Design Automation for Embedded Systems, 8, 189–210, 2003.

Performance Analysis of Arbitration Policies for SoC Communication Architectures

FRANCESCO POLETTI fpoletti@deis.unibo.it
University of Bologna, DEIS, Viale Risorgimento, 2 Bologna, Italy

DAVIDE BERTOZZI dbertozzi@deis.unibo.it
University of Bologna, DEIS, Viale Risorgimento, 2 Bologna, Italy

LUCA BENINI lbenini@deis.unibo.it
University of Bologna, DEIS, Viale Risorgimento, 2 Bologna, Italy

ALESSANDRO BOGLIOLO alessandro.bogliolo@uniurb.it
University of Urbino, STI, Piazza Repubblica, 13 Urbino, Italy

Abstract. As technology scales toward deep submicron, the integration of a large number of IP blocks on the same silicon die is becoming technically feasible, thus enabling large-scale parallel computations, such as those required for multimedia workloads. The communication architecture is becoming the bottleneck for these multiprocessor Systems-on-Chip (SoC), and efficient contention resolution schemes for managing simultaneous access requests to the shared communication resources are required to prevent system performance degradation. The contribution of this work is to analyze the impact on multiprocessor SoC performance of different bus arbitration policies under different communication patterns, showing the distinctive features of each policy and the strong correlation of their effectiveness with the communication requirements of the applications. Beyond traditional arbitration schemes such as round robin and TDMA, another policy is considered that periodically allocates a temporal slot for contention-free bus utilization to a processor which needs fixed predictable bandwidth for the correct execution of its time-critical task. The results are derived on a complete and scalable multiprocessor SoC simulation platform based on SystemC, whose software support includes a complete embedded multiprocessor OS (RTEMS). The communication architecture is AMBA compliant, and we exploit the flexibility of this multi-master commercial standard, which does not specify the arbitration algorithm, to implement the explored contention resolution schemes.

Keywords: System-on-Chip, bus, arbitration, bandwidth reservation, round-robin.

1. Introduction

Deep submicron technologies are making the integration of a large number of IP blocks on the same silicon die technically feasible. As a consequence, several heterogeneous cores can be combined through sophisticated communication architectures on the same integrated circuit, leading to the development of flexible hardware platforms able to accommodate highly parallel computation. The application domain of these Systems-on-Chip (SoC) includes mobile terminals (e.g., for multimedia applications), automotive, set-top-boxes, game processors, etc. [1].

The SoC design paradigm relies heavily on reuse of intellectual property cores (IP cores), enabling designers to focus on the functionality and performance of the overall system. This is possible if the IP cores are equipped with a highly optimized interface for their plug-and-play insertion into the communication architecture. To this purpose, the Virtual Socket Interface Alliance (VSIA) represents an attempt to set the characteristics of this interface industry-wide, thus facilitating the match of pre-designed software and hardware blocks from multiple sources [2], [3].

The most widely adopted interconnect architecture for the SoC IP blocks is bus-based, and consists of shared communication resources managed by dedicated arbiters that are in charge of serializing access requests. This architecture usually employs hierarchical buses, and tends to distinguish between high-performance system buses and low complexity and low-speed peripheral buses. Many commercial on-chip architectures have been developed to support the connection of multiple bus segments in arbitrary topologies, providing at the same time a moderate degree of scalability: Wishbone [6], Advanced Microcontroller Bus Architecture (AMBA) [5], and CoreConnect [4] are relevant examples.

As the complexity of SoCs increases, the communication architecture becomes the performance bottleneck of the system. The performance of multiprocessor systems depends more on the efficient communication among processors and on the balanced distribution of the computation among them, rather than on pure CPU speed. For integration levels in the order of hundreds of processors on the same SoC, the most efficient and scalable solution will be the implementation of micro-networks of interconnects [8], but below that limit bus-based communication architectures remain the reference solution for state-of-the-art multiprocessor systems because of the lower design effort and hardware cost. This forces designers to push the performance of these architectures to the limit, within the architectural degrees of freedom made available by existing commercial bus standards.

The arbitration process plays a crucial role in determining the performance of the system, as it assigns the priorities with which processors are granted the access to the shared communication resources. The increasing integration levels of a SoC translate to an increase of contention among the processing elements for the bus, and this might lead to the violation of real-time constraints and more in general to performance degradation. An efficient contention resolution scheme is therefore required to support real-time isochronous data flows associated with networking and multimedia data streams.

2. Contribution of this Work

An effective bus arbitration policy should satisfy several requirements: (1) enable fast, high-priority communication, while avoiding starvation of low priority transactions; (2) provide fine-grained control of the communication bandwidth allocated to individual system components; (3) reduce sensitivity of system performance to variations of the communication pattern induced by an application on the bus. Traditional arbitration policies used both by centralized and by distributed arbiters to address the bus contention problem in multi-master SoCs include priority-based selection, round robin, and time division multiple access (TDMA). More advanced arbitration algorithms have been also proposed [7].

The main contribution of this work is to point out the correlation between the effectiveness of an arbitration policy and the traffic pattern induced on the bus by the communication requirements of an application. In particular:

- Beyond investigating how system performance is affected by conventional arbitration policies such as round robin and TDMA, we extend our analysis to a "slot reservation" policy, wherein a temporal slot for contention-free bus utilization is periodically reserved to a specific processor which needs a guaranteed bandwidth for the correct execution of its task. During the inter-slot time, all other processing elements compete for accessing the bus in a round-robin fashion.

- We show that the optimal contention resolution scheme is not unique, and we analyze three case studies which are representative of different communication patterns where the considered arbitration policies compare differently. For each scenario, we use a performance metric indicating how efficiently an application (or a set of applications) running on top of a multiprocessor platform is executed, and we show the impact of the arbitration policies on this metric.

- We provide experimental results obtained by means of extensive simulations on a complete and scalable multiprocessor SoC simulation platform, hereafter denoted as MPARM [9]. The simulation backbone is SystemC, that allows the description of both hardware and software in a common language (C++). The hardware platform consists of a scalable number of cycle-accurate ARM instruction set simulators, embedded into SystemC wrappers implementing the interface between the cores and an AMBA-based communication architecture.

- We developed a complete software development and run-time support infrastructure for MPARM, including a complete port of an embedded multiprocessor operating system (RTEMS).

The paper is structured as follows: Section 3 provides an overview of previous work, Section 4 describes bus arbitration policies, Section 5 summarizes the key features of the AMBA bus specification, Section 6 describes the multiprocessor simulation platform used for our experiments, Section 7 presents experimental results and Section 8 concludes the paper.

3. Previous Work

Communication architectures defined by commercial standards always provide a certain degree of flexibility in arbitration policies. This allows end users or SoC manufacturers to tailor the hardware architecture to the particular application domain of interest.

The CoreConnect interconnect architecture from IBM makes use of a fixed priority arbiter, but the priority fairness is programmable [4]. Therefore designers must analyze the application and determine the priorities among devices. Up to eight masters on the same system bus can be managed, and address pipelining is supported.

AMBA specification from ARM [5] shares many characteristics with CoreConnect, e.g., the pipelined operation of the bus and bus segmentation and bridging to support communication diversity. However, the arbiter implementation is more flexible: although the arbitration protocol is fixed, any arbitration policy can be implemented depending on the application requirements. Wishbone from Silicore Corp. [6] is another bus specification wherein arbitration is defined by the end-user. Silicon Backplane from Sonic Inc. [10] is a solution for communication among IP cores that guarantees fixed bandwidths and latencies by means of TDMA-based arbitration.

