HeMPS-S: A Homogeneous NoC-Based MPSoCs Framework Prototyped in FPGAs

Eduardo Weber Wächter, Adelcio Biazi, Fernando G. Moraes
PUCRS – FACIN – Av. Ipiranga 6681 – Porto Alegre – 90619-900 – Brazil
eduardo.wachter@acad.pucrs.br, adelcio.biazi@acad.pucrs.br, fernando.moraes@pucrs.br

***Abstract*—Current chip transistor density enables the design of multiprocessor systems-on-chip (MPSoCs). MPSoCs are an alternative to create complex computational systems because they reduce the cost, area, power dissipation and design time per chip. Due to their complexity and huge design space to explore for such systems, CAD tools and frameworks to customize MPSoCs are mandatory. The main goal of this paper is to present an open source platform for MPSoC development, named HeMPS Station (HeMPS-S). HeMPS-S is derived from the MPSoC HeMPS. HeMPS-S, in its present state, includes the platform (NoC, processors, DMA, NI), embedded software (microkernel and applications) and a dedicated CAD tool to generate the required binaries and perform debugging. Experiments show the execution of a real application running in HeMPS-S.** *(Abstract)*

Keywords- NoC-based MPSoC; Rapid Prototyping; MPSoC; FPGAs (key words)

I. INTRODUCTION

The increase in transistor density enables the design of complete systems on a single chip. These systems, named system-on-chip (SoCs), can implement most or all of the functions of a complete electronic system. Present SoCs may contain hundreds of millions of transistors, including processing elements (PEs), memory blocks, dedicated IPs, and a communication infrastructure. The market needs cheap products with good performance, limited power budget and short time-to-market. One way to cope with such conflicting requirements is to use several programmable processors in the same SoC. A system with several processors in the same integrated circuit is named MPSoC.

The power budget is the main reason driving MPSoCs. While the transistor number per chip is increasing, the clock frequency and the power dissipation does not grown at the same pace. If the performance of a given application requires one processor running at 10 GHz, the implementation of such processor would be unfeasible due to power dissipation. One alternative would be to use an MPSoC with ten 1 or 2 GHz processors.

An important gap is observed in the MPSoC literature (Section II): a prototyped open-source state-of-the-art MPSoC, enabling design space exploration at several levels:

- NoC: explore different routing algorithms, topologies, priorities schemes, etc;
- NI: explore at the network interface (NI) level different service levels to guarantee Quality of Service (QoS), as injection and admission control;
- Operating system (OS): evaluate different scheduling polices, Dynamic Voltage and Frequency Scaling (DVFS) monitoring parameters, distributed memory architectures, task migration, for instance.
- Power dissipation: evaluate different types of processors or NoCs in terms of energy.
- PEs: evaluate different processor architectures according to the applications requirements.
- Inter-board communication: Hardware modules that enable the coordination of the MPSoC from a host computer, for example.

This is the *main goal* and the *contribution* of the paper: to present the design and the prototyping of an open MPSoC framework, enabling to validate applications in a NoC-based MPSoC. The hardware part of the framework was prototyped in a Xilinx FPGA. Being open-source, it may be used to explore different aspects of the MPSoC design. Additionally, the hardware designer can modify the framework and implement a distributed shared memory system, for example. The software developer, on the other hand, can adapt his applications with distributed tasks, to run on the MPSoC.

This work is organized as follows. Section II reviews the state of the art of MPSoC organizations. Section III presents the HeMPS-S prototype in a FPGA platform. Section IV presents the HeMPS-S framework, enabling to validate distributed applications. Finally, Section V concludes this paper and presents directions for future works.

II. MPSOC ORGANIZATIONS

Van Berkel [1] argues that power constraints lead to heterogeneous multi-core architectures. RISC processors support less than 1 algorithmic operation per clock cycle. Thus, to achieve 100 GOPS it would be necessary a single core running at 100GHz or 1000 cores running at 100MHz. Both approaches are unfeasible with present technologies. A possible solution is to employ a heterogeneous architecture, based on specialized programmable cores, and (configurable) function-specific hardware accelerators. As presented below, most MPSoC designs are homogeneous, since the design is simpler and they present higher flexibility.

Jalier et al. [2] present three MPSoCs implementations: heterogeneous MAGALI, homogeneous GENEPY v0 and homogeneous GENEPY v1. The heterogeneous MAGALI architecture contains two different cores: a DSP unit and an MMC (Microprogrammable Memory Controller) unit, for

intensive data manipulation involving synchronization, buffering, duplication and reordering. The processing element of the homo*GENE*ous *P*rocessor arra*Y* (GENEPY), named *SMEP v0*, integrates a Smart Memory Engine (SME) and a processing cluster with two DSPs. All PEs are interconnected through a Network-on-Chip. GENEPY v0 architecture contains a host processor and an array of *SMEP v0* PEs. The global control through a host processor limits the scalability of this platform. *GENEPY v1* is a distributed architecture with *SMEP v1* modules, without a host processor. *SMEP v1*is a *SMEP v0* PE with a CCC (Configuration Communication Controller) unit. All MPSoCs were implemented in a 65 nm technology. The homogeneous implementations showed superior performance compared to the heterogeneous MAGALI platform. The homogeneous GENEPY v1 platform is about 14% smaller in terms of silicon area. For a 4G standard, the comparison has shown that they achieve a performance speed-up around 3% (v0 and v1), with power savings of 10% (v1) and 18% (v2). Authors mention that heterogeneous architectures are well known solutions for wireless standards. However, they argue that these architectures have limited flexibility, and highlighted that homogeneous MPSoC approaches make sense for future Mobile Terminals.

