International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622
NATIONAL CONFERENCE on Developments, Advances & Trends in Engineering Sciences
(NCDATES- 09th & 10th January 2015)

RESEARCH ARTICLE OPEN ACCESS

Low Complexity Out-of-Order Issue Logic Using Static Circuits

[1] Mr.P.Raji Reddy, [2]Mrs.Y.Saveri Reddy & [3]Dr. D. R. V. A. Sharath Kumar
[1,3] ECE Dept Malla Reddy College of Engineering & Technology, HYD,
[2] ECE Dept,CMR College of engineering, HYD.

Abstract—
In this paper a single-cycle issue queue circuit architecture that simplifies the wakeup and selection logic is proposed. The micro-archi- tecture and fully static CMOS circuits are presented for a 32-entry queue that issues four instructions per cycle. The instruction-ready signals are di- vided into groups and processed in parallel to issue the four oldest ready instructions. The complete issue queue and prioritization logic requires 20 inversions, allowing simulated circuit operation at over 4 GHz in a foundry 45 nm SOI fabrication process.

Index Terms—CMOS digital integrated circuit, issue queue, microprocessor, out-of-order instruction issue, superscalar.

I. INTRODUCTION

Microprocessor instruction streams contain instructions that can potentially execute in parallel. This instruction level parallelism (ILP) is the foundation of superscalar processing. ILP provides a consider- able gain in instructions per cycle (IPC). With the ability to look across multiple instructions in the issue window, out-of-order execution significantly improves IPC over in-order execution. However, this performance comes at a power and complexity price.

A. Superscalar Pipeline

A typical superscalar pipeline is as shown in Fig. 1(a). In-order and speculation techniques occupy different sections of the complete pipeline. Dynamic scheduling lies between the decode and execute stages, eliminating false dependencies through register-renaming and reducing pipeline inefficiencies due to data dependencies through superscalar out-of-order execution. To maintain precise exception behavior the commit stage forces in-order commitment of results to the machine architectural state. This paper presents a simple, low power design for the critical issue stage [1], selecting the four oldest ready instructions.

B. Instruction Issue Logic

Selection of highest priority ready instructions requires a buffer to store instructions, a dependency tracking mechanism to generate ready signals (ready indicates that a given instruction inputs will be valid in the next cycle) and a mechanism to pick instructions according to a se- lection policy. These requirements are fulfilled by the instruction issue logic that has wakeup and selection logic blocks as shown in Fig. 1(b).

C. Paperrganization

Section II briefly discusses the prior related work. The proposed architecture is described in Section III, focusing on the micro-architectural organization and circuit design. Section IV covers the performance evaluation and simulation results. Section V concludes the paper.

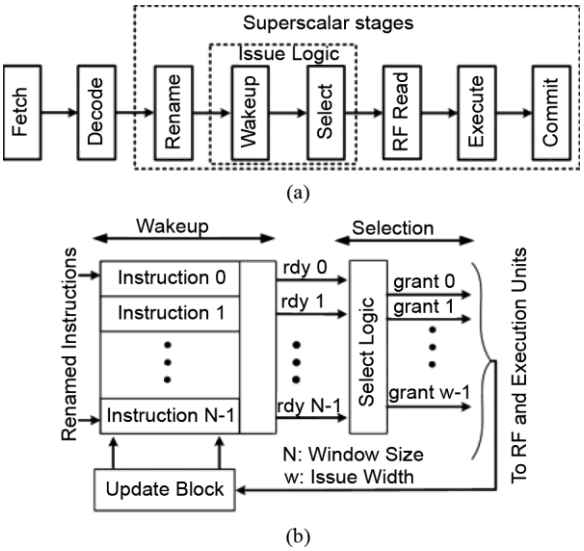

Fig. 1. Superscalar pipeline (a) showing where the issue logic resides. (b) Instruction issue logic high level functional diagram.

The wakeup logic is comprised of a queue that stores the renamed instruction registers, tracks their dependencies and generates ready signals based on the which dependencies' results are ready in the next clock. The selection logic prioritizes the ready instructions for issue. The update logic then accepts new instructions into the

queue in the next cycle, maintaining the ordering of existing entries and opened slots with new instructions at the clock edge.

