# Flexible MPSoC Platform with Fast Interconnect Exploration for Optimal System Performance for a Specific Application

Florin Dumitrascu, Iuliana Bacivarov, Lorenzo Pieralisi, Marius Bonaciu, Ahmed A. Jerraya
*TIMA Laboratory, Grenoble France*
{Florin.Dumitrascu, Iuliana.Bacivarov, Lorenzo.Pieralisi, Marius.Bonaciu, Ahmed.Jerraya}@imag.fr

## Abstract

*One of the key elements in Multi-Processor Systems-on-Chip (MPSoC) design is to select the optimal on-chip interconnect architecture, in order to maximize the overall system performance.*

*This paper proposes a flexible MPSoC platform, designed for a target application, which allows customizing the interconnect by selecting various architectures. It allows fast building of executable models from architecture specifications and performance evaluation using the cycle-accurate cosimulation.*

*We experimented a DivX encoder application with three different interconnects: DMS (Distributed Memory Server), AMBA bus and Octagon Network-on-Chip (NoC). The simulation results relative to performance metrics such as, average latency, throughput and execution time allowed to compare these different interconnect architectures, to verify the application real-time constraints and to propose further optimizations.*

## 1. Introduction

One of the most important steps in MPSoC design is the selection of the optimal architecture for the on-chip communication, under a given application workload [6][7]. Apart from the efficiency of the computing resources, the system performance depends on how efficiently the interconnect can handle the application workload.

Current design efforts are conducted to build scalable and reusable communication architectures, using on-chip interconnection networks instead of ad-hoc wiring [1]. Though, the communication design is still driven by the application requirements.

A large body of research has focused on the exploration of interconnect architectures. The most representatives are described next.

- StepNP is an exploratory simulation environment for exploring router applications, multiprocessor Network Processing architectures, and SoC tools [9]. Different NoCs, i.e. Octagon and SPIN, were integrated using a standard communication channel named SOCP. In this way, StepNP provided a fixed interface to the application. Moreover, this platform was mainly oriented to Network Processing applications.

- MPARM [10] provides a SystemC cycle-true platform to simulate a complete multi-processor system at the cycle-accurate and signal accurate level. It investigated the impact of the interconnect infrastructure on system performance at the highest level of accuracy. However, the tests run on bus-based architectures (i.e. AMBA and ST Bus), ignoring NoCs. One drawback of this platform was the evaluation speed, as the simulations run at RTL-level. Other limitation was the use of test benchmarks instead of the real application.

- A complete platform for analysis and trade-off exploration of MPSoC communication architectures, providing a realistic performance analysis of on-chip interconnects was derived in [11]. However, it did not account for switching between different interconnect architectures. The interconnect was fixed, and only some parameters could be tuned for optimization purposes.

We identified the main drawbacks of the previous exploratory platforms as: the lack of flexibility, the low level of accuracy or the slow simulation speed.

In this paper we eliminate some of these drawbacks, by proposing a flexible MPSoC platform able to evaluate fast and accurately the performances of a given application with different interconnect structures. The platform could predict the optimal interconnect first by switching between several interconnects (i.e. bus-based, NoC or other dedicated structure) and second by varying their parameters.

In this paper, our contribution is as follows.

- Flexibility: the automatic generation of various adaptation interfaces. The adaptations are performed between a target application and diverse interconnects including buses, NoCs and more complex communication structures;

- High evaluation speed: high-level evaluation models based on cosimulation;

- Precise evaluation: on one hand cycle-accurate models for the application and interconnect, and on the other hand the utilization of real interconnect implementation.

3-9810801-0-6/DATE06 © 2006 EDAA

The rest of the paper is organized as follows. Section 2 presents the used design framework. Section 3 describes the proposed design flow for interconnect performance evaluation: the definition of the application profile, the switching between several interconnect models, their adaptation and the performance evaluation methodology. Section 4 provides implementation details on these elements. Section 5 presents experimental results.

# 2. The design framework

## 2.1 The design flow

The flow used for the exploration of the interconnect architecture is illustrated in Figure 1.