A significant effort has been recently devoted to enabling the effective design of multi-core SoC starting from pre-designed and pre-verified IP blocks. An overview of design methodologies and tools proposed to address the problem is provided by [15], [16]. The communication architecture is a key point of the design stage, and the final goal of many works is to extract the communication requirements from the application and to map them to the underlying communication architectures by looking for the solution that enables to meet application-perceived performance constraints [12]. To this purpose, a design space exploration technique for SoC communication architectures is proposed in [11].

A comparison between SoC bus architectures is made in [13], where selection guidelines for such architectures are also provided. Liang et al. [14] proposes an adaptive SoC communication infrastructure that can be easily reconfigured as application-level communication pattern changes: it provides support for compile-time predicted inter-node communication.

Finally, a new high performance architecture for SoC design is presented in [7], called "Lotterybus." It consists of a randomized arbitration algorithm implemented in a centralized "lottery manager," which collects requests for bus ownership from multiple masters. The manager probabilistically chooses one of the contending masters which is granted the bus for one or more cycles. The performance of this policy is compared to that of conventional communication architectures for different traffic classes.

The main shortcoming of previous explorative work in bus arbitration is that system functionality is taken into account in a highly abstract fashion. Most previous works employ stochastic traffic generators as bus masters, while others use high-level functional models for software tasks, that do not account for nonideality of software execution on a target processor (e.g., instruction misses, operating system overhead). Our single-chip multiprocessor simulation platform is cycle accurate both at the bus transaction level and at the software execution level, and fully functional applications and OS are executed without any abstraction or simplification (no instructions are emulated on the simulation host). The approximation margins in our explorative analysis are therefore reduced to a minimum.

4. Contention Resolution Schemes

In this section we briefly present and discuss the key features of the arbitration policies analyzed in the remainder of the paper.

4.1. Round-Robin

A round-robin arbitration policy is a token passing scheme wherein fairness among masters is guaranteed, and no starvation can take place (in contrast with a static fixed priority scheme) [20]. In each cycle, one of the masters (in round-robin order) has the highest priority for access to a shared resource. If the token-holding master does not need the bus in this cycle, the master with the next highest priority who sends a request can be granted the resource. The advantages of round-robin are twofold:

- Unused time slots are immediately re-allocated to masters which are ready to issue a request, regardless to their access order. This reduces bus underutilization in comparison with a statically fixed slot allocation, that might grant the bus to a master which is not going to carry out any communication.
- The worst-case waiting time for the bus access request of a master is reliably predictable (being proportional to the number of instantaneous requests minus one), even though the actual waiting time is not. The uncertainty on the actual bandwidth that can be granted to a master is the major drawback of this scheme.

4.2. TDMA

A time division multiple access scheme is based on the fixed allocation of a slot to each master, so that each of them is guaranteed fixed and predictable bandwidth. Unfortunately, high priority communications in a TDMA-based architecture may incur significant latencies, because the performance provided by this scheme strongly depends on the time-alignment of communication requests and slot allocation, and therefore on the probability of dynamic variations of the request patterns.

4.3. Slot Reservation

This arbitration policy can be seen as a limit case of TDMA, in that only one master is periodically allocated a slot for the contention-free access to the bus. For the inter-slot time, we decided to manage the contention among the remaining masters in a round-robin fashion. Although this is not a conventional scheme for SoC communication architectures, we propose this policy to combine the advantages of the above mentioned schemes: one master is given priority in the competition for bus access (in terms of guaranteed fixed bandwidth), while all other masters can contend for the shared communication resource avoiding the risk of starvation.

The highest priority master (the one to which the slot is allocated) can therefore complete its transfers without incurring contention-related delays, which are likely to considerably increase as the number of competing masters increases. This translates to a performance degradation for the lower priority masters (those managed in a round-robin fashion), which are excluded from bus access during the slot allocation. The effectiveness of this scheme is tightly related to the ratio between the performance improvement of the highest priority master and the performance degradation suffered by the lowest priority ones, which must be as high as possible.

5. AMBA Bus

The AMBA defines an on-chip communication standard for designing high-performance multi-master SoCs. Three distinct buses are defined within the AMBA specification, as illustrated in Figure 1: (1) the Advanced High-Performance Bus (AHB), which is the highly optimized system backbone bus; (2) the Advanced System Bus (ASB), an alternative system bus used whenever less aggressive performance is required; (3) the Advanced Peripheral Bus (APB), which is a low complexity and low power bus for communication with general purpose peripherals. The system and the peripheral bus are connected to each other by means of a bridge which reduces global wires load capacitances and hence switching power consumption.

5.1. Arbitration Protocol

In this work we will focus on AHB, which exhibits high performance features such as support for multiple masters and multiple slaves, for pipelined bus operation and burst transfers, as well as for split transactions.

As already mentioned, AMBA defines the arbitration protocol but it does not define the contention resolution policy. A bus master requests to access the bus by asserting a HBUSREQ signal, as illustrated in Figure 2. The arbiter observes the different simultaneous requests and grants the bus to the highest priority master by asserting its HGRANT signal. The master effectively gains control of the address bus

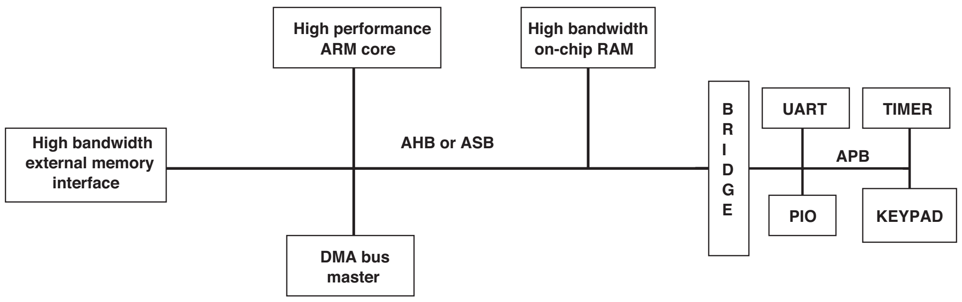
High performance ARM core
High bandwidth on-chip RAM
High bandwidth external memory interface
AHB or ASB
DMA bus master
BRIDGE
UART
TIMER
APB
PIO
KEYPAD

Figure 1. A typical AMBA system.

