

Open-Scale: A Scalable, Open-Source NOC-based MPSoC for Design Space Exploration

Remi Busseuil, Lyonel Barthe, Gabriel Marchesan Almeida, Luciano Ost, Florent Bruguier, Gilles Sassatelli, Pascal Benoit, Michel Robert, Lionel Torres

▶ **To cite this version:**

Remi Busseuil, Lyonel Barthe, Gabriel Marchesan Almeida, Luciano Ost, Florent Bruguier, et al.. Open-Scale: A Scalable, Open-Source NOC-based MPSoC for Design Space Exploration. ReConFig 2011 - International Conference on Reconfigurable Computing and FPGAs, Nov 2011, Cancun, Mexico. pp.357-362, 10.1109/ReConFig.2011.66 . hal-01139181

HAL Id: hal-01139181
https://hal.science/hal-01139181

Submitted on 3 Apr 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

OPEN-SCALE: A SCALABLE, OPEN-SOURCE NOC-BASED MPSOC FOR DESIGN SPACE EXPLORATION

Remi Busseuil, Lyonel Barthe, Gabriel Marchesan Almeida, Luciano Ost, Florent Bruguier, Gilles Sassatelli, Pascal Benoit, Michel Robert, and Lionel Torres

LIRMM – 161 rue Ada, Cedex 05 - 34095 Montpellier, France

{remi.busseuil, lyonel.barthe, marchesan, ost, bruguier, sassatelli, pascal.benoit, michel.robert, lionel.torres}@lirmm.fr

Abstract— **As complexity of embedded system increases, configurable hardware is becoming more attractive because it provides a fast and efficient basis for design development. As a consequence, one of the most promising embedded architecture consists in the replication of Processing Elements (PEs) connected through a Network-on-Chip (NoC). Such architectures provide computation parallelism, scalability, and reduced design time thanks to reusability. This paper describes the development of a scalable, distributed memory, open-source NoC-based platform called Open-Scale and its implementation into FPGA devices. The main objective of this platform is to provide a complete framework for research development on NoC-based distributed memory MPSoCs.**

Keywords: MPSoCs, RTOS, NoC

I. INTRODUCTION

The increasing complexity of application and higher performance demand make Multiprocessors System-on-Chip (MPSoCs) one valuable alternative for dealing with nowadays embedded requirements, due to their power efficiency and capability to increase system performance by using multiple processing elements (PEs).

Traditional communication infrastructure, like shared busses, will not be able to support the amount of communication required by such MPSoCs. In this direction NoC architectures are well-known solutions due to their scalability and power efficiency. NoC-based MPSoCs have gathered much attention from industry [1][2][3]. However, to the best of our knowledge, synthesizable NoC-based MPSoC architectures with Real-Time Operating System (RTOS) support are not available under public domain.

In this context, this paper describes the development of *Open-Scale*, i.e. an open-source RTL NoC-based MPSoC that executes a preemptive RTOS. MPSoC design and implementation is a complex and a time-consuming process. Further, the need for rapid prototyping and software/hardware co-design validation make the use of Open-Scale particularly attractive for detailed design space exploration of NoC-based MPSoCs, since it was totally validated in FGPA. Furthermore, Open-Scale provides a set of functions and services that can be used or even extended to allow different performance analysis (e.g. impact of using task migration [4]). The main adopted strategies during the development phases and the impacts of those decisions in the overall system performance are described and evaluated, as well. Performance results using several benchmarks are also provided according to available components that can be easily chosen by the system designer.

The remaining of this paper is organized as follows: Section 2 provides an overview and trends in the field. In Section 3, hardware components of the proposed NoC-based MPSoC architecture are described. Section 4 discusses the available features of the RTOS such as mutexes, management of FIFOs, communication stack, etc. Finally, Section 5 draws conclusions and points out directions for future work.