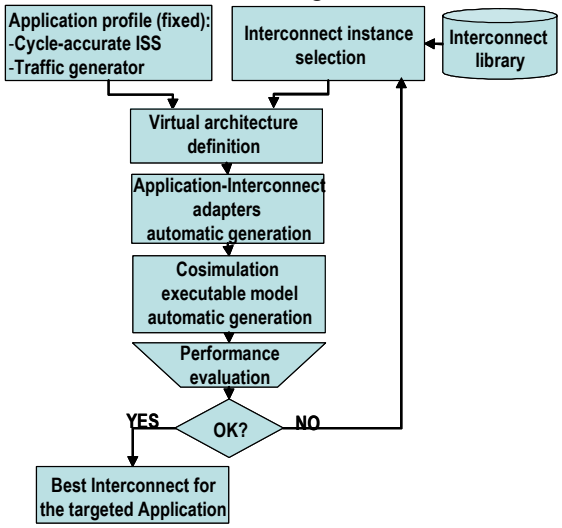



**Figure 1. Generic MPSoC design flow providing the best interconnect solution**

- The application profile is derived from the initial application. It is decomposed into SW modules running on traditional ISSs and HW modules represented by traffic generator models. In this paper, the profile is defined as a SystemC Timed Executable Model. It will be discussed in Section 3.1.

- The on-chip interconnect architecture is the customizable part. We can choose from three models available in our components library: DMS (Distributed Memory Server) [5], AMBA AHB bus and Octagon NoC [4]. They will be detailed in Section 2.3.

- Both the application profile and selected interconnect are described in a SystemC subset called virtual architecture [2]. The virtual architecture enables the automatic composition of a high-level executable model.

- The performance evaluation of the entire architecture is done through cosimulation, using the SystemC environment. If the design meets imposed constraints, the chosen interconnect will be implemented on the final chip. Otherwise, its parameters will be tuned, or another interconnect will be selected. The cosimulation results show which is the optimal interconnect, with respect to the communication workload requirements of the targeted application.

## 2.2 The adaptation between interconnect and application components

In heterogeneous MPSoC architectures, it is often the case that the interface of the subsystems must be adapted to interconnect APIs. The challenge consists in providing suitable components whose role is to adapt the chosen programming model to the communication infrastructure.

An adaptation layer is needed for this purpose. The adapters have the main role of translating the application communication primitives, into interconnect transactions defined by each protocol.

In this paper, the application communicates through message passing, by calling a set of high-level MPI primitives. At its turn, each interconnect implements specific communication protocols, e.g. master/slave protocol for an AMBA AHB bus.

## 2.3 The interconnect design library used in the experiment

The on-chip interconnection library contains three candidate models: the Distributed Memory Server (DMS), the AMBA AHB bus and the Octagon NoC. Their structure and communication APIs are presented in this section.

**Distributed Memory Server (DMS)**

The Distributed Memory Server (DMS) is a flexible and scalable data transfer architecture for MPSoC with massive distributed memory [5].

The used DMS is described as a cycle-accurate TLM model. The MPI interface provided by this model acts as a data transfer engine between the local memory of the caller subsystem and the memories of other subsystems. The data transfer primitives are **mpi_send** and **mpi_recv**.

**void mpi_send (lch, laddr, size);**

**void mpi_recv (lch, laddr, size);**

The targeted module is uniquely identified through the local channel (**lch)** parameter. The message relative address is denoted by the **laddr** parameter. The base address for the local memory is determined at module initialization time by calling the **sram_init()** primitive**.** The message length is specified using the **size** parameter.

The synchronization between two modules is insured by specific control signals. For instance, the initiation of a call for **mpi_send**/**mpi_recv** by a module is blocked until the corresponding **mpi_recv**/**mpi_send** is started by the module connected to the other end.

**AMBA AHB bus**

The AMBA AHB (Advanced High-performance Bus) bus is defined as a TLM model, using the OCCN methodology [3]. This model provides the full accuracy of the AMBA AHB protocol [8], in terms of **read**/**write** transactions timings or bus arbitration delays.

