VEBoC: Variation and Error-Aware Design for Billions of Devices on a Chip

Shoaib Akram, Scott Cromar, Gregory Lucas, Alexandros Papakonstantinou, Deming Chen

Electrical and Computer Engineering Department
University of Illinois, Urbana-Champaign
e-mail: {sakram3, scromar2, gmlucas2, apapako2, dchen}@uiuc.edu

Abstract – Billions of devices on a chip is around the corner and the trend of deep submicron (DSM) technology scaling will continue for at least another decade. Meanwhile, designers also face severe on-chip parameter variations, soft/hard errors, and high leakage power. How to use these billions of devices to deliver power-efficient, high-performance, and yet error-resilient computation is a challenging task. In this paper, we attempt to demonstrate some of our perspectives to address these critical issues. We elaborate on variation-aware synthesis, holistic error modeling, reliable multicore, and synthesis for application-specific multicore. We also present some of our insights for future reliable computing.

I. Introduction

The continuing progression of transistor-integrating technologies into the DSM scale is offering great opportunities for designing big and increasingly smart System-on-Chips (SoC) and multicore chips. At the same time it is posing enormous challenges to the engineers in IC design and manufacturing, as well as CAD. It is critical that the challenges be understood and addressed at both levels to enable acceptable performance, reliability, yield, and cost in next generation chips. These challenges can be summarized in the following paragraphs.

Process variation. Figure 1 shows the significant variability trends in a number of process parameters [1]. Transistor parameters such as channel length, gate-oxide thickness, and threshold voltage vary due to limitations in lithographic techniques, dopant variation, variations in chemical polishing due to non-uniform layout density, and layout dependent stress variation [2]. The results of these variations on the final product can include lower yield, lower reliability, high power usage, and lower than expected performance.

Design reliability. Besides process variation, a host of other issues, such as manufacturing defects, cosmic event upsets (soft error) and transistor aging, are becoming increasingly alarming. Especially, soft errors are becoming a serious problem in circuit design due to shrinking process dimensions. The smaller dimensions create a situation where the capacitance at each node in the circuit is lower, consequently requiring a smaller amount of charge to cause a glitch.

Design productivity. The gap between chip integration capacity and design productivity is growing wider. When taking into consideration the short time-to-market constraints that design teams have to deal with, the problem becomes even more acute. Meanwhile, the development of efficient techniques for extracting computational parallelism and binding it optimally in the available resources is difficult.

With the great promise of billions of devices on a chip to integrate more functionality and computing power, uncertainty of system performance and reliability also comes along. A more complete approach is to make the design of SoCs and multicore systems variation-aware and error-aware. We call this approach **VEBoC** – Variation and Error-aware design for Billions of devices on a Chip. This paper provides a high-level view of some of our design efforts to make VEBoC a reality.

The rest of the paper is organized as follows. In Section II, we present our view on variation-aware high-level synthesis. In Section III, we introduce a holistic error model and its potential usage on reliable circuit design. In Section IV, we present some of our insights on general-purpose multicore design. In Section V, we show our idea of application-specific multicore design. We then conclude this paper in Section VI.

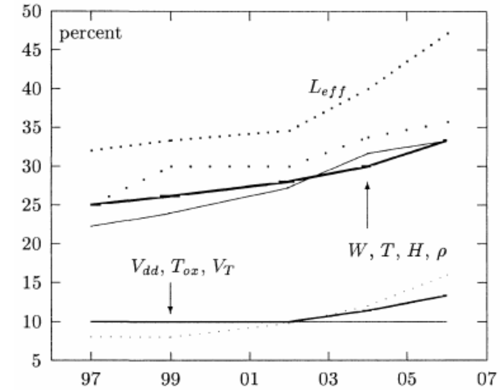

Figure 1: Variability trends in key process parameters with scaling process technology. The x-axis is year and the y-axis represents the variability in process parameters (courtesy of [1]).

