Energy Aware Register File Implementation through Instruction Predecode[*]

José L. Ayala[†], Marisa López-Vallejo[†], Alexander Veidenbaum[‡], Carlos A. López[†]

[†]Departamento de Ingeniería Electrónica
Universidad Politécnica de Madrid (Spain)
{jayala,marisa,barrio}@die.upm.es

[‡]Center for Embedded Computer Systems
University of California, Irvine (USA)
alexv@ics.uci.edu

Abstract

The register file is a power-hungry device in modern architectures. Current research on compiler technology and computer architectures encourages the implementation of larger devices to feed multiple data paths and to store global variables. However, low power techniques are not able to appreciably reduce power consumption in this device without a time penalty. This paper introduces an efficient hardware approach to reduce the register file energy consumption by turning unused registers into a low power state. Bypassing the register fields of the fetch instruction to the decode stage allows the identification of registers required by the current instruction (instruction predecode) and allows the control logic to turn them back on. They are put into the low-power state after the instruction use. This technique achieves an 85% energy reduction with no performance penalty. The simplicity of the approach makes it an effective low-power technique for embedded processors.

1 Introduction

Continuing advances in semiconductor technology have allowed dramatic performance gains for general-purpose microprocessors and embedded systems. These improvements are due both to increasing clock rates as well as to advanced support for exploiting instruction-level parallelism and memory locality using the additional transistors available in each process generation. However, as a negative consequence, this causes a significant increase in power dissipation, due to the fact that the dynamic power is proportional to both clock frequency and to switching capacitance (which increases as more devices and on-chip components are included). Thus, despite continuous attempts to reduce voltages and to design lower power circuits, power dissipation levels have steadily increased with each new microprocessor generation [10]. Moreover, a new problem arises because the power savings achievable with low level techniques are reaching their theoretical maximum.

Improvements in integrated circuit fabrication technology have enabled the number of transistors in microprocessors to more than double with every generation. At the same time, leakage current also increases with each technology generation. Thus, the energy consumption of memory structures (main memory unit, caches, register banks, etc.) will increase dramatically with future process technologies.

[*]This work was supported by the Spanish Ministry of Science and Technology under contract TIC2000-0583-C02-02.

Processor	**Integer RF**	**Floating RF**
ARM	32	8
Power PC	32	32
XScale	32	32
Transmeta Crusoe	64	64

Table 1. Register file size

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 applied. Register file power consumption in embedded systems depends very much on system configuration, mainly on the number of integrated registers, cache size and existence of a branch predictor table (i.e. depends on the relative size of other memory devices). In the Motorola's M.CORE architecture, the register file energy consumption could achieve 16% of the total processor power and 42% of the data path power [9]. In out-of-order processors with a large number of physical registers which are implemented as part of the *Re-Order Buffer* (in Pentium III, for example), this structure dissipates as much as 27% of total energy according to some estimates [8].

Many recent compiler optimization techniques increase the register pressure, and there is a current trend towards implementing larger register files. Large register files present several advantages: decreased power consumption in memory hierarchy (cache and main memory) by eliminating accesses, improved performance by decreasing path length and memory traffic by removing load and store operations. Early work showed that the existing compiler technology could not make effective use of a large number of registers [15], and the industry followed this statement implementing most of processors with 32 or fewer registers. However, current research in this area [16] and the efforts on optimizing spill code indicate a need for more registers. Sophisticated optimizations can increase the number of variables and the register pressure; furthermore, global variable register allocation can increase the number of required registers. The number of registers in recent architectures has grown considerably. The number of registers, the register pressure and the aggressive compiler optimizations (by allocating global variables, promoting aliased variables and inlining) will lead to increase the energy consumption.

Table 1 shows the register file size for several modern processors clearly showing significant increase in size. This motivates the need for efficient techniques to reduce register file energy consumption in embedded systems. This paper introduces a new technique which is characterized by an absence of performance penalty. It is based on the observation that a register is only used when an instruction reads from it or writes to it, the register is "idle" at all other times. By keeping the idle registers in a low power (or "drowsy") state a significant amount of energy can be saved. Most registers are idle in any given cycle since at most three registers are accessed by an issued instruction. The architectural modifications we propose in this paper allow the drowsy registers to be turned back to the "active" state as the instruction accesses them.

