Using On-Chip Configurable Logic to Reduce Embedded System Software Energy

Greg Stitt[1], Brian Grattan[2], Jason Villarreal[1] and Frank Vahid[1,3]

[1] *Department of Computer Science and Engineering*
[2] *Department of Electrical Engineering*
University of California, Riverside
{gstitt/bgrattan/villarre/vahid}@cs.ucr.edu, http://www.cs.ucr.edu/~vahid
[3] *Also with the Center for Embedded Computer Systems at UC Irvine*

Abstract

We examine the energy savings possible by re-mapping critical software loops from a microprocessor to configurable logic appearing on the same-chip in commodity chips now commercially available. That logic is typically intended to implement peripherals and coprocessors without increasing chip count – but we show that reduced software energy is an additional benefit, making such chips even more useful. We find critical software loops and re-implement them in the configurable logic such that a repeating software task completes sooner, allowing us to put the system in a low-power state for longer periods, thus reducing energy. We use simulations and estimations for a hypothetical device having a 32-bit MIPS processor plus configurable logic, yielding energy savings of 25%, increasing to 39% assuming voltage scaling. We physically measured several examples running on two commercial single-chip devices having an 8-bit 8051 microprocessor plus configurable logic and a 32-bit ARM microprocessor with configurable logic, with energy savings of 71% and 53% respectively, increasing to an estimated 89% and 75% assuming voltage scaling.

1. Introduction

Commercial products are beginning to appear that incorporate a microprocessor and configurable logic on a single chip, such as Triscend's 8051-based E5 chips and Arm-based A7 chips [21], Atmel's FPSLIC [3], Xilinx's Virtex II Pro [28], and Altera's Excalibur [1]. One purpose of the on-chip logic is to support the incorporation of different numbers and types of peripherals, such as timers, UARTs (universal asynchronous receiver/transmitters), pulse-width modulators, etc. An embedded system designer can thus configure such a chip to have exactly the right combination of peripherals. Such configuration is in contrast to the more typical situation of a designer having to choose from tens or hundreds of different chip variations, each with different combinations of peripherals. The on-chip logic approach enables very high volume production of a single chip design, resulting in reduced cost per chip due to better amortized non-recurring engineering costs, higher yield, and other economy of scale benefits related to chip fabrication – the Triscend E5 series, for example, is expected to sell for only around $3 per chip [4].

A second purpose of the on-chip logic is to implement coprocessors and other custom logic that would otherwise have been implemented using a separate application-specific integrated circuit (ASIC) chip. This reduces the number of chips in a system, which can have extremely important advantages in cost and size sensitive embedded systems.

We propose that available configurable logic can be additionally used for a third purpose, namely for reducing the energy of the software running on the microprocessor. Low energy is an increasingly important requirement in embedded systems, allowing for longer battery life, among other things.

Our approach makes use of the situation of embedded systems typically executing a single program for their lifetimes – the program is "fixed." For example, the program executing on a microprocessor in a digital camera typically does not change during the camera's lifetime. The approach also makes use of the situation of many embedded system programs spending a large percentage of their runtime in a few small loops, and of such programs repeating a main task with a given period.

In short, the approach consists of profiling the fixed program using representative input stimuli, detecting the most critical loops, re-implementing the program so that some subset of those loops execute on the configurable logic while the microprocessor waits in a low-power state, and taking advantage of the resulting faster task completion by putting the entire system in a low-power state during the greater idle-time of the task's period. Alternatively, we can take advantage of the faster task completion to run the entire system at a lower voltage, until the task executes at the same speed as the original all-software implementation – yielding even greater energy savings than the low-power state method.

Although software speedups using configurable logic have been reported by many previous researchers, such speedup does not necessarily imply reduced energy. Configurable logic tends to consume much more power than software. Since energy is the product of power and time, the power increases that accompany speedup could result in energy decreases or increases.

In this paper, we present our hardware/software partitioning approach and the energy results of such partitioning through estimation for a 32-bit MIPS-based device, and through physical measurements of two low-cost devices having an 8-bit 8051 microprocessor with on-chip configurable logic and a similar 32-bit ARM system. We also estimate the additional savings had those three devices been voltage scalable.

2. Previous Work

Extensive previous work has examined the performance benefits of partitioning a program among a microprocessor and configurable logic (e.g., [2][12]), including systems that can page new hardware in and out on demand [10][27], as well as language support for partitioning [9]. Such hardware/software partitioning can integrate the logic at various levels with the microprocessor, including within the datapath, on the memory bus, on the peripheral bus, or even through a special port of the microprocessor. Tighter integration tends to support the addition of custom microprocessor instructions, while more distant integration tends to support co-processing.

