Slack-based Bus Arbitration Scheme for Soft Real-time Constrained Embedded Systems

Minje Jun
School of Electrical and Electronic Engineering, Yonsei University, Republic of Korea
jjuninho@yonsei.ac.kr

Kwanhu Bang
School of Electrical and Electronic Engineering, Yonsei University, Republic of Korea
lamar49@yonsei.ac.kr

Hyuk-Jun Lee
Cisco Systems Incorporation, U.S.A.
hyukjunl@yahoo.com

Naehyuck Chang
School of Computer Science & Engineering, Seoul National University
Republic of Korea
naehyuck@snu.ac.kr

Eui-Young Chung
School of Electrical and Electronic Engineering, Yonsei University, Republic of Korea
eychung@yonsei.ac.kr

Abstract - We present a bus arbitration scheme for soft real-time constrained embedded systems. Some masters in such systems are required to complete their work for given timing constraints, resulting in the satisfaction of system-level timing constraints. The computation time of each master is predictable, but it is not easy to predict its data transfer time since the communication architecture is mostly shared by several masters. Previous works solved this issue by minimizing the latencies of several latency-critical masters, but the side effect of these methods is that it can increase the latencies of other masters, hence they may violate the given timing constraints. Unlike previous works, our method uses the concept of "slack" in order to make the latency as close as its given constraint, resulting in the reduction of the side effect. The proposed arbitration scheme consists of bandwidth-conscious arbiter and scheduler. The arbiter can be any existing bandwidth-conscious arbiter and the scheduler implements the latency-awareness proposed in this paper. The scheduler is involved in the arbitration only when it observes a request whose slack is not sufficient for the given timing constraint. The experimental results show that our method outperforms the conventional round-robin arbiter by more than 100% in the best case in terms of the longest violated cycles.

I. Introduction

As the semiconductor technology develops rapidly, it is possible to integrate billions of transistors on a single silicon die. System-on-Chip (SoC) is a typical example. As of today, many embedded systems using processing cores are designed as SoCs. A major challenge in SoC design is that it is difficult to meet Time-To-Market (TTM) requirement due to increasing die size and design complexity. Therefore, the concept of design reuse becomes extremely important and platform-based design methodology has been widely accepted [1][2]. Even though Intellectual Property (IP) components can be reused for different designs, the communication architecture connecting those components must be carefully designed in order to satisfy all the given constraints, since each design may have different performance specifications and constraints. Suppose an MPEG decoder which can manipulate various picture sizes. Digital Media Broadcast (DMB) applications require the decoder to process 15 frames (picture size: CIF) per second while a high-end Potable Media Player (PMP) requires the decoder to handle 30 frames per second even within a larger picture size. Clearly, bandwidth and latency requirements are different in these two cases. Therefore, the communication architecture must be tuned to be suitable for each application. For this reason, the communication architecture design is one of the time-consuming steps in SoC design [3].

Multi-layer bus architecture was proposed as a solution to meet both bandwidth and latency requirements [4][5]. In this architecture, the buses that bind closely coupled components are connected through bridges, which improve both bandwidth and latency. However, the number of bus layers is limited by routing density and the number of external pins for the DRAM access. Even in multi-layer bus architecture, an appropriate arbitration scheme is needed to handle the concurrent data transfer requests from multiple masters in each layer.

