H2020 FRAMEWORK PROGRAMME
ICT-01-2014: Smart Cyber-Physical Systems

PROJECT NUMBER: 645496

Agile, eXtensible, fast I/O Module for the cyber-physical era

D7.3 –Performance and Energy Evaluation of AXIOM

Due date of deliverable: 31st January 2018
Actual Submission: 14th February 2018 (agreed extended date)

Start date of the project: February 1st, 2015 Duration: 36 months

Lead contractor for the deliverable: UNISI

Revision: See file name in document footer.

Project co-founded by the European Commission within the HORIZON FRAMEWORK PROGRAMME (2020)	
Dissemination Level: PU	
PU	Public
PP	Restricted to other programs participant (including the Commission Services)
RE	Restricted to a group specified by the consortium (including the Commission Services)
CO	Confidential, only for members of the consortium (including the Commission Services)

Change Control

Version#	Date	Author	Organization	Change History
1	09.01.2018	Roberto Giorgi	UNISI	
2	11.02.2018	ALL PARTNERS	ALL	

Release Approval

Name	Role	Date
Roberto Giorgi	WP-leader	14.02.2018
Roberto Giorgi	Coordinator	14.02.2018

Project: **AXIOM - Agile, eXtensible, fast I/O Module for the cyber-physical era**
Grant Agreement Number: **645496**
Call: **ICT-01-2014: Smart Cyber-Physical Systems**

The following list of authors will be updated to reflect the list of contributors to the document.

Roberto Giorgi, Farnam Khalili, Francesco Montefoschi, Marco Procaccini
University of Siena, Italy

Xavier Martorell, Daniel Jiménez-González, Carlos Álvarez (UPC/BSC)
UPC/BSC, Spain

Paolo Gai, Stefano Garzarella
R&D Department, Evidence, Italy

Vassilis Amourianos, Dimitris Theodoropoulos, Dionisios Pnevmatikatos
Institute of Computer Science, FORTH, Greece

David Oro
RD department, Herta Security – HERTA, Spain

Davide Catani, Davide Cardillo, Christian Magnani, Stefano Viola
R&D Department, SECO, Italy

Nicola Bettin, Alberto Pomella
RD department, VIMAR S.p.A., Italy

TABLE OF CONTENTS

TABLE OF FIGURES

GLOSSARY

AEP – AXIOM Evaluation Platform
API – Application Programming Interface
ARM – Instruction set architecture developed by ARM Holdings Ltd.
ASIC – Application-specific integrated circuit
AVR – Alf and Vegard RISC processor
AXI – Advanced Extensible Interface
BRAM – Block RAM
BSP – Board Support Package
CPS - Cyber-physical system
DMA – Direct Memory Access
FF – Flip-Flop
FIFO – First In First Out
FC – Fully connected layer
FPGA – Field Programmable Gate Array
GASNet – Global Address Space Networking
GPIO – General Purpose Input Output
GPU – Graphics Processing Unit
GTH – Gigabit Transceivers of class H
HD – High Definition
HDL – Hardware Description Language
HLS – High Level Synthesis Language
HPC High Performance Computing
MPSoC Multiprocessor System-on-Chip
I2C – Inter Integrated Circuit
I2C I/F Inter Integrated Circuit Interface
IOCTL – Input/Output Control
IoT – Internet of Things
IP – Intellectual Property
IPIF - AXI4-Lite IP Interface
ISA – Instruction Set Architecture
JUMP – Distributed Shared Memory System
LUT – Look-Up Table
MGT – Multi Gigabit Transceiver
MKL-DNN – Intel Corporation math kernel libraries for deep neural networks
MPI – Message Passing Interface
MPSoC – Multiprocessor System-on-Chip
NDA – Non-disclosure agreement
NIC – Network interface
OmpSs – OpenMP Superscalar
OpenCV – Open Source Computer Vision Library
OpenGL ES – Reduced specification of the OpenGL standard that targets embedded devices
OpenMP – Open Multiprocessing standard
OpenMPI – Open Source Message Passing Interface
PL – Programmable logic
PS – Processing System
RDMA – Remote Direct Memory Access
ROI – Region of Interest
RTL – Register transfer language
RX – Receiver Subsystem
SD I/F – Secure Digital Interface
SDIO – Secure Digital Input Output
SDK – Software Development Kit
SDSM – Software Distributed Shared Memory
SHL – Smart home living scenario developed for the AXIOM project
SMP – Symmetric Multiprocessor System
SoC – System on chip
SVS – Smart surveillance scenario for the AXIOM board
SYSFS – Pseudo File System provided by the Linux Kernel
SYSMON – System Monitor
TCP – Transmission Control Protocol
TX – Transmitter Subsystem
UART – Universal Asynchronous Receiver/Transmitter
UDP – User Datagram Protocol
USB – Universal Serial Bus
XCVRS – Transceivers
XDMA – AXIOM FPGA Direct Memory Access kernel module
XSMLL – eXtreme Shared Memory Low Level thread support
XTAL – Cristal Oscillator Frequency
XTQ – X Thread Queue
YUV – Color Space

Executive summary

The main goals of the AXIOM-WP7 are:

- *Definition of the AXIOM evaluation platform (AEP) appropriate for Cyber-Physical Systems*
- *Definition and development of appropriate Design Space Exploration (DSE) tools and methodologies*
- *Integrate in the evaluation platform the main results from all the all work packages*

During the third year of the AXIOM project, we further extended the AEP and the DSE tools, while enabling all the Partners to perform their experiments in a scientifically rigorous and repeatable way by using those common tools. The availability of our own manufactured AXIOM-board permitted the migration of our simulation designs into the actual hardware and the porting of runtime tools and device drivers to the actual platform.

More extensive power and performance measurements have been carried out at any level. In the final months more optimizations and enhancements were possible. In particular the AXIOM-link speeds reached 18 Gbps in full-duplex and by using just one simple USB-C cable between two boards, this exceeding by 3 times our initial link speed target of 6Gbps on standard SATA cables.

The final key applications SHL and SVS were also extensively tested and we performed power and performance measurements through on-board facilities and also with external instruments. The different methodologies allowed also a cross-validation of our measurements.

Further improvements are possible, but we believe that we have met or exceeded all project objectives.

1 Introduction

The main goal of the AXIOM project is to build a reference board for Smart Cyber-Physical Systems. To that end, in this Workpackage we intended to select and use the best suitable methodology and tools in order to assess the performance and to steer the design by exploring the most promising options (Design Space Exploration or DSE).

In the previous reporting period we illustrated the AXIOM Evaluation Platform and in particular the tools: COTSon (simulator), MYINSTALL (one installation requires 10 to 15 minutes due to the complexity of the framework), MYDSE (to cover the interesting design point and manage several days of simulations), GTCOLLECT (to re-unite and take appropriate average across several repeated experiments), GTGRAPH (to immediately visualize the results of the experiments), GENIMAGE (to generate guest "SD-images" for the simulator – this takes hours and GB of data). In other words, the tools are the only way to make the design of a complex system feasible. In parallel, we developed the driving examples and analyzed the full-system behavior together with the Operating System activity.

In this reporting period, we further improved the DSE tools for the upcoming needs so that we could finalize the models (functional and timing) of our hardware/software components. The tools permitted us to better define the needed architectures in hardware and in software, with particular reference to the device-drivers, register maps and interfaces between hardware and software, soft-IPs like the advanced DF-Thread/XSMLL thread management system at low level and its API. Comparisons between the simulator estimation and actual AXIOM-board confirmed our initial performance predictions.

Moreover, the Partners were able to perform the last task of final verifications and validation of the two key applications for Smart Home/Living (SHL) and Smart Video Surveillance (SVS) and performed more accurate measures of performance and power on the actual AXIOM-board. This was also possible due to the availability of the on-board facility for direct power-measurements but also thanks to the on-board trace-port that permitted the synchronization of the execution with the external measurements tools.

