Power Consumption in a Real, Commercial Multimedia Core

Dominic Aldo Antonelli
Alan J. Smith
Jan-Willem van de Waerdt

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-24
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-24.html

March 21, 2008

Power Consumption in a Real, Commercial Multimedia Core[*]

Dominic Antonelli
University of California, Berkeley
dantonel<at>eecs.berkeley.edu

Alan Jay Smith
University of California, Berkeley
smith<at>eecs.berkeley.edu

Jan-Willem van de Waerdt
NXP Semiconductors
jan-willem.van_de_waerdt<at>nxp.com

March 17, 2008

Abstract

Peak power and total energy consumption are key factors in the design of embedded microprocessors. Many techniques have been shown to provide great reductions in peak power and/or energy consumption. Unfortunately, several unrealistic assumptions are often made in research studies, especially in regards to multimedia processors. This paper focusses on power reduction is real commercial processors, and how that differs from more abstract research studies.

First, such processors often already utilize several power reduction techniques, and these existing optimizations can have a huge impact on the effectiveness of further optimizations. Second, highly optimized production code tends to have significantly less schedule slack and significantly higher density of memory accesses than unoptimized code. Finally, many such studies are done using high-level simulators, which may not accurately model the power consumption of real microprocessors. In addition, in this study we focus on an embedded, synthesized processor, rather than a high performance custom and hand designed stand-alone microprocessor; a 400MHz synthesized core (the TriMedia TM3270) has significantly different characteristics than a 3GHz Pentium.

We carefully analyze the power consumption of the TriMedia TM3270, a commercial product, on both reference benchmark code and optimized code. We use commercial synthesis and simulation tools to obtain a detailed breakdown of where power is consumed. We find that increased functional unit utilization causes significant differences in power consumption between unoptimized and carefully hand-optimized code. We also apply some simple techniques for power savings with no performance degradation, and find that such techniques can greatly change the power profile of a microprocessor. We find that clock gating of individual functional units is vital to keeping the dynamic power low. Finally, we find that synthesizing for the fastest target frequency possible at a given voltage yields the most energy-efficient design.

1 Introduction

Peak power and total energy consumption are key factors in the design of embedded microprocessors. High peak power increases packaging costs, and high energy consumption reduces battery life. A wide variety of techniques targeting many different aspects of processor design have been proposed to keep these factors under control. Embedded system designers face a tough task in choosing which

[*]Funding for this research has been provided by NXP Semiconductors USA Inc., the State of California under the MICRO program, and by Toshiba Corporation.

techniques to apply and must rely heavily on prior and published evaluations of the effectiveness of these techniques. Unfortunately, this type of evaluation often makes assumptions about the software and hardware that may not hold in a production environment, as opposed to a research study.

For evaluations based on multimedia workloads, many researchers use benchmark code from MediaBench [1] or other multimedia benchmark suites. However, this software is rarely optimized, especially for the target processors, resulting in different behavior than with production software. Also, techniques are often compared against baselines that are not realistic. Researchers often ignore architectural features such as guarded (predicated) execution that can greatly reduce the number of branches. Many cache techniques are compared against a baseline of an N-way set associative cache when all N tag- and data- ways are accessed in parallel. Because of the high power overhead, this baseline configuration is rarely used in modern embedded processors such as the TriMedia TM3270 processor from NXP Semiconductors. This can result in inflated power savings measurements. As we shall see, often a simple optimization can eliminate a large percentage of the power consumption in a particular portion of a processor, which makes further optimizations to the same portion insignificant.

In order to show that these issues are significant, we start with a real design - the TM3270 - and use commercial synthesis and simulation tools to obtain a detailed breakdown of where power is being consumed. This yields accurate power consumption data, at the cost of large amounts of simulation time. We run both reference (unoptimized) and optimized versions of several applications, and compare the results. From this data, we find that there is a significant difference in the power profiles of reference and optimized code, mainly due to increased functional unit utilization and differing instruction mixes. We also synthesize the design at several different target frequencies and conclude that synthesis at lower frequencies results in suboptimal power-performance trade-offs. We also find that clock gating of individual functional units is vital to keeping the dynamic power low because each individual functional unit spends most of its time idle, even if all the issue slots of the processor are kept busy. Finally, we propose and evaluate several simple power optimizations that we found during this analysis such as skipping tag accesses on sequential cache accesses that are known to hit and using low-power instead of high-speed SRAMs.

The rest of this paper is organized as follows. In Section 2, we review related work. In Section 3, we provide an overview of our experiments. In Section 4, we evaluate the differences in power consumption between reference and optimized software. In Section 5, we take an in-depth look at where the power is going, then in Section 6, we evaluate techniques already used to reduce power consumption in the TM3270. In Section 7, we describe and evaluate several simple techniques that could be used to further reduce the power consumption. Finally, in Section 8, we present our conclusions.

2 Related Work

Power and energy consumption and dissipation are key factors in microprocessor design[2]. In [3], O. Unsal and I. Koren give an overview of many techniques that have been proposed. D. Albonesi et al. have proposed the use of reconfigurable caches for power and performance improvement [4]. Other work on caches include micro-architectural techniques such as filter caches[5, 6], way-prediction and selective direct-mapping[7]. Techniques targeting the register file include resizing the register file[8] and reducing the number of register file ports[9]. Also, in [10], Kim et al. propose a reconfigurable multiplier for power savings.

There is also a lot of research centered on fast, accurate architecture-level power estimation. The most well known model is probably Wattch[11]. We chose not to use such a model because we already had a complete HDL of the TM3270, the TM3270 is significantly different from the standard

SimpleScalar 5-stage out-of-order model, and we wanted to compare power consumption when using different synthesis parameters. Further, our tools have been used and validated in commercial development.

The work that most closely resembles our own is in [12],[13], and [14]. All three of these include a power analysis of a real microprocessor on a particular application or set of applications. Our work differs in that we analyze an embedded multimedia processor on several multimedia applications, we look at the differences between optimized and non-optimized software, we evaluate the effectiveness of existing power saving techniques, we obtain a much finer-grained breakdown of power consumption, and we apply and evaluate several additional power saving techniques.

There have been several papers e.g. [15, 16] on the power effects of using performance-oriented SIMD operations. Slingerland and Smith also performed a detailed performance evaluation of SIMD optimizations (MMX, SSE, etc) on a wide variety of general purpose processors, yielding an average speedup of 3-5x[17]. Our comparison of optimized and non-optimized software is similar, though we do not limit optimizations to replacing SISD operations with SIMD operations.

3 Overview

For this study, we use the TriMedia TM3270 multimedia processor. The TriMedia series of products by NXP Semiconductors is available commercially, appears in Philips consumer products, and has undergone several generations of refinement and optimization. By working closely with NXP, we have been able to work with a highly optimized product, typical of a class of commercial products, and have been able to focus on further improvements. We have had access to all of the tools available at NXP, and to the design of the TriMedia processor itself. We can simulate execution of optimized and non-optimized versions of several multimedia applications at several different operating frequencies and obtain reports of how much power each portion of the processor consumes.

3.1 Design

The TriMedia TM3270 multimedia processor is a 5 issue-slot VLIW processor; on each non-stall cycle, it issues a single VLIW which contains up to 5 separate operations. Each operation is a single RISC-like instruction that is independent of the other operations issued in the same VLIW. This processor has custom operations for packed 8- and 16- bit arithmetic, clipping arithmetic, and other tasks common in multimedia processing. It also supports guarded execution - conditional execution of each operation based on the contents of an extra source register - in order to reduce the number of branches. The TM3270 does not do any branch prediction, but has 5 branch delay slots. It has 128 32-bit registers, a 64KB 8-way set-associative instruction cache, and a 128KB 4-way set-associative data cache. In the instruction cache, the eight tags are first accessed in parallel, and after comparison, only the correct way from the instruction data SRAM is accessed. In the data cache, all four data and tag ways are accessed in parallel. Both caches have a block size of 128 bytes. See [18] for a more detailed description of the TM3270.

The TM3270 is already highly optimized for power consumption. It is heavily clock-gated at a very fine grain. For example, each stage of each functional unit is individually clock-gated. This is vital because each issue slot has several functional units, but at most one per issue slot is active in any given cycle. It is also implemented in a low-leakage process technology, which reduces the power consumption of clock-gated components to almost zero.

3.2 Synthesis

When using automated tools to synthesize RTL descriptions of processors, there are many choices in how to pick each standard cell. For example, Table 1 shows different choices for an inverter. As the drive strength increases, the speed of the cell increases - the 4x drive cell can drive 4 times as much capacitance as the 1x cell, at the same delay, though it is not 4 times as fast at the same capacitance. If we use faster cells, we may be able to reduce the supply voltage while still meeting the timing requirements, thus reducing the power consumption. On the other hand, the area and capacitance also increase. The increased capacitance increases the dynamic power consumption of the processor. One goal of this work is to study this tradeoff.

