DVFS in Loop Accelerators Using BLADES

Ganesh Dasika[1], Shidhartha Das[2], Kevin Fan[1], Scott Mahlke[1], and David Bull[2]

[1]Advanced Computer Architecture Laboratory
University of Michigan - Ann Arbor, MI
{gdasika, fank, mahlke}@umich.edu

[2]ARM, Ltd.
Cambridge, United Kingdom
{sdas, dbull}@arm.com

ABSTRACT

Hardware accelerators are common in embedded systems that have high performance requirements but must still operate within stringent energy constraints. To facilitate short time-to-market and reduced non-recurring engineering costs, automatic systems that can rapidly generate hardware bearing both power and performance in mind are extremely attractive. This paper proposes the BLADES (Better-than-worst-case Loop Accelerator Design) system for automatically designing self-tuning hardware accelerators that dynamically select their best operating frequency and voltage based on environmental conditions, silicon variation, and input data characteristics. Errors in operation are detected by Razor flip-flops, and recovery is initiated. The architecture efficiently supports detection, rollback, and recovery to provide a highly adaptable and configurable loop accelerator. The overhead of deploying Razor flip-flops is significantly reduced by automatically chaining primitive computation operations together. Results on a range of loop accelerators show average energy savings of 32% gained by voltage scaling below the nominal supply voltage.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Microprocessors and microcomputers*

General Terms

Design, Performance

Keywords

Low power, Voltage scaling, Frequency scaling, Embedded systems, High-level synthesis

1. INTRODUCTION

As silicon technologies enter deep sub-micron realms, circuit level techniques are increasingly employed in conjunction with architectural techniques to achieve the stringent performance and power goals of embedded applications. Dynamic voltage and frequency scaling (DVFS) are widely used to reduce the overall energy consumption of a computer system, particularly for workloads with high variation in processing requirements. DVFS can either be used to push the operating conditions of a circuit beyond the nominal operating conditions assumed during design time to achieve improved clock frequency, or to reduce energy consumption at times when the full capabilities of the hardware are not required.

A critical issue for DVFS-enabled computer systems is determining the safe operating voltage at which maximum execution efficiency is achieved, while still guaranteeing correct operation of all components. Traditional techniques for DVFS utilize a delay chain or a lookup table to determine the minimum voltage necessary to guarantee error-free operation at a particular frequency [2]. Design-time characterization of the critical paths determines the margins that need to be added in order to ensure that the synthetic delay path is guaranteed to fail before the actual paths in the presence of worst-case operating conditions, process variation, temperature hot spots, and supply voltage uncertainties.

Razor [3, 1] is a cost-effective, circuit-level timing speculation technique that allows detection and correction of speed-path failures. In the traditional worst-case design technique, safety margins are added during design time to ensure computation correctness even under the worst-case combination of input vectors and operating conditions (process, voltage and temperature conditions). Razor is able to exploit these margins when operating under more typical conditions by aggressively scaling voltage and frequency and relying on a combination of in-situ architectural and circuit-level techniques to suitably flag any resultant timing errors and recover from them.

In large designs, the Razor shadow latch cannot be used if complex control signals are on the critical path. For example, if critical clock-enable signals going into clock-gating cells are too slow, the affected flip-flops will not be able to restore to their correct state as they will have been clocked incorrectly and their shadow latches will not have the correct data either. In situations like this, the most suitable way to recover from Razor errors is to add extra storage elements for checkpointing the microarchitectural state from which the processor can be restored to a point before the error. Leaving data in flight in this manner complicates a processor's control and forwarding logic which is often on the critical path. Checkpointing for Razor error recovery requires techniques specific to the microarchitecture under consideration. This technique is invasive and, as such, requires careful analysis of individual microarchitectures before Razor can be employed.