Zhang et al. [3] present a heterogeneous MPSoC targeting Altera FPGAs. The system is composed by: (*i*) four Nios II soft cores; (*ii*) one ARM hard core, (*iii*) a memory subsystem, (*iv*) an On-Chip System Bus Network. The ARM core is the central controller of the system and the four Nios II cores provide an environment for data processing. Both cores share access to a common address space, which includes main memory (SDRAM) and the communication subsystem. The ARM core subsystem is composed of a single ARM core with AMBA AHB bus interface to communicate directly with a local memory. Each Nios II core has an instruction cache to improve performance and a local memory. Each core has full access to the shared memory and the System Bus Network. The On-chip System Network is designed as a hierarchical bus architecture. This architecture allows different subsystems associated with their own bus layers, and can operate in parallel if there are no resource conflicts. The system was implemented in an EP2S180 device. This device has 143,520 ALUTs and 9 Mbits of embedded RAM. The system running at 60 MHz uses 13% of the total area in ALUTs, while 39% of the total on-chip memory is required, due to the instruction cache of each Nios II core.

Hammami et al. [4] presents an MPSoC, targeting multi-FPGA platform prototyping. The target architecture is an MPSoC with 672 PEs, connected to 7x8 mesh NoC (Figure 1). At each NoC router it is connected a PE *cluster*. As all processor elements are equal, the system is a homogeneous MPSoC. Each cluster is in fact a NoC-based MPSoC, containing: *(i)* 12 Xilinx MicroBlaze processors; *(ii)* 8 Xilinx Coregen Memories; *(iii)* 4 NoC routers; *(iv)* an Interchip Communication Module (ICM). The last two IPs are provided by the VHDL Arteris Danube library. The cluster is implemented in one board and contains four routers interconnected through a mesh topology. Each router connects three processors and two SRAM on-chip memories. The implementation of the 672-processor MPSoC requires multi-FPGA implementation. The ICM is responsible to connect the multi-FPGA system. Each FPGA contains 2 ICMs that connects the FPGAs in a mesh topology. The emulation of the whole system is based in a 56 FPGA platform called ZeBu-XXL. The Authors also presented a Design Flow to map the design onto several FPGAs. As the used Microblaze PE is OCP compliant, the authors endorse that this processor can be replaced by another OCP compliant processor while leaving the overall design identical. As a criticism, nowadays there are no applications/benchmarks enabling to evaluate the actual performance of the proposed architectures.

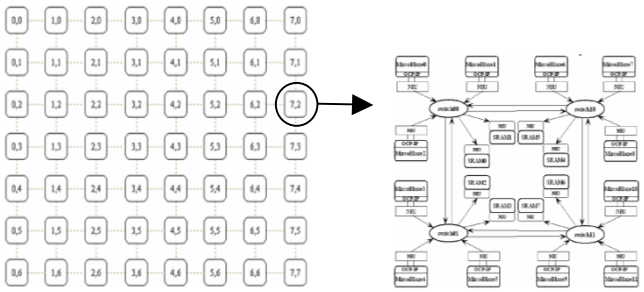


Figure 1 – Multi-FPGA MPSoC, in the detail one FPGA with 12 PEs.

Garzia et al [5] present the MPSoC design for software defined radio applications called Ninesilica. The architecture is composed of nine PEs, based on COFFEE RISC cores, communicating through a NoC. The multiprocessor system is based on Silicon Cafe template. The template provides a configurable VHDL model to create an MPSoC based on a fixed NoC infrastructure and a variable number of nodes. The Authors use a 3 x 3 mixed star-mesh network with the master in the center, connected to input/output ports. Additionally, each slave is directly connected to the master, creating point-to-point communications between each slave and the master. This MPSoC is targeted to ASIC and FPGA implementations. The ASIC implementation, using a 65nm library, used 630 Kgates, with a maximum frequency of 200 MHz. The FPGA implementation, targeting an AlteraStratix II FPGA, used 76,780 ALUTs and 50,482 Logic Regs (memory resources), with a maximum frequency of 75 MHz.

Limberg et al. [6] present a heterogeneous MPSoC, named Tomahawk, targeting signal processing for future mobile phone technology, such as 3GPP LTE and WiMAX. This platform contains: (*i*) two Tensilica DC212GP RISC processors; (*ii*) four SIMD processors (fixed-point DSPs) to handle parallel and vector processing, such as FFTs and DCTs; (*iii*) two scalar floating-point DSP processors to handle streaming data; (*iv*) hardware modules for specific functions, such as parity check code decoder ASIP, deblocking filter ASIP and an entropy decoder ASIC. All components on the chip are connected by two low latency, high bandwidth, crossbar-like master/slave NoC with 32 bit bus-width. The NoC performs static priority arbitration per slave and supports burst transfers of up to 63 data words. The Tomahawk chip was designed using a UMC 130 nm, 8 metal layer CMOS standard cell design flow. The 57M transistor chip occupies 10x10 mm^2 and runs at 175 MHz.