Extensive work on general hardware/software partitioning, in which the hardware component is implemented on configurable logic or as an ASIC, has also been performed (e.g., [5][6][17][22]).

Improved performance through hardware/software partitioning could result in reduced energy, but not necessarily so. Energy is the product of power and time. While using configurable logic may reduce time, it potentially could increase power such that the energy savings are very small, or even such that energy is increased.

Some researchers have begun investigating the energy advantages of hardware/software partitioning. Henkel [13][14] partitioned several examples among a microprocessor and semi-custom ASIC, while utilizing low-power idle modes of the microprocessor and custom hardware, obtaining average energy savings of 60% compared to an all software solution. Wan et al [26] partitioned DSP algorithms among the Pleiades project's architecture of a microprocessor connected through an interconnection network to hardware accelerators programmable at multiple levels of granularity, showing 10 times overall energy improvements for certain examples, obtained due to the reduced microprocessor clock enabled by the partitioning.

Our work differs from Henkel's and Wan's in that we utilize standard on-chip configurable logic available in current commodity parts, as opposed to an ASIC or a custom multi-level configurable architecture. While such a commodity part may not achieve as great of energy savings, it does have significant cost and time-to-market advantages over other approaches.

3. Loop Analysis

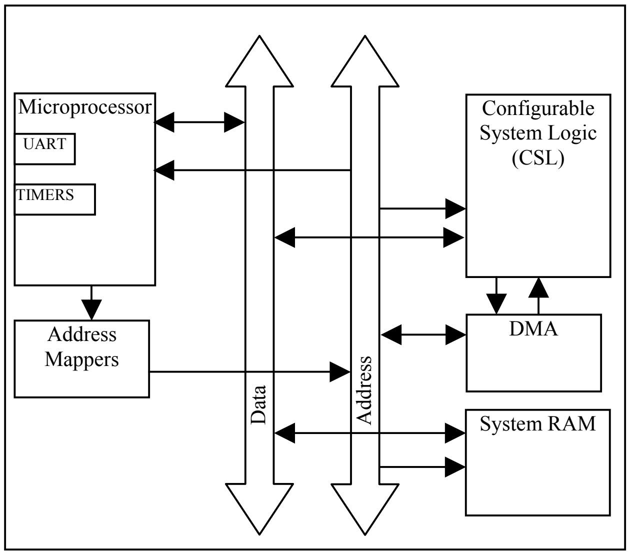
We began by profiling a number of examples from Motorola's Powerstone embedded software benchmarks [18] to determine to what extent they spent time executing small loops. If much of their time was spent in a few small highly-iterating loops, then mapping those loops to configurable logic could yield considerable energy improvements. On the other hand, if most time was spent in a few very large loops, or in a large number of smaller loops, then those loops' sizes in hardware might exceed the available on-chip logic capacity. Furthermore, if most time was spent in small loops but those loops only iterated a few times per loop visit, then the cost of switching back and forth between software and hardware would likely outweigh potential savings.

We examined 16 examples, including a voice encoder (*adpcm*), a cyclic redundancy check (*crc*), a data encryption standard (*des*), an engine controller (*engine*), a fax decoder (*g3fax*), a JPEG decoder (*jpeg*), a handwriting recognizer (*summin*), and a modem encoder/decoder (*v42*). We executed each example, using the input vectors in Powerstone, on instruction set simulators for a MIPS microprocessor as well as an 8051 8-bit microcontroller. Those simulators were augmented to output instruction traces, and we used an additional tool to parse the traces and gather loop statistics.

Complete results of the loop study appear in [23]. The main results showed that the programs running on the MIPS spend 66% of their time in loops with a static size of 256 instructions or less, while the 8051 programs spent 76% of their time in such loops. Furthermore, 77% of time spent in loops (or 51% of total time) on the MIPS was spent in loops whose static size was 32 instructions or less. More importantly, many of the examples contained several small loops that dominated the execution time, and generally iterated many times per execution. For example, *g3fax* contains two loops that represent 62% of total execution time and only consist of 6 assembly instructions each. In addition, one of these loops iterates 1,729 times on each visit. For all of the tested examples, the two most frequent loops accounted for 40% of total execution time for the MIPS and over 70% of execution time on the 8051.

Thus, there does exist the *potential* for good speedups through re-implementing just a few small loops in

Figure 1: E5 single-chip architecture.



hardware – but whether such speedup could be obtained, and whether such speedup would result in actual energy savings, depends on the speed and increase in power consumption from using configurable logic.

