

## Power-Aware Register Assignment for Multi-Banked Register Files

WANN-YUN SHIEH AND SHU-YI HSU

*Department of Computer Science and Information Engineering  
Chang Gung University  
Taoyuan, 333 Taiwan*

The multi-banked register file (MBRF) is an effective approach to reduce the complexity for a monolithic register file. In order to apply the multi-banked register file to a low-power microprocessor, we have to design a dynamic voltage scaling (DVS) approach for the MBRF to reduce its power consumption. However, when the temporary values are stored into an MBRF, their distributed storage locations will make us difficultly identifying when a register bank should be powered up or powered down. To resolve this problem, in this paper, we analyze the accessed frequencies of the temporary-values in a program and cluster them into register banks by their frequencies through register assignment. The major goal is to make those infrequently accessed register banks have more opportunities to stay in the idle mode. Through a proposed DVS circuit, we can then supply these infrequently-accessed register banks a low voltage to save the static power. Simulation results show that, for a four-banked register file, on average, our approach reduces about 60% energy consumption while performance loss can be limited to less than 17%, compared to an MBRF without DVS.

**Keywords:** register assignment, multi-banked register file, dynamic voltage scaling, monolithic register file, accessed frequency

### 1. INTRODUCTION

The register file plays a critical role in the performance and the power consumption compared with other components in a microprocessor. Conventionally, a register file improves a processor's performance by register pressure reduction. However, its energy consumption, especially static power, causes that the size of a register file cannot be extended arbitrarily when applying to a low-power processor. For example, in the Motorola M.CORE architecture, the register file up to 32 registers has consumed 16% of total processor power-consumption and 42% of the datapath power consumption [1].

The reason of such high-percentage power consumption comes from the fact that the trend of RISC processors is using the wide-issue architecture to increase ILP (Instruction Level Parallelism). Such architecture requires a large register file to store temporary values during program execution. A large register file requires not only more physical registers but also more read/write ports to reduce access conflicts. It results in more chip area and more energy consumption. There are many previous studies proposing the approaches to reduce the complexity of the register file [2, 3]. The multi-banked register file is one effective approach to resolve above problems [4, 5].

A multi-banked register file (MBRF) partitions a large monolithic register file into

---

Received November 27, 2007; revised March 25 & June 2, 2008; accepted July 3, 2008.  
Communicated by Chung-Ping Chung.

several identical, smaller register files, called the banks. Each register bank can stand along to service function units, but with fewer registers and read/write ports [5-7]. In such banked architecture, temporary values are stored distributedly in the banks, and accesses to these values can thus be parallelized. The multi-banked register file, therefore, is suitable for multi-issue processors because it provides wide bandwidth for value accesses but requires less hardware complexity, compared with the original monolithic register file.

When the MBRF architecture is implemented in a multi-issue processor, however, its distributed value storage will cause that each register bank has to standby all the time. This results in serious static power dissipation because not all banks need to be accessed for each execution cycle. Fig. 1 shows the utilization of a multi-banked register file when we apply it to a 4-issue processor. We assume that the register file has 4 banks, and each bank has 8 registers with 2-read and 1-write ports. We use SimpleScalar 3.0 [8] to simulate value-accessed events in such architecture, and select “gzip” and “art” programs in SPEC2000 [9] as benchmarks. In Fig. 1, the Y-axis represents the utilization of each register bank in percentages and the X-axis lists all banks. We found that, only one or two banks have utilization over 40%. This means that, on average, more than half of execution time a register bank may be idle, and this becomes the major source of static power dissipation. In the study of [10], authors further pointed out that when the technology of the processor drops below the 65-nm feature size, the static power in a processor will become more serious, and even exceed the dynamic power.

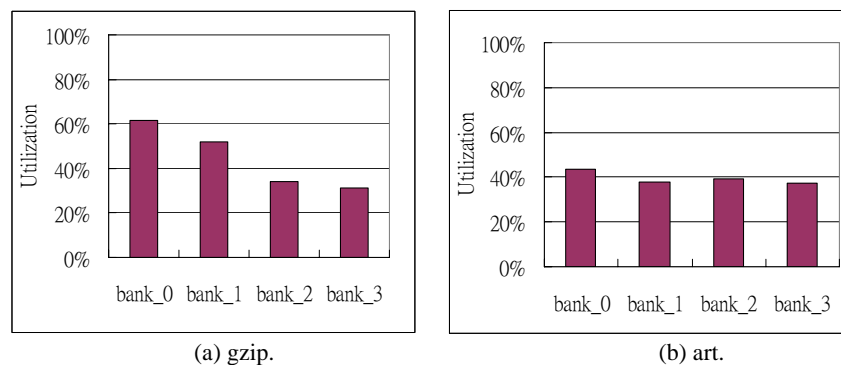


Fig. 1. Utilizations of a four-banked register file for (a) gzip and (b) art (SPEC2000).

Dynamic Voltage Scaling (DVS) is a well-known approach to reduce hardware static power [11-13]. The concept of the DVS is to dynamically control supply voltages to a hardware component by its working modes. The working modes are recorded in some additional registers (called the power-state registers), and determined by either hardware or software mechanism. Such mechanism should identify when the target hardware has to stay in the active mode or in the low-power mode.

If we apply the DVS approach to the MBRF for a multi-issue microprocessor, the problem is how to identify when a register bank will be accessed or not. For most compilers, registers are picked up distributedly to store temporary values. For example, the

GCC -O2 compilation attempts to assign register numbers in instructions in order to maximize the amount of registers used [14]. Such assignments will make frequently-used and infrequently-used values mix in register banks. Moreover, the out-of-order-execution feature, which generally appears in a multi-issue processor, makes register-accessed orders not exactly follow original register assignment orders. As a result, one cannot easily determine which register or which bank will (or will not) be accessed in the next execution cycle.

To resolve above problem, we modify the register assignment algorithm to let some register banks have more chances to enter the lower-power mode. Our concept is to assign temporary values to different register banks according to their accessed frequencies. That is, we let frequently-used values and infrequently-used values be stored separately in different banks through register assignment. By clustering different frequency values into different banks, the banks storing more infrequently-used values will have much time to stay in the idleness, and thus they can have less static power consumption via DVS control.

This idea, however, has a flaw to be solved. That is, it makes frequently-used values be stored collectively into some register banks. This will increase the chances of access-conflicts occurring at these banks, either for reads or writes. Serious access conflicts will cause performance loss, and even more energy consumption. If we can effectively reduce those access conflicts for the proposed register assignment algorithm, then the remaining problem is that we need a low-cost DVS circuit which can detect the access to which register bank, and power up or down that whole bank.