Another difficulty of deploying this technology is that the Razor flip-flop's hold-time is much larger than that of a conventional flip-flop. Any fluctuations shortly after the rising clock edge are treated as timing errors. As a result, short paths to a Razor flip-flop must be lengthened by adding buffers to ensure the hold-time constraint is met on all paths. This can be costly in terms of area and energy consumption, thereby impacting the gains achieved through DVFS.

In essence, Razor would be more easily deployed in architectures that have predictable control logic to quickly compute a restore point; storage structures for checkpointing where data is guaranteed to not be overwritten for several cycles; and a regular microarchitecture with predictable path lengths to enable the easy application of Razor. Further, it is important to have an automated system to insert Razor flip-flops and the supporting hardware in order to reduce design time.

The focus of this paper is an automated system to synthesize Razor-enabled loop accelerators (LAs) from high-level specifications. LAs are ideal for applying Razor technology due to their regularity, queue-based storage structures, and simple control. BLADES (Better-than-worst-case Loop Accelerator Design) is an energy-efficient, application-specific solution that adapts its operation to the environmental conditions, including silicon variation, temperature, and data precision. The inputs to BLADES are the target application expressed in C and the desired performance. Compiler analyses and scheduling are used to synthesize a minimum cost LA for the application to meet the given performance target [5].

We extend a baseline LA system with an application specific error recovery mechanism that is automatically derived. The LA is augmented with additional registers to enable efficient rollback and re-execution when a timing violation is detected. Further, we augment the system to automatically chain primitive computation operations together to reduce the overhead of ensuring that flip-flop hold-time constraints are met.

Across a large set of media and signal processing loops, LAs with Razor technology achieved an average energy savings of 32% with voltage scaling.

DAC 2008, June 8-13, 2008, Anaheim, California, USA.

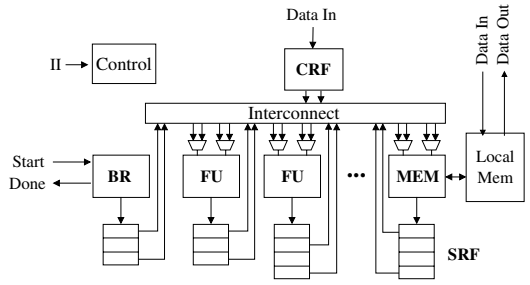

Figure 1: Hardware schema of a loop accelerator.

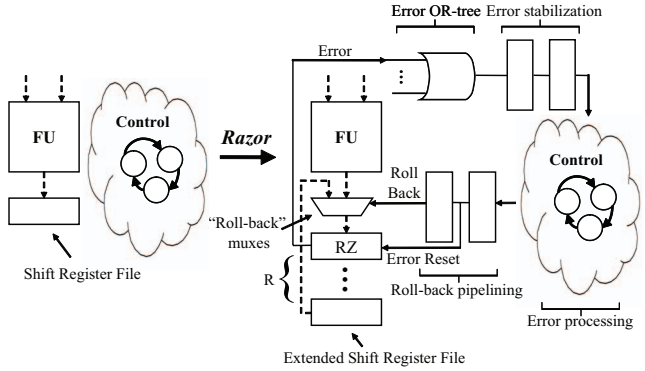

Figure 2: Structural modifications to a loop accelerator to support Razor. Data signals are dotted lines and control signals are solid lines.

2. THE BLADES ARCHITECTURE

2.1 Loop Accelerator Architecture

The LA used in this work is a hardware realization of a modulo-scheduled loop. Modulo-scheduling is a software pipelining technique that achieves high levels of parallelism by overlapping successive iterations of a loop [8]. The template for the baseline LA architecture is shown in Figure 1 [5]. The LA is designed to exploit the high degree of parallelism available in modulo-scheduled loops with a large number of functional units (FUs). Each FU writes to a dedicated shift register file (SRF); in each cycle, the contents of the registers shift downwards to the next register. The entries in a SRF therefore contain the values produced by the corresponding FU in the order they were computed. In addition, a central register file (CRF) holds static live-in (live-out) register values received from (sent to) the host processor which cannot be stored in the SRFs.