For our experiments, we modified the Powerstone benchmarks so that they would loop many times, allowing us to achieve more accurate and stable physical measurements. In addition, for several examples, we moved the data initialization code into the main program loop, since data acquisition would occur during every iteration if the benchmark were implemented in an embedded system. These changes cause slightly different loop statistic results than for the same benchmarks described in [23].

4. Partitioning Method

Our general method of using the configurable logic for energy improvement consisted of moving as much of the software execution as possible onto the logic. Thus, based on the analysis of the loop regions of a given program, we tried to partition the most time-dominating regions onto the logic. Such partitioning was limited by the size of the logic, so we sometimes had to take the second most time-dominating region. In addition, sometimes the best partitioning involved a combination of other regions that together accounted for more of the overall time.

We implemented a given region on the logic by manually writing a synthesizable VHDL model for that region. In doing so, we looked at the C source code to determine the region's high-level behavior, and we then wrote the best performing VHDL model that we could, with knowledge of the size limit of the configurable logic. Sometimes that model looked very similar to the C code, but other times it could be quite different.

Finding the best set of regions for implementation on the configurable logic, and creating the best synthesizable model of those regions, provide two dimensions to the hardware/software partitioning problem that are interdependent. We relied on human design expertise to find the solutions for each example. Automated exploration involving both dimensions (most automated approaches only consider the first dimension) remains an area of future work.

If multiple synthesizable regions were mapped to the configurable logic, we included them as sub-states of a single state machine, so that when synthesized they would share hardware. Such sharing was possible because we were guaranteed that the regions would not execute concurrently to one another, since they came from sequential software.

Our target architecture was based on the Triscend E5 and A7 chips. A general view of the E5 chip's architecture is shown in Figure 1. Communication between the microprocessor and configurable system logic (CSL) takes place via shared memory and several direct signals. The E5 uses an 8051 microprocessor (8-bit) and the A7 uses an ARM microprocessor (32-bit). We also evaluate results assuming a MIPS (32-bit) processor, as will be described. For the MIPS experiments, the architecture is slightly different. We removed the DMA because the execution of the microprocessor and the custom hardware are mutually exclusive. This has the advantage of a smaller system, but increases the complexity of the custom hardware by requiring the memory communication to be implemented in the configurable logic.

We implemented each partitioning by replacing the selected software regions with handshaking behavior. The software would activate the CSL using a start signal, and then wait for the CSL to set a done signal. The microprocessor enters a low-power state while waiting for the CSL, and the CSL enters a low-power state by not executing while waiting for the microprocessor.

5. Power and Performance Evaluation

We used the testbenches that come with the Powerstone benchmarks to generate dynamic power and performance data for the benchmarks.

The E5 chip implements a microprocessor and CSL in a 0.35 micron technology. The A7 chips typically use 0.18 micron techonology, but the version we tested was fabricated using 0.25 micron technology. We took physical measurements, utilizing a digital multimeter, to determine the current and hence the power for our examples. We executed each example first as an all-software implementation with the CSL in a low-power

Table 1: Benchmark information.

MIPS Benchmarks						8051 Benchmarks					
Eg	Size	Loop Size	Loop Time	CSL Size	Potential Speedup	Eg	Size	Loop Size	Loop Time	CSL Size	Potential Speedup
g3fax	4452	24	31%	161	1.45	g3fax	8309	71	56%	158	2.27
adpcm	7640	152	30%	469	1.43	crc	810	58	63%	87	2.69
crc	4288	68	66%	46	2.90	brev	2505	1710	92%	141	12.99
des	6116	360	52%	516	2.08	matmul	838	212	94%	539	17.24
engine	4432	64	28%	133	1.39						
jpeg	5960	116	10%	157	1.11						
summin	4136	100	48%	212	1.92						
v42	6388	60	23%	233	1.30						
	Avg:	118	36%	241	1.70		Avg:	513	76%	231	8.80

mode, and then as a partitioned example, utilizing low-power modes of the 8051 and CSL when the other was active. We enclosed each example in a long-running loop to obtain a stable power reading. Performance was measured by using the serial communication on the chip in order to specify the start and end of the application. A timer running on a workstation measured the differences between the starting and ending times in order to determine the actual execution times.