Classical arbitration schemes such as round-robin, fixed priority and hybrid of them are still used popularly. But, they ignore the latency issue or assume that the latency requirement is proportional to the bandwidth requirement. Time-Division- Multiple-Access (TDMA) scheme is another bandwidth-conscious arbitration scheme. It can provide guaranteed throughput and increase the overall throughput by allowing the unused timing slots to be filled with best effort traffic. Recently, latency-conscious [6][7][8][9] or QoS-aware [7][10][11] arbitrations are introduced. LOTTERYBUS [6] extends the TDMA scheme by using a statistical method and resolves the long latency issues. Weber et al. proposed a QoS-aware arbitration scheme which introduces the concept of "epoch" [7]. Epoch is a group of requests and its size can be different for each master. The arbiter allocates bandwidth and service order (e.g. latency) per epoch basis. These approaches resolve the latency issues by minimizing the latency. However, these methods focused more on guaranteeing bandwidth allocation and often tried to minimize the latency beyond their latency requirements. Excessive reduction in the latency of a certain master may increase the latencies of other masters. There have also been many efforts to overcome the latency and bandwidth problems by improving bus architecture, such as SAMBA BUS [8] and FLEXBUS [9]. In [8], their bus is a single-layer bus, but it can behave like a multi-layer bus system when concurrent requests on a single bus do not require the same physical resources. In [9], their bus system can be dynamically configured to balance the traffic load of busses. Their objective is to improve the bandwidth utilization by efficient use of hardware resources, thus the latency is improved as a side effect. In this paper, we especially focus on the latency and propose a latency-aware arbitration scheme by using the concept of "slack". The goal of our method is not to minimize the latency but to adjust the

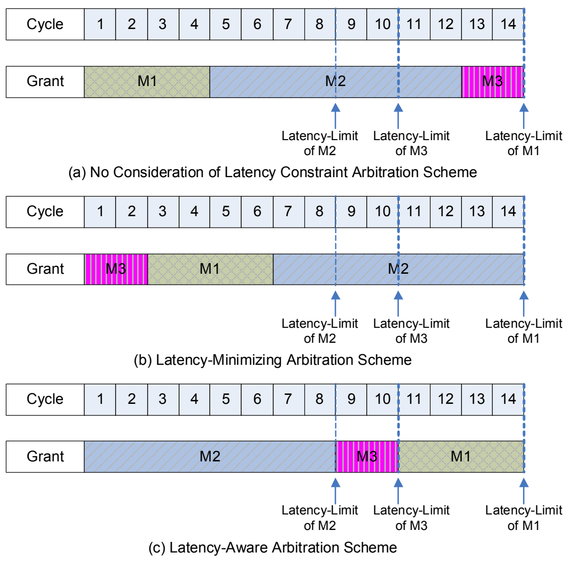


Fig. 1 Example of three arbitration schemes

latency to the given constraint by minimizing the slack with minor hardware overhead and no change in bus protocol. We introduced the basic concept in [12]. In this paper, we further extend the method and our contributions are two folds. First, the extended method can dynamically change the latency constraint of each master to support various operation scenarios. Second, time-critical requests can termi- nate the request being served and this extension can reduce the overall deadline misses. These new features differentiate our method from the Earliest Deadline First(EDF) by providing more on-line properties.

Section 2 explains the motivation of our method, Section 3 describes the basic architecture of our arbitration scheme and the experimental results are shown in Section 4 followed by conclusions in Section 5.

II. Motivation

A. *Keeping Constraints, Not Minimizing*

Latency becomes critical when many masters are sharing a common bus and their workload is heavy enough to cause bus contentions frequently in real-time systems. Minimizing the latency of specific masters is the traditional method to handle this issue. However, it will increase the latency of other masters. In systems requiring real-time constraint, it is unnecessary to minimize the latencies of some masters beyond their time constraints.

Fig. 1 depicts an example. In this example, service latency for Master 1, 2, and 3 span 4, 8, and 2 cycles respectively. The requests for three masters are all initiated at cycle 0, and M3 is the most latency sensitive master. Fig. 1 (a) shows an arbitration scheme which does not use latency constraints for scheduling. Masters 2 and 3 violate the latency constraint as the bus is granted in ascending order. Only M1 meets the constraint. Fig. 1 (b) shows the scheduling of a typical latency-minimizing arbiter. It minimizes the latency of the most latency-sensitive module, M3, causing M2 to violate its constraint. While neither of two arbitration schemes can