In addition to the basic functionalities foreseen for our system, we were able to optimize the AXIOM-link speeds to achieve a transfer rate on a single USB-C cable of about 18Gbps which is triple the initial target of about 6 Gbps (maximum transfer rate of SATA cables). We were able to manage the design at any level (from the platform to the applications and tools) of 64 bit instead of the initial target of this project of 32-bit. The XSMLL/DF-threads execution model seems also very promising to further accelerate the execution of applications by about one order of magnitude compared to what can be achieved by other programming models such as OpenMPI, Cilk, and Scope-Consistency DSM (JUMP).

1.1 Document structure

This document is organized as follows:

- In Section 2, we illustrate the completed design of tools, driver, runtime, benchmarks and the migration of models from the simulator to the RTL design tools;
- In Section 3, we illustrate the performance enhancements of the AXIOM-link and its impact in the project;
- In Section 4, we illustrate the integration efforts of the several soft-IPs and software components in order to obtain the "SD-image" the drives the AXIOM-board;
- In Section 5, we illustrate the tools for on board power measurement;
- In Section 6, we illustrate the power measurements on the actual board related to our driving benchmarks.
- In Section 7 and Section 8, we illustrate power and performance measurements on the final key applications SHL and SVS.
- Finally, we draw the conclusions.

1.2 Relation to other deliverables

Deliverables D3.3 refers to the evaluation of the user experience: cooperation between WP3 and WP7 was planned in order to make sure that we could use the same final applications and inputs.

Deliverables D4.3 refers to the fine tuning of reference kernels such as the Matrix Multiplication which is continuously driving the development of features and the performance and power evaluations.

Deliverables D5.1, D5.4 refer to the support for the OS, the runtime, the driver, the boot information, the bitstream flashing and a set of testing utilities needed to verify the basic functionalities of the system.

Deliverable D6.5 describes the system interconnect features released at the end of Jul 2017; more advanced features had been developed subsequently to better support higher performance of the system and are documented in this deliverable.

1.3 Tasks involved in this deliverable

This deliverable is the result of the work developed in tasks:

- Task T7.2 (month 4 - 30): Continuous development of performance evaluation tools and DSE (Partners: UNISI, BSC, EVI, FORTH, SECO).
- Task T7.3 (month 4 - 30): Evaluation from kernels to benchmarks and final application code.
- Task T7.4 (month 31-36): Final Verification and validation with the two key applications.

There is also a close collaboration with the following task (only reported here for reference):

- Task T3.3 (month 31-36): Testing for User Experience.
- Task T4.5 (month 25-36): Evaluation and tuning of the code generation.
- Task T5.1 (month 1-33 (extended)): Operating System.
- Task T6.5 (month 7-30): System Interconnect

2 A novel execution model leveraging the AXIOM board

In the first year of the AXIOM project, the functional model for execution Dataflow threads [31] has been used as a driving example together with basic Fibonacci and Matrix Multiplication kernels to develop the Design Space Exploration (DSE) Tools and the simulation platform (AXIOM Evaluation Platform or AEP) before the real system (the AXIOM-board) becomes available (see D7.1). During the second year we further generalize this platform in order to keep it updated with different OS distribution (and in order to evaluate the effects of the OS on the total execution of the applications) and also to provide the flexibility to adapt to software package updates. This effort provided us with a basic set of tools to achieve our development goals (MYDSE, GTCOLLECT to collect the results, GTGRAPH to visualize the results, GENIMAGE to provide a new SD-IMAGE, INST-IMGPKG to provide multiple OS-images and so on – see D7.2).

As a driving example, since the beginning we wanted to make sure to be able to have a fast way to move the data needed by an application into other boards as depicted in Figure 1.

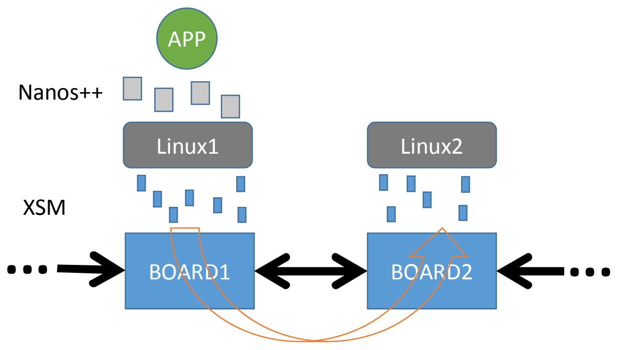


Figure 1 – The DF-Threads distribution. Nanos++ generates coarser grained threads that could be further distributed as DF-Threads across several nodes. XSM=eXtended Shared Memory.

In the third year of the project, thanks to the previous efforts, we started to define the timing models of the basic operation of the DF-Threads and translated those models in actual reconfigurable hardware for the AXIOM-board. In the following we describe the methodology that we have used, the continuous development of the software drivers and runtime hooks (thanks to the simulator) and some initial results.

2.1 *Methodology of Design*

The methodology we choose for developing the most advanced concepts of AXIOM related to the DATAFLOW-THREAD EXECUTION consists in a close co-design in the COTSon framework [9], [10] and the Xilinx Vivado HLS tools [69], [21], [23], [42], [42].

In the COTSon framework, we model and immediately test the functional behavior and the (approximate) timing behavior of the hardware/software components following the "functional-directed" approach [10], [29]. This approach is similar to what was developed in the TERAFLUX project [52], [15], [16], [5], existing tools [48], [49], [50], [51] and other related DSE methodology [6], [11], [12], [13], [14], [38], [39], [40], [27], [28], [24], [44], [45], [46], [47] but here we have new models, several simulation enhancement and the Design Exploration tools (DSE tools) developed in the previous 2 years of the project and described in D7.1 and D7.2.

In Figure 2 we show a simplified design flow to illustrate how we can extend the AXIOM platform to better fit the needs of a variety of real-world (Cyber-Physical) scenarios by using a more efficient execution model based on the DF-threads [31], [30], [7]. [8], [9], [15], [16], [32], [33], [34], [35], [36], [37], [52], [53], [55], [56], [58], [59], [62], [70] on actual hardware.