For our MIPS-based evaluation, we used a simulation-based approach for performance evaluation. We ran each example on a MIPS architectural simulator [8] that outputs the number of cycles each example executes. In order to determine power of the MIPS processor, we use the reported power consumption of a typical MIPS core [19], based on 0.18 micron technology. All examples ran at 100 MHz with a supply voltage of 1.8V. We used Xilinx's Virtex Power Estimator [24] to estimate power of the configurable logic for each example, also utilizing a 0.18 micron FPGA technology (in particular, the XCV50E). We also used Xilinx's Xpower tool, with similar results. These tools require an estimate of the average switching frequency of the FPGA's internal nets. The Virtex Power Estimator User Guide [25] states that the frequency is typically between 6% and 12%. We thus use 9% as our value.

Any configurable system on a chip requires a way to connect the microprocessor to the configurable system logic. The E5 chip uses what Triscend calls a configurable system interface (CSI) bus. This bus (a simplified version can be seen in Figure 1) allows the configurable system logic to access resources without requiring any time from the microprocessor. It also acts as the interface between the configurable system logic and the microprocessor. Fundamentally, the CSI bus consists of an 8-bit data bus and a 32-bit address bus. The power dissipation of the CSI bus can add significantly to the overall power dissipation of the chip. The number of selectors (connecting the address bus to the configurable system logic) increases as the size of the configurable system logic itself increases. Therefore, the power dissipation of the microprocessor subsystem (microprocessor plus bus) will be different depending on the size of the configurable system logic.

Synthesis of the selected loop regions to the CSL for the E5/A7 was done using the Synopsys FPGA Compiler [20], and placement, routing and mapping was done by the Triscend tools. All synthesis, placement, and routing for the MIPS examples was done by Xilinx tools.

Software performance for the MIPS examples was evaluated using the MIPS simulator, and counting the number of required cycles. Hardware performance for a given loop was evaluated by counting the number of cycles required by the synthesized hardware to execute the loop. Because some loops had multiple paths, we conservatively considered the longest path only.

Our results are based on the assumption that the power consumption of the system in power-down mode is insignificant during the idle period that is created from the speedup. Depending on the actual system used, the power may be higher, resulting in slightly lower energy savings.

6. Results

Table 1 contains information regarding the software examples. *Size* corresponds to the static size of the application in terms of instruction bytes. Therefore, for the MIPS, the number of instructions is the size divided by four. For the 8051, the number of instructions ranges between the total size and the size divided by three. This is because the 8051 uses variable length instructions, ranging from 1 byte to 3 bytes. *Loop Size* is the static number of instruction bytes in the most frequent loops. The *Loop Time* is the percentage of total dynamic instructions that are executed in these loops. *CSL Size* is the number of CSL blocks required by the CSL in order to implement the most frequent loops in hardware. For the 8051-based device, each CSL block corresponds to

Table 2: MIPS-based device energy results.

	Performance (cycles)					Power (W)					
Eg	Sw	Loop in sw	Loop in CSL	Sw / CSL	Speed-up	Sw	CSL	Total	Time(S)	Energy (J)	% Energy Savings
g3fax	15,600,000	4,720,000	599,000	11,479,000	1.36	0.070	0.111	0.182	0.11500	2.09E-02	25%
adpcm	113,000	29,300	5,440	89,140	1.27	0.070	0.181	0.187	0.00089	1.64E-04	18%
crc	5,040,000	3,480,000	460,800	2,020,000	2.50	0.070	0.061	0.181	0.02020	3.66E-03	59%
des	142,000	70,700	15,100	86,400	1.64	0.070	0.197	0.204	0.00086	1.76E-04	31%
engine	915,000	145,000	28,100	798,100	1.15	0.070	0.082	0.180	0.00798	1.44E-03	12%
jpeg	7,900,000	646,000	171,000	7,425,000	1.06	0.070	0.092	0.180	0.07430	1.34E-02	6%
summin	2,920,000	1,270,000	266,000	1,916,000	1.52	0.070	0.111	0.187	0.01910	3.58E-03	32%
v42	3,850,000	846,000	216,000	3,220,000	1.20	0.070	0.102	0.182	0.03220	5.86E-03	15%
				Average:	1.46					Average:	**25%**

roughly 12.5 gates. For the MIPS-based device, each CSL block is roughly between 15 to 20 gates. *Potential speedup* is the approximate maximum speedup possible if the loops were executed in zero time. Such speedup is approximate because it is based on the percentage of dynamic instructions instead of the percentage of execution time. Therefore, if all instructions take the same time, the speedup is exact. If not, which is the case for the 8051, the actual speedup may be slightly different.

