A system-level approach for delay compensation of process variability impact on run-time configurable memory organizations

ABSTRACT

Process variability is increasing with each new technology node and degrading the performance of electronic systems. In this paper we present a combined design- and run-time technique which can guarantee the system-level parametric yield and optimize the global energy consumption of the system. The focus is on multimedia applications without a too strong dynamic behavior, like MPEG2, audio and video codecs. It is based on measuring the actual impact of variability on the configurable memories and adjusting them to the desired system period. It consistently outperforms conventional module-level worst-case design approaches, the energy gains range between 20% and 60% for different real time constraints, even for 65nm process variations of about 10% at transistor level which will become even higher for future scaled nodes.

1. INTRODUCTION

Technology scaling has historically improved the performance of embedded systems, both in energy consumption and speed. Scaling the minimum feature sizes below 100nm however, brings a host of problems which cannot be completely solved at the technology level. Back-end performance degradation, increased leakage currents and increased process variability are a few examples. Process variability is probably the most important one because it has a direct negative impact on yield and it has a large impact on all the characteristics of the system [2]. Due to its stochastic nature the only way to maximize the parametric system yield, the number of samples that meet the timing constraints, is either by incorporating corner-point analysis [1] in the designs or run-time techniques which can measure the actual variability and adapt the operation of the system, because it is impossible to predict its impact on the system before the chip is processed.

Memories are among the most variability sensitive components of a system. The reason is that most of the transistors in a memory are minimum-sized and are thus more prone to variability[3]. Additionally some parts of the memories are analog blocks, whose operation and timing can be severely degraded by variability, see Figure 1 [10]. Furthermore, in our target domain of multimedia applications memories occupy the majority of the chip area even in current designs and contribute the majority of the digital chip energy consumption. Thus they are very important blocks for the system.

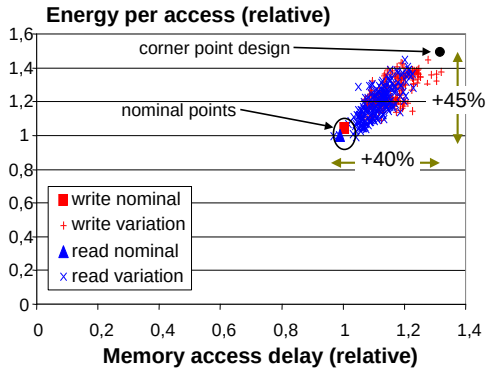

Figure 1: Impact of process variability on the energy/delay characteristics of a 1KByte memory at the 65nm technology node. The solid blocks on the bottom left indicate the simulated nominal performance assuming no process variability. The other points are the simulation results incorporating the impact of variability. The energy consumption and delay of a memory designed using corner-point analysis is also shown.

Maximizing parametric yield in memories via corner-point analysis and design will lead to severe overheads in energy consumption and delay, as indicated in Figure 1 by the size of the "clouds". The reason is that the memory design will use over-sized circuits and conservative timing margins to improve the predictability of the memory behavior by trading off performance. We want to alleviate the worst-case design margins by designing the memory in a way that allows the unpredictability of its behavior even though the parametric specifications are not met at the module level. In that case, the energy and delay of the memory will be much smaller than that of the corner-point with a very high probability. To maximize parametric yield at the system level we need to add run-time configuration capabilities to the memory. The application mapping can then be done

XXXXXX XXXXXX

for the nominal low-energy configuration point of the memories. A controller will configure the memories at run-time to their high-speed configuration if the application timing constraints are not met after fabrication.

Our approach (Section 2) includes a design-time and a run-time phase to accomplish compensation of the impact of process variability and energy consumption minimization. At design-time, the application mapping can distribute the real-time execution time constraint between the various parts of the application depending on the required bandwidth, neglecting the potential impact of process variability. At run-time, the architecture which includes a controller and run-time configurable memories will adjust itself to meet the application deadlines. This approach is scalable to future nodes, because it relies on circuit design techniques for the compensation and not on technology parameters which are becoming more limiting with each new technology node [5, 18].