The AMBA MPI interface is based on the OCCN

MasterPort/SlavePort API. The data transfer is controlled by the Master Modules through the following primitives:

**single_write (data, size, addr);**

**single_write_leaving (data, size, addr);**

**single_read (data, size, addr);**

**single_read_leaving (data, size, addr);**

These primitives write/read a single word of data with a length of **size** bytes to/from a location specified by **addr** parameter. The bus is locked with the **single_write**/ **single_read** primitives and unlocked after each transfer with the **single_write_leaving**/**single_read_leaving** primitives.

**Octagon NoC**

The Octagon NoC [4] interconnects routers using a simple and regular topology. Resources are attached to routers at the edge of the network through Network Interfaces (NI). The main advantage of communication through NIs is the full decoupling of module behavior from inter-module communication [5]. It was implemented at cycle-accurate TLM level, using OCCN methodology [3].

The Octagon specification provides the following MPI interface to the application:

**void send (const msg_type& flit);**

**msg_type* recv ();**

## 3. Performance evaluation of the interconnect subsystem

Our final objective is to determine the best on-chip interconnection scheme in terms of communication performance, for a predefined embedded application. This section presents the main steps of the design flow: the application profile design, interconnect selection, adaptation layer generation and performance evaluation.

### 3.1 Application profile definition

In order to simplify the design and benefit of a fast simulation, the application component models act as traffic generators. It is possible then to simulate with clock accuracy the output trace of the real application.

The timing information necessary to simulate the timestamp between network transactions is represented through time annotations. They correspond to **WAIT()** functions in the SystemC model. Timing information is captured in tables depending on the chosen configuration i.e. CPU subsystem architecture, cache size, CPU clock frequency. It simulates the computation delay and the times required by a CPU subsystem for operations such as local memory access or bus grant.

### 3.2 Interconnect model selection

The interconnect architecture is selected as one instance from the available models in the library. These models can be defined in three different ways: (1) using the virtual architecture specification, (2) using the OCCN methodology, and (3) using an OCCN model wrapped into a virtual component.

In this paper the interconnects are all described at clock-accurate TLM. This is useful for fast and accurate architecture exploration.

### 3.3 The executable MPSoC model

Figure 2 describes the cosimulation platform. The HW and SW subsystems are represented by their timed traffic generators. The interconnect structures are described at cycle-accurate TLM.

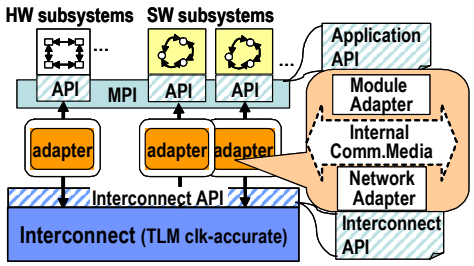



**Figure 2. The executable MPSoC model**

The adaptation function between the different APIs is provided by abstract adaptation interfaces. They mainly transform channel access via internal ports to channel access via external ports.

By using the virtual architecture specification, the adaptation wrappers can be generated automatically [2]. Their structure is detailed in Figure 2.

An adaptation wrapper mainly consists of one module adapter and one or more network adapters, one for each network access point. The module adapter is in charge with data conversion and channel resolution. The network adapter realizes the protocol conversion, when it is needed.

### 3.4 Performance evaluation

The considered performance metrics are latency, throughput and execution time. They are dynamically monitored during cosimulation, in order to analyze the effectiveness of inter-module communication and adaptation components. In order to acquire the Send/Receive primitives latency and throughput, the OCCN [3] functions named StatDelay() and respectively StatInstantThroughput() are used.

For each interconnect (i.e. AMBA bus, DMS and Octagon), the critical parameters (e.g. clock frequency, NoC topology, etc.) may be tuned during the design in the case when the performance numbers are not reached (Figure 1, the "NO" loop).

## 4. Adapters' generation

The DivX encoder application used in the experiments was designed for the DMS interconnect. The mpi_send/mpi_receive represents its native API. Based on the functional principle of the DMS, this section describes the implementation of the adapters translating the MPI primitive calls when replacing DMS with AMBA or Octagon NoC.

