Energy Advantages of Microprocessor Platforms with On-Chip Configurable Logic

Greg Stitt and Frank Vahid
University of California, Riverside

System chips that incorporate configurable logic can reduce the energy consumed in executing software. The key is to use the configurable logic to execute performance-critical loops, producing an average energy savings of 25% to 71% for embedded-system benchmarks.

One recent trend in commercial platforms is the integration of on-chip, configurable logic (also known as FPGAs) with a microprocessor. Examples include 8-bit platforms, such as Triscend's E5 and Atmel's FPSLIC (Field-Programmable System-Level IC); and 32-bit platforms, such as Triscend's A7, Xilinx's Virtex II Pro, and Altera's Excalibur. A main benefit of on-chip configurable logic is that it supports the incorporation of different peripherals in the design; rather than having a different chip for each of various sets of peripherals, vendors can offer a single chip. Designers can also use configurable logic to implement coprocessors that would otherwise require a separate ASIC chip. We propose yet another use of configurable logic: reducing the energy consumed by the platform in executing software. With the rapid increase in battery-operated systems, low energy consumption is becoming increasingly important.

Our partitioning approach involves profiling an application to identify the critical loops—those that contribute the most to an application's execution time. Instead of allowing these loops to execute in software, we reimplement them in hardware, using the configurable logic. We modify the original software by replacing the loop with code to enable the configurable logic, and to stall the microprocessor and put it into a power-down sleep mode. When the configurable logic finishes execution, it signals an interrupt that causes the microprocessor to resume normal execution. This approach speeds up an application's execution time, meaning we can put the microprocessor in an inactive, low-power state for longer periods of time. Alternatively, we can execute the loop in the same amount of time with the same performance by slowing down the clock and reducing the microprocessor's supply voltage. Lower voltage results in less energy.

Using an ASIC coprocessor to reduce energy consumption is common.[1,2] But, because of configurable logic's high power consumption, partitioning that uses configurable logic has mainly targeted speedup.[3-5] Energy is the product of power and time, so the power increase from using configurable logic could outweigh the time savings. In this article, derived from a paper presented at the 2002 IEEE Symposium on Field-Programmable Custom Computing

0740-7475/02/$17.00 © 2002 IEEE

Machines,[6] we show that using on-chip configurable logic can in fact reduce the software execution's energy consumption on microprocessor/configurable-logic platforms. We experimented with three platforms:

- a hypothetical 0.18-micron single-chip platform based on a 32-bit MIPS microprocessor and a Xilinx Virtex XCV50E FPGA supporting 25,000 gates (real single-chip platforms were not readily available when our experiments began);
- a Triscend (http://www.triscend.com) E5 chip, a 0.35-micron single-chip platform that combines an 8-bit 8051 microprocessor with configurable logic supporting 25,000 gates; and
- a Triscend A7 chip, a newer 0.25-micron single-chip platform that combines a 32-bit ARM microprocessor with configurable logic supporting 25,000 gates.

Low-energy partitioning

Our partitioning approach involves moving critical loops to hardware. Besides dominating execution time, these loops tend to be small, implying that a hardware implementation will likely require little area and power. We considered the following programs from Motorola's Powerstone benchmarks:[7]

- a voice encoder (adpcm),
- bit reversal graphics application (brev),
- cyclic redundancy check (crc),
- data encryption standard (des),
- engine controller (engine),
- fax decoder (g3fax),
- JPEG decoder (jpeg),
- matrix multiplication (matmul),
- handwriting recognizer (summin), and
- modem encoder/decoder (v42).

Villarreal et al. give complete results of a loop study on these benchmarks.[8] The main results showed that programs running on a MIPS processor spent 66% of their time in loops with a static size of 256 instructions or fewer, whereas on an 8051 processor, the benchmarks spent 76% of their time in such loops.

After identifying the most-critical loops, we manually translated the C code for the one (or at most two) most-critical loop into a VHDL description to create a hardware implementation. For two loops, we wrote the VHDL such that the loops could share hardware. In some situations, we implemented less-critical loops because the most-critical loops used too much area, were not power efficient, or were unsuitable for hardware implementation.

