A Critical Analysis of Application-Adaptive Multiple Clock Processors*

Emil Talpes, Diana Marculescu
{etalpes, dianam}@ece.cmu.edu
Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213

ABSTRACT

Enabled by the continuous advancement in fabrication technology, present day synchronous microprocessors include more than 100 million transistors and have clock speeds well in excess of the 1GHz mark. Distributing a low-skew clock signal in this frequency range to all areas of a large chip is a task of growing complexity. As a solution to this problem, designers have recently suggested the use of frequency islands that are locally clocked and externally communicate using mixed timing communication schemes. Such a design style fits nicely the recently proposed concept of voltage islands that, in addition, can potentially enable fine grain dynamic power management. This paper proposes a design exploration framework for **application-adaptive multiple clock processors** which provides the means for analyzing and identifying the right inter-domain communication scheme and the proper granularity for the choice of voltage/frequency. In addition, the proposed design exploration framework allows for comparative analysis of newly proposed or existing application-driven dynamic power management strategies. Such a design exploration framework and accompanying results can help designers and computer architects in choosing the right design strategy for achieving better power-performance trade-offs in multiple clock high-end processors.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles – *clocking strategies, multi-voltage circuits, application-adaptive processor.*

General Terms

Algorithms, Management, Performance, Design, Experimentation

Keywords

Microarchitecture, Multi Clock Processors, Globally Asynchronous Locally Synchronous, Dynamic Voltage Scaling, Simulation Framework.

1 Introduction

The last few decades have been dominated by Moore's Law, with performance being the primary driving force in processor design. This trend has lead to a vast increase in the number of transistors used in modern microprocessors, while constantly pushing clock frequencies. Unfortunately, this has also resulted in a huge increase in power dissipation as well, with current processors already dissipating more than 100 Watts. One major design bottleneck in today's high-performance VLSI systems is the clock distribution network. Large clock nets perform like very long signal paths, making it hard for designers to keep the clock skew within tolerable limits. With increasingly large processor dices and clock frequencies in the multi-gigahertz range, ever more complex clocking schemes are needed [1]. Furthermore, the clock skew is getting worse with each shrink due to the increasingly high process and system parameter variation.

*This research was supported in part by SRC Grant No. 2001-HJ-898 and by NSF CAREER Award No. CCR-008479.

To address these problems, two approaches are possible. The first option is to use fully asynchronous designs. While this has been tried successfully in isolated cases [2][3][4], the design methodology for asynchronous design is far from being mature and thus, far from widespread acceptance.

Another alternative is to use globally asynchronous, locally synchronous (GALS) architectures [5][6][7][8], which attempt to combine the benefits of both fully synchronous and asynchronous systems. A GALS architecture is composed of synchronous blocks that communicate with each other using an asynchronous communication scheme. Such systems have important advantages, as they do not require a global clock distribution network or de-skewing circuitry. Using locally generated clock signals within each individual domain, they make it possible to take advantage of the industry-standard synchronous design methodology. However, the overhead introduced by communicating data across clock domain boundaries may become a fundamental drawback, limiting the performance of these systems. Thus, the choice of granularity for these synchronous blocks or islands must be very carefully done in order to prevent the inter-domain communication from becoming a bottleneck. At the same time, the choice of the inter-domain communication scheme, as well as of the on-the-fly mechanisms for per-domain dynamic speed/voltage scaling become critical when analyzing overall power-performance trends.

The contribution of the work proposed in this paper is twofold:

- First, we propose a design exploration framework for application-adaptive multiple clock high-end processors that is fully parameterizable and allows for detailed analysis of available power/performance trade-offs.
- Second, this paper introduces a new type of dynamic control strategy for application-adaptive multiple clock processors that matches the voltage/speed for various frequency/voltage islands to the ones required by the application workload.