II. RELATED WORK

A. Complexity and Single vs. Distributed Queues

The instruction issue logic performance is quantified in terms of its critical timing path. Palacharla *et al.* analyzed the impact of issue width and window size on the complexity of wakeup and selection logic [2] where since dependent instructions cannot be simultaneously executed, they were distributed heuristically into first-input first-output (FIFO) buffers. Only instructions at the head of each buffer are considered for issue. The IBM Power4 design utilizes 11 single issue specialized queues [3]. Vangal *et al.* also distributed the issue window with two single issue, eight-entry instruction schedulers [4] where to enable fast parallel execution, complementary signal generation (CSG)-based ready and select logic was used, creating an inherent timing race condition requiring extensive manual circuit validation, i.e., design effort. Distributed windows reduce performance and require more entries to achieve the same IPC as a centralized window due to underutilization [5].

B. Speed and IPC Impact

Prioritizing ready instructions accounts for more than half of the latency of an issue queue [6] and so must be comprehended by any scheme. Stretching the issue logic operation loop over two or three clock cycles incurs an IPC loss of 10% or 19%, respectively. Oldest- first selection gives an IPC benefit of up to 8% over a random position-based scheme and provides better instruction sequencing [7]. Farrell *et al.*, used a compacting register scoreboard to preserve the temporal order of the queue for the Alpha 21264 [8] at the cost of significant data movement. This design used a dynamic tree-based re- quest-grant arbitration scheme for oldest-first selection, ordering en- tries in the queue by age.

C. Power Dissipation

The issue logic is a significant component of the overall power consumption, e.g., in the Alpha 21264, 18% of the total power was dissipated in the all dynamic logic issue queues [9]. Bahar *et al.* asserted that the arbiters in the 21264 [8] account for around 35% of the total processor power when using a two arbiter scheme [10]. Goshima *et al.* claimed dependence detection in wake-up logic is similar to register renaming dependency detection [11] and proposed scheduling using matrices instead of content addressable memory (CAM) to track instruction dependencies. Though matrix functions are faster and dis- sipate less power than CAM-based operations, the matrix nature limits their practical size [12]. Sassone *et al.* proposed a modified matrix scheduler to improve the scalability of this scheme, based on the observation that wakeup and select matrices are sparse [12].

D. Sort-Based Issue Logic

To overcome the complexity of tree-based schemes and improve the cycle time while minimizing power, sort-based issue prioritization logic [13] provides a comparison point for the issue queue design that this brief proposes. In [13] the ready generation follows that used in [8] in that it uses a scoreboard and oldest to newest instruction ordering. The priority selection logic uses multiple odd-even merge sorting net- works to select four oldest ready instructions from the issue queue. Except for the scoreboard based issue circuits the design only uses static CMOS gates. The ready instructions are selected in parallel by sorting them in small groups, resulting in manageable sorting network depth. The results of these group selects are then prioritized to determine the overall oldest four instructions. The shift logic utilizes small barrel shifters to control scoreboard compaction.

III. PROPOSED ISSUE QUEUE MICRO-ARCHITECTURE

The proposed issue queue uses a static CAM to track dependencies between instructions. Instructions are shifted to keep the oldest at the top to provide easy prioritization as in [3], [8] . Fig. 2 shows the signal flow for the proposed shift based issue logic. The select logic is implemented primarily using shifters, with simple issue count logic and shift/grant logic. As opposed to a dynamic scoreboard, the static CAM

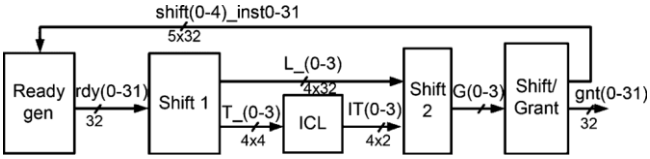

Fig. 2. Signal flow for shift-based issue logic. Simplifications by cascading

shift-based priority logic are evident by the removal of the one-hot conversion, output multiplexing, and decode stages