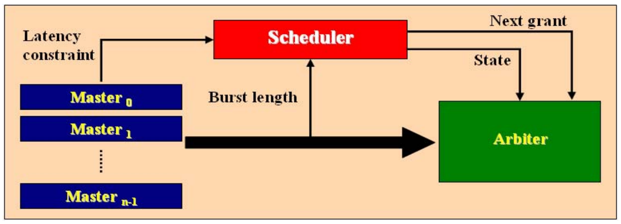


Fig. 2 The basic architecture of latency-aware arbiter

meet the latency constraints for all three masters, in the latency-aware arbitration, as shown in Figure 1 (c), all the masters are granted the bus without violation by not minimizing latency but making latency as close to the constraints as possible.

B. *Lead Latency and Transfer Latency*

At the time when a master requests to get permission to use the bus, the bus can be occupied with another master. In this case, the requesting master must wait until it is granted to use the bus. We define this latency as lead latency for the requesting master. Even after the master is granted to use the bus, the transfer latency, which is usually called 'burst length' in bus-based system, is needed to transfer all the data. The total latency is the sum of these two latencies.

In the past, many schemes were proposed to guarantee the bandwidth or minimize the latency in interconnection field. However, they have focused mostly on reducing only lead latency. There has been no approach to take into account the transfer latency and their latency limits of various masters and arbitrate based on them. Moreover, it is practically more desirable to meet the latency limit of each master than to minimize the latency of overall system or some latency-sensitive modules. In order to satisfy each constraint, it is necessary to be aware of the current latencies and reflect them in the arbitration. In the following section, we present the latency-aware arbitration scheme which keeps track of the current latencies of masters and use them for scheduling.

III. Latency-Aware Arbiter

A. *Architecture overview*

The simplified structure of the proposed scheme is depicted in Fig. 2. As shown in Fig. 2, the arbiter consists of a conventional bandwidth-conscious arbiter and a scheduler.

The proposed architecture can be incorporated with any bandwidth-conscious arbiter and the quality of bandwidth allocation is determined by the arbiter. This part is not a focus of this paper and we only concern at the scheduler for latency constraints. The scheduler is implemented in an augmented fashion so that the change of the conventional arbitration scheme is minimized. Its major function is to compute the slack of each request. "Slack" is defined as the maximum number of clock cycles to complete the service of a given request without violating the given latency constraint. The scheduler needs the burst length of each request as well as the latency constraint information for the slack computation which will be discussed in part B of this section.

For the slack computation, the latency constraint of each latency-sensitive master can be programmed either statically at design time or dynamically in run-time through the bus by considering the scheduler as a normal slave, which is just as a normal write operation supported by most of bus protocols. Such implementation strategy does not increase the number of wires for the scheduler and it is possible to keep the wire density as same as the conventional arbitration scheme. As far as the burst length is concerned, each bus request includes this information in case of most bus protocols, so we just feed this information to the scheduler as well as to the arbiter.

However, we need additional local wires for the communication between the scheduler and the arbiter. There are two signals – "State" and "Next grant" from the scheduler to let the arbiter know which requests are in urgency in terms of latency constraint. The signal "State" indicates the level of urgency for the selected request which is the most critical in terms of slack among the pending requests. The signal "Next grant" informs the master ID of the selected request. The signal "State" requires two-bits to represent the urgency which we will discuss in detail in Part C. Also, the number of bits for the signal "Next grant" is logarithmically proportional to the number of latency-critical masters. Note that these two signals are not globally routed, hence the impact on the routing is marginal. Since the scheduler has three levels of state, only two bits are necessary for the 'State' signal. The arbiter finally selects a request to be granted depending on the information from the scheduler.