II. RELATED WORK

NoC-based MPSoCs approaches are studied at different levels of abstraction that vary in flexibility, accuracy, and simulation speed. Most of them employ high-level models (e.g. analytical models) for proposing new mechanisms, such as dynamic mapping and task migration. However, such high-level models have to produce accurate results, allowing early design decisions. Thus, it is fundamental to adjust/calibrate these model parameters (e.g. task migration time), by using a reference platform (normally a RTL implementation).

Due to the number of modeled aspects, these reference platforms are provided to validate and to explore specific aspects that can contribute to the efficiency of the system. For instance, Joven et al. [5] propose the xENOC, which is a framework that allows exploring the design of NoC-based MPSoC architectures. The framework comprises a tool, called NoCWizard, for RTL Verilog NoCs generation, which uses XML file as input. Such files provide the description of each system component (e.g. NoC router, PE) and application-platform mapping. In addition, embedded message passing interface (eMPI) is adopted to support parallel task communication.

In turn, in [6] a rapid prototyping MPSoC based on model-drive approach called LAVA, is presented. The architecture provides a number of open-source IP cores such as PEs (Plasma [7], MB-Lite [8], ZPU [9]), a UART, a timer, and a CAN controller connected to a Wishbone bus [10]. Similar to the Joven's approach [5], a XML file is used together with VHDL for describing the architecture. However, no synthesizable NoC is provided.

Tian [11] et al. present a NoC-based MPSoC design that comprises 16 MicroBlazes, employed as master PEs and 16 SSRAMs, which are used as slave. PEs are connected by two NoCs. One is used for data exchange, while a OCP-based

NoC architecture is used to establish a synchronization between such PEs.

The HeMPS, a homogeneous NoC-based MPSoC platform, is described in [12]. The HeMPS architecture comprises MIPS-like processor (Plasma [7]), a local memory (RAM), a DMA controller and a NoC HERMES-based Network Interface (NI). This platform employs one Master-PE that is responsible to manage task mapping and system debug. In turn, Slave-PEs are responsible to execute application tasks. HeMPS has a preemptive microkernel that provides communication primitives such as *WritePipe()* and *ReadPipe()*, which are used to implement message passing communication.

HeMPS platform is quite similar to the NoC-based MPSoC proposed in this paper. The main differences between both hardware architectures are: (*i*) HeMPS employs the Plasma processor, while Open-Scale uses the Microblaze; (*ii*) HeMPS has a DMA controller, which is particular advantage for video streaming applications, since large data blocks can be transferred (cyclic operations) without CPU overhead. In terms of software, the RTOS supported in Open-Scale provides a considerable number of services (see Table I), which are necessary for proposing new adaptive mechanisms.

III. Open-Scale – Hardware Description

A. System Overview

The Open-Scale employs a distributed memory/message passing approach and its main component is Network Processing Unit (NPU) [13]. Figure 1 provides a general overview of the NPU internal architecture, which includes: (*i*) a SecretBlaze CPU, (*ii*) an embedded RAM, (*iii*) an interrupt controller, (*iv*) a timer, a UART, (*v*) a NI, (*vi*) a HERMES-based router [14] and a (*vii*) Wishbone bus [10].

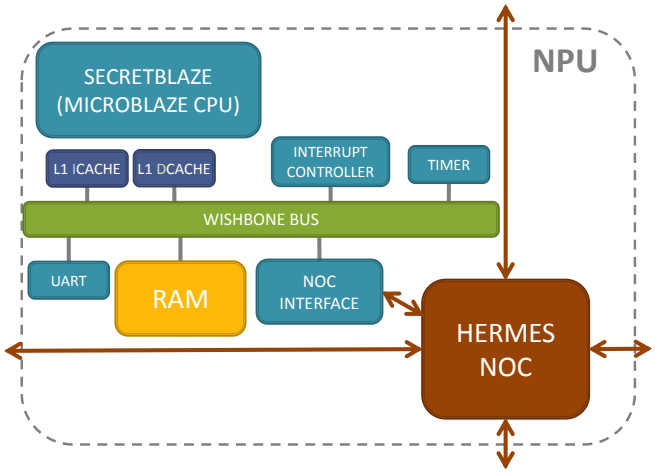


Figure 1 - NPU architecture overview.

