Capturing the interaction of the communication, memory and I/O subsystems in memory-centric industrial MPSoC platforms

Simone Medardoni[†], Martino Ruggiero[‡], Davide Bertozzi[†], Luca Benini[‡], Giovanni Strano[⋆], Carlo Pistritto[⋆],
[†] ENDIF, University of Ferrara, 44100 Ferrara, Italy.
[⋆] STMicroelectronics, On-Chip Communication System, Catania, Italy.
[‡] DEIS, University of Bologna, 40136 Bologna, Italy.

ABSTRACT

Industrial MPSoC platforms exhibit increasing communication needs while not yet reverting to revolutionary solutions such as networks-on-chip. On one hand, the limited scalability of shared busses is being overcome by means of multi-layer communication architectures, which are stressing the role of bridges as key contributors to system performance. On the other hand, technology limitations, data footprint and cost constraints lead to platform instantiations with only few on-chip memory devices and with a global performance bottleneck: the memory controller for access to the off-chip SDRAM memory. The complex interaction among system components and the dependency of macroscopic performance metrics on fine-grain architectural features stress the importance of highly accurate modelling and analysis tools. This paper takes its steps from an extensive modelling effort of a complete industrial MPSoC platform for consumer electronics, including the off-chip memory sub-system. Based on this, relevant design issues concerning the communication, memory and I/O architecture and their interaction are addressed, resulting in guidelines for designers of industry-relevant MPSoCs.

1. INTRODUCTION

Networked digital products such as set-top-boxes, high-density DVD players or digital video recorders are increasingly relying on Multi-processor Systems-on-Chip (MPSoCs). These platforms feature increasing levels of system integration and high-speed interfaces, thus posing stringent requirements on the performance of the communication architecture.

Traditionally, shared communication resources have been deployed for SoC design, however this solution incurs severe scalability limitations. Therefore, bandwidth across the system has been increased by means of multi-layer communication infrastructures, where clusters of masters and slaves are connected via bridges. This way, a wide variety of bus structures and multi-master systems can be constructed, thus ensuring flexibility to system designers.

At the same time, the complexity of interconnect design increases a lot. In fact, the specific layers building up the overall system interconnect exhibit advanced features such as pipelining, support for split transactions and for multiple outstanding transactions. Therefore, building bridges for such systems is far from trivial. Besides protocol matching, bridges are in charge of additional tasks in heterogeneous MPSoC platforms, such as frequency adaptation and datawidth conversion.

Even deploying highly optimized bridges, performance of the communication architecture of an MPSoC is not guaranteed, but is tightly related to the interaction with the memory and I/O subsystems. The memory architecture determines the prevailing traffic pattern which has to be accommodated by the interconnect fabric. For instance, in presence of many on-chip memory cores, system performance depends on the bus ability to enable concurrent memory accesses without blocking behaviour. However, many state-of-the-art MPSoC platforms for consumer electronics are memory-centric, in that most of the memory is off-chip and the on-chip memory controller becomes the performance bottleneck of the platform. In this case, the interaction between specific features of the communication protocols and the many-to-one traffic pattern has not been analysed yet, and is certainly a function of architectural parameters such as memory access latency or amount of buffering implemented in the system. Moreover, the optimizations performed by advanced memory interfaces (such as those implemented in SDRAM memory controllers) might make this interaction unpredictable.

Given the fine-grain nature of the mechanisms that determine macroscopic MPSoC performance, high-level modelling and simulation tools are likely to fail to capture the needed level of detail which makes global interconnect performance analysis trustworthy. The work in this paper is centered on cycle-accurate analysis of the communication, memory and I/O architectures of a state-of-the-art MPSoC platform for consumer electronics. We built a multi-abstraction and accurate virtual platform allowing an in-depth investigation of the behaviour of system components, captured in isolation and when inter-operating with each other in a complete MPSoC platform of industrial relevance.

We tested several architecture variants concerning both the communication infrastructure (use of different communication protocols and topologies) and the memory and I/O architectures (on-chip shared memory versus off-chip SDRAM memory, memory controllers with increasing complexity). As a consequence, we were able to shed light on interaction effects between the three main MPSoC subsystems and come up with design guidelines.

The paper is structured as follows. Section 2 illustrates previous work. Our intensive modelling effort is described in Section 3, while experimental results are reported in Sections 4 and 5. Discussion of results follows in Section 6.