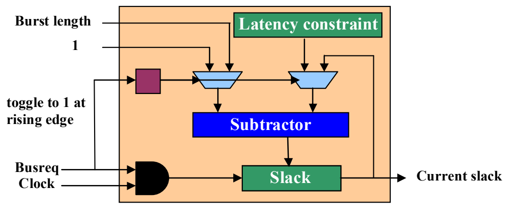
B. *Scheduler architecture*

The slack counter and the scheduler architectures are shown in Fig. 3 (a) and (b), respectively. As shown in Fig. 3 (a), the slack counter consists of several elementary components including two registers-one for the latency constraint and the other for the slack. The latency constraints are at most a few hundred cycles and, therefore, each register needs less than 10 bits in most cases. The scheduler consists of slack counters as many as the number of latency sensitive masters, and two comparators, as shown in Fig. 3 (b). Only a multiplexer is required to decide the final bus winner.

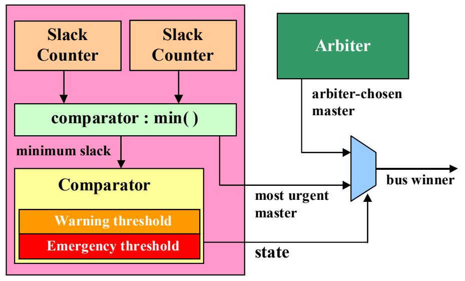
Each latency-critical master can set its latency constraint into a corresponding latency constraint register, if necessary, through a normal write operation, thus the overhead for the constraint setting is a single write transfer from the system perspective.

When a master issues a request, the slack for the request is computed (refer to Part C) and stored in the corresponding slack register. As the time advances in terms of clock cycle, the slack should be reduced, hence the slack counter decrements the slack. The most significant bit of the slack counter is used as a valid bit to represent whether the current value in a slack counter is valid or not. The valid bit is only set when a corresponding master issues a request. The valid bit is reset after the request is completely serviced.

The comparator controls the urgency of requests. By comparing the slacks of valid slack counters, the request whose slack is the smallest is selected as the most urgent request. The master information is delivered to the arbiter through the signal "Next grant" and the urgency of the selected request is delivered to the arbiter through the signal "State" in parallel. The urgency is determined by pre-programmed threshold values and the details are in Part D.



(a) Slack counter architecture



(b) Scheduler architecture

Fig. 3 Slack counter and scheduler architectures

C. *Slack computation*

Whenever any latency-critical master issues a request, the scheduler receives the burst length information and computes the slack using Equation (1) and loads it into the corresponding slack counter.

$$Slack_i = L_i - B_i \times T_i - S_j \qquad (1)$$

where, $Slack_i$ is the slack of the request from master i, L_i is the latency constraint of master i, B_i is the burst length of the request, T_i is the transfer time per beat, and S_j is the latency of a target slave j. The slack counter is decremented every clock cycle as mentioned in Part B and thus the scheduler updates slack information every clock cycle. T_i is 1 cycle in most cases and thus multipliers are not needed.

In many cases, the slave latency is not fixed and varies depending on the workload and its internal states. For instance, DRAM memory controller has a variable latency depending on the addresses of memory references. Bank conflicts or same row accesses could dramatically change the memory access time. Furthermore, it may have internal memory access scheduler to maximize its bandwidth utilization. In this work, we conservatively assume the worst case slave latency to compute the data transfer latency.

In this work, we define a slack as the remaining clock cycles to complete the service for the given request without the constraint violation. But our method is flexible enough to be adjusted depending on the purpose of the constraint. For instance, we can omit the second term in Equation (1) if the initial latency is mostly concerned.

