IEICE TRANS. INF. & SYST., VOL.E90–D, NO.3 MARCH 2007

LETTER

Latency-Aware Bus Arbitration for Real-Time Embedded Systems

Minje JUN[†], **Kwanhu BANG**[†], **Hyuk-Jun LEE**[††], ***Nonmembers*, *and* Eui-Young CHUNG**[†*a)], ***Member***

SUMMARY We present a latency-aware bus arbitration scheme for real-time embedded systems. Only a few works have addressed the quality of service (QoS) issue for traditional busses or interconnection network. They mostly aimed at minimizing the latencies of several master blocks, resulting in decreasing overall bandwidth and/or increasing the latencies of other master blocks. In our method, the optimization goal is different in that the latency of a master should be as close as a given latency constraint. This is achieved by introducing the concept of "slack". In this method, masters effectively share the given communication architecture so that they all observe expected latencies and the degradation of overall bandwidth is marginal. The experimental results show that our method greatly reduces the number of constraint violations compared to other conventional arbitration schemes while minimizing the bandwidth degradation.

key words: *latency, arbiter, QoS, performance, bus, slack*

1. Introduction

Rapid growth in semiconductor technology is prompting the integration of billions of transistors on a single silicon die. System-on-Chip (SoC) is a typical example of such a trend and many embedded systems adopt SoCs as their processing core. The major challenge in SoC design is to meet the time-to-market (TTM) requirement due to increasing die size and design complexity. For this reason, the concept of design reuse is extremely important and platform-based design methodology has been widely accepted [1]. Even though IP components can be reused for different designs, their communication architecture should be carefully designed for the given constraints, since each design may have different performance specifications and constraints. Suppose an MPEG decoder which can handle various picture sizes. Digital Media Broadcast (DMB) applications require the decoder to process 15 frames (picture size: CIF) per second while a high-end Portable Media Player (PMP) requires the decoder to process 30 frames per second even with a larger picture size. Apparently, they need different communication bandwidths and latencies, hence the communication architecture should be tuned for each application. For this reason, communication architecture design is one of the time-consuming steps in SoC design [2].

Multi-layer bus architecture was proposed as a solution to satisfy the bandwidth and latency requirements, but the number of bus layers is limited by the routing density and external pin counts to access the external DRAM memories. In contemporary designs, external DRAM memories are widely used because they are affordable. Even in multi-layer architecture, we need an arbitration scheme to handle the concurrent data transfer requests from multiple masters for each layer.

Classical arbitration schemes are round-robin and fixed priority. The former is appropriate when masters have the same bandwidth requirements while the latter is appropriate when some masters require a higher bandwidth than others. A hybrid scheme of these two is also widely used. However, they often ignore the latency issue or assume that the latency requirement is proportional to the bandwidth requirement. The Time-Division-Multiple-Access (TDMA) scheme is another bandwidth-conscious arbitration scheme. It provides guaranteed throughput but increases the overall throughput by allowing the unused timing slots to be filled with best effort traffic. Recently, latency-conscious [3]–[6] or QoS-aware [4], [7], [8] arbitrations are being introduced. LOTTERYBUS [3] resolves the long latency issues by enhancing the TDMA scheme with a statistical method. It allocates tickets to masters and a master can take the bus with a probability proportional to the number of given tickets. The arbiter selects a master based on the probabilities assigned to masters. Weber et al. proposed a QoS-aware arbitration scheme where they introduced the concept of "epoch" [4]. 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) on a per-epoch basis. These approaches resolve the latency issues by minimizing the latency. Some other works improved the latency and bandwidth efficiency by introducing a topological change while using conventional arbitration schemes such as round robin [5], [6]. SAMBA BUS allows multiple masters to share the same bus when their targets are different [5]. FLEXBUS [6] provides a reconfigurable feature to dynamically alter bandwidth and latency requirement with using 'reconfiguration unit.' Again, these methods aim at minimizing latency for QoS. The limitation of previous approaches is that excessive reduction of latency of a certain master may increase the latencies of other masters. To avoid such weakness, we propose a scheduler by using the concept of "slack". The goal of our method is not to minimize the latency but to adjust it to the given latency constraint by minimizing the slack with minor hardware overhead. Note that we do not aim at replacing the existing arbiters with our

Manuscript received July 11, 2006.
Manuscript revised September 13, 2006.
[†]The authors are with Yonsei University, Seoul, Korea.
[††]The author is with Cisco Systems Incorporation, CA, USA.
[*]Corresponding author.
a) E-mail: eychung@yonsei.ac.kr
DOI: 10.1093/ietisy/e90–d.3.676

Copyright © 2007 The Institute of Electronics, Information and Communication Engineers

method, but providing a complementary solution to the existing arbiters. More precisely, our scheduler can be added to any existing arbiters, such as round-robin, fixed-priority, TDMA and LOTTERY arbiter. Our scheme improves the latency characteristics of existing arbiters while rarely altering the bandwidth characteristics.

