Mixed-Clock Issue Queue Design for Energy Aware, High-Performance Cores

Venkata Syam P. Rapaka
Mentor Graphics Corp.
Wilsonville, OR 97070
e-mail: shyam_rapaka@mentor.com

Emil Talpes, Diana Marculescu
Carnegie Mellon University
Dept. of Electrical and Computer Engineering
Pittsburgh, PA 15213
e-mail: {etalpes,dianam}@ece.cmu.edu

Abstract - Globally-Asynchronous, Locally-Synchronous (GALS) design style has started to gain interest recently as a possible solution to the increased design complexity, power and thermal costs, as well as an enabler for allowing fine grain speed and voltage management. Due to its inherent complexity, a possible driver application for such a design style is the case of superscalar, out-of-order processors. This paper proposes a novel mixed-clock issue queue design, and compares and contrasts this new implementation with existing synchronous or mixed-clock versions of issue queues, used in stand-alone mode or in conjunction with mixed-clock FIFO (First-In, First-Out) buffers for inter-domain synchronization. Both transistor level, SPICE simulation, as well as cycle-accurate, microarchitectural analysis, show that cores using mixed-clock issue queues are able to provide better energy-performance operating points when compared to their synchronous or asynchronous FIFO-based counterparts.

I. Introduction

Moore's Law, which predicted a trend of exponential growth in transistor density and performance, still holds for current generation systems and it is expected to hold for the next few generations. In synchronous design, the master clock acts as a timing reference signal for all basic modules of the design. However, due to the increasing number of transistors and complexity of today's designs, it becomes desirable to have the capability of designing complex systems without the requirement of global synchronization. Such systems can be comprised of modules running at almost the same frequency (requiring only *mesochronous* or *plesiochronous* synchronization [8]) or can have multiple clock domains, with different parts of the chip running at different speeds. In the latter case, no assumptions can be made regarding the relative phase difference of different clock signals.

The individual domains of GALS systems may not only differ in clock frequencies, but also in supply voltages and threshold voltages of devices, thus coupling nicely with the recently introduced concept of voltage islands [12]. In such cases, the performance of each block, in terms of throughput and power consumption, can be optimized by choosing appropriate gate sizes, threshold voltages and supply voltages [2]. Frequencies and voltages of individual blocks or *islands* may be scaled (statically or dynamically) for optimal power or performance of the overall system. Nonetheless, to cope with inter-domain communication in heterogeneous systems, it is imperative to devise reliable communication design methodologies.

While being first introduced by Chapiro [6], the Globally Asynchronous, Locally Synchronous (GALS) design style has been recently explored to tackle this problem [11][15]. Such a solution eliminates the requirement of a global reference clock signal by assuming that the system is comprised of several synchronous blocks communicating asynchronously. One of the most important pieces in the overall GALS design of a superscalar, out-of-order processor is the *issue queue*.

In this paper, we describe and evaluate different **mixed-clock issue queue designs**, able to sustain different clock speeds for incoming and outgoing traffic. In addition, we demonstrate via detailed circuit level simulation, the role of the issue queue as a communication interface between the dispatch and execute clock domains. We also compare and contrast the power consumption and throughput of our implementations with existing synchronous or mixed-clock versions of issue queues used in stand alone mode or in conjunction with mixed-clock FIFOs for inter-domain synchronization.

The rest of the paper is organized as follows. Section 2 describes existing work in this area, while in Section 3, we describe the baseline superscalar pipeline organization considered. Our approach for using the issue queue for communication are illustrated in Section 4. In Section 5 we compare our approach with prior art and present the experimental results. Section 6 concludes the paper and discusses possible directions for future research.

II. Prior Work

Issue logic plays an important role in the performance of superscalar processors. Its functionality is achieved through the use of two important functions, namely *wakeup* and *select*. A detailed analysis of complexity-effective superscalar processors has been presented in [14]. Only the tag-matching part of the wakeup logic and a hierarchical position based selection scheme have been described there. A possible alternative for a high speed four-way issue queue has been presented in [13]. In the proposed solution, the issue queue is partitioned into smaller parts and one instruction is selected from each part, thus reducing the available parallelism in the pipeline drastically. A moderate speed two-way issue queue design has been presented in [10], but it requires a large number of interconnects in an asymmetric manner. This entails the usage of strong buffers for some signals, while the overall structure can lead to an inefficient layout. While our developed wake-up logic is based on the scheme described in [14], our selection logic is similar to the one described in [10], but with some significant changes.