In this paper, we illustrate this concept of system wide compensation of process variability effects on the lower level of the data memory hierarchy of a design for a Digital Audio Broadcast receiver (Sections 5,6).

2. MOTIVATING EXAMPLE

We illustrate in this section a simple example that shows how our technique can be applied and why it can provide robustness against process variability.

We assume that the memories available on the platform offer run-time configurability (Figure 2 [11], transistor level simulations at 65nm), a low-power option and a high-speed option which can be switched at run-time. Process variability severely impacts the energy/delay behavior of the memories. After fabrication, the actual energy/delay performance is a point in each cloud. One important property of these clouds is that they should not overlap in the delay axis. If this is satisfied then the actual high-speed memory performance is always faster than the actual low-power performance irrespective of the impact of variability on delay. This property is exploited by the run-time controller to guarantee parametric yield and it is also necessary to guarantee that the design-time analysis results are accurate no matter how large the process variability is.

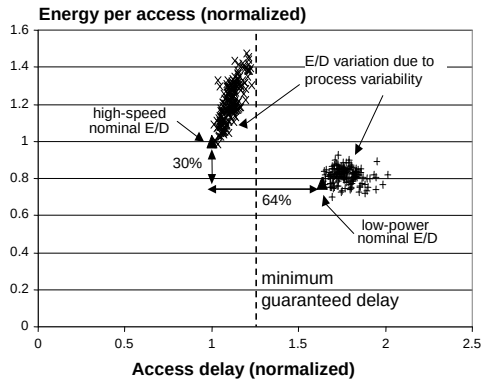

Figure 2: A configurable memory that offers two possible configurations, a slow low-energy one and a fast high-energy one is shown. Its configuration can change at run-time, but not on an individual cycle basis. The results come from a transistor-level simulation of a 1KByte memory with reconfigurable decoder and wordline drivers.

At design-time the application will be mapped in an optimal way given the bandwidth constraints of the memory organization. The only additional constraint from the platform side is that the resulting cycle time constraint cannot be lower than the minimum guaranteed delay (Figure **??**) of the critical memory of the system. Essentially we adapt the cycle time to the application in order to get a guaranteed parametric yield and an optimal energy consumption. At run-time the memory organization performance will be adapted to the optimal cycle time constraints coming from the design-time step.

Assume that a purely periodic application exists (e.g. current multimedia applications) where for the first N cycles of the period three data elements are fetched in parallel per cycle. In the last M cycles the bandwidth is only one element per cycle. If E_N,D_N and E_M,D_M are the energy and delay for the memory configurations used for the first and the second part of the code respectively. At design-time we will determine what are the optimal cycle times (D_N, D_M) such that $N * D_N + M * D_M < D_{period\ constraint}$ and the global energy consumption $3 * N * E_N + M * E_M$ is minimized, where $D_{period\ constraint}$ is the real-time application timing constraint. Given these time varying constraints on the cycle time over the application period, the run-time controller will adapt the speed of the memory organization by configuring the memories.

Figure 3 illustrates how the controller adapts the memory configurations to guarantee parametric yield. The energy/delay characteristics of two identical memories are shown. Their nominal behavior is the same, but after fabrication their actual performance will differ due to process variability. Given the cycle time constraint shown memory A has to be reconfigured to its high-speed configuration to meet the global timing constraints while the second one does not. Since the two clouds for the two configuration options of memory A do not overlap in the delay axis, it is guaranteed that this configuration will meet the timing constraints. Thus parametric yield is guaranteed, as long as the application cycle time constraints do not become lower than the minimum guaranteed delay of the slowest memory.

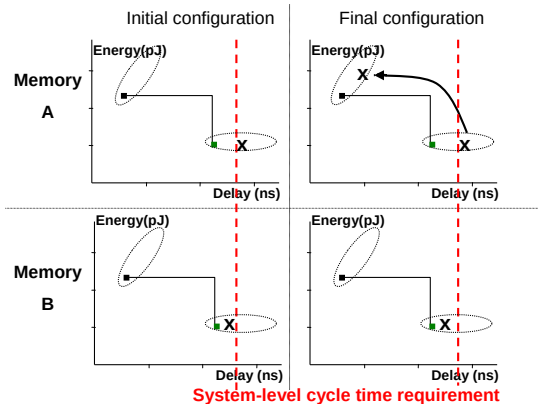

Figure 3: Two equal-size memories on the same chip are shown. Due to variability their access delays are different. The one that violates the cycle time requirement, should be switched to its fast configuration.