Drive Strength:	min	0.5x	1x	2x	3x	4x	5x	6x
Area (μm^2)	2.20	3.29	3.29	4.39	6.59	7.68	8.78	10.98
Input capacitance (fF)	1.65	2.43	4.29	8.67	12.9	17.2	21.7	25.8
Maximum output load (fF)	78.0	165	327	655	990	1310	1640	1980
Output delay at 0 load (ps)	27.6	22.8	20.9	19.6	19.8	19.2	19.3	19.1
Output delay at 137fF load	566	290	166	103	80.2	68.1	61.0	53.6
Load (fF) where output delay is 290ps	64.7	137	274	549	820	1098	1367	1641

Table 1: Different choices the synthesis tools have for implementing an inverter.

We use Synopsys Design Compiler to synthesize this processor at different target frequencies from 50MHz to 400MHz, in increments of 50MHz. In the synthesis, we use the same constraints (such as max capacitance, clock uncertainty, etc.) as is used in commercial design of TM3270-based chips. The TM3270 is synthesized in the Phillips 90nm LP (low power) process technology at 1.2V, using the standard design rule set. The mask set is compatible with the TSMC 90nm LP process (CLN90LP). Because of the number of different netlists we need to simulate, we do not do place-and- route. Table 2 shows the area taken up by each of the major components of the design for the different synthesis frequencies. Note that the area does not increase much because only a small portion of the design is on the critical path and the SRAMs do not scale with frequency. Also note that the SRAMs in the instruction fetch unit (row 2) and the load/store unit (row 4) take up from 68% (on the fastest design) to 70% (on the slowest design) of the total design area.

Synthesis Frequency (MHz)	50	100	150	200	250	300	350	400
Instruction Fetch Logic	0.11	0.11	0.11	0.12	0.12	0.12	0.12	0.13
Instruction Fetch SRAMs	1.12	1.12	1.12	1.12	1.12	1.12	1.12	1.12
Load/Store Unit Logic	0.42	0.42	0.42	0.42	0.43	0.43	0.44	0.45
Load/Store Unit SRAMs	2.67	2.67	2.67	2.67	2.67	2.67	2.67	2.67
Functional Units	0.51	0.51	0.53	0.54	0.56	0.59	0.60	0.63
Register File	0.37	0.37	0.37	0.37	0.37	0.37	0.37	0.37
Decode Logic	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Other	0.21	0.22	0.21	0.22	0.21	0.22	0.22	0.22
Total	5.43	5.44	5.45	5.48	5.50	5.54	5.56	5.61

Table 2: Area (in mm^2) taken up by the various components for each of the different synthesis frequencies we use.

3.3 Applications

The particular application being run, and whether it is optimized or not, will have a significant impact on the power consumption. Because of this, we run both non-optimized and optimized versions of three different multimedia applications: a simple ADPCM decoder, an MP3 decoder, and an MPEG-2 encoder. Our workload is restricted to applications that are simple enough to optimize in a short period of time or that have already been optimized. The ADPCM decoder is simple enough that we were able to hand optimize it. For the MP3 decoder and MPEG-2 encoder, we were able to obtain code that was already optimized for the TM3270. The optimized ADPCM decoder and MP3 decoder produce the same output as their reference counterparts, and the optimized MPEG-2 encoder produces results that are nearly identical (mainly due to differences in rounding, and the 16-bit fixed-point DCT/IDCT). See Appendix C for more details about our application mix.

Although these applications are not the bleeding-edge in multimedia encoding and decoder, they are significantly more than toy kernels. They should be fairly representative, and we expect that using more up-to-date applications would amplify, but not significantly change the nature of, the effects of optimizations that we see.

3.4 Simulation

We use Cadence NC-Verilog to simulate the execution of the various applications on the synthesized netlists. Each simulation produces switching activity information for all the nets, ports, and pins in the design. This switching activity includes information about the various clock-enable signals used in clock-gating throughout the design, so we can extract how often each clock-gated portion of the design was on. We also use the switching activity information as input to Synopsys Power Compiler which reports how much switching, internal, and leakage power each portion of the design consumed. We also gather statistics on the utilization of each type of functional unit.

For all of our experiments, we use an operating voltage of 1.2V and a DDR SDRAM frequency of 200MHz. We vary both the CPU synthesis frequency, and the CPU operating frequency from 50MHz to 400MHz, in increments of 50MHz. Due to the large simulation time, we did not also vary the DRAM frequency. In each application, we insert triggers at the start and end of the code that we wish to measure. The simulation environment then gathers the switching information over only the desired interval.

4 Effects of Optimization

Table 3 shows the relative increase in power and energy caused by optimization, and Table 4 shows the execution times and speedup. Although there is an increase in power in all cases, the largest increase is in the load/store unit (LSU) while the smallest is in the instruction fetch unit (IFU) and decode logic. The large increase in LSU power makes sense since the optimizations used generally reduced the amount of computation needed per load and store, thus increasing the density of loads and stores. As we can see from Table 5, the total number of load and store operations per VLIW more than doubled in all three applications when optimized.

	Power			Energy		
Application:	ADPCM	MPEG-2	MP3	ADPCM	MPEG-2	MP3
Instruction Fetch	1.11	1.31	1.29	0.363	0.080	0.038
Load/Store Unit	2.22	1.85	3.01	0.730	0.113	0.089
Functional Units	1.48	1.66	1.69	0.486	0.102	0.050
Register File	1.55	1.71	1.76	0.510	0.105	0.052
Decode Logic	1.32	1.28	1.10	0.435	0.079	0.033
Overall	1.47	1.51	1.71	0.484	0.093	0.051

Table 3: This table shows the relative power consumption and energy consumption of the optimized version of each application compared to the non-optimized version when running at 400MHz on the design synthesized for 400MHz.

Operating Frequency (MHz):		50	100	150	200	250	300	350	400
ADPCM	Ref.	35405	17703	11803	8852	7082	5902	5060	4428
	Opt.	11623	5812	3875	2907	2326	1939	1663	1455
	Speedup	3.05x	3.05x	3.05x	3.05x	3.04x	3.04x	3.04x	3.04x
MPEG-2	Ref.	61085	30548	20369	15283	12227	10198	8748	7661
	Opt.	3557	1783	1192	901	724	613	532	471
	Speedup	17.2x	17.1x	17.1x	17.0x	16.9x	16.6x	16.4x	16.3x
MP3	Ref.	54524	27267	18181	13641	10915	9103	7808	6837
	Opt.	1489	748	501	380	306	260	227	203
	Speedup	36.6x	36.5x	36.3x	35.9x	35.7x	35.0x	34.4x	33.7x

Table 4: This table shows the execution time (in μs) of each of the applications when operating at each frequency.

Application:	ADPCM		MPEG-2		MP3	
Version:	Ref.	Opt.	Ref.	Opt.	Ref.	Opt.
ALU (5)	1.94	3.11	0.84	1.66	0.88	1.04
DSPALU (3)	0.11	0.35	0.01	0.53	0.00	0.02
DSPMUL (2)	0.00	0.00	0.06	0.43	0.04	0.00
Branch (2)	0.06	0.04	0.06	0.06	0.06	0.05
Load (1)	0.19	0.43	0.14	0.21	0.12	0.22
Super Load (1)	0.00	0.00	0.02	0.04	0.05	0.49
Store (2)	0.06	0.17	0.04	0.20	0.04	0.18
FPALU (2)	0.00	0.00	0.07	0.00	0.07	0.67
FPMUL (2)	0.00	0.00	0.05	0.00	0.08	0.60
FPDIV (1)	0.000	0.000	0.006	0.001	0.002	0.000
Total	2.36	4.11	1.33	3.13	1.39	3.28
NOP	2.64	0.89	3.67	1.87	3.61	1.72

Table 5: Number of each type of operation per VLIW. The values in parenthesis show the number of available functional units of each type. Note that super load operations take two issue slots, so we count each one as two operations.

At first glance, the small increase in IFU power is baffling. All else being equal, we would expect the instruction buffer to end up full more and more as the operation density decreases. However, since the optimized code does a lot more loop unrolling, there are far fewer branches. Branches tend to cause power to be wasted in the instruction cache because enough instruction data must be fetched for 5 maximum size VLIWs in the branch delay slots, but those VLIWs may be (and often are) considerably smaller than the maximum of 28 bytes per VLIW. For highly optimized code, it is much less worthwhile to try to reduce the power wasted by this discarded instruction data.

The increase in power in the functional units is clearly due to the increase in ALU, DSPALU, and DSPMUL (and FPALU and FPMUL in the MP3 application) operations per cycle. The MPEG-2 and MP3 applications have a slightly bigger increase in such operations and show a larger increase in power in the functional units. In the register file, we see a very similar effect - more operations per VLIW leads to an increase in power, and the increase is slightly larger in the MPEG-2 and MP3 applications due to the larger increase in operations per VLIW.

The power increase in the decode logic is smaller than that in the register file, even though they should both be affected similarly by the increase in operations per VLIW. The reason the increase is smaller in the decode logic is that a significant piece of it has to be on for every VLIW, no matter how many operations there are.