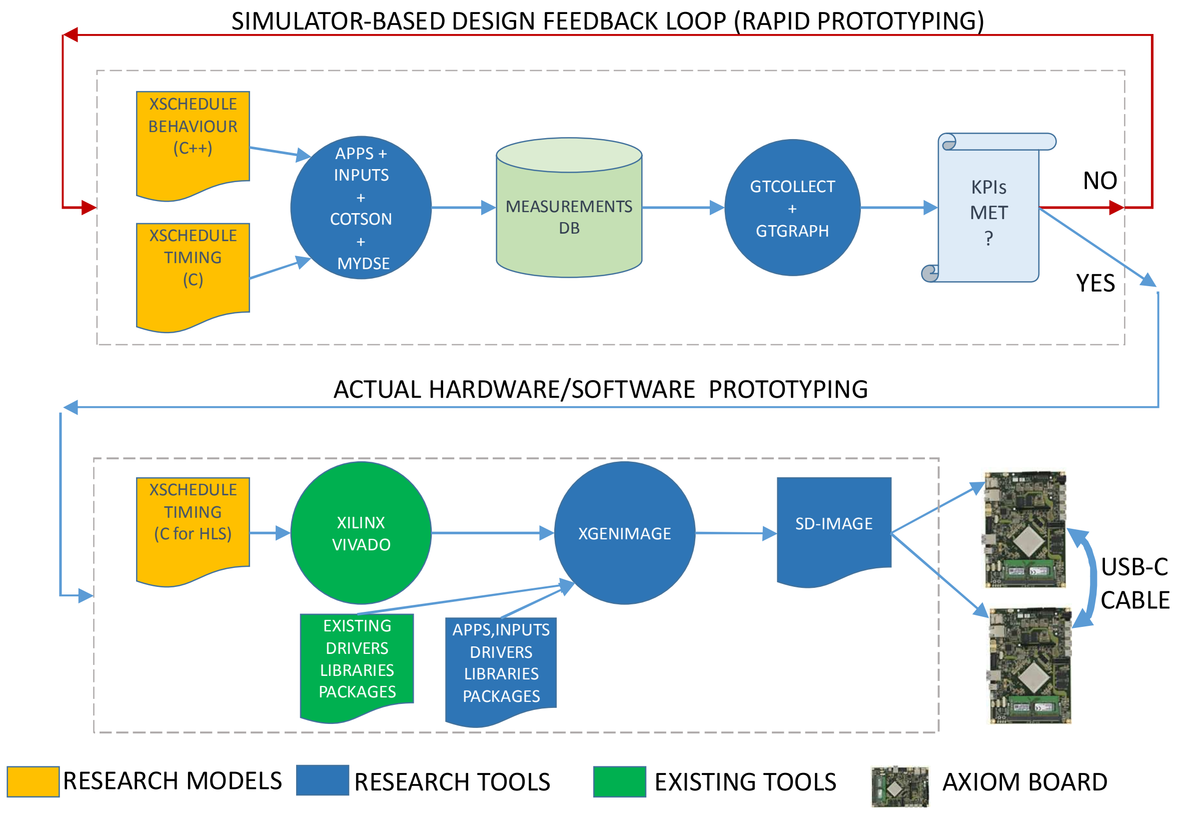

SIMULATOR-BASED DESIGN FEEDBACK LOOP (RAPID PROTOTYPING)
XSCHEDULE BEHAVIOUR (C++)
XSCHEDULE TIMING (C)
APPS + INPUTS + COTSON + MYDSE
MEASUREMENTS DB
GTCOLLECT + GTGRAPH
KPIs MET ?
NO
YES
ACTUAL HARDWARE/SOFTWARE PROTOTYPING
XSCHEDULE TIMING (C for HLS)
XILINX VIVADO
XGENIMAGE
SD-IMAGE
EXISTING DRIVERS LIBRARIES PACKAGES
APPS,INPUTS DRIVERS LIBRARIES PACKAGES
USB-C CABLE
RESEARCH MODELS
RESEARCH TOOLS
EXISTING TOOLS
AXIOM BOARD

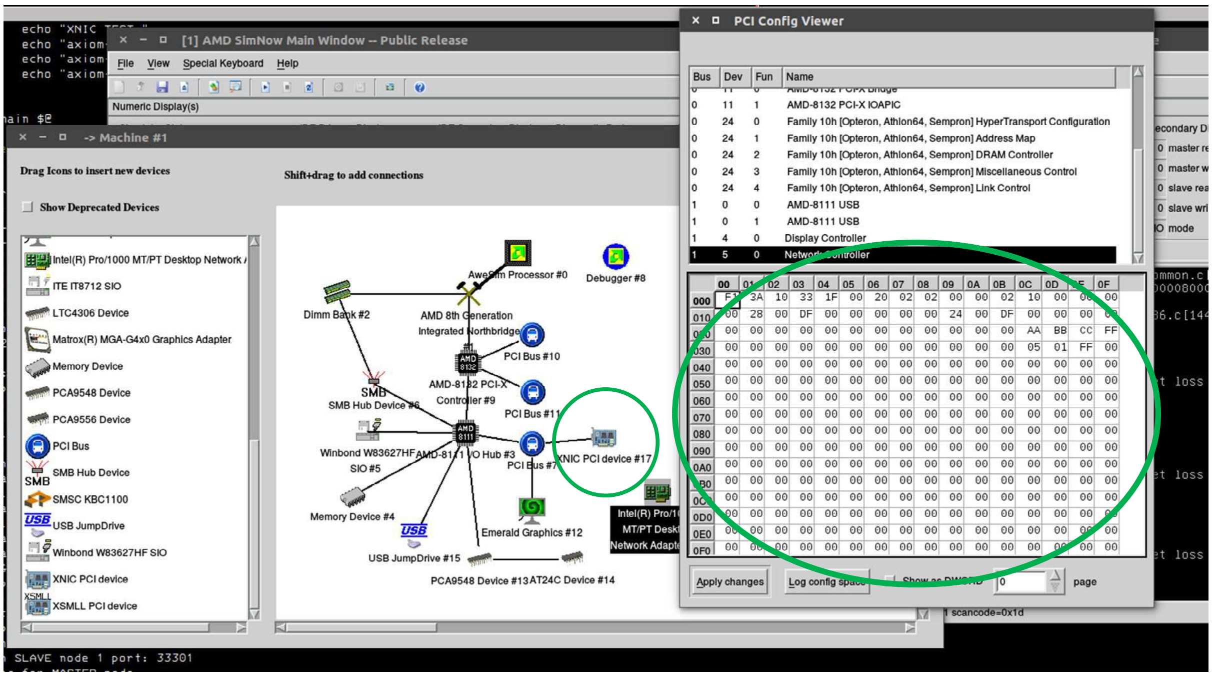
Figure 2 – An example of modeling of one of the key operation for the DF-thread execution model (XSCHEDULE). MYDSE, GTCOLLECT, GTGRAPH, XGENIMAGE are research tools used for the Design Space Exploration and testing of the outcomes. KPI=Key Performance Metrics (e.g., execution time).

In particular, we focus here, as an example, onto the XSCHEDULE operation (introduced in 2015, see D7.1, D5.2, [7]): the XSCHEDULE behavior (first orange block in Figure 2) has been first modeled in the COTSon simulator to verify the seamless execution together with the benchmarks and the operating system. As a subsequent step the actual timing-model of the XSCHEDULE is defined (second orange block in Figure 2) in order to specify the functionalities that may need to go in the hardware and "for difference", the functionalities that may be implemented in the runtime/libraries/drivers, i.e., in the software stack (sw/hw partitioning). Once the timing has been tested, we move the XSCHEDULE design into Xilinx VIVADO-HLS in order to produce the soft-IP that is used in the AXIOM-boards. This IP is put besides other AXIOM related IPs such as the NI, the AVR, the OmpSs task accelerators.

In the rest of this document, we will continue to refer to XSCHEDULE as an example to show the various implementation steps that we have accomplished. The other software components and tools have been described in D4.3, D5.1, D5.4 and the other soft-IPs and hardware components in D6.4, D6.5, D6.6 as well in other previous deliverables. Please note also that many deliverables consist in actual hardware (e.g., the AXIOM board), soft IPs, and software that have been duly prepared, so the related reports serve as a main access point of documentation.