In this paper we focus on the run-time issues of this technique and the implementation of the required architecture, the design-time issues will not be discussed in depth.

3. RELATED WORK

The impact of process variability in the operation of systems is an issue of relatively recent interest. Borkar et al [4] concluded that a major shift from deterministic to probabilistic design is required and has advocated robust module design as a solution to the problem. Other approaches advocate solutions at the architectural level including run-time techniques.

Several papers, especially by circuit designers, have been implementing a local module level robustness to process variation [6, 7, 8]. Their goal is to make the modules robust enough at design time so that no matter how large the process variability, their functional yield will not be compromised. Their focus is on minimizing the impact of variability on leakage current and functional yield.

At the architectural level, several techniques have been proposed. The first one is Razor [12]. It can be currently applied on the processor pipeline, thus it is still a local technique, but it takes into account the actual performance of the circuit after fabrication. Its goal is to find the energy-optimal operation voltage of the processor, by reducing the voltage at run-time until the circuit fails to operate. This eliminates the need for design margins on the supply voltage. It is, however, focused on super-scalar microprocessors which is a different target domain than ours. Furthermore, the allowable operating V_{dd} range is reduced with each new technology node due to reliability issues, eliminating to a large extent the capability to exploit energy-delay trade-offs by changing V_{dd}.

Another architectural approach is the V_{th}-hopping scheme [9]. This is a technique aiming to reduce leakage power in processors by dynamically adapting the threshold voltage of the all the transistors of the processor using back-gate biasing techniques. The threshold voltage is configured at run-time depending on the workload of the processor, but variability is not taken into account and it is questionable whether back-gate biasing is still effective at very deep sub-micron nodes [5].

The fore mentioned module level approaches all focus on minimizing the impact of variability on the functional yield and leakage current at design-time. The architectural approaches focus on run-time solutions for the minimization of dynamic or static energy consumption of processors. The focus of our technique is to maximize parametric yield of the system and optimize dynamic energy consumption. We target the entire memory organization, because we can exploit the multiplicative effect that many memories and the application mapping step can offer. Furthermore, all the related work is either design-time robust circuit/module design or run-time compensation techniques. Our technique involves a tightly coupled design-time application mapping step and a run-time compensation step for globally optimal results.

4. ARCHITECTURE

In order to incorporate the previously described concept in the architecture we need to extend it with a limited number of additional blocks. The first required block is a controller which implements the functionality of the system-level compensation technique. To aid this controller a number of monitors are required, which will measure the actual energy/delay characteristics of the various memories on the chip, see Figure 4. Usually these are already present on modern chips anyway for test reasons.

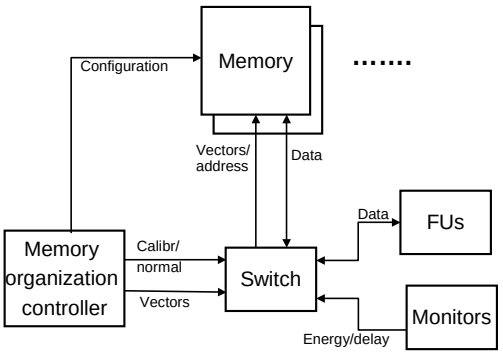

Figure 4: Simplified architecture of the complete system including the controller and the monitors.

A Register Transfer Level description of this architecture has been developed in VHDL. The functionality and implementation details for each of the modules of the architecture are discussed next.

4.1 Monitors and communication network

Monitoring circuits are required in order to measure the actual energy and delay characteristics of the memories. The delay monitor we used in our simulation is based on the work of Abas et al [13]. It consists of two separate chains of delay lines and it measures the time difference between the rising edges of two different signals. The first signal is sent through a slower delay line chain and the second through a faster delay line chain in order to catch up with the first one. Comparators are introduced along these chains. When the second signal catches up with the first one the respective comparator output becomes true. The time difference between the two edges is then calculated as the number of delay line times the delay difference between the delay lines of the fast and the slow chain.