Examples of industrial MPSoCs are TILE-GX, with up 10 100 processing elements [7], and Intel's Teraflops Research Chip with 80 processing elements [8]. In common, they are

TABLE 1 – COMPARISON BETWEEN THE PRESENTED MPSOC ARCHITECTURES.

MPSoC	PE	Number of PEs	Architecture	Interconnection	Technology
MAGALI [2]	DSP and MMC	5	Heterogeneous	Mesh NoC	ASIC
GENEPY v0 [2]	SME and 2 DSPs	3	Homogeneous with a central PE controller		
GENEPY v1 [2]	SME, CCC and 2 DSPs	2	Homogeneous		
Zhang [3]	ARM or NIOS II	5 (4 NIOS and 1 ARM)	Heterogeneous,with a central PE controller (ARM)	Hierarchical Bus	FPGA
Hammami [4]	Microblaze	672, organized in an 8x7 clusters (each cluster with 12 PEs)	Homogeneous	Hierarchical mesh NoCs	FPGA
Garzia [5]	COFFEE	9	Homogeneous	Mixed Star-Mesh NoC	ASIC / FPGA
Limberg [6]	RISC, fixed-point DSP, floating-point DSP and HW IPs	11 (2 RISC, 4 fixed-point DSP, 2 floating-point DSP and 3 HW IP)	Heterogeneous	2 crossbar NoCs	ASIC
Tilera [7]	64-bit VLIW processors	16 to 100	Homogeneous	3 Mesh NoCs	ASIC
Vengal [8]	2 FPMAC (floating-point multiply ccumulators)	80	Homogeneous	Mesh NoC	ASIC
Proposed MPSoC	RISC (MIPS)	configurable size	Homogeneous, with a central PE controller	Mesh NoC	FPGA

ASIC implementations, homogeneous architectures, interconnecting the PEs through a mesh-NoC.

Table 1 compares the reviewed works. Some trends in MPSoC design are observed:

(*i*) small heterogeneity in terms of PEs (RISC+DSP), with a trend to homogeneous systems;
(*ii*) mesh-NoC as the communication infrastructure, due to its scalability and easiest physical implementation;
(*iii*) the number of PEs is increasing: 80 [8], 100 [7] and 672 [4];
(*iv*) typical benchmarks include cell phone standards and radio applications, due to the high computation rate and heterogeneous computing: signal processing (generally DSP) and data processing (generally dataflow).

The proposed MPSoC, HeMPS, follows these trends: homogeneous; NoC-based; parameterizable size (simulation with up to 64 PEs and FPGA prototype with 12 PEs); support to execute distributed applications described in C language.

III. HEMPS-STATION

HeMPS Station (HeMPS-S) is an environment for MPSoC evaluation of distributed embedded applications. The evaluation can by carried out in a FPGA-based platform, as well as by RTL simulation. This framework contains tools running on a host computer, a communication interface between the host and the MPSoC, enabling the evaluation at running time, and a monitoring system to capture the performance data.

HeMPS-S is composed by the HeMPS NoC-based MPSoC [9], the *ConMe* unit (a DDR2 memory controller), the *ComEt* unit (MAC Ethernet communication interface), and the Main Control Unit. The architecture is present in Figure 2.

A. HeMPS MPSoC [9]

The HeMPS MPSoC contains a parameterizable number of PEs interconnected by the HERMES NoC [10]. The used NoC has the following features: *(i)* wormhole packet-switched mesh topology; *(ii)* credit-based flow control; *(iii)* XY routing algorithm; *(iv)* 16-bit flit size; *(v)* 8-flits buffer depth. The flit size is half of the processor word, to reduce the NoC area overhead.

The processing elements are named Plasma-IP. Each Plasma-IP is composed by: *(i)* 32-bit RISC processor - PLASMA (MIPS-like); *(ii)* network interface; *(iii)* DMA; *(iv)* dual port BRAM memory, which stores the object code of the tasks and the kernel; *(v) access_repository* unit, (not presented in Figure 2) only present in the Plasma-IP Master, responsible to communicate the MPSoC with the external world.

The PlasmaIP-MS is responsible for managing system resources. The PlasmaIP-SLs are responsible for executing the application tasks. The PlasmaIP-MS have access to the external task repository and to the *ComEt* unit, through the *Main Control* unit. When HeMPS starts execution, the PlasmaIP-MS reads the required tasks from the task repository, and allocates all initial tasks to the Plasma-IP SLs. During execution, tasks can be dynamically loaded from the task repository to slave processors on demand.

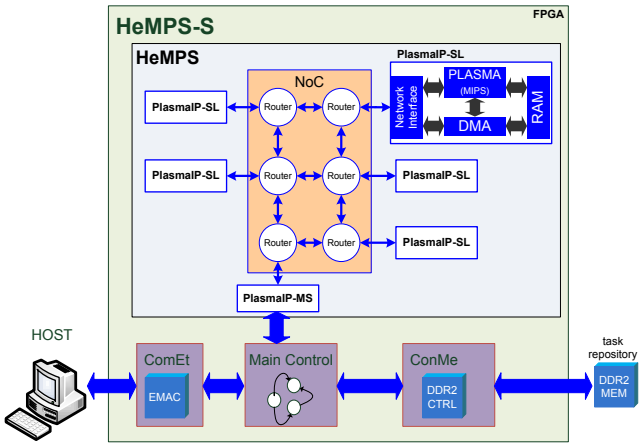


Figure 2 – The HeMPS-S Architecture – the NoC size as well as the number of processors is parameterizable.