Other important effects of optimization can inferred from Table 5. First, the larger number of operations per VLIW means that delaying operations to reduce peak power will have more impact on performance. Unfortunately, this means that techniques such as those in [19] will be less effective on optimized code. Second, even if each VLIW is fairly full, individual functional units are still unused a large fraction of the time. Because of this, clock gating of individual functional units is vital to keeping the dynamic power low.

Note that although the power increases by about 50% – 70% in each application, the total energy consumed decreases because the speedup is larger than the power increase. From Table 3, we can see that the total energy required over the entire course of each application is reduced (by ~95% in the MP3 application). Also note that some or all of the speedup gained by optimization can be traded for greatly decreased power consumption by reducing the CPU frequency and voltage.

For example, assume that our newly optimized MP3 decoder is fast enough to meet real time requirements when run at 200MHz, but we have a TM3270 that will run at 350MHz. Then, we can reduce the operating frequency and voltage. We tried this on a real TM3270 processor and found that we could reduce the voltage from 1.226V to 1.028V, which yields a 55% reduction in power consumption and a 24% reduction in energy per cycle for the optimized MP3 decoder. Comparing this with running the non-optimized MP3 decoder at 350MHz, we found that this yielded a 12% reduction in power consumption. As we can see from Table 4, the optimized MP3 decoder running at 200MHz is 20x faster than the non-optimized one MP3 decoder running at 350MHz. Thus, we obtain both a performance increase and a power reduction, which also yields a large decrease in energy consumption (by a factor of 23.3). Also, any further performance improvements that do not greatly increase the power consumption can likewise be traded for power and energy reduction.

5 Power Breakdown

Although the power changes significantly as we scale the operating and synthesis frequencies, the relative power spent in each component does not change much. Therefore, we show only the power breakdowns for the 400MHz simulation on the 400MHz design. Table 6 shows the power consumption in various portions of the design. As we can see, the instruction fetch, load/store, and functional units consume the vast majority of the power in the chip. However, the breakdown among these three units varies considerably with application.

Application:	ADPCM		MPEG-2		MP3		Average	
Version:	Ref.	Opt.	Ref.	Opt.	Ref.	Opt.	Ref.	Opt.
Instruction Fetch	95.71 (38.8)	105.8 (29.1)	77.38 (41.8)	101.0 (36.1)	80.69 (43.4)	104.2 (32.8)	84.59	103.67
Load/Store Unit	50.92 (20.6)	113.2 (31.1)	36.68 (19.8)	67.68 (24.2)	33.19 (17.9)	99.93 (31.4)	40.26	93.60
Functional Units	57.93 (23.5)	85.69 (23.6)	41.46 (22.4)	68.73 (24.5)	41.16 (22.2)	69.73 (21.9)	46.85	74.72
Register File	25.00 (10.1)	38.80 (10.7)	15.39 (8.31)	26.32 (9.39)	16.05 (8.64)	28.28 (8.90)	18.81	31.13
Decode Logic	8.61 (3.49)	11.40 (3.14)	6.00 (3.24)	7.67 (2.74)	6.38 (3.43)	7.03 (2.21)	7.00	8.70
Other	8.41 (3.41)	8.61 (2.37)	8.31 (4.49)	8.70 (3.11)	8.32 (4.48)	8.81 (2.77)	8.35	8.71
Total	246.59	363.48	185.22	280.13	185.79	317.95	205.87	320.52

Table 6: This table shows the power dissipation (in mW) in each portion of the processor when synthesized for and run at 400MHz. The values in parenthesis show the percentage of total processor power.

In particular, optimization of the code seems to increase the power consumed in the LSU much more than in the IFU. Loop unrolling, if-conversion (converting if statements into branch-free code using predicated execution), and SIMD operations all combine to reduce the average number of VLIWs between each load and store operation; however, the increased size of VLIWs due to the reduction in NOPs does not increase the IFU power much. For example, in the ADPCM application, the reference code has 12.3 bytes per VLIW while the optimized code has 18.6 bytes per VLIW, but the power in the instruction cache SRAMs only goes up by 7% (see Table 7). However, the total number of loads and stores per cycle went from 0.25 to 0.60 (up 140%), and the data cache SRAMs power went up 130%.

Because the IFU and LSU consume the most power, we look at them first. These units are mostly SRAMs, so we look primarily at the power consumption in the SRAMs themselves. Then, we take a look at the power consumption in the functional units which also consume a significant portion of the total power.

5.1 SRAM Power

Table 7 gives us some insight into SRAM power consumption. From this data, we can see that optimization causes a much bigger increase in the data cache power than in the instruction cache power. Also, the instruction cache uses more power overall, probably because it is accessed (almost) every cycle and 32-bytes of instruction data are read on each access, while the data cache is only accessed on a load or store, or when a prefetch/refill/copy-back occurs and (usually) only one (on a store) to four (on a load, 1 for each cache way) 4-byte words are accessed. In the optimized codes, the difference is much smaller due to the greatly increased density of load and store operations.

Application:	ADPCM		MPEG-2		MP3		Average	
Version:	Ref.	Opt.	Ref.	Opt.	Ref.	Opt.	Ref.	Opt.
IFU Tag	21.08 (8.55)	21.58 (5.94)	16.61 (8.97)	21.74 (7.76)	17.66 (9.50)	22.20 (6.98)	18.45	21.84
IFU Data	36.19 (14.7)	39.47 (10.9)	29.86 (16.1)	42.64 (15.2)	31.25 (16.8)	46.14 (14.5)	32.43	42.75
IFU LRU	3.70 (1.50)	3.87 (1.06)	3.09 (1.67)	3.35 (1.20)	3.24 (1.74)	3.11 (0.98)	3.34	3.44
LSU Tag	9.79 (3.97)	17.74 (4.88)	3.88 (2.09)	8.94 (3.19)	3.88 (2.09)	13.55 (4.26)	5.85	13.41
LSU Data	13.98 (5.67)	37.71 (10.4)	12.19 (6.58)	23.17 (8.27)	10.69 (5.75)	37.68 (11.9)	12.29	32.85
LSU Valid	4.44 (1.80)	12.03 (3.31)	3.75 (2.03)	7.61 (2.72)	3.04 (1.64)	13.04 (4.10)	3.74	10.89
SRAM Total	89.18 (36.2)	132.40 (36.4)	69.38 (37.2)	107.45 (38.4)	69.76 (37.5)	135.72 (42.7)	76.11	125.19

Table 7: This table shows the power consumption (in mW) in the various SRAMs, and the instruction buffer, of the baseline (400MHz) design running at 400MHz. The values in parenthesis show the percentage of total processor power. In the instruction cache, the LRU information is stored in a small SRAM, and in the data cache, the valid information is stored in a separate small SRAM.

Another interesting thing to note is that tag and data power are not always perfectly correlated, especially in the data cache. For example, the optimized ADPCM and MP3 applications consume the same amount of power in the data cache data SRAM, but their data cache tag SRAM power consumptions are very different. One major reason for this is the much larger number of super loads in the MP3 application (see Table 5). These operations access only one tag SRAM but access two consecutive data words, which requires accessing two or three data SRAMs, depending on if the address is word-aligned. This points out an important opportunity for power reduction: if possible, avoid accessing the same tag repeatedly for sequential accesses. We evaluate a method to eliminate useless tag reads in Section 7.2.1.

In the instruction cache, the correlation is tighter - the tag SRAM power varies between 0.48x and 0.58x of the instruction SRAM power. The tag SRAM power is lowest relative to the instruction SRAM power in the optimized MPEG-2 and MP3 applications, which are the two with the most instruction cache misses. This makes sense because the refilling of the instruction cache on misses loads 128 bytes of data into the instruction SRAM while only modifying one tag.

5.2 Functional Units Power

Table 8 shows the breakdown of power consumption in each of the types of functional units. In general, we expect this power to be correlated with the operation mix (see Table 5). This is the case for the ALU/DSPALU type operations, as well as the branches and floating point multiplies. Oddly, however, the power consumption in the DSPMUL unit and FPALU is quite significant even where there are no such operations. We discuss this in more detail in Section 7.3

Application:	ADPCM		MPEG-2		MP3		Average	
Version:	Ref.	Opt.	Ref.	Opt.	Ref.	Opt.	Ref.	Opt.
ALU/DSPALU	18.4 (7.46)	29.4 (8.09)	9.22 (4.95)	19.4 (6.93)	9.41 (5.06)	15.6 (4.91)	12.34	21.47
DSPMUL	6.18 (2.51)	10.2 (2.81)	6.64 (3.56)	15.9 (5.68)	6.34 (3.41)	9.13 (2.87)	6.39	11.74
FPDIV	0.00 (0.00)	0.00 (0.00)	1.08 (0.58)	0.12 (0.04)	0.37 (0.20)	0.28 (0.09)	0.48	0.13
FPMUL	0.18 (0.07)	0.18 (0.05)	1.00 (0.54)	0.20 (0.07)	1.28 (0.69)	7.44 (2.34)	0.82	2.61
FPALU	2.53 (1.03)	3.34 (0.92)	1.98 (1.06)	2.29 (0.82)	1.96 (1.05)	7.04 (2.21)	2.16	4.22
Branch	1.00 (0.41)	0.47 (0.13)	0.79 (0.42)	0.81 (0.29)	0.84 (0.45)	0.66 (0.21)	0.88	0.65
Total	28.3 (11.5)	43.6 (12.0)	20.7 (11.1)	38.7 (13.8)	20.2 (10.9)	40.2 (12.6)	23.10	40.83

Table 8: Power consumption (in mW) in each of the different types of functional units. The values in parenthesis show the percentage of the total processor power.