Notice that the *Size*, *Loop Size*, and *Loop Time* are significantly different for the MIPS and 8051. This is due to the fact that the MIPS is a 32-bit RISC processor and the 8051 is an 8-bit CISC processor. The CSL size differs for the two systems because of different synthesis tools and target combinational logic. The MIPS system uses the Virtex XCV50E from Xilinx which has a total of 1728 logic cells and the 8051 system uses the Triscend E5 chip which has 2048 CSL blocks. Both devices use blocks that consist of 4-bit lookup tables.

6.1. Estimated Results for a MIPS-Based Device

Results for several PowerStone examples running on a MIPS simulator are shown in Table 2. The first several columns describe the performance of the examples. *Sw* is the total number of cycles to execute the example completely in software. *Loop in sw* is the total cycles required by the loop when running in software. *Loop in CSL* is the number of cycles required by the loop when running in custom hardware. *Sw/CSL* is the number of cycles required to execute the entire program after partitioning. *Speedup* is the resulting speedup after partitioning. The next three columns show the power consumption, where *Sw* is the power of the software on the microprocessor, *CSL* is the power of the custom hardware, and *Total* is the power of the entire system. The last three columns represent the execution time, energy, and energy savings of the partitioned system.

The average speedup for the MIPS examples is 1.46. These speedups were achieved by moving very small amounts of the original application into hardware running on the CSL. In fact, the average percentage of total assembly instructions used by the loops was only 2.2%. This corresponds to an average of 30 instructions and required an average of 241 configurable logic blocks. These loops account for an average of 36% of the total execution time, and was as high as 66% for the *crc* example.

For our experiments, we used the same clock frequency of 100 MHz for both the MIPS and the CSL. This is based on the fact that many current platforms, such as the Triscend A7, run 32-bit processors and the configurable logic using the same clock.

The power column in the table presents power data for the microprocessor and for the CSL when they are active. By analyzing the E5 device, we estimated for the MIPS-based system that the interconnect power, namely the power consumed by the system buses and shared memory, would be about 0.1W. Furthermore, we are assuming a low-power state of 25% of the active state on the microprocessor [15], and the CSL's low-power state consisted only of quiescent power, and thus used the following equation to compute total power:

$$\text{Total power} = \%Sw * Sw + \%CSL * (CSL + .25*Sw) + \text{Interconnect Power} + \text{Quiescent Power}$$

where %Sw is the percent of time spent in software, %CSL the percent time spent in the CSL, Sw is the power of the software when the microprocessor is active, and CSL is the power of the CSL when active. Note that the values shown for software power and CSL power in Table 2 do not include the interconnect and quiescent power. Interconnect and quiescent power are only included as part of total power.

The power of the CSL consumed on average 67% more power than the microprocessor. Despite this significant difference, total power increased by only 3% compared to

Table 3: Triscend E5 device results.

Eg	Performance (S) Sw	Sw/CSL	Speedup	Power (W) Sw	Sw/CSL	Energy (J) Sw	Sw/CSL	Savings
g3fax	15.16	7.11	2.13	0.252	0.270	3.820	1.920	49.8%
crc	10.64	4.64	2.29	0.207	0.225	2.202	1.044	52.6%
brev	17.81	1.81	9.84	0.252	0.270	4.488	0.489	89.1%
matmul	32.66	2.06	15.85	0.270	0.288	8.818	0.593	93.3%
		Average:	7.53	0.245	0.263	4.832	1.011	**71.2%**

Table 4: Triscend A7 device results.

Eg	Performance (S) Sw	Sw/CSL	Speedup	Power (W) Sw	Sw/CSL	Energy (J) Sw	Sw/CSL	Savings
g3fax	11.47	7.44	1.54	1.320	1.332	15.140	9.910	34.5%
crc	10.92	4.51	2.42	1.320	1.320	14.414	5.953	58.7%
brev	9.84	3.28	3.00	1.332	1.344	13.107	4.408	66.4%
		Average:	2.32	1.324	1.332	14.221	6.757	**53.2%**

the software-only version. The reason for this small increase is that in the partitioned system, the CSL was active for only 10% of total execution time.

The average energy savings were 25%. If we had used a CSL switching frequency of 12% rather than 9%, the energy savings would have been 21%. The main limit to energy savings is lack of speedup. Some of the tested examples did not spend enough time in a small loop in order to achieve an effective speedup.

We point out that, although energy savings for the MIPS platform were modest, we achieved average overall speedups of 1.46 without sacrificing energy and in most cases saving a small amount of energy. Also, it is possible that implementing the memory communication in the configurable logic has a considerable power overhead compared to using a DMA component.