Section 2 describes the basic architecture of our arbitration scheme and the experimental results are shown in Sect. 3 followed by conclusions in Sect. 4.

2. Latency-Aware Arbiter

Figure 1 shows the architecture of the proposed latency-aware arbiter.

As shown in Fig. 1, the arbiter consists of a conventional bandwidth-conscious arbiter and a scheduler. Latency-critical masters set the latency registers in a scheduler to their latency constraints. Whenever a master issues a request, the scheduler computes the slack using Eq. (1) and loads it into the corresponding slack counter.

$$Slack_i = L_i - B_i \times T_i - S_j \tag{1}$$

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.

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.

The slack counter is decremented every clock cycle and thus the scheduler updates slack information every clock cycle. Note that the latency constraint can be dynamically set whenever a master needs to change it to perform a different operation.

A global threshold value is programmed and a comparator compares the slack of a request to the threshold value. If it is less than or equal to the threshold, the scheduler tells the bandwidth-conscious arbiter to preempt the other requests. If the slacks of all pending requests are larger than the threshold, the conventional bandwidth-conscious arbiter solely determines the control, since there is no intervention from the scheduler. More precisely, our arbiter has a two-level hierarchy. The first level is the conventional bandwidth conscious arbiter which arbitrates the requests when there is no request from the second level arbiter. The second level arbiter is the scheduler. It does not perform the arbitration itself. Instead it preempts the scheduling of the first level arbiter if the slack is equal to or less than the threshold.

Note that the request informed by the scheduler cannot stop the current data transfer and should wait until the current transfer is completed. Also, when there are multiple requests whose slacks are equal to or less than the threshold, there is no way to schedule all the requests to meet their constraints. The scheduler selects the request which has the smallest slack and informs the arbiter. For these reasons, the proposed arbitration scheme is more appropriate for soft real-time constraints rather than hard real-time constraints. However, it is true that most of IP components have input/output buffers to avoid the disaster due to constraint violation. Thus, our arbitration scheme can be applicable to hard real-time applications if the violated interval is reasonably small so that the input/output buffers can absorb the overflow.

Even though the proposed scheme seems to have a large hardware overhead, latency constraints are at most a few hundred cycles and each register needs less than 10 bits in most cases. Furthermore, latency registers and slack counters need to be allocated only for latency-critical masters. Also, *Ti* is mostly 1 cycle in most cases and thus multipliers are not needed. To summarize, the proposed scheme requires two subtractors, one latency register, one slack register, and a comparator per a single latency critical master with some glue logics.

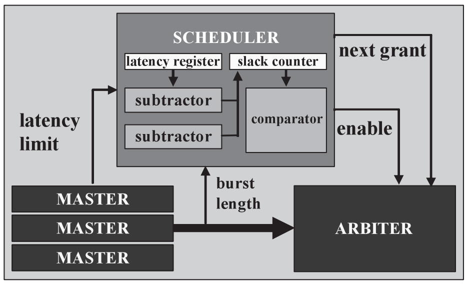

Fig. 1 Architecture of latency-aware arbiter.

3. Experimental Results

To assess the effect of the proposed arbitration scheme, we implemented AHB bus models [9] and our arbitration scheme in SystemC. More precisely, we implemented two versions of latency-aware arbitration schemes – a round-robin (R-R) arbiter with the scheduler and a fixed-priority (F-R) arbiter with the scheduler. We also implemented a stand-alone round-robin and fixed priority arbiter without the scheduler for comparison.

The workload used in this experiment is summarized in Table 1. The aggregated bandwidth requirement from all masters is set to 150% of ideal bandwidth which is heavy enough to appreciate the benefit of the proposed scheduler. The traffics for four masters are uniformly distributed so

Table 1 Workload summary.

