Bespoke Processors for Applications with Ultra-Low Area and Power Constraints

Hari Cherupalli
University of Minnesota

Henry Duwe
Iowa State University

Weidong Ye and Rakesh Kumar
University of Illinois at Urbana-Champaign

John Sartori
University of Minnesota

This article makes the case for bespoke processor design, an automated approach that tailors a general-purpose processor IP to a target application by removing all gates from the design that can never be used by the application. The resulting processors are more area- and energy-efficient without sacrificing performance.

A large class of emerging applications is characterized by severe area and power constraints. For example, wearables[1] and implantables[2] are extremely area- and power-constrained. Several IoT applications, such as stick-on electronic labels,[3] RFIDs,[4] and sensors,[5] are also extremely area- and power-constrained. Area constraints are also expected to be severe for printed plastic[6] and organic[7] applications.

Cost concerns drive many of the above applications to use general-purpose microprocessors and microcontrollers instead of much more area- and power-efficient ASICs, because, among other benefits, development cost of microprocessor IP cores can be amortized by the IP core licensor over many chip makers and licensees. In fact, ultra-low-area- and power-constrained microprocessors and microcontrollers powering these applications are already the most widely used type of processing hardware in terms of production and usage,[8,9] despite their well-known inefficiency compared to ASIC- and FPGA-based solutions. Given this mismatch between the extreme area and power constraints of emerging applications and the relative inefficiency of general-purpose microprocessors and microcontrollers compared to their ASIC counterparts, there is a considerable opportunity to make microprocessor-based solutions for these applications much more area- and power-efficient.

Published by the IEEE Computer Society
0272-1732/18/$33.00 ©2018 IEEE

One big source of area inefficiency in a microprocessor is that a general-purpose microprocessor is designed to target an arbitrary application and thus contains many more gates than what a specific application needs. These unused gates continue to consume power, resulting in significant power inefficiency. While adaptive power management techniques (such as power gating) help reduce power consumed by unused gates, their effectiveness is limited due to the coarse granularity at which they must be applied, as well as significant implementation overheads such as domain isolation and state retention. These techniques also worsen area inefficiency.

One approach to significantly increasing the area and power efficiency of a microprocessor for a given application is to eliminate all logic in the microprocessor IP core that will not be used by the application. Eliminating logic that is guaranteed to not be used by an application can produce a design tailored to the application—a bespoke processor—that has significantly lower area and power than the original microprocessor IP that targets an arbitrary application. As long as the approach to creating a bespoke processor is automated, the resulting design retains the cost benefits of a microprocessor IP, because no additional hardware or software needs to be developed. Also, because no logic used by the application is eliminated, area and power benefits come at no performance cost. The resulting bespoke processor does not require programmer intervention or hardware support, either, because the software application can still run, unmodified, on the bespoke processor.

In this article, we present a methodology to automatically generate a bespoke processor for an application out of a general-purpose processor/microcontroller IP core. Our methodology relies on gate-level symbolic simulation that identifies gates in the microprocessor IP that cannot be toggled by the application, irrespective of the application inputs, and automatically eliminates them from the design to produce a significantly smaller and lower-power design with the same performance. Because the original design is pruned at the granularity of gates, the resulting methodology is much more effective than any approach that relies on coarse-grained application-specific customization. The proposed methodology can be used by either IP licensors or licensees to produce bespoke designs for the application of interest (see Figure 1). Simple extensions to our methodology can be used to generate bespoke processors that can support multiple applications or different degrees of in-field software programmability, "debuggability," and updates.