6 Effectiveness of Current Techniques

The designers of the TM3270 have already applied several techniques to reduce the power consumption. We look at the effectiveness of a few of these techniques and their implications on further power optimizations.

6.1 Sequential Tag and Data Access

In the instruction cache, the tags in the selected cache line are read in one cycle, then compared with the program counter's tag in the next cycle. In a third cycle, if there is a hit, the corresponding instruction data is read. On a cache miss, this saves 100% of the power of reading the data from the 8 cache ways. On a cache hit, this saves 7/8ths of the power. Since the instruction data SRAMs consume 10-17% of the total power of the TM3270, this must be a huge savings. If instead, all 8 cache ways were read, the data SRAMs would consume about 8 times as much power, which would increase the total power consumption of the processor by 70-120%. Turning this around, switching from a parallel instruction cache to a sequential one saved about 41-55% of the total processor power.

In the TM3270, there is no branch prediction, so this technique simply increases the number of branch delay slots. In applications with lots of branches, this could have a significantly detrimental impact on the performance. However, the TM3270 provides guarded (predicated) execution - each operation can be selectively ignored based on the contents of any general purpose register - which greatly reduces the number of branches. In addition, function inlining and loop unrolling, which are common in highly optimized production code, further reduce the number of branches. Finally, in the cases where branch conditions can be computed well in advance of the actual branches, there is no performance loss from a few extra branch delay slots.

6.2 Clock Gating

Table 9 shows how often each of the SRAMs and several different functional units are on. These numbers are consistent with the power measurements. For example, comparing this with Table 7, we see that when the data cache tag is on a larger percentage of the time, the power consumed by it is also larger. Also, comparing this with Table 5, we can see that the functional units are turned on only when they are used. For example, in the optimized ADPCM application, the 3 DSPALUs are on an average of 11.3% of the time, which would imply about $(3 * 0.113) = 0.34$ DSPALU operations per

VLIW, and from Table 5, we see that there were 0.35 DSPALU operations per VLIW. This fine-grained clock gating means that we can add new functional units without significantly increasing the power consumption of code that doesn't use the new units. In a low leakage process, such as the TM3270 uses, the extra hardware consumes almost no power while it is clock-gated. Because of this, several application specific operations have been added to newer TriMedia processors that provide significant improvement to the performance of their target applications without having a significant negative impact on the power consumption of other applications. For example, the TM3282 has 5 different deblocking operations, one for each of H.264, VP7, and VC-1, and two for AVS. These perform filters that smooth out blocking artifacts that occur on the boundaries between encoded blocks of pixels, but each codec specifies a different filter.

Application:	ADPCM		MPEG-2		MP3	
Version:	Ref.	Opt.	Ref.	Opt.	Ref.	Opt.
LSU Tag	13.0%	31.5%	10.3%	22.4%	9.5%	34.0%
LSU Data	5.3%	14.2%	6.7%	8.9%	6.5%	29.3%
IFU Tag	66.3%	68.1%	52.8%	68.4%	55.6%	70.5%
IFU Data	60.8%	66.0%	49.9%	65.7%	51.9%	68.9%
FPALU (2)	0.0%	0.0%	3.5%	0.2%	3.6%	28.8%
FPMUL (2)	0.0%	0.0%	2.2%	0.1%	3.8%	25.8%
DSPALU (3)	0.3%	11.3%	0.3%	15.8%	0.0%	0.6%
DSPMUL (2)	0.0%	0.0%	3.0%	19.2%	1.8%	0.0%

Table 9: This table shows the percentage of time that each of the SRAMs and some of the functional units are on (not clock-gated). The numbers in parenthesis indicate how many of each type of functional unit there is.

6.3 Frequency Scaling

The TM3270 also supports frequency and voltage scaling. Unfortunately, the synthesis and simulation libraries are not characterized for different operating voltages. So, we only look at frequency scaling. We run the processor on each application at various operating frequencies from 50MHz to 400MHz, in increments of 50MHz. Table 10(a) shows that the power consumed in each portion of the design when running the reference MP3 application scales with frequency almost exactly as we would expect from the $P = fCV^2$ model. The other reference applications, and the optimized ADPCM application also show this same behavior (the data for these is in Appendix A). We hold V constant, and because the design is constant, C is constant, so we would expect P/f, the energy consumed per cycle, to be a constant, as it is. In these applications, the execution time also scales as expected - doubling the frequency reduces the execution time by half (see Table 4). However, for the optimized MPEG-2 encoder and MP3 decoder, we see something different.

Looking at Table 10(b), we can see that, in the optimized MP3 application, the energy consumed per cycle in the functional units, register file, and decode logic all decrease in a similar manner. This is because these are all clock-gated when the processor stalls for data or instruction cache misses. As the processor speeds up, a larger percentage of time is spent stalling, causing these units to be clock-gated more often, consuming less power. The instruction cache is clock-gated during a data cache stall, and the data cache is clock-gated during an instruction cache stall, but for either stall some portion of the corresponding cache has to remain on in order to resolve the miss. Thus, the power in the caches scale up closer to the expected amount. The optimized MPEG-2 application shows the same behavior (data in Appendix A), though not quite as pronounced. In both of these applications, there are more instruction cache stalls than data cache stalls, which explains why the instruction cache power scales up faster than the data cache power.

(a) Reference MP3 Decoder

Synthesis Freq Operating Freq	400MHz 50MHz	400MHz 100MHz	400MHz 150MHz	400MHz 200MHz	400MHz 250MHz	400MHz 300MHz	400MHz 350MHz	400MHz 400MHz
Instruction Fetch	203 (42.8)	202 (43.1)	202 (43.2)	202 (43.3)	202 (43.3)	201 (43.4)	201 (43.4)	202 (43.4)
Load/Store Unit	86.3 (18.2)	84.4 (18.0)	83.9 (18.0)	83.5 (17.9)	83.2 (17.9)	83.0 (17.9)	83.0 (17.9)	83.0 (17.9)
Functional Units	104 (21.9)	103 (22.1)	103 (22.1)	103 (22.2)	103 (22.2)	103 (22.2)	103 (22.2)	103 (22.2)
Register File	40.4 (8.53)	40.3 (8.60)	40.3 (8.62)	40.2 (8.63)	40.2 (8.64)	40.1 (8.64)	40.1 (8.64)	40.1 (8.64)
Decode Logic	16.0 (3.38)	16.0 (3.42)	16.0 (3.43)	16.0 (3.43)	16.0 (3.44)	15.9 (3.44)	16.0 (3.44)	16.0 (3.44)
Total	473	468	467	466	465	464	464	464

(b) Optimized MP3 Decoder

Synthesis Freq Operating Freq	400MHz 50MHz	400MHz 100MHz	400MHz 150MHz	400MHz 200MHz	400MHz 250MHz	400MHz 300MHz	400MHz 350MHz	400MHz 400MHz
Instruction Fetch	259 (31.1)	258 (31.3)	258 (31.3)	257 (31.4)	258 (31.6)	259 (32.1)	260 (32.4)	260 (32.8)
Load/Store Unit	265 (31.8)	262 (31.7)	261 (31.8)	260 (31.9)	259 (31.8)	256 (31.7)	252 (31.5)	250 (31.4)
Functional Units	188 (22.6)	187 (22.7)	187 (22.7)	185 (22.6)	184 (22.5)	180 (22.3)	177 (22.1)	174 (21.9)
Register File	76.1 (9.13)	75.7 (9.17)	75.4 (9.16)	74.7 (9.13)	74.3 (9.10)	72.9 (9.02)	71.8 (8.97)	70.7 (8.90)
Decode Logic	19.0 (2.28)	19.0 (2.29)	18.8 (2.29)	18.6 (2.28)	18.5 (2.27)	18.1 (2.25)	17.9 (2.23)	17.6 (2.21)
Total	833	826	823	818	817	808	801	795

Table 10: These tables show the per cycle energy consumption (in pJ/cycle) in each portion of the 400MHz design for both versions of the MP3 application when run at several different operating frequencies. The values in parenthesis show the percentage of the total processor energy/cycle.

7 Further Power Reduction

The breakdown of power among the various components of the design point to several possible methods for reducing the power consumption. We look at several such methods, and evaluate their effectiveness at reducing power and their performance and area implications.