D. *Urgency control*

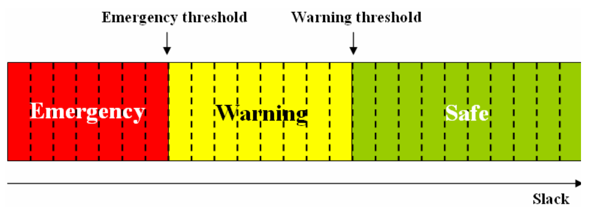


Fig. 4 State assignment for each request

As mentioned in Part B, the urgency of each request is controlled by the pre-programmed threshold pair – T(W, E). The threshold pair is globally applied to every request. T(W) indicates that a request whose slack is less than or equal to T(W) should be serviced very soon, otherwise the request has a high probability to miss the given latency constraint. On the other hand, T(E) means that the a request whose slack is less than or equal to T(E) will miss the given latency constraint if it is not serviced right now. Thus, T(W) is always greater than T(E). Based on T(W, E), the scheduler assigns one of three states to each request depending on their slacks. Three states are defined as “safe”, “warning”, and “emergency”. We say that the state “warning” is deeper than the state “safe” and similarly for the state “emergency” based on the amount of the slack. Fig. 4 shows the relation between those states and slack of a request. A request moves from the shallower state to the deeper state as time advances. In our scheduler, a request in the state “emergency” is the most urgent request to be serviced. When every request is in the state “safe”, the scheduler delivers null information to the arbiter, meaning that there is no urgent request by setting the signal “State” to “00”. The signal “State” is set to “01” when the request in the state “warning” is selected and to “10” when the request in the state “emergency” is selected. We can add one more state to exploit the state “11” if necessary. We select a request in the order of the state depth when a single request is in the deepest state. An arbiter behaves in a different manner depending on the state urgency of the selected request. When a selected request is in the state “warning”, it gives a grant to the request right after the current service is completed. However, when a selected request is in the state “emergency”, the arbiter takes a more aggressive approach by stopping the current service and hand over the ownership to the request informed by the scheduler. This will degrade the bus performance from the bandwidth perspective, but it will reduce the peak latency violation since the preempted request is in the state “emergency”. The kicked-out request is treated as same as other pending requests with the same arbitration policy. When it is selected and serviced again, it completely resends the data using “retry” feature which is supported by many contemporary bus protocols. Note that such preemption can occur repeatedly and the overall bandwidth utilization will be degraded by resending a large amount of data. To avoid such disaster, we lock the transfer of the request in the state “emergency” not to be kicked-out by any other requests. This strategy aims at the trade-off between the bandwidth utilization and minimizing the peak value of the violated cycles. When there are several requests in the same level of urgency (“warning” or “emergency”), the request which has the smallest slack is selected by the scheduler and its information is passed to the arbiter.

Note that even though this scheme can surely increase the probability in which the latency constraints of the masters are met, the constraint violation is inevitable in some situations, especially when we consider very heavy workload. For this reason, the proposed arbitration scheme is more appropriate for the soft real-time constraint rather than the hard real-time constraint. However, it is true that most of IP components have in/out buffers to avoid the disaster due to constraint violation. Thus, our scheme can be applicable to hard real-time applications if the violated interval is reasonably small so that the in/out buffers can afford to. The quality of bandwidth allocation is mainly determined by the bandwidth-conscious arbiter. The scheduler has very little impact on it, since it involves in the arbitration only when there are latency-critical requests observed.

IV. Experiment Result

A. *Experiment Setting*

To verify our method, we implemented AHB [13] bus model with the proposed scheme in SystemC, with four masters and a single slave. For comparison purpose, we implemented round-robin (R-R) arbiter and fixed-priority (F-P) arbiter both with and without the scheduler. The workload used in this experiment is shown in TABLE I.

It is assumed that all masters require the same bandwidth, that is 25% for ideal bandwidth, but we choose 35% for each so that the total required bandwidth is 140% of the ideal bus bandwidth which is heavy enough to verify our method. The ideal bandwidth is defined as the bandwidth when the bus is fully utilized for data transfer without any protocol overhead or contention. The slave latency is assumed to be 8 cycles and, therefore, the minimum latency for a request is 9 cycles, 1 for the grant and 8 for the slave latency. The average burst length is 12, hence the average service time is 21 cycles, 9 for the minimum latency, and 12 for 12-beat transfer. On this basis, we set the latency constraint of M1 and M3 to 30 cycles. A request has to wait for 21 cycles, on average, when other requests are concurrently issued with this request and one of other requests is granted. Even when the request is granted at the next arbitration round, the master has to wait for 9 cycles to observe its first data coming from the slave. As far as M2 and M4 are concerned, we set much loose latency constraint which is 72 cycles. In other words, the requests from these masters can satisfy the latency constraint even when they lose the winning chances of the arbitration three times.

