Reducing Register File Energy Consumption using Compiler Support[*]

José L. Ayala[†]
Departamento de Ingeniería Electrónica
Universidad Politécnica de Madrid
Spain
jayala@die.upm.es

Alexander Veidenbaum
Center for Embedded Computer Systems
University of California, Irvine
USA
alexv@ics.uci.edu

ABSTRACT

Most power reduction techniques have focused on gating the clock to unused functional units to minimize static power consumption, while system level optimizations have been used to deal with dynamic power consumption. After these techniques are applied, register file power consumption becomes a dominant factor in the energy consumption. This paper proposes a power-aware reconfiguration mechanism in the register file driven by a compiler. Optimal usage of the register file in terms of size is achieved and unused registers are put into a low-power state. Total energy consumption in the register file is reduced by 60%, on average, with no appreciable performance penalty. The effect of reconfiguration granularity on energy savings is also analyzed, and the compiler approach to optimize register file energy is presented.

Categories and Subject Descriptors

C.5 [**Computer System Implementation**]: Microcomputers—*microprocessors*

General Terms

Register File management

Keywords

register file management, compiler support, energy aware, power management, low power

[*]This work was supported in part by the DARPA PAC/C program under contract F336-15-00-C-1632 and by the Spanish Ministry of Science and Technology under contract TIC2000-0583-C02-02

[†]This work has been performed during a research visit to Center for Embedded Computer Systems (University of California, Irvine)

1. INTRODUCTION

Low power is an important concern in processor design, especially for mobile and embedded systems. Battery technology improvements are not satisfying the fast growing requirements in system design and performance has to be sacrificed to extend the battery-life.

There are many reasons why power consumption is becoming a limiting factor in high performance processors. New semiconductor processes allow higher integration density and larger chips, which directly translates into higher power consumption and heat radiation. Traditional cooling mechanisms are not able to manage this heat increase and aggressive refrigeration alternatives are beginning to be used. Also, packaging materials are not designed to work under these operating conditions [12].

There is a strong relationship between power consumption and fault probability. High temperature in a chip causes glitches, races and increases frequency of errors. At the same time, superscalar processors require parallel functional units and significant resources in terms of the number of transistors, increasing overall power consumption. According to [6], superscalar architectures that are capable of issuing multiple instructions increase their power consumption at a higher rate than the increase in performance when compared to conventional single issue RISC architecture.

Reduction in feature size and increase in the number of devices integrated on a similarly sized die have made the leakage current one of the most significant sources of power consumption. Most of the increase is due to very low threshold voltage devices having significantly increased subthreshold currents that charge and discharge static loads. Many architectural and system level techniques have been described to reduce energy dissipation by gating the clock tree to unused functional units, but not all the power hungry CPU units have been efficiently managed.

As will be mentioned in the next section, the register file consumes a sizable fraction of the total power in embedded processors and becomes a dominant source of energy dissipation when other power saving mechanisms have been used. Many recent compiler optimization techniques increase the register pressure, and there is a current trend towards implementing larger register files. Both of these factors increase power consumption in this unit, and new techniques to manage this situation are needed. Putting those registers unused by a code section into a low-power state ("drowsy" state) and gating the clock to these unused registers, both static and dynamic energy consumption significantly reduces. In this paper, we propose a mechanism for power-aware register file reconfiguration based on compiler support and, possibly, code profiling. With a minor change in the ISA (*Instruction Set Architecture*), the compiler dynamically selects the most suitable register file configuration for application requirements.

The remainder of the paper is organized as follows: section 2 discusses the sources of power dissipation, section 3 describes related work in this area, section 4 presents our approaches in the register file reconfiguration. Finally, section 5 outlines the compilation environment , section 6 shows the experimental methodology and results and several conclusions are drawn in section 7.

2. SOURCES OF POWER DISSIPATION

In order to achieve significant savings in power aware architectures, it is necessary to select those devices and modules that have the greatest impact on power consumption. Devices with high switching capacitance, or devices that, for any reason, are continuously working, are the *hot points* for these policies. Memories are primary example of the former, while clock is an example of the latter.