The rest of this paper is organized as follows. Section 2 summarizes previous research on this and other related topics. Section 3 presents the proposed approach and introduces the modified register file and the energy saving technique. Finally, section 4 presents the experimental setup and shows the resulting energy savings; some conclusions are drawn in section 5.

2 Related Work

Low power techniques can be applied at different abstraction levels: from technology and circuit levels to system and algorithm levels. Low level power optimization techniques are achieving their theoretical maximum which calls for investigating approaches at higher abstraction levels. Energy optimization and estimation at the system level is an area of very active research. At the system level, the sources of energy dissipation can be grouped under three broad categories: (i) processing units; (ii) memories; (iii) interconnects and communication [6]. Several techniques address more than one category, while others are more narrowly focused. The register file can be considered as a component of the memory hierarchy and, therefore, energy reduction techniques for memories can be applied with some modifications.

Several approaches focus on creatively exploiting caching to reduce energy consumption. They use a new structure (often called "cache buffer" or L0 cache) [13] or apply compilation and data type optimizations to reduce memory accesses [19]. Alternatively, architectural modifications to the memory cell and both address decoders and bitline drivers have been previously proposed by some authors [20]. All of these approaches try to exploit simple data locality, which may not be sufficient for appreciable energy reduction. Moreover, the extra hardware required is not justified by the energy savings obtained.

Other techniques target more general memory and register file hierarchy, where caches as well as various types of memories (SRAMs, DRAMs) are available. In these architectures, data transfer and placement are tightly controlled to minimize power consumption per memory access [12]. Also, hardware reconfiguration policies, based on power aware compiler support, have been studied [3]. These approaches do not always obtain high energy savings due to the coarse granularity level at which the compiler has to work and may suffer a performance penalty.

Solutions based on code versioning and selection by the compiler which use heuristics and profile data [4] have also been proposed. Finally, there are also several software approaches based on code profiling and code annotation [5], operating system management [17], and circuit level optimizations on the register file [21]. None of these techniques have been able to obtain appreciable energy savings in the register file without a time penalty or an increased cache pressure.

3 A New Approach

3.1 Register File Design Overview

In a typical configuration, the register file is an array of N words by M bits. Any of the N words can be simultaneously accessed by two read ports and a single write port. A block diagram of the register file, shown in Figure 1, shows that the register file contains seven distinct types of functional blocks [18]. These are the memory cell array, the read address decoders and word line drivers, the write address decoder, the bit line drivers, the sense amplifiers, the output latches, and the comparators.

The memory cell array stores the bits of data. It is arranged in a grid of N rows by M columns of memory cells. When any of the ports accesses the memory cell array, the read or write operation is performed on every memory cell in the selected row simultaneously. Each read address decoder is responsible for decoding a $log_2 N$-bit address to determine