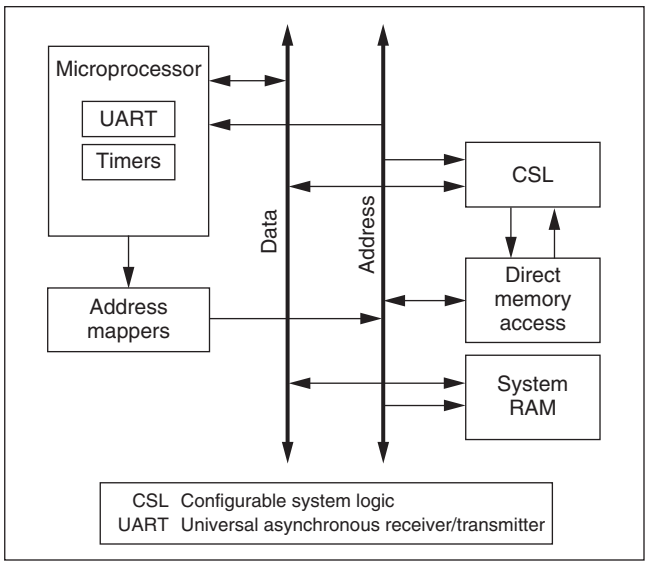


Figure 1. Triscend E5 single-chip architecture.

We based our model of communication between the configurable logic and microprocessor on the Triscend E5 architecture, which Figure 1 shows. We refer to the on-chip FPGA as configurable system logic (CSL). In this architecture, the microprocessor and CSL can communicate directly or through shared memory. We used direct communication to implement handshaking between the hardware and software. Therefore, we replaced the software loop with code that enables the hardware and puts the microprocessor into a low-power state. The hardware then executes, and signals the microprocessor upon completion, causing software execution to resume.

Power and performance evaluation

We used the test benches that come with the Powerstone benchmarks to generate power and performance data. In our MIPS platform evaluation, we used a simulation-based approach for performance evaluation. We ran

Table 1. Benchmark profiles running on a MIPS processor.

Benchmark	Size (no. of instruction bytes)	Loop size (no. of instruction bytes)	Loop time (%)	Speedup bound
g3fax	4,452	24	31	1.45
adpcm	7,640	152	30	1.43
crc	4,288	68	66	2.90
des	6,116	360	52	2.08
engine	4,432	64	28	1.39
jpeg	5,960	116	10	1.11
summin	4,136	100	48	1.92
v42	6,388	60	23	1.30
Average	5,427	118	36	1.70

each example on a MIPS architectural simulator that output the number of cycles that each example executed. To determine the MIPS processor's power, we used the reported power consumption of the 0.18-micron, MIPS32 4-Kbyte processor core. All examples ran at 100 MHz with a 1.8-V supply voltage.

These estimations assumed that the configurable logic consisted of a 0.18-micron Virtex XCV50E FPGA from Xilinx, and we used Xilinx's Virtex Power Estimator to estimate the configurable logic's power. To make its estimations, the Power Estimator tool requires an estimate of the average percentage of FPGA internal nets that switch per clock cycle. The tool's user guide lists a typical percentage as 6% to 12%. Hence, we used 9% as our percentage. We evaluated hardware performance for a given loop by counting the number of cycles that the synthesized hardware required to execute the loop. Because some loops had multiple paths, we conservatively considered only the longest path.

We assumed that the MIPS platform had an interconnect power (the power that the system buses and shared memory consume) of about 0.1 W; we obtained this number by experimenting with the E5 platform. We also assumed that the microprocessor had a low-power state that consumed only 25% of the power of the active state, and we assumed that the CSL low-power state included only quiescent power. Hence, we used the following equation to compute total power P_T:

$$P_T = Sw \times P_{Sw} + CSL(P_{CSL} + 0.25P_{Sw}) + P_I + P_Q$$

where Sw is the percentage of time spent in software, CSL is the percentage of time spent in the CSL, P_{Sw} is the software's power when the microprocessor is active, P_{CSL} is the CSL's power when active, P_I is the interconnect power, and P_Q is the quiescent power.