Open-Scale scalability is achieved by replicating as many NPUs as required. The NPU components are following described.

B. NPU components description

The SecretBlaze CPU [15] is a configurable open-source RISC soft-core processor developed by our research group. It implements the MicroBlaze instruction set architecture with a five-stage pipeline. Most instructions execute in a single clock cycle, achieving optimized performance for FPGA implementations. The development of the processor was mainly conducted keeping a modular approach to ensure reliability, efficiency across the whole design, while providing better design reuse opportunities in various research and educational projects.

The flexibility is one of the driving aspects of the Secret-Blaze design. On the one hand, the core provides several optional logical and integer instructions such as multiplication, division, and pattern operations, which balances computing performance and area cost to meet embedded system requirements. On the other hand, the SecretBlaze is a MMU less processor with a simplified memory sub-system that offers optional configurable data and instruction caches, implementing the pipelined Wishbone protocol for external memory interfaces [10]. However, no global cache coherency is provided by the Open-Scale platform.

The SecretBlaze uses an embedded RAM as local memory. The interrupt controller can handle up to 8 interrupts with masking, arming, and polling mechanisms. The timer is a 32-bit counter that can generate an interrupt according to a configurable time window. Besides, a UART interface, which is adjustable via software, can be used for debugging purposes. These components are interconnected by a standard open-source Wishbone bus [10]. The communication between the NPU and the NoC router is implemented in the NI, which defines HW/SW integration (e.g. bus width, bandwidth), as well as packing/unpacking the packets from/to the NoC.

The adopted NoC router uses XY router based on the HERMES infrastructure [14]. The NoC employs packet switching of wormhole type: the incoming and outgoing ports used to route a packet are locked during the entire packet transfer. The routing algorithm is an XY engine that allows deterministic routing. Each router processes one incoming FIFO per port. The size of FIFOs can be tuned for balancing area and performance.

C. Open-Scale area evaluation

Due to the numerous parameters that can be tuned on the platform, the evaluation of the implementation will not be given exhaustively. Indeed, both instruction and data cache sizes can be adjusted, as well as the size of NoC FIFOs. In addition, the processor can optionally include hardware multiplier, divider and/or barrel shifter.

To highlight the scalability of the proposed system, the size of the whole MPSoC has been measured as a function of the number of NPUs. The development platform is based on a Virtex 5 LX 110T FPGA, using optional multiplier, divider, and barrel shifter instructions [15]. These instructions are present into the Integer Unit (IU) of the SecretBlaze instance. The cache size was set to 8KB for both instruction and data. NoC FIFOs was defined to 256 32- bits

words. Furthermore, each NPU possesses an internal RAM of 64KB. The area occupation results are given in Figure 3. The area occupation is almost linear, with an increasing number of slices with order of magnitude of 2.3 and number of 6-input Look-Up Tables (LUTs) of 2.12 in average when the number of cores doubles.

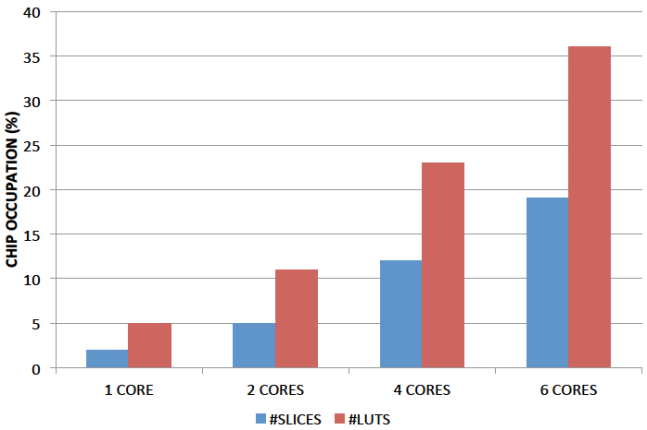


Figure 2 - Open-Scale hardware area occupation.