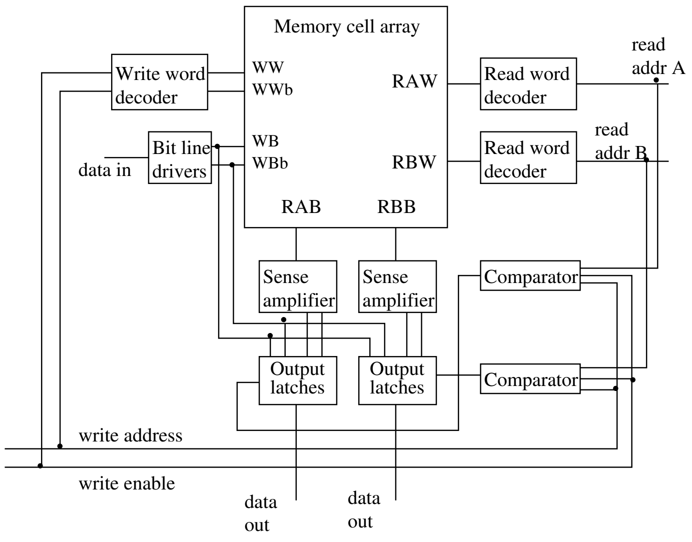

Figure 1. Register file block diagram

which of the rows is selected for each read operation. The read word line drivers are responsible for driving the read word lines accordingly. The write address decoder selects the row to be written. The write word line drivers drive the write word lines while the bit line drivers provide input data to the memory cells. The independent read and write address decoders allow parallel access to up to three register operands of an instruction. The rest of the register file components and operation are not described in this paper due to space limitations.

3.2 Background on Energy Reduction

It has been shown that current processors have a large register file. Nevertheless, this device is underutilized most of the time. Embedded processors are rapidly becoming wide issue and speculative. The register file is one of the units which limits the clock frequency in wide issue processors. With i instructions or micro-operations issued per cycle each consuming (up to) two operands and producing (up to) one result, the register file needs to support $2i$ reads and i writes per cycle. High clock frequencies require deeper pipelines combined with wide instruction issue and increased depth of speculation call for an even larger number of physical registers. Unfortunately the silicon area, the energy consumption and the access time of the register file are proportional to the number of write and read ports and the total number of physical registers.

The $2i+i$ registers have to be on to provide the source operands and store the results in a given cycle. But the rest of the N registers do not need to be on and are unused but energy-hungry resources. They are not being read or written by any instruction and should be turned into a low-power state.

Turning memory devices into a low-power state is not a brand new idea. Previous research has focused on turning off unused memory banks (or other resources) by gating the power source. When the objective is a memory device, the cost of recovering the lost information could hide any power saving or, at least, represent a very significant time penalty[1]. When working with the register file, there is no way to recover data from memory

[1]For example, to load the data from main memory.

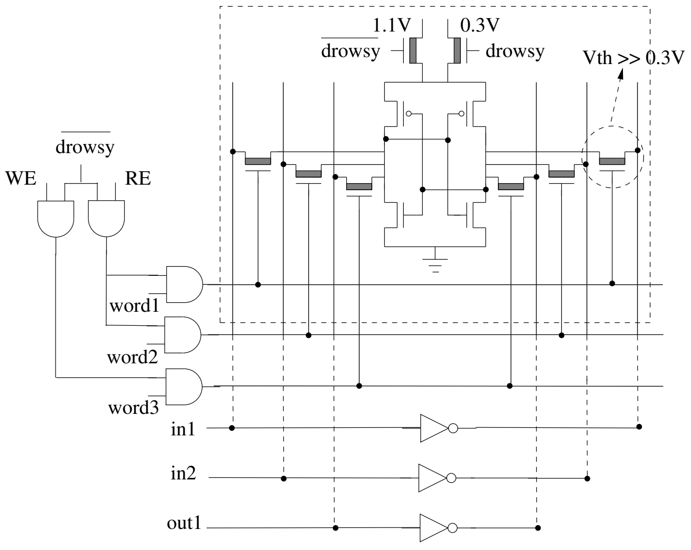

Figure 2. Register file cell

without extra accesses to the cache, and something has to be done in the unused registers to prevent the information from being lost. Turning these unused registers into a low-power state (*drowsy* state) the power consumption can be reduced to a minimum without data lost [11].

The information in a memory cell is preserved while it is in the drowsy state. However, the data line must be restored to a high-energy mode before its contents can be accessed. One circuit technique for implementing drowsy memory devices is adaptive body-biasing with multi-threshold CMOS (ABB-MTCMOS), where the threshold voltage is increased dynamically to yield reduction in leakage energy. This leakage reduction technique requires that the voltages of the N-well and of the power and ground supply rails are changed whenever the circuit enters or exits the drowsy mode. Since the N-well capacitance of the PMOS devices is quite significant, this increases the energy required to switch the memory cell to high-power mode and can also significantly increase the time needed to transition to/from drowsy mode. A more efficient approach to achieve the drowsy state is proposed by Flautner et al. [7], where a dynamic voltage scaling (DVS) technique is exploited to reduce static power consumption. Due to short-channel effects in deep-submicron processes, leakage current is significantly reduced with voltage scaling. The combined effect of reduced leakage current and voltage yields a dramatic reduction in leakage power. This is the solution used in our approach to reduce energy consumption.