Synchronization is crucial for a mixed-clock design and various schemes have been proposed to address this problem. A robust FIFO-based approach has been presented in [7]. In this case, synchronizing latches are used for communication of handshake signals between clock domains. Although it has a low latency in steady state operation, the worst case latency is two clock cycles. The use of such a FIFO for inter-domain synchronization seems to be feasible, but it increases the number of transistors in the circuit and introduces an additional stage in the pipeline. A stretchable clock approach has been proposed in [16]. However, such a technique cannot be used if the producer and consumer clocks are very different in speed. Our synchronization approach is similar to the one described in [7], in the sense that it is based on synchronizers for inter-domain communication, but we use the issue queue as the interfacing structure, without the need of additional storage structures. A mixed-clock issue queue design has been proposed recently [19], but although the power cost is reduced when compared with the FIFO based implementation, its performance penalty is three cycles in the worst case. This paper introduces three new different designs for such a mixed-clock issue queue and shows available energy-performance trade-offs at both circuit and microarchitectural level.

III. The Baseline Superscalar Pipeline

For the purpose of our study, we consider a superscalar, out-of-order processor, as shown in Fig. 1. The architecture of our processor is based on the MIPS R10000 Superscalar processor [18][19].

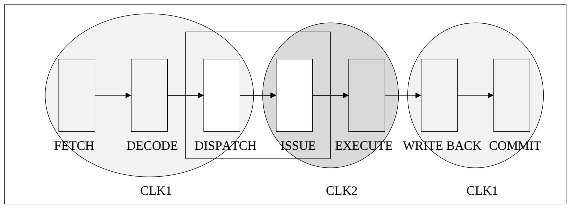

Fig. 1 The basic pipeline

In this paper, we only concentrate on the interplay between the *dispatch* and *issue* stages where it has been suggested that an asynchronous interface may be introduced in GALS processors [11] [15]. To this end, we consider the underlying microarchitecture organization from Fig. 1 and concentrate on the interface between the dispatch and issue logic in three cases: (i) the fully synchronous case, where both dispatch and issue stages run at the same, globally generated clock signal, (ii) a GALS case, in which the dispatch and issue stages are placed in different clock domains and interfaced via an asynchronous FIFO [7], (iii) another GALS case, in which the issue queue itself is mixed-clock and supports dispatching from and issuing instructions to different clock domains. We will describe different methodologies for designing the mixed-clock issue queue.These designs assume that instructions have been decoded and renamed, and the availability of the source operands is known.

Without loss of generality, we consider a 32-entry issue queue with an issue width of two. However, our implementation is scalable and can be easily extended to more entries or wider pipelines. The issue logic has two primary functions, namely *Wakeup* and *Select*. An entry in the issue queue waits until its source operands become ready and is selected for execution.

Wakeup and Select Logic

A typical entry in the issue queue is illustrated in Fig. 2 [14]. It is made up of five components, the physical mappings of the two source registers, the two availability flags for these two registers and a valid bit for the whole entry. The source registers (implemented as CAM cells) are compared with the tags of destination registers being written back during each cycle, and they become ready if a match occurs. A request signal is generated when both (or the only) source operand(s) become ready.

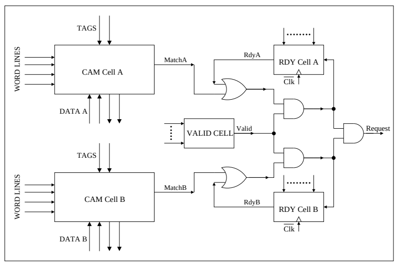

Fig. 2 Request generation logic

The valid bit is set when the instruction enters the issue queue and is reset when it has been selected for execution. The CAM Cells consist of two four-ported SRAMs with a *matching* circuitry similar to the one presented in [14]. In the CAM Cells, the first two ports as used for writing entries and the other two are used for sending selected instructions to the execution units. Each entry in the issue queue generates a request signal and receives a grant signal from the selection logic. This grant signal is used as the word signal and the positive phase of the clock signal is used as the read signal while reading the data. This signal is reset to LOW after the falling edge of the clock to allow precharge of the bit lines before the next read.

The 32-entry issue queue generates up to 32 request signals during the positive phase of each cycle. In a two-way wide pipeline, the selection logic has to select up to two of the available activated request signals. We have assumed the implementation of the selection unit proposed recently [19][20].