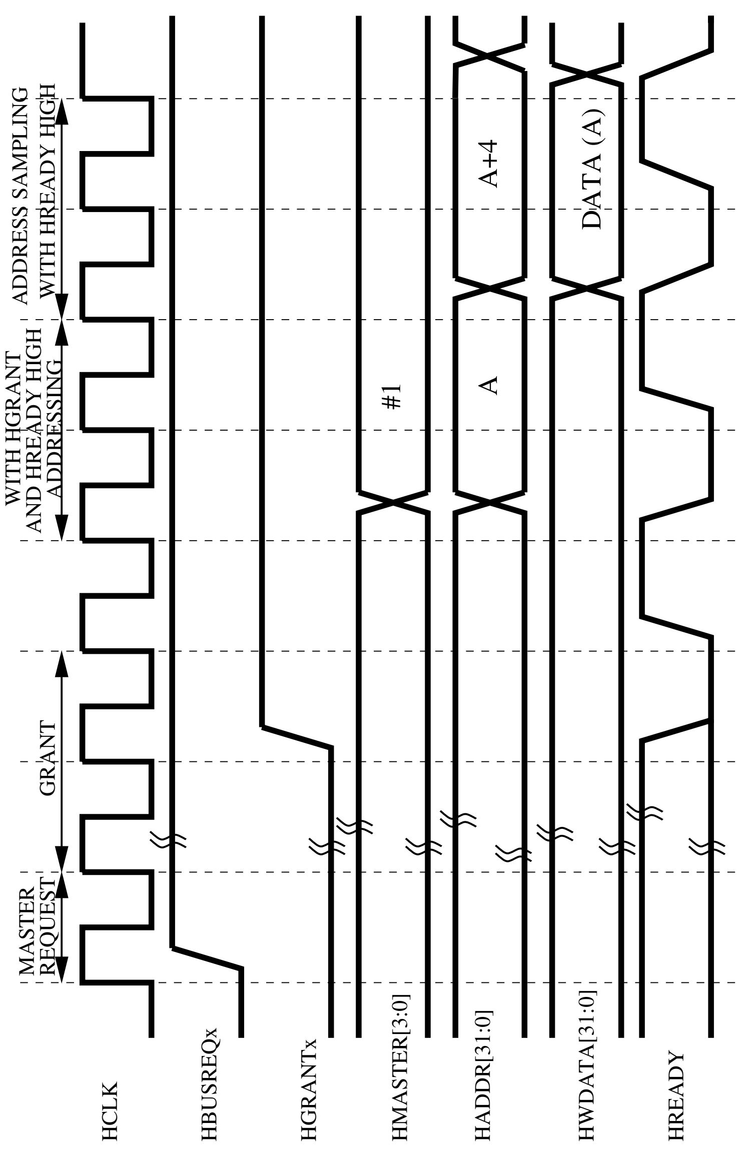

Figure 2. Bus handover within the AMBA specification.

when both the HGRANT and the HREADY signals (indicating that the last transfer has completed) are sampled high. The ownership of the data bus is delayed with respect to the ownership of the address bus, thus allowing pipelined operation.

5.2. Implementation of Arbitration Policies

The implementation of a round-robin based contention resolution scheme is straightforward, because the bus arbitration process can take place at the penultimate cycle of an outstanding transfer. At that time, the priority is given by examining the pending access requests.

Embedding TDMA or slot-reservation schemes into a bus arbiter is instead a nontrivial task, because in these cases the arbitration process must take place at predefined instants of time, and this might result in a bus preemption for the master that owns it. For single outstanding transfers this is not too much of an issue, because they are so short that they can be considered as atomic and the arbiter has a sufficiently fine-grained control of the bus. The main difficulty lies in the need to support bus preemptions during burst transfers. The master that looses bus ownership in the middle of a burst must be able to properly complete the remaining transfers once it regains access to the bus.

We found that TDMA and slot-reservation based arbitration policies can be implemented in an AMBA arbiter without loosing compliance with the standard by exploiting an option of the AMBA specification called *Early Burst Termination* (EBT). This option was originally meant to support bus preemption so to be able to set an upper bound on the bus ownership time for each master and to prevent other masters from incurring unacceptable access waiting times. Providing support for EBT is a responsibility of the designers of AMBA compliant masters and slaves. The AMBA specification only states that the master that looses bus ownership must re-arbitrate for it in order to complete the burst, and this has to be done by means of any legal burst encoding (e.g., incremental burst of unspecified length).

6. Multiprocessor Simulation Platform

Our experiments have been performed on a complete multiprocessor SoC simulation platform, called MPARM [9]. SystemC [18] is used as backbone simulation engine, and this provides the advantage of describing both hardware and software in a common language (C++).

6.1. Hardware Support

The efficient integration into the simulation platform of a scalable number of instruction set simulators (ISSs) can be easily carried out by encapsulating them into SystemC wrappers. The ISSs used in our architecture are cycle accurate simulators of cached ARM cores, written in C++ and called SWARM [17]. Along with the CPU, a set of peripherals is emulated (timers, interrupt controller, UART) to provide support for an operating system running on the platform.

The cycle accuracy of SWARM greatly simplifies the synchronization with the SystemC environment, as the control is returned to the SystemC process scheduler at every clock cycle. The simulation platform also includes hardware support for parallel programming. In fact, in this multiprocessor system process synchronization must be ensured, to allow mutually exclusive access of the processes to shared memory resources. To this purpose, we have equipped the simulator with a bank of memory-mapped registers, connected to the AMBA bus as a slave, working as hardware semaphores.

The platform simulates two memory hierarchies: instruction and data caches and main memories. While cache memories are simulated within SWARM, each processing element has its own external private memory which is instantiated as an AMBA slave. Therefore, read or write accesses to the private memory take place through the AMBA system bus and incur contention-related latency. An overview of the system is reported in Figure 3. Up to 32 cores can be instantiated in the system.

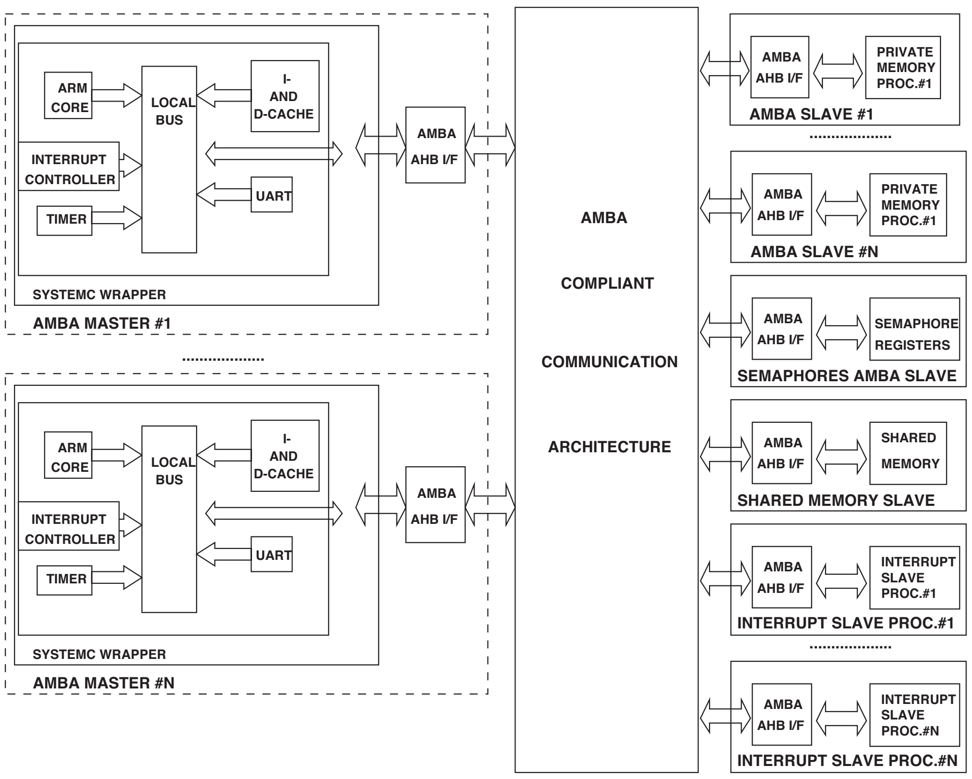

Figure 3. Multiprocessor SoC architecture.

