Improving Scalability and Complexity of Dynamic Scheduler through Wakeup-based Scheduling

Kuo-Su Hsiao and Chung-Ho Chen
Department of Electrical Engineering, National Cheng Kung University
No.1, Ta-Hsueh Road, Tainan 701, Taiwan
newjimmy@ee.ncku.edu.tw, chchen@mail.ncku.edu.tw

Abstract

This paper presents a new scheduling technique to improve the speed, power, and scalability of a dynamic scheduler. In a high-performance superscalar processor, the instruction scheduler comes with poor scalability and high complexity due to the inefficient and costly instruction wakeup operation. From simulation-based analyses, we find that 98% of the wakeup activities are useless in the conventional wakeup logic. These useless activities consume a lot of power and slowdown the scheduling speed. To address this problem, the proposed technique schedules the instructions into the segmented issue window based on their wakeup addresses. During the wakeup process, the wakeup operation is only performed in the segment selected by the wakeup address of the result tag. The other segments are excluded from the wakeup operation to reduce the useless wakeup activities. The experimental results show that the proposed technique saves 50-61% of the power consumption, reduces 42-76% in the wakeup latency compared to the conventional design.

1. Introduction

The power consumption and latency of a dynamic scheduler are two of the important issues for designing high performance microprocessors. The wakeup logic contributes the most limiting factors to the dynamic scheduler. In the dynamic scheduler, the wakeup logic traces the instruction dependence and wakes the instructions up when their source operands become available.

In current schedulers, the wakeup logic is implemented by using the content-addressable memory (CAM) that fully matches all the source tags in the issue window with the result tags. However, the CAM structures result in large power consumption and long wakeup latency due to considerable circuit activities and heavy load capacitance.

In an effort to extract more instruction level parallelism, scheduler designs often employ a larger window and aggressive issue width. In other words, scheduler designs employ accordingly larger and more complex wakeup logic that deteriorates the power consumption and wakeup latency.

For the scheduling speed, the complexity of the wakeup logic leads to the major critical path, which limits the clock cycle time, of the pipeline stages. Although a pipelined dynamic scheduler can increase the clock frequency, the operations of instruction wakeup and instruction selection should be an atomic operation to avoid significant performance degradation. Recent study has shown that the latencies associated with the wakeup and selection form the critical path of the pipeline stages [1]. The wakeup latency increases significantly with both the issue width and window size, and the wakeup logic dominates the latency for the scheduler [1]. Increasing the window size will continue to increase the burden to the clock cycle time.

As for the power consideration, the power consumption associated with the CAM-based scheduler constitutes a significant portion of the processor power consumption and may lead to costly cooling system. For example, the issue logic is the most power hungry component of the Compaq Alpha 21464 processor; it is responsible for 46% of the total processor power [2]. Similarly, the out-of-order scheduler of the Intel Pentium 4 processor accounts for 40% of the total power consumption [23]. The wakeup logic dominates the most power consumption of the dynamic scheduler. As a result, the wakeup logic not only slows down the clock speed but also shifts more power budget to the scheduler.

Our analyses reveal that most wakeup activities are unnecessary for the instruction wakeup operations. During the wakeup process, 95% of the wakeup operations wake up only two or less instruction(s) in the issue window as shown in Figure 1. Clearly, most activities of the wakeup operations are useless; however, these activities consume a lot of energy and may lead to a slower clock cycle time.

In this paper, we propose a scheduling technique that improves the power, speed, and scalability of the dynamic scheduler. The proposed technique classifies the issue window into multiple segments according to the wakeup address, which is defined as the high-order bits of the source tags or result tags. Each segment handles the wakeup operations only for the instructions that have the same wakeup address. The instructions that have the same wakeup address are inserted into the same segment. During the wakeup process, the result tag is only matched with the source tags in the segment that is indexed by the wakeup address of the result tag. The other segments are avoided

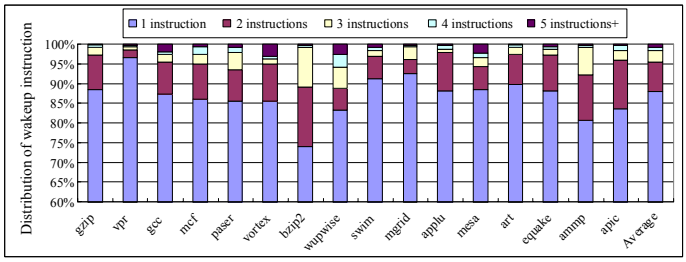

Figure 1: Number of instructions woken up per wakeup operation in a 4-wide 128-entry processor.

from the costly wakeup operations.

