Pipelined Bidirectional Bus Architecture for Embedded Multimedia SoCs

Gang-Hoon Seo, Won-Yong Jung, Seongsoo Lee, and Jae-Kyung Wee

School of Electronics Engineering, Soongsil University, 156-743, Korea
wjk@ssu.ac.kr

Abstract. This paper proposes novel high-performance bus architecture for memory-intensive embedded multimedia SoCs. It has a pipelined bidirectional bus for high speed and small area. It has two separate bus called system bus and memory bus, where memory-intensive IPs are connected to memory bus so not to degrade system bus performance. To avoid starvation of low-priority masters, the proposed bus exploits probability-based arbitration policy where the arbitration probability of each master is determined in proportion to its execution time. To increase transmission bandwidth, it also exploits bus partitioning where several masters often access their slaves concurrently without multilayer structure. The proposed bus is designed, implemented, verified, and evaluated in hardware level. Simulation results show that the proposed bus improves effective bandwidth by 2.8~3.6 times and communication latency by 3.1~4.7 times when compared to AMBA bus.

1 Introduction

Recently, advances of integrated circuit technology have enabled development of complex embedded systems into systems-on-chip (SoC) [1], where pre-designed IP cores are stitched together through various communication links. Design of high performance global communication architecture has become a key to successful embedded SoC designs. On-chip buses [2]-[6] are often used for on-chip communication between internal IP cores.

On-chip buses are classified into standard buses [2]-[5] and wrapper-based buses [6]. Standard on-chip buses specify its own protocol over wiring connections between IP cores. Any IP cores complying that protocol can be reused in other embedded SoCs using same bus type. However, interface logic should be redesigned when different bus type is used. Wrapper-based approaches use IP core interface protocol independent of physical bus protocol, and they use hardware wrappers to handle core-to-core communication. Hence, IP cores complying with the interface protocol can be integrated into embedded SoC, even if their physical bus types are different. However, complex wrapper architecture is required to reduce access latency. In

This work was supported by grant No. R01-2005-000-10540-0 from the Basic Research Program of the Korea Science and Engineering Foundation.

practical applications, wrapper-based buses such as virtual component interface (VCI) and open core protocol (OCP) are widely used.

Recently, embedded multimedia systems are often designed into SoCs based on IP cores. Most multimedia applications store large amount of picture data in the external memory. Therefore effective on-chip bus architecture is required to handle large memory access. In this paper, novel wrapper-based on-chip bus architecture is proposed for embedded multimedia SoCs. It has a separate memory bus so that it can handle large memory bandwidth and frequent memory access. It has a pipelined architecture to support multitasking on a single layer bus, i.e. in some cases several masters can access several slaves simultaneously. It has a probability-based arbitration policy to avoid starvation of low-priority masters.

2 Problems of Conventional Bus Architectures

AMBA bus [2] is one of the most popular on-chip bus architecture in IP-based embedded SoCs. It consists of AHB bus and APB bus, where the former is designed for high-performance operation and the latter is designed for low-performance external devices. AMBA bus shows good performance in on-chip IPs. However, it suffers from the following problems when applied to multimedia applications such as MPEG.

AMBA bus shows serious performance degradation in external memory with long or variable latency time. For example, MPEG system often requires external SDRAM memory to store large picture data. Therefore, SDRAM is connected with AHB bus due to huge memory access. However, SDRAM is accessed so frequently in the multimedia applications, and it has long latency time from the master's request. In AMBA bus, other IPs cannot use the bus when one IP already occupies it. Therefore, the IP accessing SDRAM occupies AHB bus during most of the operation time, and other IPs hardly share AHB bus. This seriously reduces the efficiency of AMBA bus. To overcome this problem, AMBA bus provides multilayer scheme to support multi-masters-to-single-slave connection. However, as the number of masters increases, bus area becomes large and wiring scheme becomes complicated. Furthermore, in the multilayer scheme, complex arbiter hardware is required because AMBA bus has only one arbiter that decides the priority of multi-masters.

