dispatch time, we check whether an instruction is a 0, 1, or 2 operand instruction and whether zero or more of its operands are ready. Note that whenever an instruction with one or two ready operand(s) is dispatched to an entry in the issue queue, it is not necessary to access the corresponding operand tag locations of the entry to write the ready input operand tags and compare the operand tags with the result tags. Exploiting reduced tag comparators for instructions with zero or one non-ready operands is already explored in the context of power-performance efficient dynamic scheduling [10]. In order to minimize the performance loss due to process variation, we exploit disabling wakeup logic for process variation affected operand tag locations to steer instructions with ready operands.

Through out the paper, we assume that an issue queue entry is said to be *fully affected* if both operand tag locations are affected by process variation; *partially affected* if only one operand tag location affected by process variation; and *clean entry* if none of the tag locations are affected by process variation. Based on the above assumption, the issue queue is logically partitioned into four sub-queues, one sub-queue with clean entries ($IQ_{1,1}$), two sub-queues with partially affected entries ($IQ_{1,2}$ and $IQ_{2,1}$), and one sub-queue with fully affected entries ($IQ_{2,2}$).

If an instruction with two ready operands is encountered, we steer it to an entry of $IQ_{2,2}$, provided there is a free entry in $IQ_{2,2}$. As *PV* bits are *reset* initially for fully affected entries, the precharge operation for the matchlines is disabled and hence the wakeup logic for the operand tag locations is disabled. Since each operand tag location has an issue width (IW) number of matchlines (one matchline for every result tag), we disable all the match lines of an operand tag location. Note that disabling the wakeup logic for issue queue entries is already explored in the context of dynamic power minimization [11]. As wakeup logic for process variation affected operand tag location is disabled, it has no impact on the CAM cell search time, and hence the worst-case search time of CAM cells remains *one* cycle. If there is no free entry in $IQ_{2,2}$, the instruction is steered onto $IQ_{1,1}$.

If an instruction with one non-ready operand is encountered, we steer it to either $IQ_{2,1}$ or $IQ_{1,2}$ in such a way that the source tag of the non-ready operand is written onto the *clean* tag location of the issue queue entry. The wakeup logic of the other tag location is disabled as its *PV* bit is *reset*. If there is no free entry available in the corresponding $IQ_{2,1}$ or $IQ_{1,2}$, the instruction is steered onto $IQ_{1,1}$.

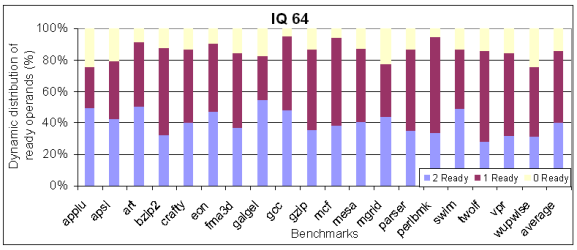

Figure 3. Percentage runtime distribution of ready input operands for issue queue.

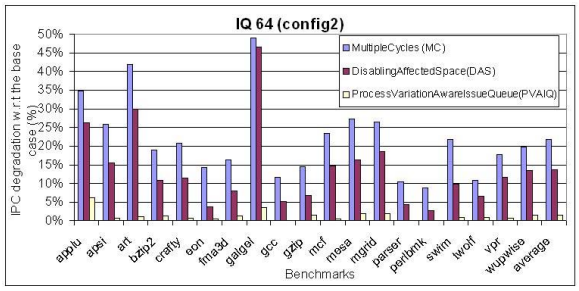

Figure 4. Performance degradation of various techniques w.r.t. the base case.

Disabling the wakeup logic for the affected tag locations does not effect the operation of the issue queue as the corresponding *ready* bits are *set* initially. For any instruction with two non-ready operands, we always steer it to $IQ_{1,1}$.

Note that the PV_1 and PV_2 bits of all the entries of the issue queue are associatively searched in parallel to find a free entry along with the register renaming and checking the status of the source physical registers. The steering algorithm is a bit more complicated when compared to the traditional designs but there is no extra delay involved as the searches occur in parallel. Similar complexity issues are discussed in various techniques like *instruction packing* [16] and *dynamic scheduling through tag elimination* [10]. The *instruction wakeup* and the *instruction selection* logic remain the same as that of the traditional designs except that the wakeup logic has reduced tag comparisons.