For the E5 and A7 platforms, we used a digital multimeter to physically measure the current, and hence the power (knowing the supply voltage), for our examples. Each example executed as an all-software implementation and then as a partitioned implementation, using low-power modes for the microprocessor and CSL when the other was active. To obtain a stable power reading, each example ran as part of a long-running loop. We measured performance by using the on-chip serial communication to specify the application's start and end. A timer running on a workstation measured the differences between the starting and ending times to determine the actual execution times.

For the MIPS examples, we used Xilinx tools for all synthesis, placement, and routing. For the E5 and A7, we synthesized the selected loop regions for the CSL using the Synopsys FPGA Compiler, and we used the Triscend tools for placement, routing, and mapping.

Estimated results for the MIPS platform

Table 1 contains basic information about how eight of the largest Powerstone benchmarks run on a MIPS processor. Size corresponds to the application's static size in terms of instruction bytes. Therefore, the number of instructions is the size listed in Table 1 divided by 4, since MIPS instructions are 4 bytes each. Loop size is the static number of instruction bytes in the most frequent loops. Loop time is the percentage of total dynamic instructions executed in these loops. Speedup bound is the approximate speedup that the application

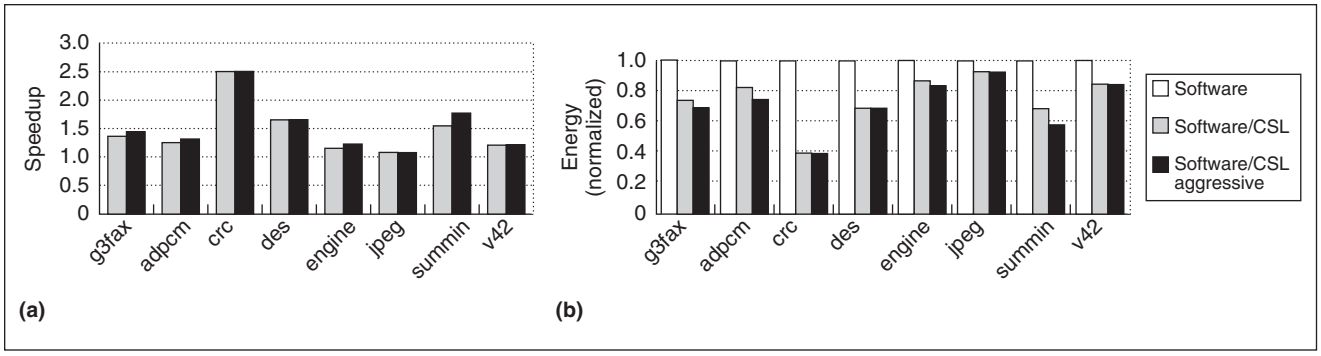


Figure 2. Estimated results for MIPS platform: speedup (a) and energy consumption (b) for partitioned examples.

would achieve if the configurable logic executed the loops in zero time.

Figure 2 shows speedup and energy results for the Powerstone benchmarks running on the MIPS platform. We normalized all energy results, using the energy consumption of an all-software implementation as the normalization factor. The results are from two different types of hardware implementations:

- a standard translation from C code to VHDL, and
- a more aggressive translation using loop unrolling and pipelining optimizations.

The average speedup for the standard translation was 1.46. For the aggressive translation, the average speedup was 1.51. (Aggressive optimizations don't gain much, because of limited memory bandwidth.) We achieved these speedups by moving small amounts of the original application to hardware running on the CSL. In fact, the average percentage of total assembly instructions used by the loops moved to the CSL was only 2%. This corresponds to an average of 30 instructions and an average of only about 4,000 gates. These loops account for an average of 36% of the total execution time, and this percentage was as high as 66% for the crc benchmark.