IV. The Mixed-Clock Superscalar Pipeline

We have assumed a simplified mixed clock pipeline as illustrated in Fig. 1. In the synchronous version of the pipeline, *Clk1* and *Clk2* are the same. In the GALS version of the pipeline, *Clk1* and *Clk2* are different. The interface between the two clock domains is ensured by a mixed-clock issue queue. For comparison, we also consider the case when the synchronization is done via a generic mixed-clock FIFO [7]. In all cases, we have assumed a 32-entry, 2-way wide issue queue.

A. Mixed-Clock Issue Queue

In this case, the *Dispatch* logic writes entries into the issue queue in the negative phase of *Clk1*. The issue queue *wakes* up the instructions by comparing the source operands with the tags, which are available in the positive phase of the *Clk2*. Comparison of tags without synchronizing the written data to *Clk2* might lead to erroneous behavior. The data (physical addresses of the source operands) can be synchronized to *Clk2* by synchronizing the ***Valid*** bit, which is set when new data is written into the CAM cells.

The Valid Bit

The *Valid* bit implementation is illustrated in Fig. 3. Similar to the design of CAM Cells, it also has four ports and four word signals. No explicit data signals are required as the Valid bit is set when an entry is written into the issue queue and it is reset when the entry is read. Hence, if one of *w1* or *w2* is HIGH, the *Valid* bit is set to HIGH, and is reset to LOW when either *w3* or *w4* goes HIGH. Since the *Valid* bit has to be initialized to LOW during reset, the negative pulse of the *reset* is also used to reset the valid bit to LOW. The *Valid* bit plays the crucial role for synchronization between the two clock domains. To achieve this, we use a two-latch synchronizer to increase the *Mean Time-to-Failure (MTF)* of the *Valid* bit while using it in the clock domain of issue logic. Since its value is set to LOW in the issue clock domain, it need not be synchronized again. So, the falling edge of the *Valid* bit is used for generating a negative pulse which resets the two synchronizers to LOW. The pulse generator circuit is illustrated in Fig. 4

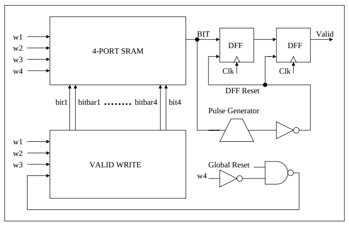

Fig. 3 The Valid bit

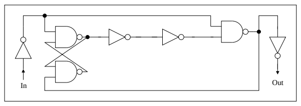

Fig. 4 The pulse generator

Tag Match Synchronization

The performance of a superscalar processor depends on its ability to execute the maximum number of instructions in the minimum possible time. The instructions waiting in the issue queue for other instructions to complete can be issued only after a tag match occurs for the corresponding source operand(s). The synchronization delay introduced due to running the *Wakeup* logic and the execution units on different clocks can introduce some penalty in performance. Thus, it is desirable to have the *Wakeup* logic of the issue queue in the same clock domain as the execution units.

However, we still have to synchronize the tags generated by the execution units with the write-back and dispatch units running on *Clk1*. Consider the worst case situation in the pipeline illustrated in Fig. 5. The execution units generate the results of destination register *X* (say) in the negative phase of clock cycle *a0* of *Clk2*, and the tags are broadcast in the positive phase of *a1*. The tags are synchronized to *Clk1* by using two synchronizers and they can be used only after two rising edges of *Clk1*. In the worst case, the rising edge of clock cycle *b1* just misses the generated tags. In this case, tags become available in the positive phase of clock cycle *b3* of *Clk1*. Thus, any instruction entered into the issue queue before *b3* and requiring the result of *X* marks the corresponding source register as unavailable. If an instruction requiring the result of *X* is entered into the issue queue in the negative phase of clock cycle *b2*, it becomes valid in the domain of *Clk1* in *b3* and in the domain of *Clk2* in *a4*. So, this instruction is waiting in the issue queue for a tag which has been generated four cycles ago and was missed.Hence, this instruction can never be issued and a deadlock occurs.

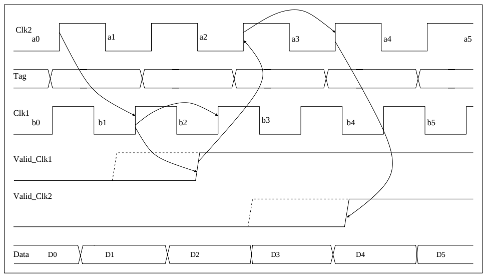

Fig. 5 Synchronizing tag matches

B. Tag storing

A straightforward solution to this problem is to delay the tag matching by three cycles [19][20]. The tags generated in the cycle *a0* are made available for comparison in cycle *a4*. However, if there are other instructions in the issue queue waiting for the result of *X*, which could have been potentially woken up in cycles *a1*, *a2* or *a3* can become awake only from clock cycle *a4*. This not only introduces a considerable delay in issuing the instructions, but also increases the probability of having more than two requests (the width of the pipeline) in each clock cycle.