II. Variation-Aware High-Level Synthesis

Behavioral synthesis (or high-level synthesis, HLS) transforms a behavioral description of a digital system into a register-transfer level hardware implementation consisting of a datapath and a control unit. The datapath consists of four types of components or resources, including functional units (FUs), storage units (such as registers), multiplexers, and interconnects. HLS usually consists of three subtasks: scheduling, resource allocation, and binding. The design specification is first compiled into an internal representation (such as a control-data flow graph), which is then mapped to the resources, selected from the resource library, to optimize design goals (such as delay, power, and area). Synthesis at the behavioral level is desirable because, compared to synthesis at RTL, code density can be reduced by ten times, and

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

simulation time can be reduced by one hundred times. Such a design productivity boost is much needed for next-generation chips with billions of devices.

The traditional approach has utilized the worst-case delay and power for each FU for design-space exploration during HLS. Unfortunately, the significant variations in device parameters in new process generations cause large variations in delay and power for FUs, multiplexers, and other resources, resulting in inadequate and less than optimal solutions. To overcome this, HLS must consider process variation. This is facilitated by the statistical characterization of power usage, delay, and area for each of the resources in the resource library. By accurately modeling and characterizing the random and systematic variations inherent in each resource, statistical information can be incorporated into the design.

Accurate test structures for characterization and modeling of the different sources of variation are critical in making statistical HLS successful. The resource models used must accurately reflect the actual fabricated behavior. This means that data concerning the types of variations (whether they be systematic or random), their scope (within-die, die-to-die, wafer-to-wafer, etc.), and how they correlate with one another, must be accurately gathered from real wafers and then processed. Complicating the resource characterization is the fact that device parameters and variations for a given process (and a given wafer fabrication facility) drift over time. To compensate for this, test structures designed for capturing variation data (such as those proposed in [3]) must be run frequently, and then resource libraries must be updated.

Once resources have been characterized for statistical delay and power variations, they can be used in new statistical HLS scheduling, allocation, and binding routines. As power, delay, and area are all interrelated, the variation-aware synthesis engine must address all of these factors simultaneously. The incorporation of multiplexer and interconnect variation is particularly important due to their dominant contribution to overall power and delay in modern technologies. Additionally, to accurately capture physical design information, the synthesis engine must connect to the actual layout of the design.

Probability distributions of power and delay for each of the resources will need to be considered during the synthesis steps. For example, for a scheduling and binding solution consisting of a multiplexer (mux) followed by an adder, the total delay is given by: $delay = d_{mux} + d_{wire} + d_{add}$, where d_{mux}, d_{wire}, and d_{add} are delay values for the mux, wires between the mux and the adder, and the adder. With statistical modeling, each of these delay values will become a delay distribution. In general, if there are k components in a path p, the delay distribution of the path with random variable x is $PDF_p(x) = Conv_{i=1}^{k} PDF_i(\mu_i, \sigma_i)$, where μ_i and σ_i are the mean and standard deviation of the delay distribution of component i, and *Conv* is the convolution operation. The timing yield will be evaluated probabilistically, which will be used in evaluating the effectiveness of the solution at meeting design goals. Similar statistical methods can be used for power calculations.

Another key to the effectiveness of the statistical HLS will be the consideration of physical design information, such as circuit floorplanning information. To optimize for power, area, and delay, the relative placement of FUs must be optimized, which largely determines layout area and interconnect usage. This is important because layout area and interconnect have a major effect on overall power consumption and delay. Exploration of the design space in the presence of these variables can be accomplished by an iterative solver using Pareto point pruning, while the cost of each solution under variation can be evaluated through a novel statistical cost function. A statistical resource library in behavioral synthesis enables early and accurate design space exploration. This allows for the mitigation of the anticipated negative effects of variation, and increased delay in interconnects, at design time.

Future work in variation aware synthesis will further integrate variation information to enable more robust and reliable circuits. This will be accomplished through the synthesis of circuits that are dynamically aware of variations in circuit parameters due to events such as device failure and temperature variations. High-level design and synthesis tools will need to be developed that can automatically anticipate these types of variability, and build in circuits to compensate. As stated in [4], this will mean moving "away from the concept of building margins (whether worst case or statistical) and focus on adaptive techniques for addressing variability and building robust circuits." These developments will be critical in the future to enable higher performance and lower power chips at ever higher levels of integration.