Sonics bus [4] makes arbitration based on time schedule (TDMA). When SDRAM is connected to Sonics bus, SDRAM interface can directly require a master to read or write in order to prevent delaying due to long latency time of SDRAM. Sonics bus is easier to optimize than AMBA bus, and IP reuse is also easier because it supports OCP. However, very complicated SDRAM interface is required to increase bus efficiency, and the very detailed design information of IPs is also required.

SAMBA bus [7] is a synchronous bi-directional bus with multitasking. It shows better bus bandwidth over AMBA. It also reduces performance degradation due to long arbitration latency. However, it shows long latency time to read data from external memory, and it is not suitable for multimedia applications.

In segment bus [8], the response of slow slaves may cause severe performance degradation because it remains idle while the master is waiting for the response. Split

transactions of segment bus are used to avoid such performance degradation. However, in split transactions, the bus access right of the master is released after the slave obtains the communication request. The slave has to initiate a new bus transaction to transfer the response. Therefore, split transactions lead to an increase in arbitration complexity. Also, it can not assure the compatibility of the developed IPs with the conventional chip architectures.

3 Proposed Bus Architecture

3.1 Overview

In this paper, we propose a hierarchical bus structure for multimedia applications. Fig. 1 illustrates the proposed bus architecture. It exploits probability-based arbitration policy to avoid starvation of low-priority masters, which is described in the next section. It consists of two separate buses, i.e. system bus (SBUS) and memory bus (MBUS). SBUS is a bi-directional bus with pipelined architecture to support multitasking. MBUS is a multi-masters-to-single-slave single layer bus, and it has an efficient communication protocol with external SDRAM memory. External SDRAM memory and IPs with large memory access is connected to MBUS. When an IP directly accesses SDRAM via MBUS, other IPs can communicate with each other via SBUS, and it significantly reduces the performance degradation from memory access. SBUS and MBUS communicate with each other via memory bus bridge (MBridge).

The proposed bus architecture has three types of arbiters. Local arbiter (LArbiter) is attached to each master in SBUS. It performs arbitration based on TDMA. The priority is calculated from the pre-design analysis of each master. Central arbiter (CArbiter) performs channel control of SBUS. It grants approval of each master according to the channel status when local arbiter requests channel access. Control signals of LArbiters and CArbiter have short latency time, since they are separate from command transfer signals, address transfer signals, and data transfer signals. LArbiters and CArbiter are not integrated in order that additional masters can be

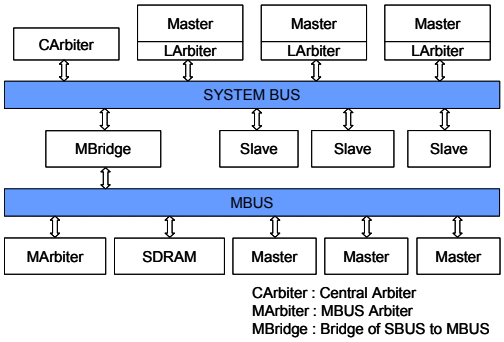


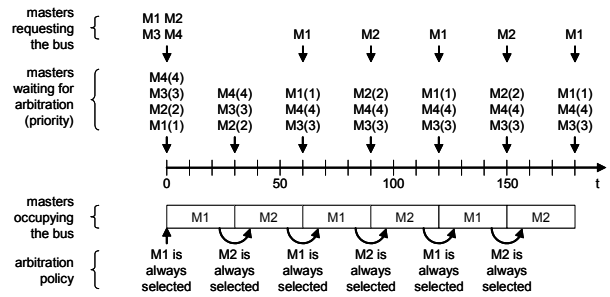
Fig. 1. Proposed bus architecture

easily inserted. MBUS arbiter (MArbiter) controls MBUS when a master in MBUS accesses external SDRAM memory. When a master in SBUS accesses SDRAM, it communicates with MBridge via SBUS, and then MBridge accesses SDRAM via MBUS. MBridge acts as a slave in SBUS and a master in MBUS.