The paper is organized as follows: in Section 2 we present previous work related to the aspects addressed in this paper. In Section 3 we present the baseline microarchitecture under consideration. Section 4 details our simulation framework and the experimental setup, while the results of all the tests are presented in Section 5. Finally, we conclude the paper with an analysis of the trends observed in the experimental results together with our conclusions and directions for future research.

2 Related Work

With high clock frequencies driving an ever-increasing number of transistors available on-chip, the power burned in the clock distribution network starts to become a limitation. In [9], the authors identify the clock distribution circuitry as a primary source of power consumption. An approach that allows for aggressive future frequency increases, maintains a synchronous design methodology, and exploits the trend towards making functional blocks more autonomous is the *Globally Asynchronous, Locally Synchronous* clocking style [5][12]. Several VLSI designs that can benefit from such a technique were proposed [10][11]. All of them are based on the observation that in these specific cases the communication performance is not critical.

Previous studies focused on assessing the viability of a GALS clocking strategy to a superscalar, out-of-order processor. The performance and power consumption of such a processor are evaluated in [7]. While performance is worse than in the fully synchronous case (with an average of 10%), the paper identifies the ability of the GALS processor to use different clock frequencies and supply voltages for each synchronous island. The same idea of scaling the clock frequency and the supply voltage is studied in [8], concluding that such processors can be more power efficient than their fully synchronous counterparts. The paper proposes an algorithm based on the attack-decay strategy, which can select an optimum voltage and clock frequency out of a large number of possible levels.

Another area of research related to the proposed design exploration framework addresses mixed-clock interface design for robust communication among frequency islands. A number of mechanisms for avoiding race conditions are evaluated in [13]: asynchronous wrappers, stretchable clock generators, demand ports, poll ports. While these designs can ensure a proper behavior of the system, they introduce some performance penalties when compared to their synchronous counterparts. To address this issue, asynchronous queues were proposed in [14]. This mechanism does not improve the communication latency but it increases the bandwidth, allowing data transfers on each clock cycle.

3 Baseline microarchitecture

In this paper, we start with a fairly typical out-of-order, superscalar microarchitecture and analyze the impact of various design decisions on the power and performance of the GALS processor. To this end, we assume a 10-stage pipeline implementing a 4-way processor. While this pipeline is significantly longer than the ones studied in [7] or [8], we feel that this increased length resembles the pipelines that are currently implemented in commercial processors more accurately. The underlying microarchitecture organization is shown in Figure 1.

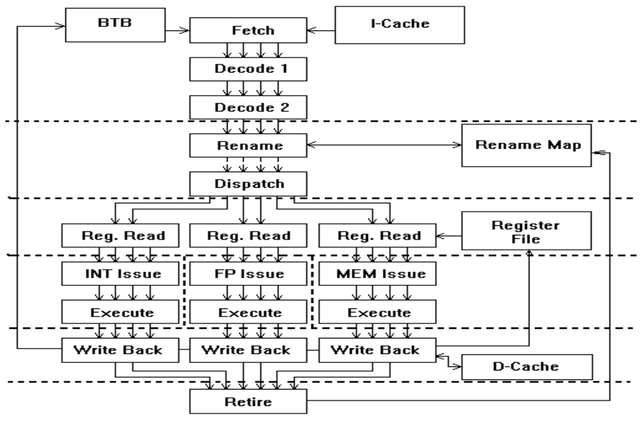

Figure 1. The baseline microarchitecture

4 GALS microarchitecture design knobs

Of extreme importance for the proposed design exploration framework is the choice of various design knobs that impact the overall power-performance trade-offs in GALS processors. The following design choices are of interest for such processors:

- The choice of the asynchronous communication scheme
- The granularity chosen for the frequency islands.
- The dynamic control strategy for adjusting voltage/speed of clock domains so as to achieve better power efficiency.