The remainder of this paper is organized as follows. Section 2 surveys the wakeup logic of the dynamic schedulers used in superscalar processors and discusses the related wakeup designs. Section 3 details the proposed scheduling technique. Section 4 presents the experimental methodology and the evaluation results. Finally, Section 5 concludes this paper.

2. Background

This section gives the background of the wakeup logics, including the current design and related works.

2.1. Conventional Wakeup Logic

The wakeup logic of a dynamic scheduler is responsible for handling the wakeup operations for the instructions in the issue window. When an instruction is issued for execution, its result (destination) tag is used to wake up the instructions that need the result as their source operand.

The conventional implementation of wakeup logic is based on the CAM structure [1]. This design employs two monolithic CAM structures to match the result tags with the left and right source tags. The nature of the CAM-based wakeup logic is inefficient in terms of energy and latency. The load capacitance on the tag bus line is heavy for driving the long tag bus and all the match circuits of the CAM structure. Additionally, all the match circuits are activated in the wakeup operation no matter it is a match or not. As a consequence, the tag driving and tag matching consume a lot of energy and slow down the wakeup speed.

In an effort to improve the instructions per cycle (IPC), scheduler designs often employ a larger window and aggressive issue width. In other words, a larger window leads to larger wakeup logic and this leads to heavier load capacitances and more match activities. Wider issue width leads to driving more tag buses into the wakeup logic. We can see that increasing the window size and issue width leads to larger power consumption and slower scheduling speed. As a result, the scheduler can not scale well with the increasing of window size and issue width.

2.2. Related Works

There have been many other efforts to reduce the complexity of dynamic schedulers, many of which can be used in combination with our proposed wakeup-based scheduling.

The bank design in [1], which is most similar to our design, segments the monolithic issue window into multiple banks to improve wakeup delay; however the result tags still need to be broadcast to all the banks. This design induces extra wakeup delay and power consumption due to the additional driver-transistors and tag buses.

Hrishikesh et al. proposed a pipelined-wakeup design that segments the issue window and wakes up the instructions in the segments in multiple sequential cycles [3]. Nonetheless, all the segments stills need to be searched; besides the dependent instructions can be issued back to back only if they are in the first segment.

In [4], Folegnani and González presented a gate-off technique that disables the useless (empty and ready) entries of the issue window from tag matching. Besides, Ramirez et al. proposed a similar gate-off mechanism based on a multi-bank issue window to improve the power consumption of the scheduler [5]. However, this mechanism employs an extra large RAM structure that may slow down the wakeup speed.

Several approaches dynamically manage the sizes of the issue window and turn off the useless entries [4] [6] [7] [8] [9] [10]. These designs improve the power consumption of the scheduler with extra dynamic managers that may complicate the scheduler.

Ernst and Austin proposed a tag-elimination scheduler that employs less tag comparators to reduce the complexity of the scheduler. This scheduler also has a last tag speculator to reduce the frequency of tag matching [11]. Based on the same observation, Sharkey et al. presented an instruction-packing technique. This technique schedules the two instructions, which have only one non-available source operand, into the same entry of issue window [12].

Kim and Lipasti proposed a sequential wakeup mechanism to reduce the complexity of scheduler [13]. This mechanism places the last-arrival operand into the fast wakeup logic and wakes up the left and right source operands of an instruction in two sequential steps. Besides, Aggarwal et al. proposed a reduced wakeup width scheduler to reduce the complexity of the scheduler by reducing the maximum number of input result tags for the wakeup logic [14].

The wakeup-free schedulers [15] [16] predict the issue latency of the instructions and then issue the instructions into a FIFO-based issue queue. These schedulers replace the complex wakeup logic with a simple FIFO queue. On the other hand, Brown et al. presents the select-free scheduler [17]. This scheduler removes the instruction-selection process from the scheduling critical path.

Some works [18] [19] [20] employ two-level issue window to reduce the complexity of the scheduler. The critical instructions are dispatched to the small and first issue window and the non-critical instructions, for example: the instruction waiting for a load that misses in cache, are dispatched to the large and slow window.