2. RELATED WORK

Several embedded-system design houses and semiconductor companies employ proprietary on-chip bus architectures [18, 19, 7]. More recently, independent companies and consortia have been

978-3-9810801-2-4/DATE07 © 2007 EDAA

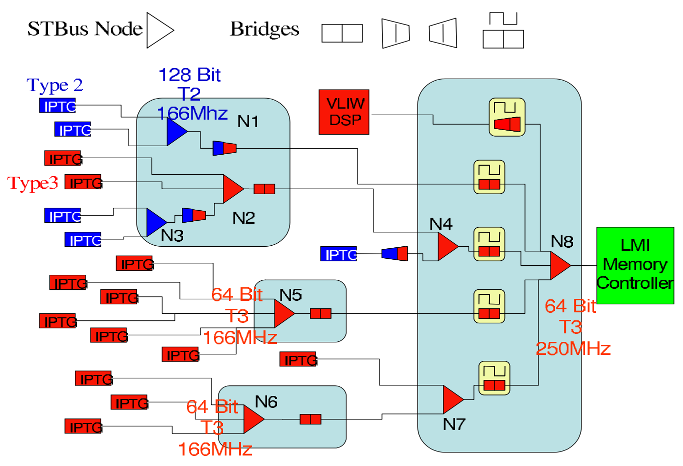

Figure 1: STBus reference MPSoC platform.

established to develop and license system-level communication architectures [9, 23, 22, 8].

A number of advances in this field comes from the open literature. In the context of communication synthesis methodologies, several co-synthesis frameworks are proposed, to couple interconnect synthesis with processor architecture or memory subsystem design [3, 14]. Other approaches exploit configurability of the communication architecture to come up with adaptive systems, as discussed in [11, 15]. The level of abstraction at which interconnect design methodologies operate range from floorplannig-awareness [6] to formal concurrency models [16]. Automated bus topology synthesis frameworks are presented in [2, 4, 5].

The resource sharing mechanism of the communication architecture is the focus of many works. SoC busses can implement priority-based policies [7, 8], time-division multiplexing [9] or token-ring mechanisms [10]. [1] and [21] propose more flexible bus bandwidth allocation schemes. The impact of common arbitration strategies on system performance is illustrated in [13].

The impact of the communication architecture on system performance is studied in [12]. [17] illustrates a comparative analysis between two industry-standard communication systems under realistic workloads and with different system configurations. Finally, scalability analysis of SoC interconnect protocols is performed in [20]. Our work extends these preliminary explorations by focusing on an industry-relevant target system, featuring a much higher level of complexity and accounting for the behaviour of real-life system components.

3. MPSOC PLATFORM MODELLING

Industrial MPSoC platforms often implement multi-layer communication architectures, built up by composing basic interconnect units through bridges. Our modelling effort targets such complex multi-layer systems. In particular, we focus on an internally-developed MPSoC platform for consumer applications. Its mission-critical subset is illustrated in Fig.1.

The platform is made up of a number of IP Cores, grouped in several functional clusters, each one implementing functionalities like video stream decrypting and decoding, image resizing or more generic DMA tasks, and therefore features different combinations of data width, clock frequency and STBus protocol type. Proprietary STBus converters and adapters (named *GenConv*) are in charge of bridging the heterogeneous clusters, and make use of buffering resources to store bus requests, responses and outstanding transactions.

The system uses the traditional unified memory architecture with a single off-chip DDR SDRAM, which then becomes the target for the bulk of the bus transactions. This architectural template is frequently used to cut down on the cost for technology integration of embedded memories and when data footprint forces to store processing data in a large off-chip memory.

Message-based arbitration is set in STBus nodes, i.e. packets are grouped in messages and arbitration rounds in the nodes occur at the message granularity. Messaging is a solution to generate memory controller-friendly traffic, in fact it ensures that a sequence of transactions that can be optimized by the memory controller and turned into high performance memory accesses are kept together all the way to the controller and are not interleaved with other transactions.

The ST220 VLIW DSP core (400 MHz, 32 bit, data and instruction caches) acts as the general purpose processor. It is connected to an upsize (from 32 to 64 bit) and frequency (from 400 to 250 MHz) converter.

The whole platform was modelled and simulated with clock-cycle accuracy and a SystemC-based virtual platform [17] was used as the backbone environment. The modelling effort resulted in a platform model with multiple abstraction levels. In order to speed up the analysis, functional traffic generated by the most critical audio and video IP cores was reproduced by means of configurable traffic generators (*IPTGs*). The DSP core was then modelled at the level of its instruction set, and runs a synthetic benchmark tuned to generate a significant amount of cache misses interfering with the traffic patterns of the other cores. Reverse engineering was then needed to generate a SystemC model of the memory controller. Deriving SystemC models from the RTL description would have been a time consuming process and would have resulted in slow simulation performance. We therefore built the model by careful inspection of the RTL waveforms, thus coming up with a more abstract representation of the controller with respect to RTL-equivalent description, while retaining timing accuracy at the memory controller interface. Finally, the modelling effort was completed by the integration of each sub-component into the complete platform and by setting up a statistics collection system.