This monitor can measure the delay difference between two one-bit signals. Five bits for the measurement result give enough granularity and range to the measurement for our purposes. Since we want to measure the output of memories with a maximum bitwidth of 32, the monitor of our architecture consists of 32 such one-bit monitors and 36 registers of 5 bits each. The 32 monitors measure asynchronously the delay of the different memory output bits and store them in a register. The maximum delay is then calculated and stored. For each memory we measure the read and the write delay of the high-speed and the low-energy configuration so we need four entries in the memory to store the results for a total of 36 registers of 5 bits each.

In the implementation of the delay monitor each delay line comprises 23 stages. The delay of the stages of the fast line is 50psec, for the stages of the slow line it is 100psec. Additionally an initial delay offset of 400psec exists before these delay lines. The range of delays that can be measured by this circuit is from 400psec to 1.55nsec and the granularity of the measurement is 50psec. Given the performance of current embedded memories the accuracy of this circuit is sufficient.

The energy monitor used is a plain current monitor. Kim et al [16] have proposed a circuit that monitors the current drawn by a block that supplies the measurement result in a digital form. Since energy consumption is proportional to the supply voltage multiplied by the current the block

draws, calculating it is straightforward. In order to reuse these same current monitor for all the memories in the organization we need to introduce sleep transistors in the voltage supply and the ground of each one. When one memory is measured, all the other memories have their sleep transistors in cut so they do not draw current, making the measurement as accurate as possible. Furthermore, a small register file of 4 entries is required to store the current measurements for the read and write operations for each of the two configurations of the memory under measurement.

The switch shown in Figure 4 is implementing a communication network based on segmented buses. The memory organizations of low-power systems are typically distributed so a centralized switch implementation would be a bad design choice. This communication network provides a bandwidth of three parallel transfers per cycle. The buses are bi-directional shared connections among a number of blocks and the long wires are split into segments using switches.

A small network controller with a look-up table controls the configurations of these switches. Based on the source and the destination blocks of the communication the controller finds the correct entry in the table and applies it to the switches. This information should come from the other system blocks. The two sources for this information are either the memory organization controller in the measurement phase or the memory management unit of the functional units during normal operation.

4.2 Memory organization controller

This block is responsible for the control of the entire memory organization. It synchronizes the operation of all the other blocks and changes the phase of operation. It is implemented as a Finite State Machine comprising 6 major states, see Figure 5. Its full functionality will be discussed next.

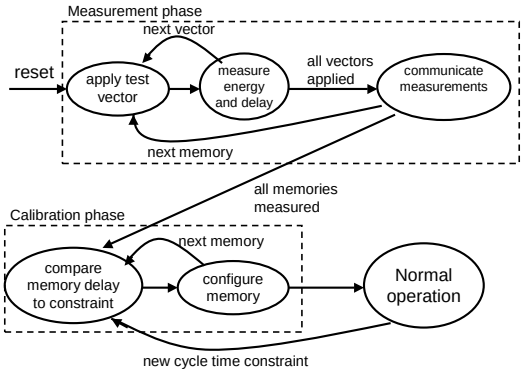

Figure 5: The operation of the controller can be divided into 6 major states.

The operation of the system is divided into three phases, namely the measurement, the calibration and the normal operation phase. As the names suggest, during the measurement phase the characteristics of all the memories will be measured. In the calibration phase the controller selects the appropriate configurations for the memories and applies them. Normal operation is the phase where the target application is ran on the platform.

4.3 Measurement phase

In the measurement phase the main goal is to measure the energy and delay characteristics of all the possible configurations of all the memories and update the controller with this information. Furthermore, for each of these configurations we need to extract the worst case access delay and energy consumption. A methodology has been proposed in [10] to generate the test vectors based on two vector transitions that can excite the memory addresses which exhibit the worst-case access delay and energy consumption. These test vectors are generated in a BIST-like manner.