The synthesis system for this architecture takes an application loop in C and generates behavioral Verilog for a hardware accelerator that implements the loop within a given performance constraint. The performance requirement is specified as an Initiation Interval (II), the number of cycles between the start of execution of subsequent iterations of the loop.

2.2 Applying Razor

Razor flip-flops are employed in LAs to facilitate aggressive DVFS beyond the point of erroneous operation. To ensure proper operation, errors must be dynamically detected and the erroneous operations must be re-executed. Razor is supported in the LA architecture with three important changes as illustrated in Figure 2: extending the shift register files, adding roll-back multiplexers and incorporating an error-cognizant controller.

First, all SRFs are extended to keep data values live for more cycles. Each SRF need only be extended as per the roll-back penalty, *R*, of the design – the number of clock cycles spent detecting timing errors and restoring all the registers to a previously-known state.

The second extension is a set of "roll-back multiplexers" that are added in order to support restoration of prior state. When an error occurs, an SRF entry is restored to its value stored *R* cycles earlier using the additional entries in the SRF. In this manner, all FUs are re-executed with their inputs from *R* prior cycles. Similarly, the memory units in the BLADES architecture write their outputs to store queues first and these values are only committed to main memory after *R* cycles, when the addresses and values are known to be error-free. In the event of a Razor error, the pipeline state in the FUs is disregarded and execution resumes from an older state.

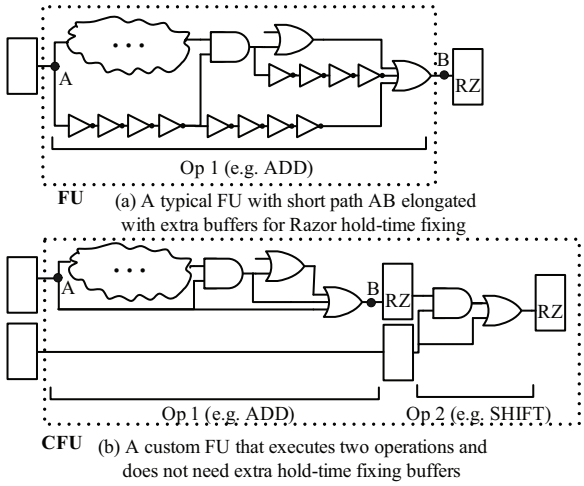

Figure 3: The effect of using CFUs

The final extension for a Razor LA is in the controller and its supporting logic. As stated earlier, the LA is a hardware implementation of a modulo-scheduled loop and is, as such, statically scheduled and control signal values for all cycles of executions are all statically determined. This makes reverting back to a prior state simple – if an error was detected while the LA is executing, the controller reverts to a state *R* cycles earlier.

The different aspects of the design that determine the value of *R* are illustrated in Figure 2:

1. FU pipeline: Values in pipeline registers are not saved. Therefore, on an error, all data in a FU's pipeline registers have to be discarded, and the corresponding operations re-executed. In this work, the maximum pipeline depth for any FU is 2 cycles, but this number may vary between designs.
2. Error OR-tree: The error signals from all the Razor flip-flops across the design are OR'd together to create one unified error signal. If the design has a very large number of Razor flip-flops, the fan-in to this OR gate can be quite large, requiring the gate to be pipelined. This is never the case in this work.
3. Error stabilization: In some corner cases, it is possible for the OR'd error signal to be metastable. To overcome this, the signal is passed through 2 flip-flops.
4. Error processing and roll-back generation: The control sees the error signal, prepares to revert to a previous state and sends a "roll-back" signal to the SRFs.
5. Roll-back pipeline: The "roll-back" signal is pipelined in order to prevent the recovery process from being timing critical. A cycle after it is generated by the controller, all the error flags in the Razor flip-flops are reset.
6. Roll-back: A cycle after the error flags are reset, the old data is restored. If the restoration process fails, error flags are set again and the process restarts.