The CSL, when active, consumed an average of 67% more power than the microprocessor when it was active. Despite this significant difference, total power increased by only 3% compared to the software-only version. The reason for this small increase is that in the partitioned example, the CSL was active for only 10% of the total execution time.

The average energy savings was 25%. If we had used a CSL switching frequency of 12% rather than 9%, the energy savings would have been 21%. For the aggressive optimizations, the average savings was 28%.

The energy results represent the amount of energy required to accomplish a particular task. Assuming the system consumes almost no energy during the task period's idle time, we get overall energy savings. Alternatively, we could take advantage of the speedup to reduce the clock speed, achieving approximately the same energy results. Clock scaling is possible on many platforms, including the Triscend E5 and A7.

Measured results for the E5 platform

We implemented four of the Powerstone benchmarks on the 8051-based E5 platform—fewer benchmarks than on the hypothetical MIPS platform because implementing on the physical platform is more time-consuming. Different examples were used because different groups conducted these experiments at different times than for the MIPS experiments. Table 2 (next page) contains information about those four benchmarks run on an 8051 processor, providing the same type of information that was in Table 1. The number of instructions ranges from the size listed in Table 2 to the size divided by 3, because 8051 instructions range from 1 to 3 bytes in length.

The crc and brev results in Tables 1 and 2 come from a partitioning of the software algorithms given in the benchmark code, even though specific hardware algorithms exist to

Table 2. Benchmark profiles running on an 8051 processor.

Benchmark	Size (no. of instruction bytes)	Loop size (no. of instruction bytes)	Loop time (%)	Speedup bound
g3fax	8,309	71	56	2.27
crc	810	58	63	2.69
brev	2,505	1,710	92	12.99
matmul	838	212	94	17.24
Average	3,116	513	76	8.80

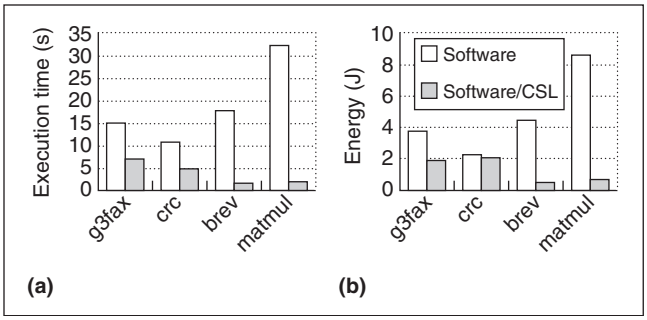


Figure 3. Measured results for Triscend E5 platform: execution times (a) and energy consumption (b) for the software and partitioned examples.

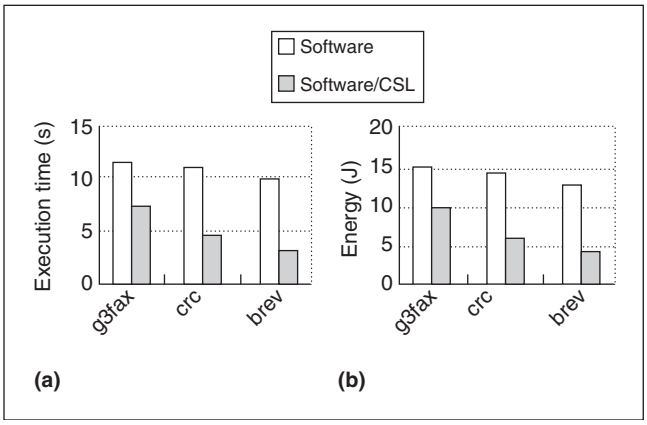


Figure 4. Measured results for Triscend A7 platform: execution times (a) and energy consumption (b) for software and partitioned examples.

accomplish the same task. Therefore, changing these benchmarks to use hardware algorithms would result in even greater savings.