Besides private memories, the system includes one shared memory which is used, at the hardware level, only to implement application-level inter-processor communication. For the same purpose, interrupt slaves are instantiated, as outlined in the next subsection.

6.2. *Software Support*

The software support for this simulation platform includes a complete port of an embedded OS, RTEMS [19]. RTEMS is a real-time OS that features POSIX APIs, synchronization and inter-task communication primitives for a multiprocessor scenario.

Inter-processor communication at the application layer takes place through message passing, according to the procedure briefly illustrated in Figure 4. By means of high-level send and receive communication primitives called by a task, the message to be exchanged is read by RTEMS kernel and transferred, in packets, at the MPCI layer. Packets are then written into the shared memory. At this point, we have configured the kernel to use an interrupt based notification technique to signal the destination processor that there is an outstanding packet for it, and we have provided hardware support for this methodology. In particular, we force the source processor

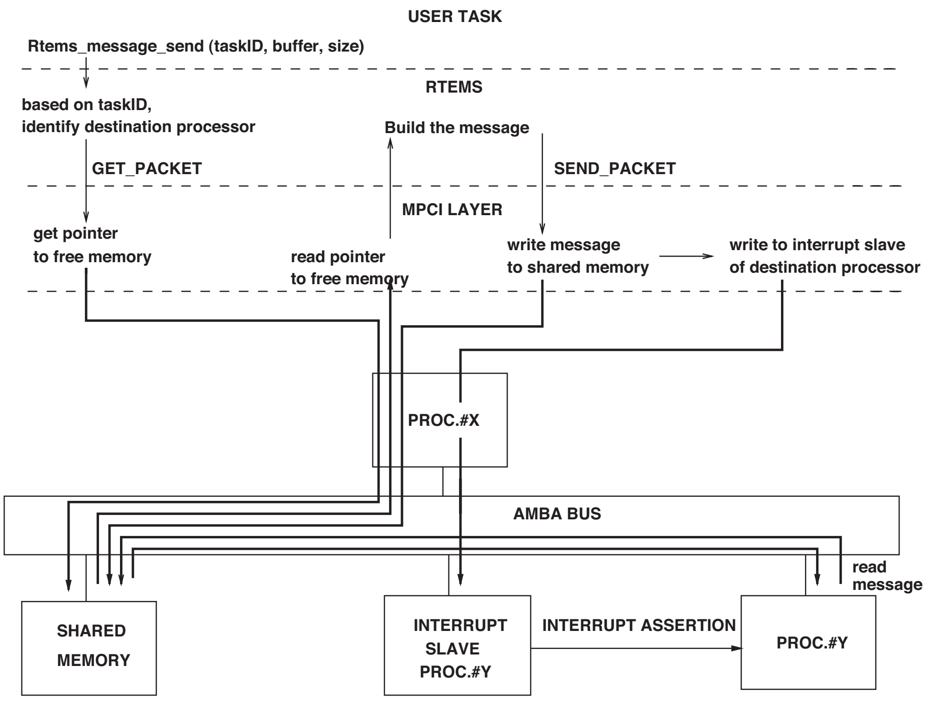

Figure 4. Inter-processor communication procedure.

to carry out a write transfer to a memory-mapped slave, which asserts a dedicated interrupt of the destination processor. The assertion of this external interrupt triggers a service routine that reads the message from the shared memory, thus completing communication.

It is important to note that inter-processor communication can contribute to a large fraction of bus transactions, increasing the contention for accessing the shared bus. In multiprocessor scenarios wherein synchronization among processors is required for task execution (e.g., exchange of data among processors in distributed signal processing applications), communication-related traffic could be dominant and the way it is handled by bus arbitration policies could play a key role in determining the overall system performance.

7. Performance Analysis of Arbitration Algorithms

Our objective was to stress the distinctive features of the considered arbitration algorithms so to come up with selection guidelines under different system workloads. To this purpose, we identified three scenarios at the application level, corresponding to three different communication patterns: mutually dependent tasks, independent tasks, and pipelined tasks.

7.1. Mutually Dependent Tasks

Let us assume a workload wherein one task is running on each processor and that the correct execution of each task involves synchronization with the other ones. In particular, let us assume that all tasks have to synchronize with each other at predefined points of the multiprocessor benchmark. In this case, system performance optimization translates to avoiding that some tasks reach the synchronization point much earlier or much later than the others, because this would generate idle waiting time for the unsynchronized task.

An example thereof is represented by the bootstrap stage of RTEMS on the multiprocessor system. RTEMS selects one processor to act as a master and all other ones are considered as slaves, and they play a slightly different role in the booting operation. Each processor (master and slaves) at first independently initializes its private memory and hardware devices, then synchronization has to take place at the shared memory. In fact, the master processor is in charge of initializing the shared memory and of allocating the structures for inter-processor communication. Then it starts polling the status variables of the slave processors, until they are all set to “ACTIVE,” indicating that the slave processors have defined their own data structures in the shared memory. When this synchronization condition occurs, the master processor sets those variables to “FINISHED,” notifying the slaves that the initialization of the shared memory is over and that each processor can independently complete its bootstrap stage and load its tasks.

We ran the RTEMS bootstrap routine several times, with different arbitration policies implemented in the AMBA arbiter. The performance metric in this scenario is the bootstrap execution time. Each contention resolution scheme is assessed based on its ability to minimize this time and on the associated cost.

Results are shown in Figure 5, where the execution time for a bootstrap on five processors is plotted as a function of the slot duration. With slot reservation, the contention-free slot is assigned to the master processor, denoted as *Proc#1*. Its slot duration is reported on the *x*-axis, while the inter-slot period is kept constant at 1000 cycles. With TDMA, the *x*-axis refers to the slot allocated to each processor. Finally, with round robin we have no parameters to set, and this corresponds to the constant value observed on the plot.

Round robin exhibits a good performance, depending on the contention level for accessing the bus. Due to asymmetric workload associated to master and slave processors, with a round-robin policy we observe that the slaves have to wait for the master (which has more operations to carry out, and in general is more computation-intensive) at the synchronization point, and this slows down the overall RTEMS bootstrap on the multiprocessor system. This suggests to allocate the slot for contention-free access to the bus to the master processor. In fact, for a slot duration of about half the inter-slot period, the slot reservation arbitration policy outperforms round robin, because the increased bandwidth given to the master processor makes up for the asymmetric workload. This minimizes the waiting time of the processors at the synchronization point.