IV. Open-Scale – Software Description

In order to keep the distributed memory structure and to preserve the scalability of the system, each NPU operates asynchronously and uses a MPI-like API for message passing communication. Global decisions are performed in a distributed fashion and no global shared-memory is used.

Due to the distributed memory characteristic of Open-Scale, applications are described using a Kahn Process Network (KPN) formalism [16], which allows parallel computation of the tasks. The KPN computation model allows deterministic behavior of the application in an asynchronous way. Furthermore, tasks placement can be optimized depending on the user requirements (e.g. computation time, energy consumption).

MPI provides a comprehensive number of primitives that relate to general-purpose distributed computing; a number of works have devised lightweight implementations supporting only a subset of MPI mechanisms for embedded processors and systems. This makes sense since KPN formalism offers a sufficient support that requires only blocking read operations, which are necessary to model, for instance, data flow (e.g. video and audio) applications. Some MPI implementations are layered, and advanced communication synchronization primitives (e.g. collective) found in the upper layers make use of the simple point-to-point primitives such as *MPI Send()* and *MPI Receive()*. This enables using these collective mechanisms in an application-specific basis in case they prove necessary.

Each NPU runs a tiny preemptive RTOS that was further extended from Steve Roads Plasma RTOS [7]. Such RTOS structure is depicted in Figure 3, which comprises 4 categories: (*i*) basic RTOS services (e.g. function calls), (*ii*) communication, (*iii*) drivers, and (*iv*) libraries.

Furthermore, the RTOS provides multi-threaded preemptive execution, using a scheduler based on thread priorities that is executed periodically according to a fixed *timeslot*, which can be defined by the user. A round robin scheduling algorithm is executed when all tasks have the same priority.

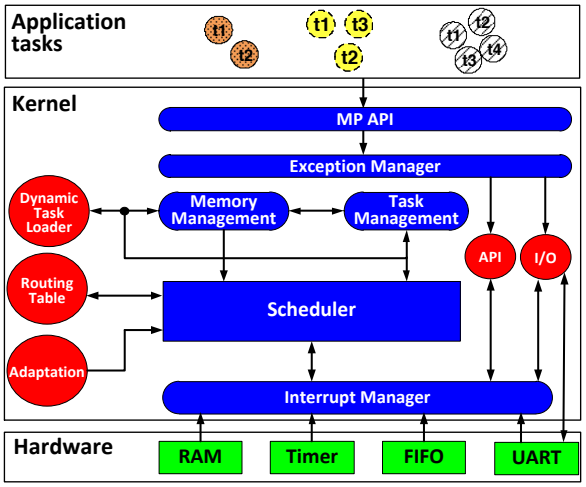


Figure 3 - Structure of Open-Scale RTOS.

The RTOS allows the use of semaphores and mutexes, communication between local and remote tasks, and dynamic memory allocation, as well. Further, it also provides the standard C library together with a compact math library that allows floating point operations as well as software multiplications/divisions. Timer and UART drivers are also available in the platform.

A. Open-Scale RTOS development

The Open-Scale RTOS was implemented in such a way that users can easily choose which features are needed in their implementation in order to either save memory or meet performance requirements. In this scenario, new services and features were implemented in order to be compliant with the SecretBlaze architecture, while providing more efficiency in terms of management and QoS support (Table I).

Table I summarizes some services that were included in Open-Scale RTOS. As mentioned before, one of the goals of Open-Scale is to explore adaptive mechanisms (e.g. dynamic frequency scaling, task migration). For instance, to enable dynamic load balancing, the system has to be able to migrate running tasks from one to another NPU. For that reason, a run-time loading mechanism was included to allow compiled separately applications from the RTOS being dynamically uploaded at run-time.