7.1 Synthesis Frequency Scaling

One possibility for reducing the power consumption is to synthesize the processor with a lower target frequency. This will cause the synthesis tools to select smaller, more power-efficient standard cells to implement the necessary logic. In this set of experiments, we synthesize the processor at several different target frequencies from 50MHz to 400MHz, in increments of 50MHz. We then simulate the execution of each application on each of these different designs. Each design is run at its synthesis frequency.

From Table 11, we can see that, when we scale the synthesis target frequency down, the energy consumed per cycle decreases. This is because the synthesis tool uses lower drive-strength standard cells in order to meet the more relaxed timing constraints of the lower frequency designs. This decreases the capacitance as the frequency decreases. This shows only the data for the reference ADPCM application. For the data on the other applications see Appendix B.

Synthesis Freq Operating Freq	50MHz 50MHz	100MHz 100MHz	150MHz 150MHz	200MHz 200MHz	250MHz 250MHz	300MHz 300MHz	350MHz 350MHz	400MHz 400MHz
Instruction Fetch	226 (39.4)	225 (39.5)	225 (39.4)	228 (39.4)	230 (39.1)	232 (38.8)	238 (39.2)	239 (38.8)
Load/Store Unit	125 (21.8)	124 (21.7)	124 (21.6)	124 (21.5)	125 (21.2)	126 (21.1)	126 (21.0)	127 (20.7)
Functional Units	119 (20.7)	119 (20.8)	123 (21.4)	124 (21.5)	130 (22.0)	134 (22.5)	139 (22.9)	145 (23.5)
Register File	60.4 (10.5)	60.3 (10.6)	60.0 (10.5)	60.8 (10.5)	61.8 (10.5)	63.4 (10.6)	60.2 (9.93)	62.5 (10.1)
Decode Logic	20.8 (3.63)	20.8 (3.65)	20.8 (3.64)	20.7 (3.58)	20.7 (3.52)	21.0 (3.51)	22.0 (3.64)	21.5 (3.49)
Total	575	570	573	578	588	598	606	616

Table 11: These tables show the per cycle energy consumption (in pJ/cycle) in each portion of each of the designs while running the reference ADPCM application at their synthesis target frequency. The values in parenthesis show the percentage of the total processor energy/cycle.

The decrease in energy/cycle is much larger in the functional units than in any other part of the design. From Table 2, we can see that synthesizing at higher frequencies increases the area in the functional units by much more than any other component of the design. This explains the fact that the power consumed in the Functional Units scales the fastest - the synthesis tool is using the larger, faster versions of the required standard cells much more in the functional units than anywhere else in the design. This indicates that a relatively large portion of the functional units are on the critical path. As we saw when we just scaled the operating frequency, the power scales similarly regardless of application, except for the effects of the stalls in the optimized MPEG-2 and MP3 applications.

The obvious alternative to synthesis at a lower frequency is to synthesize it at a high frequency and then scale down the supply voltage. If we synthesize at 100MHz instead of 350MHz, we save 3-7% in terms of energy per cycle. As an alternative, we take a real TM3270 that runs at 350MHz, and scale its operating frequency down to 100MHz. This allowed us to reduce the operating voltage from 1.226V down to 1.012V, which saves an average of 17% of the energy per cycle across our 6 applications. In fact, this particular TM3270 consumes the least energy per cycle at 200MHz, where it saves an average of 22% compared to running at 350MHz. Thus, we conclude that decreasing the synthesis frequency is not energy-efficient, by itself. However, we may be able to use the extra time in each clock cycle to further reduce the power of other components (i.e. by using slower SRAMs) or to increase the performance (i.e. by reducing the latencies of operations or reducing the number of jump delay slots).

7.2 Reducing SRAM Power

Because the SRAMs are such large contributors to the power consumption, we look at them next. The instruction data SRAM consumes the most, followed by the LSU data SRAM and IFU tag SRAM. These are the largest SRAMs and have the most potential for power savings.

7.2.1 Skipping Duplicate Tag SRAM Reads

Instruction cache lines are 128 bytes long, but only 32 bytes of instruction data are read each cycle. In the absence of branches, all four 32-byte blocks of each cache line will be read sequentially. In this case, the tag SRAM is accessed four times in a row, even though the tag and index aren't changing. This observation leads to a very simple optimization that reduces the tag SRAM power by nearly 75% by disabling the tag SRAM whenever the current PC and next PC belong to the same cache line. This is very similar to the idea of the quadword buffer used in [20] and the cache line buffer in [21], but because the instruction data SRAM is accessed after the tag has been read and compared, we do not need the extra complexity. Synthesizing the design with this new technique did not change the total area or maximum operating frequency significantly. Table 12 shows the power savings of this technique.

Application:	ADPCM		MPEG-2		MP3	
Version:	Ref.	Opt.	Ref.	Opt.	Ref.	Opt.
Old IFU Tag	21.07	21.58	16.61	21.74	17.66	22.20
New IFU Tag	7.194	6.206	5.544	6.376	5.396	6.33
Savings	65.9%	71.2%	66.6%	70.7%	69.4%	71.5%
Old Total	246.59	363.48	186.37	280.13	185.79	317.95
New Total	232.23	348.12	174.03	264.75	173.42	302.02
Savings	5.82%	4.23%	6.62%	5.49%	6.66%	5.01%

Table 12: This table shows the power savings obtained by turning off the instruction cache tag SRAM whenever the current and next program counters belong to the same cache line.

Note that the percentage of the total power that this technique saves is less for the optimized applications than for the non-optimized ones. If we had used only reference applications, we would have overestimated the power savings from this technique. In addition, any further optimizations to the tag SRAM power will be nearly insignificant compared to the power consumed by the other SRAMs. Thus, using techniques like way-prediction and way-halting just to save the tag power will not be very useful. These techniques make more sense when the tag and data are accessed in parallel. Even if they are accessed in parallel, we can "predict" that the cache way will be the same way that was accessed in the previous cycle whenever the current and next PC belong to the same cache line without any extra lookup tables.

This same technique could be used in the data cache to reduce the power of tag accesses; however, the data cache tag accesses aren't as large a percentage of the total power, and sequential access isn't as common in the data cache. In addition, when two addresses are statically known to be sequential, the compiler or programmer can combine two load operations into one super load operation, which already reduces the tag power because the tag is only read once for both of the two words. Also, for performance reasons, the instruction scheduler does not always put sequential data accesses back-to-back.

7.2.2 Instruction Cache SRAM Reorganization

Another possibility for power reduction is to change the size and shape of the individual SRAMs. For example, in the instruction cache, the instruction data is stored in two separate 32KB SRAMs, each 16 bytes wide. Each time instruction data is read, both SRAMs are accessed to get a total of 32 bytes. Another possibility would be to use SRAMs that are 32 bytes wide; Then, only one SRAMs would be read each cycle. Table 13 shows several different possible organizations for the instruction data SRAMs and the effects on area, cycle time and energy. This data comes from Virage Logic (a provider of SRAMs), which also provides the real physical memory instantiations (LEF).

Using a single 32-byte wide 64KB SRAM consumes slightly less energy, but is slower, and because these SRAMs are on the critical path, this would cause a significant performance loss. If, instead, we use four 32-byte wide 16KB SRAMs, we can save nearly half of the SRAM energy with approximately a 15% increase in SRAM area. The SRAMs themselves are also faster than the 16-byte wide 32KB SRAMs, but an additional multiplexor would have to be added at the output of the four SRAMs. In addition, the layout and wiring of the four separate SRAMs is significantly more complicated. The overhead in silicon area is also significant. Without being able to quantify all these effects, it is impossible to draw any real conclusions, but we suspect that these overheads combine to make this technique energy-inefficient. Trying the same technique for the data cache SRAMs is even less promising, though for brevity, we have omitted that data.

Number of SRAMs	SRAM size	Line Width	Total Area	Max Cycle Time	Energy
2	32KB	16B	0.934mm^2	1.34ns	371.8pJ/access
4*	16KB	16B	1.040mm^2	1.15ns	317.0pJ/access
1	64KB	32B	0.868mm^2	1.44ns	346.9pJ/access
2*	32KB	32B	0.968mm^2	1.30ns	295.5pJ/access
4*	16KB	32B	1.064mm^2	1.22ns	195.5pJ/access

Table 13: Different choices for the organization of the instruction data SRAMs. This data comes from the data sheets for Philips' 90nm LP-process, high-speed, ultra-high density SRAMs. The first row is the configuration used for all simulations. For the 16B line widths, two SRAMs are accessed each cycle, and for the 32B line widths, only one SRAM is accessed each cycle. The area given does not include external wiring and multiplexing. The ones marked with *'s require a multiplexor at the output.

An alternative to this is to insert a small L0 cache in front of the L1 cache. According to Kin et al. [5, 6], this can have a large performance penalty. Tagless loop caches, proposed by Lee et al. [22]

and Gordon-Ross et al. [23, 24], can save a lot of power with no performance loss in applications with small loops. Unfortunately, highly optimized applications tend to have few, if any, small loops. For example, a reference MPEG-2 encoder may have many loops over each of 8 rows in a block of pixels. However, when this code is optimized, these loops are generally fully unrolled. In fact, a major portion of our optimized MPEG-2 encoder consists of several different branchless sequences of several hundred VLIWs. These sequences range in size from 3852 bytes to 7445 bytes, and two different sequences are required in each frame. Other, more complicated, encoders and decoders would be even harder to fit in a small loop cache.