In this paper, we propose a power-aware register assignment algorithm and design a bank-level DVS circuit to reduce the energy consumption for an MBRF with minimized side-effects. For the power-aware register assignment algorithm, we use the compiler to analyze the accessed frequency of each temporary value. We use these analysis results to assign registers, according to the clustering principle described above. To resolve the problem of access conflicts, caused by collective register-assignment, we use a heterogeneous read-port configuration, in which each register bank has different read ports according to the popularity of values stored in it. For the bank-level DVS circuit, we design a power-aware detector in the pipeline datapath to detect the access time for each register bank. We also design a DVS controller to switch the supply-voltages to a whole register bank, instead of each individual register. Such DVS circuit can save large amount of power-state registers required.

To achieve the objective, there are some issues to be considered. First, how does a compiler measure a temporary value's accessed frequency before register assignment? Second, how many access conflicts can we reduce by adding one more read port to each register bank? And the final issue is how to change a register bank's power mode on time? If a register in a bank cannot be activated before the time of being referenced (register-read) or being committed (register-write), stall cycles in pipeline will happen. These issues will be considered in our design.

The rest of this paper is organized as follows. In section 2 we discuss related works. In section 3, we show how to analyze value-accessed frequency 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

Many previous studies have applied different register file architectures to multi-issue processors for hardware complexity reduction. Most of them can be classified into three types: cluster register file [2, 15], two-level MBRF [6, 16], and one-level MBRF [4, 5, 7]. The clustered register file is a well-applied technique in most VLIW processors. In this architecture, pipeline datapath is divided into two clusters, and each cluster owns a local register file. This architecture reduces hardware complexity (*i.e.*, port number, wire lines, controls) for each cluster, but requires inter-cluster communications when a register is accessed by a different cluster.

For the two-level MBRF architecture, a register file is partitioned into two levels, like L1 and L2 caches, to speedup value accesses for level 1 registers and reduce port requirements for level 2 registers. In [6], level 1 register file acts as a regular register file and is visible to the datapath. Level 2 register file is used for keeping checkpoints of the registers which are released speculatively. The objective of this architecture is to reduce average access latency. However such register caching concept may require significant hardware costs for control logic.

For the one-level MBRF, a register file is partitioned into multiple banks, and each bank can be accessed by all function units. The authors in [7] showed that a 4-bank register file can increase 7.2% performance loss than a monolithic register file, but theoretically can achieve 80% power saving. Compared with above two MBRF architectures, the one-level MBRF provides more bandwidth with less hardware overheads, and without unnecessary data movements between clusters or upper-lower levels. Our approach, in fact, can be applied to all of them, but without loss of generality we focus on the one-level MBRF architecture.

Authors in [17] proposed a power-aware compilation approach to reduce the energy consumption of the register file by inserting voltage-scaling instructions before each loop or basic block. [18, 22] used a similar approach. The difference between [17] with [18] is that [18] inserted the DVS instructions into a program by selected points which are determined by register-usage analysis, not by specific loop or basic-block structures. The maximal benefits from static power reduction, therefore, can be achieved through global program analysis, not limited by fixed program sections.

Other researches have used hardware solutions to detect the right time of voltage scaling [19]. In [19], authors proposed an approach that can reduce energy consumption of register file through instruction predecode. Their idea is to bypass the register-access signals (*e.g.*, register-reads) from the instruction-fetch stage to the decode stage. The registers which will be accessed in the next stage are therefore being awakened; other unused registers will be changed to the low-power mode for energy saving. In this approach, the accessed registers must be known at least one cycle early before the access happens. Although their approach can perform exactly voltage-scaling detections and controls, lack of compiler supports (*e.g.*, register usage analysis) may cause too many redundant actions on power management. Compared with this approach, we use compilation analysis to cluster temporary values in register banks by accessed frequencies, which can largely avoid intensive DVS controls.

### 3. POWER-AWARE REGISTER ASSIGNMENT

The goal of the power-aware register assignment is to cluster less-used values into the register banks such that they can have more chances to enter the drowsy mode, and to cluster frequently-used values into the other banks but they have to avoid serious access conflicts. Fig. 2 shows our register assignment approach implemented in a typical compilation flow. In Fig. 2, most front-end optimizations (largely machine independent) must finish before the register assignment stage. After that, the power-aware register assignment algorithm takes IR (Intermediate Representation) and optionally with execution profiling data as inputs for register allocation. It neither affects any compiler optimizations, nor destroys any data parallelisms in instructions.

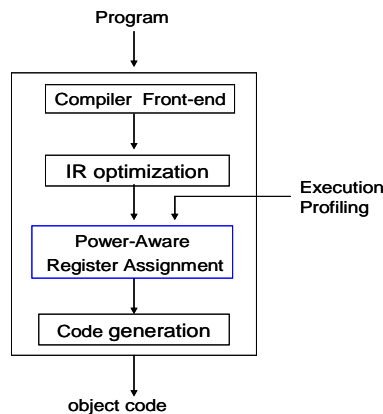


Fig. 2. Power-aware register assignment in the compilation flow.

The power-aware register assignment algorithm can be partitioned into three phases. In the first phase, we analyze each value's popularity (We will quantify this factor later) by scanning the register-live-range graph. In the second phase, we cluster temporary values into different register banks according to their popularities. And in the last phase, we assign free registers to values one by one. In this step, if free registers in one bank is not enough for assignment, we need borrow free registers from other banks to avoid register spilling.

#### 3.1 Value Popularity Analysis

We define the term of value popularity ( $VP_i$ ) to judge a temporary value  $i$  as a frequently-used one or a less-used one in a program. The  $VP_i$  can be expressed as

$$VP_i = \frac{RC_i}{LR_i}, \quad (1)$$

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

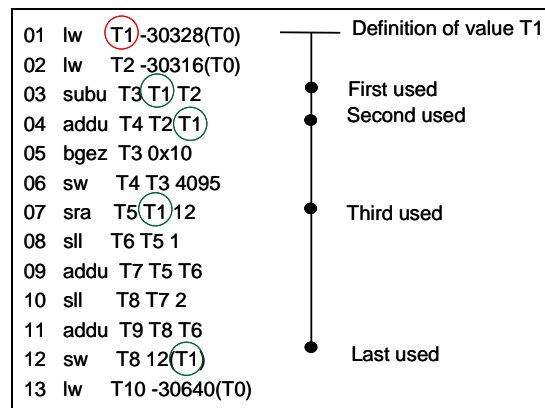


Fig. 3. An example to count the value popularity.