A description of the main system component models follows.

3.1 STBus platform component models

The **STBus Interconnect** is the proprietary communication system developed at STMicroelectronics[19]. STBus leverages two physical channels, one for initiator requests and one for target responses, and supports split transactions. While a system initiator is receiving data from an STBus target, another one can issue a second request to a different target. As soon as the response channel frees up, the second request can be immediately serviced, thus hiding target wait states behind those of the first transfer. The amount of saved wait states depends on the depth of the prefetch FIFO buffers at the target side.

STBus implements three different protocols featuring increasing complexity. Type 1 is the low cost implementation for low/medium performance. Type 2 introduces compound operations, source labeling, priority labelling and posted writes. Split and pipelined transactions are fully supported. Finally, Type 3 provides the additional system efficiency associated with shaped request/response packets and the ability to support out-of-order transactions.

The **Generic Converter** is an STBus configurable block performing clock domain crossing, data with and STBus protocol type conversion. These functions can be performed standalone or in any combination within the same instance, according to the given configuration. Combining conversions has the advantage of minimizing the latency and the area with respect to having them managed separately.

IPTG is a SystemC block developed at STMicroelectronics aimed at reproducing the communication behaviour of a generic IP. In its simplest configuration, IPTG can generate bus traffic which obeys some statistical properties, i.e in terms of burst length, transaction types, addressing schemes, or it can also issue a transaction according to a specified sequence. However, IPTG is best used to emulate the behaviour of complex real-life IPs: such IPs can be often seen as having a number of internal sub-process (or agents), each one with its own characteristics (buffering space, transaction pipelining capability) but in some way dependent on each other (e.g., when operating in pipeline). With IPTG, each agent traffic is handled automatically according to its characteristics, and inter-agent synchronization points can be set to emulate dependencies between them. Once instantiated in a platform, IPTGs will generate bus transactions at different abstraction levels (transaction-level, bus cycle-accurate) according to what is specified in a per-IP configuration file, where all the required options and parameters are set. IPTG turns out to be a quite powerful and handy tool to the system integrator, as it allows to try out the SoC communication infrastructure in real-life conditions such as heavy-loaded transients which are not likely to be reproduced using random packet injection.

We developed a **memory controller** SystemC model and validated its functional correctness and timings with RTL signal waveforms on a cycle-by-cycle basis. The controller follows the internal LMI specification of SDRAM memory interface. The model includes a bus dependent and a bus independent part, thus easily allowing porting to several bus infrastructures. In this work, an STBus target interface is implemented. Input and output FIFOs allow storage of incoming packets or injection of outgoing packets into the bus. FIFO size and bus data width are tunable parameters.

The controller implements an optimization engine. This latter performs memory access optimizations such as opcode merging and variable-depth lookahead, and generates the corresponding sequence of SDRAM commands (e.g., precharge, autorefresh, active, read, write) while meeting SDRAM timing specifications (e.g, TRAS, TCAS), which are model parameters. Data size and burst size are tunable as well. The controller can drive both SDR SDRAM and DDR SDRAM memory devices.

Since the model comes out of reverse engineering from careful inspection of RTL waveforms, operation latencies are back-annotated so to keep timing accuracy at the memory controller bus interface.

3.2 Architectural variants

We extended the modelling capability of the virtual platform described so far in order to explore architectural variants and assess the interaction between the communication architecture and the other MPSoC subsystems.

On one hand, we ported new communication protocols (AMBA AHB and AMBA AXI) and made them inter-operate with the traffic generators and the off-chip memory controller, so to keep the same platform template. Since in the real system both audio and video IP cores and the memory controller natively come with STBus interface, we developed bridges for protocol conversion purposes, and accounted for the corresponding latencies. On the other hand, we replaced the off-chip SDRAM memory with an on-chip shared memory, so to test the system with a memory core featuring a cheaper access cost. Finally, we developed several platform instances (which we call the *collapsed* variants) where the most heavily congested cluster (node N5 in Fig.1) is removed and its communication actors attached to the central N8 cluster. This way, we intend to compare a centralized versus a distributed solution for the communication architecture, thus spanning the bus access versus multi-hop latency trade-off.