2.2 Completing the software components on the simulator

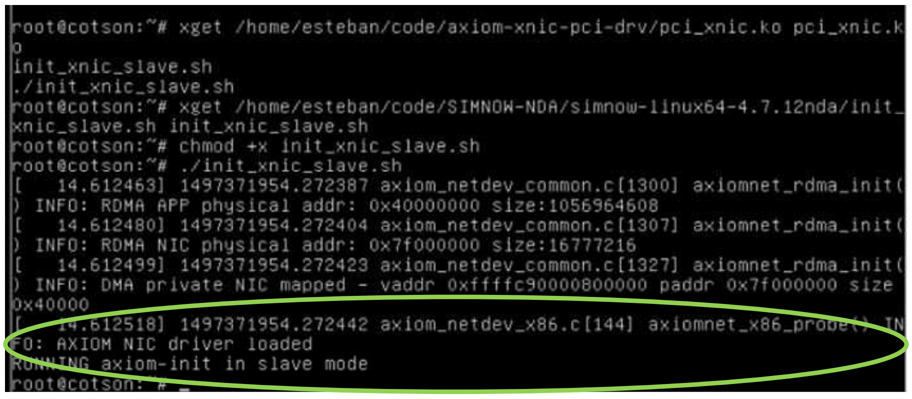
The COTSon framework has been used to develop a preliminary model of a device that could have been visible from the application side. This device was initially modeled only for a direct usage of the simulator. In the third year, thanks to the availability of the software utilities and runtime developed in the second year (see D5.2, D5.3), we interfaced more properly such device as a PCI device (XNIC in Figure 3, where we see also the 256 internal configuration registers of the PCI device) for more general and easy interfacing: in this way we were able to develop custom register set and interface as illustrated in D5.4).



```
Bus  Dev  Fun  Name
0    11   1    AMD-8132 PCI-X IOAPIC
0    24   0    Family 10h [Opteron, Athlon64, Sempron] HyperTransport Configuration
0    24   1    Family 10h [Opteron, Athlon64, Sempron] Address Map
0    24   2    Family 10h [Opteron, Athlon64, Sempron] DRAM Controller
0    24   3    Family 10h [Opteron, Athlon64, Sempron] Miscellaneous Control
0    24   4    Family 10h [Opteron, Athlon64, Sempron] Link Control
1    0    0    AMD-8111 USB
1    0    1    AMD-8111 USB
1    4    0    Display Controller
1    5    0    Network Controller

     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000  F1 3A 10 33 1F 00 20 02 02 00 00 02 10 00 00 00
010  00 28 00 DF 00 00 00 00 00 24 00 DF 00 00 00 00
020  00 00 00 00 00 00 00 00 00 00 00 00 AA BB CC FF
030  00 00 00 00 00 00 00 00 00 00 00 00 05 01 FF 00
040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 3 – The 256 register PCI configuration space and the collocation of the XNIC peripheral representing the device we used to model both the NI and the DF-threads components in the COTSon simulator.

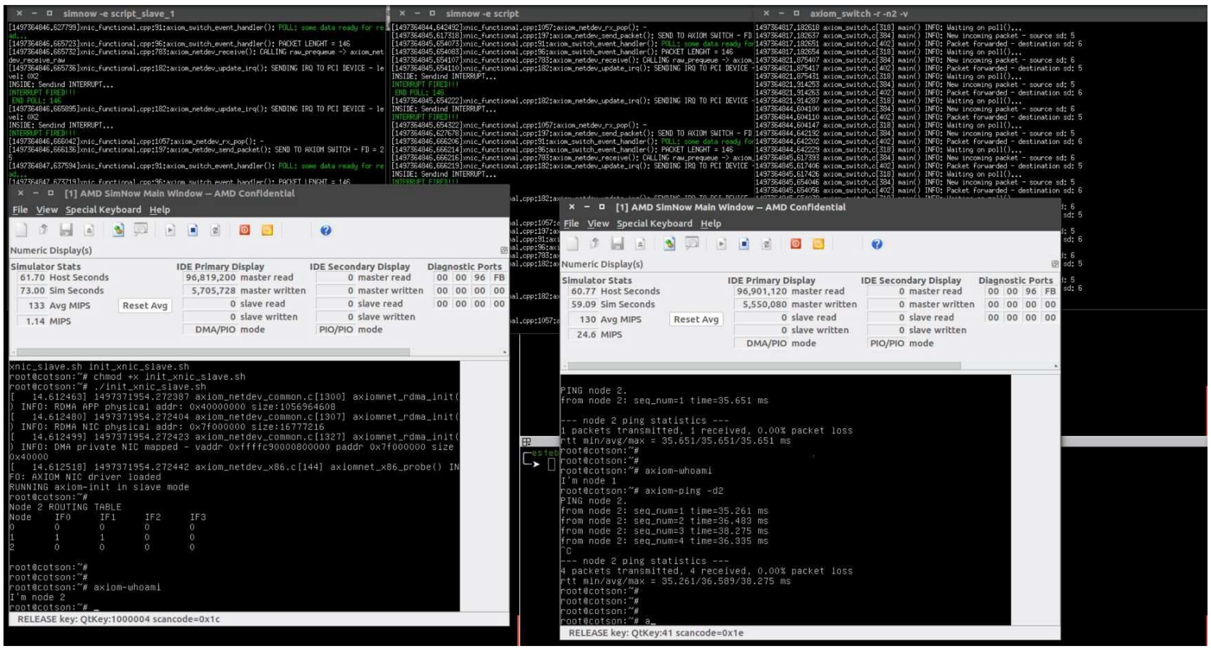
On the software side, we also used the simulator to run the developed corresponding device driver. This has been successfully accomplished and we show that our device is correctly loaded among the peripherals of the simulator in Figure 4.



```
root@cotson:~# xget /home/esteban/code/axiom-xnic-pci-drv/pci_xnic.ko pci_xnic.k
o
init_xnic_slave.sh
./init_xnic_slave.sh
root@cotson:~# xget /home/esteban/code/SIMNOW-NDA/simnow-linux64-4.7.12nda/init_
xnic_slave.sh init_xnic_slave.sh
root@cotson:~# chmod +x init_xnic_slave.sh
root@cotson:~# ./init_xnic_slave.sh
[   14.612463] 1497371954.272387 axiom_netdev_common.c[1300] axiomnet_rdma_init(
) INFO: RDMA APP physical addr: 0x40000000 size:1056964608
[   14.612480] 1497371954.272404 axiom_netdev_common.c[1307] axiomnet_rdma_init(
) INFO: RDMA NIC physical addr: 0x7f000000 size:16777216
[   14.612499] 1497371954.272423 axiom_netdev_common.c[1327] axiomnet_rdma_init(
) INFO: DMA private NIC mapped - vaddr 0xffffc90000800000 paddr 0x7f000000 size
0x40000
[   14.612518] 1497371954.272442 axiom_netdev_x86.c[144] axiomnet_x86_probe() IN
FO: AXIOM NIC driver loaded
RUNNING axiom-init in slave mode
root@cotson:~# _
```

Figure 4 – The device driver is loaded in the Linux system.

The utilities (`axiom-ping`, `axiom-netperf`) developed by Partner EVI during the second year (D5.2, D5.3) have been used to test (inside the simulator, as we can see from Figure 5) the functionality of the nodes and the interfacing with the runtime (`axiom-run`) and other utilities for network topology discovery and routing table check (Figure 5). In Figure 6, we can see also the `axiom-netperf` utility to verify the functionality of our modeled AXIOM-link interconnection.



```
Simulator Stats              IDE Primary Display        IDE Secondary Display    Diagnostic Ports
 61.70 Host Seconds          96,819,200 master read            0 master read     00 00 96 FB
 73.00 Sim Seconds            5,705,728 master written         0 master written  00 00 00 00
   133 Avg MIPS  Reset Avg            0 slave read             0 slave read      00 00 00 00
  1.14 MIPS                           0 slave written          0 slave written
                               DMA/PIO mode               PIO/PIO mode
```

```
xnic_slave.sh init_xnic_slave.sh
root@cotson:~# chmod +x init_xnic_slave.sh
root@cotson:~# ./init_xnic_slave.sh
[   14.612463] 1497371954.272387 axiom_netdev_common.c[1300] axiomnet_rdma_init(
) INFO: RDMA APP physical addr: 0x40000000 size:1056964608
[   14.612480] 1497371954.272404 axiom_netdev_common.c[1307] axiomnet_rdma_init(
) INFO: RDMA NIC physical addr: 0x7f000000 size:16777216
[   14.612499] 1497371954.272423 axiom_netdev_common.c[1327] axiomnet_rdma_init(
) INFO: DMA private NIC mapped - vaddr 0xffffc90000800000 paddr 0x7f000000 size
0x40000
[   14.612518] 1497371954.272442 axiom_netdev_x86.c[144] axiomnet_x86_probe() IN
FO: AXIOM NIC driver loaded
RUNNING axiom-init in slave mode
root@cotson:~#
Node 2 ROUTING TABLE
Node    IF0     IF1     IF2     IF3
0       0       0       0       0
1       1       1       0       0
2       0       0       0       0