On the other hand, many wakeup designs employ the custom components instead of the CAM structures. Goshima et al. presented a wakeup design that uses bit matrix structures [21]. Henry et al. presented a cyclic segmented prefix (CSP) circuit to improve the performance of the wakeup logic [22]. We presented a wakeup design, which pre-decodes the source tags and matches the decoded outputs directly with the grant lines, to improve the wakeup speed and power consumption [23]. Ponomarev et al. used three techniques, efficient comparators, 0-B encoding, and bitline segmentation, to reduce the energy dissipation of the issue window [24].

Several designs reduce issue logic complexity through index-based techniques, using pointers to connect the

producer instructions and consumer instructions [25] [26] [27]. Some works reduce the complexity of scheduler by pre-scheduling dependent instructions into a data-flow based issue window [28] [29] [30].

3. Wakeup-based Scheduling

The conventional wakeup design matches the result tags with all the source tags in the issue window only to wakeup a few instructions. It is inefficient in terms of both energy and wakeup speed. To alleviate these useless wakeup operations, the proposed technique schedules instructions into the issue window based on the wakeup address, which is defined as the high-order bits of the source tag or result tag. In this way, during the wakeup process, the result tag is compared with only the source tags that have the wakeup address the same as the wakeup address of the result tag.

The structure of the proposed design is a multi-bank issue window as shown in Figure 2. Each segment is assigned a number as its wakeup address, and the wakeup logic in the segment is the two CAM structures, which are used to handle the wakeup operations for the left and right source operands. To access the segments, the bits of the tag, source tag or result tag, are divided into two parts: index (wakeup address) and reduced tag. The index is the high-order bits of the tag that are used to select a segment. The other part, reduced tag, is the low-order bits of the tag that are used as input for the wakeup logic.

In the example of Figure 2, the 16 segments are numbered from 0-15 (0000-1111$_b$) as their wakeup addresses. The four high-order bits of the source (result) tag are used as the wakeup address to index the segments. Since the high-order bits of the source tag are used as the wakeup address, the tag fields in the wakeup logic for storing the source tags can be reduced to store only the four low-order bits of the source tags. In the same way, the input (result tag) of the wakeup logic is also reduced to the four low-order bits of the result tags.

After rename, the instruction's wakeup address is used as index to select one of the segments. In the selected segment, an entry of the wakeup logic is allocated for storing the four low-order bits of the left and right source tags. In this way, the instructions that have the same wakeup address are dispatched to the same segment.

During the wakeup process, the wakeup operation is performed only in the selected segment. The high-order bits (wakeup address) of the result tag are used to select one of the segments. The low-order bits of the result tag are used as the input for the wakeup logic to match with the source tags in this segment. The source tags in the other segments are not involved in this wakeup operation.

While inserting instructions into the issue window, two situations will lead to that the instruction can not be dispatched to the selected segment. First, the instruction has the two wakeup addresses that index to two different segments. That is, the two wakeup addresses (high-order bits) of the instruction's left and right source operands are not identical to each other. Second, the segment selected by the

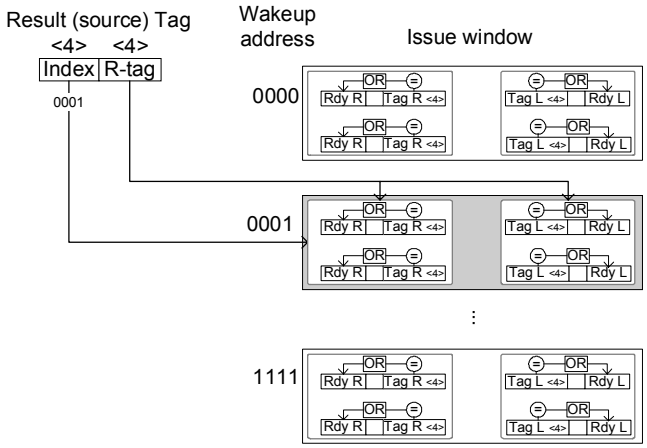

Figure 2: An example of the wakeup-based scheduling design for a 256-entry window.

wakeup address of the instruction is full; no more available entry of this segment can be allocated for the incoming instruction.