6.2. Measured Results for Triscend E5

Results for several Powerstone examples on the E5 are shown in Table 3. All columns labeled *Sw* correspond to the software only version and *Sw/CSL* corresponds to the partitioned version. The first several columns show the measured performance results. *Speedup* is the corresponding speedup after partitioning. The next two columns show the power consumption of the software version and the partitioned version. The final three columns represent energy results.

Note that the average speedup is 7.53. This speedup is achieved by moving an average of 513 instruction bytes into the CSL. Excluding the *brev* example, which uses by far the largest number of instructions in the loop, the average instruction bytes moved to hardware is only 114. The reason for this large amount is that the 8051 is an 8-bit processor, and may require a large amount of instructions to implement even a small loop. These instructions account for approximately 76% of the total execution time.

Power consumption only increased by an average of 18 milliwatts compared to the software versions. This resulted in energy savings of 71.2%.

6.3. Measured Results for Triscend A7

We have recently started testing energy savings on the Triscend A7 chip, which combines a 32-bit ARM processor with configurable logic. The results so far are shown in Table 4.

The average speedup for the three examples was 2.32. Power consumption only increased by an average of 8 milliwatts when using the CSL. This resulted in energy savings of 53.2%.

Comparing these three systems shows that the E5 achieves the greatest energy savings. This is due mainly to the large speedups that were achieved on the E5. This was possible because the 8051 on the E5 is a relatively slow microprocessor compared to the MIPS and ARM. Therefore, given the same percentage of execution time for a loop, it is much easier to get a larger speedup using the E5. We do not have loop analysis results for the ARM processor and therefore could not compare the percentage of total execution time of the loops. We thus assume that the speedup of *crc* is higher on the A7 than on the E5 due to the application spending more time in the implemented loops on the ARM processor for this example. All three devices showed similar power increases when using the

Table 5: Voltage scaling results (estimated).

MIPS Voltage Scaling Results						
Eg	PSR	CLK	V	Power	Energy	Savings
g3fax	26.4%	73.58	1.61	1.07E-01	1.67E-02	40.1%
adpcm	21.1%	78.89	1.64	1.22E-01	1.38E-04	31.7%
crc	59.9%	40.08	1.34	4.02E-02	2.03E-03	77.6%
des	39.2%	60.85	1.51	8.74E-02	1.24E-04	51.2%
engine	12.8%	87.24	1.71	1.42E-01	1.30E-03	20.8%
jpeg	6.0%	93.99	1.75	1.60E-01	1.26E-02	10.7%
summin	34.5%	65.51	1.54	8.97E-02	2.62E-03	49.9%
v42	16.4%	83.60	1.68	1.33E-01	5.11E-03	25.8%
					Average:	**38.5%**

Triscend A7 Voltage Scaling Results						
Eg	PSR	CLK	V	Power	Energy	Savings
g3fax	35.1%	25.94	2.03	5.70E-01	6.53	56.8%
crc	58.7%	16.52	1.70	2.52E-01	2.75	80.9%
brev	66.7%	13.33	1.58	1.79E-01	1.76	86.6%
					Average:	**74.8%**
Triscend E5 Voltage Scaling Results						
g3fax	53.1%	11.72	2.19	5.58E-02	8.5E-01	77.9%
crc	56.4%	10.90	2.12	4.05E-02	4.3E-01	80.4%
brev	89.8%	2.54	1.30	4.26E-03	7.6E-02	98.3%
matmul	93.7%	1.58	1.17	2.28E-03	7.5E-02	99.15%
					Average:	**88.9%**

CSL. We plan to implement the remaining Powerstone examples on the E5 and A7 devices in the future – implementing them on the MIPS simulator was simpler and thus we completed those results first.

6.4. Potential Results with Voltage Scaling and Low-Power Configurable Logic

Voltage-scalable processors are increasing in popularity in low-power embedded systems. Since much of the power consumed in CMOS-based systems is proportional to the voltage-squared, reducing voltage to the minimum possible while meeting timing constraints can yield excellent power savings. In this section, we update our results based on hypothetical microprocessor/CSL devices that are voltage scalable.

The large speedups achieved by implementing loops in the CSL creates the potential for reducing the voltage and clock frequency of the system to a point where performance is equal to that of the software version, but yet consumes much less power due to the quadratic reduction in power. We know of no current devices that implement voltage scaling for CSL. However, we can estimate the energy consumption of such a system for the previous examples, using the following formulas [11]:

$$T \alpha V / (V - V_t)^2$$
$$T = k * V / (V - V_t)^2$$

where T is the delay of the critical path, V is the supply voltage, V_t is the threshold voltage, and k is a design-dependant constant.