root@cotson:~#
root@cotson:~#
root@cotson:~# axiom-whoami
I'm node 2
root@cotson:~# _
```

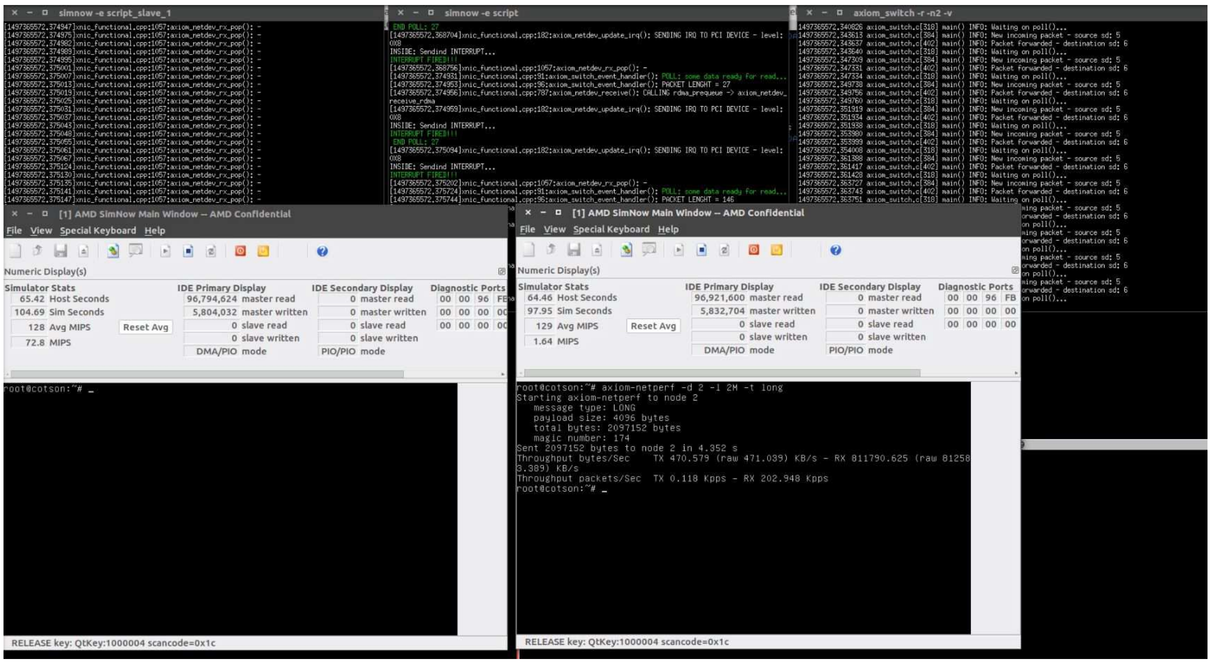
RELEASE key: QtKey:1000004 scancode=0x1c

```
Simulator Stats              IDE Primary Display        IDE Secondary Display    Diagnostic Ports
 60.77 Host Seconds          96,901,120 master read            0 master read     00 00 96 FB
 59.09 Sim Seconds            5,550,080 master written         0 master written  00 00 00 00
   130 Avg MIPS  Reset Avg            0 slave read             0 slave read      00 00 00 00
  24.6 MIPS                           0 slave written          0 slave written
                               DMA/PIO mode               PIO/PIO mode
```

```
PING node 2.
from node 2: seq_num=1 time=35.651 ms

--- node 2 ping statistics ---
1 packets transmitted, 1 received, 0.00% packet loss
rtt min/avg/max = 35.651/35.651/35.651 ms
root@cotson:~#
root@cotson:~#
root@cotson:~# axiom-whoami
I'm node 1
root@cotson:~# axiom-ping -d2
PING node 2.
from node 2: seq_num=1 time=35.261 ms
from node 2: seq_num=2 time=36.483 ms
from node 2: seq_num=3 time=38.275 ms
from node 2: seq_num=4 time=36.335 ms
^C
--- node 2 ping statistics ---
4 packets transmitted, 4 received, 0.00% packet loss
rtt min/avg/max = 35.261/36.589/38.275 ms
root@cotson:~#
root@cotson:~#
root@cotson:~#
root@cotson:~# a_
```

RELEASE key: QtKey:41 scancode=0x1e

Figure 5 – The utilities "axiom-ping" on the right and "axiom-whomai" on the left. In the background we can see the backlog of operation that are running on the host machine (where the simulator is running).



```
Simulator Stats              IDE Primary Display        IDE Secondary Display    Diagnostic Ports
 65.42 Host Seconds          96,794,624 master read            0 master read     00 00 96 FB
104.69 Sim Seconds            5,804,032 master written         0 master written  00 00 00 00
   128 Avg MIPS  Reset Avg            0 slave read             0 slave read      00 00 00 00
  72.8 MIPS                           0 slave written          0 slave written
                               DMA/PIO mode               PIO/PIO mode
```

```
root@cotson:~# _
```

RELEASE key: QtKey:1000004 scancode=0x1c

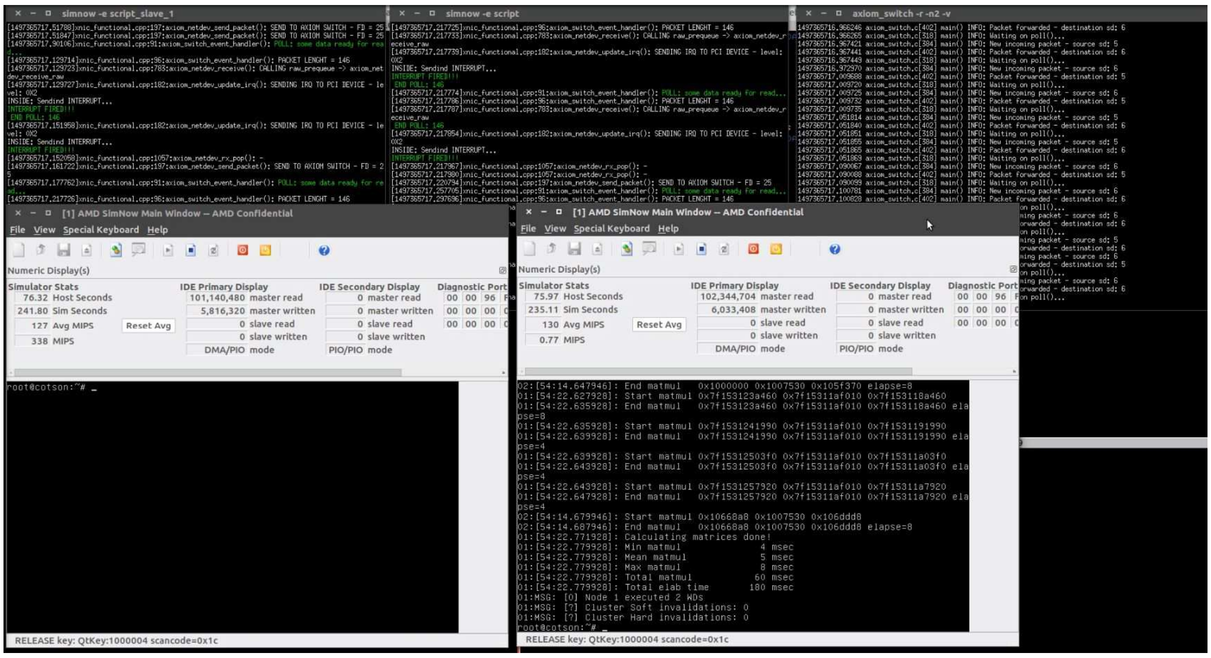
```
Simulator Stats              IDE Primary Display        IDE Secondary Display    Diagnostic Ports
 64.46 Host Seconds          96,921,600 master read            0 master read     00 00 96 FB
 97.95 Sim Seconds            5,832,704 master written         0 master written  00 00 00 00
   129 Avg MIPS  Reset Avg            0 slave read             0 slave read      00 00 00 00
  1.64 MIPS                           0 slave written          0 slave written
                               DMA/PIO mode               PIO/PIO mode
```

```
root@cotson:~# axiom-netperf -d 2 -l 2M -t long
Starting axiom-netperf to node 2
   message type: LONG
   payload size: 4096 bytes
   total bytes: 2097152 bytes
   magic number: 174
Sent 2097152 bytes to node 2 in 4.352 s
Throughput bytes/Sec    TX 470.579 (raw 471.039) KB/s - RX 811790.625 (raw 81258
3.389) KB/s
Throughput packets/Sec  TX 0.118 Kpps - RX 202.948 Kpps
root@cotson:~# _
```

RELEASE key: QtKey:1000004 scancode=0x1c

Figure 6 – The "axiom-netperf" utility showing the throughput between two simulated nodes, while transferring AX-IOM-long packets (defined by Partner FORTH).

At this point, we have added also the OmpSs [2], [3], [4], [8], [26] (and GASNET [1]) related packages and tested also the feasibility of running code (again our pipe-cleaner matrix-multiplication benchmark) and, in Figure 7, we can see that correct execution (producing correct result) of a simple run (please note that in such test the timing is not yet actual, but only refers to a particular test model) inside the COTSon simulator.


simnow -e script_slave_1