To achieve high performance in the processing elements, the Plasma-IP architecture separates communication from computation. The network interface and DMA modules are responsible for sending and receiving packets, while the Plasma processor performs task computation and wrapper management. The local RAM is a true dual port memory, allowing simultaneous processor and DMA accesses, which avoids extra hardware for elements like *mutex* or cycle stealing techniques.

Each PlasmaIP-SL runs a tiny Operating System (OS), called *microkernel*, which supports multitasking and task communication. The PlasmaIP-MS also runs a microkernel, but do not execute applications tasks. The microkernel segments memory in pages, which it allocates for itself (first page) and tasks (subsequent pages). Each Plasma-IP has a *task table*, with the location of local and remote tasks. A simple preemptive scheduling, implemented as a round robin, provides support to multitasking.

The microkernel protects memory pages, and all communication among tasks occurs through message passing only. Message passing is supported through a global message pipe located in the microkernel and communication primitives (*WritePipe()* and *ReadPipe()*), which compose the current HeMPS API. The kernel is described mostly in C and some special functions in assembly (e.g. interruption treatment and context saving).

The communication model employs non-blocking writings and blocking reads. This is an important feature, since a message is only sent through the network when it is requested, reducing the traffic in the NoC.

HeMPS API assumes that user applications are modeled by task graphs. For example, Figure 3 illustrates the graph which edges represent the communication between tasks. In this example, the application is composed by four tasks where tasks A and B send messages to task C, and this to the task D. These tasks are described in C language.

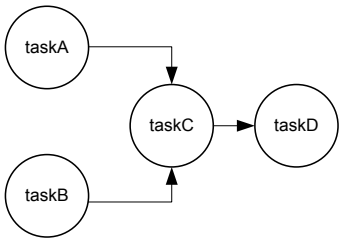


Figure 3 – Task graph modeling a given application.

B. ConMe Unit

The ConMe unit connects the Main Control to the external DDR2 controller. The ConMe is subdivided in two blocks: (*i*) interface, which translates the commands/addresses/data from the Main Control to the DDR2 controller (it is in fact a wrapper); (*ii*) DDR2 controller, generated by the Xilinx Coregen tool.

It is important to understand that all applications to be executed by HeMPS are initially stored in the external DDRs memory (*task repository*). Initially the host computer write all tasks in this repository (*ComEt→Main Control→ConMe*), keeping the HeMPS in a reset state. After loading all tasks into the repository, the host sends a command to start HeMPS. At this moment, the PlasmaIP-MS has full access (read only) to the task repository.

C. ComEt Unit

This unit is responsible for the communication between the MPSoC and the host computer. Unlike most projects, the TCP/IP stack is implemented in hardware, without the need of using a processor dedicated to process packets received from the network. This strategy is adopted to improve performance and to reduce area.

ComEt is composed by a transmission module, a reception module and a MAC Ethernet core. The modules communicate with the MAC Ethernet and the Main Control module. These modules implements three protocols of TCP/IP stack: UDP, IP and ARP. The reception module removes the IP packet overhead and transmits payload to Main Control. The transmission module inserts the header on messages from the master PE. These messages are mostly debug messages generated by the executing tasks, enabling system evaluation and debug online.

D. Main Control Unit

This module is responsible to control the interaction among HeMPS-S units. It has the following functions: *(i)* process commands received from ComEt; *(ii)* write/read data on ConMe; *(iii)* transmit task codes to HeMPS; *(iv)* transfer debug messages from HeMPS to ComEt. The Main Control receives from *ComEt* the payload of UDP packets generated by the host computer. This payload contains commands to manage HeMPS-S, as shown in Table 2.

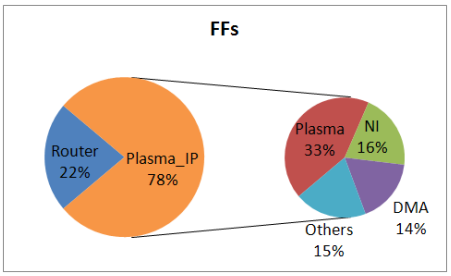
TABLE 2 – HEMPS-S COMMANDS.

Command	Description
Connect	Connect the host computer to HeMPS-S
Write	Write the binary codes on the repository
Start	Reset HeMPS, starting the execution
Disconnect	Disconnect the host computer from HeMPS-S

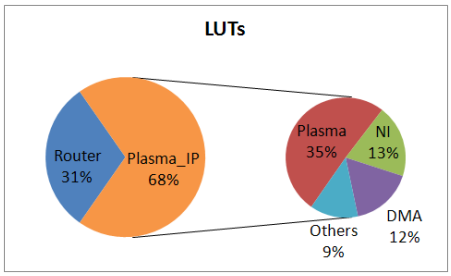
All these hardware modules (*ConMe*, *ComEt*, *Main Control*) are MPSoC independent. The hardware designer can easily adapt the commands and the interface to comply with other MPSoC.