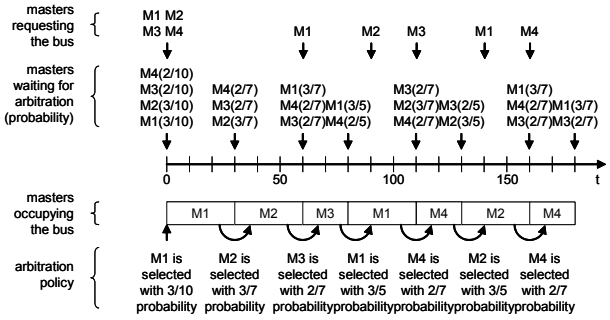
3.2 Probability-Based Arbitration Policy

AMBA bus exploits priority-based arbitration policy. Each master has its own fixed priority, and a master with higher priority is selected among waiting masters. Unfortunately, in AMBA bus, masters with lower priority sometimes fall into starvation. For example, assume that (1) there are four masters M_i (i=1..4) where lower i has higher priority, (2) their execution times are 30ms, 30ms, 20ms, and 20ms, respectively, and (3) they send next bus requests at 30ms after they finish current execution. As shown in Fig. 2 (a), when M_1, M_2, M_3, and M_4 send bus requests simultaneously at t=0ms, M_1 occupies the bus at t=0ms and M_2 occupies it at t=30ms. At t=60ms, M_1 sends next request and it occupies the bus again. At t=90ms, M_2 sends next request and it occupies the bus again. In this way, M_3 and M_4 can hardly access the bus and they fall into starvation, unless M_1 and M_2 stop sending their requests.

To avoid this problem, the proposed bus architecture exploits probability-based



(a) Priority-based arbitration



(b) Probability-based arbitration

Fig. 2. Probability-based arbitration policy

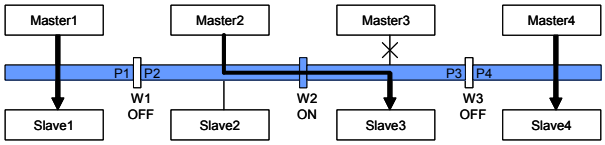
arbitration policy, where the arbitration probability of each master is determined in proportion to its execution time. In the previous example, M_1, M_2, M_3, and M_4 get the bus at t=0ms with the probability of 3/10, 3/10, 2/10, and 2/10, respectively, as illustrated in Fig. 2 (b). Assume that M_1 and M_2 are determined to get the bus at t=0ms and t=30ms, respectively. At t=60ms, M_1, M_3 and M_4 wait for bus arbitration, and M_3 and M_4 still have chance to get the bus with the probability of 2/7 and 2/7, respectively. Assume that M_3 and M_1 are determined to get the bus at t=60ms and t=80ms, respectively. At t=110ms, M_2, M_3 and M_4 wait for bus arbitration, and M_4 still have chance to get the bus with the probability of 2/7. Thus, starvation hardly occurs. The probability-based arbitration is implemented by simple logic gates with negligible hardware overhead.

3.3 Bus Partitioning for Concurrent Multiple Master-to-Slave Communications

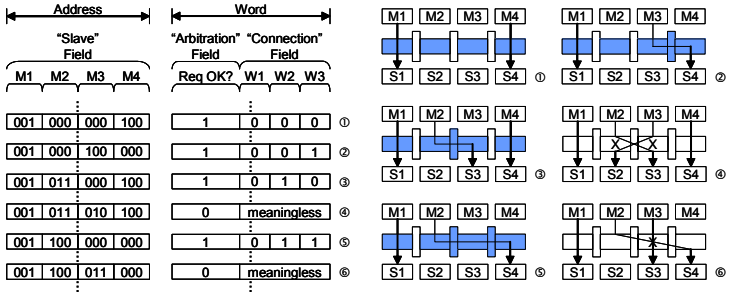
In AMBA bus, several masters and slaves are connected to a common bus. Consequently, when a master accesses a slave, other masters cannot access other slaves even if each master tries to access different slave, as shown in Fig. 3 (a).