```
[1497365717.51788]xnic_functional.cpp:197:axiom_netdev_send_packet(): SEND TO AXIOM SWITCH - FD = 25
[1497365717.51847]xnic_functional.cpp:197:axiom_netdev_send_packet(): SEND TO AXIOM SWITCH - FD = 25
[1497365717.90106]xnic_functional.cpp:91:axiom_switch_event_handler(): POLL: some data ready for rea
d...
[1497365717.129714]xnic_functional.cpp:96:axiom_switch_event_handler(): PACKET LENGHT = 146
[1497365717.129723]xnic_functional.cpp:783:axiom_netdev_receive(): CALLING raw_prequeue -> axiom_net
dev_receive_raw
[1497365717.129727]xnic_functional.cpp:182:axiom_netdev_update_irq(): SENDING IRQ TO PCI DEVICE - le
vel: 0X2
INSIDE: Sendind INTERRUPT...
INTERRUPT FIRED!!!
 END POLL: 146
[1497365717.151958]xnic_functional.cpp:182:axiom_netdev_update_irq(): SENDING IRQ TO PCI DEVICE - le
vel: 0X2
INSIDE: Sendind INTERRUPT...
INTERRUPT FIRED!!!
[1497365717.152058]xnic_functional.cpp:1057:axiom_netdev_rx_pop(): -
[1497365717.161722]xnic_functional.cpp:197:axiom_netdev_send_packet(): SEND TO AXIOM SWITCH - FD = 2
5
[1497365717.177762]xnic_functional.cpp:91:axiom_switch_event_handler(): POLL: some data ready for re
ad...
[1497365717.217726]xnic_functional.cpp:96:axiom_switch_event_handler(): PACKET LENGHT = 146
```

simnow -e script

```
[1497365717.217725]xnic_functional.cpp:96:axiom_switch_event_handler(): PACKET LENGHT = 146
[1497365717.217733]xnic_functional.cpp:783:axiom_netdev_receive(): CALLING raw_prequeue -> axiom_netdev_r
eceive_raw
[1497365717.217739]xnic_functional.cpp:182:axiom_netdev_update_irq(): SENDING IRQ TO PCI DEVICE - level:
0X2
INSIDE: Sendind INTERRUPT...
INTERRUPT FIRED!!!
 END POLL: 146
[1497365717.217774]xnic_functional.cpp:91:axiom_switch_event_handler(): POLL: some data ready for read...
[1497365717.217786]xnic_functional.cpp:96:axiom_switch_event_handler(): PACKET LENGHT = 146
[1497365717.217787]xnic_functional.cpp:783:axiom_netdev_receive(): CALLING raw_prequeue -> axiom_netdev_r
eceive_raw
 END POLL: 146
[1497365717.217854]xnic_functional.cpp:182:axiom_netdev_update_irq(): SENDING IRQ TO PCI DEVICE - level:
0X2
INSIDE: Sendind INTERRUPT...
INTERRUPT FIRED!!!
[1497365717.217967]xnic_functional.cpp:1057:axiom_netdev_rx_pop(): -
[1497365717.217980]xnic_functional.cpp:1057:axiom_netdev_rx_pop(): -
[1497365717.220794]xnic_functional.cpp:197:axiom_netdev_send_packet(): SEND TO AXIOM SWITCH - FD = 25
[1497365717.257705]xnic_functional.cpp:91:axiom_switch_event_handler(): POLL: some data ready for read...
[1497365717.297696]xnic_functional.cpp:96:axiom_switch_event_handler(): PACKET LENGHT = 146
```

axiom_switch -r -n2 -v

```
1497365716.966246 axiom_switch.c[402] main() INFO: Packet forwarded - destination sd: 6
1497365716.966265 axiom_switch.c[318] main() INFO: Waiting on poll()...
1497365716.967421 axiom_switch.c[384] main() INFO: New incoming packet - source sd: 5
1497365716.967441 axiom_switch.c[402] main() INFO: Packet forwarded - destination sd: 6
1497365716.967449 axiom_switch.c[318] main() INFO: Waiting on poll()...
1497365716.972970 axiom_switch.c[384] main() INFO: New incoming packet - source sd: 6
1497365717.009688 axiom_switch.c[402] main() INFO: Packet forwarded - destination sd: 5
1497365717.009720 axiom_switch.c[318] main() INFO: Waiting on poll()...
1497365717.009725 axiom_switch.c[384] main() INFO: New incoming packet - source sd: 6
1497365717.009732 axiom_switch.c[402] main() INFO: Packet forwarded - destination sd: 5
1497365717.009735 axiom_switch.c[318] main() INFO: Waiting on poll()...
1497365717.051814 axiom_switch.c[384] main() INFO: New incoming packet - source sd: 5
1497365717.051840 axiom_switch.c[402] main() INFO: Packet forwarded - destination sd: 6
1497365717.051851 axiom_switch.c[318] main() INFO: Waiting on poll()...
1497365717.051855 axiom_switch.c[384] main() INFO: New incoming packet - source sd: 5
1497365717.051865 axiom_switch.c[402] main() INFO: Packet forwarded - destination sd: 6
1497365717.051869 axiom_switch.c[318] main() INFO: Waiting on poll()...
1497365717.090067 axiom_switch.c[384] main() INFO: New incoming packet - source sd: 6
1497365717.090088 axiom_switch.c[402] main() INFO: Packet forwarded - destination sd: 5
1497365717.090099 axiom_switch.c[318] main() INFO: Waiting on poll()...
1497365717.100781 axiom_switch.c[384] main() INFO: New incoming packet - source sd: 6
1497365717.100828 axiom_switch.c[402] main() INFO: Packet forwarded - destination sd: 6
                                                   on poll()...
                                                   ming packet - source sd: 6
                                                   orwarded - destination sd: 6
                                                   on poll()...
                                                   ming packet - source sd: 5
                                                   orwarded - destination sd: 6
                                                   ming packet - source sd: 6
                                                   orwarded - destination sd: 5
                                                   on poll()...
                                                   ming packet - source sd: 6
                                                   orwarded - destination sd: 6
                                                   on poll()...
```

[1] AMD SimNow Main Window — AMD Confidential

Numeric Display(s)

Simulator Stats		IDE Primary Display		IDE Secondary Display		Diagnostic Port
76.32	Host Seconds	101,140,480	master read	0	master read	00 00 96
241.80	Sim Seconds	5,816,320	master written	0	master written	00 00 00
127	Avg MIPS	0	slave read	0	slave read	00 00 00
338	MIPS	0	slave written	0	slave written	
		DMA/PIO	mode	PIO/PIO	mode	

```
root@cotson:~# _
```

RELEASE key: QtKey:1000004 scancode=0x1c

[1] AMD SimNow Main Window — AMD Confidential

Numeric Display(s)

Simulator Stats		IDE Primary Display		IDE Secondary Display		Diagnostic Port
75.97	Host Seconds	102,344,704	master read	0	master read	00 00 96
235.11	Sim Seconds	6,033,408	master written	0	master written	00 00 00
130	Avg MIPS	0	slave read	0	slave read	00 00 00
0.77	MIPS	0	slave written	0	slave written	
		DMA/PIO	mode	PIO/PIO	mode	