5 Experimental Validation

5.1 Experimental Setup

The baseline processor configuration used in our experiments is given in Table 1. We validate our technique by simulating 19 SPEC2000 CPU benchmarks [3] using the Simplescalar 3.0 simulator [2]. For each benchmark, we fast-forward 1 billion instructions and then simulate the next 500 million instructions. As part of sensitivity analysis, we conduct experiments by assuming different configurations of partially affected and fully affected entries of the issue queue as shown in Table 2. In the config1 and config2 process variation cases, we assume equal number of partially affected and fully affected entries, while in the config3 process variation case, to model the worst case affected space, we assume more number of fully affected entries as compared to the partially affected entries.

5.2 Experimental Results

Figure 3 shows benchmark-wise percentage distribution of 0, 1, and 2 input operand ready instructions. Percentage of 0, 1, and 2 input operand ready instructions, on an average, is 14.3%, 45.4%, and 40.3%, respectively, for a 64-entry issue queue. These results are similar to the results presented in [10]. PVAIQ technique exploits this high percentage of instructions with 1 or 2 ready input operands to minimize performance penalty due to process variation by steering such instructions onto partially or fully affected issue queue entries.

Figure 4 shows benchmark-wise IPC degradation for different techniques with respect to the base case with config2. Performance degradation of the MC technique is very significant and it ranges from 8.7% ("perlbmk") to 49.1% ("galgel"). High performance penalty for the MC technique in the case of "galgel" and "art" is due to the fact that the performance of "galgel" and "art" is heavily dependent on the issue queue size. As a result even the DAS technique performs poorly in these benchmarks. The PVAIQ technique reduces the performance penalty with respect to the base case to a very large extent as compared with the other two techniques and the performance penalty ranges from 0.22% ("gcc") to 6.24% ("applu"). Notable performance penalty for the PVAIQ technique in the case of "applu" can be attributed to two factors. One is that the performance of "applu" is heavily dependent on the issue queue size. Second reason is clearly visible from Figure 3, where "applu" has the highest percentage of non-ready instructions (24.8%).

Figure 5 shows average performance degradation of different techniques with respect to the base case for issue queue with different configurations of process variation affected entries. The performance degradation for the MC technique remains same for different configurations of the affected space. For the DAS technique the performance degradation varies from 5.52%("config1") to 29.9%("config3"). The DAS techniques has poor performance when compared to the MC technique in config3 due to the fact that there are lesser number of clean entries. In the case of config2 even though both MC and DAS techniques have the same number of issue queue entries the DAS technique gives a better performance when compared to the MC technique because the MC technique deals with the affected space also for which writing onto the issue queue takes 2 cycles. In the case of PVAIQ technique

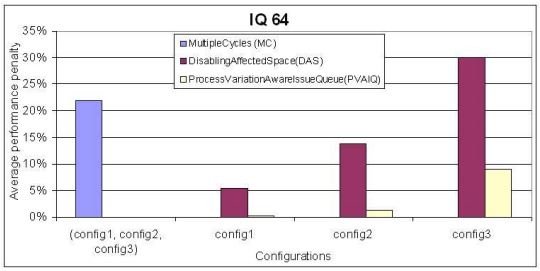

Figure 5. Average performance degradation of various techniques w.r.t. the base case.

the average performance degradation varies from as less as 0.28% ("config1") to a maximum of 8.9% ("config3") for an issue queue size of 64. Overall, the PVAIQ techniques incurs a very small performance penalty (less than 2%) for a 64-entry issue queue with half of the entries being process variation affected as compared to the MC (21.8%) and DAS (13.8%) techniques.

For sensitivity analysis, we also conducted experiments by considering issue width of 4 and 128-entry issue queue and we observed that our technique significantly reduces the performance loss due to process variation. For an issue width of 8 and 128-entry issue queue the performance penalty is 0.02%("config1"), 0.11%("config2"), 2.84%("config3"), for an issue width of 4 and 64-entry issue queue it is 0.34%("config1"), 1.31%("config2"), 6.18%("config3") and for an issue width of 4 and 128-entry issue queue it is 0.02%("config1"), 0.05%("config2"), 2.22%("config3").