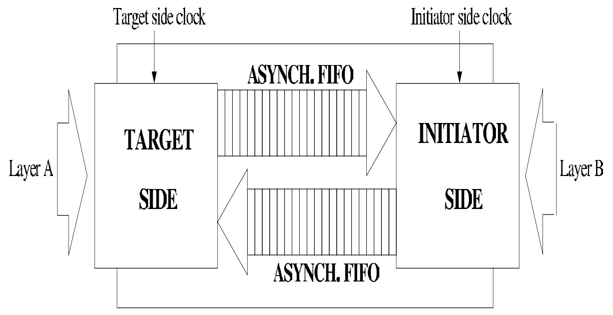

Figure 2: Generic scheme for hybrid bridges.

The design of these architectural variants required the development of new component models, which are hereafter briefly described.

The **AMBA AHB** [7] system backbone consists of a shared communication channel connecting multiple system cores. The channel is composed of two split and unidirectional data links (one for reads, one for writes), but only one of them can be active at any time, thus preventing the multiplexing of requests and responses on the interconnect signals. Transaction pipelining (*i.e.* split ownership of data and address lines) is supported to provide for higher throughput but not as a means of allowing multiple oustanding transactions. Bursts are supported by AHB masters and arbiter as a way to amortize arbitration time, and the non-posted paradigm for write transactions is implicitely assumed. The SystemC model of the AHB interconnect we developed does not implement split transactions.

AMBA AXI [7] builds upon the concept of point-to-point connection. Therefore, it can be used to connect (i) a communication initiator to a bus, (ii) a communication target to a bus, or (iii) directly an initiator to a target. In practice, the connection between any two devices translates into the instantiation of a master interface and of the symmetrical slave interface. Five different logical monodirectional channels are provided in AXI interfaces, and activity on them is largely asynchronous and independent (2 address channels, a read data and a write data channel, and a channel for write responses). This allows to support multiple outstanding transactions (with out-of-order or in-order delivery selectable by means of transaction IDs). Other advanced features include burst transactions with only the first address issued and the support for register insertion for timing closure transparent to the protocol. AMBA AXI modelling was based upon the SystemC libraries provided within the Synopsys CoCentric/DesignWare suites.

We modelled in SystemC a number of **bridges**: AHB-AHB, AXI-AXI, AHB-STBus, AXI-STBus, AHB-AXI, STBus-AHB, STBus-AXI. Their generic architecture is reported in Fig.2, where we identify a target side, an initiator side and asynchronous FIFOs providing support for different clock domains. The developed bridges have some common features: (i) they handle write transactions in a store-and-forward fashion, (ii) they have a blocking target side in presence of read transactions and (iii) they have tunable latency. These bridges were not designed to be competitive with the highly optimized STBus-STBus ones. They rather implement basic bridging functionality and do not exploit some advanced features of the communication protocols. For instance, they are always blocking on read transactions. This does not affect our results, since our goal is not a crossbenchmarking of communication protocols, but rather an analysis of how several communication protocol features, interconnect fabric topologies and memory architecture configurations combine together to determine global performance metrics. From this viewpoint, deploying lightweight bridges even for advanced communication protocols saves power and area (a typical GenConv bridge performing frequency conversion between T3 nodes at 64 bits can be as large as an STBus node with 5x3 crossbar topology at 64 bits), but penalizes across-layer communications. Therefore, it will be our interest to find out how sensitive system performance is with respect to bridge functionality.

4. INTERACTION BETWEEN COMMUNICATION AND MEMORY SUB-SYSTEMS

Since system performance in a multi-layer architecture results from the interaction of traffic patterns at many congestion points, we first analyze the single layer scenario, and get indications that pave the way for a better understanding of the multi-layer scenario.

4.1 Single-layer architecture

4.1.1 Many-to-many traffic pattern

The work in [20] analyzed single-layer shared busses with many slave devices attached, and proved the effectiveness of STBus and AXI in handling parallel communication flows. While memory wait states translate into idle cycles for AMBA AHB, STBus and AXI are able to mask them. If we set the size of AXI and STBus buffer stages to the minimum needed for correct operation, the two schemes perform similarly with bus utilizations up to 80% (ratio of bus busy cycles over execution time), while above that threshold AXI proves more robust to traffic congestion. Its fine granularity arbitration and its high number of physical channels (5) make the difference, however STBus was showed to bridge the performance gap by adding more buffering resources at the target interfaces. In general, large room for performance differentiation of communication protocols does exist when many-to-many traffic patterns have to be accommodated. This is the case of SoCs with many embedded memory cores and with communication spread evenly in the design. Video processors are an example thereof [24].

4.1.2 Many-to-one traffic pattern

Here we still focus on single-layer shared busses, but with a single slave device, hence accommodating a many-to-one traffic pattern. This reflects the architecture of the clusters in Fig.1. In this experiment, all IPTGs generate bursty read accesses to the shared memory (an on-chip core with 1 wait state), however we proved the independence of the results from the mix of bus transactions.