3.3 System Design

Figure 2 shows the modified register file cell used to support the drowsy state. As can be observed, the dual power supply is switched to low V_{DD} when the cell is in drowsy state. It is necessary to use $high - V_{th}$ devices as pass transistors because the voltage on bit lines could destroy the cell contents. Before a register cell can be accessed, the power supply has to be switched to high V_{DD} to restore the contents and allow the access. An extra read_enable/write_enable gating circuit assures the memory cell is not accessed (read/written) while being in drowsy state.

Due to the necessity of restoring the register contents previously to the access, the

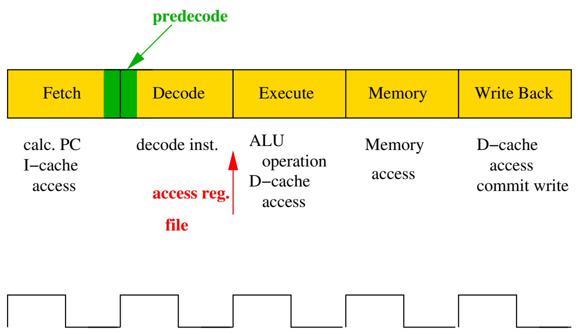

Figure 3. Pipeline configuration

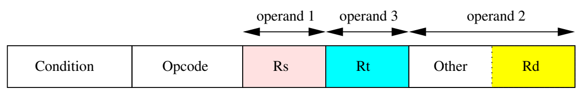

Figure 4. General instruction format

accessed registers must be known at least one cycle before the access happens.

In a typical pipeline organization (Figure 3) the register read access occurs during the same clock cycle as the instruction decode. Therefore, the register power supply must be switched to high V_{DD} prior to this phase: the fetch cycle (we are assuming no renaming phase in an embedded in-order processor). In some pipelines, e.g. *ARM10*, there is a prefetch cycle which allows to detect branch instructions ahead of the fetch stage, predict those branches that are likely to be taken or remove those branches that are not likely to be taken [1]. However, this stage is only used when a new instruction has to be reloaded from memory instead of I-cache, and it may not be available in all architectures.

During the fetch stage the instruction is loaded from I-cache into the instruction register. Our approach takes advantage of this fact by forwarding the operand fields of the instruction to the register file address decoders. This is possible due to the fixed instruction format used by RISC processors shown in Figure 4. The register designators are in a fixed position and can be easily extracted for the current instruction. This intermediate step can be performed in the *issue* stage of the pipeline. If the issue stage is not present in the processor pipeline, the time required to complete this simple decoding can be overlapped with the delay between the availability of the cache data and the completion of the tag comparison.

In order to reduce the complexity of the additional logic and to assure the register identification in the shortest time possible (and before the start of the next clock cycle), the register predecode cannot make use of any memory access to help the decoding. Therefore, in our approach only the address decoders present in the register file architecture are used. The delay of these decoders is small enough to identify the accessed registers before the end of a clock cycle and, therefore, to turn these on before the following access.