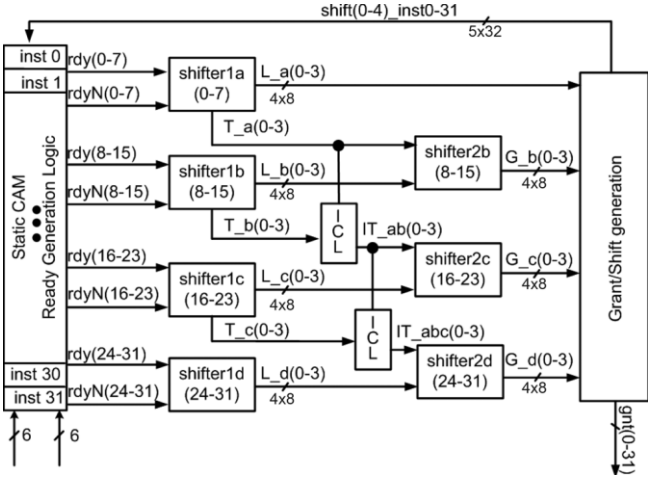

Fig. 3. Overall micro-architecture of the proposed issue queue.

Minimizes instruction wakeup power while the low complexity static shifters and multiplexers maximize the performance by reducing the critical timing path while minimizing power. The CAM compares currently executing operation destination tags
with pending operation source tags, setting latched ready signals. Since the CAM window shifts instructions before accepting new instructions, oldest instructions are always at the top of the queue, i.e., instruction 0 has highest priority while instruction 31 has the lowest priority (see Fig. 3). The select logic selects the four highest priority, ready instructions by checking if three or less prior (higher priority) instructions are ready to issue before each ready instruction. If this condition is satisfied, the ready instruction grant signal is asserted, i.e., issued. The clock cycle begins by comparing the destination tags with the source operands of each pending operation in the static CAM. If both the operands of an instruction are ready and the instruction is valid, the
ready signal, which is the output of a latch, is set for that entry. Ready signals, *rdy(031)* are forwarded to the select logic to generate grants and shifter controls for compaction.
The ready signals are divided into four groups each processing eight entries in parallel (see Fig. 3). The first shifter one-hot outputs *L_ad(0-3)* for each instruction indicate the number of entries ready to issue prior to the current one (above it) within the local (L) group, labeled by suffixes a-d. The first shifter circuit (e.g., shifter1a) one hot outputs *T_a-d(0-3)* indicate the total number of ready instructions in the a group. The issue count logic, labelled ICL combines the total ready instructions to calculate the multiple-group totals (signals starting with IT) to generate the global issue signals *G_a-d(0-3)*. The *G* signal generation in terms of *L* and *T* follows:

$$G\_a(x) = L\_a(x) \quad (1)$$
$$G\_b(x) = T\_b(x) + L\_b(x) \quad (2)$$
$$G\_c(x) = T\_a(x) + T\_b(x) + L\_c(x) \quad (3)$$
$$G\_d(x) = T\_a(x) + T\_b(x) + T\_c(x) + L\_d(x) \quad (4)$$

that are calculated in shifter2. This shifter basically sums the and terms. An instruction is granted if it is ready and the total number of ready instructions before it is less than four, i.e., *G_a<4*. The grant signals and shift signals set up to the clock rising edge. To illustrate the signal flow for a specific case, consider for example, if instructions in issue queue locations 1, 2, 8, 11, and 27 are ready. *L_a(0-3)=0010* indicating two instructions are ready. T_a(0-3)=0010 indicating two instructions are ready from group a. T_b(0-3)=0010 as two instructions, 8 and 11 are ready from group b. instructions 12–15 since more than three instructions are ready before them. Finally, depending on whether a given entry is ready and the total number of instructions ready before it, *G(0-3)* , grant and shift signals are generated for each instruction.

A. Static Wakeup CAM Logic

The CAM shown in Fig. 4 is fully static and overcomes scoreboard limitations, particularly high power dissipation, with negligible performance impact. The CAM storage is separated from the CAM compare in order to accommodate four simultaneous searches, and uses static CMOS shift and update circuits. CAMstorage for each instruction consists of two encoded source operands that are 6-bits, and one valid bit. The storage/update circuit of the valid bit is similar to the CAM store circuitry. The shift(0–4) signals arrive before the rising edge of the clock to update the CAM opposed to a dynamic scoreboard, the static CAM minimizes instruction wakeup power while the low complexity static shifters and multiplexers maximize the performance by reducing the critical timing path while minimizing power. The CAM compares currently executing operation destination tags with pending operation source tags, setting latched ready signals. Since the CAM window shifts instructions before accepting new instructions, oldest instructions are always at the top of the queue, i.e., instruction