In the proposed bus architecture, we propose bus partitioning as illustrated in Fig. 3

(a) Non-partitioned bus



(b) Bus partitioning



(c) ROM contents and bus operation

Fig. 3. Bus partitioning for concurrent multiple master-to-slave communications

(b). The proposed bus is partitioned into N partitions, where N is the number of masters. Each partition is connected via N-1 partition switches. When a switch is off, partitions in the left and right of the switch act as two separate buses. Therefore, more than two masters can access their slaves concurrently. In Fig. 3 (b), there are 4 masters (master1, master2, master3, and master4) and 4 slaves (slave1, slave2, slave3, and slave4). The bus is divided into 4 partitions (P1, P2, P3, and P4), and they are connected via 3 partition switches (W1, W2, and W3). When master1, master2, and master4 try to access slave1, slave3, and slave4, respectively, the bus arbiter opens W1 and W3, and it closes W2. Thus, the bus acts as three separate buses (P1, P2+P3, and P4), and three masters access three slaves concurrently (master1→slave1, master2→slave3, and master3→slave4).

Proposed bus arbitration is performed as follows. Assume the followings: (1) there are M masters denoting master$i$ (i=1..M) and S slaves denoting slave$i$ (i=1..S), (2) the bus is divided into max(M, S) partitions denoting Pi (i=1..max(M, S)) and they are connected via max(M, S) – 1 switches denoting Wi (i=1..max(M, S) – 1), (3) c connections denoting $C_i(m_i, s_i)$ (i=1..c, c<M) are made currently, where $C_i(m_i, s_i)$ indicates that masterm_i is connected to slaves_i. When masterm_k tries to access slavem_k, it can get bus arbitration when Eqn. (1) is satisfied. Otherwise, masterm_k should wait for bus arbitration until Eqn. (1) is satisfied.

$(m_k > m_i$ and $s_k > s_i)$ or $(m_k < m_i$ and $s_k < s_i)$ for all connections C_i (i=1..c, c<M) (**1**)

When masterm_k gets bus arbitration, the bus arbiter opens W*left* and W*right*, and it closes W*mid*, where *left*, *right*, and *mid* are given as Eqn. (2)-(4).

$$left = \min(m_k, s_k) \quad (\mathbf{2})$$

$$right = \max(m_k, s_k) \quad (\mathbf{3})$$

$$mid = (left + 1) \,..\, (right - 1) \quad (\mathbf{4})$$

Practically, the proposed bus arbitration is implemented by simple ROM as illustrated in Fig. 3 (c). ROM address is $\{M \times \lceil \log_2(S+1) \rceil\}$ bits and ROM word is max(M, S) bits, where M and S are the number of masters and slaves, respectively. 'Slave' field indicates the slave ID in connection, and 0 means that the corresponding master accesses no slave currently. 'Arbitration' field indicates whether the requested access is possible or not. 0 means that such request is impossible and the requested master should wait for next chance. 'Switch' field indicates the partition switch status, and 0 means that the corresponding partition switch is off. When there are 4 masters and 4 slaves, ROM size is only 12bit address × 4 bit word, which is quite small. Even when there are 16 masters and 16 slaves, ROM size is only 80bit address × 16 bit word, which is quite tolerable.