First, we will show the average latencies and the average violated cycles of all masters for six different arbitration schemes: R-R and F-P without scheduler, R-R and F-P with only “warning” state (labeled as RR-W and FP-W), and R-R and F-P with both “warning” and “emergency” state(labeled

TABLE I
Workload summary

	Bandwidth requirement	Burst length	Latency constraint(cycles)
M1	35%	8	30
M2	35%	8	72
M3	35%	16	30
M4	35%	16	72

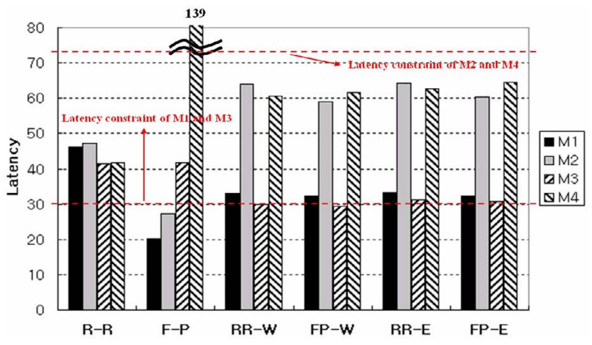


Fig. 5 Average latency in four different arbitration schemes

as RR-E and FP-E). Second, the longest violated cycles versus total bandwidth requirement will be shown. Third, we will show how well the proposed scheme reacts to the dynamic change of latency constraint of masters. All data are obtained by averaging the results from ten simulations.

B. *Experiment Result*

Fig. 5 shows the average latencies of all masters. Even though it seems that R-R shows the smallest overall latency, M1 and M3 do not meet its latency constraint by more than 10 cycles. On the other hand, the proposed scheme makes almost all masters meet the latency constraints except for M1 (which shows a slight violation).

Fig. 6 depicts the average violated cycles that are measured as additional cycles taken beyond the given constraints. It can be said that the proposed scheme balances out the violated cycles of the masters so that one master does not violate its latency constraint too much.

From Fig. 5 and Fig. 6, it is not clear to see the advantage of using an "emergency" state in addition to a "warning" state. It can be seen in Fig. 7 which shows the longest violated cycles versus total bandwidth requirement. The longest violated cycles for the R-R arbiter without scheduler are rapidly saturated to its maximum as the bandwidth requirement increases. On the other hand, RR-W and RR-E arbiter are saturated to their maximum values much more slowly than the R-R arbiter, and you can find that RR-W is worse than RR-E for all data points.

To see how well the proposed scheme supports the latency constraint change, we changed the latency constraints of M2 and M3 from 72 to 51 and from 30 to 51, respectively, in the middle of the simulation. Changing the latency constraints of masters does not violate or need extra signals added to AHB protocol. It can be done with a normal write transfer by treating

