Power-Aware Register Assignment for Multi-Banked Register Files

Wann-Yun Shieh*, Shu-Yi Hsu
Department of Computer Science and Information Engineering
Chang Gung University, Taiwan
259 Wen-Hwa 1st Road, Kwei-Shan Tao-Yuan, 333, Taiwan
TEL: 886-3-2118800-3336, FAX: 886-3-2118700
**Corresponding author: wyshieh@mail.cgu.edu.tw*

ABSTRACT

The multi-banked register file (MBRF) is one of the most effective approaches to resolve the complexity of the monolithic register files. In order to apply the multi-banked register file to a high-end embedded processor, we design the dynamic voltage (DVS) scaling approach for MBRF to satisfy the energy constraints. However, we found that distributed bank-access behavior prevents voltage scaling from identifying when a bank is active or not. To resolve this problem, in this paper, we analyze the access behavior of temporary-values, and change their storage in banks to increase the opportunities of power saving for infrequently-used register banks. We then turn these infrequently-used register banks into lower mode by our proposed DVS circuit. For a MBRF architecture with four banks, simulation results show that, on average, our approach reduces about 19% energy consumption while performance lose can be limitted by less than 2%, compared with the MBRF without DVS.

1: Introduction

A multi-banked register file (MBRF) partitions a monolithic register file into several identical, smaller register files, each requires fewer registers [2] [3] [4]. In such architecture, temporary values are distributively stored into each bank, and accesses to these values thus can be distributed. By this feature, the demand of port number for each bank can be effectively reduced, and thus the chip area and energy consumption. So, the multi-banked register file is suitable for multi-issue processors because it provides wide bandwidth for value accesses but requires less hardware complexity, compared with a monolithic register file. However, when we apply the MBRF architecture to a high-end embedded processor, the distributed value-accessed style causes that every bank has to standby for each clock cycle. This results in serious leakage power. Therefore, if we design a power-saving mechanism to reduce its energy consumption, then it would be more suitable for a high-end embedded system.

Dynamic Voltage Scaling (DVS) is a well-know approach to reduce hardware leakage power [5] [6] [7]. The main idea of the DVS approach is to dynamically control supply voltages to a hardware component by its working mode. To implement the DVS approach in a computing system, it must have a mechanism to identify when the target hardware will stay in the active mode or in the drowsy mode. However, when we apply such DVS approach to the MBRF within multi-issue processor architecture, we found that distributed bank-access behavior prevents voltage scaling from identifying when a bank is active or not. This is because the registers in multi-banks are always assigned to values horizontally. More seriously, the out-of-order-execution feature will make value-access sequences not follow original register assignment sequences. Therefore during each cycle, all of banks have to stay in the active mode to standby for value accesses; that is, there is no opportunity to make the banks change to low-power mode (e.g., drowsy mode).

To resolve above problem, our concept is to assign temporary values to different banks according to their popularities. This will make frequently-used values and infrequently-used values to be stored in different banks. By such popularity classification, some banks can have more cycles to be idle such that they can enter the drowsy mode. This idea, however, makes high-popularity values be stored collectively into some banks. This increases more chances of access collisions occurring at register file read/write ports. In addition, when we design the DVS circuit for the MBRF, we found that if we focus on controlling the supply voltages to each individual register, we may get serious extra DVS hardware overheads.

In this paper, we design a power-aware register assignment algorithm and propose a bank-level DVS circuit to reduce the energy consumption for a MBRF. For the power-aware register assignment algorithm, we use compiler approach to analyze the behavior of value accesses and to classify values by accessed frequency. We then use the accessed frequency to assign registers to each value. Our goal is to increase the probability of infrequently-used banks entering drowsy mode. To resolve the problem of access collisions due to collective register-assignment, we design heterogeneous organization for MBRF architecture in which each bank equips different port numbers according to the popularity of stored values. For the bank-level DVS circuit, we design a power-aware controller in pipeline datapath to detect the right time to turn-on/off a register bank. Our goal is to reduce DVS hardware overheads to avoid additional power consumption.