We employ an extra backup segment to handle these two situations. The backup segment is the segment that employs the conventional wakeup logic, of which the source tag fields and the bit length of the input tag are not reduced. When the scheduler encounters the instruction that has two different wakeup addresses or the instruction that indexes to a full segment, these instructions are inserted into the backup segment. During the wakeup process, the backup segment is always searched.

In addition, we employ a ready segment for the instructions that come with both their two source operands available. When instructions are inserted into the issue window, some instructions have already both their left and right source operands ready. These instructions are dispatched into the ready segment. Since these instructions are ready for execution, the source tags in the ready segment need not be compared with the result tags during the wakeup process. If there is no available entry for the incoming ready instruction, the scheduler will assign this instruction to the other segments or the backup segment.

Compared to the conventional design, the proposed design has three major advantages: smaller load capacitance on the tag buses, fewer match activities during the wakeup process, and fewer bits for the source tag fields and the result tag inputs. Another advantage of this design is the excellent scalability. No matter what the issue window size is, the wakeup operation is performed only in one segment and the backup segment.

4. Experimental Evaluation and Analysis

This section presents the experimental methodology and discusses the results of performance, power consumption, and latency for the proposed design and some related designs.

Table 1: Processor configurations.

	4-wide	8-wide
Out-of-order Execution	4-wide fetch/issue/commit, 128 RUU, 64 LSQ.	8-wide fetch/issue/commit, 256 RUU, 128 LSQ.
Functional units	4 IALU, 1 IMUL, 2 FALU, 1 FMUL, 2 LSU.	8 IALU, 2 IMUL, 4 FALU, 2 FMUL, 4 LSU.
L1 I-cache L1 D-cache	4-way, 64KB, 32-byte line, 2-cycle latency. 4-way, 64KB, 32-byte line, 2-cycle latency.	
L2 cache TLB	4-way, 512KB, 64-byte line, 10-cycle latency. 4-way, 128-entry, 4KB page size.	
Memory	64-bit wide, 75 cycle latency, 4-cycle burst.	
Branch predictor	Combination of bimodal (2k entries) and 2-level global predictor (2k entries, 8-bit history), 1024-entry chooser, 1024-entry (4-way) BTB, 16-entry RAS (return address stack), 8-cycle penalty.	

4.1. Experimental Methodology

The power consumption and IPC results of the evaluated designs were obtained through architectural simulation, which was conducted by using Wattch [31] and SimpleScalar [32] toolsets. These execution-driven simulators simulate a superscalar processor with two-level caches, branch predictors, dynamic scheduler, and et al. by performing cycle by cycle instruction-level simulation, including execution down any speculative path until a branch misprediction is detected.

Table 1 lists the architectural parameters for the 4-wide and 8-wide superscalar processors. In Wattch, the CAM cell of the evaluated designs was based on the CAM model in [1]. The other configurations for the Wattch include 1GHz clock frequency, 1.8V voltage, and 0.18µm technology process.

The simulation results were collected from seven integer and nine floating point programs of the SPEC2000 benchmark suite. All the selected benchmark programs were compiled with full optimization (-O4). The test input set was used for the benchmark programs. The programs were fast-forwarded the first 50 million instructions and the following 500 million instructions were simulated.

To understand the effects on the wakeup delay, the circuit characteristics of the evaluated designs must be examined. The circuit models were extended from the one proposed by Ernst and Austin [11] and the timing results for the evaluated designs were extracted by using the Avant! Hspice tool. Finally, the CMOS transistors and wires were all conformed to the parameters of the TSMC 0.18µm process.

4.2. Performance Comparison

Figure 3 presents the IPC results of the 4-wide and 8-wide processors that employ different wakeup logics. These results are normalized to the IPC of the baseline processor, which employs the conventional wakeup logic.

The first two bars show the IPC drops of the tag elimination design and the sequential wakeup design. The tag elimination design, configured as 32 two-tag stations, 64 one-tag stations, and 32 zero-tag stations for the 4-wide processor and twice the stations for the 8-wide processor, loses 6% of IPC on average. On the other hand, the IPC drop of the sequential wakeup design is measured to be 6.5% in the 4-wide processor and 12% in the 8-wide processor. Obviously, waking instructions up in two sequential cycles induces non-negligible performance degradation in the wide-issue and large-window processor.