7.2.3 Low Power SRAMs

One possibility for reducing the power consumption is to use low power SRAMs. Unfortunately, these SRAMs have about twice the delay, and this much performance loss is unacceptable, even though they consume about one-third as much energy per cycle. Since SRAMs consume about 40% of the total processor power, this would save about 27% of the total processor energy/cycle. If instead we use the fast SRAMs, we can run at the same reduced frequency (reduced by 50%), and reduce the operating voltage. Using a real TM3270 chip with fast SRAMs, we found that reducing the frequency from 350MHZ to 175MHz allowed us to reduce the operating voltage by 16%, which yielded a 29% reduction in energy/cycle.

7.3 Functional Units

Looking into the design, we notice that the first stage of some of the functional units share the same source and opcode flip-flops. There are only two sets of flip-flops, one for the floating point multiply and divide units, and one for everything else. This means that when an ALU operation is executed, the first stage of the DSPMUL unit and FPALU both see new source values, causing useless switching. This may have been done as an area or timing optimization, but it increases the power consumption. Table 14 shows the power savings that result if we give the first stage of the DSPMUL unit its own flip-flops. As we can see, in the applications that contain no integer multiplies, the power consumption in the DSPMUL is almost completely eliminated. A similar optimization could be applied to the FPALUs, though the gains would be much more modest since they consume much less power to begin with. Similarly, some power could be saved in the integer ALUs using this technique; even though they are the most utilized functional units, they are still on but unused some of the time.

Application:	ADPCM		MPEG-2		MP3	
Version:	Ref.	Opt.	Ref.	Opt.	Ref.	Opt.
Old DSPMUL	6.178	10.15	6.637	15.84	6.342	9.127
New DSPMUL	1.211	1.208	2.850	11.35	2.304	1.036
Savings	80.4%	88.1%	57.1%	28.3%	63.6%	88.6%
Old Total	246.59	363.48	185.22	280.13	185.79	317.95
New Total	240.32	353.43	180.94	275.16	180.94	308.77
Savings	2.54%	2.76%	2.31%	1.77%	2.61%	2.89%

Table 14: This table shows the power savings obtained by giving the first stage of the DSPMUL unit its own flip-flops and clock-gating them when not in use.

8 Conclusions and Future Work

We have taken a detailed look at the power consumptions of a real, commercially available embedded multimedia processor. To obtain accurate and detail power consumption data, we used commercial synthesis, simulation, and power analysis tools. We compared the breakdown of where power is consumed on non-optimized benchmark code with the breakdown on optimized code. In the future, we plan to compare these numbers against production silicon, and investigate any systematic errors found.

We have found that power reduction techniques such as clock gating and sequential tag and data access not only provide significant improvement in power consumption (41-55% in the case of sequential tag and data access in the instruction cache), but also greatly impact other decisions about power reduction. When fine-grained clock-gating is used in conjunction with a low-power process, then adding additional functional units for application specific tasks can improve the power-performance of those tasks without significantly damaging the power-performance of other tasks. It will be interesting to look at other techniques such as way-prediction and study both how current techniques affect its power-savings potential and how way-prediction affects the usefulness of further techniques.

In addition, we found a few simple changes could have a significant impact on the power consumption and help us to focus on more critical components. For example, disabling the instruction tag access on sequential instructions is extremely simple, but makes the instruction tag power consumption almost negligible, so we can focus further efforts elsewhere. Also, giving each functional unit its own flip-flops for holding source values and intermediate results can, at a small cost to area, eliminate useless switching, particularly important for power-hungry functional units such as multipliers. We plan to look for any further optimizations such as these.

Finally, we looked at the difference in processor power consumption when running non-optimized and (performance-)optimized software. First, we found that hand-optimization was enormously effective, and the total energy consumption dropped by a large fraction; in fact, the decrease in energy from code optimization swamped most hardware and architecture improvements. Comparing optimized with non-optimized code, we found that optimization had a moderate impact on how effective various power savings techniques were. Running optimized versions of even more modern software, such as a VC-1 decoder or a temporal video up-converter, may show an even larger impact as applications become more memory-bound. For future work, we plan to look in more detail at the effects of individual software optimizations.

References

[1] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems," in *Proc. 30th Ann. ACM/IEEE Int. Symp. Microarch.*, pp. 330–335, 1997.

[2] M. Horowitz, "Scaling, Power and the Future of CMOS," in *Proc. 20th Int. Conf. VLSI Design*, p. 23, 2007.

[3] O. S. Unsal and I. Koren, "System-Level Power-Aware Design Techniques in Real-Time Systems," *Proc. IEEE*, vol. 91, no. 7, 2003.

[4] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. L. Scott, "Integrating Adaptive On-Chip Storage Structures for Reduced Dynamic Power," in *Proc. Int. Conf. Parallel Archit. and Compil. Tech.*, p. 141, 2002.

[5] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," in *Proc. 30th Ann. ACM/IEEE Int. Symp. Microarch.*, pp. 184–193, 1997.

[6] J. Kin, M. Gupta, and W. H. Mangione-Smith, "Filtering Memory References to Increase Energy Efficiency," *IEEE Trans. Comput.*, vol. 49, no. 1, pp. 1–15, 2000.

[7] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing Set-Associative Cache Energy Via Way-Prediction and Selective Direct-Mapping," in *Proc. 34th Ann. ACM/IEEE Int. Symp. Microarch.*, pp. 54–65, 2001.

[8] L. Wehmeyer, M. Jain, S. Steinke, P. Marwedel, and M. Balakrishnan, "Analysis of the Influence of Register File Size on Energy Consumption, Code Size and Execution Time," *IEEE TCAD*, vol. 20, no. 11, pp. 1329–1337, 2001.

[9] I. Park, M. D. Powell, and T. N. Vijaykumar, "Reducing Register Ports for Higher Speed and Lower Energy," in *Proc. 35th Ann. ACM/IEEE Int. Symp. Microarch.*, (Los Alamitos, CA, USA), pp. 171–182, 2002.

[10] S. Kim and M. C. Papaefthymiou, "Reconfigurable Low Energy Multiplier for Multimedia System Design," in *Proc. IEEE Workshop on VLSI*, p. 129, 2000.

[11] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *Proc. Int. Symp. Comp. Archit.*, pp. 83–94, 2000.

[12] Y.-K. Cheng, D. Beared, and K. Suryadevara, "Application-Based, Transistor-Level Full-Chip Power Analysis for 700 MHz PowerPC(tm) Microprocessor," in *Proc. ICCD*, p. 215, 2000.

[13] K. I. Farkas, J. Flinn, G. Back, D. Grunwald, and J. M. Anderson, "Quantifying the Energy Consumption of a Pocket Computer and a Java Virtual Machine," in *Proc. SIGMETRICS*, (New York, NY, USA), pp. 252–263, 2000.

[14] S. Segars, "ARM7TDMI Power Consumption," *IEEE Micro*, vol. 17, no. 4, pp. 12–19, 1997.

[15] M. Lorenz, P. Marwedel, T. Dräger, G. Fettweis, and R. Leupers, "Compiler Based Exploration of DSP Energy Savings by SIMD Operations," in *Proc. Conf. Asia South Pacific Design Automation*, (Piscataway, NJ, USA), pp. 838–841, 2004.

[16] J. Sebot and N. Drach, "SIMD ISA Extensions: Power Efficiency on Multimedia on a Superscalar Processor(Special Issue on High-Performance and Low-Power Microprocessors)," *IEICE Trans. Elec.*, vol. 85, no. 2, pp. 297–303, 20020201.

[17] N. Slingerland and A. J. Smith, "Measuring the Performance of Multimedia Instruction Sets," *IEEE Trans. Comput.*, vol. 51, no. 11, pp. 1317–1332, 2002.

[18] J.-W. van de Waerdt, S. Vassiliadis, S. Das, S. Mirolo, C. Yen, B. Zhong, C. Basto, J.-P. van Itegem, D. Amirtharaj, K. Kalra, P. Rodriguez, and H. van Antwerpen, "The TM3270 Media-Processor," in *Proc. 38th Ann. ACM/IEEE Int. Symp. Microarch.*, pp. 331–342, 2005.

[19] W. Zhang, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, D. Duarte, and Y.-F. Tsai, "Exploiting VLIW Schedule Slacks for Dynamic and Leakage Energy Reduction," in *Proc. 34th Ann. ACM/IEEE Int. Symp. Microarch.*, pp. 102–113, 2001.

[20] J. Cho, A. J. Smith, and H. Sachs, "The memory architecture and the cache and memory management unit for the fairchild clipper processor," tech. rep., Berkeley, CA, USA, 1986.