Besides, a preemptive round-robin scheduler based on thread credits has been implemented, avoiding task execution starvation. Intra/extra-NPU communications were extended to provide more flexibility and system performance. For example, the RAW protocol was implemented in order to achieve better performance when compared to TCP/UDP (as shown in Figure 6). Further, three online system-monitoring mechanisms were included: (*i*) CPU utilization, (*ii*) FIFO filling, and (*iii*) CPU frequency. Once monitored information is provided, online decisions

can be taken by decision-making mechanisms, like a run-time control system used for regulating NPU frequency that were added [18]. Furthermore, an API, new drivers (e.g. UART, frequency scaling and timer), as well as dynamic mapping heuristics were included to provide more design space exploration alternatives

TABLE I. SERVICES INCLUDED INTO THE OPEN-SCALE RTOS

Steve Roads Plasma RTOS	Newly Supported in Open-Scale
1- generation of a single object file; 2- preemptive round-robin sheduler based on thread priorities; 3- intra-NPU communication based on local FIFOs; 4- Extra-NPU communication (e.g TCP protocol) through ethernet; 5- interrupt and exception handling; 6- dynamic memory allocation and deallocation; 7- queues, semaphores, mutexes.	1- run-time dynamic applications loading; 2- preemptive round-robin scheduler based on thread credits; 3- intra-NPU communication based on messages exchanged by software FIFOs; 4- Extra-NPU communication (RAW protocol was included), as well as *MPI_Send* and *MPI_Receive*; 5- run-time monitoring support; 6- decision-making mechanisms; 7- a run-time control system used for regulating NPU frequency; 8- API with new primitives, etc; 9 – development of new drives; 10- dynamic mapping heuristics.

B. Memory Management

As explained in Section III.B, the Open-Scale hardware architecture does not include a memory management unit (MMU). This design choice is crucial in the development of the RTOS, once memory management in software must be carefully handled. Two major issues have been faced concerning memory management: the heap control and the dynamic task loading.

The first strategy to deal with memory management is the use of paging. In such a scenario, each task has its own virtual memory, and address translations are performed at run-time in order to access physical memory. However, a Translation Logic Buffer (TLB) provided by the MMU usually executes this process. A software run-time address translation has a heavy cost in term of computational time as translation has to be performed for each memory access. For this reason we have not adopted this solution in our design.

The second way of managing memory is using dynamic memory allocation/deallocation: whenever a new space is required, the RTOS searches for a contiguous available space in the memory large enough for storing the information. Once the information is not required any more, the memory space is deallocated. This approach creates memory fragmentation, but leads to smaller computation cost compared to software paging. Moreover, some techniques exist to defragment the memory efficiently [17].

One possible solution for enabling the loading of tasks without MMU relies on a feature that is partly supported by the GCC compiler that enables to emit relocatable code (PIC: Position Independent Code). This feature, generally used for shared libraries, generates only relative jumps and accesses data locations and functions using a Global Offset Table (GOT), which is embedded into a generated ELF file. A specific post-processing tool, which operates on this format, was used for reconstructing a completely relocatable executable. Experiments show that both memory and performance overheads remain under 5% for this solution which is clearly acceptable [13].

C. Communication Services

Open-Scale supports two types of communication: (*i*) *Intra-NPU communication*, and (*ii*) *Inter-NPU communication*.

Intra-NPU communications are handled through software FIFOs. Whenever a packet is sent to a specific task, an exception is raised to notify the receiving task that there is an incoming data. The receiving task is then scheduled for execution in order to process the received packet. In turn, inter-NPU communications are more difficult to handle due to the globally asynchronous network management that is required. A communication stack, based on classical TCP/UDP and IP standard has been adopted in [19]. Each service (e.g. task communication, task loading) is linked to a particular port. For each incoming packet of the NPU, the appropriated service associated to the destination port is executed. Concerning reliability issues, the communication stack also provides optional re-routing and CRC error checking.

Figure 4 illustrates the four layers of the RTOS communication protocol. At the physical level, the packet reception is handled by both interrupt and polling methods. The interrupt occurs when the number of elements inside the incoming FIFO reaches a given threshold, while the polling procedure is triggered at fixed timeslot from a timer interrupt.