3.4 Bus Architecture

Fig. 4 shows the SBUS architecture for command signals, address signals, and data signals. When a master transmits a request, LArbiter sends a 'Request' signal to CArbiter. After CArbiter checks channel status, it sends 'Done' and 'Accept' signals

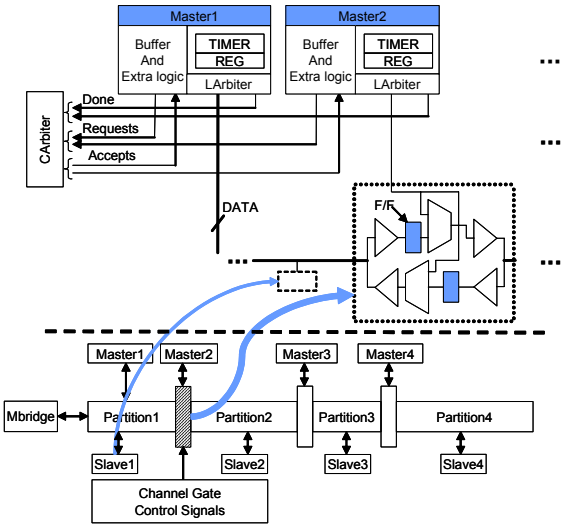


Fig. 4. SBUS architecture

to the master. Simultaneously, channel gate signals of CArbiter control channel gates with pipelined architecture. Dashed block in Fig. 4 is a transmission control unit for command signals, address signals, and data signals. It consists of multiplexers, flip-flops, and tri-state buffers. It exploits bidirectional transmission for wiring efficiency. Pipelined structure with flip-flops is also employed for performance improvement and accurate data transmission. Masters in SBUS communicate with SDRAM via MBridge and access latency time is different for each master. Leftmost master in SBUS has minimum latency time, so the master accessing SDRAM most frequently should be assigned to partition1. Therefore, when designing SBUS, masters should be assigned according to the descending order of SDRAM access frequency.

When LArbiter of a master sends 'request' signal to CArbiter, CArbiter looks up its internal ROM. When its 'arbitration' field is 1, CArbiter sends channel gate control signals to open or close partition switches. Partition switch is tri-state buffers in Fig. 4, and CArbiter directly controls it to reduce latency time. At the same time, it sends 'accept' signal to the master. Then, the master begins data transmission.

3.5 Bus Operation

There are two modes in SBUS bus protocols, i.e. request mode and response mode. Request mode is used when a master sends data to a slave, and response mode is used when a slave sends data to a master. These two modes have symmetrical structures with command bus, address bus, and data bus. SBUS has four direct control signals. DCS1 is 'request' signals from LArbiter to CArbiter. DCS2 is 'accept' signals from CArbiter to master. DCS3 is 'done' signals from slave to CArbiter. DCS4 is channel control gate signals (CGCS) from CArbiter to partition switch. Bit width of DCS1,

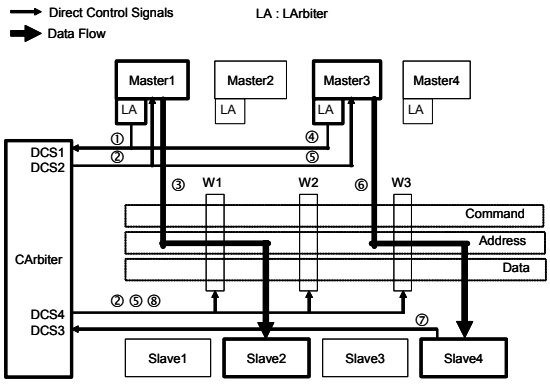


Fig. 5. Data flow of concurrent multiple master-to-slave communications

DCS2, DCS3, and DCS4 are *M* bits, *M* bits, *S* bits, and 4*P* bits, respectively, where *M*, *S*, *P* are the number of masters, slaves, and partitions, respectively.