III. Error Modeling and Fault Tolerance

In addition to manufacturing defects and process variation, problems involving soft errors and transistor aging also need to be addressed in new DSM technologies. In order to counteract these issues, fault tolerance techniques need to be integrated into designs. There have been a number of fault tolerance techniques proposed in the literature. They span the hierarchy from gate level techniques to architectural level techniques, but they can all be classified into three categories: 1) time-redundant techniques, where results are recomputed after errors are detected (e.g., Razor [5]); 2) resource-redundant techniques, where multiple copies of same logic are used to compute results for comparison (e.g., TMR – triple modular redundancy [6]); and 3) information-redundant techniques, where check bits are added to form code words and checkers are used to check whether the outputs belong to the code space (e.g., [7]).

Selecting and applying the best fault tolerance technique or techniques to a design is a problem that will need to be addressed. Currently, the majority of error types are modeled, and fault tolerance techniques are applied, independently of each other. As devices continue to scale, it will no longer be possible to maintain this independence. By modeling each error type independently, a worst case condition is assumed. As has been shown with the current paradigm shift towards statistical static timing analysis (SSTA), worst case design in nanometer processes is not feasible. Therefore, accurate statistical error models will need to be created that take a holistic approach by incorporating many different types of errors into a single model. Probability transfer matrices (PTMs) are proposed by [8] to model the susceptibility of a circuit to defects and soft error. This method can be extended to arrive at a holistic model if timing errors due to process variation and aging can be considered also. The computational complexity of the method might limit its ability to efficiently model large circuits. This problem can be partially addressed by compression of the matrices similar to that shown in [9].

With a holistic error model of the circuit, fault tolerance techniques can then be applied. Currently, the majority of fault tolerance techniques are targeted towards a specific type of circuit error. The interactions between different techniques, as well as the effect a fault tolerance technique has on other types of errors will need to be considered due to the potential to both improve and decrease the performance of the circuit. The following situation is given as a motivating example.

An error model for a circuit has been constructed through the methods described above. It shows that the probability of the circuit working correctly is 0.75 when considering the effects of process variations, manufacturing defects, NBTI (negative bias temperature instability) transistor aging, and soft errors. This probability for correct operation does not meet the design specification and requires the addition of fault tolerance techniques in order to boost the reliability of the design.

Since the types of errors are varied there might not be a single fault tolerance technique that can efficiently solve the problem. Multiple techniques will be needed in order to sufficiently boost the reliability of the chip. By using the error model, the overhead due to the fault tolerance techniques can be minimized. For example, if the designer were to look at each fault independently, (s)he might try to boost the soft error resilience through selective node engineering, and (s)he might add TMR to protect against manufacturing defects, as well as other techniques for the remaining error types. The end result is an over-designed chip dissipating a large amount of power while delivering low performance.

However, if the designer were to use a holistic error model, the overhead due to fault tolerance would be minimized. For example, the error model would show that through adding some local TMR logic for manufacturing defects, the susceptibility of the design to soft errors and process variations has been lowered, reducing or eliminating the number of places where selective node engineering is required.

The above example demonstrates the need for a holistic approach when selecting proper fault tolerance techniques. It shows the importance of the holistic error model and the need to understand how a fault tolerance technique affects all types of errors. It also shows one of the many opportunities to lower the overhead caused by fault tolerance techniques. The example also exposes the need for improved fault tolerance techniques. As mentioned, the majority of fault tolerance techniques are meant for a specific type of error. They are tested against the assumption that the only type of error in the circuit is the one they are correcting. This assumption will not be valid as processes continue to shrink. New techniques will be needed that can operate in the face of different fault types and numerous faults of the same type.

One promising technique that can be used to meet these requirements is pair modular redundancy (PMR), as shown in Figure 2. The idea behind this design is to reduce the number of duplications compared to TMR, but augment the circuit with a coding scheme. Therefore, it is a combination of resource-redundancy and information-redundancy. The code is then used in the multiplexers to determine which output is correct. By interconnecting the modules at the output, it is possible to route around hard or soft errors in different stages of the structure.

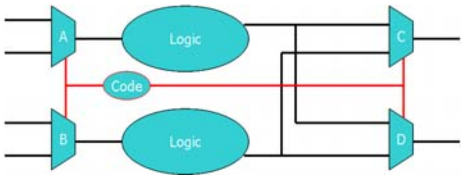

Figure 2: Pair modular redundancy (PMR)

Holistic error modeling in conjunction with fault tolerance techniques will become a requirement in future processes. Minimizing the power, timing, and area overhead due to fault tolerance techniques will be the key to continue increasing performance at current rates.