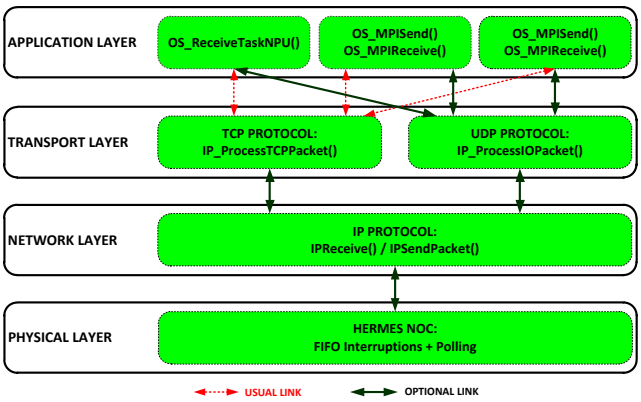


Figure 4 - RTOS communication stack protocol.

D. Open-Scale RTOS evaluation

1) Memory Occupation

The compiled operating system is 57KB big. Users can easily choose on using or not particular features in order to reduce the RTOS size. If communication between remote tasks is not required, the RTOS size can be reduced down to 47KB. Table II shows the resulting RTOS size considering different hardware/software optimizations.

TABLE II. RTOS SIZE

	with HD optimization	without HD optimization
with communication layer	57.4 KB	65.4 KB
without communitcation layer	47.6 KB	55.4 KB

2) Time Performance

To enhance the overall computing performance, each NPU must have the capability to process incoming data regarding the available bandwidth of the NoC. Although this assumption is completely dependent of the application's nature with its mapping onto the platform, two kinds of results can be expected when evaluating the performance. The first is the service performance provided by the RTOS while the second concerns the computation time of well-known applications. The performance of basic RTOS operations has been measured and results are shown in Table III.

TABLE III. RTOS OPERATION PERFORMANCE

	Start	Rescheduling	System Latency
Time (clock cycles)	237,845	487 - 1307	305

The RTOS takes 237,000 clock cycles to boot (i.e. about 4.7ms at 50MHz). This time is required to initialize all services including the communication stack. The scheduler operation that consists on searching the task to be executed has been optimized. Hence, the rescheduling of a running task takes less than 500 clock cycles, while scheduling a different task takes around 1300 clock cycles since context switching is necessary. The system call latency is measured by the time that the RTOS service takes to handle an application request.

The execution performance has been measured based on three applications: (*i*) an advanced encryption standard application (AES), (*ii*) a classical integer benchmark application (Dhrystone 2.1 [20]), and (*iii*) a video decoder (MJPEG). Processing time for these applications are shown in Figure 5.

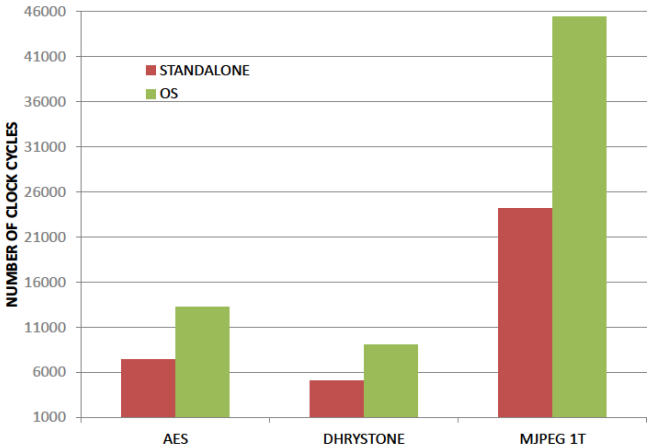


Figure 5 - Execution time for 3 applications for a standalone and Open-Scale RTOS.

The standalone method shows results of an optimized execution without RTOS services and cache misses. RTOS execution presents a performance decrease of about 1.8 times. This can be mainly explained by the fact that there are cache misses, while RTOS services consist in less than 3% of the execution time. Furthermore, memory allocation (Section IV.B), leads to misaligned applications that produce several cache misses.