[21] K. Ali, M. Aboelaze, and S. Datta, "Predictive Line Buffer: A Fast, Energy Efficient Cache Architecture," in *Proc. IEEE SoutheastCon*, pp. 291–295, 2006.

[22] L. H. Lee, J. Scott, B. Moyer, and J. Arends, "Low-Cost Branch Folding for Embedded Applications with Small Tight Loops," in *Proc. 32nd Ann. ACM/IEEE Int. Symp. Microarch.*, pp. 103–111, 1999.

[23] A. Gordon-Ross and F. Vahid, "Dynamic Loop Caching Meets Preloaded Loop Caching: A Hybrid Approach," in *Proc. ICCD*, p. 446, 2002.

[24] A. Gordon-Ross, S. Cotterell, and F. Vahid, "Exploiting Fixed Programs in Embedded Systems: A Loop Cache Example," *IEEE Comput. Archit. Lett.*, vol. 1, no. 1, p. 2, 2006.

[25] N. T. Slingerland and A. J. Smith, "Design and Characterization of the Berkeley Multimedia Workload," *Multimedia Syst.*, vol. 8, no. 4, pp. 315–327, 2002.

[26] K. Lagerstrm, "Design and Implementation of an MP3 Decoder," Master's thesis, May 2001.

[27] J.-W. van de Waerdt and S. Vassiliadis, "Instruction Set Architecture Enhancements for Video Processing," in *Proceedings IEEE Int. Conf. on Application-Specific Systems, Architecture Processors*, pp. 146–153, 2005.

Appendix A.

(a) Reference ADPCM Decoder

Synthesis Freq	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	241 (38.5)	240 (38.7)	240 (38.8)	239 (38.8)	239 (38.8)	238 (38.8)	239 (38.8)	239 (38.8)
Load/Store Unit	130 (20.8)	128 (20.7)	128 (20.6)	128 (20.7)	127 (20.7)	127 (20.7)	127 (20.7)	127 (20.7)
Functional Units	145 (23.3)	145 (23.4)	145 (23.5)	145 (23.5)	145 (23.5)	145 (23.5)	145 (23.5)	145 (23.5)
Register File	62.7 (10.0)	62.6 (10.1)	62.6 (10.1)	62.6 (10.1)	62.5 (10.1)	62.4 (10.1)	62.5 (10.1)	62.5 (10.1)
Decode Logic	21.6 (3.45)	21.5 (3.48)	21.6 (3.48)	21.5 (3.49)	21.5 (3.49)	21.5 (3.49)	21.5 (3.49)	21.5 (3.49)
Total	625	620	619	618	617	616	616	616

(b) Optimized ADPCM Decoder

Synthesis Freq	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	266 (29.0)	265 (29.1)	265 (29.1)	265 (29.1)	265 (29.1)	264 (29.1)	264 (29.1)	264 (29.1)
Load/Store Unit	285 (31.1)	283 (31.1)	283 (31.1)	282 (31.1)	283 (31.1)	282 (31.1)	283 (31.1)	283 (31.1)
Functional Units	215 (23.4)	215 (23.5)	215 (23.6)	214 (23.6)	214 (23.6)	214 (23.6)	214 (23.6)	214 (23.6)
Register File	97.3 (10.6)	97.2 (10.7)	97.2 (10.7)	97.1 (10.7)	97.1 (10.7)	96.8 (10.7)	96.9 (10.7)	97.0 (10.7)
Decode Logic	28.5 (3.11)	28.5 (3.13)	28.6 (3.14)	28.5 (3.14)	28.5 (3.14)	28.5 (3.14)	28.5 (3.14)	28.5 (3.14)
Total	917	911	911	909	909	907	908	909

(c) Reference MPEG-2 Encoder

Synthesis Freq	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	194 (41.2)	194 (41.5)	194 (41.6)	193 (41.6)	193 (41.7)	193 (41.7)	193 (41.8)	193 (41.8)
Load/Store Unit	95.0 (20.1)	93.1 (19.9)	92.5 (19.9)	92.1 (19.9)	92.0 (19.8)	91.8 (19.8)	91.7 (19.8)	91.7 (19.8)
Functional Units	104 (22.1)	104 (22.3)	104 (22.4)	104 (22.4)	104 (22.4)	104 (22.4)	104 (22.4)	104 (22.4)
Register File	38.7 (8.21)	38.6 (8.28)	38.6 (8.30)	38.6 (8.31)	38.5 (8.31)	38.4 (8.31)	38.5 (8.31)	38.5 (8.31)
Decode Logic	15.0 (3.19)	15.0 (3.22)	15.0 (3.23)	15.0 (3.24)	15.0 (3.24)	15.0 (3.24)	15.0 (3.24)	15.0 (3.24)
Total	472	467	465	464	464	463	463	463

(d) Optimized MPEG-2 Encoder

Synthesis Freq	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	253 (34.9)	253 (35.1)	252 (35.2)	251 (35.2)	252 (35.3)	252 (35.7)	252 (35.9)	253 (36.1)
Load/Store Unit	175 (24.2)	173 (24.1)	173 (24.1)	173 (24.2)	173 (24.2)	171 (24.2)	170 (24.2)	170 (24.2)
Functional Units	182 (25.1)	181 (25.2)	181 (25.2)	179 (25.1)	178 (25.0)	175 (24.8)	174 (24.7)	172 (24.5)
Register File	69 (9.56)	69 (9.61)	69 (9.61)	68 (9.58)	68 (9.56)	67 (9.47)	66 (9.44)	66 (9.39)
Decode Logic	20.3 (2.79)	20.2 (2.81)	20.2 (2.81)	20.0 (2.80)	19.9 (2.79)	19.6 (2.76)	19.4 (2.75)	19.2 (2.74)
Total	726	720	718	714	713	707	704	700

(e) Reference MP3 Decoder

Synthesis Freq	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	203 (42.8)	202 (43.1)	202 (43.2)	202 (43.3)	202 (43.3)	201 (43.4)	201 (43.4)	202 (43.4)
Load/Store Unit	86.3 (18.2)	84.4 (18.0)	83.9 (18.0)	83.5 (17.9)	83.2 (17.9)	83.0 (17.9)	83.0 (17.9)	83.0 (17.9)
Functional Units	104 (21.9)	103 (22.1)	103 (22.1)	103 (22.2)	103 (22.2)	103 (22.2)	103 (22.2)	103 (22.2)
Register File	40.4 (8.53)	40.3 (8.60)	40.3 (8.62)	40.2 (8.63)	40.2 (8.64)	40.1 (8.64)	40.1 (8.64)	40.1 (8.64)
Decode Logic	16.0 (3.38)	16.0 (3.42)	16.0 (3.43)	16.0 (3.43)	16.0 (3.44)	15.9 (3.44)	16.0 (3.44)	16.0 (3.44)
Total	473	468	467	466	465	464	464	464

(f) Optimized MP3 Decoder

Synthesis Freq	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	259 (31.1)	258 (31.3)	258 (31.3)	257 (31.4)	258 (31.6)	259 (32.1)	260 (32.4)	260 (32.8)
Load/Store Unit	265 (31.8)	262 (31.7)	261 (31.8)	260 (31.9)	259 (31.8)	256 (31.7)	252 (31.5)	250 (31.4)
Functional Units	188 (22.6)	187 (22.7)	187 (22.7)	185 (22.6)	184 (22.5)	180 (22.3)	177 (22.1)	174 (21.9)
Register File	76.1 (9.13)	75.7 (9.17)	75.4 (9.16)	74.7 (9.13)	74.3 (9.10)	72.9 (9.02)	71.8 (8.97)	70.7 (8.90)
Decode Logic	19.0 (2.28)	19.0 (2.29)	18.8 (2.29)	18.6 (2.28)	18.5 (2.27)	18.1 (2.25)	17.9 (2.23)	17.6 (2.21)
Total	833	826	823	818	817	808	801	795

Table 15: These tables show the per cycle energy consumption (in pJ/cycle) in each portion of the 400MHz design when run at several different operating frequencies. The values in parenthesis show the percentage of the total processor energy/cycle.

Appendix B.