In the designs presented in this work, *R* is 6 cycles (2 cycle-deep FU pipeline + 2 cycle error stabilization + 2 cycle error reset and roll-back).

2.3 Overheads of Using Razor

One of the main challenges to overcome when deploying Razor is the increased hold-time constraint. During the speculation window (the period of time after the rising clock edge where a change in the input signal is treated as a timing error), if the outputs of short paths in the design change correctly due to changing inputs, their change in value might be misinterpreted as a timing error. To avoid this, the shortest path in the design should be longer than the speculation window. To ensure this, extra buffers are inserted by the synthesis and place and route tools along the short paths.

For example, in Figure 3(a), the different paths connecting point A to point B would have several buffers inserted on them since there are very few gates on these paths. Based on the size of the design and the number of short paths, the energy overhead can be quite large; over 20% if the speculation window is 40% of the clock period.

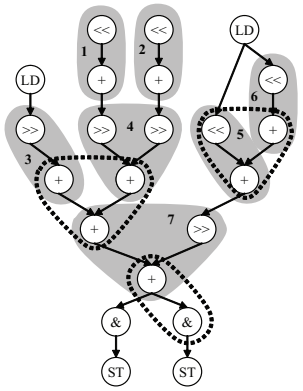

Figure 4: Dataflow graph for a portion of the `fsed` loop kernel. Operations chained to form COPs are in the numbered, shaded regions. Some operations that were not selected to be combined are shown within dotted lines.

There are a few different ways to circumvent the Razor hold-fixing problem. One way is to not issue back-to-back operations on the same FU. This would prevent the inputs to the FU from changing every cycle provided each FU's inputs are gated. However, this would halve the performance and would not be an appropriate solution in most situations. The performance need not be hurt if the number of FUs was doubled, each executing a new operation every 2 cycles. This would preserve the throughput of the LA. This method would not double the area since only the number of FUs would have to double and not the controller or the number of SRFs and CRFs, but this technique would certainly have a noticeable area cost for reducing the dynamic energy consumption. Another technique is a hybrid solution – identifying opportunities to create multiple-operation FUs that require no hold fixing latches; and, when that is not possible, resorting to using buffers to fix the increased hold time.

2.4 Shaving the Overheads of Razor

To address the hold-fixing problem, we propose to combine multiple FUs into 2-cycle "custom functional units" (CFUs). A CFU uses 2 cycles to execute 2 or more operations back-to-back. Its inputs are only changed every 2 cycles and the values computed after 1 cycle are stored in a register. This way, the values that fan-in to Razor flip-flops are guaranteed not to change during the speculation window unless there is a timing error, thereby eliminating the need to insert extra buffers, as shown in Figure 3(b). A 2-operation CFU is illustrated in Figure 3(b), but a third operation could also be executed, in parallel to op 1 and feeding op 2.

These CFUs are identified by the compiler after analyzing the dataflow between different instructions. The input to the compiler are custom operation (COP) graphs of different dataflow patterns and opcodes; each node in the graph represents an input, an output or an opcode and each edge represents dataflow. The compiler finds subgraphs in the loop kernel that are isomorphic to the input CFU graphs and converts these subgraphs to COPs.

When the hardware for these different CFUs is generated, CFUs that implement different opcodes, but have the same COP graph, can be combined into a single FU. For example, the operations

```
r0 = (r1 * r2) - (r3 + r4)
r5 = (r6 + r7) - (r8 + r9)
```

can be executed on the same FU. Adding an extra adder and some simple control logic can extend an MPY/ADD/SUB CFU to also execute ADD/ADD/SUB operations. Overlapping similar COPs in this manner reduces the overall number of FUs and therefore reduces the number of SRFs in the design at the cost of a few more muxes. For the purpose of this paper, the opcodes supported by each COP are limited to binary arithmetic operations since these make up the vast majority of opcodes used in loop kernels.