**DMS communication principle**

Figure 3 illustrates the DMS communication principle. The links which connect a module $M_X$ to its network access point (NA) are grouped into virtual channels (VC_$M_X$_NA).The module M1 sends data to modules M2 and M3.

The DMS addressing scheme defines point to point connections. In our example these are configured as follows:

**M1 -> M2 is configured as [NA_1 : 4 – NA_2 : 0 ]**

**M1 -> M3 is configured as [NA_1 : 5 – NA_3 : 0 ].**

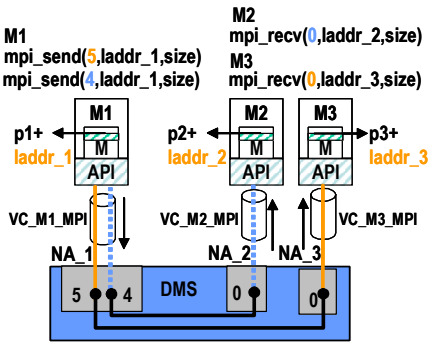



**Figure 3. DMS communication principle**

After module initialization $\mathbf{p_i}$ stores the base address for local message memory of module $M_i$.

The data transfer from module M1 to M2 is done through the following steps:

- M1 calls **mpi_send**;
- NA_1 processes the request by retrieving the message located in module's M1 memory at **p1 + laddr_1** address, and transferring it to the NA_2 buffer;
- data transfer from NA_1 to NA_2;
- M2 calls **mpi_recv;**
- NA_2 processes the request by transferring the message from its buffer to the local memory of module M2, at the address **p2+laddr_2**;

Data transfer from M1 to M3 occurs in a similar manner.

**Different interconnect cosimulation wrappers generation for AMBA bus and Octagon NoC**

In order to enable the communication through different interconnect structures, the **mpi_send/mpi_recv** calls are now processed by the adapters, instead of the DMS NA interfaces.

The adapters operate similarly. Their main functionalities consist in address translation and information formatting for the corresponding interconnect. The adapter's functionality for the OCCN AMBA bus and the Octagon NoC are presented in pseudo-code in Figure 4 (a) and (b).

The **mpi_send** primitive is called by an AMBA Master Module and respectively by a NoC Sender Module. The address translation uses the **laddr_1** parameter to retrieve the data from the local memory of the caller module. The **lch** parameter is translated into a valid receiver module address. For AMBA slave address-space range, the **lch** value could correspond to a hexadecimal address. For the NoC, **lch** will be translated into a valid destination ID.

**(a)**

```
mpi_send (lch_1,laddr_1,size )
(1) copy message memory block into buffer
(2) for (i=0;i<size;i++){   // segmentation
      create Header;        // add bus_addr
      create Data;          // add 32 bits from buffer
      send (Data+Header);
    }
mpi_recv (lch_2,laddr_2,size )
(3) - call receive( ) into a thread
    - store Data into buffer
    - if read finished, call recv_completion.notify()
(4) - wait (recv_completion)
    - copy buffer into message memory block
```

**(b)**

```
mpi_send (lch_1,laddr_1,size )
(1) copy message memory block into buffer
(2) create Header;   // add destID, source ID, msg. length
    Data = Message;
    send (Data+Header);
mpi_recv (lch_2,laddr_2,size )
(3) - call recv( ) into a thread
    - store Data into buffer
(4) - recover the original message (strip the header)
    - copy buffer into message memory block
```

**Figure 4. Send/Receive primitive implementation for (a) the AMBA bus; (b) Octagon NoC.**

The next step is sending the message over the datapath. For the AMBA bus, the message is split into 32bit words, each preceded by a header containing specific AMBA signals (e.g. HBUSREQ, HBURST, HLOCK, HADDR). Each pair data + control is sent over the bus using **single_write()/single_write_leaving()** primitives. They are based on the **send()** primitive of the OCCN MasterPort/SlavePort communication API [8].

For Octagon, the message will be preceded by a header with the destination ID, source ID and the message length. The resulted data is dispatched towards the NoC NI, using the **send()** Octagon API.