Fig. 5 illustrates data flow of concurrent multiple master-to-slave communications when master1 sends data to slave2 and master3 sends data to slave4. When master1 tries to send data to slave3, master1 sends 'request' signal to CArbiter (①). CArbiter checks if such arbitration is possible, and sends 'accept' signals to master1 (②). At the same time, CArbiter sends CGCS to open or close partition switches (②). In this case, partition gates W1, W2, and W3 are on, off, and off, respectively. After that, data is transmitted from master1 to slave2 (③).When master3 tries to send data to slave4, master3 sends 'request' signal to CArbiter (④). CArbiter checks if such arbitration is possible, and sends 'accept' signals to master3 (⑤). At the same time, CArbiter updates CGCS to open or close partition switches (⑤). In this case, partition gates W1, W2, and W3 are on, off, and on, respectively. After that, data is transmitted from master3 to slave4 (⑥). When the transmission from master3 to slave4 finishes, slave4 sends 'done' signal to CArbiter (⑦). Then CArbiter updates CGCS to open or close partition switches (⑧). In this case, partition gates W1, W2, and W3 are on, off, and off, respectively.

4 Simulation Results

To evaluate the performance of the proposed bus architecture, we designed both AMBA bus and the proposed bus, and implemented them with Verilog HDL. They were verified and simulated on hardware level, and their performances were evaluated for comparison. We used SDRAM as external memory.

The number of masters, slaves, and partition are assumed as 3, 4, and 4, respectively. We assumed that the workload of master1, master2, master3, and master4 are 40%, 30%, 20%, and 10%, respectively. We used three test patterns, i.e. uniform traffic pattern, non-uniform traffic pattern, and random traffic pattern. In the uniform traffic pattern, each master accesses 4 slaves equally. Master1 equally accesses slave1, slave2, slave3, and slave4 in 25%. Master2, master3, and master4

access slaves in a same way. In the non-uniform traffic pattern, each master accesses adjacent slave more frequently than other slaves. Master1 accesses slave1 in 85% and it equally accesses slave2, slave3, and slave4 in 5%. Master2 accesses slave2 in 85% and it equally accesses slave1, slave3, and slave4 in 5%. Master3 and master4 access slaves in a same way. In the random traffic pattern, each master accesses 4 slave randomly. Bit width of data bus and address bus is 32 bits. TDMA is exploited as priority scheme. In multimedia applications, large amount of picture data is sequentially accessed in frame memory, so the transmission is modeled as burst mode with 16 increments. Inter-communication interval is 3, 6, and 9. We used two performance measures to evaluate the proposed bus architecture. Normalized effective bandwidth is defined as the normalized value of (amount of transmitted data) / (elapsed bus cycles) with AMBA bus. Average communication latency is the average of latency cycles from bus request to bus accept.

Fig. 6 shows the simulation results. Both AMBA bus and the proposed bus are designed in Verilog HDL and synthesized on Synopsys CAD tools supported by IC Design Education Center. The bus operation is simulated in Synopsys CAD tools, not in C++. Therefore, the simulation results consider all hardware effects including gate delay, bus line delay, buffer operation, and pipelining. Note that the simulation results of AMBA bus are identical regardless of traffic patterns. In AMBA bus, only one master can access a slave at a time and there is no difference in bus performance which slave a master accesses. We performed 320 bursts from every 4 masters to every 4 slaves, and the amount of transmitted data is 320 (burst number) × 32 bits (bandwidth) × 16 (16 increments burst) × 4 (masters) × 4 (slaves) = 327,680 bytes. In this case, AMBA bus takes about 20,000 cycles. On the contrary, the proposed bus takes about 7,250 cycles in the uniform traffic pattern. Consequently, the proposed bus achieves 2.8 times speed-up in the uniform traffic pattern. Similarly, it achieves 3.6 and 2.8 times speed-up in the non-uniform traffic pattern and random traffic pattern, respectively. As for average communication latency, the proposed bus achieves 3.1, 4.7, and 3.3 times speed-up in the uniform traffic pattern, non-uniform