Fig. 3 shows an example to count  $VP_{T1}$  for value  $T1$ . The temporary value  $T1$  is generated (or say defined) at the first instruction in line 1. When we analyze its live-range, we can find that there are four references occurred after its producer (*i.e.*,  $RC_{T1} = 4$ ), and its live-range continues until line 12 (*i.e.*,  $LR_{T1} = 11$ ). Therefore,  $VP_{T1} = 4/11 = 0.36$ .

Note that if a temporary value's live ranges are across a branch instruction and this value may be referenced in several branch-paths, then the definition of  $VP_i$  can be modified as

$$VP_i = \sum_{j=1}^m VP_{i,j} \times W_j, \quad (2)$$

where  $VP_{i,j}$  denotes the value popularity of value  $i$  if the control flow goes through the  $j$ th branch-path, and  $W_j$  denotes the weight (or probability) of the  $j$ th branch-path that will be taken. The value of  $W_j$  can be estimated from execution profiling. That is why we need to feed the profiling data into our register assignment algorithm.

The value popularity of each temporary value, therefore, can be calculated by scanning the live-range graph in one pass. A value  $i$  having large  $VP_i$  indicates that it has been referenced many times or will have high probability to be referenced again after a short period. This will make the registers holding this value have large access activities. For those values having smaller  $VP_i$ , the registers will hold them for a long time, possibly with few references. They would be the major source of the static power occurred during program execution. Fig. 4 shows the value popularity distribution in "gzip" benchmark [9]. We find that the value popularity did not follow the normal distribution. If we assume that the  $VP_i$  is "low" if it falls below 0.5, then about 55% temporary values have low  $VP_i$ s.

### 3.2 Value Clustering and Register Assignment

After value popularity analysis, we cluster all temporary values into groups by their  $VP$ s. The number of groups can equal to the number of register-banks such that one group maps to one bank. All values in the same group will be assigned to the same register-bank.

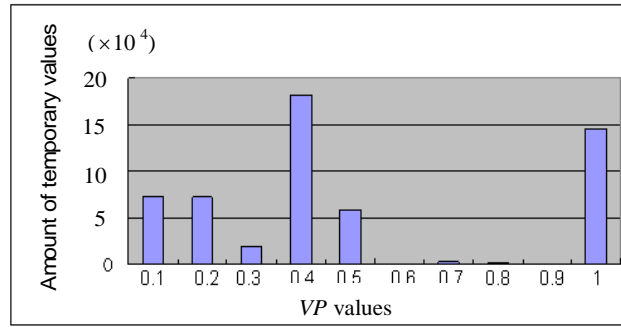


Fig. 4. Value popularity distribution for "gzip" benchmark.

For example, we assume that the register file is a four-banked architecture, and we can partition all temporary values into four groups by ranges of VPs: (1 ~ 0.5), (0.5 ~ 0.3), (0.3 ~ 0.1), and (0.1 ~ 0). Such classification makes the values of high-VPs be grouped into one register bank, and other values will be assigned averagely to the remaining banks.

To achieve better clustering on register assignment, we can partition the values by an optimization algorithm. We model the energy cost function as follows. Assume that  $W_{active}$  is the power consumption of a register bank staying in the active mode per cycle (Joule/cycle), and  $W_{drowsy}$  is for the drowsy mode. If a register bank  $i$  occurs  $N_i$  mode-transitions (each transition suffers  $E_{mode\_trans}$  energy overheads) and totally has  $T_i$  execution cycles to stay in the drowsy mode, then the total register file energy saving is

$$E_{saving} = \sum_{\text{register bank } i} T_i \times (W_{active} - W_{drowsy}), \quad (3)$$

and the total energy overhead due to mode transitions is

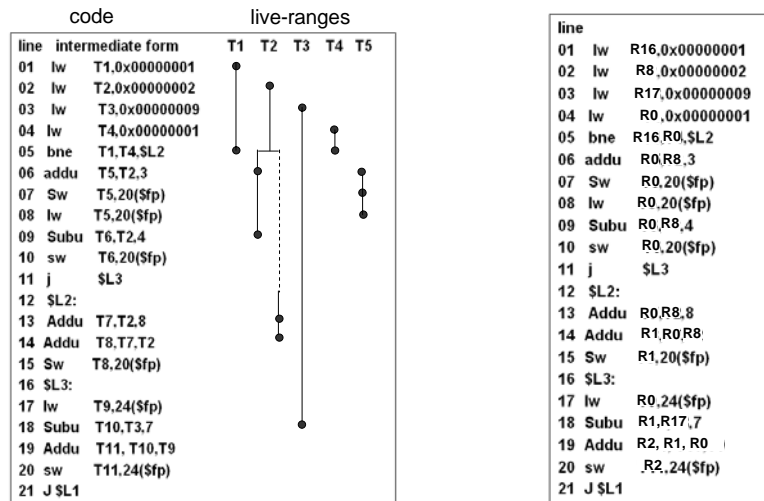
$$E_{overhead} = N_i \times E_{mode\_trans}. \quad (4)$$

The objective of clustering temporary values in register banks is to maximize  $E_{saving}$ , subject to the constraint  $E_{saving} > E_{overhead}$ . All energy variables in Eqs. (3) and (4) can be estimated by the power model in section 5.1 shown later.

The final step is to assign free registers to temporary values along program sequence by above clustering. When a temporary value needs to be stored in a register, we select a free register in the corresponding register bank to assign. There is one case that should be noticed during register selection. If free registers in a bank are exhausted and cannot be assigned to any temporary values, we should select one free register from other banks to avoid register spilling. In this case, we select a free register from the bank that has stored the values of higher-VPs. There are some reasons to do so. First, most high-VP values are short-lived [21]; it means that they occupy the registers in very short periods, and it may have more free registers to be released in that bank. So "borrowing" a free register from that bank would not cause serious access conflicts. Second, it would not affect the probability of the banks that store low-VP values entering the drowsy mode. Therefore it has very few effects on static power increases for the whole register file.