Figure 4 shows an example of the COPs identified in a portion of the `fsed` benchmark's inner-loop. The operations in shaded areas 1, 2, 3, 5 and 6 are combined to form 3-operand, 2-operation CFUs and the ones in shaded areas 4 and 7 are combined to form 4-operand, 3-operation CFUs. Depending on the schedule, COPs 4 and 7 could potentially be assigned to the same CFU due to their dataflow and functional similarities. Similarly, COPs 1, 2, 3, 5, and 6 could also be combined.

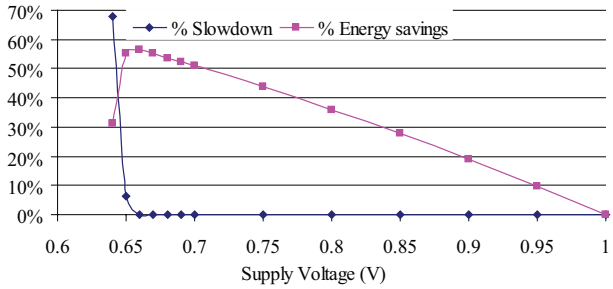

Figure 5: Energy savings and slowdown with voltage-scaling for the `sobel` benchmark.

A loop kernel may have several different COPs that are not chosen to be implemented as a CFU. Examples of some of these are shown inside dotted lines in Figure 4. The selection algorithm used in this paper is a greedy algorithm that selects the largest possible subgraphs searching upwards from the bottom of the loop. The final goal is to try to encompass as many operations into CFUs as possible while minimizing the number of unique individual opcodes, giving priority to fewer, larger CFUs over many, smaller CFUs.

COP 5, for example, could have been expanded to include the addition operation in COP 6. However, this would leave the left-shift operation in COP 6 alone and a separate left-shift FU would be required to implement it. The ADD/AND COP circled at the bottom of the figure could be selected but priority would be given to the larger COP 7. Further, since an AND FU would still be required to implement the one remaining AND operation, creating an ADD/AND CFU would not necessarily have any benefit.

An optimal branch-and-bound-based algorithm was also implemented to select subgraphs, but was not used because it resulted in no noticeable difference in quality-of-result (QoR) but required significantly more time to execute.

3. EXPERIMENTS

3.1 Setup

The designs in this paper were synthesized at their maximum possible frequencies (i.e at the point where increasing the target frequency did not lead to improved QoR) with Design Compiler 2007-03, and Galaxy ICC 2007-03 using a 65nm process with a nominal supply voltage of 1 Volt. Simulations to observe error rates were performed in Nanosim 2005-09. Energy numbers were obtained using Nanosim and PrimeTime-PX 2006-12. The synthesis target assumed slow conditions (0.9V, slow silicon, 125°C) and simulations assumed typical conditions (1.0V, typical silicon, 25°C).

LA hardware was synthesized for ten compute-intensive loop kernels from various application domains. In the designs presented in this paper, every FU was followed by a Razor flip-flop - i.e. all the bits in all the top entries in SRFs were Razor flip-flops and all the others were normal D-flip-flops. This would likely be too conservative at mature process nodes, but in the worst-case scenario where process variation is a significant problem, it is reasonable. In a mature process, only the the most critical paths need to terminate in Razor flip-flops.

3.2 Proof of Concept

A simple experiment illustrates that an LA's behavior matches that of a general purpose processor when applying DVFS with Razor. An LA for `sobel` – an edge detection kernel often seen in image processing applications – was run at various voltage levels and at the nominal clock frequency. Whenever an error occurred, the LA was rolled-back 6 cycles, the clock frequency was halved, and the offending operation was re-executed. The frequency was restored after the offending instruction completed. Figure 5 shows the energy savings obtained using Razor and the slowdown from re-executing operations that cause an error. Energy savings continue to increase until approximately 70% of the nominal operating voltage. The similarity of this graph to data presented in prior Razor work [3, 1] that used general purpose processors verifies that it is just as effective when applied to LAs.