In the beginning of this phase the communication network is configured so that only the controller and the monitors have access to the memories under measurement, decoupling the functional units from this procedure. Then the controller generates the test vectors and applies them to the memory under measurement. Each vector is applied twice, once in a write operation and once in a read operation. Access delays and energy of each operation is measured and locally stored in the monitor. This measuring operation is actually very similar to what a BIST scheme would do to determine whether the functionality of the memory is preserved, thus the measurement phase and testing phase could be combined. After all the test vectors have been applied to a memory, the monitor compares all the measurement results and finds the maximum read and write access delays and the maximum read and write energy per access and reports these values to the controller. The same procedure is repeated for each memory that should be measured. It is clear that this is a rather tedious and time consuming procedure. Each word in the memory organization has to be accessed 4 times to measure read and write performance of the high-speed and the low-power configuration and the total number of words is about 22K. This means that about 94K cycles are required to measure the memory organization of our architecture, which has nine memories. It is sufficient to perform it every time the system starts-up. The temporal effects due to process variability, related to reliability for instance, can be assumed to be very slow.

After the measurement phase is complete the memory organization controller has collected all the information about the accessed delay and the energy consumption of all the configurations of all the memories.

4.4 Calibration and normal operation phases

The next phase is the calibration phase. During this phase the controller configures the memories based on the actual timing requirements of the application, using dedicated control lines.

The controller itself includes a memory-mapped register file where information about the platform and the application are stored. Apart from the delay and the energy consumption of each memory, information about the current cycle time constraint for the given part of the application is given. This constraint is generated during the design-time analysis of the application. The distribution of the execution time throughout the application according to the bandwidth requirements yields a varying constraint, which is stored in one of these registers and updated every time it changes. The Pareto controller then has to adjust the memory configurations based on this system level cycle time target. This is a simple operation and it adds a negligible time overhead in the application execution time. Thus the configuration phase can be performed often. The RTL implementation of the controller in our system performs the comparison of the memory performance to the cycle time constraint and the

memory configuration in parallel for the different memories. This is feasible because no dependencies between the various memories exist. As a result, the calibration of the entire memory organization can be done in a single cycle. This enables a very frequent re-configuration of the memory organization, which provides freedom to the design-time analysis to do a more fine-grain cycle time adaptation enabling further energy gains.

During the normal operation phase the target application is executed on the functional units of the platform. The rest of the blocks can be virtually hidden by an appropriate configuration of the communication network. During this phase the memory management unit of the functional units takes care of supplying the source and destination block information to the network. The memory organization controller and the monitors can be kept completely transparent to the functional units.

5. EXPERIMENTAL SETUP

To demonstrate the architecture and the concepts that we propose we have developed the architecture of Figure 4 in RTL. The application that is running on the functional units is a Digital Audio Broadcast (DAB) receiver [15], see Figure 6.

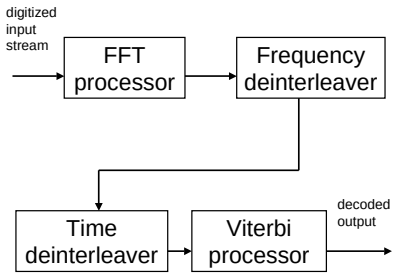

Figure 6: Block diagram of the Digital Audio Broadcast receiver

The transmission system in the DAB standard is based on an Orthogonal Frequency Division Multiplex (OFDM) transportation scheme using up to 1536 carriers (Mode I) for terrestrial broadcasting. At the DAB receiver side the OFDM carrier spectrum is reconstructed by doing a forward 2048-point FFT (Mode I) on the received OFDM symbol. This functionality is assigned to the FFT processor shown in Figure 6.

Forward error correction and interleaving in the transmission system greatly improve the reliability of the transmitted information by carefully adding redundant information that is used by the receiver to correct errors that occur in the transmission path. The frequency and time de-interleaver blocks unscramble the input symbols and the Viterbi processor is the one performing the error detection and correction based on the redundant information.

The energy optimal mapping of the DAB results in a distributed data memory organization that consists of nine memories, seven of them have a capacity of 1 KByte, one of 2 KByte and the last one of 8 KByte. Their bitwidths are either 16 or 32 bits. These memories have been simulated in HSPICE using the BPTM transistor model [17] for the 65nm technology node. A wrapper was developed in VHDL in order to include the impact of process variability on the energy and delay of the memories simulated at RT level.