3) Communication

To measure the communication performance, a simple application consisting of two tasks (sender/receiver) that exchange packets of 200 bytes continuously. Figure 6 depicts the communication delay (in clock cycles) between two send/receive calls considering three protocols: (*i*) RAW, (*ii*) UDP, and (*iii*) TCP. Note that the results are classified in four categories: (*i*) SEND-START that represents the number of necessary clock cycles to open the socket and to send the first packet, (*ii*) SEND that comprises the remaining packets sending, (*iii*) RECEIVE-START that is used to defined the socket opening and the receiving of the first packet, and (*iv*) RECEIVE that comprises the time of the remaining received packets.

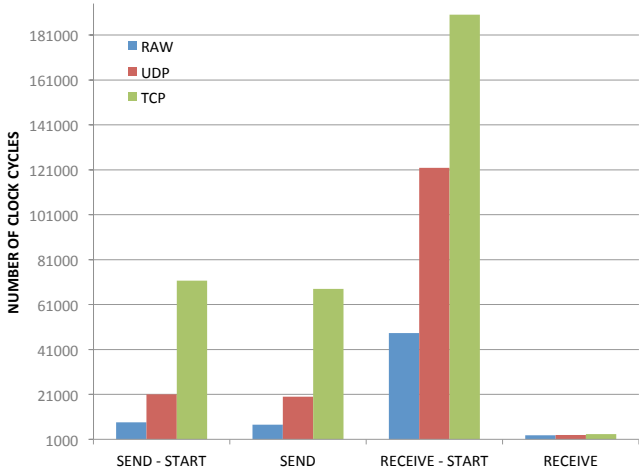


Figure 6 - Communication delay between send and receive calls regarding three protocols.

The RAW protocol consists in bypassing the services provided by the communication stack in order to produce the highest bandwidth. For that reason, the RAW protocol provides lower communication latency results for the analyzed communication. For instance, regarding the SEND-START communication scenario is observed a reduction of 59% and 88%, respectively, when comparing the RAW to the UDP and the TCP protocols. In turn, the TCP protocol achieves the highest latency in the four send/receive communication scenarios.

In general, sending functions are more time consuming than receiving functions, except for the beginning of transactions. This phenomenon can be explained by the latency caused by the FIFOs: receiving functions must be in blocking-mode waiting for the incoming packet before receiving and processing it. In this context, the RAW protocol achieves the highest bandwidth with 1.3MB/s at

50MHz while UDP and TCP protocols achieve 476KB/s and 147KB/s respectively.

However, it is important to mention that the RAW protocol does not provide any reliability once there is no checksum or CRC implementation. On the other hand, the UDP/TCP communication protocols provide the basic services such as opening/closing sockets, handshaking, and CRC error detection.

V. CONCLUSION AND FUTURE WORK

This work described a complete Open-Source framework for academic research and development of NoC-based MPSoC. Open-Scale join different features inherent to the state-of-the-art in MPSoCs design, which include: (*i*) customized RTOS implementation, covering important aspects like inter-task communication primitives, dynamic loader; (*ii*) NoC-based platform development; and (*iii*) hardware-software integration. These features were described and their design decisions were made targeting a scalable distributed memory MPSoC platform.

Open-Scale brings important advantages in the MPSoC design field like run-time monitoring that allows optimized decisions distribution for different NPU at software level. Therefore, making it possible to explore the use of dynamic mechanisms (e.g. dynamic task mapping not explored in this work), while assessing performance figures using different benchmark applications together with a RTOS.

Future works on this platform will include load balancing through task migration or frequency scaling, memory management and global cache-coherency, security development and reliability issues. More information about Open-Scale platform, as well as the open-source is available at: http://www2.lirmm.fr/openscale

REFERENCES

[1] Tilera processors. [Online]. Available at: http://www.tilera. com/

[2] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, "Synergistic processing in cell's multicore architecture". IEEE Micro, vol. 26(2), 2006, pp. 10 – 24.