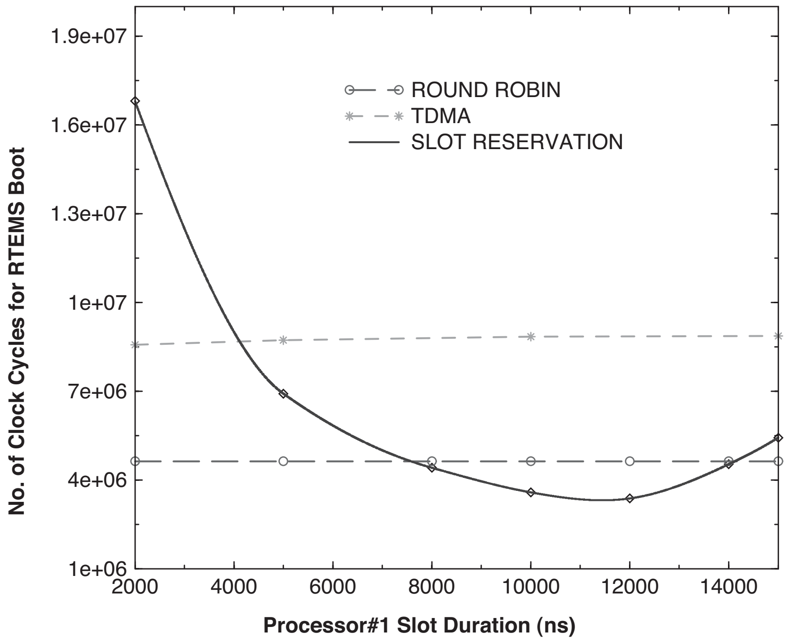

Figure 5. Execution time of the bootstrap routine of RTEMS on the multiprocessor platform.

For higher values of the slot duration, too much bandwidth is assigned to the master respect to its needs: the effect is that this time the master reaches the synchronization point much in advance respect to the slaves, and has therefore to wait for them. On the other hand, the small amount of time reserved to slave processors for bus utilization significantly degrades their performance, and the overall execution time increases. An excessively small slot duration causes the same effect, because the master operation slows down respect to a round robin arbitration, and the negative impact on the execution time is even more remarkable because of its asymmetric workload.

Finally, TDMA exhibits the worst performance, in that no balancing effect takes place but only a redistribution of the bus request patterns.

Next, we analyzed the cost incurred by slot reservation for the offered performance. This cost is assessed in terms of average waiting time, defined as the period between the time a processor asserts its bus request signal and the time its grant signal is asserted by the arbiter, indicating that the ownership of the bus has been actually granted. Results are reported in Figure 6. The average waiting time of the master processor is compared to that of the other processors when slot reservation is activated. With the other policies, as the average waiting times of all processors are more balanced, only the overall average value is reported.

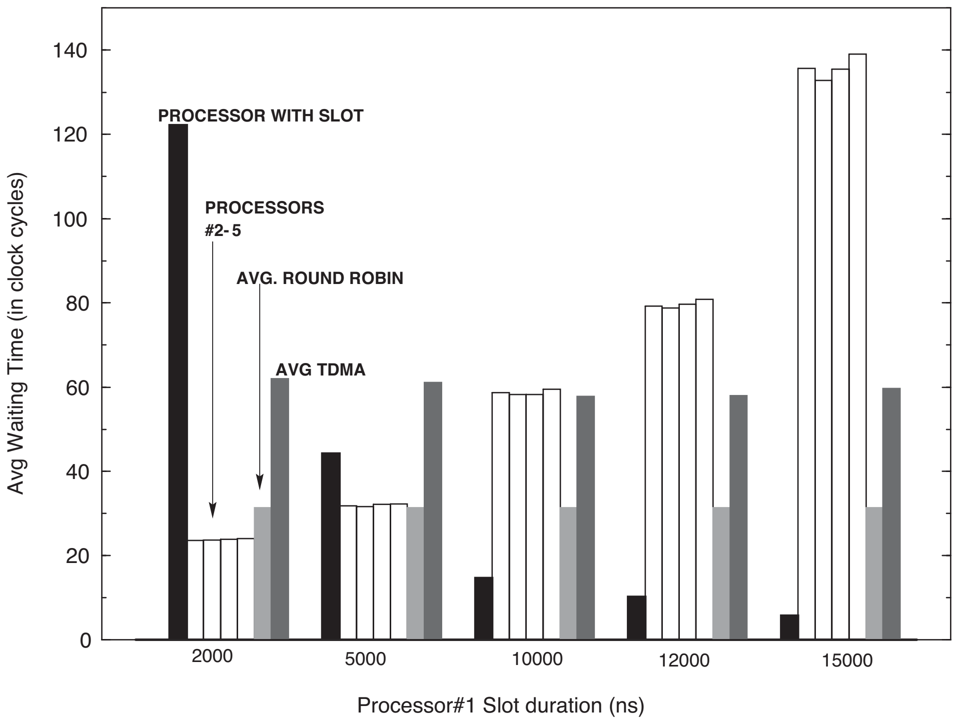

Figure 6. Average waiting time of the processors for accessing the bus.

It is interesting to observe that for an optimal slot duration of about 12,000 ns (600 clock cycles, the clock period being 20 ns) derived from the previous plot, the average waiting time of the high priority master is more than halved and that of the other processors is more than doubled respect to the round-robin case. This effect does not play any role in this context, as the performance metric of interest is the minimization of the total execution time. Therefore, an increase of the latency for accessing the bus can be sometimes tolerated, provided it is not directly related to the system performance perceived at the application level.

7.2. Independent Tasks

The second scenario we investigated makes use of a benchmark consisting of independent tasks, each running on a specific processor. This system workload does not have any synchronization point, nor it involves inter-processor communication.

The above scenario has been implemented on our simulation platform by executing the same matrix multiplications on each processing element. Matrices are initially stored in each processor's private memory, and the traffic generated on the bus is associated with read operations of matrix elements and to write transactions storing the results back in the memory. Tasks execution and consequent measurements are triggered once RTEMS has booted on all of the processors.

The performance metric we select for this class of benchmarks is the average task execution time, given the independent nature of the tasks themselves. Our experiments have been carried out ranging the number of processors from 2 to 10, analyzing the scaling properties of the performance metric.

Results relative to the tasks execution times are reported in Figure 7, for the cases of 4 and 8 active processors. When four tasks are running, we observe that round robin outperforms the other schemes. In fact, if we randomly select one processor (e.g., processor no. 1) and periodically grant it a slot for contention-free access to the bus, the improvement of its execution time translates to a relevant degradation of the performance for the other processors, and the average task execution time of the system increases. Though it is interesting to observe that a slot allocation of 9000 ns manages to balance the execution times of all processors, so that on average all tasks complete within the same time, similarly to what happens with round robin or TDMA, and this is the most efficient approach for this scenario. The relevant difference between the three arbitration algorithms is in the average execution time that can be obtained by each of them under the hypothesis of balanced task execution times. The balancing effect for slot reservation (achieved by properly tuning the slot duration) occurs at an average execution time which lies between that provided by round robin (the optimal one) and that provided by TDMA (worst case).

The same effect can be observed with 8 processors, even though the average values increase and the gap between round robin and slot reservation decreases.