Four distinct functionalities are performed in the decoding of the input signal as shown in Figure 6. In the actual implementation the frequency and time de-interleaving are performed in a single processor and we have dedicated hardware for the FFT and the Viterbi processing. The energy consumption of these units as well as the communication network will not be reported in this paper. The goal of this work is to keep the input/output behavior of the memory organization unchanged, so the processing elements are not affected by the architecture we propose. The impact on the communication network is negligible because the segmented buses architecture has the characteristic of scaling without a significant energy penalty, since the parts of the bus that are not needed are not activated.

6. RESULTS

Using the experimental setup described in the previous section we have performed a number of experiments on the architecture to evaluate how it adapts to the impact of process variability and varying timing constraints. Additionally we have evaluated the behavior of the same architecture when the corner-point design approach would be used and the behavior of a system fabricated in an ideal technology without variability.

The architecture we propose contains run-time configurable memories, so for fairness of comparison we have assumed that the corner-point and the nominal design approaches also have such memories available. The main difference is that these two approaches lack the controller to adapt the configurations at run-time, so one timing constraint has to be fixed at synthesis-time and no run-time adaptation is possible.

The results of the simulations are shown in Figure 7. They show the relative execution time and energy consumption for decoding a DAB audio frame using our approach compared to the "worst-case" design approach. The application mapping in all three case is the same, thus the number of cycles is constant. The dashed line shows the behavior of the system without variability, the nominal design. The dotted line represents the system design under worst case margins in the corner-point design fashion. Finally, the fine lines represent a number of statistical simulations of our architecture assuming different random drifts in the characteristics of the memories each time.

The nominal design, which is a non-implementable option, is obviously faster and more energy-efficient than the other two. The corner-point design is slower and less energy efficient, due to the overhead in energy and delay that is caused per memory by the worst-case design margins. Note that the points shown on the figure are all the available system-level energy/execution time points for these two design approaches assuming configurable memories are available. The actual implementation will, however, only correspond to one of these points, because these approaches cannot exploit the run-time configuration capabilities of the memories. In other words, each die can only implement one of the points. Furthermore, we have assumed that the corner-point memory design produces an accurately predictable memory behavior, thus a fully predictable system behavior, which is shown in the figure. In reality small statistical variations can still exist in the individual memory and the system behavior.

The characteristics of the proposed architecture, on the other hand, are less predictable. A number of experiments

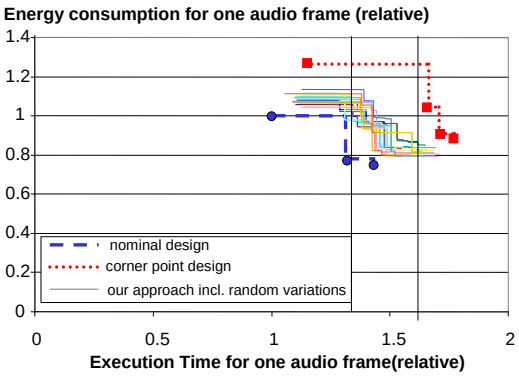

Figure 7: Global system-wide energy vs. delay trade-off points. The more tight the application timing constraint is the more energy needs to be consumed. The upper curve represents the result of corner-point design with configurable memories. The lower curve shows the result, assuming process variability does not exist. The middle curves are the results for various random injections of process variability on the system.

were performed assuming random injections of process variability each time to assess these characteristics. Each solid line in the figure represents a system with a given variability injection (a single die) working under a number of different timing constraints. The different lines represent different variability injections (different dies). The proposed system performs much better than the corner-point designed one and it approaches the ideal curve of the nominal system for some execution time constraints. For a given cycle time constraint it always consumes less energy or for a given energy budget it always has a faster execution time, compared to the curve of the corner-point design methodology; our approach leads to an energy gain ranging from 20% to 60%. If the execution time constraint is very tight, around 1-1.3 in Figure 7, the energy gains are around 20%. For more relaxed constraints (1.5-1.7) the energy consumption of our architecture can be as much as 60% lower than the worst-case. So even for a not that large transistor level process variability range of about 10%, large gains are observed. For future scales nodes even more variation hence gain is expected.