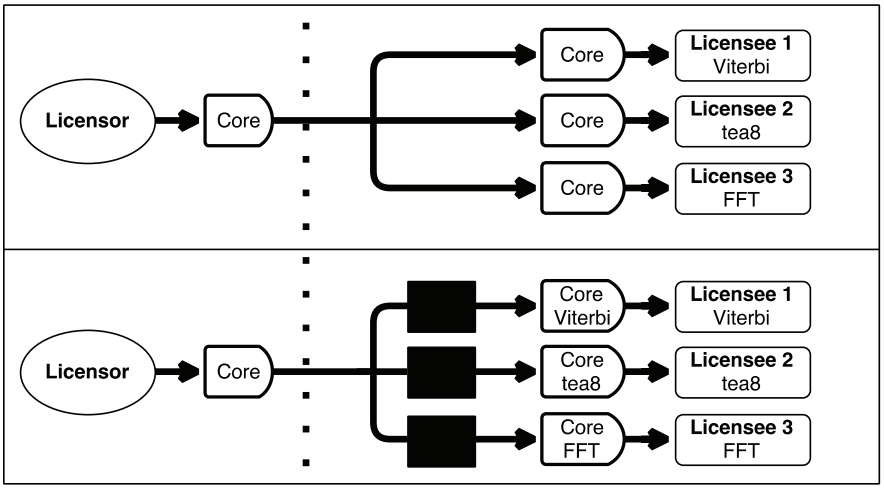

Figure 1. General-purpose processors are overdesigned for a specific application (top). A bespoke processor design methodology allows a microprocessor IP licensor or licensee to target different applications efficiently without additional software or hardware development cost (bottom).

WHY BESPOKE PROCESSORS?

Area- and power-constrained microprocessors and microcontrollers are the most abundant type of processor produced and used today, with projected deployment growing rapidly.[8,9] This explosive growth is fueled by emerging area- and power-constrained applications, such as the IoT,

wearables, implantables, and sensor networks. The microprocessors and microcontrollers used in these applications are designed to include a wide variety of functionalities to support many diverse applications with different requirements. On the other hand, the embedded systems designed for these applications typically consist of one application or a small number of applications, running over and over on a general-purpose processor for the lifetime of the system.[10] Given that a particular application might only use a small subset of the functionalities provided by a general-purpose processor, there might be a considerable amount of logic in a general-purpose processor that is not used by an application. Figure 2 illustrates this point, showing the fraction of gates in an openMSP430[11] processor that are not toggled when a variety of applications are executed on the processor with many different input sets. The bars in the figure show the intersection of all gates that were not exercised (toggled) by the application for any input, and the intervals show the range in fraction of unexercised gates across different inputs. For each application, a significant fraction (around 30-60 percent) of the processor's gates were not toggled during any execution of the application. These results indicate that there might be an opportunity to reduce area and power significantly by removing logic from the processor that cannot be exercised by the software running on the processor, if it can be guaranteed that removed logic will never be needed for any possible execution of the software.

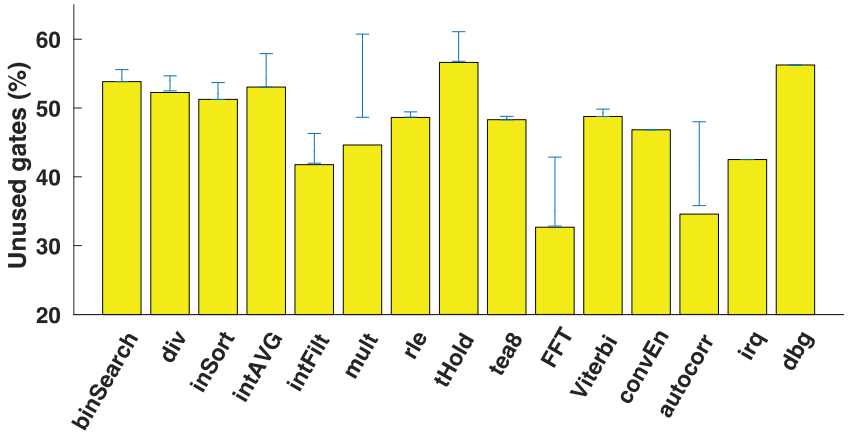

Figure 2. A significant fraction of gates in an openMSP430 processor are not toggled when an application executes. Each bar represents gates not toggled by any input for an application; the interval shows the range of unexercised gates for different inputs.

Identifying all the logic that is guaranteed to never be used by an application is not straightforward, however. One possible approach is profiling, wherein an application is executed for many inputs, and the set of gates that were never exercised is recorded, as in Figure 2. However, profiling cannot guarantee that the set of gates used by an application will not be different for a different input set. Indeed, profiling results in Figure 2 show considerable variations in exercised gates (up to 13 percent) for different executions of the same application with different inputs. Thus, an application might require different gates and execute incorrectly for an unprofiled input. Another possible approach is performing a static analysis of the instructions that are used in an application and the corresponding software-visible modules those instructions use. Unfortunately, different applications use different portions of logic at a fine granularity (for example, two applications might use a different set of gates within one module). Additionally, which logic is used also depends on the ordering of instructions within an application.