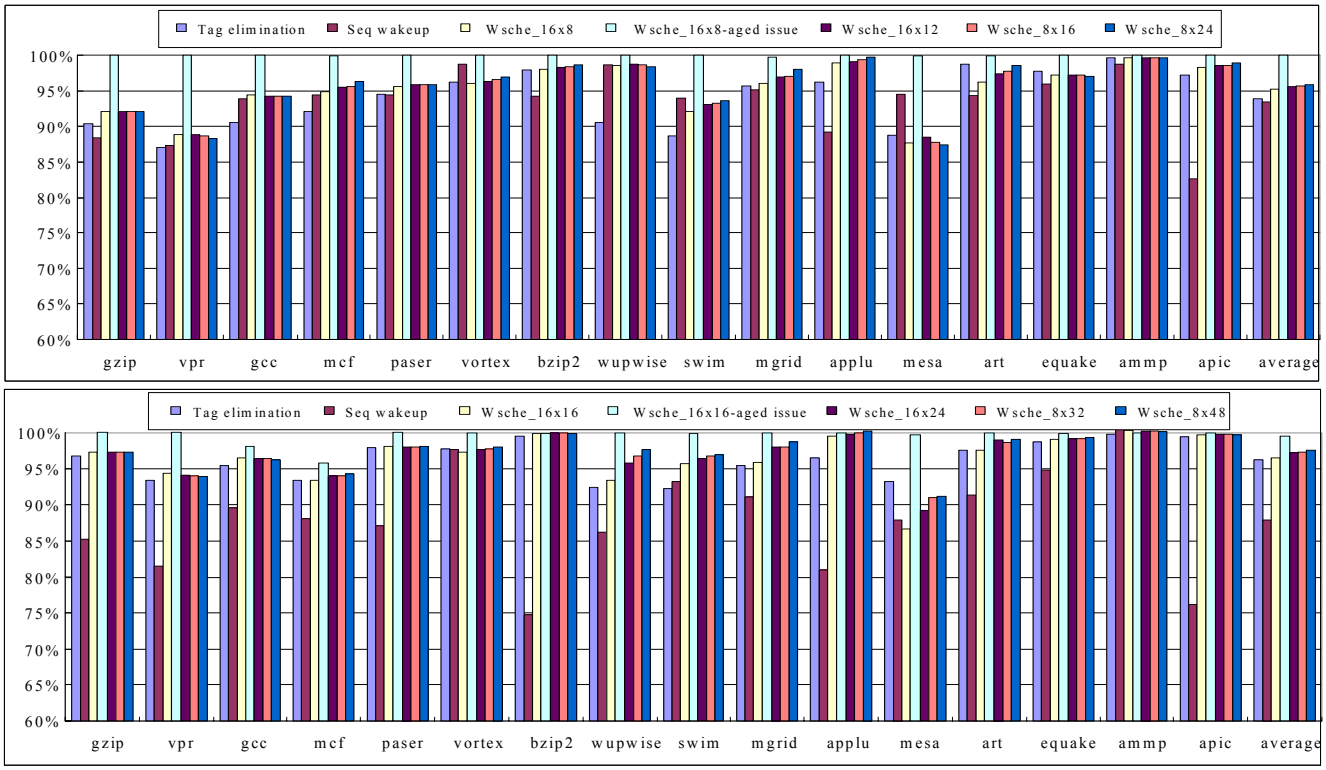

Figure 3: The normalized performance of the evaluated designs for the 4-wide (upper part) and 8-wide (lower part) processors.