IV. General Purpose Multicore

While superscaler processors are efficient for applications that can exploit high Instruction Level Parallelism, chip multiprocessors are practical for applications that can spawn concurrent threads to be run on multiple cores. As a commercial example, the system-level diagram of the IBM Power4 is shown in Figure 3.

The multiple cores in current chip multiprocessors (CMPs) are connected via a shared bus. However, as many cores will be integrated in future, microarchitecture changes may be needed to scale the shared bus fabric without performance degradation. One alternative to the shared bus fabric is Network-on-Chip (NoC) communication infrastructure used in many recent designs. A generic NoC is shown in Figure 4.

The shared bus mechanism to connect multiple cores is well-understood and works well for current microarchitectures. The concept of sharing resources among cores can be taken from conventional multiple computer systems and applied to chip multiprocessors. Also, for moderate increase in the number of cores, hierarchical bus architectures and segmented or pipelined wires can be used instead of a simple shared bus fabric. However, for large number of cores, the shared bus fabric will become a bottleneck as more components are connected to the same set of wires. With NoC fabric, a generic network router and associated protocols can be used for chip multiprocessors of any number of cores, and scaling issues may be removed. However, system designers will need to learn the details of new on-chip communication design and rewrite software code to control synchronization among network components. A detailed study of related concepts can be found in [11].

Chip multiprocessors present new challenges to both the application software developers and the microarchitects. Some of the issues involved in the design of efficient chip multiprocessors addressed in previous research are discussed below.

The design of interconnection network, cache hierarchy, memory consistency model, bandwidth requirements, and power management are interrelated in the design of chip multiprocessors as shown in [12]. Fair caching is also an

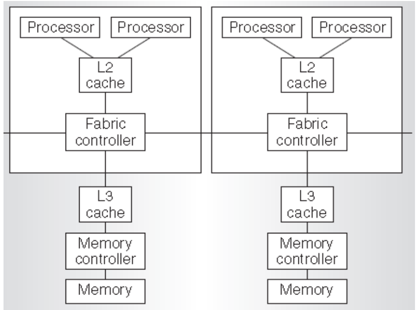

Figure. 3: High-level diagram of IBM Power4 [10]

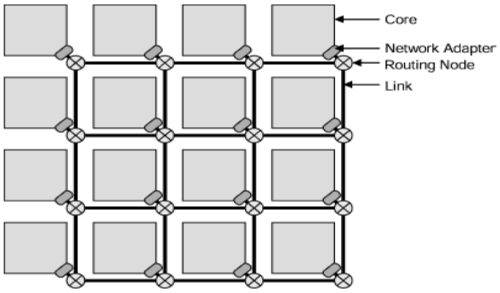

Figure 4: A general NoC structure illustrating the routing infrastructure to connect multiple cores [11]

important issue as number of cores is increased [13]. The number of cores that share a cache resource and the level of private cache hierarchy for a single core have a high impact on the overall throughput of a chip multiprocessor. The level in a cache hierarchy at which memory consistency must be maintained is also an important issue. In [14], the authors provide coherence protocols considering the impact on interconnect latency. In order to exploit maximum parallelism, thread management is an important task and can be done either at the operating system layer or in hardware. In [15], a hardware approach is provided that dynamically maps threads to cores on a run-time basis.

As more cores are integrated on a chip, the scaling of current microarchitecture techniques is a growing concern. For instance, the overhead for snooping based cache coherence protocols will increase linearly as more cores are added on a chip. Similarly, some threads may benefit from more execution units while other may work as efficient with lesser execution units. Dynamic system configuration based on workload characteristics can be a good performance booster. Techniques may include turning off snooping if it is not beneficial, or allocating communication-intensive threads to cores connected through short buses dynamically.

Heterogeneous chip multiprocessor is promising to reduce power consumption while maintaining high performance by allowing the processor to better match execution resources to each application's needs [16]. To build reliable multicore systems, different architecture-level techniques can be applied. For example, the ALU may be replaced by a PMR module. Other standard techniques include pipeline-level redundancy techniques and core-level redundancy techniques, where one core checks another core.