To achieve above objectives, there are some issues to be considered. First, how a compiler analyzes the popularity of each register-value before register assignment? We can use the intermediate representation (IR) with profiling data to analyze the utilization of each value. Second, how to allocate resources (i.e., port numbers) to different banks to reduce number of collisions? And the final issue is how to control a bank changing to the active mode or drowsy mode on time? If a register cannot be powered up to the active mode before the time of being referenced (register-read) or being committed (register-write), stall cycles would happen.

The rest of this paper is organized as follows. In section 2 we discuss related works. In section 3, we discuss how to analyze value popularity for register assignment. Section 4 shows the proposed bank-level DVS circuit. Section 5 shows the simulation environment and simulation results. Section 6 gives conclusion.

2: Related Works

2.1: Register File with DVS using Compiler

Several researches have used compiler-aided approaches to reduce leakage power of register file [8] [10]. These approaches use additional registers, called the power state registers, to control supply voltages to each hardware component. These approaches analyzed the profiling data of the program to identify which hardware components during which instructions sections should be powered up or down. Then, they inserted DVS instructions into the program to change supply voltages.

[8] proposed a power-aware compilation approach to reduce the energy of the register file by inserting voltage-scaling instructions before each loop or basic block. Also, [10] used a similar approach. The difference between [8] with [10] is that [10] inserted the DVS instructions into a program by selected points which are determined by register-usage analysis, not by specific structures, e.g., basic blocks or loops. Therefore, the maximal benefits from leakage power reduction can be achieved through global program analysis, not limited by fixed program sections.

These approaches have some disadvantages. First, inserting extra DVS instructions will cause additional power consumption and performance lost during program execution. Second, their register-level DVS controller results in high hardware costs. Compared with these compiler-aided approaches, our DVS circuit can automatically change supply voltages of each bank on time without any DVS instruction annotation, and using a bank as the DVS target component will result in minimized hardware costs.

2.2: Register File with DVS using Hardware

Some researches have used pure hardware solutions to detect the right time of voltage scaling [9]. In [9], authors proposed an approach that can reduce energy of register file through instruction predecode. Their idea is to bypass the register-information that will be used from the fetch stage to the decode stage, so the unused registers can change to a low power state in early pipeline stages. The register values are awakened before being accessed. The accessed registers must be known at least one cycle before the access happens.

Those approaches have some disadvantages. First, it may increase more hardware energy or cost in the decode stage due to the need of extra logic to perform voltage-scaling detection and control. Second, a pure hardware solution cannot precisely analyze global information in a program, e.g., register usage. Third, those approaches have not considered transition energy between different power modes.

Compared with these pure hardware approaches, our approach use compiler to get global information of register usage in a program without extra logic to collect, and to design bank-level DVS circuit to avoid extra power consumption.

3: Power-Aware Register Assignment

The power-aware register assignment algorithm can be partitioned into three phases. In the first phase, we analyze value popularity (defined as VP) by scanning the register-live-range graph. In the second phase, we partition temporary values into banks according to their VPs. And finally, we assign registers to values one by one. During register assignment, if register numbers of one bank is not sufficient for assigning to the values belong to that bank, we will use a simple compensation approach to avoid register spilling.

3.1: Value Popularity Analysis

We define the term of value popularity (*VP*) to judge whether a temporary value is frequently-used or not in a program. The VP_i for temporary value *i* is defined as

$$VP_i = \frac{RC_i}{LR_i},$$

where RC_i is the reference count (i.e., number of consumers) for value *i*, and LR_i is the live-range of value *i* (i.e., the distance in instructions between the producer and the last-used consumer instruction).