E. HeMPS-S Prototyping

This Section describes the HeMPS-S prototyping process, and the required design modifications to cope with the timing constraints. The device is a Xilinx Virtex 5 5vlx330tff1738-2, chose due to the large amount of available embedded memory blocks, and a MAC Ethernet hard core, which is required by the *ComEt* unit. The main CAD tools employed were Xilinx *PlanAhead* 11.1 and *ModelSim*. *PlanAhead* enables floorplaning, constraints insertion, debugging and conducting timing/area results analyses. Differently from ISE, in *PlanAhead* we can synthesize multiples implementation strategies in the same design. Thus, it becomes possible to evaluate a larger number of design alternatives, speeding up the prototyping process. *Modelsim*, although not used for prototyping, is used to simulate the entire system.



(a) Flip-Flop utilization: router 202, Plasma-IP 706.



(b) LUT utilization: router 685, Plasma-IP 1554.

Figure 4 – Area utilization for the *Plasma_Router*, targeting the 5vlx330tff1738-2 device.

The original HeMPS description uses the Hermes NoC as an IP. Such approach is not adequate for prototyping purposes, since the router should be located physically near to the PE with which it communicates to avoid long wires. Therefore, a new module was created, called *Plasma_Router*. *Plasma_Router* contains the Plasma-IP and a Hermes NoC central router. This router contains five ports. As a function of the placement of the *Plasma_Router* in the NoC, some ports are not used (e.g. north port of the top routers). Unused ports have their control signals connected to ground. The synthesis tool removes the unused logic, reducing the area, as in the original description. Figure 4 shows the flip-flops and LUTs utilization, respectively, of each module of *Plasma_Router*. Note that the Router being evaluated has five ports. Thus, results are upper bounds.

This prototype has three external clock sources: a 50 MHz, and two 25 MHz clocks. Figure 5 illustrates how clock signals for the 4 main units are obtained from the 50 MHz external clock. The two 25 MHz clocks are generated by the external PHY, which communicates with the MAC Ethernet.

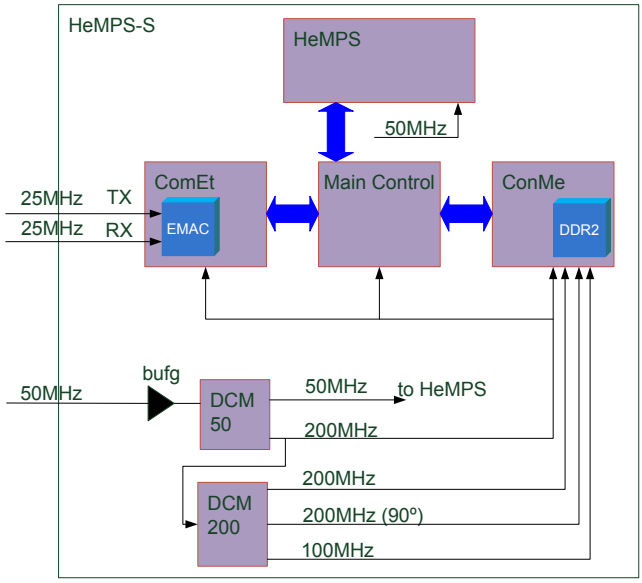


Figure 5 – HeMPS-S clock organization – DCM (digital clock manager) is a FPGA component to generate clock signals.

The number of PEs is limited by the amount of available memory blocks (BRAMs). Each prototyped Pasma-IP has a 16 kB RAM, which uses 16 BRAMs. The selected device could hold up to 20 PEs, since it has 324 BRAMs. Figure 6 illustrates the floorplan of a 2x3 HeMPS-S instance. Each PE requires 16 BRAMs, using two 8-BRAM columns. The location of *ComEt* and *ConMe* units are chosen according to the proximity of the pins location to these modules in the device.

The HeMPS MPSoC reset signal is generated by the Main Control unit, after the complete task repository transmission. One problem found during prototyping is that the synthesis tool does not recognize the reset wire to each PE as a special wire. To avoid timing errors, we used a buffer (BUFG) to drive the reset signal of the HeMPS MPSoC.

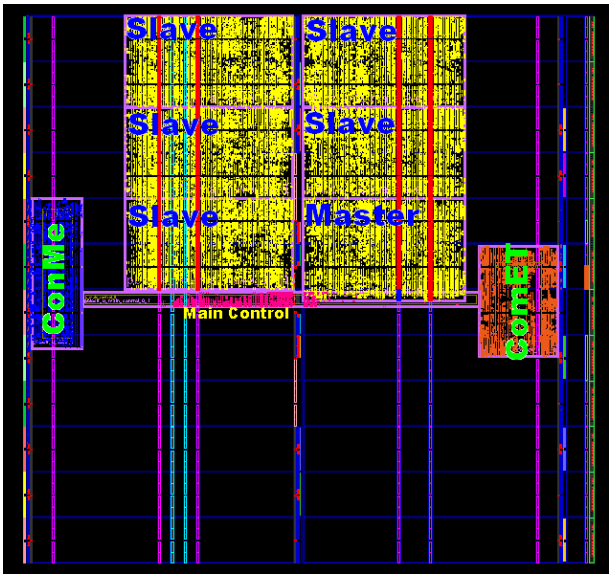


Figure 6 – HeMPS-S floorplan: the columns in red at each PE indicates BRAMs used as PE memory.