Clock power consumption reaches 32% of total power consumption in Alpha 21364, and for previous generation Alpha microprocessors this amount was nearly 40%. The reasons behind this fact are the long clock tree and the undesired switching activity of the devices driven by this network. Memory devices represent a large area in current microprocessors. These devices are not only the on-chip main memory and caches, but also the register file, the instruction and load-store queues, the re-order buffer and the branch prediction tables. 15% of power consumption in Alpha 21364 comes from the 128 kB of on-chip cache [7].

Duplicate resources in current microprocessors are a non-negligible source of power consumption. Integer and floating execution units add up to 20% of total power consumption in Alpha 21364 but rarely all of them are working[1]. These over-sized resources allow high performance in superscalar architectures, but they also impose a high penalty on power consumption.

A multi-ported register file is also a large source of power dissipation due to its increasing size, frequency of access and continuous operation cycle. Register file power consumption in embedded processors reaches 25% [3] when running average applications, and future processors will likely exhibit a more power-hungry behavior. This amount is even higher when the associated clock tree is taken into account.

This paper focuses on the register file in embedded processor. The register file would dominate energy consumption if other power-saving techniques are applied to the rest of the processor but nothing is done in the register file.

[1]Depending on program resource requests

3. RELATED WORK

There has been a lot of interesting work on power oriented resource management in microprocessors. Iyer et al. [9] propose a hardware approach to adjust the RUU (*Register Update Unit*) size and effective pipeline width to the program requirements, saving a considerable amount of energy. This work is based on a previous one where Merten et al. [15] propose a hardware scheme to detect *hot spots*[2] in running programs. Maro et al. [14] followed an approach similar to ours to capitalize on this phenomenon of underutilization of resources. However, though they deal with reduction in power consumption, they focus on monitoring IPC variation and do not evaluate alternative power reduction approaches or model the power consumption in the power management logic. Compiler optimization mechanisms have been proposed to reduce power consumption in microprocessors [10], but these approaches only work on code optimizations and do not use hardware support. Also, there are solutions based on code versioning and selection by the compiler using heuristics and profile data [2]. Finally, there are also several software approaches based on code profiling and code annotation [3], operating system management [18], and circuit level optimizations on the register file [20].

The work presented in this paper differs by proposing an energy management of the register file driven by the compiler. The design of the new register file storage cell and related management policies reduce power and performance penalty in average execution.

4. REGISTER FILE RECONFIGURATION

4.1 Background

Current microprocessors are designed to allow the parallel execution of independent instructions by allocating source operands in different architectural registers and providing multiple functional units, the lack of parallelism in source code, the locality of data and the number of available resources make the architectural register file underutilized most of the time. Many registers are thus consuming power while not being used.

Large register files present several advantages: decreased power consumption in the memory hierarchy (cache and main memory) by eliminating accesses, improved performance by decreasing path length and memory traffic. Previous work showed that existing compiler technology could not make effective use of a large number of registers [13], and the industry followed this statement implementing the majority of processors with 32 or fewer registers. However, current research on this area [16] and the efforts on optimizing spill code indicate a need for more registers. Sophisticated compiler optimizations can increase the number of variables and the register pressure; for instance, global vari-

[2]A collection of frequently executing basic blocks

able register allocation can increase the number of registers that are required.

The total number of registers in recent architectures (IA-64) has grown larger. The number of registers, the increased register pressure and the aggressive compiler optimizations (by allocating global variables, promoting aliased variables and inlining) lead to increase in power consumption.

Many of the power reduction policies work on gating the clock to functional units and architectural modules, but the register file is always clocked. The power management policies disable other power hungry devices and as a result the register file becomes a dominant factor in power consumption.

The main objective of our work is power reduction in embedded processors. Embedded processors usually use *in-order* architectures because of power, area and cost constraints (e.g. ARM and MIPS processors).