Our proposed way to tackle this problem instead, is to *store* the tags for four clock cycles (*a1, a2, a3* and *a4*) and expand the tag matching capability from two to eight. The tag which was initially available for comparison in cycle *a1* is now available for comparison in *a1, a2, a3* and *a4*. Consider the situation in Fig. 5 again. The instruction written into the issue queue in *b2* requires the results of two destination registers *X* and *Y* (say). The result of *X* becomes available in the negative phase of *a0*, and the tag can be broadcast in the positive phase of *a1*. The result of *Y* becomes available in the negative phase of *a3* and the tag can be broadcast in the positive phase of *a4*. So, delaying the broadcast of tags by three cycles would have resulted in the first operand becoming ready in *a4*, but the second operand would become ready in *a7*. Hence, the instruction can generate a request for selection only four rising edges after *a3*, and can be selected for execution in *a8*. In the synchronous case, it could have been selected in *a5*. In the worst case, such an approach introduces a synchronization delay of two cycles, thus the overall latency in the worst case would be three cycles.

C. Clock Cycle Stealing

In the synchronous pipeline the *Valid* bit is set to HIGH when the instruction is written into the issue queue. In Fig. 5, *D2* is written in the negative phase of *b2*, but cannot be used before *b4*. It is to be noted that there are two extreme ways in which the *Valid* bit can be set to HIGH in the domain of *Clk2*: one in which the rising edge of *Clk2* just misses the rising edge of *Valid_Clk1*, and the other extreme in which the rising edge just meets the setup time requirement. In the former case the *Valid_Clk2* bit is seen as HIGH after two cycles of *Clk2*, and in the latter case it is seen as HIGH after one cycle of *Clk2*. So, the *Valid_Clk2* bit will take at least one cycle of *Clk2* after *Valid_Clk1* is set before it can be used, and at most two cycles in the worst case.

The issue logic of the pipeline is very complex and is likely to be the most critical structure of the pipeline [14]. Hence, it can be assumed that the front end of the pipeline running on *Clk1* can run faster than the issue logic running on *Clk2*. In this case, we can further reduce the synchronization latency by setting the *Valid* bit to HIGH one cycle before the actual data is written into the issue queue. This can be done since the data is written into the issue window one clock cycle (of *Clk1*) after the *Valid_Clk1* bit is set to HIGH. At the same time, the *Valid_Clk2* bit will be set to HIGH at least one clock cycle (of *Clk2*) after *Valid_Clk1* bit is set to HIGH. Since the clock period of *Clk1* is less than *Clk2*, data can be used immediately without any timing violation.

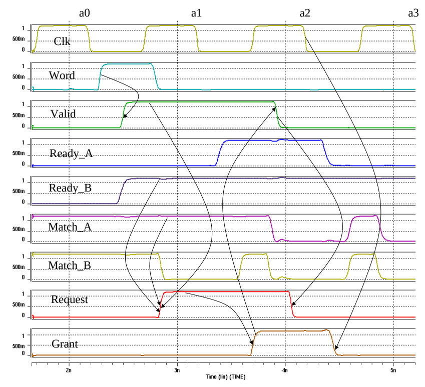

Fig. 6 The synchronous case

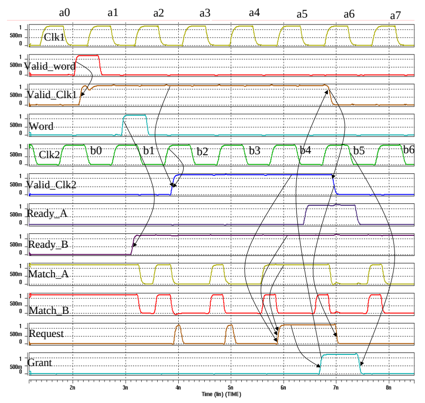

Fig. 7 Mixed-clock with delayed tags and cycle stealing (IQSD)