Note that if a temporary value's live range is across a branch instruction and several branch-paths will use that value, then the definition of VP_i should be changed to

$$VP_i = \sum_{j=1}^{m} VP_{i,j} \times W_j,$$

Where $VP_{i,j}$ denotes the value popularity of value *i* in the *j*th branch-path, and W_j denotes the weight of the *j*th path that the branch instruction will be taken to. The value of W_j can be calculated from profiling data.

The value popularity of each temporary value, therefore, can be counted by scanning the value-live-range graph in one pass. A large VP_i

indicates that the value *i* has high probability to be referenced again after a short period, and vice versa. For those values having small VP_i, if they hold registers for a long time, then these registers will cause the major source of leakage power during program execution. Figure 1 shows the distribution of the value popularity in FFT benchmark. We found that the value popularity in a program, actually, did not follow the normal distribution. If we assume that a VP_i is "high" when its value falls into the range of 1~0.5, then about 45% temporary values have high VP_is.

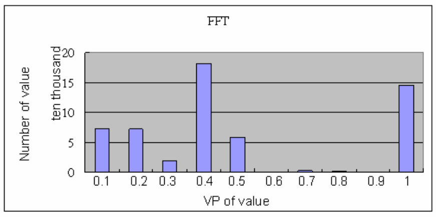

Figure 1: Value popularity distribution of FFT benchmark

3.2: Value Classification

After value popularity analysis, we classify all temporary values into groups by their *VP*s. The number of groups can be equal to the number of register-banks such that one group maps to a bank, and all values in the same group will be assigned to the same register-bank. In this paper, we assume that the register file is a four-bank architecture. So we partition all temporary values into four groups by ranges of *VP*s: (1~0.5), (0.5~0.3), (0.3~0.1), and (0.1~0). By such classification, the values of high-*VP* will be grouped into one bank, and other values will be averagely assigned to the remainder banks.

3.3: Register Assignment

The final step is to assign registers to temporary values along program sequence. When a temporary value requires to be stored in a register, we select a free register in the corresponding bank to be assigned. There is one case that should be noted during register selection. That is, when the number of registers in a bank is not enough to be assigned to a value, we should select one free register from other banks to avoid register spilling problem. In this case, we select the free register from the bank that stores the values of the highest-*VP*. There are some reasons to do so. First, most of high-*VP* values are short-lived [11]; it means that they occupy the registers in short periods, and it may have more chances to release free registers in that bank. So "borrowing" a free register from that bank would not cause serious access conflicts. Second, it will not affect the probability of the banks that store low-*VP* values entering the drowsy mode, and therefore the leakage power of whole structure would not increase.

4: Bank-Level DVS Circuit

In this section, we show the design of the bank-level DVS circuit. First, we show the pipeline architecture assumed in this paper. Next, we show how to add a register-access detection mechanism in pipeline datapath to detect register reads or writes on time. Then we show the design of the voltage controller to supply voltages for each bank.

4.1: Register-Access Detection Mechanism

Figure 2 shows the timing that a register-access signal should be detected such that the corresponding bank can be activated at the next cycle for normal access. According to the study in [6], changing the supply voltages from one mode to the other mode requires one-cycle delay. For register-read detection, therefore, an operand-read signal should pass to the voltage controller one cycle ahead of the Read-operand stage; that is, at the Decode/issue stage. Similarly, a register-write signal should be detected before the commit stage; that is, at the WB stage.

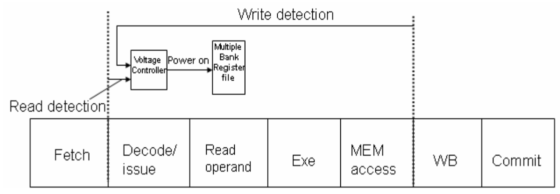

Figure 2: The timing of detecting register-access signals

Figure 3 shows the implementation of the register-access detection mechanism. For register-read operand, we can detect the first two most-significant-bits of R_s and R_t to know which bank required to be activated. The register transfers of register-read detection can be shown as