4.1 Clock domain granularity

To assess the impact of introducing a mixed-clock interface inside our pipeline, we have to break it into several synchronous blocks. The natural approach – minimize communication over the synchronous blocks boundaries – does not necessarily work here. An instruction must pass through all the pipeline stages in order to be completed. Thus, we have to find other criteria for partitioning.

One possible criterion for placing the asynchronous interfaces is to minimize clock skew, allowing for faster local clocks. Our results [17][1] show that the most significant speed-up can be achieved by increasing the clock speed in the Fetch or Memory, followed by Integer and FP partitions. Thus, these modules should be placed in separate clock domains if possible.

[1] Due to the page limit, we were unable to include detailed results for this analysis and the newly proposed dynamic control mechanism.

For the execution block, we have used the same partitioning scheme proposed in [7] and [8]. Starting with a processor with separate clusters for integer, floating point and memory execution units (much like the Alpha 21264 design) we can naturally separate these clusters in 3 synchronous modules. The drawback of this scheme is that it increases the latency of forwarding a result to another functional unit. This can effect be seen mainly in the overall latency of load-use operations that are executed in separate clock domains.

To limit the latency of accessing register data, the Register Read and the Write Back stages must be placed together, in the same synchronous partition as the Register File. Using the same logic, the Rename and Retire stage access the Rename Table and the list of available registers so they must be placed in the same partition.

Thus, we can now split the pipeline into at least 4 clock regions. The first one is composed of the Fetch stage, together with all the branch prediction and I-cache. The Decode stages can be included either in the first clocking region or in the second one – all instructions will be passing through both of them. In order to limit the load capacitance variations [15] inside synchronous blocks, we introduce the boundary after the Decode stages. The second clocking region will be organized around the Renaming mechanism, containing the Reorder Buffer and Retire logic. Given the variation in the pipeline register width, an asynchronous boundary can be also introduced after Dispatch. The third clocking region must be organized around the Register File, containing Register Read and Write Back. Finally, the out-of-order part of the pipeline (Issue logic and execution) is split into clusters that amount for 3 different clock regions. The forwarding paths can thus be internal – towards a unit of the same type, placed in the same clock region – or external – towards other clock regions.

4.2 Inter-domain communication scheme

The conventional scheme to tackle such problems is the extensive use of synchronizers - a double latching mechanism that conservatively delays a potential read, waiting for data signals to stabilize. This makes classical synchronizers rather unattractive, as their performance diminishes and the probability of failure for the whole system rises with the number of synchronized signals.

Pausable clocks have been proposed as a scheme that relies on stretching the clock periods on the two communicating blocks until the data is stable or the receiver is ready to accept it [6]. While the latency is smaller, stretching the clock is reflected in the performance of the synchronous blocks and thus can be applied on a per cycle basis only when the two blocks use a similar clock frequency.

Another approach is to use arbiters for detecting any timing violation condition (i.e. the write-to-read time is smaller than a certain threshold). While the mechanism is conceptually similar to that of synchronizers, it offers a much smaller latency.

Asynchronous FIFO queues were proposed [14], using either synchronizers or arbiters. This approach works well under the assumption that the FIFO is neither completely full, nor completely empty. The scheme retains the extra latency introduced by the use of synchronizers, but improves the bandwidth through pipelining. During nominal operation, a read is serviced by a different cell than the one handling the next write, without the need of further synchronization.

4.3 Dynamic control strategy

One of the main advantages offered by the GALS approach is the ability to run each synchronous module at a different clock frequency. If the original pipeline stages are not perfectly balanced, the synchronous blocks can naturally be clocked at different frequencies. Furthermore, even with a perfectly balanced design (resizing the transistors to speed-up the longer signal paths), we can slow down synchronous blocks that are off the critical path while keeping the others running at nominal speed. The slower clock domains could also operate at a lower supply voltage, thus producing additional power savings. Since energy is quadratically dependent on V_{dd}, reducing it can lead to significant energy benefits: $D \approx V_{dd} / (V_{dd} - V_t)^{\alpha}$ where D is the logic delay, V_{dd} is the supply voltage, V_t is the threshold voltage and α is a technology dependent constant (currently, 1.2-1.6).