### 3.3 Example

We use an example code in Fig. 5 (a) to show the process of the register assignment described above. In Fig. 5 (a), the left-hand side is the code represented by the intermediate representation form, in which  $T_x$  is the name of temporary values, and the right-hand side shows their live-ranges. Each value's popularity can be calculated by Eq. (1), except  $T_2$ . Note that the value of  $T_2$  is defined at Line 02, and is referenced across the branch instruction at Line 05 in two paths: not-taken (Lines 06 ~ 11), and taken (Lines 12~). On the not-taken path,  $T_2$  is being referenced twice, and the length of the live-range along this path is 7, which is counted through Lines 3 ~ 5 and Lines 6 ~ 9. On the taken-path,  $T_2$  is also being referenced twice, but the length of the live-range along this path is 5, which is counted through Lines 3 ~ 5 and Lines 13 ~ 14. According to Eq. (2), if we assume that the weights of both paths equal to 0.5, then  $VP_{T_2}$  is  $(2/7) * 0.5 + (2/5) * 0.5 = 0.34$ . Table 1 lists temporary values  $T_1 \sim T_5$  and shows each value's popularity at column two.



(a) IR and register live-ranges. (b) Code after register assignment.

Fig. 5. An example code and live-ranges of values  $T_1 \sim T_5$ .

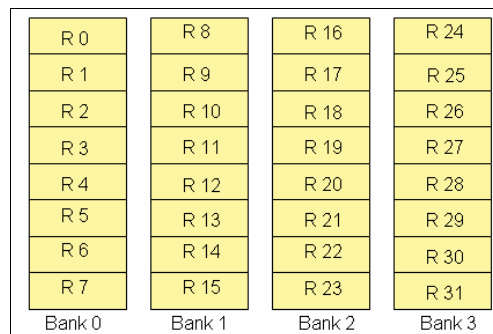


Fig. 6. Configuration of a four-banked register file.

**Table 1. Register assignment for values  $T1 \sim T5$ .**

Temporary values	VP	Bank clustering	Register assignment
$T1$	0.25	Bank 2	R16
$T2$	0.34	Bank 1	R8
$T3$	0.12	Bank 2	R17
$T4$	1	Bank 0	R0
$T5$	1	Bank 0	R0

Assume that the register file is a four-banked architecture, and the register naming is shown in Fig. 6. From the value clustering design, we classify all temporary values into corresponding register banks by their  $VP$ s, as shown in the third column of Table 1. The final step is to pick up (assign) free registers in different banks to all values. The results of register assignment are shown in the last column of Table 1. Fig. 5 (b) shows the code after power-aware register assignment.

## 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. Then we show the design of the voltage controller which controls the supply voltages to each bank. When the detection mechanism detects the access signals, the voltage controller should change the supply voltages of the corresponding register banks by the active mode (1Volt) on time. Otherwise, the voltage controller will turn register banks into the drowsy mode (0.3Volts). Finally, we show how to reduce performance loss by heterogeneous architecture design.

### 4.1 Pipeline Architecture

We assume the pipeline contains seven stages: Fetch, Decode/issue, Read-operand, Exe, Mem-access, WB, and Commit. In the Fetch stage, we assume that four instructions are fetched from the instruction cache to the instruction queue. In the Decode/issue stage, instructions are decoded and placed into the issue queue and the reorder buffer. In the Read operand stage, instructions are waiting for source operands from register file or pipeline forwarding bus. In the Exe stage, instructions which all source operands are available can be issued to the functional units to be executed. In the Mem-access stage, the load instructions will access data memory. In the WB stage, the results produced from functional units or data memory will be written to the reorder buffer. In the Commit stage, registers will be updated when the instruction executes completely.

### 4.2 Register-Access Detection Mechanism

Fig. 7 shows the timing of the register-access detections. According to the study in [13], we know that a 1-bit storage cell consists of 6 transistors, and the wake-up latency of  $64 \times 32$  storage cells (*i.e.*, 64 registers) is 0.28ns. In our power model and the following

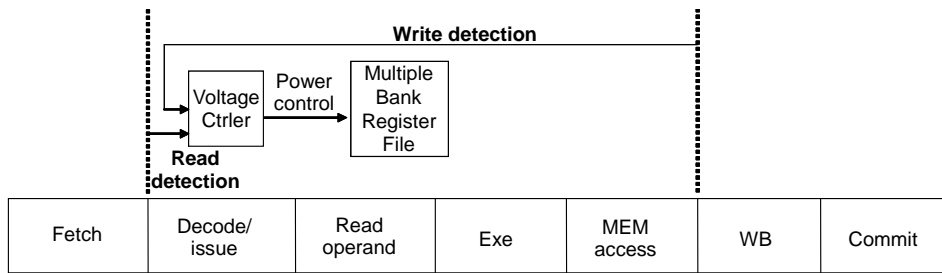


Fig. 7. The timing of register-access detections.

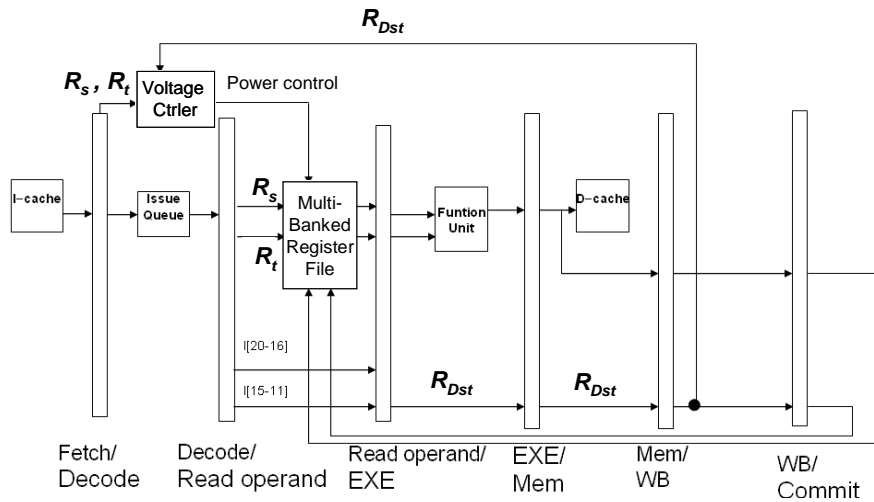


Fig. 8. The implementation of the register-access detection mechanism.

experiment, the register file access time is 0.85ns. So changing the supply voltages from one mode to the other mode requires at least one-cycle delay. For register-read detections, a register-read signal should be transferred 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.