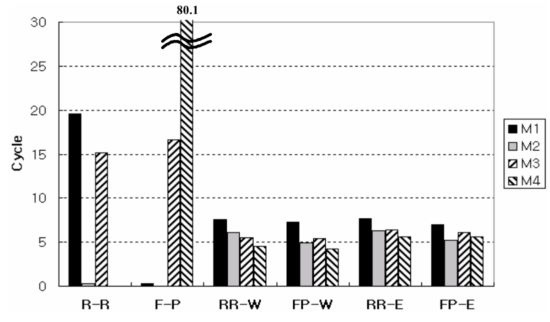


Fig. 6 Average violated cycles

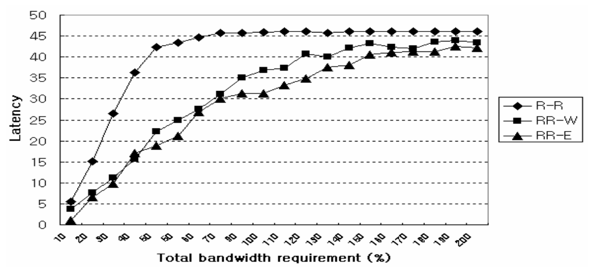


Fig. 7 Longest violated cycles versus total bandwidth requirement

the scheduler as a slave of the system. The result is shown in Fig. 8 and you can easily find the changed latency constraints are applied correctly.

To verify robustness of our method to the traffic variation, this time we change the bandwidth requirement of M3 in the middle of simulation. From 15000th cycle to 35000th cycle (between 3 and 7 on x-axis in Fig. 8), M3 changes its bandwidth requirement to 50%. Note that the default traffic is already quite heavy and, therefore, the increasing bandwidth requirement for M3 makes the traffic even heavier. The result is shown in Fig. 9. The R-R arbiter not only shows the highest latency at all data points, but also is much influenced by bandwidth requirement change. On the other hand, the proposed scheme is less influenced by the bandwidth requirement change.

Next, in order to show that scheduler rarely affects the bandwidth characteristics of the arbiter, the bandwidth utilization of R-R, RR-W, and RR-E is shown in Fig. 10. You can find that the bandwidth utilization is not or rarely degraded. This is because the scheduler in "warning" state does not affect the current transfer but only informs the arbiter which one to be granted next. Even in "emergency" state, in which the current transfer can be preempted and retransferred later, the number of retry is small, therefore, the bandwidth degradation is negligible.