Because different applications can exercise substantially different parts of a processor at a fine granularity, and simply profiling or statically analyzing an application cannot guarantee which parts of the processor can and cannot be used by an application, tailoring a processor to an application requires a technique that can identify all the logic in a processor that is guaranteed to never be used by the application and remove unusable logic in a way that leaves the functionality of the processor unchanged for the application. The next section provides our approach to safely producing general-purpose processors that have been tailored to an individual application. We

call the resulting designs bespoke processors, reminiscent of bespoke clothing, in which a generic clothing item is tailored for an individual person.

TAILORING A BESPOKE PROCESSOR

A bespoke processor, tailored to a target application, must be functionally equivalent to the original processor when executing the application. As such, the bespoke implementation of a processor design should retain all the gates from the original processor design that might be needed to execute the application. Any gate that could be toggled by the application and propagate its toggle to a state element or output port performs a necessary function and must be retained to maintain functional equivalence. Conversely, any gate that can never be toggled by the application can safely be removed, as long as each fanout location for the gate is fed with the gate's constant output value for the application. Removing constant (untoggled) gates for an application could result in significant area and power savings and, unlike conventional energy-saving techniques, will introduce no performance degradation (indeed, no change at all in application behavior).

Figure 3 shows our process for tailoring a bespoke processor to a target application. The first step—input-independent gate activity analysis—performs a type of symbolic simulation,[12] where unknown input values are represented as Xs and gate-level activity of the processor is characterized for all possible executions of the application and for any possible inputs to the application. The second phase of our bespoke processor design technique—gate cutting and stitching—uses gate-level activity information gathered during gate activity analysis to prune away unnecessary gates and reconnect the cut connections between gates to maintain functional equivalence to the original design for the target application. (See the "Bespoke Processor Tool Flow Example" sidebar for an example of this tool flow.)

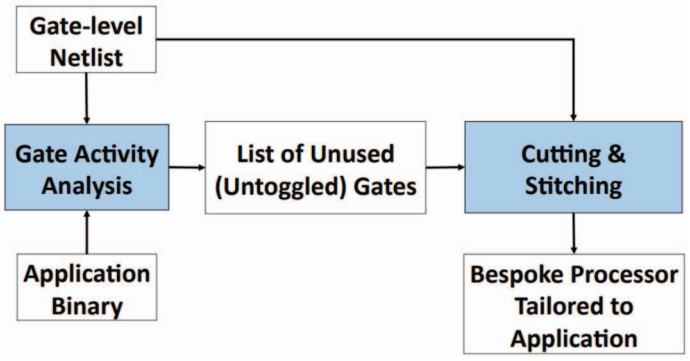

Figure 3. Bespoke processor tool flow. Our technique performs input-independent gate activity analysis to determine which gates of a processor cannot be toggled in any execution of the application. These gates are then cut from the design to form a custom, bespoke processor with reduced area and power.

Input-Independent Gate Activity Analysis

The set of gates that an application toggles during execution can vary depending on application inputs. This is because inputs can change the control flow of execution through the code, as well as the data paths exercised by the instructions. Because exhaustive profiling for all possible inputs is infeasible, and limited profiling might not identify all exercisable gates in a processor, we have implemented an analysis technique based on symbolic simulation that is able to characterize the gate-level activity of a processor executing an application for all possible inputs with a single gate-level simulation. During this simulation, inputs are represented as unknown logic values (Xs), which are treated as both 1s and 0s when recording possible toggled gates.

During input-independent gate activity analysis, the values of all memory cells and gates are initialized to Xs. The application binary is loaded into program memory, providing the values that

effectively constrain which gates can be toggled during execution. During simulation, our simulator sets all inputs to Xs, which propagate through the gate-level netlist during simulation. After each cycle is simulated, the toggled gates are removed from the list of unexercisable gates. Gates where an X propagated are considered toggled, because some input assignment could cause the gates to toggle. If an X propagates to the PC, indicating input-dependent control flow, our simulator branches the execution tree and simulates execution for all possible branch paths, following a depth-first ordering of the control flow graph.