Fig. 8 shows the detail implementation of the register-access detection mechanism. For register-reads, we can detect the first two most-significant-bits of source registers (*i.e.*,  $R_s$  and  $R_t$ ) to identify which banks required to be activated. The register transfers of the register-read detection can be expressed as

$$\text{Voltage\_Controller.RsWL} = \text{IF/Decode.IR}[R_s]\text{'s 2 most significant bits,} \quad (5)$$

and

$$\text{Voltage\_Controller.RtWL} = \text{IF/Decode.IR}[R_t]\text{'s 2 most significant bits,} \quad (6)$$

where Voltage\_Controller.RsWL (RtWL) denotes the wire lines to trigger the voltage controller from  $R_s$  ( $R_t$ ), and IF/Decode.IR denotes the instruction register between the

Fetch and the Decode stages. Also, for register-write, we can detect the first two most-significant-bits of destination register ( $R_{Dst}$ ) to identify which banks required to be activated for value-commitment. The register transfer of the register-write detection can be expressed as

$$\text{Voltage\_Controller.RdWL} = \text{MEM/WB.IR}[R_{Dst}] \text{'s 2 most significant bits,} \quad (7)$$

where  $\text{Voltage\_Controller.RdWL}$  denotes the wire lines to trigger the voltage controller from  $R_{Dst}$ , and  $\text{MEM/WB.IR}$  denotes the instruction register between the MEM-access and WB stages.

### 4.3 Voltage Controller

The voltage controller controls the supply voltages to each bank according to the signals triggered by the register-access detection, as described above. The voltage controller supports two voltage levels: 1 Volts (active mode) and 0.3 Volts (drowsy mode), as used in [11]. Fig. 9 shows the implementation of the voltage controller.

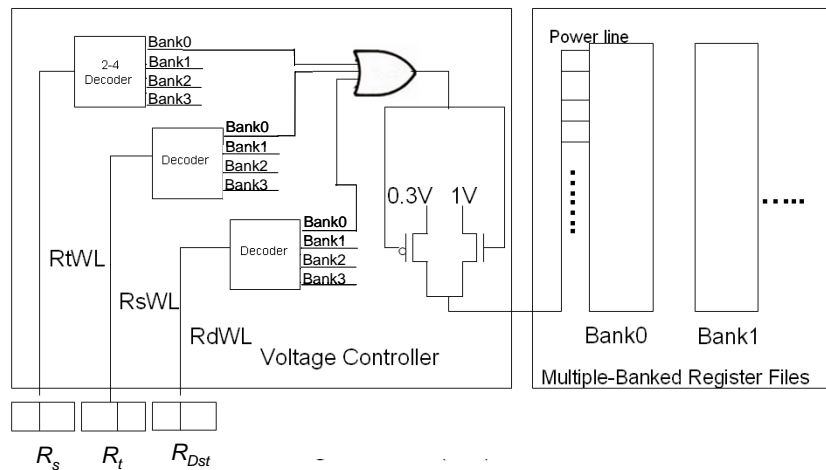


Fig. 9. Voltage controller for a four-banked register file.

The controller has two parts. First, access signals (*i.e.*,  $R_tWL$ ,  $R_sWL$ ,  $R_dWL$ ) in Eqs. (5)-(7) 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-banked 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).

### 4.4 Reducing Performance Loss

If the pipeline is a multi-issue and out-of-order execution structure, our approach will have performance loss which comes from the access conflicts occurring at the regis-

ter banks storing frequently-used temporary values. We can design a heterogeneous port configuration to resolve this problem. That is, we allocate different read ports to different register banks according to the utilization in each bank. If the utilization of a bank is high, we allocate more read 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 redundant power consumption and hardware complexity.

## 5. EXPERIMENT

In this section, we evaluate the effectiveness of the proposed approaches. At first, we show the experiment methodology, including the experiment environment and parameters. Then we show the evaluation results.

### 5.1 Experiment Methodology

We perform experiments and analysis in SimpleScalar simulator [8]. We assume that the target processor model, as shown in Table 2, follows the default hardware configurations of MIPS in SimpleScalar [8]. The baseline architecture of the register file is a four-banked register file, and each bank has 16 registers with 2-read and 1-write ports in default.

**Table 2. Hardware configurations of MIPS in SimpleScalar.**

Machine width	4-wide fetch, 4-wide issue, 4-wide commit
Window size	32-entry fetch queue, 16-entry RUU, 32-entry load/store queue
L1 I-cache	32KB, 2-way set-associative, 32-byte line, 2 cycles hit time
L1 D-cache	32KB, 4-way set-associative, 32-byte line, 2 cycles hit time
L2 Cache combined	512KB, 4-way set-associative, 64-byte line, 4 cycles hit time
BTB	1024 entry, 2-way set-associative
Memory	128-bit wide, 12 cycles first chunk, 2 cycles interchunk
TLB	64 entry (I), 128 entry (D), 4-way set-associative, 30 cycles miss latency
Function Units and Latency (total/issue)	4 Int Add(1/1), 1 Int Mult(3/1)/Div(20/19), 2 Load/Store(2/1), 4 FP Add(2), 1 FP Mult(4/1)/Div(12/12)/Sqrt(24/24)

**Table 3. Energy parameters and drowsy transition time.**

Active mode leakage energy per bit	1.63E-15J
Drowsy mode leakage energy per bit	2.59E-16J
Mode transition energy	7.18E-16J
Mode transition time	0.28ns
Drowsy transition latency	1 cycle

To evaluate the energy consumption for a multi-banked register file, we adopted the same power model in [11]. This model can measure not only the leakage energy of the drowsy mode and the active mode, but also the mode-transition delay, based on the

HSPICE and the Berkeley Predictive Model [24] for a 0.07 $\mu$ m process. To measure the transition delay, we connected our four-banked register file to the supply voltage controllers and then estimated the capacitances of the supply voltage bit lines. The detailed power values are listed in Table 3. The access time and the area of the register file were estimated using the CACTI model [25] for the same technology process. (Though the CACTI model was used to model SRAM caches, previous major register-file studies [6, 16, 23] also used it as the analytical model.) Through these models, the register file access time is 0.85ns and the transition delay to and from the drowsy mode is 0.28ns. Therefore we assume a single cycle transition delay for our approaches.

Our experiments used SPEC2000 [9] as benchmark suite, including seven integer programs (gzip, bzip2, gcc, mcf, parser, twolf, vortex), and two floating point programs (art, mesa). Programs were compiled with the GCC compiler using -O2 optimization flags. All simulations skip first initialization instructions and run 100 million instructions to evaluate our design.

## 5.2 Experiment Results

The evaluations include two parts. First, we evaluate the effects of the register assignment on energy consumption reduction. We compare the proposed power-aware register assignment approach with the other two conventional assignment approaches proposed in [4], *i.e.*, the horizontal assignment, and the vertical assignment. Take Fig. 6 as example. The horizontal register assignment is to assign free registers to temporary values “horizontally” along register banks. That is, if bank0 contains registers  $R0 \sim R7$ , bank1 contains registers  $R8 \sim R15$ , and so on, then the horizontal assignment will sequentially assign free registers by  $R0, R8 \dots$  and *etc.* Oppositely, the vertical assignment will sequentially assign registers by  $R0, R1 \dots$  and *etc.* The former approach emphasizes value distribution, while the latter approach emphasizes space utilization.