Voltage_Controller.RsWL=IF/DECODE.IR[rs]'s 2 most significant bits, and

Voltage_Controller.RtWL=IF/DECODE.IR[rt]'s 2 most significant bits,
where Voltage_Controller.RsWL (RtWL) denotes the wire line to trigger the voltage controller from *Rs* (*Rt*), and IF/Decode.IR denotes the pipeline register between the Fetch and Decode stages. Also, for register-write operand, we can detect the first two most-significant-bits of R_{Dst} to know which bank require to be activated for value-commitment. The register transfers of register-write detection can be shown as

Voltage Controller.RdWL=MEM/WB.IR[R_{Dst}]'s 2 most significant bits,
where Voltage_Controller.RdWL denotes the wire line to trigger the voltage controller from R_{Dst}, and MEM/WB.IR denotes the pipeline register between the MEM-access and WB stages.

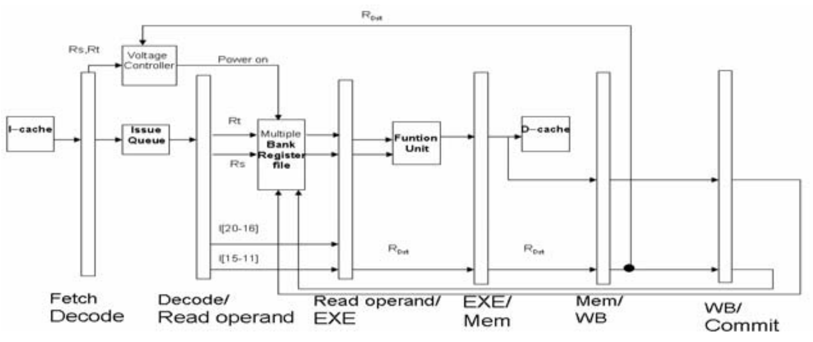

Figure 3: The implementation of the register-access detection mechanism

4.2: Voltage Controller

The voltage controller controls the supply voltages to each bank according to the signals triggered by the register-access detection, described above. The voltage controller support two voltage levels: 1 Volts (active mode) and 0.3 Volts (drowsy mode), as defined in [5]. Figure 4 shows the implementation of the voltage controller. The controller has two parts. First, access signals (RtWL, RsWL, RdWL) will be decoded. Second, if one of these signals for a bank is being set, then the power line of that bank will change voltage level. For the banks without trigger from any access-signals, they will stay in the drowsy mode. The total hardware costs of the voltage controller, for a 4-bank register file, require only three decoders (for R_s, R_t, R_{dst}), four 3-input OR-gates (one per bank), and 8 transistors (two per bank).

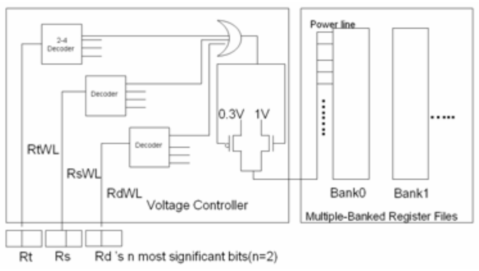

Figure 4: Voltage controller for multi-banked register file

4.3: Reducing Performance Loss

The performance loss of our approach comes from the access collisions occurring at the register bank which stored frequently-used temporary values, if the pipeline is multi-issued structure. We can design heterogeneous multi-bank organization to resolve this problem. Our concept is to allocate different port numbers to different banks according to the utilization of each bank. If the utilization of a bank is high, then we allocate more read/write ports to that bank to reduce access conflicts. If the utilization of a bank is low, we allocate fewer ports to that bank to reduce leakage power and hardware complexity.