The current executed instruction may not correspond to the general instruction format presented in Figure 4, i.e. may not include the three register fields, or these fields may have a different meaning (memory address or immediate operand), but this is only known when the instruction decode has been accomplished (unless predecoded info is stored in the I-cache). Therefore, the predecode logic will be fitted to the general instruction format (three operand registers) and, when the instruction shows a different configuration the

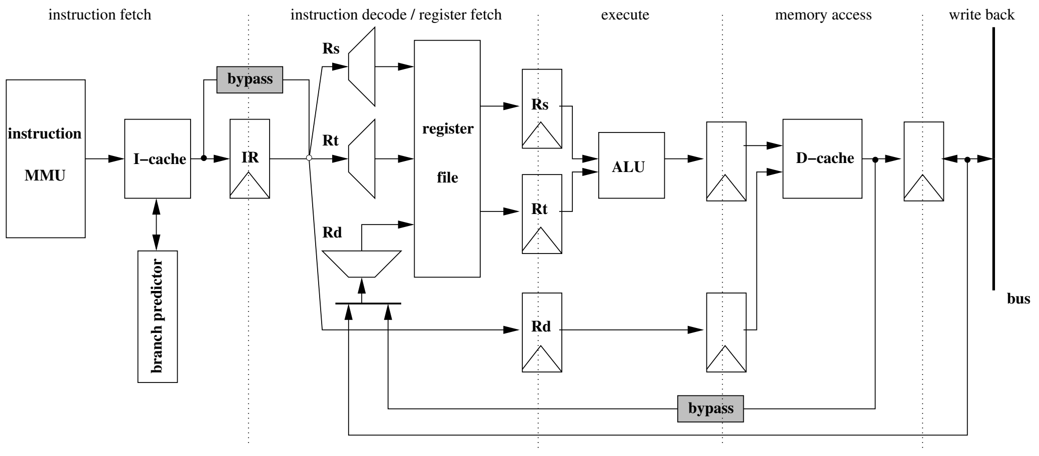

Figure 5. Schematic of the general architecture

effect will be the awakeness of registers that are not used. Consequently, several registers that could be kept in the low power state are activated and extra power is consumed. This effect will be analyzed in detail in section 4, let us say for now that the worst case involves waking up three registers when none is used. The overall energy savings from our approach are still significant.

So far we have discussed dealing with register reads. The register write access happens after the *memory* stage by which time the write register designator is known in any case. Thus the register to be written can be turned on after the execute or memory stage (depending on pipeline structure) and written in the write-back stage.

Figure 5 represents the general architecture of the processor supporting the proposed power aware register file. The bypasses required by our technique have been represented shadowed.

As can be noticed, the main advantages of this approach are the no-time penalty on the predecode stage, the simplicity of the extra logic (only a few modifications to the memory cell, the gating circuit and the bypass network) that means small energy and area overhead and, finally, the possibility of applying this approach to different microprocessor architectures present in many embedded systems. The downside of the approach is that the extra dynamic consumption (wake-up cost) has to be amortized over a small number of accesses. Flautner et al. have estimated this value to be around seven accesses. Taking into account the number of drowsy registers per instruction in the register file (all of them but the three accessed and the control registers), this cost has not a big impact on power savings.

4 Experimental Results

The architecture described in previous paragraphs has been validated with a simulator derived from the SimpleScalar tool suite [2]. 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. The compiler/architecture reserves four registers for special use. These are the stack pointer, the zero register, the return address register and the return value register. These registers are going to be left in the active state permanently.

The baseline system models a typical architecture of a high-performance embedded

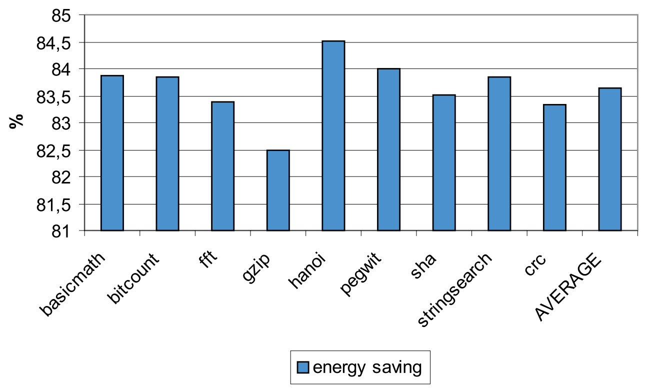

Figure 6. Register file energy savings

system. It is then modified to support the power aware register file. Table 2 presents the main characteristics of the baseline embedded architecture.