The ComEt unit has two clock domains. From one side the Main Control reads/writes data at 200 MHz, and from the other site the PHY reads/writes at 25 MHz. Also, there is no detectable relationship between these phases, although there is a relationship in frequency. The adopted solution to synchronize both clock domains is to uses a synchronous write/asynchronous read LUTRAM to store data, and a two-flop synchronizer for the control signals. Figure 7 shows the architecture to process the UDP packet transmission, which is sent to the host. The *tx_ack* signalizes available data to be consumed in the asynchronous buffer, in the 200 MHz clock domain. This signal is synchronized in the other clock domain, and consumption starts. After reading the contents of the

LUTRAM, the *tx_done_in* indicates end of consumption in the 25 MHz clock domain, and this control signal is synchronized to the other clock domain. The same structure is used in the opposite direction.

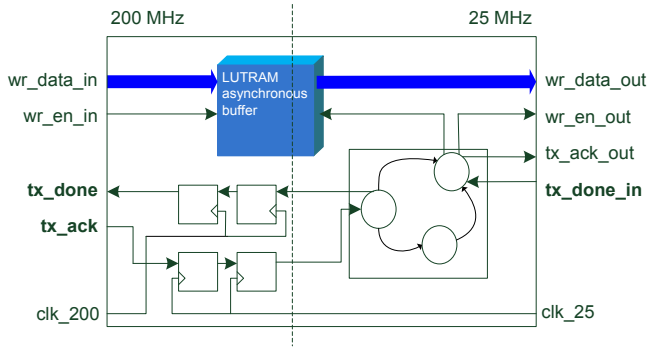


Figure 7 – Part of the UDP transmission module.

Another issue was the interface between the Plasma-IP Master and the Main Control module. The master requests data from the repository at 50 MHz, and the Main Control access the DDR2 memory at 200 MHz, but both clocks have the same phase (guaranteed by the DCMs). To synchronize this interface a 4-phase protocol is used - Figure 8, *read_req* (50 MHz) and *data_valid* (200 MHz). The first read is slower than the subsequent ones, since the DDR2 controller reads 16 words simultaneously, buffering them. To avoid slack errors during reading, the *data_valid* signal is asserted one cycle after the *data_read*. Figure 8 shows in (1) the Plasma-IP requesting a read, and one cycle after (2) the *data_valid* fall because the word is not available in the bus yet. One cycle after data is available on *data_read*, the signal *data_valid* is asserted (3). The Plasma-IP signalizes that the read was executed (4), falling *read_req*.

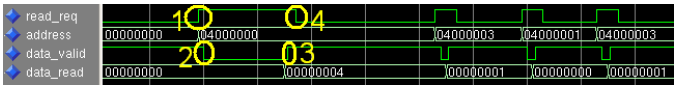


Figure 8 – 4-phase protocol at the interface Master PE – Main Control.

IV. HeMPS-S Framework

The user on a host computer, using the *HeMPS-S generator* framework – Figure 9, controls the HeMPS-S. The framework is responsible for the generation of the task repository, and the communication management with the FPGA board. The *HeMPS-S Generator* environment enables:

1) *Platform Configuration*: number of processor connected to the NoC, through parameters *X* and *Y*; the maximum number of simultaneous running tasks per slave; the page size (and consequently the RAM size); and the processor abstraction description level (simulation only). User may choose among a processor ISS or VHDL description. Both are functionally equivalent, with ISS being faster and VHDL giving more detailed debug data.
2) *Insertion of Applications into the System*: the left panel of Figure 9 shows applications *mpeg* and *communication* inserted into the system. The *communication* application has 4 tasks, and the *mpeg* application 5 tasks (*start, ivlc, idtc, iquant, print*).
3) *Define the initial task mapping*: note in Figure 9 the *start* task (initial task of *mpeg*) mapped in processor 01, and tasks *taskA* and *taskB* (initial tasks of *communication*) allocated to processors 11 and 12 respectively. The remaining tasks are assigned to the master processor, to be dynamically mapped during system execution.
4) *Generation of the binary codes*: through the *Generate* button, all binary codes are generated. The *Generate* button also parameterizes the VHDL files for the

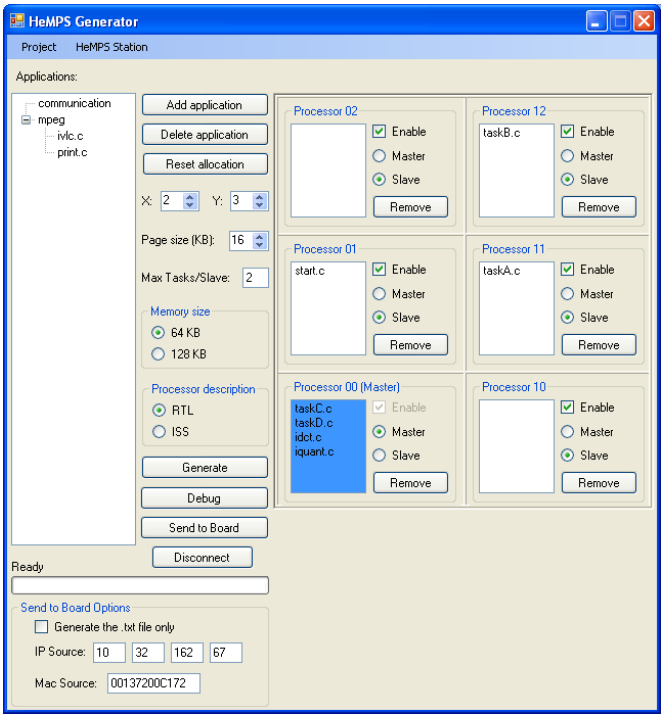


Figure 9 – HeMPS-S generator framework.