Different schemes have been previously proposed for selecting the optimal frequency and voltage supply. In [7], a simple threshold-based algorithm is used for selecting the best operating point for modules that have a low power mode. It monitors the occupancy of each issue window, deciding to switch to the low power mode when this occupancy drops below a predefined threshold, or ramp the voltage up when a high-threshold is exceeded. A more complex model is proposed in [8]. Here, an attack-decay algorithm selects the operating point for processors that offer a wide range of frequencies and supply voltages. It monitors the instruction window occupancy and, based on its *variation*, decides whether the frequency should be increased or decreased. Any significant variation triggers a rapid change of the clock frequency in order to counter it. Otherwise, the clock frequency is decayed continuously, while monitoring the overall performance.

Looking at each of these methods, we note that instruction window occupancy may not be the only significant aspect to be considered for deciding a switch. Even though an issue window has high occupancy, this could be due to a bottleneck in another cluster. In this case, *inter-domain dependencies* may be more significant than the window occupancy. Furthermore, both [7] and [8] allow dynamic voltage scaling for the execution core only, assuming the speed of the front-end to be critical for the overall performance. However, there are portions of the code where the achievable IPC is significantly smaller than the theoretical pipeline throughput. In these cases, it makes sense to reduce the speed of the front-end since it produces more instructions than can be processed by the back-end. In order to study the efficiency of these observations, we propose to modify the previously described methods to include both information about *cross-domain dependencies* and dynamic scaling capabilities for the *front-end* of the pipeline [17].

5 Simulation Framework

To measure the impact of our GALS microarchitecture, we have simulated a cycle-accurate model of the original pipeline (Figure 1). Our simulator is based on SimpleScalar [15], but reflects the target pipeline more accurately. It uses normal pipeline registers, separate Instruction Windows for each execution cluster and a Retire Buffer. Register Renaming is similar to the MIPS R10000 processor. We have also moved the execution from Decode (as it is done in SimpleScalar) to the Execute stage, to better reflect the behavior of the pipeline. To model a GALS environment without global synchronization points, we developed an event-driven simulation engine. Events associated with each frequency island are synchronized with a local, randomly started clock signal. This event-driven simulation engine allows for any mixture of clocks speeds and starting phases.

We have used Wattch [16] for including power models in our framework. These models (including the ones for the asynchronous communication) are integrated in our baseline and GALS simulator versions to provide energy statistics. Similar to [16], unused units are modeled as consuming 10% of their normal in-use power consumption to account for all the overheads associated with clock gating.

In addition to modeling the switching capacitance of memories and buses inside the processor, we have also modeled the clock grids. We have assumed a clock distribution hierarchy resembling the one used by the Alpha 21264 processor. We have modeled one global clock grid and local clock grids corresponding to each of the synchronous domains. The area and metal density for each clock grid are the ones published for the 21264 processor.

Table 1. Microarchitecture parameters

Parameter	Value
Pipeline	10 stages, 4 way out-of-order
Instruction Window	64 entries – 32 Int, 16 FP, 16 Mem
Load / Store Queue	32 entries
I-Cache	32k, 2 ways, 1 cycle hit time
D-Cache	32k, 4 ways, 2 cycles hit time
L2 Cache	Unified, 256k, 4 way set-associative, 10 cycles
Memory access time	100 cycles
Functional Units	4 Integer ALUs, 2 Integer MUL/DIV 2 Memory ports 2 FP Adders, 1 FP MUL/DIV
Branch Prediction	G-share

The parameters for our test microarchitecture are presented in Table 1. We have used integer and floating-point benchmarks from both SPEC95 and SPEC2000 suites. For all experiments, we have fast-forwarded over the first 500 million instructions and then continued simulation for another 50 million. Since clock signals are randomly staggered in the GALS cases, simulations were run three times averaging the results. To compare the various strategies, we have used arbiter-based FIFOs with a synchronization time of 30% of the smallest cycle time. To have a fair comparison, we used similar frequency levels for all methods, further divided into 8 sub-levels for the attack-decay algorithm.