Using the previous formulas, we are able to derive an equation for determining the clock frequency at a given supply voltage:

$$F = 1/T$$
$$F = (V - V_t)^2 / (k * V),$$

where F is the clock frequency.

We first estimate the delay of the critical path by using the maximum clock frequency. Using this delay, we can determine k. We use a threshold voltage of 0.8 V. The normal supply voltages for the systems are 1.8 V for the MIPS, 2.5 V for the A7, and 3.3 V for the E5. We next determine how much we can reduce the clock in order to match the performance of the software-only design. With this information, we can determine the minimum supply voltage that allows the design to run at the reduced clock speed. Once we have determined this voltage, we can estimate power for the voltage-scaled system in the following way:

$$C = P_o / (0.5 * V_o^2 * a * F_o)$$
$$P = 0.5 * V^2 * C * a * F$$
$$P = 0.5 * V^2 * (P_o / (0.5 * V_o^2 * a * F_o)) * a * F$$
$$P = (V^2 / V_o^2) * (F / F_o) * P_o,$$

where P_o, V_o, and F_o are the power, voltage, and clock frequency of the system before voltage scaling, C is the capacitance of the system and a is the switching fequency. P, V, and F are the power, voltage, and clock frequency after voltage scaling. Therefore, we are first estimating the capacitance of the system in terms of the power, voltage, switching activity, and clock frequency. Once we have determined C, we can estimate the power of the voltage scaled system using the power of the original system and the voltages and clock frequencies of both systems.

Results are shown in Table 5. *PSR* is the potential speed reduction of the system while still achieving the same performance as software. *CLK* is the updated clock frequency used to achieve this speed reduction. The original clock speeds were 100 MHz for the MIPS, 40 MHz for the A7, and 25 MHz for the E5. *V* is the lowest voltage possible for the given clock speed, assuming the appropriate technology for each device. Power, energy, and savings are the results of the entire system based on the new voltage and clock. Power is measured in Watts and energy is in Joules. Note that the Triscend A7 and E5 do not support voltage scaling, but we are estimating the results in order to show the potential benefits of such scaling. There are currently voltage scalable ARM-based chips available [15][16] but none of them currently contain configurable logic. Although the A7 doesn't support voltage scaling, it does support clock scaling. We tried using the speedups to slow the clock down, but found almost no energy savings, compared to the normal partitioning, due to a linear increase in execution time and linear decrease in power consumption. In some cases, clock scaling may lead to worse energy consumption due to a slightly nonlinear decrease in power. This occurs because some of the power consumption on this chip is unrelated to the clock, i.e. quiescent power.

Notice that the savings shown in Table 5 for the MIPS based system increased from 25% to 39% when using voltage scaling. For the A7, energy savings improved from an average of 53% to 75%. The E5 improved from 71% to 89%.

Furthermore, as configurable logic continues to find its way into final products rather than just prototypes, low-power configurable logic will likely begin to appear (most low-power FPGAs today are actually just low-power when idle, and still consume much more power than microprocessors). A low-power FPGA by George [7] showed power reductions of an order of magnitude over commercial low-power FPGAs.

7. Conclusions

We examined the potential benefits of using on-chip configurable logic to reduce software energy by moving critical loops to the configurable logic in order to decrease execution time. We based our target architecture and our time and power evaluation on commercial commodity chips with configurable logic available at low cost. We conservatively estimated savings of 25% for a 32-bit MIPS microprocessor based device, while we found through physical measurements energy savings of 71% for a commercial 8-bit microprocessor based device, and 53% for a commercial 32-bit microprocessor based device. We also showed that if voltage scalability were added to such devices, the energy savings could be increased substantially. Furthermore, we showed that the configurable logic currently consumes much more power than the microprocessor – pointing to the need for low-power configurable logic.

Energy savings could be further improved by parallelizing the execution of the software and configurable logic when possible, by considering moving entire subroutines rather than just loops to the configurable logic, and by parallelizing the hardware through loop unrolling and other means – these remain areas for future investigation. Furthermore, as on-chip configurable logic size increases, more software can be mapped to that logic, for even greater savings.

8. Acknowledgements

This work was supported in part by the National Science Foundation (CCR-9876006) and UC MICRO.

9. References

[1] Altera Corporation, ARM-Based Embedded Processor PLDs, August 2001.