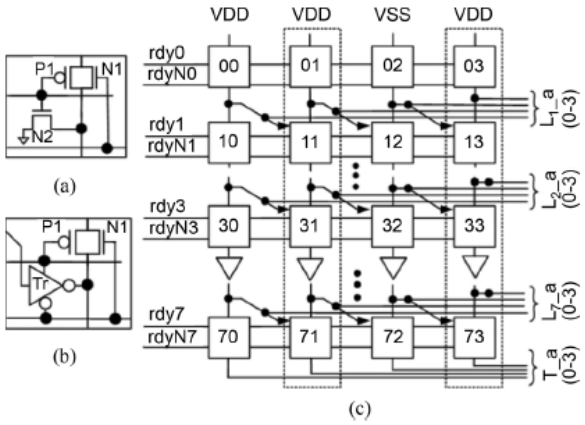

Fig. 5. First shifter, (a) first column cell, (b) shifter cell, (c) overall circuit architecture. The outlined columns use inverted logic.

B. Shifter-Based Select and Update

The all static CMOS select logic is implemented using a first level shifter, issue count logic, a second level shifter and shift grant generator logic. As mentioned, the ready signals from the wakeup logic are divided into four groups of 8-instructions to parallelize the prioritization. *1) First Shifter Based Priority Stage:* The four first stage prioritization shifters, shifter1a through shifter1d, each handle eight sequential instruction-ready signals. The first column shift unit cell circuit is shown in Fig. 5(a) and others in Fig. 5(b). To determine if three or less instructions are ready above the current one, four columns are used as shown in Fig. 5(c). The layout of each of these corresponding blocks is shown in Fig. 6. Columns 1 and 3 are driven with inverted inputs, allowing inverters rather than buffers in the basic cell to limit the total Number of inversions. The column inputs "1101" indicate zeros instructions are ready before the first instruction. The outputs *L_a—d(0-3)* Indicate the number of instructions that are ready to be issued within the group. If four or more instructions are ready, *L(0-3)=0000*. The Shift1 block operation is depicted in Fig. 7(a). The first column shows ready instructions in the group of eight. The first row has a hard- coded value as "1000". These values will pass to next row if the subsequent ready signal is "0" else will shift right (see Fig. 7(a) for the logical, and Fig. 7(b) with actual signal polarities as implemented to limit in- versions).

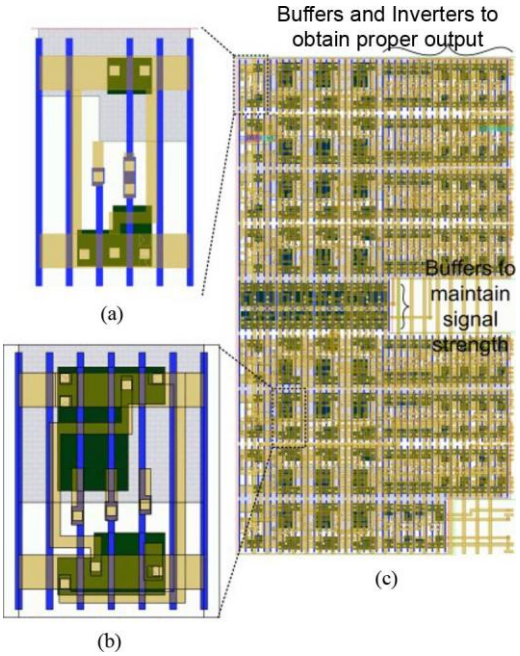

Fig. 6. Layout of the shifter1 structure implemented on the 45 nm foundry process. Details of the gates are shown to the left (a) and (b), while a full entry is shown to the right. (a) First column cell. (b) Static shifter cell. (c) Overall layout of the shifter block.