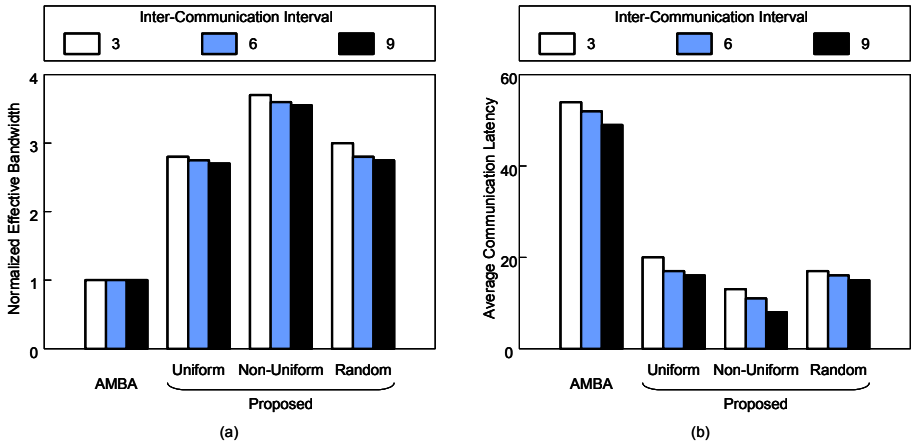


Fig. 6. Simulation results (a) Normalized effective bandwidth (b) Average communication latency

traffic pattern, and random traffic pattern, respectively. In the simulation results, the proposed bus shows best performance improvement in the non-uniform traffic pattern. This is due to the fact that the bus partitioning shows better performance when a master accesses its adjacent slaves more frequently than other slaves.

5 Conclusion

This paper proposes novel bus architecture for embedded multimedia SoCs. It consists of two separate buses called system bus (SBUS) and memory bus (MBUS) for effective memory access. It effectively handles huge memory bandwidth and frequent memory access of multimedia applications, since most memory-intensive IPs are connected to MBUS and they hardly disturb SBUS operations. The proposed bus architecture exploits probability-based arbitration policy, and masters with low priority hardly fall into starvation. It exploits bus partitioning to enable concurrent multiple master-to-slave communications to increase transmission bandwidth. It exploits pipelined bidirectional architecture to achieve high performance. It supports VCI and OCP to reuse IPs easily. The proposed bus architecture is designed, implemented, verified, and evaluated in hardware level. From the simulation results, it shows 2.8~3.6 times speed-up in effective bandwidth and 3.1~4.7 times speed-up in communication latency. Especially, it shows significant performance improvement when each master accesses its adjacent slave than other slave, which is typical in embedded multimedia SoCs.

References

1. Keating, M., Bricaud, P.: Reuse Methodology Manual for System-on-a-Chip Designs, Kluwer Academic Publishers (1998).
2. ARM: AMBA Specification Overview, http://www.arm.com/ Pro+Peripherals/AMBA.
3. IBM: CoreConnect Bus Architecture, http://www.chips.ibm.com/products/coreconnect/docs/cron_wp.pdf.
4. Sonics: Sonics μNetwork Technical Overview, http: http//:www.sonicsinc.com/Documents/Overview.pdf.
5. OCP International Partnership: Open Core Protocol Specification, http://www.ocpip.org.
6. Anjo, K., Okamura, A., Motomura, M.: Wrapper-Based Bus Implementation Techniques for Performance Improvement and Cost Reduction, IEEE Journal of Solid-State Circuits **35** (2004) 804-817.
7. Lu, R., Koh, C.-K.: SAMBA-bus: A High Performance Bus Architecture for System-on-Chips, Proceedings of International Conference on Computer-Aided Design (2003) 8-12.
8. Plosila, J., Seceleanu, T., Liljeberg, P.: Implementation of a Self-Timed Segmented Bus, IEEE Design and Test of Computers **20** (2003) 44-45.