V. Application-Specific Multicores

We aim to address the DSM challenges by automating the design of efficient and high-performance systems that are customized for a particular application. Following the same design trend in general-purpose multicore architecture, we use application-specific multicores for high throughput with low power, where each core does not need to run in high frequency. The key optimization step is system-level synthesis (SLS). Our target is to automate application-specific multicore design for any size and type of HLL (high-level language) application.

Our system level synthesis flow (Figure 5) is based on three key constituent elements:

- An advanced software compiler for extracting both the fine-grain and coarse-grain parallelism available in the application.
- A custom instruction-less processor which is built to fit the tasks it is assigned to.
- A hardware compiler which combines the parallelism information extracted by the software compiler and the customization flexibility of the instruction-less processor to build an efficient multicore system for the application.

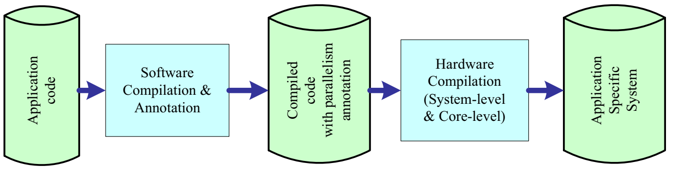
Application code
Software Compilation & Annotation
Compiled code with parallelism annotation
Hardware Compilation (System-level & Core-level)
Application Specific System

Figure 5: Multicore system level synthesis flow

A. *Software Compilation*

Our software compiler is based on the IMPACT compiler [17][18]. IMPACT has been traditionally focused towards instruction-level parallelism (ILP) extraction for EPIC (Explicitly Parallel Instruction Computers) architectures [19]. This type of parallelism is extracted through:

- traditional compiler optimizations, such as function inlining, loop distribution and unrolling, etc.
- statistical techniques based on application profiling and the formation of superblocks and hyperblocks.

We refer to this type of parallelism as fine-grain parallelism, and our system processing cores are designed specifically to take advantage of the fine-grain parallelism exposed by the IMPACT compiler. More information on the instruction-less processor cores is available in the following subsection.

Apart from the fine-grain parallelism, many media, scientific and other modern application have a great deal of coarse-grain parallelism which may be hidden in the sequential HLL description [20]. Extracting and mapping these coarse level tasks in different cores of the system can lead to remarkable improvements in performance and efficiency of the application execution. During recent and ongoing work, the IMPACT compiler is enhanced with a strong synergistic portfolio of different analysis techniques that can help expose the coarse-grain parallelism [21] inherent in many of the popular applications used in embedded systems today. In particular, a combination of different analysis techniques focusing on 1) pointers, 2) arrays and structures and 3) variable constraints and relationships, are effectively applied to discover different forms of coarse-grain parallelism. The different types of coarse-grain parallelism are distinguished in:

- intra-loop parallelism (iterations within loop are independent).
- cross-loop parallelism (iterations of a loop are only dependent on specific iterations of a second loop but not on the entire second loop).
- region parallelism (independent static code regions).

Our SLS flow expands the IMPACT coarse-grain parallelism extraction scheme by explicitly exposing the extracted parallelism on the compiled code. In particular, we are designing an efficient parallelism-annotation which will be applied on the IMPACT compiled application code exposing the different opportunities for parallel execution. This annotation will be read by the backend hardware compiler and will be used to build an efficient multicore system as described in a later subsection.

B. Processor Core

Our synthesized system is based on the utilization of multiple cores, each one of which is an appropriately parameterized version of the instruction-less custom processor. This processor is based on a VLIW-like array of Functional Units (FUs) which are controlled by a microcode (MC) memory (Figure 7) which stores microcode words. Each microcode word controls the data processing and data flow in the custom processor for one cycle. A program-counter register stores the address of the microcode word that will be executed in the next clock cycle, while an address generation circuit generates the next microcode memory address and stores it in the program counter.

In order to take advantage of the ILP extracted by the IMPACT compiler, our instruction-less processors are extended with extra architectural features. These features include:

- a predicate-register-file for storing the predicate values used in hyperblocks.
- a shifting-register-file at the tail of each functional unit for storing predicated results until the predicate value is determined.
- a memory-conflict-buffer for handling data-speculated loads.
- extra control circuitry for cancelling speculated operations and ensuring correct execution.