[2] P. Athanas, H. Silverman. Processor reconfiguration through instruction-set metamorphosis. Computer, Volume: 26 Issue: 3, March 1993 Page(s):11-18.

[3] Atmel FPSLIC, http://www.atmel.com/atmel/products/prod39.htm.

[4] E5 Press Release, http://www.triscend.com/about/indexrelease051401.html.

[5] P. Eles, Z. Peng, K. Kuchchinski and A. Doboli. System Level Hardware/Softeare Partitioning Based on Simulated Annealing and Tabu Search. Kluwer's Design Automation for Embedded Systems, vol2, no 1, pp. 5-32, Jan 1997.

[6] D. Gajski, F. Vahid, S. Narayan and J. Gong. Specification and Design of Embedded Systems. Prentice Hall, 1994.

[7] V. George, H. Zhang, and J. Rabaey. The Design of a Low Energy FPGA Varghese George, Hui Zhang and Jan Rabaey, ISLPED 1999, pp. 188-193.

[8] T. Givargis, F. Vahid, and J. Henkel. System-Level Exploration for Pareto-Optimal Configurations in Parameterized Systems-on-a-Chip. International Conference on Computer-Aided Design (ICCAD), San Jose, November 2001.

[9] M. Gokhale, J. Stone. NAPA C: Compiling for hybrid RISC/FPGA architectures. IEEE Symposium on FPGAs for Custom Computing Machines, FCCM '98.

[10] S.C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R.R. Taylor, R. Laufer. PipeRench: A Coprocessor for Streaming Multimedia Acceleration. International Symposium on Computer Architecture, pp. 38-49, 1999.

[11] R. Gonzalez, B. Gordon, and M. Horowitz. Supply and Threshold Voltage Scaling for Low Power CMOS. IEEE Journal of Solid-State Circuits, Vol. 32, No. 8, August 1997.

[12] J. Hauser, J. Wawrzynek. Garp: a MIPS processor with a reconfigurable coprocessor. IEEE Symposium on FPGAs for Custom Computing Machines, pages 12-21, Napa Valley, CA, April 1997.

[13] J. Henkel, Y. Li. Energy-conscious HW/SW-partitioning of embedded systems: A Case Study on an MPEG-2 Encoder. Proceedings of Sixth International Workshop on Hardware/Software Codesign, March 1998, pp. 23-27.

[14] J. Henkel. A low power hardware/software partitioning approach for core-based embedded systems. Proceedings of

the 36th ACM/IEEE conference on Design automation conference, pp. 122 – 127,1999.

[15] Intel StrongArm 1110 Processor, http://developer.intel.com/design/strong.

[16] Intel XScale Processor, http://developer.intel.com/design/intelxscale.

[17] A. Kalavade and E.A. Lee. The Extended Partitioning Problem: Hardware/Software Mapping, Scheduling and Implementation-Bin Selection. Kluwer Design Automation for Embedded Systems, vol 2, no 2, pp. 125-163, Mar 1997.

[18] A. Malik, B. Moyer, D. Cermak. A Low Power Unified Cache Architecture Providing Power and Performance Flexibility. International Symposium on Low Power Electronics and Design. June 2000.

[19] MIPS Technologies, Inc., http://www.mips.com.

[20] Synopsys, http://www.synopsys.com.

[21] Triscend Corporation, http:/www.triscend.com.

[22] G. Vanmeerbeeck, P. Schaumont, S. Vernalde, M. Engels and I. Bolsens. Hardware/Software Partitioning of Embedded System in OCAPI-xl. International Symposium on Hardware/Software Codesign, pp. 30-35, 2001.

[23] J. Villarreal, R. Lysecky, S. Cotterell, K. Miller and F. Vahid. Loop Analysis of Embedded Applications. UC Riverside Technical Report UCR-CSE-01-03, 2001.

[24] Virtex Power Estimator, http://support.xilinx.com/cgi-bin/powerweb.pl.

[25] Virtex Power Estimator User Guide, Xilinx Inc., 2000.

[26] M. Wan, Y. Ichikawa, D. Lidsky, J. Rabaey. An energy conscious methodology for early design exploration of heterogeneous DSPs. Proceedings of the IEEE 1998 Custom Integrated Circuits Conference, p.111-117, Santa Clara, May 1998.

[27] M. Wirthlin, B. Hutchings. A dynamic instruction set computer. Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, 1995, pg. 99-107.

[28] Xilinx Corporation, Virtex-II Pro Platform FGPA Handbook, January 31, 2002.