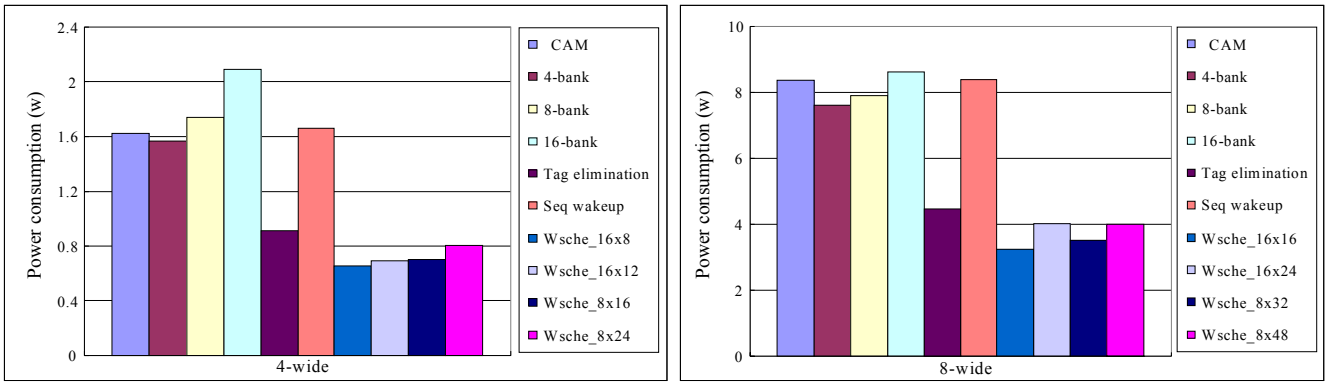

Figure 4: Power consumption of the evaluated designs (w).

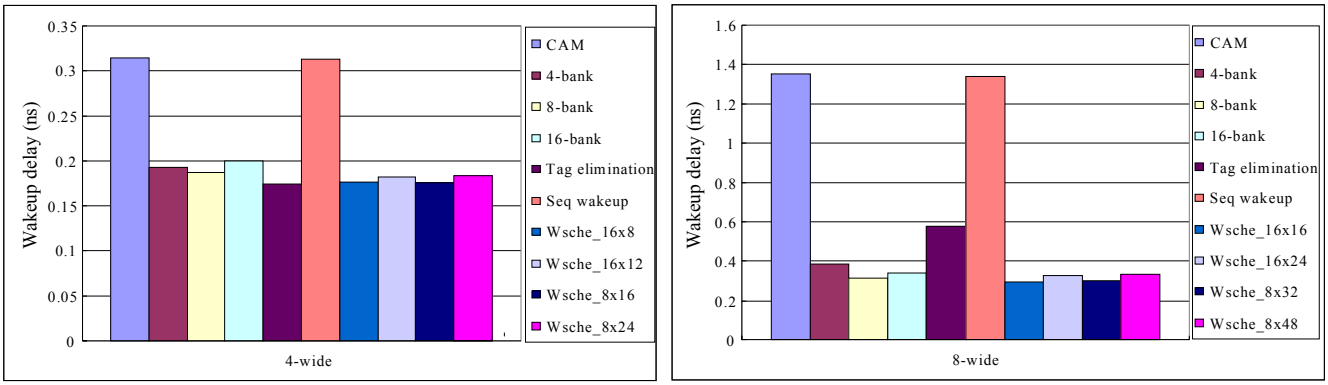

Figure 5: Wakeup latencies of different wakeup approaches (ns).

The other bars show the performance of the proposed design in different configurations. The configuration "$x \times y$" indicates that this scheduler has x segments and each segment consists of y entries. Besides, the backup window and the ready window both have one fourth of x segments.

The performance degradation of the proposed design is measured to be 2.5-5% for the 4-wide and 8-wide processors. This slight IPC drop comes mainly from the issue policy used in the proposed design in which the instruction order is not maintained in the issue window and the select logic can not do an age-based issue easily. Without complicating the select logic, the instructions are issued based on the first-come first-served policy in the proposed design. If the proposed design employs an age-based issue policy, the performance degradation can be decreased within 0.5% as shown in the fourth bars. Besides, the last three bars show that enlarging the capacity of the segments benefits only 1% performance gain; this is because the performance degradation comes mainly from the issue policy but not capacity conflict.

4.3. Power Consumption

Figure 4 presents the power consumption for the wakeup logics in the 4-wide and 8-wide processors. The left most bars show that the conventional wakeup design consumes a lot of power due to the heavy load capacitance and the surplus activities of the monolithic CAM structure.

The followed three bars show the power consumption for the banked wakeup design. Overall, the banked design improves the power consumption by reducing the load capacitance of tag driving in the CAM structures. However, the overhead for the extra tag line and driver transistors becomes significant in the wider-banked design. This overhead deteriorates 16% power consumption of the conventional design in the 4-wide 16-bank issue window.

The power consumption of the tag elimination and sequential wakeup designs is shown in the fifth and sixth bars. The configuration of the tag elimination design is equivalent to half of the entries of the conventional scheme; thus this design saves 44-48% power consumption of the conventional design. In contrast, although the sequential wakeup design wakes up instructions in two phases, this design still drives two monolithic CAM structures as the conventional design does. It is measured that the sequential wakeup design improves only 7-8% power consumption.