We refer to our custom instruction-less processor as Explicitly Parallel Operations System (EPOS) and preliminary results show that it can offer a great performance speed-up compared with the No-Instruction-Set-Computer (NISC) [22] for a variety of applications (Figure 6).

C. Hardware Compilation

The backend compilation of our SLS flow takes as input the compiled code and the coarse-grain parallelism annotation produced in the front end compilation, as well as the user constraints on performance, area and/or power. Its output is a multicore system based on EPOS cores, which satisfies the user constraints. The coarse-grain parallelism extracted and annotated in the front-end compilation is used to determine the number of EPOS cores and the type of inter-core communications. Depending on the user constraints, the type of annotated parallelism and the amount of data sharing, different types of inter-core communication schemes are considered, including:

- Global Asynchronous Local Synchronous (GALS)
- Data Memory shared
- Interrupt Driven
- Register shared

During the system level compilation phase, a preliminary estimation of each core schedule and latency is obtained by a first round of scheduling using infinite resources. The preliminary scheduling results from each core determine the minimum latency (critical path) of the task assigned to each core. These critical paths are used in the system level scheduling and interconnection scheme. When this phase of the hardware compilation is over, a complete system incorporating multiple EPOS cores and blocks of shared and/or distributed data memories with appropriate interconnection signals/buses is produced (Figure 7). Next, core-specific optimizations are performed such as register allocation and forwarding network minimization. Process variation modeling and error resilient design issues can be considered as well.

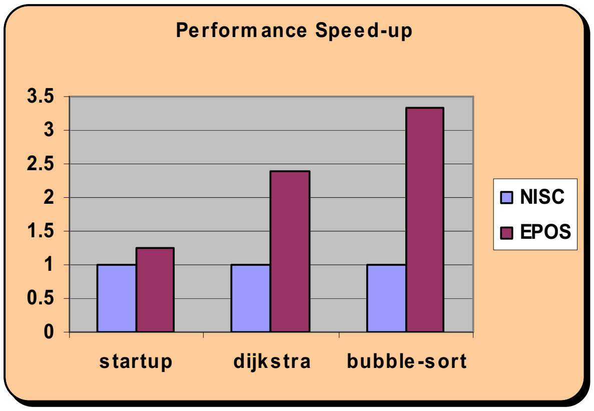

Figure 6: EPOS to NISC performance comparison results

VI. Summary and Conclusions

In this paper, we presented the emerging issues faced by designers when billions of devices are going to be available in the coming technology generations. We discussed the viability of variation-aware high-level synthesis, holistic error modeling and its usage for reliable circuit design, future multicore design issues, and the synthesis for application-specific multicores. Our goal is to achieve the potential for high chip integration, and high performance, available through technology scaling in a power-efficient and error-resilient way.

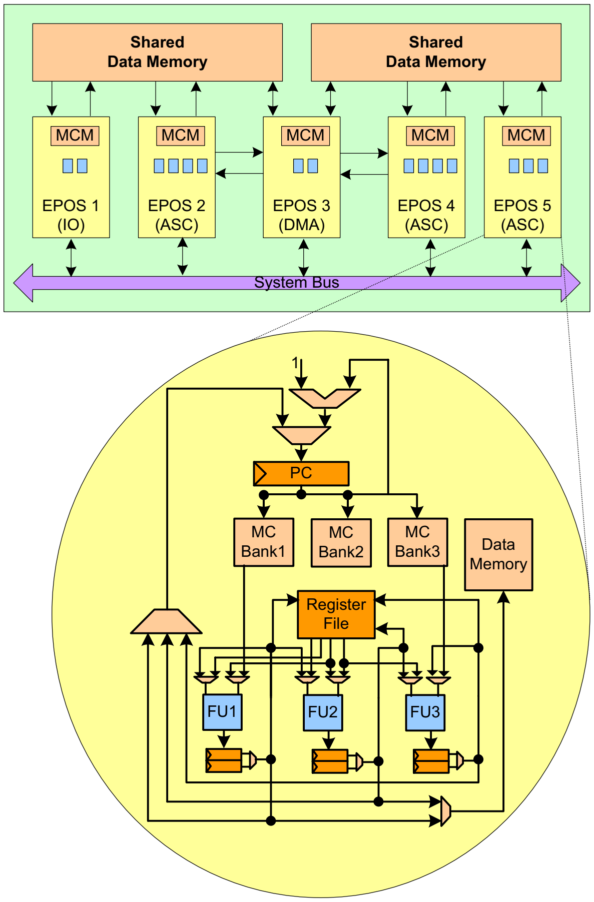

Figure 7: Multicore system and EPOS core detail

References

[1] A. Srivastava, D. Sylvester, D. Blaauw, *Statistical Analysis and Optimization for VLSI: Timing and Power*, Springer, 2005.