3.3 Dynamic Frequency Scaling

To simulate behavior under reduced voltages, the LA was run with dynamically changing frequencies, shown in Figure 6. When several

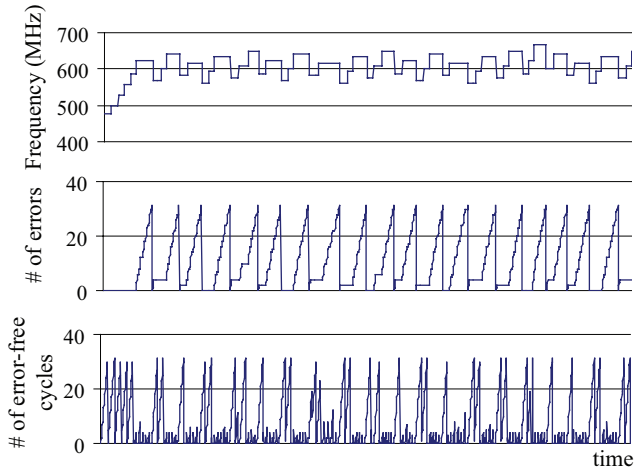

Figure 6: Dynamic execution of a `sobel` BLADES processor.

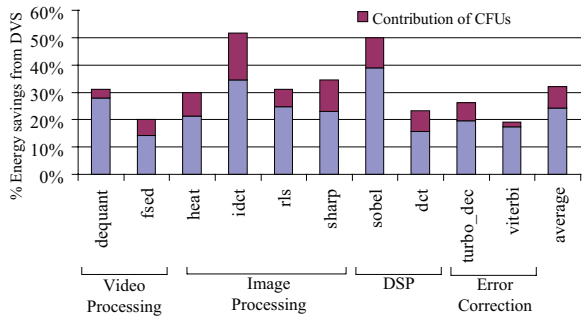

Figure 7: Dynamic energy savings when using DVS. The top portion is the contribution to this number by using CFUs.

cycles of error-free operation are observed, the clock period is gradually decreased. When a timing error does occur, the supplied clock frequency remains unchanged during the error recovery process but every 2 clock cycles are gated, effectively halving the frequency in the LA. The frequency is only reduced after several consecutive errors are observed.

3.4 Effect of CFUs

Figure 7 shows the savings in energy from using DVS across four application domains. The contributions of the CFU technique presented in this paper make up the top portion of each bar. On average, there is a 32% savings in energy, of which 8% was due to the CFU technique.

The benchmarks `idct`, `sharp` and `dct` had not only multiple CFUs, but also a reduction in the total number of FUs in the design since some FU types were entirely subsumed into CFUs. In `idct`, for example, none of the multiply units required extra hold-fixing since they were all within CFUs. The same was true in `sharp`, but for subtract units. The CFUs contributed to approximately 33% of the energy savings observed in these benchmarks.

However, the benchmarks `viterbi` and `dequant` did not do as well; the CFUs contributed to only 9% and 10% to the overall energy savings, respectively. The primary reason for this is that both of these benchmarks have a significant amount of control flow within the inner-loop. These results can be improved by adding better predicate support to the CFUs. Currently, all the operations within a CFU are guarded by the same predicate; allowing different operations within a CFU to execute independently would add some complexity to the replacement algorithm and the hardware generated but would result in more energy savings.

4. RELATED WORK

Frequency scaling applied to ASICs has been researched by different groups. Dhar et al. [4] proposed an adaptive voltage scaling scheme that utilized several blocks of logic around the main ASIC to dynamically change the voltage as is needed for the desired system speed. Their work, however, is purely external to the processor's design and is not interacting with it in any way unlike the scheme proposed in this paper. In [11], the authors propose several modifications built in to the main logic of the design for the purposes of scaling the voltage. Their technique, however, utilizes no knowledge of the application itself - what the constraints are on data, when specific values will actually be used, etc., and as such have a limited field of view.