2) Second Shifter and Issue Count Logic (ICL):

The second shifter combines the local outputs with the number of instructions ready in previous groups to determine the total number of ready instructions before the current instruction. The input to shifter2d is the *L_d(0-3)* for instructions 24 to 31 and sum of the number of instructions ready in groups a, b, and c The ICL calculates number of instructions ready in previous groups using static combinational logic in two inversions

Two ICL blocks reside in the critical path to shifter2d that produces the aggregate number of instructions ready in all of the previous groups in four inversions. Fig. 8 shows the second shifter circuit, again implemented to minimize signal inversions and delay. AND gates drive an nMOS transistor, pulling the output to ground and ensuring that the output is always strongly driven.

3) Shift/Grant Generation Logic:

The shift/grant generator is a two inversion combinational circuit. If the instruction is ready and the number of instructions ready before it, i.e., *G(03)* , is less than or equal to three, grant *(gnt)* for the instruction is set high. The number of shifts that a particular instruction should undergo is equal to the number of instructions granted before it, one hot encoded as. *G(03)* If all previous ready signals are zeroes, it implies that more than four

instructions are ready before the current instruction. This is indicated by the logical OR of complements of previous ready signals. In this case, the output is asserted active low.

IV. PERFORMANCE EVALUATION

The complete design of the proposed and the sort based issue queue in [13] was carried out using a foundry high performance SOI 45-nm CMOS process. Simulations were carried out with V_{DD} = 1 V using Cadence Ultrasim. Key blocks were laid out (see Fig. 6) and other in- terconnects use estimated wire-loads.

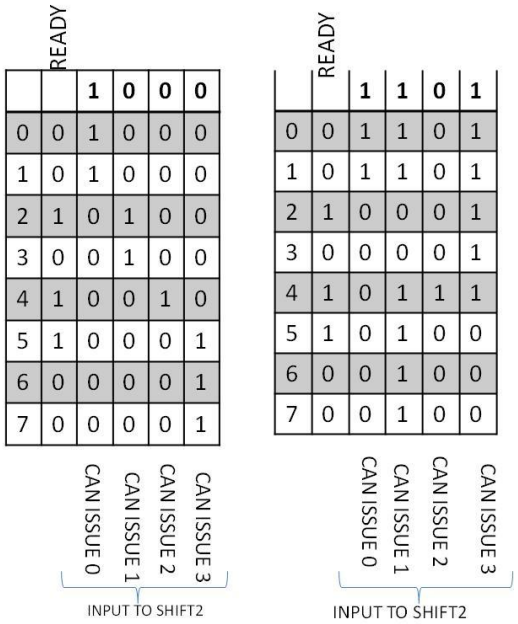

Fig. 7. First shifter operation details. (a) The logical flow of ready signals and (b) shows actual implemented flow with inverted logic for the outlined columns.

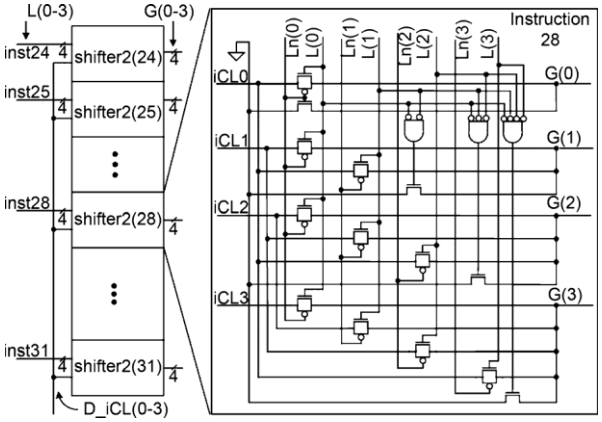

Fig. 8. Second shifter (shifter2) logic to obtain the signals that drive that control the queue entry compaction multiplexers.

A. Worst-Case Timing Path Simulation

The sorter based issue queue [13] requires 30 inversions Including the latch t_{d2q} and t_{setup} times. The proposed shifter-based issue logic design requires 20 inversions. Fig. 9 shows the critical path timing simulation for the shift based design. The shift inputs to the CAM storage multiplexer setup 30 ps before the rising edge of the clock to update the CAM with source operand data from one of the five instructions. The rising edge triggered flip-flop outputs the data to the comparator after a t_{clk2q} delay of 18 ps. The comparator compares the source and destination tags and generates a source match signal after 24 ps (three low fan-out inversions). The source match and previous ready information is combined to generate the instruction ready signal 56 ps after the clock rising edge.