4.2 Approach

When a section of code does not use all the registers in the register file, it should be possible to use the accessed registers and turn off the unused ones. Turning off these unused registers will lead to high energy savings due to the static power consumption decrease and due to the possibility of gating the clock to these devices. When code using the turned off registers is reached they are turned back on.

To power down the unused registers one has to prevent the information from being lost. Registers that are not used can be turned into a low-power mode which keeps the register contents but reduces static power consumption to a minimum. A number of previous results try to reduce leakage power consumption in cache memories by gating V_{dd} [17] or by reducing voltage source with DVS (Dynamic Voltage Scaling) techniques. The main difference between these approaches and the objective that has to be satisfied in the register file case is that the content of the cache line can be lost when it is turned into the low-power mode, and then reloaded from main memory or L2 cache. Recent research on this topic has lead to the design of a memory cell that can be turned into a low-power mode ("drowsy" as opposed to "sleep") where power consumption is almost negligible but the contents are preserved [5]. Extending these ideas to the register file architecture, leads to a drowsy register file cell like the one presented in Figure 1. The area overhead of this approach is similar to the results showed by Flauter et al. (less than 3%), but the time overhead would be smaller due to simpler access to the register file as compared to the cache (less than 1% on average).

The proposed solution to deal with the "scalable" register file works as following. Instead of working with the entire register file, only the currently used registers are addressable and maintained on. The rest of the registers are put into a drowsy state where the static power consumption is reduced to a minimum and the clock distribution network is gated as well.

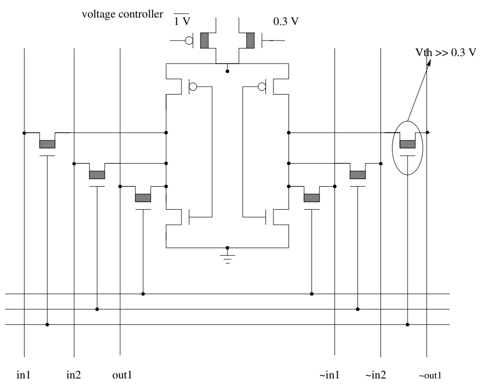

Figure 1: A drowsy register file cell

It is assumed that the ISA is augmented with 2 instructions to turn a set of registers on and off. The compiler is responsible for using these instructions to manage the register file. It selects a section of code where N registers are used and inserts an instruction to turn others registers off.

This register file management technique requires a code generation phase that marks the code sections where the register file reconfiguration should be performed. This phase has to detect those frequently executed code sections with reduced number of required registers and mark the beginning and end of such code. The run-time management and the compiler approach are described below.

While it is theoretically possible to turn individual registers on and off, such an approach would require very sophisticated compiler analysis, have high (additional instructions) overhead, and likely reduce performance. Instead we limit the number of registers used at any time to either 8/16 or 32.

5. COMPILATION

The register file management policy that has been presented needs few ISA modifications. It needs a signal mechanism to mark the beginning and the end of a code section. This signal mechanism is performed by incorporating a new instruction that signals when turning registers on and off has to be performed. After register allocation, the compiler knows exactly which registers are used by instructions, and can specify the register requirements per function (or loop inside the function if a lower granularity is used) in a new instruction that turns the unused registers into the drowsy state. After this optimized portion of code, the same instruction turns on the unused registers. ARM Thumb uses a similar approach for changing the execution context to a second ISA.

The main objective of the compiler approach is to select the optimum configuration for the register file satisfying one important constraint: not to increase the code size by inserting new instructions. A similar problem arises in the area of code-size reduction techniques. One approach used for reducing code size employs a dual instruction set, one of them with shorter operation code and limited number of accessible registers. Obviously, the processor architecture has to support this dual instruction set [8].

A mode instruction is used to switch between the two sets. We use a similar instruction to alter register file configuration.