6 Experimental results

We have analyzed the impact of the granularity and asynchronous communication choice, as well as the impact of the dynamic control strategy on the overall performance and power. Assuming the communication mechanism uses FIFOs, a penalty of up to 16% is observed for 6 clock domains. As shown in Figure 2, the performance hit increases with the number of asynchronous interfaces – from an average of 7% for 4 clock domains to almost 11% for 6 clock domains. In all cases, the baseline is the fully synchronous microarchitecture.

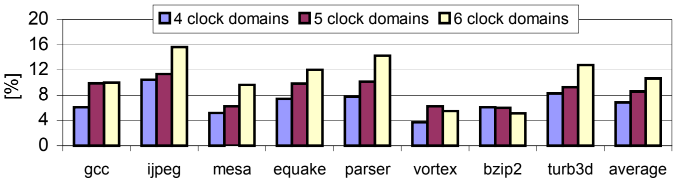

Figure 2. Performance degradation for a GALS microarchitecture

In terms of power consumption, the GALS processor is more efficient due to its lack of global clock grid. However, due to the increase in execution time, even though the power per cycle is significantly improved, the total energy per task required by the GALS processor can actually be higher (Figure 3). Since a 6 clock domain architecture would only allow an independent speedup in the register file, it seems that the best choice is a 5 clock domain design which allows the Fetch and Execution to run at possibly different speeds.

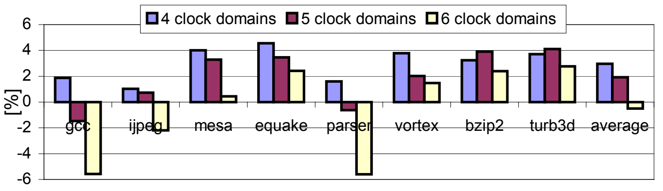

Figure 3. Energy reduction for a GALS design for 4-6 partitions

To evaluate the effectiveness of each asynchronous communication scheme, we have considered arbiter-based and synchronizer-based FIFOs, as well as pausable clocks. For both arbiters and pausable clocks, we assume that an additional interval of 0.3 cycles is needed for ensuring correct synchronization. Since pausable clocks effectively delay an active clock edge when the synchronization cannot be done, the effective producer-to-consumer latency of this approach is 0.3 cycles. In the arbiters approach however, a failed synchronization is followed by a normal consumer cycle, completely unsynchronized with respect to the producer clock. This introduces an additional average delay of 0.5 cycles, bringing the total latency to 0.8 cycles. For similar reasons, the 1 cycle latency associated with the synchronizers is actually 1.5 cycles when coupled with random starting phases for the producer and consumer clocks. As expected, the worst performance corresponds to synchronizers. In this case, the performance hit for a 5-clock domain design can be up to 25%, with an average of 17.7%. The smallest hit in performance is achieved when using pausable clocks, with an average of 4.8%.

None of the 3 mechanisms brings a significant energy reduction. While a small gain can be noticed for pausable clocks and arbiters (3.6% and 1.9% on average), the use of synchronizer-based FIFOs leads to an increase of 2.6% in the energy demands (Figure 5).

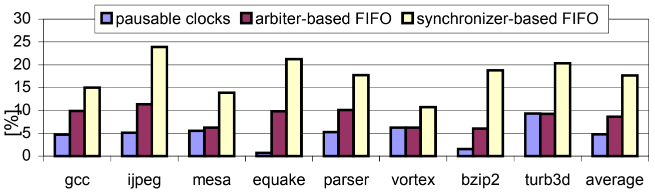

Figure 4. Performance degradation using different mechanisms for the asynchronous communication

As expected, our results show that the mechanism introducing the smallest additional latency (pausable clocks) is the best both in terms of performance and energy. However, by delaying the clock signal in the consumer to observe the synchronization latency, pausable clocks do not allow the use of different speeds across domains. Thus, in order to be able to implement dynamic control of local speeds/voltages, we have to use FIFOs based on either arbiters or synchronizers.