Razor-style flip-flops have been used in the past for different reasons, the most significant of these [3] being for the purpose of reducing the supply voltage of a processor and in turn reducing the overall energy consumption. Razor flip-flops, or more generically, flip-flops which are able to detect changes in the input data near the clock edge, have also been used for reliability reasons in [7].

Opcode chaining was previously used in [10, 6, 9]. However, prior work focuses on chaining multiple simple operations into a single cycle to improve performance by performing a single complex computation instead of multiple simple ones.

5. CONCLUSION

In this paper, we propose the automatic design of adaptive loop accelerators. The designs leverage Razor technology that consists of a delay-error tolerant flip-flop placed on critical paths to identify timing errors in circuits. Razor enables a design to dynamically adjust its supply voltage and operating frequency to an optimal point based on the environmental conditions and characteristics of the computation. Adaptive loop accelerators consist of strategically deployed Razor flip-flops at the vulnerable locations of the loop accelerator as well as additional flip-flops to enable rollback and re-execution in the event of a timing error. Our loop accelerators utilize a stylized architecture template that allows highly customized designs to be automatically adapted to detect and recover from errors. These accelerators can be used to reduce energy consumption, increase performance, or provide adaptability in the presence of process variation. For our experiments, we show the clock frequency and voltage can easily be scaled in accelerators by adding a small number of Razor flip-flops. Specifically, voltage can be reduced to below 70% of the nominal supply voltage resulting in over 50% energy savings with negligible performance penalty, 32% on average. Further, by fusing primitive operations into custom operations, the hold-fixing requirement of Razor is relaxed and the associated energy overhead is reduced by as much as 33%, 24% on average.

6. ACKNOWLEDGEMENTS

We thank the anonymous referees who provided excellent feedback. We would also like to thank Hyunchul Park and Manjunath Kudlur for their help in designing the Loop Accelerator synthesis system and Krisztian Flautner for his support of this project. This research was supported by ARM Ltd. and by the National Science Foundation grant CCF-0347411.

7. REFERENCES

[1] D. Blaauw et al. Razor 2: In-situ error detection and correction for pvt and ser tolerance. In *2008 IEEE International Solid-State Circuits Conference*, Feb. 2008.

[2] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. *Journal of Solid State Circuits*, 35(11):1571–1580, Nov. 2000.

[3] S. Das et al. A self-tuning DVS processor using delay-error detection and correction. *Journal of Solid State Circuits*, 41(4):792–804, 2006.

[4] S. Dhar, D. Maksimovic, and B. Kranzen. Closed-loop adaptive voltage scaling controller for standard-cell ASICs. In *Proc. of the 2002 International Symposium on Low Power Electronics and Design*, pages 103–107, Aug. 2003.

[5] K. Fan, M. Kudlur, H. Park, and S. Mahlke. Cost sensitive modulo scheduling in a loop accelerator synthesis system. In *Proc. of the 38th Annual International Symposium on Microarchitecture*, pages 219–230, Nov. 2005.

[6] S. Hu and J. E. Smith. Using dynamic binary translation to fuse dependent instructions. In *Proc. of the 2004 International Symposium on Code Generation and Optimization*, pages 213–226, 2004.

[7] M. Nicolaidis. Time redundancy based soft-error tolerance to rescue nanometer technologies. In *Proc. of the 1999 IEEE VLSI Test Symposium*, pages 86–94, 1999.

[8] B. R. Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Proc. of the 27th Annual International Symposium on Microarchitecture*, pages 63–74, Nov. 1994.

[9] P. Sassone, D. S. Wills, and G. Loh. Static strands: safely collapsing dependence chains for increasing embedded power efficiency. In *Proc. of the 2005 ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 127–136, June 2005.

[10] M. Sivaraman and S. Aditya. Cycle-time aware architecture synthesis of custom hardware accelerators. In *Proc. of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 35–42, 2002.

[11] K. Usami et al. Design methodology of ultra low-power MPEG4 codec core exploiting voltage scaling techniques. In *Proc. of the 35th Design Automation Conference*, pages 483–488, 1998.