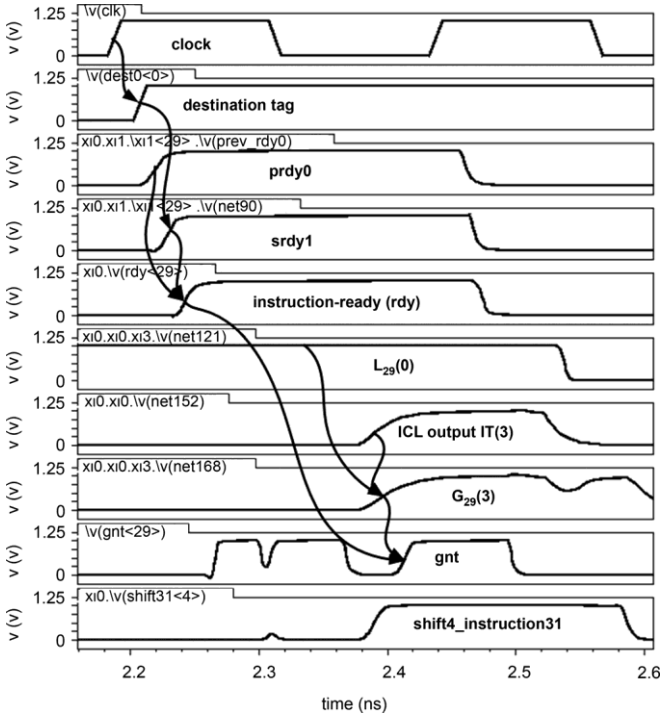

Fig. 9. Simulated waveforms of the proposed issue queue using shifter-based priority selection logic.

The ICL outputs *IT_abc(0-3)* are 151 ps after the worst-case ready signal assertion and are fed to the second shifters that combine it with *L(03)* for each instruction to generate total number of instructions ready before the current instruction, *G(03)* for each instruction. The *G(03)* signals and ready signals for an instruction drive the shift/grant generator to generate grant signals for execution units after another 14 ps and one-hot shift signals for the CAM storage multiplexer. The last grant signal, *gnt(31)*, is generated 227 ps after rising edge of the clock for the worst case critical path. This provides

sufficient setup time to allow a 4 GHz clock rate, assuming reasonable clock skew.

The sort-based design [13] dissipates over 5x *more* energy per cycle than this static shift based design (see Table I). The area of the design proposed here is also considerably smaller, primarily due to fewer transistors. The proposed issue queue is compared against other issue queue architectures in Table II. While implementations are on differing fabrication technologies, clock frequencies and areas are normalized to the 45-nm technology node using standard scaling values. A higher clock frequency can be obtained with a lower issue width per queue, e.g., the CSG design [4] and Power4 [3], as opposed to the Alpha implementation [8]. However, the IWB implementation achieves a higher clock frequency together with larger issue width by sacrificing window size.

POWER AND AREA COMPARISON

Our proposed issue queue circuit architecture provides a unified queue with good window and issue width at adequate clock rates for most modern, i.e., power limited, CPUs.

Design		Energy (pJ/clock)	Area (no. of transistors)
Sort-Based	Select & Update	0.40	9152
	Overall	6.31	224160
Shift-Based	Select & Update	0.23	5044
	Overall	1.15	35396
Improvement (%)	Select & Update	42.5	44.9
	Overall	81.7	84.2

TABLE II
ISSUE WIDTH AND WINDOW SIZE
FOR DIFFERENT ARCHITECTURES

Design	Issue Width/ queue	Max window size	Cycles per issue	Process node (nm)	Scaled Clk rate (GHz)	Clk rate (Hz)
21264 [8]	2	20	1	350	4.8	600 M
Intel CSG [4]	1	8	1	130	14.3	5G
IWB [14]	4	64	3	180	7.2	1.8G
Power4 [3]	1	12	1	180	5.2	1.3G
Sort-based [13]	4	32	1	45	3.2	3.2G
Shift-based	4	32	1	45	4.1	4.1G

As mentioned, in order to overcome scaling and portability issues, the proposed design employs only static CMOS gates. Thus the pro- posed design is amenable to auto place and route methods, if not full synthesis. While the shifter block was implemented as a regular embedde block, its layout was also accomplished using a commercially available APR tool (Encounter).