Cutting and Stitching

Once gates that the target application cannot toggle have been identified, they are cut from the processor netlist for the bespoke design. After cutting out a gate, the netlist must be stitched back together to generate the final netlist and laid-out design for the bespoke processor. Figure 4 shows our method for cutting and stitching a bespoke processor. First, each gate on the list of unusable (untoggled) gates is removed from the gate-level netlist. After removing a gate, all fanout locations that were connected to the output net of the removed gate are tied to a static voltage (1 or 0) corresponding to the constant value of the gate observed during simulation. Because the logical structure of the netlist has changed, the netlist is re-synthesized after cutting all unusable gates to allow additional optimizations that reduce area and power. Because some gates have constant inputs after cutting and stitching, they can be replaced by simpler gates. Also, toggled gates left with floating outputs after cutting can be removed, because their outputs can never propagate to a state element or output port. Cutting can reduce the depth of logic paths, so some paths might have extra timing slack after cutting, allowing faster, higher-power cells to be replaced with smaller, lower-power versions of the cells. Finally, the re-synthesized netlist is placed and routed to produce the bespoke processor layout, as well as a final gate-level netlist with necessary buffers, introduced to meet timing constraints.

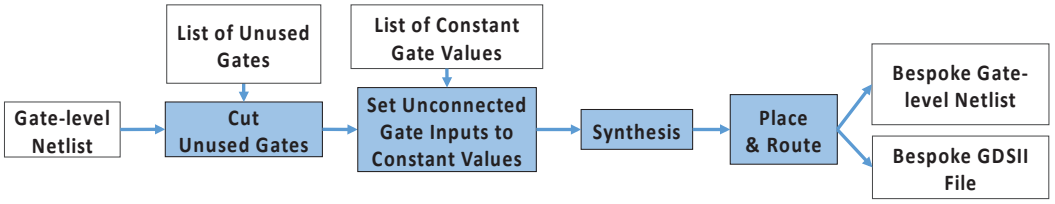

Figure 4. Tool flow for cutting and stitching.

RESULTS

Bespoke processors have fewer gates, lower area, and lower power than their general-purpose counterparts. Figure 5 shows the reduction in gates, area, and power afforded by bespoke processors tailored to several applications. Area savings are up to 92 percent, and power savings are up to 74 percent, relative to the baseline design. Even the application with the smallest gate count reduction (44 percent) reduces area by 47 percent and power by 37 percent. In this way, tailoring bespoke processors out of general-purpose processors provides power- and area-efficient designs with a low design cost.

CONCLUSION

Ultra-low-power (ULP) processors are already the most abundant type of processor manufactured and used today. Due to emerging trends like the IoT, computing systems built around these low-power processors are projected to be even more ubiquitous in the future. Considering the sheer number of low-power processors being produced, their importance for future technologies, and the stringent power and cost constraints of these systems, techniques to automatically reduce

the power and cost of such systems can have a significant enabling impact on the future of computing systems. This article focuses on an effective and automated tool to reduce the power and cost for such systems. We envision our work having several impacts.

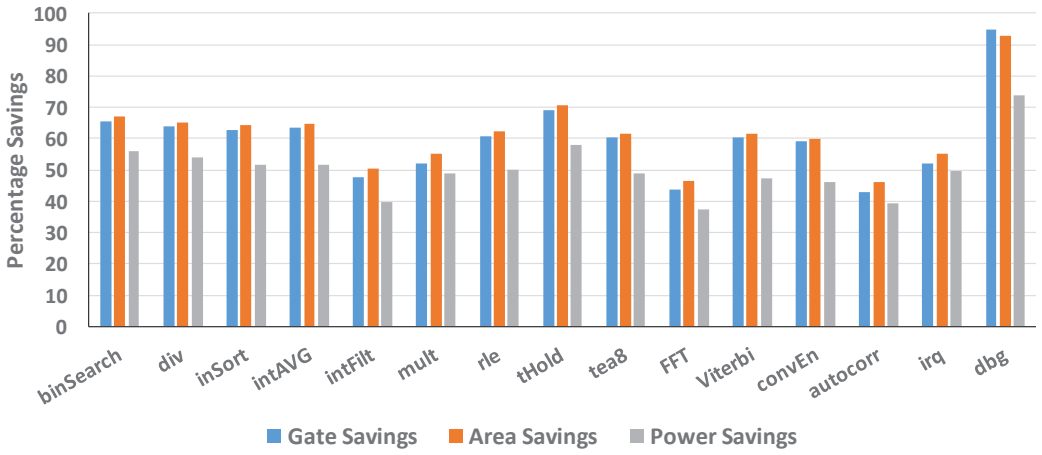

Figure 5. Percent reduction in gate count, area, and power for a bespoke design, compared to the baseline processor.