The second part of the evaluation is the effects of increasing port numbers in different register banks on performance-loss reduction. We compare several port allocation policies under the same register assignment algorithm and the same bank-level DVS circuit.

For reference convenient, we list the abbreviations of all compared approaches in Table 4. In Table 4, the notation “+<assignment>” denotes that a selected register assignment algorithm is used, the notation “+DVS” denotes that the DVS approach is implemented, and the notation “+HP(a/b/c/d)” denotes the port allocation policy, in which bank0  $\sim$  bank3 have a  $\sim$  d read ports, respectively.

### 5.2.1 The effects of the register assignment

Fig. 10 shows the comparisons of the register file energy consumptions for the horizontal register assignment, the vertical register assignment, and the proposed power-aware register assignment. All approaches are applied with the same bank-level DVS. The results are normalized to the energy consumption of the baseline register file without DVS, which are shown as the Y-axis. While the X-axis lists all benchmarks we select for comparison, we also show the average results at the end of the X-axis.

**Table 4. Abbreviations of all compared approaches.**

Approach abbreviation	Description
Baseline	Four-banked register file without DVS
4 Bank + HRA + DVS	Four-banked register file with <i>Horizontal Register Assignment</i> and bank-level DVS circuit
4 Bank + VRA + DVS	Four-banked register file with <i>Vertical Register Assignment</i> and bank-level DVS circuit
4 Bank + PRA + DVS	Four-banked register file with <i>Power-aware Register Assignment</i> and bank-level DVS circuit
4 Bank + PRA + DVS + HP( <i>a/b/c/d</i> )	Four-banked register file with <i>Power-aware Register Assignment</i> , bank-level DVS circuit, and <i>Heterogeneous read-Port</i> configurations. The read ports for four banks are <i>a, b, c, and d</i> , respectively.

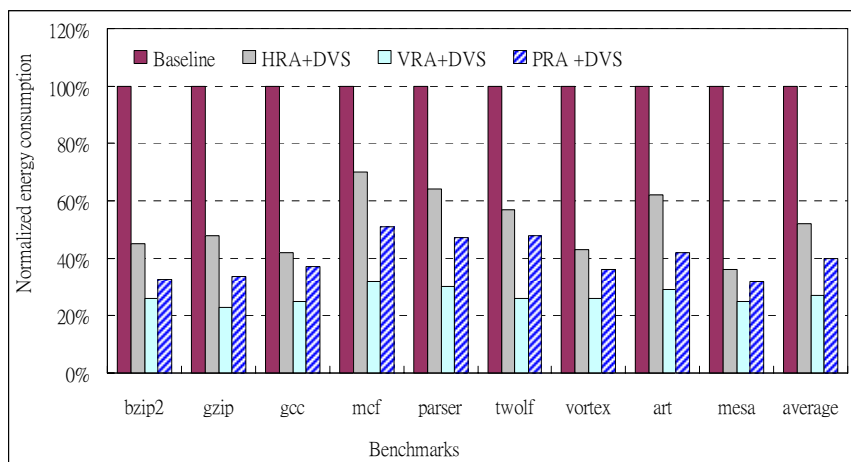


Fig. 10. Energy consumptions for the three register-assignments on a four-banked register file.

From Fig. 10 we found that, on average, the power-aware register assignment with the bank-level DVS circuit can reduce original energy consumption by 60%, and outperforms the horizontal register assignment by about 13%. On the other hand, the vertical register assignment has the lowest energy consumption, which outperforms the horizontal assignment by 25%, and outperforms the power-aware assignment by 12%. These results show that the storage locations of temporary values in four banks indeed affect the utilization of each bank, and thus their static power. Conventionally the horizontal assignment represents a “load-balanced” approach, and it will have high throughput for a multi-issue pipeline. However, as we state in the first section, mixing frequently-accessed temporary values with infrequently-accessed ones into each bank will make all register banks have to standby almost all the time, and therefore cause higher static power. By classifying these two kinds of values into different banks and then applying the DVS mechanism to each bank, the register banks with few accesses will thus have more free cycles to enter the drowsy mode. This is why the power-aware register assignment can effectively reduce a register file’s energy consumption.

The vertical assignment approach, on the other hand, represents a “space-efficient” approach because it will localize most accesses only on some register banks. If we cooperate such register assignment with a bank-level DVS, the other banks without accesses will have the maximum energy-saving due to their very low utilization. This is why the vertical assignment outperforms the other two assignment approaches.

Nevertheless, the vertical assignment approach has more serious performance loss than the other two approaches. Fig. 11 shows the ratios of the delay cycles to the total execution cycles for the three assignment approaches with the same DVS circuit. The delay cycles come from the access conflicts occurred at the banks without sufficient read ports. From Fig. 11, we found that the horizontal register assignment has the fewest access conflicts because the temporary values are widely distributed into banks. The vertical register assignment increases almost double delay cycles. This side-effect on performance loss made its benefits of energy-saving drop drastically or even become unprofitable. On the other hand, the power-aware register assignment increases about 24% delay cycles than the horizontal register assignment, but outperforms the vertical assignment by about 68%.

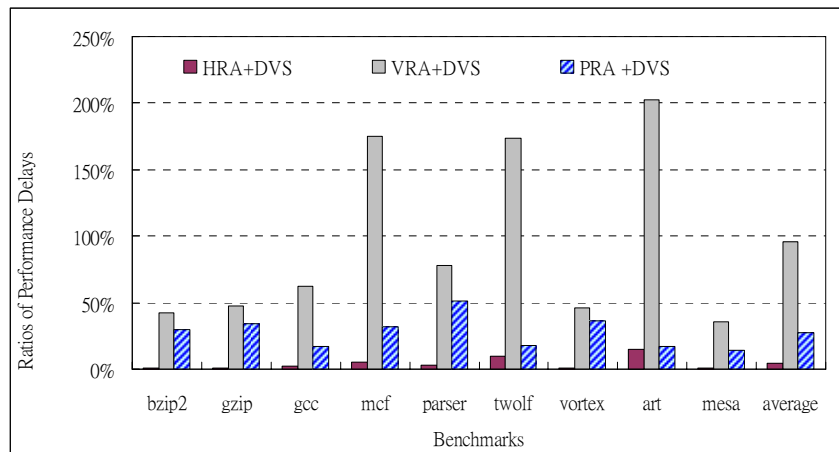


Fig. 11. Performance losses for the three register assignments.