```
02:[54:14.647946]: End matmul   0x1000000 0x1007530 0x105f370 elapse=8
01:[54:22.627928]: Start matmul 0x7f153123a460 0x7f15311af010 0x7f153118a460
01:[54:22.635928]: End matmul   0x7f153123a460 0x7f15311af010 0x7f153118a460 ela
pse=8
01:[54:22.635928]: Start matmul 0x7f1531241990 0x7f15311af010 0x7f1531191990
01:[54:22.639928]: End matmul   0x7f1531241990 0x7f15311af010 0x7f1531191990 ela
pse=4
01:[54:22.639928]: Start matmul 0x7f15312503f0 0x7f15311af010 0x7f15311a03f0
01:[54:22.643928]: End matmul   0x7f15312503f0 0x7f15311af010 0x7f15311a03f0 ela
pse=4
01:[54:22.643928]: Start matmul 0x7f1531257920 0x7f15311af010 0x7f15311a7920
01:[54:22.647928]: End matmul   0x7f1531257920 0x7f15311af010 0x7f15311a7920 ela
pse=4
02:[54:14.679946]: Start matmul 0x10668a8 0x1007530 0x106ddd8
02:[54:14.687946]: End matmul   0x10668a8 0x1007530 0x106ddd8 elapse=8
01:[54:22.771928]: Calculating matrices done!
01:[54:22.779928]: Min matmul              4 msec
01:[54:22.779928]: Mean matmul             5 msec
01:[54:22.779928]: Max matmul              8 msec
01:[54:22.779928]: Total matmul           60 msec
01:[54:22.779928]: Total elab time       180 msec
01:MSG: [0] Node 1 executed 2 WDs
01:MSG: [?] Cluster Soft invalidations: 0
01:MSG: [?] Cluster Hard invalidations: 0
root@cotson:~# _
```

RELEASE key: QtKey:1000004 scancode=0x1c

Figure 7 – The OmpSs Matrix-Multiplication benchmark running in the COTSon simulator and using two nodes connected by the modeled AXIOM-link and network protocol. OmpSs is relying on GASNET and in turn relying on the AXIOM-link.

In summary, the implemented models where completed in tasks T7.2 and T7.3 allowing a smoother migration of the related codes into the AXIOM-board, by following our methodology. The related software is at the following links (clonable git repositories):

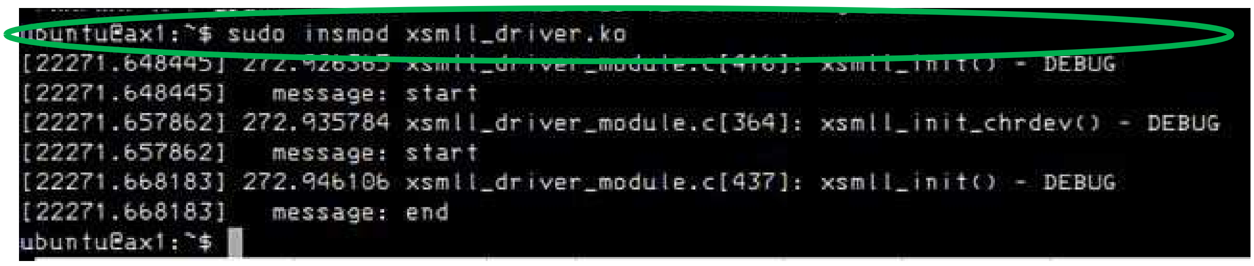
- XNIC/XSMLL device (Figure 3):
  https://git-nda.axiom-project.eu/xnic-simnow-device
  (The above link is not public as it is using an SDK under NDA agreement with AMD).
- Testing scripts for the drivers (Figure 4, Figure 5, Figure 6):
  https://git-private.axiom-project.eu/xnic-utils/
- Device drivers (Figure 4):
  https://git-private.axiom-project.eu/axiom-xnic-pci-drv/

Also, here are other components visualizable via a web interface (browser):

- OmpSs program for matrix-multiplication (Figure 7):
  https://git.axiom-project.eu/axiom-evi/blob/master/tests/ompss/src/ompss_evimm.c
- Three-levels memory allocator (see also D5.2):
  https://git.axiom-project.eu/axiom-evi-nic/axiom-x86-pci-ettore/

2.3 Porting from the simulator to the AXIOM-board

Once the features have been tested on the simulator, we obtained the double advantage of a preliminary prove of functionality (the efforts were started before the availability of the AXIOM-board), and also more flexibility of running the software not only on ARM-based systems but also on X86_64 based systems. For example, in Figure 8, we show the driver for XSMLL has been ported from the simulator to the AXIOM-board and it is loading successfully on our board. Please note that in the simulator, for convenience, we had a single driver for the network interface and for the XSMLL, while on the AXIOM-board it was more efficient to separate the two drivers.