[2] V. Moroz, L. Smith, X.-W. Lin, D. Pramanik and G. Rollins, "Stress-Aware Design Methodology", *Intl. Symp. Quality Elec. Design*, 2006.

[3] K. Agarwal, S. Nassif, "Characterizing Process Variation in Nanometer CMOS", *Design Automation Conference*, 2007.

[4] D.J. Frank, "Design and CAD Challenges in 45nm CMOS and Beyond", *International Conference of Computer-Aided Design*, 2006.

[5] D. Ernst, et. al., "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," *MICRO*, Dec. 2001.

[6] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," Automata Studies, Ann. Of Math Studies, no. 34, C. E. Shannon and J. McCarthy, Eds, Princeton University Press, pp 43-98, 1956.

[7] M. Favalli and C. Metra, "Optimization of error detecting codes for the detection of crosstalk originated errors," *Conference on Design, Automation and Test in Europe*, 2001.

[8] S. Krishnaswamy, I. Markov, J. Hayes, "Tracking Uncertainty with Probabilistic Logic Circuit Testing," *IEEE Design & Test of Computers,* Jul-Aug 2007.

[9] S. Krishnaswamy, G. Viamontes, I. Markov, J. Hayes, "Accurate Reliability Evaluation and Enhancement via Probabilistic Transfer Matrices," *Design, Automation, and Test in Europe Conference,* 2005.

[10] R. Kalla, B. Sinharoy, and J. M. Tendler, "IBM Power5 Chip: A Dual-Core Multi-threaded Processor," *IEEE Micro*, 24(2):40-47, 2004.

[11] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, Vol.38, March 2006.

[12] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling," *32nd International Symposium on Computer Architecture*, June 2005.

[13] S. Kim, D. Chandra, and Y. Solihin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," *13th International Conference on Parallel Architecture and Compilation Techniques (PACT04)*, 2004.

[14] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. B. Carter, "Interconnect aware coherence protocols for chip multiprocessors," *33rd International Symposium on Computer Architecture*, 2006.

[15] S. Kim, D. Chandra, and Y. Solihin, "Hardware-modulated parallelism in chip multiprocessors," *Workshop on Design, Architecture and Simulation of Chip Multi-Processors Conference (dasCMP)*, 2005.

[16] R. Kumar, D. Tullsen, N. Jouppi, and P. Ranganathan, "Heterogeneous Chip Multiprocessors," *IEEE Computer*, November 2005.

[17] Wen-mei W. Hwu, et. al., "The Superblock: An Effective Technique for VLIW and Superscalar Compilation," *Journal of Supercomputing*, 1993.

[18] Scott A. Mahlke, David C. Lin, William Y. Chen, Richard E. Hank, Roger A. Bringmann, "Effective compiler support for predicated execution using the hyperblock," *MICRO,* 1992

[19] Michael S. Schlansker, B. Ramakrishna Rau, "EPIC: Explicititly Parallel Instruction Computing," *IEEE Computer* 33(2): 37-45, 2000.

[20] Matthew I. Frank, "System Support for Implicitly Parallel Programming," Center for Reliable and High-Performance Computing Technical Report CRHC-07-06, Oct. 2007.

[21] Shane Ryoo, et. al., "Automatic Discovery of Coarse-Grained Parallelism in Media Applications," *Trans. High-Perf. Embedded Arch. Compilers,* 2006.

[22] Mehrdad Reshadi, Bita Gorjiara, Daniel D. Gajski, "Utilizing Horizontal and Vertical Parallelism with a No-Instruction-Set Compiler for Custom Datapaths," *International Conference of Computer Design,* 2005.