Master	Bandwidth requirement	Latency constraint (cycles)
M1	60%	26
M2	60%	60
M3	15%	26
M4	15%	60

that the bus contention ratio is very high. The burst length (HBURST signal in AHB) of all requests is assumed to be 8 cycles and the slave service time (or latency) is set to 8 cycles. Also, we assume all requests are for read transfer. Therefore, the minimum latency of a request is 9 cycles which is the sum of 1 cycle for grant and 8 cycles for slave latency, and the minimum service time is 17 cycles, the sum of minimum latency and 8 cycles for 8-beat transfer. Since the latency constraints of M1 and M3 are 26, which is the sum of the minimum service time and the slave latency, at most 1 transfer can be completed while one of them is requesting, which means M1 and M3 must get into scheduling condition as soon as they request. For this reason, we set the threshold value to the same as the latency constraints of M1 and M3 so that any request is completed within the given timing constraint.

In this setting, M1 is the most critical master since it requires high bandwidth with low latency. M2 is also demanding from a bandwidth perspective, but its latency constraint is relatively loose. M3 is the opposite case where the bandwidth requirement is relatively low while its latency constraint is tight. M4 is the least important master due to its low bandwidth requirement and a loose latency constraint. In case of fixed-priority schemes (for both with the scheduler and without the scheduler), we assign the highest priority to M1 and the lowest priority to M4. As far as M2 and M3 are concerned, M2 is assigned to a higher priority than M3 in Fixed Priority 1 (F-P1). In Fixed-Priority 2 (F-P2), M3 is assigned to a higher priority than M2 to give priority to a latency sensitive master.

Figure 2 shows the average latency of each master in four different arbitration schemes. Although it seems that the R-R shows a minimum overall latency, the average latencies of M1 and M3 are over their constraints while those of M2 and M4 have a large margin to their constraints. In the F-Px cases, M4 and M3 (only in F-P1) violate their constraints quite a bit while the latencies of M1 and M2 are over-minimized. On the other hand, the arbiters with the proposed scheduler nicely mitigate the latency issues by distributing the bus ownership based on the slack information.

Even though the average latencies for the proposed scheduler are within given constraints, individual requests may violate the given constraint. Table 2 shows the percentage of total requests that violate constraints and the longest violated cycles of all masters for the different arbitration schemes.

Table 2 shows that the proposed scheme has not much effect over R-R, but greatly improves over F-P cases. R-S shows a little degradation for M2 and M4 over R-R in terms of longest violated cycles. Even though the violation ratio is quite high, the proposed scheduler balances the violated cycles so that a certain master does not violate its constraint too much. It is worth to repeat that our method is aiming at soft real-time constrained applications rather than the hard real-time applications, thus the ratio of violation is much less important than the violated cycles. Figure 3 shows the average violated cycles of each master beyond its latency constraints. It shows that the arbiters with the proposed scheduler have much smaller deviation from the given latency constraints. This is possible by considering the slack in addition to the bandwidth requirement. Considering the results from Table 2 and Fig. 3, and the fact that the masters can change their characteristics including the latency constraints, we could use this feature to bound the violated cycles for different masters. Bounding upper-limit of violation may allow system designer to choose more reasonable buffer size to each master.

To analyze the impact on the existing arbitration schemes, we also implemented TDMA arbitration scheme with and without our scheduler and compared it with fixed-priority scheme. In this experiment, we set the ratio of bandwidth requirements as 4 : 4 : 1 : 1 for M1, M2, M3 and M4, respectively. Also, the latency constraints are same to those given in Table 1.

Table 2 Latency violation statistics for each scheme.

Scheme	ratio of violations (%)				longest violated cycles			
	M1	M2	M3	M4	M1	M2	M3	M4
R-R	69.4	0	64.6	0	34	0	34	1
F-P1	0	0	85.5	85	0	0	523	3365
F-P2	0	17.7	36.4	81.8	0	164	17	1558
R-S	38.8	29.8	51.7	43.6	34	33	34	32
FP1-S	39	29.2	53.5	47.2	34	34	34	33
FP2-S	38.3	29.2	52.6	46.5	34	34	34	32

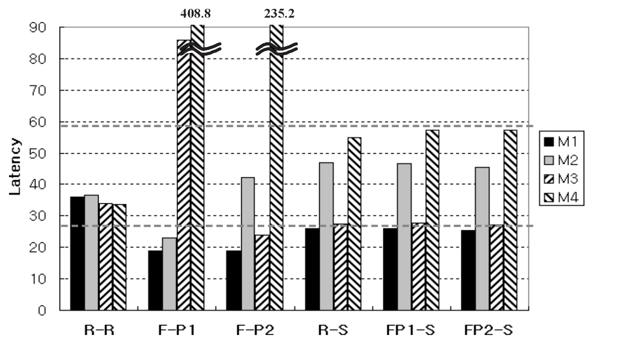

Fig. 2 Average latency of four arbitration schemes. (R-R: Round-Robin, F-P: Fixed-Priority R-S: R-R with Scheduler, FPx-S: F-Px with Scheduler)