Fig. 12 further shows the energy-delay products for the three assignment approaches. The term of the energy-delay product is a widely-used metric to balance benefits in energy saving with any degradation in performance. We found that the improvements of the power-aware register assignment approach varied among benchmarks. For “mcf”, “twolf”, and “art” benchmarks, the vertical register assignment has larger energy-delay products due to serious performance penalties (also see Fig. 11), but the power-aware register assignment does not. While these three programs represent a minimum cost network flow solver, place and route simulator, and neural network simulator, respectively, they all have heavy loop structures, compared with other programs. This means that they will have intensive accesses on some register sets during execution, and thus applying the vertical register assignment to them will cause so higher access conflicts. For the power-aware register assignment, however, those frequently-used values will be

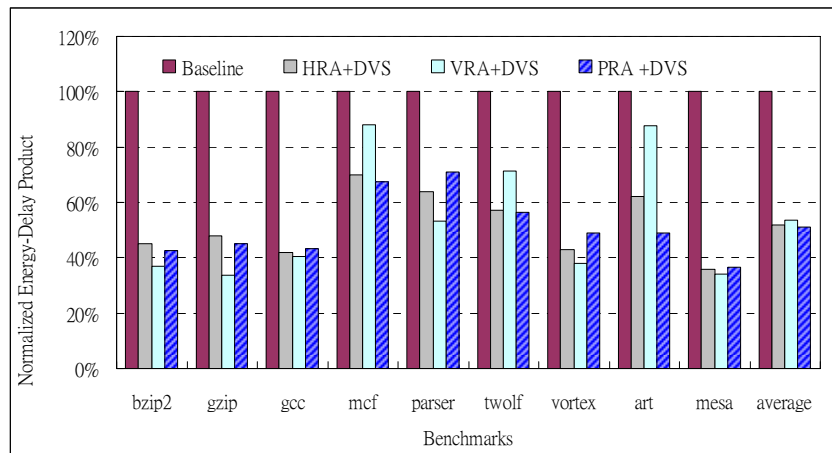


Fig. 12. Energy-delay products for the three register assignments.

distributed into the first or second register bank, and thus may not suffer the same problem. On the other hand, the power-aware register assignment achieves a comparable energy-delay product to the horizontal assignment, except the “parser” (natural language processing) and “votext” (object oriented database) programs. The possible reason is due to the program characteristics that most temporary values in those two programs have very average access frequency. Therefore the power-aware register assignment has higher access-conflict ratios in each bank.

### 5.2.2 The effects of the heterogeneous port configuration

We next evaluate the relationships of conflict reduction and port number increasing for each bank. We take the benchmark “bzip2” as example. Fig. 13 shows the conflict counts at each bank when the read ports of each bank increase from two to five. Initially, the bank 0 and the bank 1 have serious conflicts by our proposed register-assignment. This is because these two banks store most “popular” temporary values. For bank 0, if we add one more read ports (from two to three), the number of conflicts reduces about 57%, and if we add two more ports (from two to four), the reduction can even achieve

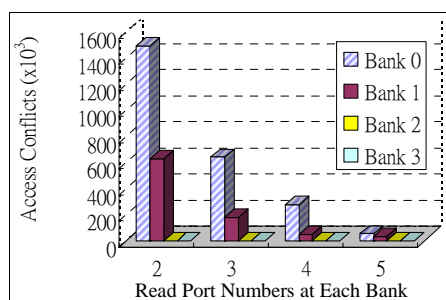


Fig. 13. Access conflicts at each register bank for benchmark “bzip2”, where the read ports of each bank increase from two to five.

more than 80%. For bank 1, if we increase one read port from two to three, the conflicts can reduce more than 70%. On the other hand, the bank 2 and the bank 3 have very few conflicts; they need not any additional read ports.

Although increasing read ports is an effective approach to reduce access conflicts, it will have side-effects in additional energy consumption. To evaluate this effect, we select three configurations to compare: (2, 2, 2, 2), (3, 2, 2, 2), and (4, 3, 2, 2). The first configuration is the baseline configuration, in which each bank has the same amount of read ports. In the second configuration, we add one more read port only at bank 0 to resolve additional conflicts. In the third configuration, we add two more read-ports at bank 0, and one more read-port at bank 1. Fig. 14 shows the energy-delay products for these three configurations. We found that on average the results of (4, 3, 2, 2) are close to the results of (2, 2, 2, 2). This means that the third configuration causes too much additional energy consumption though it can reduce most conflicts. On the other hand, although the second configuration increases only one more read-port, it can effectively improve total access conflicts but without too much additional energy consumption.

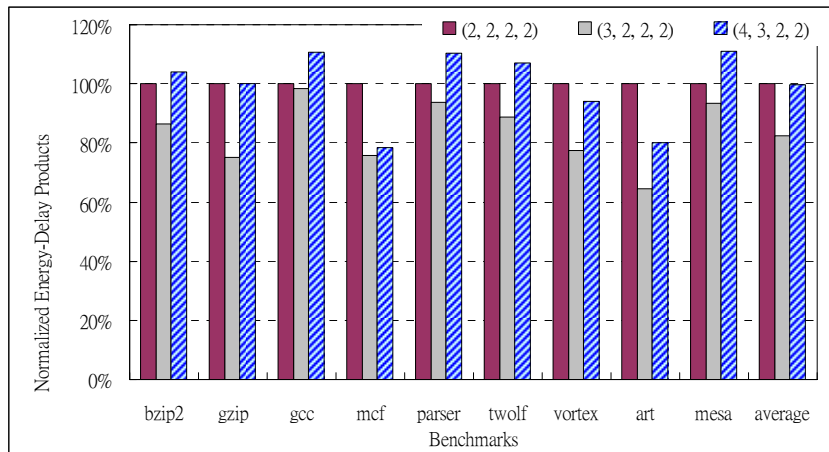


Fig. 14. The energy-delay products of three read-port configurations under the power-aware register assignment

**Table 5. Comparisons of the access time and the area overhead for the three read-port configurations.**

	MBRF(2, 2, 2, 2)	MBRF(3, 2, 2, 2)	MBRF(4, 3, 2, 2)
Access time (ns)	0.852	0.861	0.872
Area (mm <sup>2</sup> )	0.273	0.322	0.4318
Area increasing ratio	1	1.18	1.58

We further evaluate the access time and area overheads for these three configurations as adding more read ports. The results are shown in Table 5. We found that the access time increases not very obviously, but the area rises very significantly when we add totally three read ports to the register file (*i.e.*, 4, 3, 2, 2). Compared with the baseline con-

figuration, the second one increases 18% area, while the third one increases 58%. From Fig. 14 and Table 5, the read-port configuration of (3, 2, 2, 2) outperforms the others by about 17% in the term of the energy-delay product, but suffers only 18% area overhead increasing. Therefore this configuration would be a better choice to couple with the proposed register assignment approach.