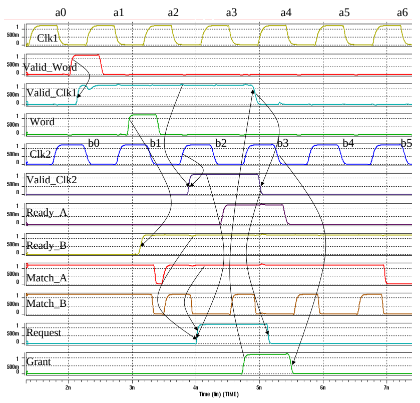

Fig. 8 Mixed-clock with tag storing and cycle stealing (IQSM)

This case is illustrated in Fig. 5 by the dashed lines. By following similar reasoning as in the tag storing case, it can be deduced that the broadcast of the tags has to be delayed by two cycles (instead of three), or the tag matching capability has to expanded to six (instead of eight), if tag storing is also used. The worst case latencies in these cases would be three and two cycles (of *Clk2*) respectively.

D. Mixed-Clock FIFO

As a comparison, we also consider the case in which the interfacing between the front-end and back-end clock domains is done through a mixed clock FIFO [7][11][15]. In this case, the issue queue is fully synchronous, running at the speed of the back-end *Clk*2. In this case, the worst case synchronization delay is two cycles, while two clock cycles are need for dispatch and issue stages. Hence, the worst case latency of a FIFO based scheme is four cycles (of *Clk2*) after an entry is made into the FIFO.

V. Experimental Results

The circuits described in this report have been custom designed using the STMicro 0.13μm technology. In our designs, we have given priority to speed over power consumption. Both static and domino logic families have been used for designing the circuits. The circuits in the critical path have been designed using domino logic, and the ones in the non-critical path have been designed using static logic gates. Optimization techniques like logical effort [17] and transistor tapering [9] have been employed to improve the performance of the circuits. Standard circuits have been used for designing the RAM cells and the peripheral circuitry (read and write amplifiers). The address decoders have been designed using a split row scheme (*preliminary decoders*). The details of these circuits have been described in [5]. To assess the feasibility of our proposed approach, we have employed both a circuit level detailed analysis, as well as a cycle-accurate, microarchitectural validation on real instruction traces.

A. Circuit-Level Analysis

We have simulated (pre-layout, HSPICE) the issue logic design for all cases described in Section 4. In the synchronous case, the dispatch and issue logic run at the same speed of 1GHz. In the GALS pipeline, the mixed clock issue window interfaces two different clock domains: the dispatch unit running on *Clk*1 of 1.1 GHz, and the execution unit running on a *Clk*2 of 1 GHz. The clock speed of the front-end domain has been chosen based on timing analysis of the dispatch and rename logic (that are likely to be on the critical path in the front-end of the pipeline). We have analyzed different GALS architectures employing the following mixed-clock interfacing schemes:

- Mixed-clock issue queue with delayed tags (IQTD).
- Mixed-clock issue queue with tag storing (IQTM).
- Mixed-clock issue queue with cycle stealing and delayed tags (IQSD).
- Mixed-clock issue queue with cycle stealing and tag storing (IQSM).
- FIFO based inter-domain synchronization (FIFO).

We have compared these designs by evaluating the average power and the Energy consumed Per Issue (EPI), given by Equation (1).

$$EPI = \frac{Power \times T_n}{n} \quad (1)$$

where *Power* is power consumed by the circuit, T_n is the number of clock cycles of *Clk2* required to issue *n* instructions.

Table I Comparison of different schemes

Component	SYNC	IQTD [19]	IQSD	IQTM	IQSM	FIFO
Power [mW]	21.02	22.08	22.22	32.97	29.48	25.11
EPI [mW/instr.]	13.13	17.94	16.66	20.60	18.42	18.83
WCL[cycles]	1	4	3	3	2	4

The results are shown in Table I (WCL stands for Worst Case Latency). These results were obtained by simulating a typical trace in the pipeline. Due to computational limitations, full instruction traces could not be simulated using HSPICE.

As it can be seen, the power consumed in the previously proposed IQTD scheme [19] is more than that of the synchronous case; this is due the addition of synchronizing flip-flops for the *Valid* bits. Two additional decoders (along with the synchronizing flip-flops) are required for setting the *Valid* bit to HIGH in the IQSD scheme. The power consumed by these decoders is very small and hence the power consumption in IQSD is almost the same as the one of IQTD. In the IQTM scheme, each CAM cell has to compare the physical addresses of the source operands with each of the eight stored tags. The tag matching circuitry has been implemented using domino logic and increasing the tag matching capability has an adverse effect on the power consumption of the issue queue. In the IQSM scheme, a CAM cell has to compare only six tags instead of eight as in the IQTM scheme. The FIFO based scheme produces a significant increase in the power consumption due the addition of extra synchronization latches, when compared to the synchronous case.