[3] J. Rattner, "Intel single-chip cloud computer," 2009. [On- line]. Available at: http://techresearch.intel.com/ProjectDetails. aspx?Id=1

[4] G. Marchesan Almeida, et al. "Evaluating the impact of task migration in multi-processor systems-on-chip". In: Symposium on Integrated Circuits and System Design (SBCCI'10), 2010, pp. 73–78.

[5] J. Joven, et al. "xENOC – An eXperimental Network-on-Chip Enviroment for Parallel Distributed Computing on NoC-based MPSoC Archtectures". In: Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'08), 2008, pp. 141-148.

[6] M.Meier, M.Engel, M.Steinkamp, and O.Spinczyk, "Lava: An open platform for rapid prototyping of mpsocs". In : Field Programmable Logic and Applications Conference (FPL'10), 2010, pp. 452 – 457.

[7] S. Rhoads, "Plasma - most mips i(tm)" [Online]. Available at: http://www.opencores.org/project,plasma

[8] T. Kranenburg and R. van Leuken, "Mb-lite: A robust, light- weight soft-core implementation of the microblaze architec- ture". In : Design, Automation Test in Europe Conference Exhibition (DATE'10), 2010, pp. 997 –1000.

[9] Ø. Harboe, "Zylin – zcpu," 411 W Miner St., West Chester, PA, 19382, 2010. [Online]. Available at: http: //opensource.zylin.com/zpu.htm

[10] O. O. Richard Herveille, "Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores", Revision B.4, OpenCores, 2010. [Online]. Available at: http://www. opencores.org/

[11] G. Tian, G Hammami, O. "Performance measurements of synchronization mechanisms on 16PE NOC based multi-core with dedicated synchronization and data NOC". In: International Conference on Electronics, Circuits, and Systems (ICECS'09), 2009, pp. 988 – 991.

[12] Carara, E.; et al. "HeMPS - A Framework for NoC-Based MPSoC Generation". In: IEEE International Symposium on Circuts and Systems (ISCAS'09), 2009, pp. 1345 – 1348.

[13] G. M. Almeida, G. Sassatelli, P. Benoit, N. Saint-Jean, S. Varyani, L. Torres, and M. Robert, "An adaptive message passing mpsoc framework". International Journal of Reconfigurable Computing, vol. 2009, p. 20.

[14] F. Moraes, N. Calazans, A. Mello, L. Moller, and L. Ost, "Hermes: an infrastructure for low area overhead packet- switching networks on chip". Integration VLSI Journal, vol. 38(1), 2004, pp. 69–93.

[15] L. Barthe, L. V. Cargnini, P. Benoit, and L. Torres, "The secretblaze: A configurable and cost-effective open-source soft-core processor". In : IEEE International Parallel & Distributed Processing Symposium, Workshops and Phd Forum IPDPS/RAW 2011. [Online]. Available: http://www.ipdps.org/

[16] G. Kahn and D.B. MacQueen. "Coroutines and networks of parallel programming". In IFIP Congress, 1977, pp. 993–998.

[17] J. E. et al., "File system defragmentation technique via write allocation". U.S. Patent 6 978 283, 2005.

[18] Almeida, G.; Busseuil, R.; Ost, L.; Bruguier, F.; Sassatelli, G.; Benoit, P.; Torres, L.; Robert, M.; , "PI and PID Regulation Approaches for Performance-Constrained Adaptive Multiprocessor System-on-Chip" Embedded Systems Letters, IEEE , vol. 99(1), 2011, pp.1.

[19] R. Busseuil, G. M. Almeida, S. Varyani, P. Benoit, and G. Sassatelli, "A self-adaptive communication protocol allowing fine tuning between flexibility and performance in homogeneous mpsoc systems". In : Reconfigurable Communication-centric Systems on Chip (ReCoSoC'10), 2010.

[20] R. P. Weicker, "Dhrystone benchmark: rationale for version 2 and measurement rules". In ACM SIGPLAN Notices, vol. 23 (8), 1988, pp. 49 – 62.