Baseline architecture	
Pipeline	in-order
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
Fetch 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 + 32 registers

Table 2. Baseline processor configuration

Figure 6 shows the energy savings obtained in the register file of the modified architecture running programs from SPEC2000, MediaBench, and MiBench benchmark suites. All results are normalized with respect to the register file power consumption in the baseline architecture. In these experiments it has been assumed that the energy consumption of a drowsy register is negligible. This assumption is based on the results published by Flautner et al., who show a 2% of energy consumption during the drowsy state for a cache memory cell. Also, area and energy overhead due to the extra logic (gating circuit) is completely negligible and has not been considered in the results.

As can be seen, the register file power consumption is reduced by nearly 84%, on average. The results are very similar for all benchmarks because the number of active registers per clock cycle is always the four reserved registers and up to three register operands. The slight differences between benchmarks are due to the variations in the number of register file accesses and instruction type distribution.

It should be noted that for out-of-order systems which have a large number of physical registers (80 in the case of Alpha 21264) even higher total energy savings are expected. In addition, the register file may be implemented as part of the Reorder Buffer (ROB) in which case the energy savings are still higher. In this case, turning physical registers into

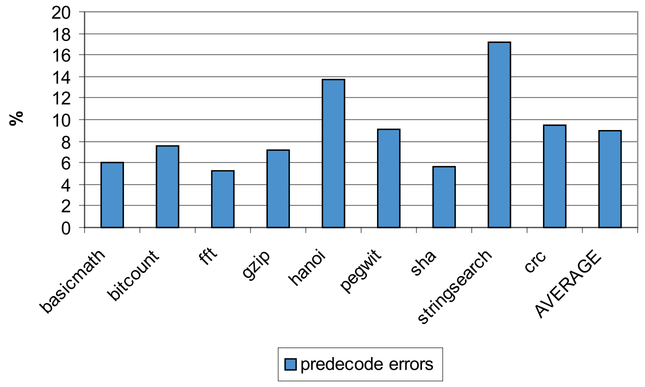

Figure 7. Erroneous register awakeness (MIPS-IV)

the drowsy state also allows the rest of the ROB to be more energy efficient by turning off unused memory cells and ports [14].

4.1 Accuracy of the Approach

As was explained in section 3.3, the predecode logic assumes that a three register operand instruction is always executed, what could not be true. In order to analyze how far is our approximation to the real execution, the following analysis has been performed. If no register operand is required by any instruction, the register file power savings would go up to $\frac{60}{64} = 94\%$ (4 awake reserved registers per instruction). This is the theoretical maximum we have on power saving. Figure 7 shows the number of erroneous register awakeness normalized to the total number of required registers during program execution. A 100% value in this graph would mean that every instruction requires no register operand. However, on average only a 9% of registers are erroneously awaken, what means that the maximum power savings expected in the register file would be less than 88% on average, only 1% higher[2] than our approximation.

This behavior will vary slightly depending on the processor and the corresponding ISA (*Instruction Set Architecture*), but is expected to be similar for other machine configurations. Figure 8 shows comparable results for an Alpha 21264 processor. Even for the extended ISA of this high-performance processor our approach achieves results that are quite near to the MIPS-IV case.

Now, collected stats show how a 20% of the awake registers could have been avoided, keeping these registers into the drowsy state and saving a little more extra energy than our approach (2% for 32 integer and 32 floating-point registers).

5 Conclusions

The register file is a power hungry device in modern embedded processors. At the same time, current research on compiler technology and computer architecture encourages the implementation of an increasing number of registers. Thus, the register file power consumption becomes a significant factor. The number of registers actually used per