The power consumption of the proposed design with different configurations is shown in the last four bars. It is measured to be only 39-50% that of the conventional CAM scheme. This excellent energy saving comes from the wakeup-based scheduling. Most needless wakeup operations are avoided; thus this design is highly efficient in terms of energy usage. Obviously, the proposed design is more energy saving than other designs.

4.4. Wakeup Latency

Figure 5 shows the wakeup latencies of the evaluated wakeup designs. The wakeup latencies of the conventional design are presented in the first bars. Compared to the conventional design, the banked design improves 36-40% wakeup latency in the 4-wide processor and improves 72-

77% wakeup latency in the 8-wide processor. This improvement comes from that the smaller banked segments performs wakeup operations in parallel. For the similar reason, the tag elimination design improves 57% wakeup latency of the conventional design for the 8-wide 256-entry processor. In the 4-wide 128-entry processor, the improvement is not as significant as that in the 8-wide processor. In contrast to the banked design and tag elimination design, the sequential wakeup design has almost the same wakeup latency as that of the conventional design because the critical path of the sequential wakeup design is the same to that of the conventional design.

The proposed design also performs much faster than the conventional design. Compared to the banked design, the proposed design is measured to be 6% faster in both the 4-wide and 8-wide processors. This advantage comes from that only necessary segments are activated during the wakeup process. Compared to the tag elimination design, the proposed design is about 50% faster in the 8-wide processor. We can observe that the proposed design suits for sophisticated scheduler in the wide-issue large-window processor.

5. Conclusion

The latency and power consumption of the dynamic scheduler become the important problems for designing a high-performance superscalar processor. Most of the latency and power consumption of the scheduler result from the complex wakeup logic. In this paper, we propose a wakeup-based scheduler that schedules the instructions into the issue window based on their wakeup addresses. During the wakeup process, only the segment, which has the same wakeup address of the result tags, is involved for wakeup operation. This technique avoids most of the useless and inefficient wakeup operations. The proposed design has the following advantages: smaller load capacitance on the tag buses, fewer match activities during wakeup process, reduced bit length of tag fields and for the input of the wakeup logic. Another advantage of this design is the excellent scalability. No matter what the issue window size is, only one segment and the backup segments are activated during the wakeup process. The proposed design improves 50-61% power consumption and 42-76% wakeup latency of the conventional design with only 3-5% IPC degradation. In conclusion, the proposed wakeup design removes the complexity from the dynamic scheduler and enables the processor to employ a more sophisticated scheduler.

Acknowledgments

This work was supported in part by the National Science Council, Taiwan under Grant No. NSC 94-2220-E-006-008.

References

[1] S. Palacharla, N. P. Jouppi, and J. E. Smith, “Quantifying the Complexity of Superscalar Processors,” University of Wisconsin-Madison, Tech. Rep. CS-1328, May 1997.

[2] K. Wilcox and S. Manne. “Alpha processors: A history of power issues and a look to the future,” Cool Chips Tutorial, 32nd Annu. Int. Symp. Microarchitecture, Nov. 1999.

[3] M. S. Hrishikesh, N. P. Jouppi, and K. I. Farkas, “The optimal useful logic depth per pipeline stages is 6-8 FO4,” in Proc. ISCA, May 2002, pp. 14-24.

[4] D. Folegnani and A. Gonzalez, “Energy-Effective Issue Logic”, in Proc. ISCA, Jul. 2001, pp. 230-239.

[5] M. A. Ramírez et al., "A Simple Low-Energy Instruction Wakeup Mechanism," in International Symposium on High-Performance Computing (ISHPC), Oct. 2003, pp. 99-112.

[6] D. Ponomarev, G. Kucuk, and K. Ghose, “Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources,” in Proc. MICRO, Dec. 2001, pp. 90-101.

[7] J. Abella and A. González, “Power-Aware Adaptive Issue Queue and Register File,” in Proc. Int. Conf. High-Performance Computing (HiPC), Dec. 2003.

[8] David H. Albonesi. “Dynamic IPC/Clock Rate Optimization,” in Proc. ISCA, June 1998, pp. 282–292.