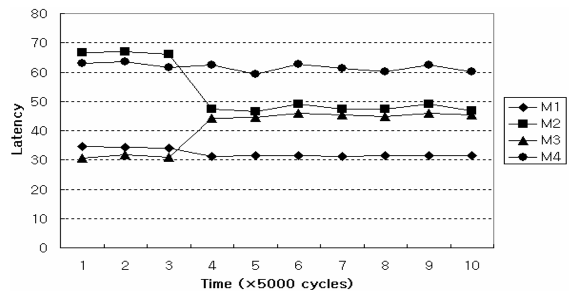


Fig. 8 Effect of latency constraints change

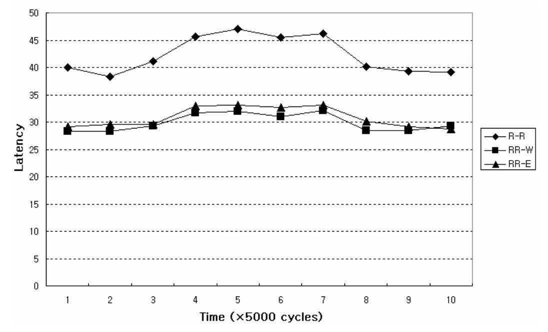


Fig. 9 The latency variation on a dynamic traffic pattern

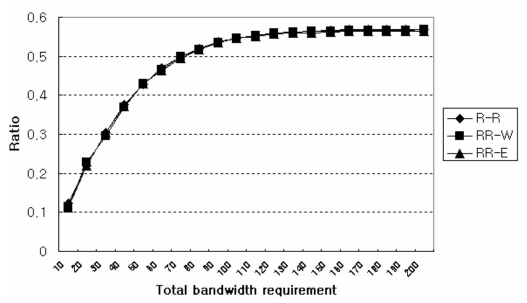


Fig. 10 Bandwidth utilization versus bandwidth requirement

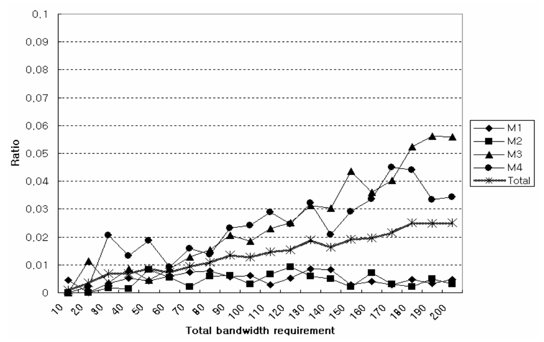


Fig. 11 The ratio of retried transfer to the total transfers

The ratio of the number of retried transfers to that of the total requests is shown in Fig. 11. The total retry ratio is bound to 2.5%, and the maximum retry ratio is 5.5% for M3, which is so small that its effect on total bandwidth utilization is negligible, as shown in Fig. 10.

V. Conclusion

We propose a latency-aware arbitration scheme which introduces a latency-aware scheduler in addition to an existing bandwidth-conscious arbiter. The bandwidth utilization is not degraded in "warning" state, and rarely degraded in "emergency" state, but making the latency properties even better, as shown in the experimental results. The scheduler can be augmented with any bandwidth-conscious arbiters such as round-robin, fixed priority, TDMA. Latency constraints of each master can be updated without additional signal wires but just as a normal write transfer to the scheduler. Also, it can be further extended to support multi-thread communication fabrics including Network-on-Chip (NoC) switching elements.

Acknowledgement

This work was partially supported by grant No. R01-2006-000-10156-0 from the Basic Research Program of the Korea Science & Engineering Foundation, by IT R&D Project funded by Korean Ministry of Information and Communications, and by The Korean Intellectual Property Office.

References

[1] A. Sangiovanni-Vincentelli, L. Carloni, F. De Bernardinis, and M. Sgroi, "Benefits and challenges for platform-based design", *DAC 2004*, pp.409-414, 2004.

[2] K. Keutzer, A. R. Newton, J. M. Rabaey and A. Saniovanni-Vincentelli, "System-level design: orthgonalization of concerns and platform-based design", *IEEE TCAD*, vol. 19, no. 20, pp. 1523-1543, 2002.

[3] U. Y. Ogras, J. Hu, and R. Marculescu, "Communication- centric SoC design for nanoscale domain", *ASAP 2005*, pp. 73-78, 2005.

[4] W. Cesario, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, and M. Diaz-Nava, "Component-Based Design Approach for Multicore SoCs", *DAC 2002*, pp. 789-794, 2002.

[5] K. Ryu and V. Mooney, "Automated Bus Generation for Multiprocessor SoC Design", *DATE 2003*, pp. 282-287, 2003.

[6] K. Lahiri, A. Raghunathan, and G. Lakshminarayana, "LOTTERYBUS: A new high-performance communication architecture for system on chip designs", *DAC 2001*, pp.15-20, 2001.

[7] W. D. Weber, J. Chou, I. Swarbrick, and D. Wingard, "A quality of service mechanism for interconnection networks in system on chips", *DATE 2005*, pp. 1232-1237, 2005.

[8] R. Lu and C.K. Koh, "SAMBA-BUS : A high Performance bus architecture for system on chips", *ICCAD 2003*, pp.8, 2003.

[9] K. Sekar, K. Lahiri, A. Raghunathan, and S. Dey, "FLEXBUS-A high performance system-on-chip communication architecture with a dynamically configurable topology", *DAC 2005*, pp571-574, 2005.

[10] A. Rădulescu, J. Dielissen, K. Goossens, E. Rijpkema, and P. Wielage, "An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration", *DATE 2004*, vol.2,, pp878-883, 2004.

[11] K. Kim, S.J. Lee, K. Lee, and H.J. Yoo, "An arbitration look-ahead scheme for reducing end-to-end latency in networks on chip", *IEEE*, 2005.

[12] M. Jun, K. Bang, H.J. Lee, and E.Y. Chung "Latency-aware Bus Arbitration for Real-time Embedded Systems", accepted for *IEICE* transaction.

[13] ARM, Limited. AMBA Specification, 1999.