In this memory-centric scenario, the maximum bus efficiency is posed by the memory controller, since there are no opportunities to parallelize access patterns to different memory devices. With our simple controller, the response data channel in each communication architecture (carrying read data) is forced to work with 50% efficiency: 1 data transfer followed by 1 idle cycle (corresponding to 1 memory wait state). What can degrade such efficiency is the handover overhead incurred by the communication protocols. Let us consider the handover between two consecutive burst read transactions. While the last read data word of the first burst is injected into the bus by the slave device, the new address associated with the next burst should be concurrently driven by the next master actor on the bus. This way, the new memory access can be readily initiated on the next clock cycle.

AMBA AHB can hide bus handover overhead by changing the HGRANTx signals when the penultimate address in a burst has been sampled. The new master can then readily drive the bus address lines without any handover overhead. Interestingly, the many-to-one traffic pattern is the best operating condition for AMBA AHB. STBus achieves the same performance but with a different mechanism. The target asserts a grant signal to the next initiator in the same cycle it provides the last response data unit to the previous initiator. Since this grant signal is propagated asynchronously from the target to the waiting initiator through the STBus node in the same clock cycle, the next transfer can start immediately without handover overhead and the response channel can be driven with 50% efficiency. Finally, AXI is able to sustain the required efficiency by means of the burst overlapping mechanism. A master can drive another burst address after the slave accepts the address of the previous burst transaction. This enables the slave to begin processing data for the second burst in parallel with the completion of the first burst or, at least, to have the control information for the next transfer readily available for sampling. Therefore, even the internal read data lane of an AXI bus can be operated with 50% efficiency.

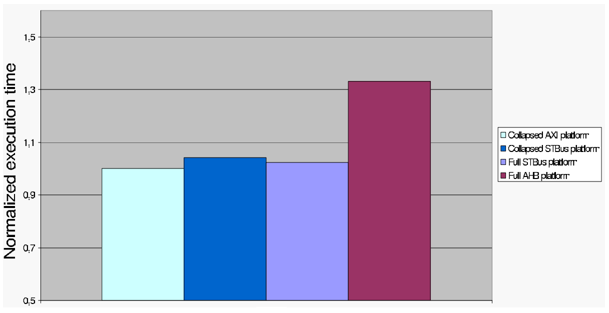

Figure 3: Performance of MPSoC platform instances.

Given the above considerations, our simulations did not show significant differences between performance of the communication architectures in the single-layer, single-slave scenario, hence results are not reported here. More complex memory controllers may implement buffering and perform optimizations on queued transactions, hence at least split transaction support would be required to the communication protocols.

4.2 Multi-layer architecture

Let us now extend our analysis to the complete MPSoC platform described in section 3. IPTGs reproduce traffic patterns of real-life IP cores.

As regards the memory architecture, we still consider a simple memory controller driving an on-chip shared memory with 1 wait state and postpone the analysis with LMI memory controller.

The first two bars in Fig.3 compare the normalized execution time of the collapsed platform instances, since this makes the role of the bridges and initiator interface complexity negligible. AXI and STBus collapsed variants exhibit almost the same performance, thus confirming the findings of subsection 4.1.2. In fact, the collapsed communication architectures resemble the single-layer single-slave scenario, where all interconnects were showed to perform almost the same.

The third bar in Fig.3 allows to compare the single-layer STBus with the full (multi-layer) STBus platform. Again, the two solutions show negligible differences in their performance metrics. The multi-layer approach introduces distributed buffering and relies on multiple outstanding transaction support of its master bus interfaces. The longer transaction latency associated with crossing the master-to-slave path in the multi-layer architecture is compensated by the initiators' capability to initiate new transactions before the

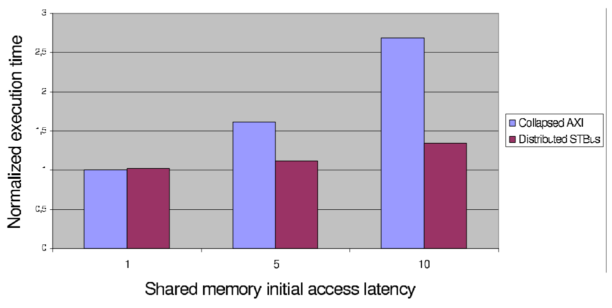

Figure 4: Performance of distributed vs centralized communication architectures as a function of memory speed.