Several RISC processors for embedded systems implement this functionality: ARM7TDMI from ARM Inc., ST100 Core from ST Microelectronics or Tangent-A5 from ARC. One example of this is the 16 bit Thumb instruction set in ARM processor core. By using the Thumb instruction set it is possible to obtain significant reductions in code size in comparison with the ARM code. As a result of the reduction in code size, the instruction cache energy expended in Thumb mode is also significantly lower. However, many times this reduction in code size is obtained at the expense of a significant increase in the number of instructions executed by the program [11]. Also, these architectures do not employ any power saving mechanism with the unused registers in the register file.

As has been previously discussed, this approach requires a compilation support that selects the most suitable configuration for the smaller register file and marks when it has to be used after a code analysis. Profiling information can also be used to improve the frequency estimates.

The compiler approach would involve the following tasks:

1. a first phase in which the compiler marks the sections of code that can be optimized in register usage (functions, loops or code blocks which are supposed to be frequently executed)
2. a second phase in which the profitability of reconfiguring the register file is analyzed
3. and a last phase that inserts the marking instructions at the beginning and the end of selected code sections.

There is a tradeoff in the granularity and the obtained energy savings, and the compiler has to analyze the options in order to obtain optimal results. Higher granularity is usually translated into higher energy savings because the portion of code for which the register file size is optimized is also larger. This reduces the overhead of turning registers on and off. Therefore, a function-level granularity should lead to the best results in power consumption. However, this is not true because the whole function may require more registers than are available. In such a case, reducing the granularity level could help finding time-consuming loops inside the function that can be mapped onto a smaller number of registers. To achieve these, a profiling phase that characterizes the execution frequency in the code can be used. This is the approach that has been followed in the experiments described below.

6. EXPERIMENTAL METHODOLOGY

The approach presented in this paper was modeled via simulation The simulator used in this work is derived from the SimpleScalar tool suite [1]. SimpleScalar is an execution-driven simulator that implements a derivative of the MIPS-IV instruction set. The version used in our experiments is 3.0c. We have included in the simulator the following enhancements:

- Register file management architecture
- Register file reconfiguration by code annotation

The power models used to validate the ideas come from Wattch [4], an extension of the SimpleScalar simulator capable of estimating power consumption at the architectural level. Wattch has also been modified to take into account our reconfiguration policies and the overhead from the new reconfiguration logic. CACTI [19] models were used for the register file, parameterized to work with the run-time reconfiguration.

The baseline configuration of the processor used for our tests is given in table 1.

Baseline architecture	
Clock	600 MHz
Data Width	32 bits
Scheduling	in-order
Decode width	2 insts/cycle
Issue width	2 insts/cycle
Commit width	2 insts/cycle
Functional Units	1 integer ALU
	1 integer multiply/divide unit
	1 FP ALU
	1 FP multiply/divide unit
Register File	32 registers
Two bank approach	
Register File	
bigger bank	32 registers
smaller bank	{ 16 , 8 } registers

Table 1: Baseline processor configuration

We performed experiments using programs from Spec2000, MediaBench and MiBench and for two configurations of the register file: 16 and 8 registers used at a time. The code sections where the register file is reconfigured are selected using the profiling tool *gprof* and use a *function-level* granularity. For every benchmark, only the most time consuming function was optimized.

Granularity is an important issue in these experiments. As a first approach and in order to keep consistency between experiments, a function level granularity is selected. However, some cases require a more careful management. These are cases when the whole function cannot be mapped into the available number of registers. If the most time consuming function that is considered for reconfiguration contains any time consuming loop, reducing the granularity to the loop-level and performing the register file reconfiguration in this portion of code can allow to use a smaller number of registers and saves an extra amount of energy.

6.1 Results

Figure 2 shows a reduction in the total energy consumption in the register file for the two register configurations, 8 and

16 registers total. As can be observed in the graph, total energy consumption in the register file has been reduced by more than 40%, on average. Even higher power savings can be expected if the reconfiguration policy was not restricted to only the most time consuming function. Profiling shows that 90% of execution time is expended in 2 or 3 functions.