The optimal port numbers allocated to each bank, in fact, cannot be determined easily. This is because the port number requirement depends on the access behavior in a program. Here we use a simple strategy to allocate ports to four banks. Assume that the total port number is eight, and initially each bank has been allocated two ports. We remove one port from the bank having the smallest *VP*s to the bank having the largest *VP*s; that is, the port numbers for four banks are 3, 2, 2, and 1, respectively. Simulation results show that such allocation performs lower access conflicts for the frequently-used bank, and performs lower leakage power for the infrequently-used bank.

5: Experiment Results

5.1: Experiment Methodology

We carry out our experiments and analysis by using Simplescalar simulator [14], and obtain our power model by Wattch [15]. We assume the proposed approach is implemented in a MIPS-like processor. In addition, nine benchmarks from SPEC2000 suite are used: seven for integer (gzip, bzip2, gcc, mcf, parser, twolf, vortex), and two for floating point (art, mesa). These benchmarks are often used in embedded system evaluation.

5.2: Results

The evaluation metrics in the paper include the energy consumption, performance penalty, and energy-delay product. All proposed approaches are compared with a baseline architecture which is the four-bank register file, and each bank has 2-read and 1-write ports. The evaluations include two parts. First, we evaluate the effects of register assignment. We use the power-aware register assignment to compare with the horizontal register assignment under the same register file architecture with the same bank-level DVS circuit. Second, we evaluate the effects of the heterogeneous register-bank architecture. We use the heterogeneous port allocation to compare with the homogeneous port allocation under the same register assignment and the same bank-level DVS circuit. For reference convenient, we list the abbreviations of all compared approaches in Table 1.

Table 1 Abbreviations of all compared approaches in experiment

Abbreviation	Approach
4-bank+HRA	Four-bank register file with horizontal register assignment
4-bank+HRA+DVS	Four-bank register file with horizontal register assignment and bank-level DVS circuit
4-bank+PRA+DVS	Four-bank register file with power-aware register assignment and bank-level DVS circuit
4-bank+PRA+DVS +HP	Four-bank register file with power-aware register assignment and bank-level DVS circuit. The read-port number for each bank is 3, 2, 2, and 1, respectively.

A. Energy consumption

Figure 5 shows the energy consumption for the two register assignment approaches, compared with the baseline register-file without DVS. After counting up the state transition energy, on average, the power-aware register assignment outperforms the horizontal register assignment about 23%. This shows that value distribution in four banks indeed affects the utilization of register file. By applying the DVS circuit to the register file, the leakage energy during those idle cycles can be effectively reduced.

B. Performance penalty

Figure 6 shows the performance penalty for the two register assignment approaches. For the horizontal register assignment, its performance penalty is equal to the baseline architecture. For the power-aware register assignment, its performance penalty increases about 27% than the horizontal register assignment. This extra performance penalty occurs from the power-aware register assignment which increases more access conflicts at the bank storing high-*VP* values.

C. Energy-delay product

Figure 7 shows the product of energy and performance penalty for integer and floating point benchmarks. We found that the improvements for the power-aware register assignment varied by different benchmarks. The results show that the access conflicts would be the major problem for the power-aware register assignment.

D. Performance penalty using heterogeneous bank architecture

Figure 8 shows the performance penalty when we use the heterogeneous port allocation to overcome the problem of access conflicts. We found that for the power-aware register assignment, the performance penalty increase only about 0.93% ~ 1.11%. This shows that the heterogeneous architecture can effectively reduce large conflicts of banks.

E. Energy consumption using heterogeneous bank architecture

Figure 9 shows the energy consumption for the heterogeneous bank architecture with the power-aware register assignment. We found that the energy consumption increases averagely about 2%, compared with the homogeneous bank architecture. This is because removing a read-port from one bank to another bank may increases hardware area a little, and thus the energy consumption.

F. Energy-delay product using heterogeneous bank architecture

Figure 10 shows the energy-delay product for the heterogeneous bank architecture with the power-aware register assignment. We found that the energy-delay product outperforms the horizontal register assignment by 18% on average. It shows that the heterogeneous bank architecture with the power-aware register assignment would be a good choice for implementing the multi-bank register file, in the terms of the energy-delay product.