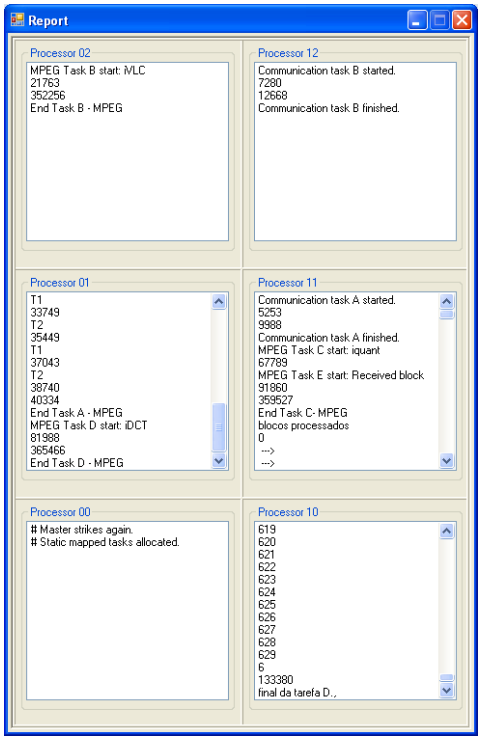


Figure 10 – Hardware debug GUI, displaying the results sent to the host from the FPGA board.

hardware synthesis (executed once) and creates the memory images for the *microkernel*.

5) *Definition of the HEMPS-S network addresses*: IP and MAC addresses. This is still a system limitation, the IP is not acquired dynamically by the DHCP protocol.
6) *Debug button*: used to display the RTL simulation results, in a dedicated simulation debug GUI (Figure 11, discussed later).
7) Debug the system in the FGPA board: the *Send to Board* button execute 4 actions: (*i*) connect the framework with the system in the FPGA (see Table 2 commands); (*ii*) fill the task repository by sending all binary codes to the external memory; (*iii*) release the HeMPS reset through command *start*; (*iv*) receive debug messages.

When the task repository data has been transferred to the external memory, the host computer starts the execution of the applications on HeMPS (command start). Then, the Plasma-IP Master starts reading the repository through the Main Control and *ConMe* units and sends tasks to each slave. Once a given task is received in a Plasma-IP Slave, it is scheduled to be executed. Tasks can communicate with other tasks, in the same or in a different processor, and can also send debug data to the master processor. The master sends debug data to the host computer through the *Main Control* and *ComEt* module. In the host computer, after pressing the *Send to Board* button, a GUI displays this debug data, enabling system performance evaluation at running time, as illustrated in Figure 10

The debug window (Figure 10) contains one panel for each processor. In the panel corresponding to the Plasma-IP MS (processor 00), two kernel messages are displayed. Processor 11 executes *communication* task A, and then simultaneously it executes *MPEG* tasks C (*iquant*) and E (*print*). This textual map enables to verify the correct behavior of the application, as well as to measure the performance of each task. The numbers after *communication x started* and before *communication x finished* corresponds to the number of clock cycles spent since the beginning of the system execution, obtained through the *gettick()* system call. The use of this system call enables to compute the task execution time, latency, and throughput.

The simulation debug GUI (Figure 11), invoked after RTL simulation, contains one tab per task executing in each processor. For example, in Figure 11, a HeMPS instance with 15 PEs is simulated, with 8 processors executing simultaneously 3 tasks each. In this way, messages are separated, allowing to the user to visualize the execution results for each task separately. The hardware debug GUI is currently being adapted to display tasks separately, as in the simulation debug GUI. The main difference between prototyping and simulation is that the former receive data at runtime, and the later reads a file generated after simulation.