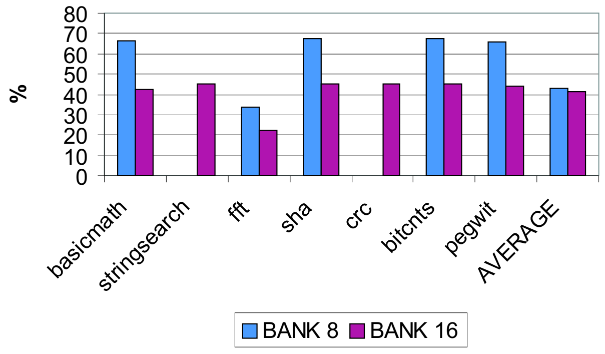

Figure 2: Reduction in the total energy for two register configurations

Two of the benchmarks in Figure 2, *crc32* and *stringsearch*, cannot be executed with only 8 registers. They require the 16-register configuration. They are the main reason why 8- and 16-register configuration averages are the same, and they are good candidates for a lower granularity compilation approach.

Applying loop-level granularity to *crc* and *stringsearch* leads to further energy saving. Figure 3 compares the total energy savings for the benchmarks and different granularity levels. If the register file reconfiguration could be performed during the whole program execution (from the beginning to the end of the main function) the highest energy savings would be obtained. However, the number of registers required by the applications cannot fit in the smaller configurations, and no energy decrease is achieved. The most time consuming functions in these benchmarks (*main* and *strlen*) can only fit into a 16-register configuration and a lower energy is obtained. If granularity is further reduced and the reconfiguration is performed in the dominant loop inside these functions, the smallest configuration for the register file can be used, and the energy reduction is significantly increased.

After this new granularity selection, greater reduction in the total energy consumption is achieved for the 8 register configuration. Figure 4 shows the average power savings when a lower granularity level is selected. Up to 60% reduction for the 8 register configuration is achieved.

7. CONCLUSIONS

After clock gating and power-aware system level policies have been applied for power minimization in embedded systems, energy consumption in the register file becomes dominant. Current sophisticated compiler optimizations require larger register files and increase the register pressure. In this paper, a power-aware reconfiguration of the register file driven by a compiler is presented. The lack of parallelism in

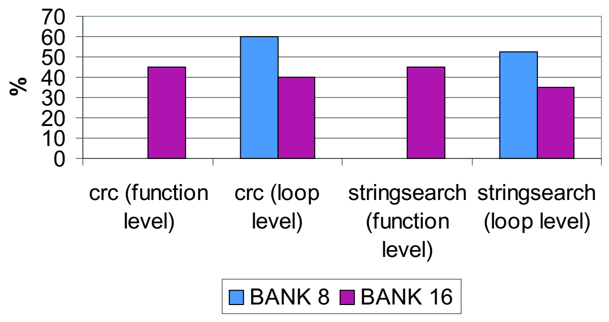

Figure 3: Effect of granularity selection in *crc32* and *stringsearch* benchmarks

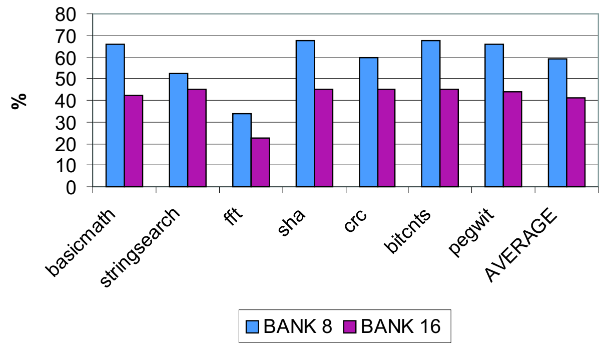

Figure 4: Reduction in the total energy with lower level granularity

the source code, the locality of data and the locality in the accesses to the register file leave most of the registers unused at any given time. The mechanism described in this paper marks sections of code where this behavior can be exploited. Turning the unused registers into a low-power state in these code sections leads to significant total energy savings.

Overall, experimental results collected from selected SPEC, MediaBench and MiBench benchmarks show a 60% total energy reduction in the register file when the most time-consuming function is selected to perform the reconfiguration of the register file. In many cases, these results can be improved if a lower granularity in the code section is selected by the compiler. The increase in the execution time is negligible.