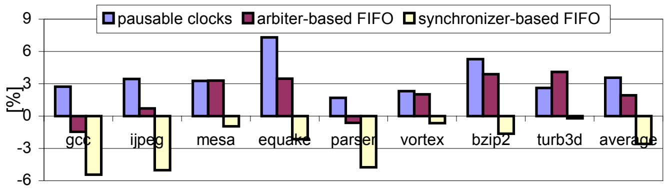

Figure 5. Energy reduction for a GALS design using different asynchronous communication mechanisms

One advantage of the GALS architectures is the ability to scale the voltage and clock speed independently, for each of its synchronous partitions. We evaluate both the performance and the energy using two previously proposed algorithms: the threshold-based one that can select the best out of two operating points [7] and the attack-decay algorithm that assumes a much larger set of operating points [8]. For both of them, we test the efficiency of focusing on the average Instruction Window occupancy (threshold TO and attack-decay ADO) or on the number of inter-domain dependency (TD and ADD).

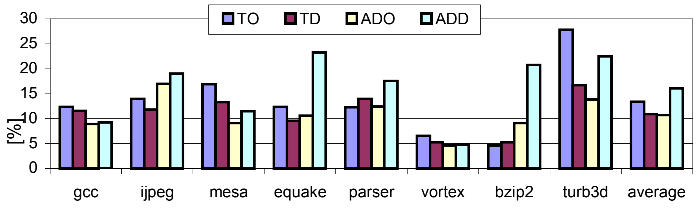

Figure 6. Performance of the DVS-enabled GALS design

All the dynamic control mechanisms introduce an average drop in performance of 10.5% to 16% when compared to the synchronous baseline architecture. A very interesting aspect is that the inter-cluster dependency information does not improve performance in both cases. While TD performs better than TO, in the case of the attack-decay the additional information actually decreases the performance. This behavior is caused by a significant variance of the communication across decision windows. In the case of the threshold-based approach, this variance is covered by the hysteresis of the algorithm and large variations can trigger a frequency change. In the case of the attack-decay algorithm on the other hand, smaller variations are taken into consideration because of the finer speed/voltage control.

In terms of energy requirement, the DVS-enabled GALS design saves between 10% and 37% (Figure 7). On average, the savings is between 20% and 25% for the four DVS algorithms that we study, thus making it possible to achieve better power efficiency when compared to a synchronous, DVS-enabled counterpart.

7 Conclusion

In this paper, we propose a simulation framework that allows for rapid evaluation of the different design choices available when implementing a GALS processor microarchitecture.

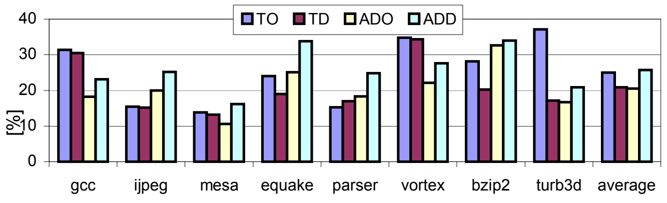

Figure 7. Energy consumption of the DVS-enabled design

By using this framework, we can evaluate the power and performance achieved by several GALS implementations of a superscalar, out-of-order processor. Our results show that asynchronous interfaces introduced between the several synchronous modules can have a very significant effect, varying from 5% when using pausable clocks to almost 18% when using synchronizers. However, even though pausable clocks seem to be a better choice, they do not allow communicating modules to run at different clock frequencies. Thus, for implementing DVS we have to select another mechanism – asynchronous FIFOs based on arbiters in our case. In terms of power consumption, the GALS design paradigm does not offer a significant benefit when dynamic voltage scaling is not implemented. The reduced clock power is offset in this case by the additional runtime needed to finish the same computation. By using DVS however, an average energy reduction of up to 25% can be achieved at the expense of a 10 to 15% reduction in performance.