We also show detailed timing simulation of the synchronous issue queue and our best performing mixed-clock issue queue implementations (IQSD and IQSM; complete results can be found in [20]). Fig. 6 shows the synchronous case. The dispatch unit writes an entry into the issue window in the negative phase of the clock cycle *a0* of *Clk*. The entry is set to valid, the source operand A is not available (*Ready_A* is set to LOW) and the source operand B is available (*Ready_B* is set to HIGH). The entry tries to match the source operand A with the tags of the destination registers forwarded from the execution unit. In this example the result is available during the negative phase of *a0*, and the tag is broadcast during the positive phase of *a1*. Since the entry is valid and both the operands are available, the *Request* signal is asserted. In this case, this entry is selected in the next clock cycle (*a2*), and the *Grant* signal is asserted HIGH. The entry is read in this cycle, and is also erased from the issue queue by resetting the *Valid* bit to LOW. We use the *Grant* signal as the word line for the corresponding entry in the issue queue. The *Grant* signal is reset to LOW in the negative phase of *Clk* to allow precharging of the bit lines before the next read operation (which may happen in the positive phase of the following clock cycle)

The simulation results of the mixed-clock issue queue with clock stealing and delayed tags are shown in Fig. 7. The *Valid_Clk1* bit is set in *a0* and the data is written into the issue queue in *a1*. *Valid_Clk2* becomes HIGH in *b2*, and the required result is generated in the negative phase of *b1*. The tag is broadcast in *b4*, in which a request is generated as a result of the tag match. It is selected for execution in *b5* and the *Valid* bits are reset.

The simulation results of the mixed-clock issue queue with clock stealing and tag storing are shown in Fig. 8. Since the tag can be compared in *b2*, in this case the request is generated in *b2*. Note that the tag is available for comparison for only three cycles after the results are generated, and the *Match_A* signal goes LOW in *b*5.

The GALS pipeline with the FIFO acting as the interface adds an additional stage in the pipeline along, with an additional synchronization penalty. The pipeline behaves in the same manner as the synchronous version, but entails a worst case latency of four cycles.

As it can be seen, the mixed-clock issue queues with delayed tags are better in terms of power consumption, but are worse in terms of latency than the cases where the tags are memorized. The EPI is an important statistic for comparing various schemes, but drawing a clear conclusion cannot be achieved unless full chip simulation is performed on real input traces, as described next.

B. Microarchitectural Level Analysis

To assess which of the described issue queues is better in terms of both performance and energy, we have developed a GALS microarchitectural simulator, conceptually similar to the ones described in [11][15].

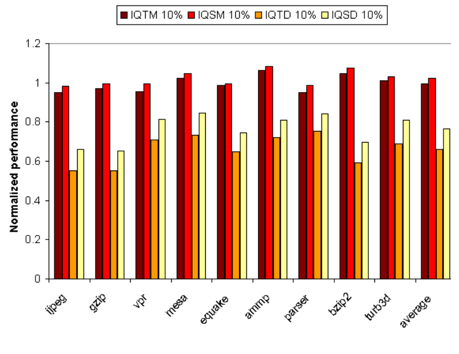

Fig. 9 Normalized performance for the entire core for various mixed-clock issue queues. The front-end and back-end are assumed to run at 1.1GHz and 1GHz, respectively. (The baseline is the fully synchronous core running at 1GHz)

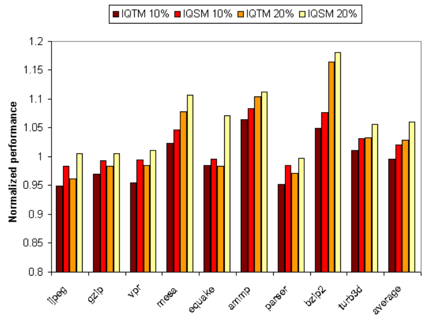

Fig. 10 Normalized performance for mixed-clock issue queues with tag storing, with (IQSM) and without (IQTM) cycle stealing. The front-end runs at 1.1GHz (10% case) or 1.2GHz (20% case), while the back-end runs at 1GHz. (The baseline is the fully synchronous core running at 1GHz)