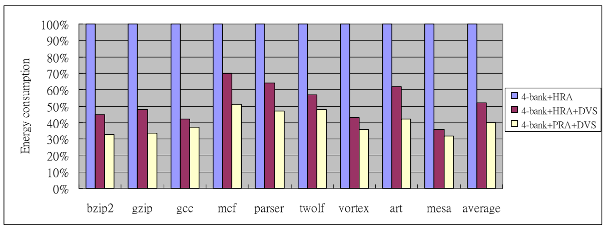

Figure 5: Energy consumption of two register assignments on the 4- bank register file

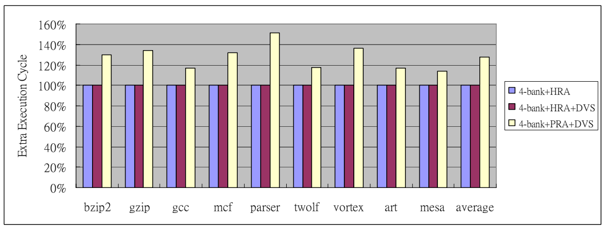

Figure 6: Performance penalty for two register assignments on the 4- bank register file

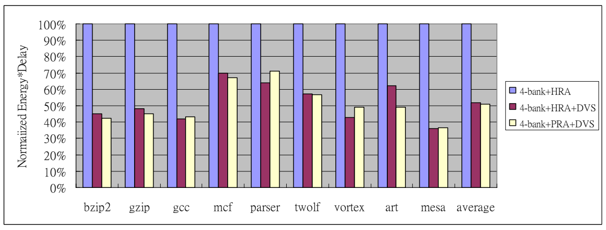

Figure 7: Energy-delay product for two register assignment on the 4- bank register file

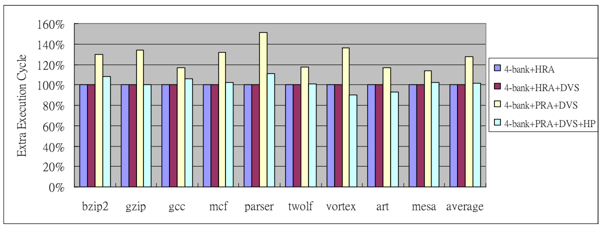

Figure 8: performance penalty of the heterogeneous port allocation approach

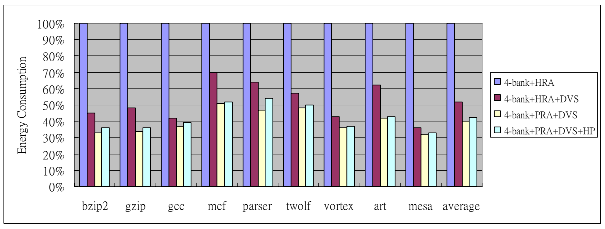

Figure 9: energy consumption for the heterogeneous bank architecture with the power-aware register assignment

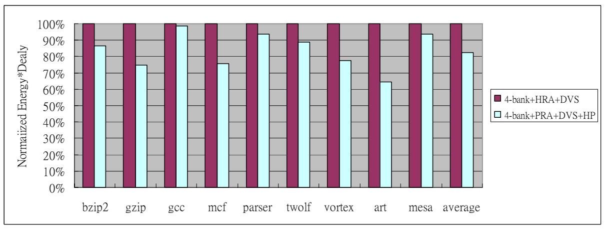

Figure 10: Energy-delay product for the heterogeneous bank architecture with the power-aware register assignment

6: Conclusion

In this paper we proposed a power-aware register assignment algorithm with bank-level DVS circuit to save the leakage power for multi-banked register files, and minimize performance loss through heterogeneous bank-architecture design. We exploit the characteristics that include the access behavior of temporary values, and their access-popularity to get more opportunities of power saving for infrequently-used register banks. To reduce performance penalty due to the power-aware register assignment, we allocate different port numbers to different banks by their utilizations. Simulation results show that the leakage power of a four-bank register file can be reduced about 19% with acceptable performance penalty (only 2%), compared with the baseline register file architecture.