First, bespoke processors represent a new design point on the programmability and efficiency continuum. Emerging trends such as the IoT rely on the ability to generate millions of application-specific systems that are resource-constrained, because they are powered by energy harvesters or implemented in new, less-dense substrates that target IoT applications, such as printable thin-film processors. These opposing forces require customization to meet resource constraints, yet easy development to support the massive number of unique applications. To meet these challenges, the bespoke process presented here can effectively support efforts to make hardware design as approachable and popular as software development. At first glance, bespoke processors lower the design effort to produce a custom chip by providing pushbutton customization that an application developer can use without relying on a hardware designer. Digging deeper, the application developer gets all the benefits of having a mature general-purpose processor infrastructure. This includes having mature compilation and debug tool-chains, as well as extensive libraries available for rapid development. On top of the added tool-chain support, a developer can prototype the whole system using a full, non-custom version of the baseline processor rather than spending more time debugging system-level issues concurrently with any silicon-level bugs.

Second, the underlying approach of the bespoke processor tool flow establishes the foundations for the automatic editing of processors. Example uses of automatic editing of processors include the removal of instructions or components within a processor that are deprecated, contain bugs, or have security vulnerabilities. Another use occurs when processors are used to debug immature technology processes and limited, yet known functionality is required (such as being able to run a small number of code segments that perform certain I/O tasks on the new technology node).

Third, this work represents a new type of analysis: hardware-software co-analysis. The gate-level hardware-software co-analysis framework that forms the critical backbone of the bespoke processor approach can be leveraged in many other use cases. For example, the analysis can be used for the control of novel module-oblivious power-gating domains, for determining stricter peak power and energy constraints for ULP processors and for verifying software-based information-flow security on commodity ULP processors. These applications of the co-analysis provide several launch points from which further software and hardware techniques can be built.

In the coming IoT era, where extreme power and area efficiency must be balanced with design costs, bespoke processor design provides a balanced approach that allows designers to easily generate efficient hardware for a specific use case.

SIDEBAR: BESPOKE PROCESSOR TOOL FLOW EXAMPLE

Bespoke processor design is a fully automated process. Here, we illustrate how bespoke processor design tailors a processor design to a particular application. Figure 6 illustrates the bespoke design tool flow. The left part of Figure 6 shows input-independent gate activity analysis for a simple example circuit (top right). During symbolic simulation of the target application, logical 1s, 0s, and unknown symbols (Xs) are propagated throughout the netlist. Because tmp2 is never toggled during any of the possible executions of the application, Gate c is marked for cutting, and its constant output value (1) is stored for stitching. Although Gate d is never toggled in Cycles 0-2 nor down the left execution path, it does toggle in the right execution path and thus cannot be marked for cutting. Gates a and b also toggle and thus are not marked for cutting.

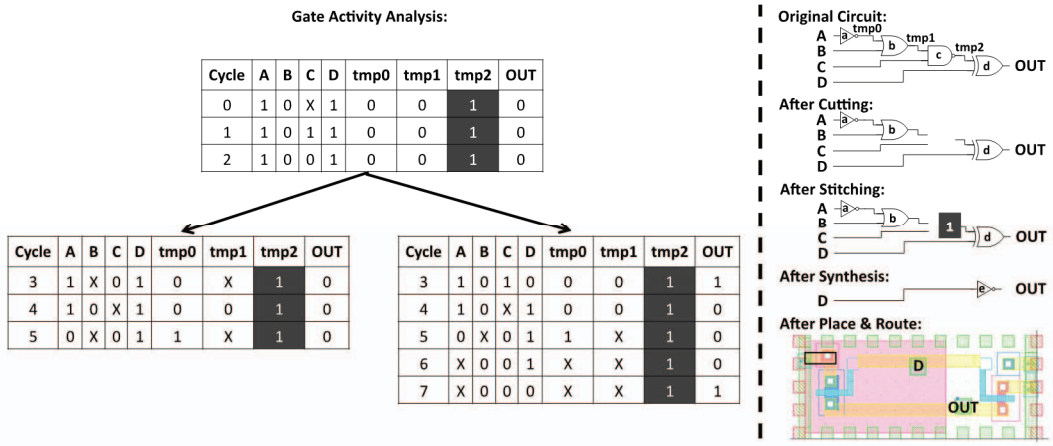

Figure 6. An end-to-end example of the bespoke processor tool flow.