[2] $(94 - 83.6) \cdot \frac{9}{100} \approx 1\%$

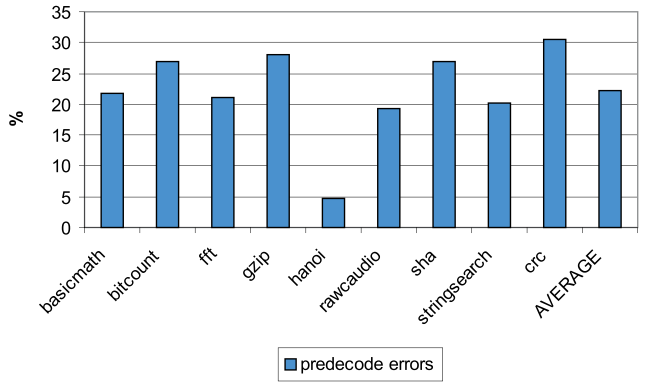

Figure 8. Erroneous register awakeness (Alpha 21264)

instruction is very small: less than or equal to three. This fact is exploited by the technique proposed in this paper to save power by turning the unused registers into a low-power state. They are then awakened before being accessed by an instruction.

An efficient hardware mechanism to turn the unused registers into a low power state has been described in this paper. This is performed at the micro-architecture level for each instruction. A DVS technique is used to keep the information stored in the registers while reducing the power consumption to a minimum. The required extra logic has been simplified to reduce the register specifier predecode time.

The proposed technique reduces the register file power consumption by 84%, on average, when running benchmarks on embedded processors. Assuming all instructions use three register operands results in 9% of extra register wake-ups over the theoretical minimum. Therefore, the optimal case where only the required registers would be awakened would only save an additional 1% over the proposed approach. The presented technique has no performance penalty and the area/energy overhead is negligible.

The simplicity of the approach and the minimal additional logic required makes it an effective low-power technique for embedded processors. Future work will apply the approach to the ROB in out-of-order processors to reduce its energy consumption.

References

[1] ARM. *ARM10022E: Technical Reference Manual*, 2001.

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

[3] J. L. Ayala and A. Veidenbaum. Reducing Register File Energy Consumption using Compiler Support. In *Workshop on Application-Specific Processors*, 2002.

[4] 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 *International Workshop on Innovative Architecture*, 2001.

[5] 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 *Design And Test in Europe*, 2002.

[6] Luca Benini and Giovanni De Micheli. System-Level Power Optimization: Techniques and Tools. In *Design And Test in Europe*, 2000.

[7] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and Mudge T. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture*, 2002.

[8] Daniele Folegnani and Antonio González. Energy-Effective Issue Logic. In *International Symposium on Computer Architecture*, 2001.

[9] D. R. Gonzales. Micro-RISC Architecture for the Wireless Market. In *International Symposium on Microarchitecture*, 1999.

[10] S. H. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the Impact of Increasing Microprocessor Power Consumption. *Intel Technology Journal*, 1 st quarter 2001.

[11] Heather Hanson, M.S. Hrishikesh, Vikas Agarwal, Stephen W. Keckler, and Doug Burger. Static Energy Reduction Techniques for Microprocessor Caches. In *International Conference on Computer Design*, 2001.

[12] K. Inoue. *High-Performance Low-Power Cache Memory Architectures*. PhD thesis, Kyushu University, 2001.

[13] J. Kin, M. Gupta, and W. H. Mangione-Smith. The Filter Cache: an Energy Efficient Memory Structure. In *International Symposium on Microarchitecture*, 1997.

[14] Gurhan Kucuk, Dmitry Ponomarev, and Kanad Ghose. Low-Complexity Reorder Buffer Architecture. In *International Conference on Supercomputing*, 2002.

[15] 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.

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

[18] Samuel A. Steidl. *A 32-Word by 32-Bit Three-Port Bipolar Register File Implemented Using a SiGe HBT BiCMOS Technology*. PhD thesis, Rensselaer Polytechnic Institute, 2001.

[19] Hiroshi Takamura, Koji Inoue, and Vasily G. Moshnyaga. Reducing Power Consumption of Register Files through Operand Reuse. *SIGNotes computer ARChitecture*, (149), August 2002.

[20] Jessica H. Tseng and Krste Asanovic. Energy-Efficient Register Access. In *Symposium on Integrated Circuits and System Design*, 2000.

[21] 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 for High Performance & Embedded Systems*, 2001.