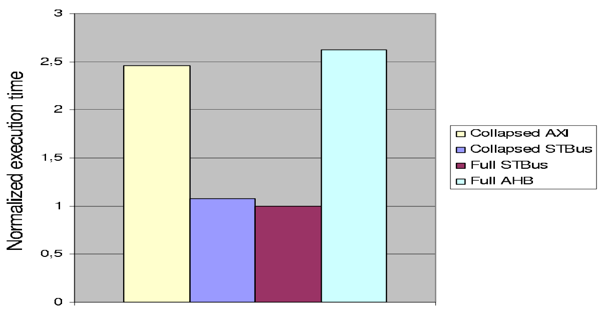

Figure 5: Performance of platform instances with LMI memory controller.

previous ones complete. The resulting performance is equivalent to that of a single-layer architecture, where the multiple oustanding transaction capability does not help much since the target interface has a single-slot buffering here. Therefore, each transaction is blocking, which compensates the benefits of a shorter master-to-slave path.

The performance ratio between collapsed and distributed interconnect solutions however changes if the memory device gets progressively slower in responding to access requests. Fig.4 clearly shows the increasing advantage of distributed solutions as the memory latency increases. Please note that the use of AXI and STBus is interchangeable here, what really matters is the architecture topology. A fast memory penalizes communication architectures with large crossing latencies. In contrast, a slow memory makes distributed solutions preferrable, since the distributed buffering allows multiple outstanding transactions capable bus interfaces to keep pushing transactions into the bus. The master-to-slave multi-hop path gets therefore filled, and the system is able to deliver high throughput.

However, collapsed solutions are not always feasible. First, physical limitations as well as layout and routability constraints prevent from connecting tens or hundreds of actors on the same interconnection layer. Moreover, communication actors with similar interfaces (e.g., datawidth, protocol type) may have to be grouped in the same cluster, thus cutting down on the number of required adapters. Finally, clusters of homogeneous cores favour performance closure. If we restrict our analysis only to distributed architectures, we can easily realize that bridges become a key component for system performance. In fact, from Fig.3 we can compare performance of multi-layer STBus and AHB architectures. Please note that this is the best operating condition for AHB, since the memory device responds with just 1 clock cycle delay. We can observe that AHB solution is ineffective, due to the fact that AHB-AHB bridges are blocking on each transaction. The source node of the bridges are non-split AHB layers, and are therefore blocked until the transaction in progress is completed.

The blocking behaviour of the bridge may depend either on intrinsic limitations of the protocol semantics or on low-cost implementations of the bridges. For instance, we realized lightweight AXI-AXI bridges as illustrated in Fig.2 with blocking target side on read transactions. In fact, while write transactions can be easily managed by means of a store-and-forward policy, implementing non-blocking read transactions has a heavier impact on bridge complexity (control information must be stored and reassociated with response data, read requests might be serviced in a different order than they were received, etc.). Because of this lightweight bridge design, we found a performance for the distributed AXI platform almost equivalent to the full AHB platform. Therefore, advanced features of AXI, which could potentially reach the same performance level of STBus, are vanished by poor bridge functionality. For this reason, bridge engineering is becoming increasingly complex as advanced features are introduced in communication protocols, and they can be viewed as true IP blocks. In our case, the availability of proprietary STBus bridges made performance of STBus distributed platforms hardly achievable by the other distributed schemes.

The final scenario which is left to investigate is the multi-layer architecture with LMI memory controller and off-chip DDR SDRAM memory in place of the on-chip shared memory. The main differences lie in the higher response latency of the memory subsystem (11 cycles to get the first read data word since the request was sampled), in the multi-slot FIFO implemented in the memory bus interface and in the optimizations performed on queued transactions (opcode merging, lookahead) by the memory controller. Since the LMI controller natively exhibits an STBus bus interface, a bridge is required to connect it to other interconnect fabrics. This bridge must be able to implement split transactions, i.e. not to block the bus when a new transaction is sampled and until the associated memory access is completed. Otherwise the input FIFO of the memory controller would never contain more than one pending transaction, and no optimizations could be performed. Fig.5 illustrates execution time of various platform instances. Since the memory response latency is high, we expect distributed platforms to outperform collapsed ones. However, collapsed AXI is much worst than collapsed STBus, since (i) this latter does not require a bridge, (ii) the multiple oustanding transaction capability of STBus initiators can be fully exploited to fill in the FIFO of the memory bus interface and (iii) thus memory controller optimizations can be exploited. In contrast, collapsed AXI was using a simple protocol converter unable to perform split transactions. For these reasons, collapsed STBus can approach the performance of distributed STBus.

If we restrict our analysis to distributed solutions, we notice that the performance gap between STBus and AHB has increased a lot with respect to Fig.3, due to the higher latency of the memory subsystem, which makes the penalty associated with non-split blocking bridges even more severe.