The **mpi_recv** call is handled by the AMBA Slave Adapter, and respectively by the NoC Receiver Adapter. They will wait to receive data, by calling into a thread the corresponding primitive, i.e. **receive()** for the OCCN AMBA API and **recv()** for the Octagon API.

Then, the adapter will strip the control information and will rebuild the data into the original message. In both cases, the message is written into the local memory of the receiver module, at the address specified by **laddr_2**.

## 5. Experiments

### 5.1 DivX application

In order to test the various interconnection schemes, we will use the specification of a DivX real-time encoder, illustrated in Figure 5.

The DivX functionality is as follows. The Splitter module receives a stream of non-compressed video data (QCIF). Each video frame is split in 4 sub-frames which are sent to 4 parallel encoding units (P0-P3). The data processed by each of the encoding processors is then forwarded to a Variable Length Coding (VLC) module which performs some additional encoding and also reconstructs the frame before forwarding it to the Storage module [5].

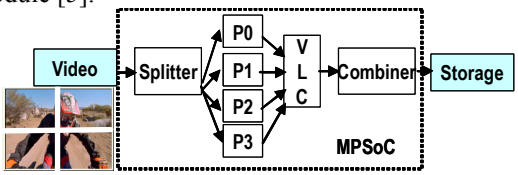



**Figure 5. DivX encoder application scheme**

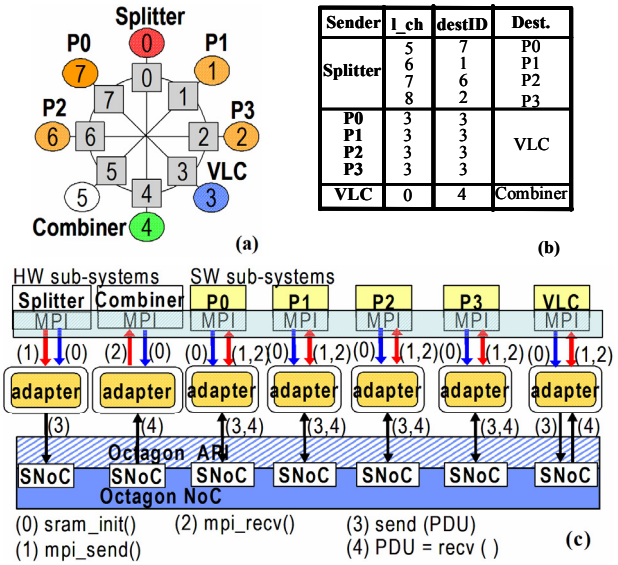
In order to map this application onto the evaluation platform, the virtual architecture mapping is the following. Both Splitter and Combiner modules are mapped onto HW. All the four encoding units (P0-P3) and the VLC are mapped onto ARM7 CPUs.

## 5.2 On application mapping onto AMBA bus and Octagon NoC architectures

### Application mapping on the AMBA bus

The DMS addressing scheme is identified on the base of the **lch** parameters. For example, the Splitter calls **mpi_send** for a write operation to P0, with the **lch** value of 5, and the VLC calls **mpi_recv** for a read operation from P0 by with the **lch** value of 0. The **lch** is then translated into the corresponding hexadecimal address on the AMBA bus.

### Application mapping on the Octagon NoC




| Sender | l_ch | destID | Dest. |
|---|---|---|---|
| Splitter | 5 | 7 | P0 |
| | 6 | 1 | P1 |
| | 7 | 6 | P2 |
| | 8 | 2 | P3 |
| P0 | 3 | 3 | VLC |
| P1 | 3 | 3 | |
| P2 | 3 | 3 | |
| P3 | 3 | 3 | |
| VLC | 0 | 4 | Combiner |

**Figure 6. (a) DivX mapping over the Octagon NoC; (b) local channel translation into a destination ID; (c) DivX executable MPSoC model, with Octagon NoC**

The NoC allows various mapping possibilities for the application modules, with different impact over the performance, depending on the traffic pattern. The ideal would be to have the Splitter-P0/P1/P2/P3, P0/P1/P2/P3-VLC and VCL-Combiner peers separated by only one hop (number of intermediate routing nodes) over the network. Figure 6 (a) shows the mapping used in our example.