Figure 3 shows speedup and energy results on the E5 for those four Powerstone benchmarks. On average, execution time decreased by 66%, corresponding to a speedup of 7.5. The E5 achieved this speedup by moving an average of 513 instruction bytes into the CSL. Excluding the brev benchmark, which by far uses the largest number of instructions in the loop, the average number of instruction bytes moved to hardware was 114. The reason the 8051 requires more loop instructions than the MIPS processor is that the 8051 is an 8-bit processor, and implementing even a small loop can require many instructions. Overall, the 513 loop instruction bytes account for approximately 76% of the total execution time. Speedups obtained by partitioning for the 8051-based E5 platform are much greater than for the MIPS or A7 platforms because the 8051 is a far slower processor; thus, using configurable logic has a greater impact.

Total power for the partitioned implementations increased by an average of only 7% compared with the software-only version. The reason for this small increase is that the partitioned examples required very little configurable logic, and this logic was active for only a short time. The average energy savings was 71%.

Measured results for A7 platform

We also tested energy savings on the Triscend A7 chip, which combines a 32-bit ARM processor with configurable logic. Figure 4 shows the speedup and energy results achieved on three of the Powerstone benchmarks.

The average reduction in execution time was 53%. This corresponds to a speedup of 2.3. Power consumption increased by an average of only 1% when using the CSL. This small increase was also due to a very small area overhead and to the CSL being active for only a short time. The average energy savings was 53%. By comparing the g3fax and crc data in Figure 2 with that in Figure 4, we see that our

estimation results for the MIPS platform matches reasonably with our measured results for the A7 platform.

Size requirements

In the previous sections, we looked at the speedup obtained from partitioning one or two frequent loops into the CSL. Table 3 provides the number of gates required to obtain those speedups for the Xilinx Virtex XCV50E FPGA (MIPS platform) and the Triscend E5 and A7 CSL. We see that surprisingly few gates are needed—just a few thousand, on average.

The Powerstone benchmarks are fairly small programs. To further investigate the size requirements of on-chip CSL to speed up critical loops, we have recently begun investigating the MediaBench benchmark suite,[9] a set of larger programs typical in embedded and media-processing systems. Figure 5 shows results on three MediaBench programs for our MIPS-based platform.

For the G721 example (a voice-encoding application), we achieved a reasonable speedup of 1.8 by using only 1,307 gates. With an additional loop, the speedup increased slightly to 2.2 but required 5,811 gates. After this point, any additional speedup requires many more gates, which in most cases outweighs the performance benefits.

The adpcm (adaptive, differential pulse code modulation) example is interesting because the processor spends more than 90% of the execution time in a small loop. By implementing this loop in hardware, we achieved a speedup of 27.2 using 14,132 gates. Implementing any other software regions in hardware gives minimal improvements because the original loop accounts for such a large percentage of the execution time.

The Pegwit example (a public-key-encryption program), shows a steadier speedup increase than the other two examples. By using 4,301 gates, we can achieve a speedup of 1.5. Speedup increases dramatically to 3.0 for 13,419 gates. The speedup then slowly starts to level off, reaching 3.4 at 15,678 gates and 3.7 at 18,150 gates. After that point, any additional partitioning has a very large area overhead.

Triscend's current offerings of single-chip microprocessors combined with CSL parts have from 5,000 to 40,000 gates of CSL. New Xilinx and Altera single-chip platforms can have several hundred thousand CSL gates. Therefore, these current offerings can easily support the amount of CSL required to obtain these speedups.

Table 3. FPGA size requirements for the Powerstone benchmarks.

Benchmark	Xilinx Virtex FPGA (no. of gates)	Triscend E5/A7 CSL (no. of gates)
g3fax	2,737	1,896
adpcm	7,973	NA
crc	782	1,044
des	8,772	NA
engine	2,261	NA
jpeg	2,669	NA
summin	3,604	NA
v42	3,961	NA
brev	NA	1,692
matmul	NA	6,468
Average	4,095	2,775