These gains come purely from the alleviation of the energy and delay overheads that the worst-case design margins introduce in corner-point design. The reason is the statistical nature of process variability. If the memories are designed using worst-case assumptions their behavior is more predictable, but a penalty in energy and delay has to be paid. In nominal memory design the memory behavior is less predictable, but no overheads exist. Given that the points in the cloud in Figure 1 follow an almost normal distribution, the nominally designed memories are faster and less energy consuming than the worst-case ones with an extremely high probability. This difference propagated to the system level is the one seen in Figure 7.

Furthermore, due to the existence of the controller this system can be configured at run-time to any of the points on a single solid line (Figure 7). So parametric yield is also 100% which is possibly an even more important property of our approach. This provides run-time level adaptation to varying timing constraints. Only a calibration phase is required to re-configure the memories to meet the new constraint, which inserts negligible overhead in energy and delay, as discussed in section 4.

7. CONCLUSIONS

A novel way to tackle the impact of process variability has been described. It is based on a run-time controller, which measures the actual performance of the various memories in the design and independently per memory adapts their energy/delay characteristics in order to meet the cycle time constraints. These have been extracted in a design time phase, where the application is analyzed. The resulting execution time and energy consumption of this approach is consistently better than that of the existing industrial solution, the energy gains range from 20 to about 60% depending on the timing constraints.

8. REFERENCES

[1] P. Gelsinger, "Giga-scale integration for Tera-ops performance: opportunities and new frontiers", Keynote speech at the 41st DAC, 2004.

[2] H. Chang et al., "The certainty of uncertainty: randomness in nanometer design", Proc. of PATMOS, pp. 36-47, 2004.

[3] J. Croon et al., "Physical modeling and prediction of the matching properties of MOSFETs", Proc. of ESSDERC, pp. 193-196, 2004.

[4] S. Borkar et al., "Design and reliability challenges in nanometer technologies", Proc. of DAC, pp.75, 2004.

[5] A. Keshavarzi et al., "Effectiveness of reverse body bias for leakage control in scaled dual Vt CMOS ICs", Proc. of ISLPED, pp. 207-210, 2001.

[6] M. Yamaoka et al., "A 300MHz 25um/Mb leakage on-chip SRAM module featuring process-variation immunity and low-leakage-active mode for mobile phone application processor", Proc. of ISSCC, Feb 2004.

[7] C. Kim et al., "A process variation compensating technique for sub-90nm dynamic circuits", VLSI Symp. Digest, pp. 205, 2003.

[8] A. Agarwal et al., "Process variation in nano-scale memories: failure analysis and process tolerant architecture", Proc. of CICC, pp. 353-356, 2004.

[9] K. Nose et al., "Vth-hopping scheme to reduce subthreshold leakage for low-power processors", JSSC, vol.37, no.3, pp. 413-419, March 2002.

[10] omitted for blind review

[11] omitted for blind review

[12] T. Austin et al., "Making typical silicon matter with Razor", IEEE Computer, pp.57, March 2004

[13] M.A. Abas et al., "Design of sub-10-picoseconds on-chip time measurement circuit", Proc. DATE, vol.2, pp. 804-809, Feb. 2004.

[14] omitted for blind review

[15] Radio broadcasting systems; digital audio broadcasting to mobile, portable and fixed receivers. *Standard RE/JTC-00DAB-4*, ETSI, ETS 300 401, May 1997.

[16] C. Kim et al., "An on-die CMOS leakage current sensor for measuring process variation in sub-90nm generations" VLSI Symposium Digest 2004, pp.250 - 251

[17] Y. Cao et al., "New paradigm of predictive MOSFET and interconnect modeling for early circuit design", Proc. of CICC, pp. 201-204, May 2000 .

[18] K. von Arnim et al., "Efficiency of body biasing in 90 nm CMOS for low power digital circuits", Proc. of ESSCIRC, pp. 175-178, Sept. 2004.