Again address translation is necessary between DMS and Octagon NoC. Starting from the DMS addressing scheme in Figure 6 (b), the **lch** identifier associated with a send operation is translated into a destination ID in the range 0-7 (in the presented application, Octagon has 8 nodes).

Figure 6 (c) illustrates the executable model of the DivX encoder with Octagon interconnection. This is possible due to the adapters which translate application APIs into Octagon communication primitives. The adapters are generated automatically from the virtual architecture.

## 5.3 Performance evaluation and results

The test case is a movie encoding, for 125 frames at a resolution of 352x288 pixels. In order to make performance comparisons and optimization proposals, we investigated the total execution time, the average throughput and Send/Receive latency. Each of these metrics was computed for the modules that initiate communication: Splitter, VLC and Combiner.

**Table 1.Execution time comparison (clock cycles)**

| DMS | AMBA bus | Octagon NoC |
|---|---|---|
| 233,689,205 | 233,431,813 | 233,154,449 |

Table 1 illustrates the execution times for the three interconnects. Even if they are very close, not all of these interconnects can be used because of the real-time constraints imposed by the application.

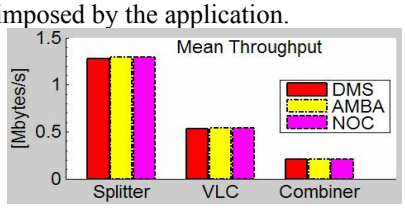



**Figure 7. Average throughput for Splitter, VLC and combiner, evaluated for the three networks: DMS, AMBA and NoC.**

The interconnect occupancy degree is a useful measure for optimization purposes. It is evaluated as the average ratio between the average throughput and the network bandwidth. In our case, we obtain almost the same throughputs for the three interconnects (Figure 8), indicating that none of these networks do saturate. Although average throughput values are similar, it will be seen later that this may hide real-time constraints violation.

By analyzing the latency, we can figure out if the interconnect is fitting the given application. In the case when the latencies exceed the real-time range, the real-time conditions imposed for the application are violated. In simple words, the real-time range is the time between

two consecutive frames. Due to design constraints, the Antenna is not buffered and it cannot be blocked. This means that all the computations and the data transfer should be achieved in this interval.

The latency for the AMBA bus (Figure 7 (a)) shows that in average, the bus can handle the application requests: 9 cycles transfer latency between Splitter and P0-P3. However, conflicts are detected in the bus. The large spikes, in the same figure, indicate an abnormal functioning of the application: thus real-time constraints are not met. However, these spikes cannot be detected by the average throughput because they have a very low rate, and they are over-whelmed by the average.

The latency per transaction (i.e. Send/Receive MPI) for the DMS interconnect was evaluated at 19 cycles for instance between Splitter and P0-P3, constant over the whole simulation time. This is due to the DMS communication that uses point-to-point interconnects, without conflicts.

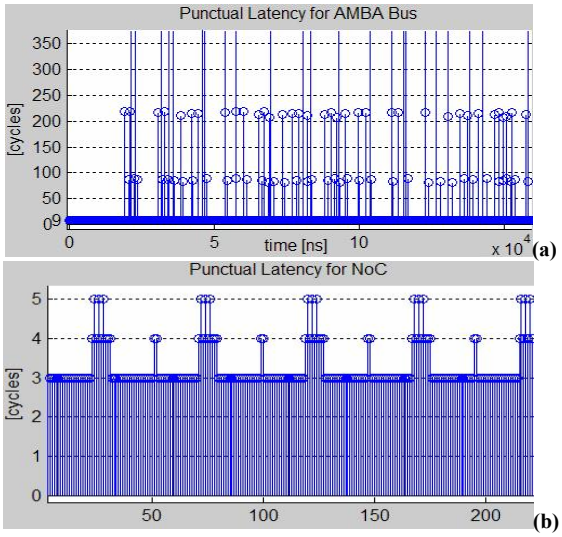



**Figure 8. (a) AMBA bus (b) Octagon NoC punctual latency**

Figure 7 (b) represents the latency for the NoC, for Send/Receive primitives. The values are lower compared to the other interconnects: varying between 3 and 5 cycles. This is due to the ASIC implementation of the routers and the fact that the NoC is designed to be congestion-free.