5. FINE-GRAIN PLATFORM PERFORMANCE ANALYSIS

The memory controller drains the bulk of all bus transactions in the platform of Fig.1. Therefore, properly monitoring the behaviour of the bus-memory controller interface can help system designers identify where bottlenecks are. For instance, should low bandwidth communication be monitored at the I/O interface, this might be due to the actual inefficiency of the memory controller or to the poor performance of the system interconnect. Our modelling

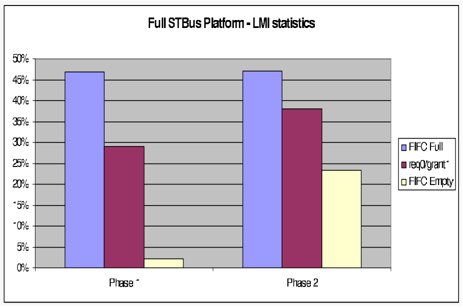

Figure 6: LMI statistics for the full STBus platform.

environment enables this kind of analysis at a fine granularity.

As an example, we report in Fig.6 the statistics taken at the bus interface of the LMI memory controller in our full STBus-based MPSoC platform. Two working regimes are illustrated out of the MPSoC application lifetime. During the first execution phase, the FIFO of the bus interface is full for 47% of the time, while it is available to store new transactions for the remaining 53%, partitioned as follows: for 29% of the time there are no incoming requests (request signal is 0 while grant signal is 1), and for remaining 24% the bus interface is storing new memory access requests. The FIFO is empty only for a marginal time fraction. We conclude that this execution phase exhibits intensive memory traffic which the interconnect is able to handle pretty well.

During the second phase, the time percentage during which the FIFO is full remains unaltered, while the FIFO is empty for a longer time. Therefore, the new traffic pattern has a lower intensity than the previous one on average, but is it more bursty. We repeated the same test for a full AHB platform, and found that the FIFO is never full (since our AHB implementation does not support split transactions) and that for 98% of the time there are no incoming requests. This clearly indicates that the system interconnect is the performance bottleneck, and not the memory controller.

6. SUMMARY AND CONCLUSIONS

We now turn our analysis findings into guidelines for designers of state-of-the-art industrial MPSoCs:

1) For single-layer systems, a significant performance differentiation between different communication protocols can be observed only when they have to deal with a many-to-many traffic pattern. In this context, advanced interconnets are able to hide slave response latency by processing parallel communication flows. This is achieved by means of a fine granularity of arbiter decisions (e.g., rearbitration of data links on a cycle-by-cycle basis) and of multiple physical resources (multiple communication channels, buffering in the bus and/or in the bus interfaces).

2) In single-layer systems with a centralized slave, the performance of this latter and of its control logic bounds the maximum performance that communication protocols can achieve. This upper bound depends on the slave response latency, on the amount of buffering implemented at its bus interface and on the optimizations which might be performed by its controller. When the required bus efficiency is low (e.g., 50%), simple interconnect fabrics may provide the same performance of advanced and more complex communication infrastructures.

3) In the absence of technology constraints, a distributed multilayer system interconnect results in significant performance speedups with respect to centralized ones only if (i) initiator bus interfaces support multiple outstanding transactions (ii) the target side of bridges is able to handle split/non-blocking transactions (iii) the response latency of target devices is long enough with respect to the data transport latency across a multi-hop interconnect.

4) As long as the features of the previous point are supported by a distributed system interconnect, performance differentiation of competing communication protocols is only marginal in presence of a centralized target bottleneck. On one hand, this puts emphasis on the generation of memory controller-friendly traffic at the initiators rather than on the optimization of the system interconnect. On the other hand, this calls for optimizations of the I/O architecture to remove the system bottleneck.

5) Bridges are becoming true IP blocks. The introduction of new features in communication protocols might be vanished by the deployment of lightweight bridges with basic functionality. More research is needed to understand whether it is really worth increasing bridge complexity, instead of keeping lightweight bridges for path segmentation and traffic routing and pushing complexity at the system interconnect boundaries, which is known as the network-on-chip solution.

6) The availability of complete modelling and simulation frameworks like the one developed for this work makes it easier to accurately identify system bottlenecks and to fine-grain tune the architecture for the application domain of interest. For instance, we have showed how to identify working conditions during application lifetime and how to discriminate between poor performance of the memory controller and of the communication architecture.

Acknowledgements

We would like to thank Francesco Poletti for his support in SystemC model development. We also acknowledge support of STMicroelectronics SPG for providing the IPTG traffic generators, and all the OCCS team for its contribution to this work.