(a) Reference ADPCM Decoder

Synthesis Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	226 (39.4)	225 (39.5)	225 (39.4)	228 (39.4)	230 (39.1)	232 (38.8)	238 (39.2)	239 (38.8)
Load/Store Units	125 (21.8)	124 (21.7)	124 (21.6)	124 (21.5)	125 (21.2)	126 (21.1)	126 (21.0)	127 (20.7)
Functional Units	119 (20.7)	119 (20.8)	123 (21.4)	124 (21.5)	130 (22.0)	134 (22.5)	139 (22.9)	145 (23.5)
Register File	60.4 (10.5)	60.3 (10.6)	60.0 (10.5)	60.8 (10.5)	61.8 (10.5)	63.4 (10.6)	60.2 (9.93)	62.5 (10.1)
Decode Logic	20.8 (3.63)	20.8 (3.65)	20.8 (3.64)	20.7 (3.58)	20.7 (3.52)	21.0 (3.51)	22.0 (3.64)	21.5 (3.49)
Total	575	570	573	578	588	598	606	616

(b) Optimized ADPCM Decoder

Synthesis Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	249 (29.5)	248 (29.5)	248 (29.4)	252 (29.5)	254 (29.2)	257 (29.1)	263 (29.4)	264 (29.1)
Load/Store Unit	275 (32.5)	273 (32.5)	273 (32.3)	274 (32.2)	277 (31.8)	280 (31.7)	282 (31.6)	283 (31.1)
Functional Units	175 (20.7)	175 (20.8)	181 (21.4)	182 (21.4)	191 (22.0)	198 (22.4)	204 (22.8)	214 (23.6)
Register File	94.9 (11.2)	94.8 (11.3)	94.3 (11.2)	96.3 (11.3)	98.8 (11.3)	98.6 (11.2)	94.9 (10.6)	97.0 (10.7)
Decode Logic	28.3 (3.34)	28.3 (3.36)	28.2 (3.34)	27.9 (3.27)	27.8 (3.20)	28.2 (3.20)	29.6 (3.31)	28.5 (3.14)
Total	845	841	846	853	871	884	894	909

(c) Reference MPEG-2 Encoder

Synthesis Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	182 (41.9)	182 (42.1)	181 (42.0)	184 (42.2)	185 (41.8)	189 (41.8)	192 (42.1)	193 (41.8)
Load/Store Unit	91.4 (21.0)	89.5 (20.8)	89.3 (20.7)	89.3 (20.5)	90.1 (20.3)	90.4 (20.0)	91.4 (20.0)	91.7 (19.8)
Functional Units	86.9 (20.0)	86.9 (20.2)	89.0 (20.6)	90.6 (20.8)	94.0 (21.2)	97.9 (21.6)	100 (21.9)	104 (22.4)
Register File	37.5 (8.62)	37.5 (8.70)	37.2 (8.63)	37.7 (8.69)	38.6 (8.71)	39.8 (8.79)	37.3 (8.17)	38.5 (8.31)
Decode Logic	14.2 (3.27)	14.2 (3.31)	14.2 (3.30)	14.2 (3.25)	14.2 (3.21)	14.6 (3.22)	15.1 (3.30)	15.0 (3.24)
Total	435	431	432	436	443	452	457	463

(d) Optimized MPEG-2 Encoder

Synthesis Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	239 (35.8)	238 (36.0)	238 (35.8)	240 (35.8)	243 (35.5)	246 (35.8)	251 (36.2)	253 (36.1)
Load/Store Unit	169 (25.3)	167 (25.2)	167 (25.1)	168 (25.1)	169 (24.8)	170 (24.7)	170 (24.5)	169 (24.2)
Functional Units	148 (22.2)	148 (22.4)	153 (22.9)	154 (23.0)	161 (23.6)	164 (23.8)	167 (24.1)	172 (24.5)
Register File	67.5 (10.1)	67.3 (10.2)	66.8 (10.0)	67.6 (10.1)	68.8 (10.1)	68.1 (9.88)	64.8 (9.33)	65.8 (9.39)
Decode Logic	19.8 (2.96)	19.8 (2.98)	19.7 (2.96)	19.4 (2.89)	19.3 (2.82)	19.2 (2.79)	19.8 (2.86)	19.2 (2.74)
Total	668	663	666	671	684	689	694	700

(e) Reference MP3 Decoder

Synthesis Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	190 (43.6)	190 (43.9)	190 (43.8)	192 (43.9)	194 (43.5)	195 (43.4)	201 (43.8)	202 (43.4)
Load/Store Unit	82.9 (19.0)	81.1 (18.8)	80.8 (18.7)	80.8 (18.5)	81.5 (18.3)	82.3 (18.3)	82.8 (18.1)	83.0 (17.9)
Functional Units	86.4 (19.8)	86.4 (20.0)	88.5 (20.4)	90.1 (20.6)	93.4 (21.0)	96.0 (21.3)	99.1 (21.7)	103 (22.2)
Register File	39.1 (8.94)	39.0 (9.02)	38.8 (8.95)	39.5 (9.02)	40.3 (9.07)	40.7 (9.03)	38.9 (8.50)	40.1 (8.64)
Decode Logic	15.2 (3.47)	15.2 (3.51)	15.2 (3.50)	15.1 (3.45)	15.2 (3.41)	15.4 (3.42)	16.1 (3.51)	16.0 (3.44)
Total	437	432	433	438	445	451	458	464

(f) Optimized MP3 Decoder

Synthesis Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Operating Freq	50MHz	100MHz	150MHz	200MHz	250MHz	300MHz	350MHz	400MHz
Instruction Fetch	245 (31.7)	244 (31.8)	244 (31.7)	246 (31.8)	250 (31.7)	253 (32.0)	258 (32.6)	260 (32.8)
Load/Store Unit	254 (32.9)	252 (32.8)	252 (32.8)	252 (32.6)	254 (32.3)	254 (32.2)	252 (31.9)	250 (31.4)
Functional Units	156 (20.2)	156 (20.4)	160 (20.8)	161 (20.9)	167 (21.2)	169 (21.4)	170 (21.6)	174 (21.9)
Register File	74.4 (9.62)	74.0 (9.65)	73.2 (9.52)	74.2 (9.59)	75.7 (9.62)	74.1 (9.38)	70.2 (8.87)	70.7 (8.90)
Decode Logic	18.8 (2.43)	18.7 (2.44)	18.7 (2.42)	18.3 (2.36)	18.1 (2.30)	18.0 (2.27)	18.4 (2.33)	17.6 (2.21)
Total	773	767	769	773	787	790	791	795

Table 16: These tables show the per cycle energy consumption (in pJ/cycle) in each portion of each of the designs when run at their synthesis target frequency. The values in parenthesis show the percentage of the total processor energy/cycle.

Appendix C.   Application Mix

C-1.   ADPCM decoder

The ADPCM decoder application is an audio decoder primarily intended for speech. This application appears in several multimedia workloads including MediaBench and the Berkeley Multimedia Workload (BMW) [25]. We use the reference code from the BMW. It is simple, consisting of a single loop that decodes each 4-bit piece of the encoded bitstream into a 16-bit sample. Our fairly straightforward optimization consists of loop unrolling, use of TM3270 specific operations for clipping and guarded execution, and some reordering of the code. Combined, these optimizations result in a speedup of approximately 3x. These optimizations reduced the amount of computation needed, and gave the compiler more opportunity to overlap independent operations.

We use the "gore_internet" data set from the BMW for all of our experiments with this application. The encoded bitstream is 95KB long, and is decoded into a 380KB PCM sound file. The clip is approximately 24 seconds long at a sample rate of 8kHz. This data set is large enough that cache warm-up effects are negligible, but small enough that simulation can be done in a reasonable amount of time.

C-2.   MP3 decoder

The MP3 decoder application is an audio decoder commonly used for music. This application is widely used by consumers on PCs and in portable MP3 players. It also appears in multimedia workloads such as the BMW (the mpg123 application is an MP3 decoder). We use the low sampling frequency reference code in revision 10 of the reference code for ISO 13818-3 MPEG-2 Audio Codec, but the optimized code we obtained is an optimized version of Lagerstrom's MP3 decoder described in his thesis[26]. Lagerstrom's MP3 decoder uses faster algorithms for various portions of the MP3 decoder, such as the Requantizer, and the IMDCT. Further improvements for the TM3270 include loop unrolling, hardcoding of known constants, and if-conversion, which together greatly increase the amount of available parallelism.

C-3. MPEG-2 encoder

The MPEG-2 encoder application is a very popular video encoder. This application also appears in the BMW (and an MPEG-2 decoder appears in MediaBench). It consists of motion estimation, motion compensation, discrete cosine transform (DCT), quantization, run-level encoding, Huffman encoding, inverse quantization, inverse discrete cosine transform (IDCT), motion reconstruction, and adaptive bit-rate control.

We obtained a highly optimized version of this application to compare against the reference implementation. The optimizations used include fast (but highly accurate) DCT and IDCT using packed 16-bit fixed-point arithmetic and combining of the motion compensation, DCT, quantization, run-level encoding, inverse quantization, IDCT, and motion reconstruction into a single pipeline. The fast DCT and IDCT require many fewer arithmetic operations, while the combination of all these components into a single pipeline reduces the number of loads and stores needed, and provides the compiler more opportunity for overlapping independent operations. For a more complete descriptions of the optimizations used, see [27].

The optimized encoder uses a much faster, but slightly lower quality motion estimator. In order to factor out this quality difference between the two encoders, we precompute the motion vectors, and use the same set of motion vectors for both the optimized and non-optimized version. Thus, the power measurements do not include the motion estimation portion of the application.

Because netlist simulation is so slow, we limit this application to encoding a single 352x16 slice. We use the first row of the second frame of the mei16v2 data set from the BMW for all of our experiments with this application. This frame is encoded as a P-frame. We only measure power consumption of the actual encoding - the power consumption during initialization (setup of quantization tables, etc) is ignored.