8 References

[1] N. Kurd, J. Barkatullah, R. Dizon, T. Fletcher, and P. Madland. Multi-GHz Clocking Scheme for Intel Pentium 4 Microprocessor, in Proceedings of the International Solid-State and Circuits Conference, February 2001.

[2] S. Furber, J. Garside, and D. Gilbert. AMULET3: A High-Performance Self-Timed ARM Microprocessor, in Proceedings of the International Conference on Computer Design, September 2000.

[3] K. Stevens, S. Rotem, R. Kol, C. Dike, and M. Roncken. An Asynchronous Instruction Length Decoder, in Proceedings of the Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems, April 1999.

[4] A. Martin, A. Lines, M. Nystrom, P. Penzes, R. Southworth, U. Cummings, and T. K. Lee. The Design of an Asynchronous MIPS R3000 Microprocessor, in Proceedings of the 17th Conference on Advanced Research in VLSI, September 1997.

[5] T. Meincke, A. Hemani, S. Kumar, P. Ellervee, J. Oberg, D. Lindqvist, H. Tenhunen, and A. Postula. Evaluating benefits of Globally Asynchronous Locally Synchronous VLSI architecture, in Proceedings of the 16th Norchip, November 1998.

[6] J. Muttersbach, T. Villiger, N. Felber, and W. Fichtner. Globally Asynchronous Locally Synchronous Architectures to Simplify the Design of On-Chip Systems, in Proceedings of the 12th IEEE International ASIC/SOC Conference, September 1999.

[7] A. Iyer and D. Marculescu. Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors, in Proceedings of the International Symposium on Computer Architecture, May 2002.

[8] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, and M. L. Scott. Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling, in Proceedings of the International Symposium on High Performance Computer Architecture, February 2002.

[9] V. Tiwari, and F. Baez,.Reducing Power in High-performance Microprocessors, in the Proceedings of the 35th Design Automation Conference, June 1998.

[10] C. Jesshope and A. Shafarenko. Asynchrony in Parallel Computing – A question of Scale, in Proceedings of the International Conference on Massively Parallel Computing Systems, April 1998.

[11] L. Bengtsson and B. Svensson. A Globally Asynchronous, Locally Synchronous SIMD Processor, in Proceedings of the International Conference on Massively Parallel Computing Systems, April 1998.

[12] R. Ronen, A. Mendelson, and J.P. Shen. Coming Challenges in Microarchitecture and Architecture, in Proceedings of the IEEE, Vol 89, No. 3, March 2001.

[13] J. Muttersbach, and W. Fichtner. Practical Design of Globally-Asynchronous Locally Synchronous Systems, in Proceedings of the 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems, April 2000.

[14] T. Chelcea and S. Nowick. Robust Interfaces for Mixed Systems with Application to Latency-Insensitive Protocols, in Proceedings of the Design Automation Conference, June 2001.

[15] P. Zarkesh-Ha, and J.D. Meindl. Characterization and Modeling of Clock Skew with Process Variations, IEEE Custom Integrated Circuit Conference, May 1999.

[15] D. Bourger, T. Austin, and S. Bennet. Evaluating Future Microprocessors: the SimpleScalar Tool Set, Technical Report 1308, University of Wisconsin, July 1996.

[16] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, in Proceedings of the International Symposium on Computer Architecture, June 2000.

[17] E. Talpes and D. Marculescu, Application-Adaptive Multiple Clock Processors, Technical Report, Carnegie Mellon University, July 2003