One might guess that the performance of TDMA is likely to increase for smaller values of TDMA slot respect to those reported in Figure 7, so to reduce bus idleness. The answer to this question is reported in Figure 8, where the average execution time

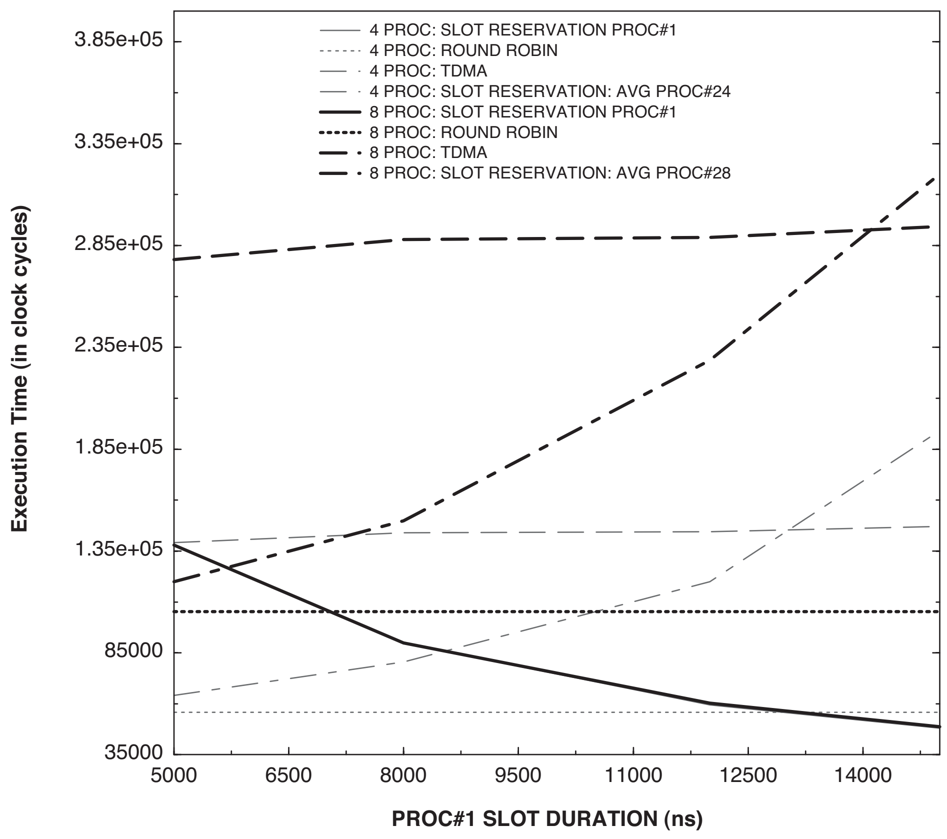

Figure 7. Execution time for a benchmark consisting of independent tasks.

of the tasks is plotted as a function of (smaller) TDMA slot. Although the performance offered by TDMA actually increases, it never performs like round robin. The shortest execution time occurs when the TDMA slot is in the order of the duration of a burst transfer. For smaller values, bus preemptions start playing a dominant role and their high frequencies (and their associated costs in terms of bus cycles) determine a performance degradation. The behavior showed in Figure 8 can be explained with the redistribution of request patterns operated by TDMA and by the misalignment of such patterns with the slot allocation.

Going back to Figure 7, another consideration is worth mentioning. Let us assume that one processor has to be allocated more bandwidth, tolerating the performance degradation of other schemes. For this processor, a slot must be allocated such that its final execution time be less than that exhibited in the round robin case. Therefore, if we check the crossing point between the "Proc#1" curve and the round robin curve, we see that as the number of processors increases, the slot duration that ensures such a performance decreases (it is almost halved from 13,000 to 7000 ns), and this is tightly related to the increased contention levels on the bus.

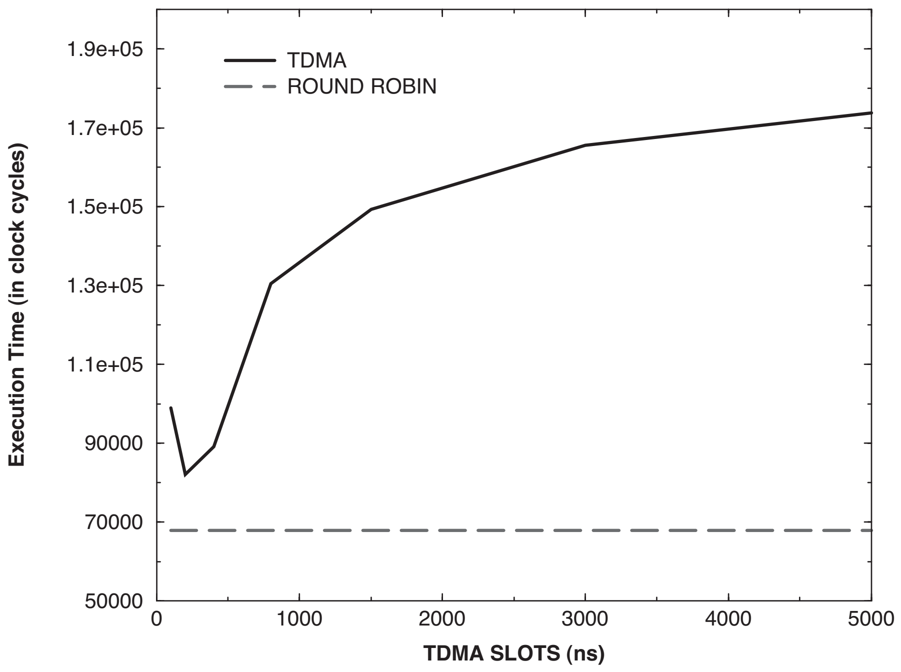

Figure 8. Comparison between performance of round robin and TDMA for small values of TDMA slot.

As regards the average waiting time for accessing the bus, an histogram is reported in Figure 9. For a slot value of 8000 ns, close to the optimal value that balances the execution times, the average waiting times of the processors are balanced as well, but higher than that achievable by means of a round robin arbitration. Note the very poor performance of TDMA for such values of the slot.

Finally, we want to show how the execution times of the processors scale as a function of the number of active processors. This result is reported in Figure 10: the values for the high priority processor (e.g., "slotx-I," where *x* is the slot duration) are compared with the average ones of the remaining processors ("slotx-avg") when slot reservation is used, and with the average round robin and TDMA values ("tdmax"). It is interesting to observe the rapidly degrading performance of TDMA, while round robin and slot reservation are more scalable under this point of view.

7.3. Pipelined Tasks

A last scenario that is worth investigating is the one wherein the impact of arbitration policies on the throughput of a distributed signal processing application can be assessed. While in the previous subsection we analyzed a system workload wherein the traffic across the bus did not depend on inter-processor communication at all, but

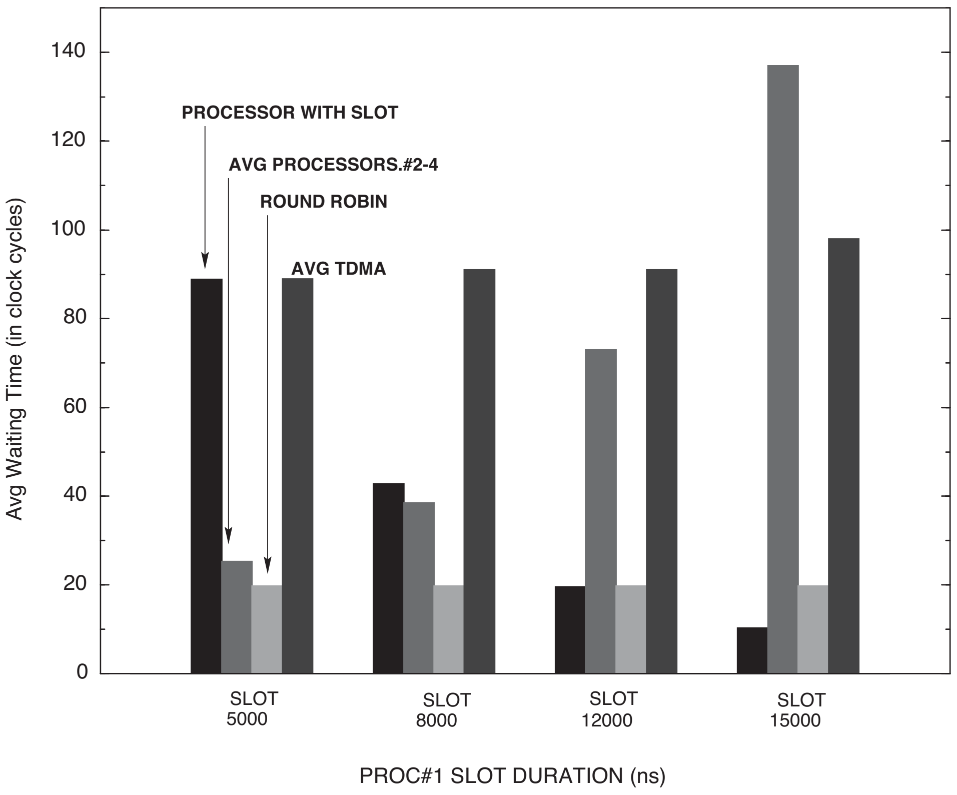

Figure 9. Bus access delays for the benchmark with independent tasks.