The use of the static CMOS gates also allows the use of conventional timing tools for the timing analysis of the design. Timing analysis of the shifter block was carried out using Primetime, with results consistent with the circuit simulations.

V. CONCLUSION

An oldest-first priority, 32 entry issue queue that divides the instruction ready signals into groups and selects the four highest priority instructions has been described. By processing ready signals in parallel, the complexity is reduced and select operations are completed in a single cycle. All logic is static CMOS, and can be clocked at 4 GHz in the target foundry 45-nm SOI process at the typical process conditions, with an energy consumption of 1.15 pJ per cycle at 1 V. The design is amenable to auto place and route, as well as static timing analysis, enhancing portability. The circuits are, of course, applicable to distributed issue queues.

[1] J. Hennessey, D. Patterson, and A. Arpaci Dusseau, *Computer Archi- tecture: A Quantitative Approach*, 4th ed. San Mateo, CA: Morgan Kaufmann, 2006.

[2] S. Palacharla, N. Jouppi, and J. Smith, "Complexity-effective super- scalar processors," in *Proc. 24th Annu. Int. Symp. Comput. Arch.*, 1997, pp. 206–218.

[3] T. N. Buti, R. McDonald, Z. Khwaja, A. Ambekar, H. Le, W. Burky, and B. Williams, "Organization and implementation of the register re- naming mapper for out-of-Order IBM Power4 processors," *IBM J. Res. Develop.*, vol. 49, no. 1, pp. 167–188, Jan. 2005.

[4] S. Vangal, N. Borkar, E. Seligman, V. Govindarajulu, V. Erraguntla, H. Wilson, A. Pangal, V. Veeramachaneni, M. Anders, J. Tschanz, Y. Ye, D. Somasekhar, B. Bloechel, G. Dermer, R. Krishnamurthy, S. Narendra, M. Stan, S. Thompson, V. De, and S. Borkar, "A 5 GHz32b integer-execution core in 130 nm dual-VT CMOS," in *ISSCC'02 Dig. Tech. Papers*, 2002, pp. 412–413.

[5] M. Johnson, *Superscalar Microprocessor Design*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

[6] M. Brown, J. Stark, and Y. Patt, "Select-free instruction scheduling logic," in *Proc. 34th Annu. Int. Symp. Microarch.*, 2001, pp. 204–213.

[7] A. Buyuktosunoglu, A. El-Moursy, and D.

Albonesi, "An oldest-first selection logic implementation for non-compacting issue queues," in *Proc. ASIC/SOC Conf*, 2002, pp. 31–35.

[8] J. A. Farrell and T. C. Fischer, "Issue logic for a 600-MHz out-of-order execution microprocessor," *IEEE J. Solid-State Circuits*, vol. 33, no. 5, pp. 707–712, May 1998.

[9] R. Kessler, E. McLellan, and D. Webb, "The alpha 21264 micropro- cessor architecture," in *Proc. Int. Conf. Comput. Design: VLSI Comput. Processors*, 1998, pp. 90–95.

[10] R. Bahar and S. Manne, "Power and energy reduction via pipeline bal- ancing," in *Proc. Int. Symp. Comput. Arch.*, 2001, pp. 218–229.

[11] M. Goshima, K. Nishino, Y. Nakashima, S. Mori, T. Kitamura, and S. Tomita, "A high-Speed dynamic instruction scheduling scheme for superscalar processors," in *Proc. Int. Symp. Micro-arch.*, 2001, pp.225–236.

[12] P. Sassone, J. Rupley, E. Brekelbaum, G. Loh, and B. Black, "Ma- trix scheduler reloaded," in *Proc. Int. Symp. Comput. Arch.*, 2007, pp. 335–346.

[13] S. Mhambrey, L. Clark, S. Maurya, and K. Berezowski, "Out of order issue logic using sorting networks," in *Proc. 20th Great Lakes Symp. VLSI*, 2010, pp. 385–388.

[14] J. Leenstra, J. Pille, A. Mueler, W. Sauer, and D. Wendel, "A 1.8-GHz instruction window buffer for an out-of-order microprocessor core," *IEEE J. Solid-State Circuits*, vol. 36, no. 11, pp. 1628–1635, Nov. 2001