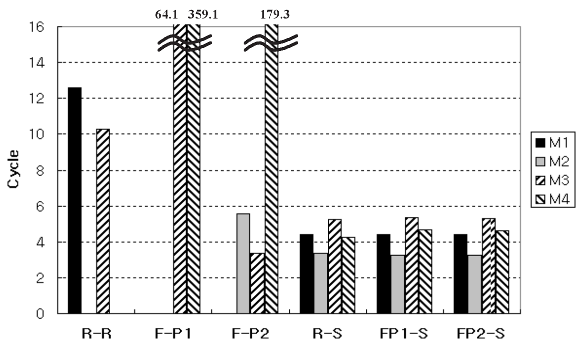

Fig. 3 The average violated cycles.

Table 3 Comparison of TDMA and fixed-priority scheme. (TDMA-s: TDMA with Scheduler)

	M1		M2		M3		M4	
	BW (%)	Late -ncy	BW (%)	Late -ncy	BW (%)	Late -ncy	BW (%)	Lat -ncy
F-P1	57.3	10.4	28.7	37.3	10.5	42.3	3.5	343
FP1-S	51.6	14.5	26	45.6	13	22.1	9.7	64
TDMA	40.2	23.2	39.8	23.4	10	61.2	10	62
TDMA-S	40	23	37.1	27.7	12.2	27.9	10.6	53.8

We have three major observations from this experiment as shown in Table 3. First, it was shown that TDMA outperformed the fixed-priority scheme from the bandwidth perspective when they are not augmented with the scheduler as expected. Second, the bandwidth allocation ratio of both arbitration schemes is slightly affected by the proposed scheduler. In other words, our scheduler rarely alters the bandwidth characteristics of the given arbiter. Third, the latency violations are greatly reduced with our scheduler as shown in M4 (fixed-priority) and M3 (TDMA). This is because our scheduler only reorders the requests based on the slack while the total number of requests served per master remains the same.

4. Conclusions and Future Work

We propose a latency-aware arbitration scheme which introduces a latency-aware scheduler in addition to an existing bandwidth-conscious arbiter. Our scheduler rarely affects on the bandwidth characteristics of the given arbitration scheme, while its latency constraint satisfaction is improved by the added scheduler as shown in the experimental results. The scheduler can be augmented with any bandwidth-conscious arbiter such as TDMA and LOTTERY. The arbitration scheme can be further enhanced from a latency perspective by utilizing “retry” and/or “split” that are supported by many contemporary bus protocols. Also, it can be further extended to support multi-thread communication fabrics including Network-on-Chip (NoC) switching elements.

Acknowledgments

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, by IDEC (IC Design Education Center), and by The Korean Intellectual Property Office.

References

[1] A. Sangiovanni-Vincentelli, L. Carloni, F.D. Bernardinis, and M. Sgroi, “Benefits and challenges for platform-based design,” DAC, pp.409–414, 2004.
[2] U.Y. Ogras, J. Hu, and R. Marculescu, “Communication-centric SoC design for nanoscale domain,” ASAP 2005, pp.73–78, 2005.
[3] K. Lahiri, A. Raghunathan, and G. Lakshminarayana, “LOTTERYBUS: A new high performance communication architecture for system on chip designs,” DAC, pp.15–20, 2001.
[4] W.D. Weber, J. Chou, J. Swarbrick, and D. Wingard, “A quality of service mechanism for interconnection networks in system on chips,” DATE, pp.1232–1237, 2005.
[5] R. Lu and C. Koh, “SAMBA-BUS: A high performance bus architecture for system on chips,” ICCAD, 2003.
[6] K. Sekar, K. Lahiri, A. Raghunathan, and S. Dey, “FLEXBUS: A high performance system-on-chip communication architecture with a dynamically configurable topology,” DAC, pp.571–574, 2005.
[7] 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, vol.2, pp.878–883.
[8] K. Kim, S. Lee, K. Lee, and H. Yoo, “An arbitration look-ahead scheme for reducing end-to-end latency in networks on chip,” IEEE Circuits and Systems, pp.2357–2360, 2005.
[9] ARM, Limited. AMBA Specification, 1999.