was only related to computation (e.g., cache line refills), now we want most of bus transactions to be related to communication among processors. We want to relate the performance of such a system to the way communication related traffic is accommodated on the bus by the different arbitration policies.

To this purpose, we set up a multiprocessor system wherein different tasks execute in a pipelined fashion, with balanced computation workloads for all of the processors (they execute matrix multiplications). On top of the first processor, a task generates matrices that are handed over to the second processor of the pipeline. At each stage, the computation is carried out and the result transmitted to the next stage. In other words, the pipeline consists of couples of producer–consumer tasks, and the communication occurs, at a high level of abstraction, by means of FIFO queues.

The performance metric for this system is the throughput, defined as the number of matrices per second produced by the last processor of the pipeline (i.e., frame rate).

Figure 11 shows the frame rate provided by the arbitration policies, changing the value of the slot duration for slot reservation and TDMA. While the performance of slot reservation is highly sensitive to the slot time, the performance of TDMA is almost independent of it. Surprisingly enough, although both the workload and the

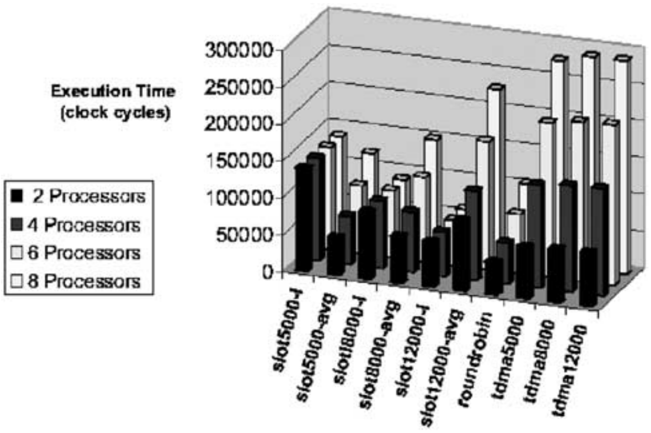

Figure 10. Scalability property of the execution times with different arbitration policies.

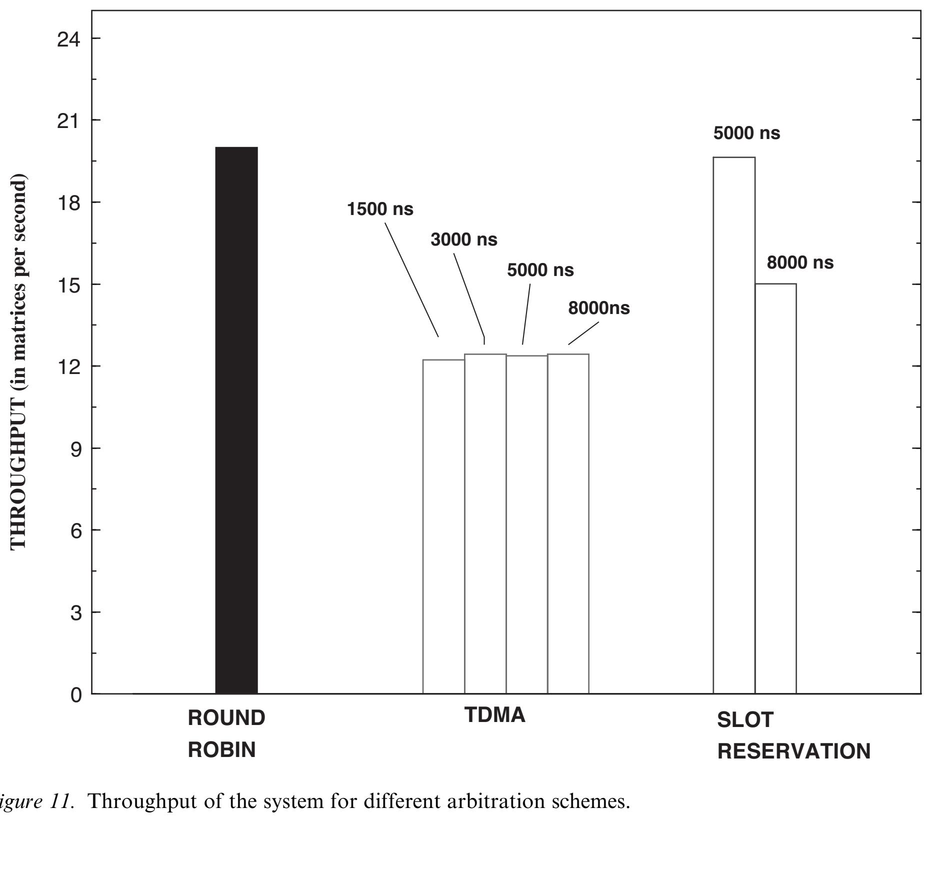

Figure 11. Throughput of the system for different arbitration schemes.

communication needs of the pipelined processors are perfectly balanced, slot reservation performs better than TDMA for a wide range of slot durations. This can be explained by looking at the performance of round robin, that is always much better than TDMA. Since our slot reservation algorithm implements a round-robin arbitration policy during inter-slot times, as long as the slot duration is much shorter than the inter slot time, the performance of slot reservation is dominated by the performance of round robin, while it becomes much worse when larger slots are used. Therefore, in Figure 11 only two experiments for slot reservation have been carried out, because they are sufficient to clarify the dependence of execution time as a function of the slot duration.

Since the frame rate provided by slot reservation is always smaller than that of round robin, we can say that slot reservation is counter productive in this case. In fact, there is no reason for guaranteeing a constant bandwidth to a single stage of a pipeline if the same bandwidth cannot be guaranteed to all stages.

On the other hand, TDMA guarantees a constant bandwidth to all processors in the pipe, but its overall performance is lower than that of round robin. This fact can be explained only by looking at the hardware implementation of high-level interprocessor communication primitives.

In our system, the producer–consumer paradigm is implemented by means of the RTEMS message manager, which makes use of a communication protocol among tasks based on message queues. At the core of this protocol there is the interprocessor communication mechanism seen in Figure 4. The procedure is initiated by the producer, which creates a global queue in its private memory, and writes messages to be sent in it.