6 Conclusions and Future work

In this paper, we proposed an issue queue design to minimize the performance penalty due to process variation. Experimental results reveal that our technique can significantly reduce performance penalty incurred due to access latency variations. As part of the future work we would like to study the effect of process variation on the RAM(payload) part of the issue queue.

References

[1] Semiconductior Industry Association. International Technology Roadmap for Semiconductors, 2005. http://public.itrs.net.

[2] SimpleScalar toolset. http://www.simplescalar.com.

[3] SPEC 2000 Benchmark. http://www.spec.org.

[4] A. Agarwal, *et. al.* "Process variation in embedded memories: failure analysis and variation aware architecture". *JSSC*, 40(9), 1804-1814, 2005.

[5] A. Datta, *et. al.* "Speed binning aware design methodology to improve profit under process variations". *ASPDAC*, 712-717, 2006.

[6] A. Mupid, *et. al.* "Variation analysis of CAM cells". *ISQED*, 333-338, 2007.

[7] A. Tiwari, *et. al.* "ReCycle: pipeline adaptation to tolerate process variation". *ISCA*, 2007.

[8] B.R. Fisk and R.I. Bahar. "The non-critical buffer: using load latency tolerance to improve data cache efficiency". *ICCD*, 538-545, 1999.

[9] D. Burger and T.M. Austin. "The SimpleScalar Tool Set, Version 2.0". University of Wisconsin, Madison.

[10] D. Ernst and T. Austin. "Efficient dynamic scheduling through tag elimination". *ISCA*, 37-46, 2002.

[11] D. Folegnani and A. Gonzalez. "Energy-efficient issue logic". *ISCA*, 230-239, 2001.

[12] I. Arsovski and R. Wistort. "Self-referenced sense amplifier for across-chip-variation immune sensing in high-performance content-addressable memories". *CICC*, 453-456, 2006.

[13] I. Kim, *et. al.* "Half-Price Architecture". *ISCA*, 2003.

[14] J. Abella and A. Gonzalez. "Low-complexity distributed issue queue". *HPCA*, 73-82, 2004.

[15] J. Croon, *et. al.* "Physical modeling and prediction of the matching properties of MOSFETs". *ESSDERC*, 193-199, 2004.

[16] J. Sharkey, *et. al.* "Reducing delay and power consumption of the wakeup logic through instruction packing and tag memoization". *PACS*, 2004.

[17] J. Tschanz, *et. al.* "Adaptive body bias for reducing impacts of die-to-die and within-die variations on microprocessor frequency and leakage". *JSSC*, 37(11), 1396-1402, 2002.

[18] K. Pagiamtzis *et. al.* "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey". *JSSC*, 41(3), 712-727, 2006.

[19] M.L. Bushnell, *et. al. Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*, Kluwer, 2000.

[20] P. Michaud, *et. al.* "Data-flow prescheduling for large instruction windows in out-of-order processors". *HPCA*, 27-36, 2001.

[21] Q. Chen, *et. al.* "Modeling and testing of SRAM for new failure mechanisms due to process variations in nanoscale CMOS". *VTS*, 292-297, 2005.

[22] S. Borkar, *et. al.* "Parameter variation and impact on circuit and microarchitecture". *DAC*, 338-342, 2003.

[23] S. Ozdemir, *et. al.* "Yield-aware cache architectures". *MICRO'06*, 15-25, 2006.

[24] S. Palacharla. "Complexity effective superscalar processors". PhD Thesis, University of Wisconsin-Madison, 1998.

[25] S.E. Raasch, *et. al.* "A scalable instruction queue design using dependence chains". *ISCA*, 318-329, 2002.

[26] T. Chen *et. al.* "Comparison of adaptive body bias (ABB) and adaptive supply voltage (ASV) for improving delay and leakage under the presence of process variation". *TVLSI*, 11(5), 888-899, 2003.