Following are the advantages of our approach. First, the register assignment using the software approach can provide more flexibility between energy consumption and performance penalty for different applications. Second, our approach does not need to modify ISA. Third, our approach can effectively save leakage power with minimized performance penalty for a multi-bank register file architecture.

There are some future works. First, the MBRF can be modified to support reconfigurable computing; that is, bank configuration can be changed dynamically. Such architecture lacks related DVS studies. Second, we know that most of the high-*VP* values are short-lived; they stay in the registers, actually, in short periods. Therefore we can allocate less registers to a register bank to store these short-lived values for further leakage energy saving.

7: Reference

[1] Johan Janssen, et al., “Partitioned Register File for TTAs,” *Proceedings of MICRO-28*, pp. 303-312, November 1995.

[2] R. Balasubramonian, et al., “Reducing the Complexity of the Register File in Dynamic Superscalar Processor,” *Proc. 34th Microarchitecture (MICRO-34)*, pp. 237-248, 2001.

[3] J.H. Tseng, et al., “Banked Multiported Register files for High-Frequency Superscalar Microprocessors,” *Proc. the 30th ISCA*, pp.62-71, June 2003.

[4] Tadashi Saito, M. Maeda, et al., “Design of Superscalar Processor with Multi-Bank Register File,” *IEEE International Symposium on Circuits and Systems*, pp. 3507-3510, May 2005.

[5] K. Fkautner et al., “Drowsy Caches: Simple Techniques for Reducing Leakage Power,” *Proc. the 29th ISCA*, pp. 148-157, May 2002

[6] Nam Sung Kim, et al., “Circuit and Microarchitectural Techniques for Reducing. Cache Leakage Power,” *IEEE Transactions on VLSI*, pp. 167-184, Feb 2004.

[7] Nam S. Kim, et al., “Drowsy Instruction Caches: Leakage Power Reduction Using Dynamic Voltage Scaling and Cache Sub-bank Prediction,” *Microarchitecture (MICRO-35)*, pp. 219-230 ,Nov 2002.

[8] L. Ayala, et al., “Power-Aware Compilation for Register File energy Reduction,” *International journal of parallel programming*, pp. 151-167, Dec 2003.

[9] Ayala, J.L, “Energy Aware Register File Implementation through Instruction Predecode,” *Proceedings of the Application-Specific Systems, Architectures, and Processors*, pp. 86-96, June 2003.

[10] Wann-Yun Shieh, and Chien-Chen Chen, “Exploiting Register-usage for Saving Register-file Energy in Embedded Processors,” *Lecture Notes in Computer Science*, pp. 37-46, Dec 2005.

[11] Luis A. Lozano C. et al., “Exploiting short-lived variables in superscalar processors,” *Proceedings of MICRO-28*, pp. 292-302, Dec 1995.

[12] Wann-Yun Shieh, Hsin-Dar Chen, “Saving Register-file Leakage Power by Monitoring Instruction Sequence in ROB,” *Proceedings of the2006 International Conference on Embedded System and Application*, pp. 141-147 , June 2006.

[13] Wann-Yun Shieh, Hsin-Dar Chen, “Saving Register-file Leakage Power by Monitoring Instruction Sequence in ROB,” *The 1st International Workshop on Embedded Software Optimization (ESO 2006)*, pp.765-774, August 2006.

[14] SimpleScalar, http://www.simplescalar.com/

[15] D. Brooks, V. Tiwari, et al., “Wattch: A framework for architectural-level power analysis and optimizations,” *In 27th Annual International Symposium on Computer Architecture*, pp. 83-94, June 2000.

.