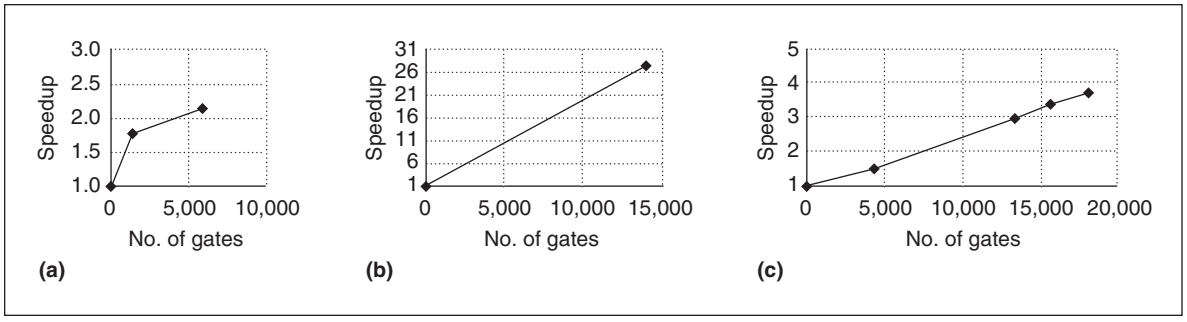


Figure 5. Relationship between size and speedup for several MediaBench examples: G721 (a); adpcm (b); and Pegwit (c).

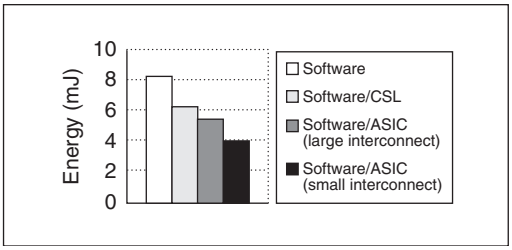


Figure 6. Comparison of estimated energy savings for MIPS-CSL design and MIPS-based ASIC design.

Estimated ASIC results for the MIPS platform

Platforms with on-chip logic are useful not only for final products but also for ASIC prototyping. After verifying system functionality using configurable logic, a designer could create an ASIC with the necessary microprocessor and on-chip coprocessing hardware to achieve improved power, performance, size, and cost. Such an ASIC could also include additional components on-chip. We therefore estimated the speedup and energy savings for ASIC versions of our examples, assuming a 200-MHz ASIC design.

Figure 6 compares energy savings between the MIPS-CSL chip and a MIPS-based ASIC design. These results represent the average energy required for the eight Powerstone benchmarks in Table 1. You might notice that the magnitude of energy is far smaller than in Figures 3 and 4. This is due to the use of 0.18-micron rather than 0.25- or 0.35-micron technology, and also to the noninclusion of peripheral components. As we mentioned earlier, using the CSL yields energy savings of 25%. If we simply replace the CSL with an ASIC coprocessor version of the hardware, the average energy savings increases to 33%. However, this assumes that we use the same interconnect. Because ASIC interconnects can be more efficient, we also show data based on the assumption that the power for the ASIC interconnect is half that for the CSL. In this case, the average energy savings increases to 53%.

The average speedup achieved by the ASIC was 1.57, compared to 1.46 for the MIPS-CSL chip. This additional speedup was due to the ASIC's higher clock frequency. The power consumed by the ASIC coprocessor hardware was only 11% of the CSL power. This low power consumption didn't result in significantly larger energy savings, mainly because the ASIC coprocessor was active for only 5% of the benchmarks' total execution time.

OUR EXPERIMENTS indicate that software execution can be made significantly faster and more energy efficient by using a surprisingly small amount of configurable logic available with a microprocessor on a single-chip platform. The data suggests that embedded microprocessor platform vendors might seriously consider including configurable logic for improved software execution, and that software compiler vendors for such platforms might consider including automatic hardware/software partitioners in their compilers. Existing vendors of such platforms should consider targeting low-power applications and focus on minimizing the power consumption of the various platform components.

Our data also suggests that such single-chip platforms can achieve most of the available software speedup and about half of the available energy savings as an ASIC. So, when compared to the cost, time, and risk associated with creating an ASIC, such prefabricated platforms appear extremely practical.