For systems that meet the real-time constraints (i.e. DMS, Octagon), the designer has several optimization possibilities. For optimization, we can push the latency or throughput to their maximum: we can slow-down the NoC frequency in order to save power; or reduce some of the unused interconnects in order to save on-chip area. The NoC advantage is that it is more flexible, easier to tune and it offers several freedom degrees.

On the other hand, for the systems that do not meet the constraints, the interconnect should be re-designed. For instance, for the AMBA bus this can be solved by re-considering the priorities, or the arbitration policy.

## 6. Conclusions

This paper presented a flexible platform for the performance evaluation of the interconnect subsystem for a target application, providing optimal MPSoC performance. The final objective was to determine the best on-chip interconnection scheme, from a library of interconnect structures.

The basic principle was the automatic generation of adapters between the application and the different interconnect structures. In the experiments, the used application was a DivX real-time encoder. It was defined at the macro-architecture level, annotated with cycle-accurate timing information, while all the interconnect networks were at TLM cycle-accurate level. The interconnect networks used in this paper were: the Distributed Memory Server (DMS), the AMBA AHB bus and the Octagon NoC. The experiments proved the flexibility of our approach in exploring the design space for an efficient interconnect in terms of total execution times, average throughput and latency.


### Acknowledgement

This work was supported by Medea+ Project Lomosa+ 2A708.


### References


[1] L.Benini, G.de Micheli “Networks On Chips: A New SoC Paradigm”, IEEE Computer, 2002.

[2] W. Cesario, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A.A. Jerraya, M. Diaz-Nava, "Component-Based Design Approach for Multicore SoCs", DAC, New Orleans, USA, 2002.

[3] M. Coppola, S. Curaba, M. Grammatikakis, G. Maruccia, F. Papariello, “On-Chip Communication Network: User Manual v1.0.1”, available online at http://occn.sourceforge.net/occn_user_manual.html

[4] F. Karim, A. Nguyen, S. Dey, and R. Rao. “On-chip communication architecture for OC-768 network processors”, Proceedings of Design Automation Conference, Las Vegas, NV, June 2001, pp. 678-683.

[5] S.-I. Han, A. Baghdadi, M. Bonaciu, S.-I. Chae, A.A. Jerraya, "An Efficient Scalable and Flexible Data Transfer Architecture for Multiprocessor SoC with Massive Distributed Memory”, DAC, San Diego, USA, June 2004.

[6] S. Mahadevan, F. Angiolini, M. Storgaard, R. G. Olsen, J. Sparsø, J. Madsen, “A Network Traffic Generator Model for Fast Network-on-Chip Simulation”, DATE, Munich, Germany, 2005, pp. 780-785 Vol. 2

[7] T. Salminen and J.-P. Soininen, ”Evaluating application mapping using network simulation.”, SOC2003, Tampere, Finland, November 2003.

[8] AMBA Specification (Rev 2.0), ARM Limited 1999, available online at: http://www.arm.com/products/solutions/AMBA_Spec.html .

[9] P. Paulin, C. Pilkington, E. Bensoudane. "Network Processing Challenges and an Experimental NPU Platform", DATE 2003, Designers' Forum, p. 64.

[10] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, R. Zafalon, Analyzing On-Chip Communication in a MPSoC Environment, DATE , Paris, France, 2004, pp. 752-757 Vol. 2

[11] S. Pestana, E. Rijpkema, A. Radulescu, K. Goossens, O. P. Gangwal, “Cost-Performance Trade-Offs in Networks on Chip: A Simulation-Based Approach”, DATE 2004: 764-769.

[12] K.Lahiri, A.Raghunathan, S.Dey, "System-Level Performance Analysis for Designing On-Chip Communication Architectures", IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, vol.20, no.6, pp.768-783, June 2001.