Once gate activity analysis has generated a list of unexercisable (cuttable) gates and their constant values, cutting and stitching begins. Because Gate c was marked for cutting, it is removed from the netlist, leaving the input to its fanout (d) unconnected. During stitching, d's floating input is connected to c's known constant output value for the application (1). After stitching, the gate-level netlist is re-synthesized. Synthesis removes gates that are not driving any other gates (Gates a and b), even though they toggled during symbolic simulation, because their work does not affect the state or output function of the processor for the application. Synthesis also performs optimizations, such as constant propagation, which replaces Gate d with an inverter, because the constant controlling input of 1 to the XOR gate makes it function as an inverter. Finally, place and route produces a fully laid-out bespoke design.

ACKNOWLEDGMENTS

We thank James Myers and the anonymous reviewers for helpful suggestions and feedback.

REFERENCES

1. M. Magno et al., "Wearable low-power dry surface wireless sensor node for healthcare monitoring application," *IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2013, pp. 189–195.
2. S. Narasimhan, H. Chiel, and S. Bhunia, "Ultra-low-power and robust digital-signal-processing hardware for implantable neural interface microsystems," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 5, no. 2, 2011, pp. 169–178.

3. K. Myny et al., "A thin-lm microprocessor with inkjet print-programmable memory," *Scientific Reports*, 2014.
4. B. Ransford, J. Sorber, and K. Fu, "Mementos: system support for long-running computation on RFID-scale devices," *ACM SIGPLAN Notices*, 2012.
5. G. Hackmann et al., "Cyber-Physical Codesign of Distributed Structural Health Monitoring with Wireless Sensor Networks," *IEEE Transactions on Parallel and Distributed Systems*, 2014.
6. K. Myny et al., "An 8b organic microprocessor on plastic foil," *IEEE International Solid-State Circuits Conference*, 2011.
7. B.K. Kjellander et al., "Optimized circuit design for exible 8-bit RFID transponders with active layer of ink-jet printed small molecule semiconductors," *Organic Electronics*, 2013.
8. H. Blodget et al., "The Internet of Everything: 2015," Business Insider, 2014.
9. "Microcontroller Sales Regain Momentum after Slump," *IC Insights*, 2017; www.icinsights.com/news/bulletins/Microcontroller-Sales-Regain-Momentum-After-Slump.
10. "Products with an MSP430," 43oh, 2012; http://43oh.com/2012/03/winner-products-using-the-msp430/.
11. O. Girard, *OpenMSP430 Project*, 2013.
12. R. Bryant, "Symbolic Simulation -- Techniques and Applications," *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC)*, 1990, pp. 517–521.

ABOUT THE AUTHORS

Hari Cherupalli is a PhD candidate at the University of Minnesota. His research interests are in power management in ULP processors and hardware security. He has a master's degree in electrical engineering from the Indian Institute of Technology Kharagpur. Contact him at cheru007@umn.edu.

Henry Duwe is an assistant professor at Iowa State University. His research interests range from automated generation of application-specific processors to designing secure low-power processors and architecting transient processors. Contact him at duwe@iastate.edu.

Weidong Ye has a master's degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign. He previously studied at Arizona State University. Contact him at wye5@illinois.edu.

Rakesh Kumar is an associate professor in the Electrical and Computer Engineering Department at the University of Illinois at Urbana-Champaign. His research and teaching interests are in computer architecture, hardware design, and low-power, trustworthy, and error-resilient computer systems. Contact him at rakeshk@illinois.edu.

John Sartori is an assistant professor at the University of Minnesota. His research interests include computer architecture, electronic design automation, embedded systems, and algorithm development, especially focused on energy-efficient computing, high-performance computing, stochastic computing, and application-aware design and architecture methodologies. Contact him at jsartori@umn.edu.