## 6. CONCLUSION

In this paper we proposed a power-aware register assignment algorithm with the bank-level DVS circuit to save the static power for a multi-banked register file. In this algorithm, we exploit the accessed frequency for each temporary value, and assign those popular temporary values collectively in the first or second banks. Through experiments we found that the other register banks which store less-popular temporary values can save at least half of static energy consumption. This algorithm has a side effect which will cause many access conflicts occurred at those banks storing popular values. We found that adding one more read port to those banks can simply and effectively resolved this problem.

There is one future work related to this study. During register assignment, we found that most popular temporary values are short-lived; that is, they stay in the registers, actually, in very short periods. This means that the register bank storing those values needs not so many registers. If the register bank can be resized just like a reconfigurable architecture, the energy consumption of a register file can be further reduced without performance penalty. Detailed design has been under development.

## ACKNOWLEDGEMENT

I would like to thank anonymous reviewers and Professor Chung-Ping Chung for their kindness help and valuable comments on this paper. The manuscript has been revised carefully according to these comments, which are very significant to improve this work.

## REFERENCES

1. J. Scott, *et al.*, "Designing the low-power M-CORE architecture," in *Proceedings of the Power Driven Microarchitecture Workshop at ISCA98*, 1998, pp. 145-150.
2. S. Palacharla, *et al.*, "Complexity-effective superscalar processors," in *Proceedings of the International Symposium on Computer Architecture*, 1997, pp. 206-218.
3. V. Zyuban and P. Kogge, "The energy complexity of register files," in *Proceedings of International Symposium on Low-Power Electronics and Design*, 1998, pp. 305-310.
4. J. Janssen, *et al.*, "Partitioned register file for TTAs," in *Proceedings of the 28th Microarchitecture*, 1995, pp. 303-312.
5. T. Saito, M. Maeda, *et al.*, "Design of superscalar processor with multi-bank register file," in *Proceedings of IEEE International Symposium on Circuits and Systems*,

- 2005, pp. 3507-3510.
6. R. Balasubramonian, *et al.*, "Reducing the complexity of the register file in dynamic superscalar processor," in *Proceedings of the 34th Microarchitecture*, 2001, pp. 237-248.
  7. J. H. Tseng, *et al.*, "Banked multiported register files for high-frequency superscalar microprocessors," in *Proceedings of the International Symposium on Computer Architecture*, 2003, pp. 62-71.
  8. SimpleScalar, <http://www.simplescalar.com/>.
  9. SPEC2000, <http://www.spec.org/cpu/>.
  10. R. Jones, "Modeling and design techniques reduce 90nm power," <http://www.eetimes.com>, 2004.
  11. K. Fkautner, *et al.*, "Drowsy caches: Simple techniques for reducing leakage power," in *Proceedings of the International Symposium on Computer Architecture*, 2002, pp. 148-157.
  12. N. S. Kim, *et al.*, "Circuit and microarchitectural techniques for reducing cache leakage power," *IEEE Transactions on VLSI*, Vol. 12, 2004, pp. 167-184.
  13. N. S. Kim, *et al.*, "Drowsy instruction caches: Leakage power reduction using dynamic voltage scaling and cache sub-bank prediction," in *Proceedings of the 35th Microarchitecture*, 2002, pp. 219-230.
  14. GCC, <http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>.
  15. I. Keith, *et al.*, "The multicluster architecture: Reducing cycle time through partitioning," in *Proceedings of the 30th Microarchitecture*, 1997, pp. 149-159.
  16. J. L. Cruz, A. González, M. Valero, and N. P. Topham, "Multiple-banked register file architectures," in *Proceedings of the International Symposium on Computer Architecture*, 2000, pp. 316-325.
  17. L. Ayala, *et al.*, "Power-aware compilation for register file energy reduction," *International Journal of Parallel Programming*, Vol. 31, 2003, pp. 151-167.
  18. W. Y. Shieh and C. C. Chen, "Exploiting register-usage for saving register-file energy in embedded processors," *Lecture Notes in Computer Science*, Vol. 3824, 2005, pp. 37-46.
  19. L. Ayala, *et al.*, "Energy aware register file implementation through instruction pre-decode," in *Proceedings of the Application-Specific Systems, Architectures, and Processors*, 2003, pp. 86-96.
  20. D. Brooks, V. Tiwari, *et al.*, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, 2000, pp. 83-94.
  21. L. A. Lozano, *et al.*, "Exploiting short-lived variables in superscalar processors," in *Proceedings of the 28th Microarchitecture*, 1995, pp. 292-302.
  22. W. Y. Shieh and H. D. Chen, "Saving register-file leakage power by monitoring instruction sequence in ROB," in *Proceedings of the International Conference on Embedded System and Application*, 2006, pp. 141-147.
  23. J. Abella and A. González, "On reducing register pressure and energy in multiple-banked register files," in *Proceedings of the 21st International Conference on Computer Design*, 2003, pp. 14-20.
  24. <http://www-device.eecs.berkeley.edu>.
  25. CACTI, [http://www.hpl.hp.com/personal/Norman\\_Jouppi/cacti4.html](http://www.hpl.hp.com/personal/Norman_Jouppi/cacti4.html).



**Wann-Yun Shieh (謝萬雲)** received the B.S. and Ph.D. degrees in Computer Science and Information Engineering from the National Chiao Tung University, Hsinchu, Taiwan in 1996 and 2002, respectively. Since 2003 he has been with the Department of Computer Science and Information Engineering at the Chang Gung University, Taoyuan, Taiwan, where he is an Assistant Professor. His research interests include power-aware architecture design, embedded real-time systems, and system-on-a-chip.



**Shu-Yi Hsu (許書譯)** received the B.S. degree in Information Management from Lung Hwa University in 2004, and the M.S. degree in Computer Science and Information Engineering from the Chang Gung University, Taiwan, in 2006. He is currently a Digital IC Design Engineer at Sunplus Innovation Technology Inc., Hsinchu, Taiwan. His research interests include low power processor architectures, VLSI architecture design and implementation for the BCH and reed-Solomon codec, USB 2.0.