The main contributions of the paper can be summarized as follows. It has been shown how energy can be saved in embedded processors with a "gated" register file. A power-aware reconfiguration policy to fit the register file size to the program requirements has been presented. Finally, a compiler environment to perform these tasks has been outlined and the influence of granularity of optimizations has been analyzed.

The work on the compiler algorithms and granularity selection is still in progress. Higher overall energy savings are expected when this work is completed because analysis and tradeoffs will be performed over the entire programs.

8. REFERENCES

[1] T. Austin, E. Larson, and Dan Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. *Computer*, 35(2):59–67, February 2002.

[2] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau. Architectural and Compiler Strategies for Dynamic Power Management in the COPPER Project. In *IWIA*, 2001.

[3] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau. Profile-based Dynamic Voltage Scheduling using Program Checkpoints in the COPPER Framework. In *DATE*, 2002.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a Framework for Architectural-level Power Analysis and Optimizations. In *ISCA*, 2000.

[5] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and Mudge T. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *ISCA*, 2002.

[6] R. Gonzalez and M. Horowitz. Energy Dissipation in General Purpose Microprocessors. *IEEE Journal of Solid-State Circuits*, 9(21):1277–1284, September 1996.

[7] M. K. Gowan, L. L. Biro, and D. B. Jackson. Power Considerations in the Design of the Alpha 21264 Microprocessor. In *DAC*, 1998.

[8] A. Halambi, A. Shrivastava, P. Biswas, N. Dutt, and A. Nicolau. An Efficient Compiler Technique for Code Size Reduction using Reduced Bit-width ISAs. In *DATE*, 2002.

[9] A. Iyer and D. Marculescu. Power Aware Microarchitecture Resource Scaling. In *DATE*, 2001.

[10] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye. Influence of Compiler Optimizations on System Power. In *DAC*, 2000.

[11] A. Krishnaswamy and R. Gupta. Profile Guided Selection of ARM and Thumb Instructions. In *ACM SIGPLAN*, 2002.

[12] S.-W. Lee and J.-L. Gaudiot. Power Considerations in the Design of High Performance Multi-Threaded Architectures. Technical report, Department of Electrical Engineering, University of Southern California, 2000.

[13] S. A. Mahlke, W. Y. Chen, P. P. Chang, and W. Hwu. Scalar Program Performance on Multiple-Instruction Issue Processors with a Limited Number of Registers. In *Hawaii International Conference on System Sciences*, pages 34–44, 1992.

[14] R. Maro, Y. Bai, and R.I. Bahar. Dynamically Reconfiguring Processor Resources to Reduce Power Consumption in High-Performance Processors. In *Workshop on Power Aware Computing Systems*, 2000.

[15] M. C. Merten, A. R. Trick, C. N. George, J. C. Gyllenhaal, and W. W. Hwu. Hardware-Driven Profiling Scheme for Identifying Program Hot Spots to Support Runtime Optimization. In *ISCA*, 1999.

[16] M. Postiff, D. Greene, and T. Mudge. The Need for Large Register File in Integer Codes. Technical Report CSE-TR-434-00, Electrical Engineering and Computer Science Department. The University of Michigan (USA), 2000.

[17] M. Powell, S. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-V_{dd}: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories. In *ISLPED*, 2000.

[18] G. Savransky, R. Ronen, and A. Gonzalez. Lazy Retirement: A Power Aware Register Management Mechanism. In *Workshop on Complexity Efficient Design*, 2002.

[19] S. J. E. Wilton and N. P. Jouppi. An Enhanced Access and Cycle Time Model for On-Chip Caches. Technical Report 93/5, Computer Science Department. University of Wisconsin-Madison (USA), 1994.

[20] J. Zalamea, J. Llosa, E. Ayguadé, and M. Valero. Software and Hardware Techniques to Optimize Register File Utilization in VLIW Architectures. In *International Workshop on Advanced Compiler Technology*, 2001.