A future technical challenge is to create a hardware/software partitioning approach that fits easily with existing embedded-system tool flows. Typical embedded-system flows don't incorporate complex and expensive toolsets that are common in ASIC flows. We are therefore developing partitioning methods that minimize the impact on existing embedded-system tool flows, including binary-level partitioning and even dynamic on-chip partitioning. To further improve the speedup and energy results of such partitioning, we are also examining methods to include both software and hardware algorithms into software descriptions. ■

Acknowledgments

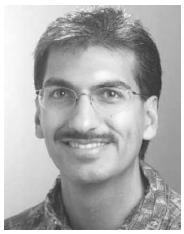
This work was supported in part by the National Science Foundation (CCR-9876006), the University of California Microelectronics Inno-

vation and Computer Research Opportunities program, and a Department of Education Graduate Assistance in Areas of National Need (GAANN) fellowship. We thank Brian Grattan for his assistance running experiments on the Triscend platforms, Jason Villarreal for developing the loop analysis tools, and Shawn Nematbakhsh for his help with partitioning the MediaBench programs.

References

1. J. Henkel, "A Low Power Hardware/Software Partitioning Approach for Core-Based Embedded Systems," *Proc. 36th Design Automation Conf.* (DAC 99), ACM Press, New York, 1999, pp. 122-127.
2. M. Wan et al., "An Energy Conscious Methodology for Early Design Exploration of Heterogeneous DSPs," *Proc. IEEE Custom Integrated Circuits Conf.* (CICC 98), IEEE Press, Piscataway, N.J., 1998, pp.111-117.
3. P. Athanas and H. Silverman, "Processor Reconfiguration through Instruction-Set Metamorphosis," *Computer*, vol. 26, no. 3, Mar. 1993, pp. 11-18.
4. J. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," *Proc. 5th Ann. IEEE Symp. FPGAs for Custom Computing Machines* (FCCM 97), IEEE Press, Piscataway, N.J., 1997, pp. 12-21.
5. M. Wirthlin and B. Hutchings, "A Dynamic Instruction Set Computer," *Proc. IEEE Symp. FPGAs for Custom Computing Machines* (FCCM 95), IEEE Press, Piscataway, N.J., 1995, pp. 99-107.
6. G. Stitt et al., "Using On-Chip Configurable Logic to Reduce Embedded System Software Energy," *Proc. 10th IEEE Symp. Field-Programmable Custom Computing Machines* (FCCM 02), IEEE CS Press, Los Alamitos, Calif., 2002.
7. A. Malik, B. Moyer, and D. Cermak, "A Low Power Unified Cache Architecture Providing Power and Performance Flexibility," *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED 00), ACM Press, New York, 2000, pp. 241-243.
8. J. Villarreal et al., *Loop Analysis of Embedded Applications*, tech. report UCR-CSE-01-03, Computer Science and Eng. Dept., Univ. of California, Riverside, 2001.
9. C. Lee, M. Potkonjak, and W. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems," *Proc. 30th Ann. IEEE/ACM Int'l Symp. Microarchitecture* (MICRO 30), IEEE CS Press, Los Alamitos, Calif., 1997, pp. 330-335.

Greg Stitt is a PhD student in the Department of Computer Science and Engineering at the University of California, Riverside. His research interests include low-power embedded-system design. Stitt has a BS in computer science from the University of California, Riverside. He is a member of the IEEE.



Frank Vahid is an associate professor in the Department of Computer Science and Engineering at the University of California, Riverside, and a member of the Center for Embedded Computer Systems at the University of California, Irvine. His research interests include architectures and design methods for low-power embedded systems, with an emphasis on tuning system-on-a-chip platforms to software programs. Vahid has a BS in computer engineering from the University of Illinois at Urbana-Champaign, and an MS and PhD in computer science from the University of California, Irvine. He is a member of the IEEE and the ACM.

■ Direct questions and comments about this article to Frank Vahid, Dept. of Computer Science and Engineering, College of Engineering, Univ. of California, Riverside, CA 92521; vahid@cs.ucr.edu; http://www.cs.ucr.edu/~vahid.