When the consumer is ready to receive a message, a notification is given to the producer by writing a request message into the shared memory and by generating an interrupt for the producer itself. The interrupt service routine of the producer reads the message from shared memory and assembles data to be sent in a message which is written back to shared memory. Finally, a write transaction to the consumer interrupt slave asserts an interrupt which allows the consumer to pick up its message from shared memory.

In this context, TDMA poor performance can be explained in terms of its inability to support the communication handshake between the producer and the consumer, which is necessary for the hardware implementation of the high level inter-processor message passing. This handshake involves a ping–pong interaction between the two tasks, and is inefficiently accommodated in a TDMA based architecture, wherein only one processor is active during each slot. This results in a higher latency for the interaction respect to the round robin case, and this explains the poor performance of TDMA observed in the experiments.

This low level implementation of message passing primitives made available by RTEMS to the applications involves a large overhead in terms of bus transactions. This overhead may result in a relevant system performance penalty, and derives from a mismatch between the software architecture and the underlying hardware platform. In other words, these two layers should be aware of each other to maximize system performance.

As an example, it is worth mentioning that our multiprocessor simulation platform does not support global cache coherency, therefore the shared memory is declared "non-cacheable." Furthermore, the ARM ISS embedded into our platform does not support burst transfers except for cache line refills (accessing only private memories). As a consequence, reading or writing data to shared memory is a highly inefficient operation, because it only takes place by means of single transfers instead of burst transfers.

Finally, we observe that the poor performance exhibited by TDMA is also related to the fact that it is inefficiently accommodated in an AMBA based communication architecture. In fact, the ultimate objective of the AMBA bus protocol is contention avoidance, and the signals used by masters and slaves have to be seen under this perspective (e.g., HBUSREQ, etc.). On the contrary, TDMA would require a simpler communication protocol, as the whole contention management procedure is arbiter driven. As a consequence, TDMA might outperform other arbitration algorithms in proprietary communication architectures.

Despite the lower performance, TDMA-based arbitration is also attractive in many real-time applications where predictability is a critical requirement. In fact, TDMA reserves a slot to each processor regardless of the current workload, thus making constant in time the bandwidth perceived by each processor, independently of the traffic generated by the other masters. Consider, for instance, a system composed of 10 processor cores. If five of the cores are used to implement the pipelined streaming application described in this subsection the frame rate achieved will be constant and predictable, independently of the traffic generated by the processors that do not take part in the pipeline (hereafter called external processors). Using round robin, the frame rate would be much better than that provided by TDMA when the traffic generated by the external processors is negligible, but it would be strongly dependent on the overall workload, possibly becoming worse than that of TDMA when external processors perform memory/communication intensive tasks. Nondeterminism is not acceptable in many real-time situations.

8. Conclusions

In this work, we analyze the impact on multiprocessor SoC performance of different bus arbitration policies under different communication patterns, showing the distinctive features of each policy and the strong correlation of their effectiveness with the communication requirements of an application.

Beyond two traditional bus arbitration policies (round robin and TDMA) we consider another technique that periodically allocates fixed predictable bandwidth to time-critical processors ("slot reservation"). Three workloads are analyzed on our multiprocessor simulation platform (mutually dependent tasks, independent tasks and pipelined tasks), and some important guidelines for designers of SoC communication architectures have been derived:

1. The optimal bus arbitration policy is not unique, but strongly depends on the traffic conditions (computation-dependent, communication-dependent, etc.).

2. The software support for inter-processor communication plays a crucial role in determining system performance, as it has to be matched with the underlying hardware platform. High level communication primitives, although facilitating the programming step, could be inefficiently implemented on the available platform, degrading system performance.

3. There exists a trade-off between contention-avoidance bus arbitration policies (such as TDMA) and contention-resolution bus protocols (such as AMBA bus). Even though commercial standards provide degrees of freedom for performance optimization, the performance achievable by contention-avoidance policies implemented within contention–resolution protocols cannot be fully exploited, because of their different characteristics.

References

1. Oka and Suzuoki. Designing and Programming the Emotion Engine. *IEEE Micro*, vol. 19-6, no. 8, pp. 20–28, 1999.
2. Wingard, D. MicroNetwork-Based Integration for SOCs. In *DAC*, Las Vegas, June 2001.
3. Virtual Socket Interface Alliance, http://www.vsi.org.
4. CoreConnect Bus Architecture, http://www.chips.ibm.com/products/coreconnect.
5. AMBA Specification Overview, ARM, http://www.arm.com./Pro+Peripherals/AMBA.
6. Peterson, W. Design Philosophy of the Wishbone SoC Architecture. *Silicore Corporation, 1999*, http://www.silicore.net/wishbone.htm.
7. Lahiri, K., A. Raghunathan, and G. Lakshminarayana. LOTTERYBUS: A New High-Performance Communication Architecture for Systems-on-Chip Design. In *Proceedings of DAC 2001*, pp. 15–20, Las Vegas, USA, June 2001.
8. Benini, L. and G. De Micheli. Networks on Chip: A new SoC Paradigm. Computer, vol. 35, no. 1, pp. 70–78, January 2002.
9. Loghi, M., F. Angiolini, D. Bertozzi, and L. Benini. Analyzing On-Chip Communication in a MP-SoC Environment. Submitted to *Journal of VLSI Signal Processing*, January 2003.
10. Wingard, D., and A. Kurosawa. Integration Architecture for System-on-a-Chip Design. In *Proceedings of IEEE 1998 Custom Integrated Circuits Conference*, pp. 85–88, May 1998.
11. Lahiri, K., A. Raghunathan, and S. Dey. Efficient Exploration of the SoC Communication Architecture Design Space. In *IEEE-ACM International Conference on Computer Aided Design 2000*, pp. 424–430, San Jose, USA, 2000.
12. Kreuz, M. E., L. Carro, A. Zeferino, and A. A. Susin. Communication Architectures for Systems-on-Chip. In *14th Symposium on Integrated Circuits and Systems Design*, pp. 14–19, Brazil, 2001.
13. Ryu, K. K., E. Shin, and V. J. Mooney. A Comparison of Five Different Multiprocessor SoC Bus Architectures. In *Euromicro Symposium on Digital Systems Design*, pp. 202–209, Poland, 2001.
14. Liang, J., S. Swaminathan, and R. Tessier. aSOC: A Scalable, Single-Chip Communication Architecture. In *International Conference on Parallel Architectures and Compilation Techniques*, pp. 37–46, Philadelphia, USA, 2000.
15. Cesario, W., A. Baghdadi, L. Gauthier, and D. Lyonnard. Component-Based Design Approach for Multicore SoCs. In *Proceedings of DAC 2002*, pp. 789–794, New Orleans, USA, 2002.
16. Bergamaschi, R. A., and W. R. Lee. Designing Systems-on-Chip Using Cores. *Proceedings of DAC 2000*, pp. 420–425, Los Angeles, USA, 2000.

17. Dales, M. SWARM: Software ARM, http://www.dcs.gla.ac.uk/michael/phd/swarm.html.
18. SystemC Community, http://www.systemc.org.
19. Colin, A., and I. Puaut. Worst-case Execution Time Analysis of the RTEMS Real-time Operating System. In *Euromicro Conference on Real-Time Systems*, pp. 191–198, Netherlands, 2001.
20. Shin, E. S., V. J. Mooney III, and G. F. Riley. Round-Robin Arbiter Design and Generation. In *International Symposium on System Synthesis*, pp. 243–248, Kyoto, Japan, 2002.