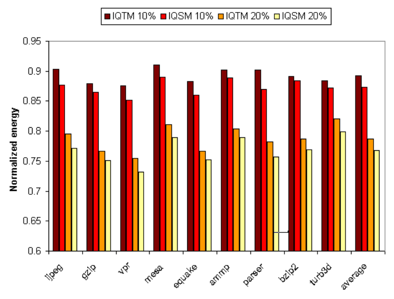

Fig. 11 Normalized energy for mixed-clock issue queues with tag storing, with (IQSM) and without (IQTM) cycle stealing. The front-end runs at lower voltage to account for 10% and 20% slack relative to the back-end, respectively. Both clock domains run at 1GHz, with random relative phases. (The baseline is the fully synchronous core running at 1GHz)

Based on an event-driven simulation engine, the simulator is compatible with SimpleScalar [4] or Wattch [3] microarchitectural simulators. The power models for all major microarchitectural modules are built using Wattch's activity based power macromodeling methodology which has been shown to be within 10% accurate when compared to real power values for three different high-end processors [3]. The simulation engine can be used in a fully synchronous mode (with both clock domains *Clk*1and *Clk*2 running synchronously at the same speed), or a GALS mode in which *Clk*1 and *Clk*2 can have arbitrary relative speeds or starting phases. In this latter case, inter-domain communication is achieved via mixed-clock issue queues (all four flavors described in this paper), or through FIFO-based communication, in conjunction with a fully synchronous issue queue in the back-end clock domain.

TABLE II MICROARCHITECTURAL ORGANIZATION

Parameter	Value
Issue Queue size	64 instructions
Fetch width	
Decode width	
Issue width	4 instructions
Commit width	
Functional units	4 IntALUs, 1 IntMult/Div
	2 FPALUs, 1 FPMult/Div
Branch prediction	GShare, 12 bit history
L1 I-Cache	32K, 2-way set associative
L1 D-Cache	32K, 4-way set associative
L2 Unified Cache	256K, 4-way set associative

To this end, we have considered a more aggressive pipeline organization than the one based on the 2-way, 32-entry issue queue, as described in previous sections. We have opted instead for a seven-stage, four-way deep pipeline, with a 64-entry issue queue design. The worst case latencies characterizing the mixed-clock issue queues will also hold for this larger number of entries. To characterize the power consumption of the issue queues, we have started from the 32-entry design analyzed in detail using HSPICE and used the models described in [3][14] to obtain power values for the 64-entry version of the various mixed-clock issue queue designs. The characteristics of the simulated pipeline are presented in Table II. Benchmarks considered are from both Spec95 and Spec2000 suite.

As shown in Fig. 9, although the worst-case latency of various mixed-clock issue queues ranges in the same interval, the impact on the overall performance is very different. The tag storing scheme is able to hide most of the synchronization penalty (thus producing a slight performance increase of up to 7-8% if cycle stealing is used), while delaying the tags delays instruction commit, thus producing a large hit in performance (25-30% in some cases). To assess how the tag storing schemes behave for aggressive pipelines, we have also considered the case where the front-end can run 20% faster then the back-end running at 1GHz (Fig. 10 and Fig. 11). Such a scenario is very likely with increased issue queue size and more complex microarchitectures. As seen in Fig. 10, by allowing the front-end to run at the faster rate, an overall increase in performance of up to 17% (5% on average) is possible. In addition, if energy is the main concern, the 10% and 20% slack in the cycles times of the front-end and back-end clock domains can be exploited to lower the front-end voltage according to the equation: $Delay \propto V_{dd}/(V_{dd}-V_T)^{\alpha}$ (we have used a value of $\alpha = 1.2$). As shown in Fig. 11, such a core could run at up to 28% less energy than the original synchronous processor. Thus, a mixed-clock issue queue allows for better energy-performance trade-off points, in either a statically or dynamically voltage/speed scaling scenario.

VI. CONCLUSION

In this paper we have proposed and analyzed various **mixed-clock issue queue** designs for high-end, out-of-order superscalar processors, able to sustain different clock rates and speeds for the incoming and outgoing traffic. We have compared and contrasted our implementations with existing synchronous versions of issue queues used stand-alone or in conjunction with mixed-clock FIFOs for inter-domain synchronization. As expected the mixed-clock pipelines have to entail extra latencies, and the synchronization hardware consumes additional power. However, when compared to their synchronous counterparts in terms of overall performance and energy operating points, mixed-clock cores may be able to achieve better energy for little performance loss, or higher performance by allowing separate optimizations for the various clock domains. Future work may be able to offer strategies for dynamically managing local speeds and voltages depending on the application profile.

ACKNOWLEDGEMENTS

This research was supported in part by National Science Foundation Early Faculty CAREER Award No. CCR-008479.