7. REFERENCES

[1] K.Lahiri, A.Raghunathan, G.Lakshminarayana, "The LOTTERYBUS on-chip communication architecture", Trans. on VLSI Systems, Vol.14, no.6, pp.596-608, June 2006.
[2] S. Pasricha, Y. Park, F. Kurdahi, N. Dutt, "System-Level Power-Performance Trade-Offs in Bus Matrix Communication Architecture Synthesis", CODES+ISSS 2006.
[3] S. Pasricha, N. Dutt, "COSMECA: Application Specific Co-Synthesis of Memory and Communication Architectures for MPSoC", Design Automation and Test in Europe, pp.700-705, 2006.
[4] S.Pasricha, N.Dutt, M.B.Romdhane, "Constraint-driven bus matrix synthesis for MPSoC", Asia South Pacific Design Automation Conf., pp.30-35, 2006.
[5] S. Murali and G. De Micheli, "An Application-Specific Design Methodology for STbus Crossbar Generation", Design Automation and Test in Europe, pp. 1176-1181, 2005.
[6] S. Pasricha, N. Dutt, E.Bozorgzadeh, M. Ben-Romdhane, "FABSYN: Floorplan-aware Bus Architecture Synthesis" IEEE Trans. on VLSI Systems, Vol.14, no.3, pp.241-253, March 2006.
[7] ARM Ltd., Sheffield, U.K., AMBA 2.0/3.0 Specifications. Available: http://www.arm.com/armtech/AMBA
[8] Siemens AG, Open Microprocessor Initiative, OMI 324 PI Bus, Rev 0.3d 1994, OMI Standards Draft.
[9] Sonics Inc., Sonics Integration Architecture. Available: http://www.sonicsinc.com
[10] J. Turner and N. Yamanaka, Architectural choices in large scale ATM switches, IEICE Trans. Commun., vol. E-81B, no. 2, pp.120-137, Feb. 1998.
[11] Sekar, K.; Lahiri, K.; Raghunathan, A.; Dey, S.; "Integrated Data Relocation and Bus Reconfiguration for Adaptive System-on-Chip Platforms", Design Automation and Test in Europe, Vol.1, pp.1-6, march 2006.
[12] F. Polloni et al., "Fast System-Level Design Space Exploration for Low Power Configurable Multimedia System-on-Chip", 15th IEEE Int. ASIC/SoC Conference, pp 150-154, Sep. 2002.
[13] F.Poletti, D.Bertozzi, A.Bogliolo, L.Benini, "Performance analysis of arbitration policies for SoC communication architectures", Journal of Design Automation for Embedded Systems, pp.189-210, 2003.
[14] A. Wieferink et al., "System level processor/communication co-exploration methodology for multiprocessor system-on-chip platforms", IEE Proc. on Computers and Digital Techniques, Vol. 152, Issue 1, pp.3-11, 2005.
[15] Chulho Shin et al., "Fast exploration of parameterized bus architecture for communication-centric SoC design", Design Automation and Test in Europe, pp.352-357, Vol.1, 2004.
[16] Xinping Zhu; Wei Qin; Malik, S.; "Modeling operation and microarchitecture concurrency for communication architectures with application to retargetable simulation", IEEE Trans. on VLSI Systems, Volume 14, Issue 7, pp.707 - 716, 2006.
[17] Loghi, M.; Angiolini, F.; Bertozzi, D.; Benini, L.; Zafalon, R.; "Analyzing on-chip communication in a MPSoC environment", Design Automation and Test in Europe, pp. 752-757, Vol.2, 2004.
[18] IBM Corporation, Armonk, NY, CoreConnect Bus Architecture. Available: http://www.chips.ibm.com/products/coreconnect/.
[19] ST Microelectronics, STBus Interconnect. Available: http://www.st.com/stonline/prodpres/dedicate/soc/cores/stbus.htm
[20] M. Ruggiero, F. Angiolini, F. Poletti, D. Bertozzi, L. Benini, R. Zafalon, "Scalability Analysis of Evolving SoC Interconnect Protocols", Int. Symposium on System-on-Chip, Nov 16-18, 2004, pp. 169-172.
[21] N.Wang, M.A. Bayoumi, "Dynamic fraction control bus: new SOC on-chip communication architecture design", IEEE Int.SoC Conference, pp.199- 202, 2005.
[22] Open Core Protocol International Partnership (OCP-IP). Available: http://www.ocpip.org
[23] Crossbow Technologies. Available: http://www.crossbowip.com
[24] S.Murali, M.Coenen, A.Radulescu, K.Goossens, "A methodology for mapping multiple use-cases onto networks-on-chips", Design Automation and Test in Europe, pp.118-123, 2006.