[9] A. Buyuktosunoglu et al., ”A Circuit Level Implementation of an Adaptive Issue Queue for Poweraware microprocessors,” in Proc GLVSLSI, Mar. 2001, pp. 73-83.

[10] S. Dropsho et al., “Integrating Adaptive On- Chip Storage Structures for Reduced Dynamic Power,” in Proc. 11th Parallel Architectures and Compilation Techniques, Sep. 2002, pp. 141-152.

[11] D. Ernst and T. M. Austin, “Efficient dynamic scheduling through tag elimination,” in Proc. ISCA, May 2002, pp. 37-46.

[12] J. J. Sharkey et al., “Instruction packing: reducing power and delay of the dynamic scheduling logic,” in Proc. ISLPED, Aug. 2005, pp. 30-35.

[13] I. Kim and M. H. Lipasti, “Half-Price Architecture,” in Proc. ISCA, Jun. 2003, pp. 28-38.

[14] A. Aggarwal, et al., “Defining Wakeup Width for Efficient Dynamic Scheduling,” in Proc. ICCD, Oct. 2004, pp. 36-41.

[15] D. Ernst, A. Hamel, and T. Austin, “Cyclone: A Broadcast-Free Dynamic Instruction Scheduler with Selective Replay,” in Proc. ISCA, Jun. 2003, pp. 253-262.

[16] J. Hu, N. Vijaykrishnan, M. Irwin, “Exploring Wakeup-Free Instruction Scheduling,” in Proc. HPCA, Feb. 2004, pp. 232-241.

[17] M. Brown, J. Stark, Y. Patt. “Select-Free Instruction Scheduling Logic,” in Proc. MICRO, Dec. 2001, pp. 204-213.

[18] A.R. Lebeck et al., “A Large, Fast Instruction Window for Tolerating Cache Misses,” in Proc. ISCA, May 2002, pp. 59-70.

[19] B. Fields, S. Rubin, and R. Bodík, “Focusing Processor Policies via Critical-Path Prediction,” in Proc. ISCA, Jul. 2001, pp. 74-85.

[20] E. Brekelbaum et al., “Hierarchical Scheduling Windows,” in Proc. MICRO, Nov. 2002, pp. 27-36.

[21] M. Goshima et al., “A High-Speed Dynamic Instruction Scheduling Scheme for Superscalar Processors,” in Proc. MICRO, Dec. 2001, pp. 225-236.

[22] D. S. Henry, B. C. Kuszmaul, G. H. Loh, and R. Sami, “Circuits for Wide-Window Superscalar Processors,” in Proc. ISCA, Jun. 2000, pp. 236-247.

[23] K. S. Hsiao and C. H. Chen, "An Efficient Wakeup Design for Energy Reduction in High-Performance Superscalar Processors," in Int. Con. Computing Frontiers (CF), May 2005. pp. 353-360.

[24] D. V. Ponomarev et al., “Energy-Efficient Issue Queue Design,” in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 11, Oct. 2003, pp. 789-800.

[25] M. Huang, J. Renau, and J. Torrellas, “Energy-Efficient Hybrid Wakeup Logic,” in Proc. ISLPED, Aug. 2002, pp. 196-201.

[26] R. Canal and A. González, “A Low-Complexity Issue Logic,” in Proc. ICS, May 2000, pp. 327-335

[27] R. Canal and A. Gonzalez, “Reducing the Complexity of the Issue Logic,” in Proc. ICS, Jun. 2001, pp. 312-320.

[28] S. Palacharla, N. P. Jouppi, and J. E. Smith, “Complexity-effective superscalar processors,” in Proc. ISCA, Jun. 1997, pp. 206-218.

[29] P. Michaud and A. Seznec, “Data-flow prescheduling for large instruction windows in out-of-order processors,” in Proc. HPCA, Jan. 2001, pp. 27-36.

[30] S. E. Raasch, N. L. Binkert, and S. K. Reinhardt, “A Scalable Instruction Queue Design Using Dependence Chains,” in Proc. ISCA, May 2002, pp. 318-329.

[31] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A framework for architectural-level power analysis and optimizations,” in Proc. ISCA, Jun. 2000, pp. 83-94.

[32] D. Burger and T. M. Austin,”The SimpleScalar tool set, version 2.0,” University of Wisconsin-Madison, Tech. Rep. CS-1342, Jun. 1997.