REFERENCES

[1] K. Bernstein et al, *High Speed CMOS Design Styles,* Kluwer Academic Publishers, Boston, 1998.
[2] R. W. Brodersen, M. A. Horowitz, D. Markovic, B. Nikolic, and V. Stojanovic, "Methods for True Power Minimization," in *Proc. IEEE/ACM Intl. Conference on Computer-Aided Design (ICCAD),* pp. 35-42, San Jose, CA, Nov. 2002.
[3] D.Brooks, M. Martonosi, V. Tiwari, "Wattch: A Framework for Architectural Level Power Analysis and Optimization," in *Proc. Intl. Symposium on Computer Architecture (ISCA)*, June 2000.
[4] D. Burger, T. Austin, "The SimpleScalar Toolset," University of Wisconsin, CSD Technical Report #1342, June 1997.
[5] A. Chandrakasan, W. J. Bowhill, and F. Fox, *Design of High-Performance Microprocessor Circuits,* IEEE Press, NY, 2001.
[6] D. M. Chapiro, *Globally Asynchronous Locally Synchronous Systems,* PhD thesis, Stanford University, 1984.
[7] T. Chelcea, and S. M. Nowick, "Robust Interfaces for Mixed Timing Systems with Application to Latency-Insensitive Protocols," in *Proc. Design Automation Conference (DAC),* pp. 21-26, 2001.
[8] W. J. Dally and J. W. Poulton, *Digital Systems Engineering,* Cambridge University Press, Camebridge, 1998.
[9] L. Ding and P. Mazumder, "On Optimal Tapering of FET Chains in High-Speed CMOS Circuits," in *IEEE Transactions on Circuits and Systems II,* vol. 48, pp. 1099-1109, Dec 2001.
[10]J. A. Farell and T. C. Fischer, "Issue Logic for a 600-MHz Out-of-Order Execution Microprocessor," in *IEEE Journal of Solid-State Circuits,* vol. 33, pp. 707-712, May 1998.
[11]A. Iyer and D. Marculescu, "Power Efficiency of Multiple Clock, Multiple Voltage Cores," in *Proc. IEEE/ACM Intl. Conference on Computer-Aided Design (ICCAD),* pp. 379-386, San Jose, CA, Nov. 2002.
[12]D. E. Lackey, P. S. Zuchowski, T. R. Bednar, D. W. Stout, S. W. Gould, and J. M. Cohn, "Managing Power and Performance for System-on-Chip Designs using Voltage Islands," in *Proc. IEEE/ACM Intl. Conference on Computer-Aided Design (ICCAD),* pp. 195-202, San Jose, CA, Nov. 2002.
[13]J. Leenstra, J. Pille, A. Mueler, W. M. Sauer, D. F. Wendel, "A 1.8 GHz Instruction Window Buffer for an Out-of-Order Microprocessor Core," in *IEEE Journal of Solid-State Circuits,* vol. 36, pp. 1628-1635, Nov 2001.
[14]S. Palacharla, N. Jouppi, and J. E. Smith, "Complexity-Effective Superscalar Processors," in *Proc. Intl. Symp. on Computer Architecture (ISCA)*, pp. 206-218, June 1997.
[15]G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Dynamic frequency and voltage control for a multiple clock domain microarchitecture," in *Intl. Symp. on Microarchitecture (MICRO)*, pp. 356-367, 2002.
[16]A. E. Sjogren and C. J. Myers, "Interfacing Synchronous and Asynchronous Modules Within A High-Speed Pipeline," in *IEEE Tran. on VLSI Systems,* 8(5):573-583, Oct 2000.
[17]I. Sutherland, B. Sproull and D. Harris, *Logical Effort: Designing Fast CMOS Circuits,* Morgan Kaufmann, San Francisco, 1999.
[18]K. C. Yeager, "The Mips R10000 superscalar microprocessor," in *IEEE Micro,* vol. 16, pp. 28-41, Apr 1996.
[19]V. S. P. Rapaka, D. Marculescu, "A Mixed-Clock Issue Queue for Globally Asynchronous, Locally Synchronous Processor Cores," in *Proc. ACM/IEEE Intl. Symposium on Low Power Electronics and Design(ISLPED)*, pp. 372-277, Seoul, Korea, Aug. 2003.
[20]V. S. P. Rapaka, "Design and Analysis of Mixed-Clock Issue Queues for Globally Asynchronous, Locally Synchronous Processor Cores," Master's Thesis, Technical Report CSSI 03-05, Carnegie Mellon University, June 2003.