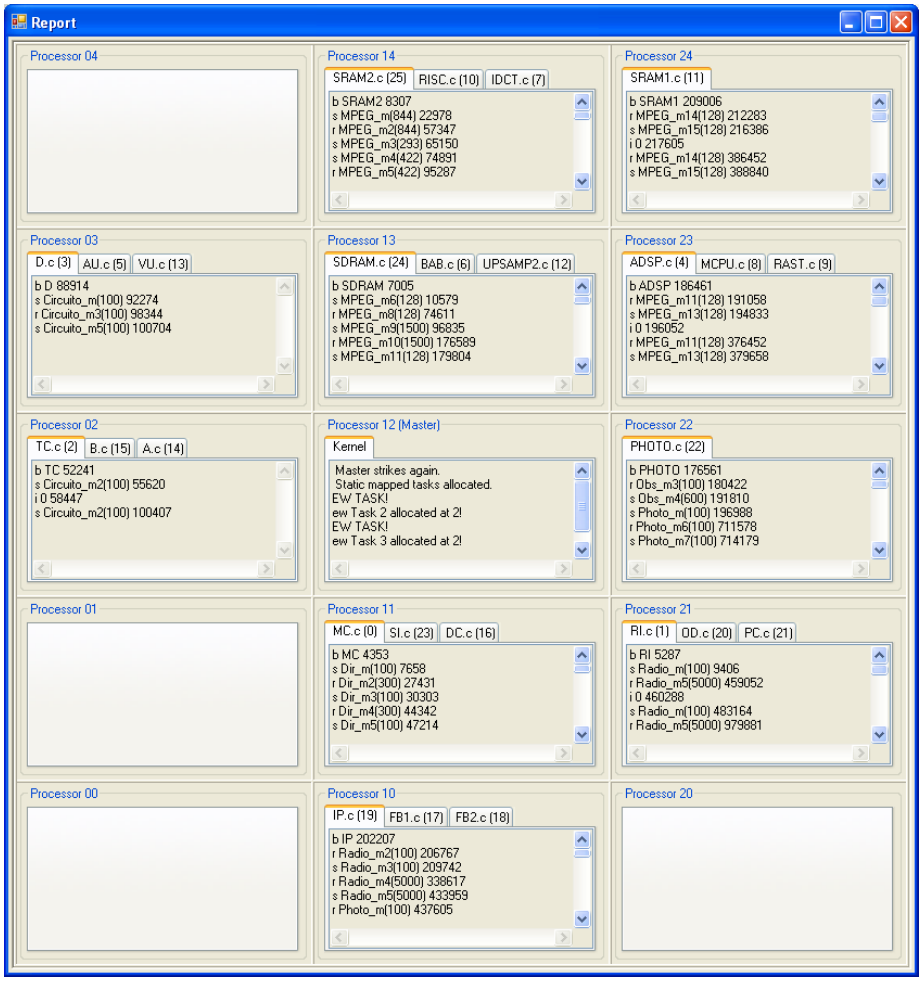


Figure 11 – Simulation debug GUI, displaying the results obtained after RTL simulation.

V. Conclusion and Future Works

This paper presented the HeMPS-S framework, composed by software executing in a given host, and an NoC-based MPSoC executing in FPGA, both interconnect through an Ethernet connection. The basis for the HeMPS-S framework is the HeMPS MPSoC, which is open source, available at www.inf.pucrs.br/~gaph. New hardware or software techniques can be easily integrated in HeMPS-S, enabling to construct proof of concept prototypes for these new techniques. The hardware modules, extern to the MPSoC, can be ported to comply with other MPSoC. Then the designer can use these modules to validate his MPSoC platform.

Ongoing work includes a distributed shared memory with cache coherence [11]. Future works include the addition of: (*i*) hardware monitors to collect data related to power, latency and throughput; (*ii*) include new processors; *(iii)* include the DHCP protocol on the communication infrastructure; *(iv)* a decentralized control mechanism, to avoid a communication bottleneck on the master processor.

VI. Acknowledgements

The Authors acknowledge the support of CNPq and FAPERGS, projects 301599/2009-2 and 10/0814-9, respectively.

VII. References

[1] Van Berkel, C.H. "*Multi-core for mobile phones*". In: DATE, 2009, pp. 1260-1265.

[2] Jalier, C.; Lattard, D.; Jerraya, A. A.; Sassatelli, G.; Benoit, P. and Torres, L. "*Heterogeneous vs Homogeneous MPSoC Approaches for a Mobile LTE Modem*". In: DATE, 2010, pp 184 - 189.

[3] Zhang, W; Geng, L.; Zhang, D.; Du, G.; Gao, M; Zhang W.; Hou, N.; Tang Y. "*Design of heterogeneous MPSoC on FPGA*". In: ASIC, 2007, pp. 102-105.

[4] Hammami, O.; Li, X.; Larzul, L.; Burgun, L. "*Automatic design methodologies for MPSOC and prototyping on multi-FPGA Platforms*". In: ISOCC, 2009, pp. 141-146.

[5] Garzia, F.; Airoldi, R.; Ahonen, T.; Nurmi, J.; Milojevic, D. "*Implementation of the W-CDMA cell search on a MPSoC designed for software defined radios*". In: SiPS, 2009, pp. 30-35.

[6] Limberg T.; et. al. "*A Heterogeneous MPSoC with Hardware Supported Dynamic Task Scheduling for Software Defined Radio*". In: DAC University Booth, 2009.

[7] Tilera Corp. http://www.tilera.com/products/processors

[8] Vangal, S.R.; et al. "*An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS*". IEEE Journal of Solid-State Circuits, vol.43, no.1, Jan. 2008, pp.29-41.

[9] Carara, E. A.; Oliveira, R. P.; Calazans, N. L. V.; Moraes, F. G. "*HeMPS - a framework for NoC-based MPSoC generation"*. In: ISCAS 2009, pp. 1345-1348.

[10] Moraes, F.; Calazans, N.; Mello, A.; Möller, L.; Ost, L. "*Hermes: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip*". Integration, the VLSI Journal, Vol. 38(1), October, 2004, pp.69-93.

[11] Chaves, T.; Carara, E. A.; Moraes, F. G. "*Energy-efficient Cache Coherence Protocol for NoC-based MPSoCs*". In: SBCCI, 2011.