```
ubuntu@ax1:~$ sudo insmod xsmll_driver.ko
[22271.648445] 272.926365 xsmll_driver_module.c[416]: xsmll_init() - DEBUG
[22271.648445]   message: start
[22271.657862] 272.935784 xsmll_driver_module.c[364]: xsmll_init_chrdev() - DEBUG
[22271.657862]   message: start
[22271.668183] 272.946106 xsmll_driver_module.c[437]: xsmll_init() - DEBUG
[22271.668183]   message: end
ubuntu@ax1:~$
```

Figure 8 – Loading the driver for XSMLL on the AXIOM-board.

The corresponding software components are available here:

- XSMLL driver for AARCH64 (Figure 8) – clonable git repository:
  https://git-private.axiom-project.eu/XDRIVER/
- Porting of XSMLL runtime on AARCH64 (web link):
  https://git-private.axiom-project.eu/XSMLL/arm64/

2.4 From the functional model to the timing model

The timing models in the simulator have been constantly refined and rapidly checked in parallel to the activity of porting the corresponding architectures into the RTL designs through the Xilinx Vivado [69]. In Figure 9, we continue to show, as an example, how the top-level view of the XSCHEDULE command as it is defined in the Xilinx Vivado tools.

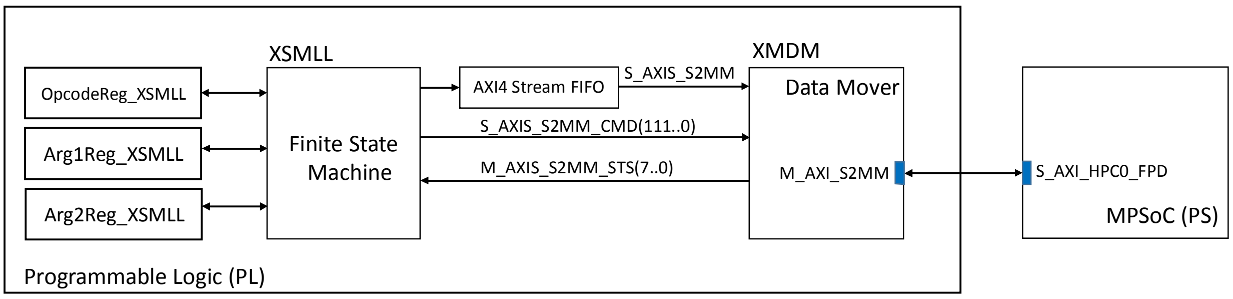


Figure 9 – Schematic of the timing model as ported in the Xilinx Vivado tools (top-level view). The picture refers the XSCHEDULE command, following the same initial example.

As a further step the internal structure of the modules of the XSCHEDULE operation (and similarly for the other XSMLL commands) have been detailed (Figure 10) by following the previously defined timing model that was tested in the simulator.

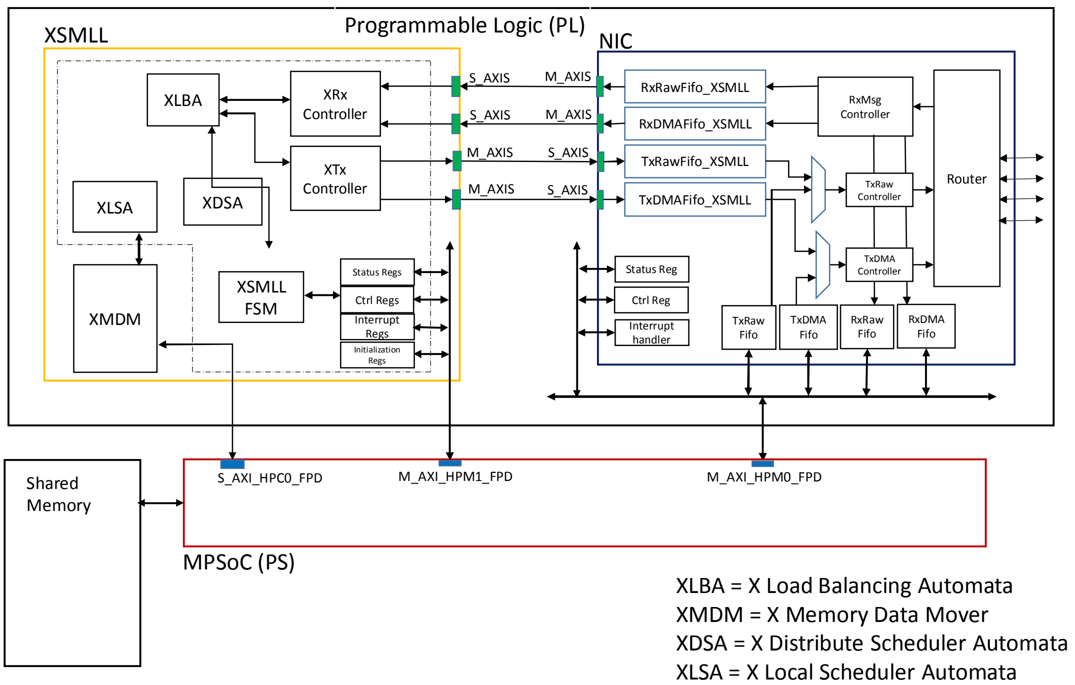


Figure 10 – More detailed view of the XSCHEDULE implementation in the FPGA.

For example, the finite state machine "XSMLL FSM" of Figure 10 is implemented as shown in Figure 11.

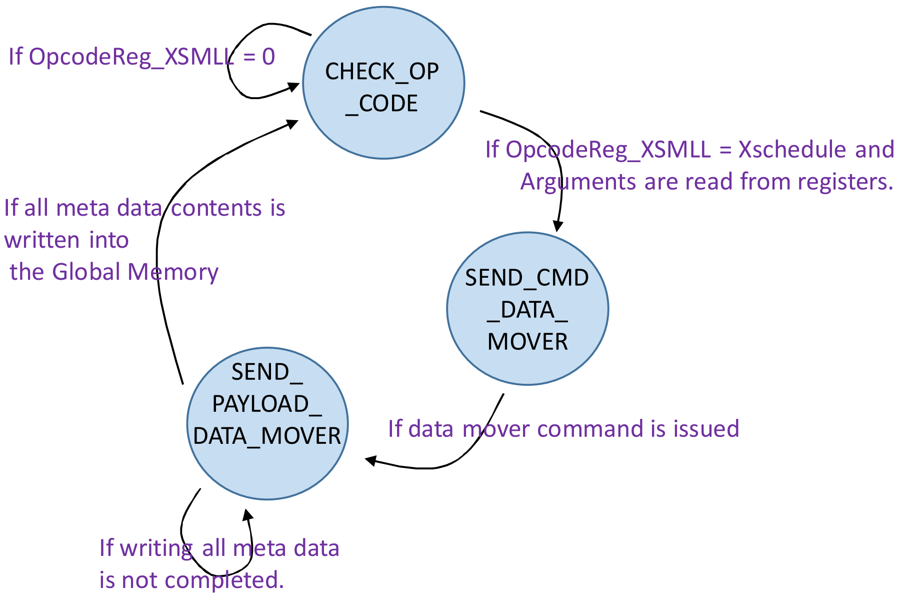
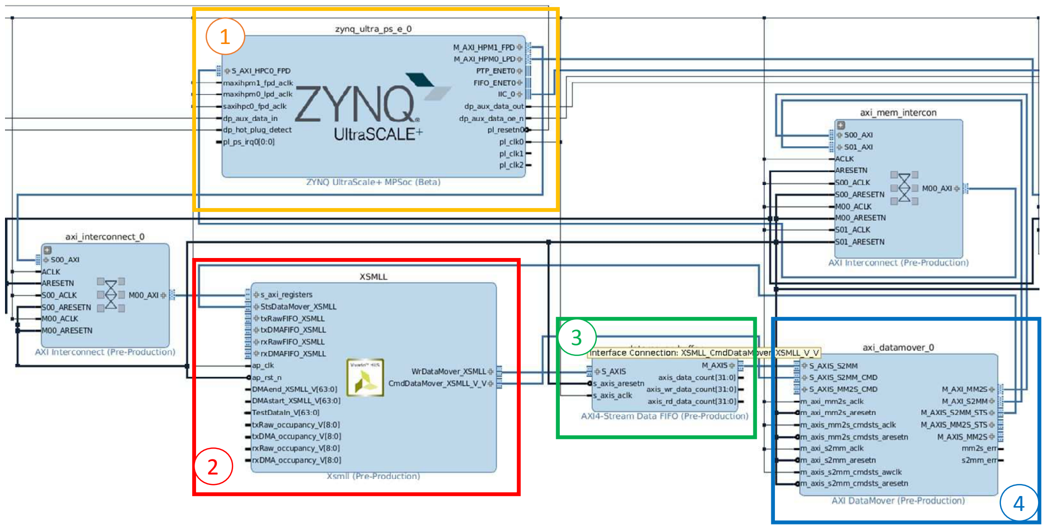

If OpcodeReg_XSMLL = 0
CHECK_OP _CODE
If OpcodeReg_XSMLL = Xschedule and Arguments are read from registers.
SEND_CMD _DATA_ MOVER
If all meta data contents is written into the Global Memory
SEND_ PAYLOAD_ DATA_MOVER
If data mover command is issued
If writing all meta data is not completed.

Figure 11 – The finite state machine (FSM) that checks the opcode of the XSCHEDULE command.

The module for XSMLL is then integrated in the rest of the hardware design still using the Vivado graphical interface as shown in Figure 12

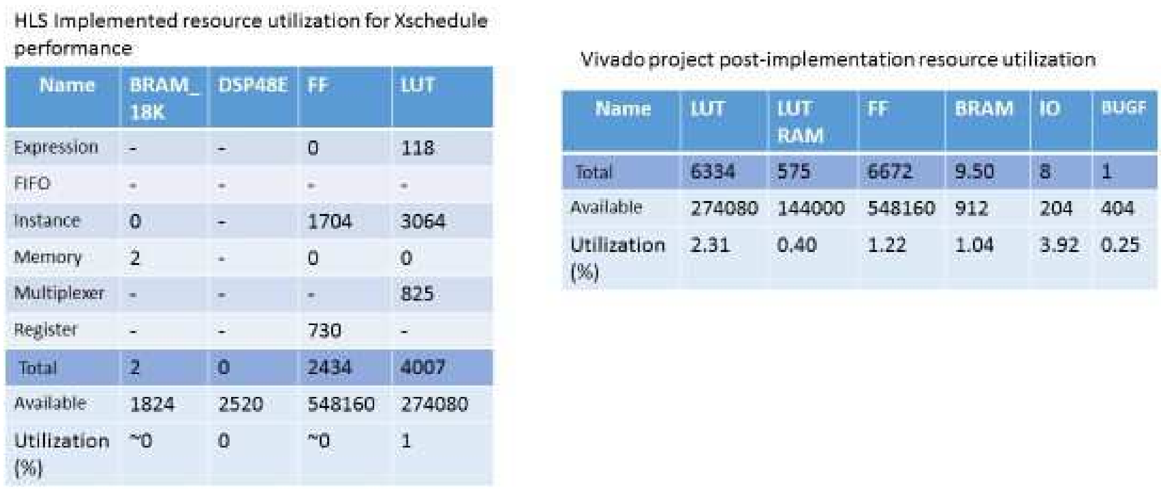
Hardware included to perform Xschedule:


1
zynq_ultra_ps_e_0
ZYNQ UltraSCALE+
ZYNQ UltraScale+ MPSoc (Beta)
axi_mem_intercon
AXI Interconnect (Pre-Production)
axi_interconnect_0
AXI Interconnect (Pre-Production)
XSMLL
Xsmll (Pre-Production)
2
3
AXI4-Stream Data FIFO (Pre-Production)
axi_datamover_0
AXI DataMover (Pre-Production)
4

1 Processing System (PS) 2 XSMLL module 3 AXI4 Stream FIFO 4 Data Mover

Figure 12 – The XSMLL soft-IP as it appears in the schematic entry tool of Xilinx Vivado.

The results of the synthesis are shown in Figure 13, as we can see the occupation of the FPGA resources for the XSCHEDULE operation is very reasonable since it is about 4000 LUTS, 2 BRAM blocks and about 2400 Flip-Flops.



HLS Implemented resource utilization for Xschedule performance

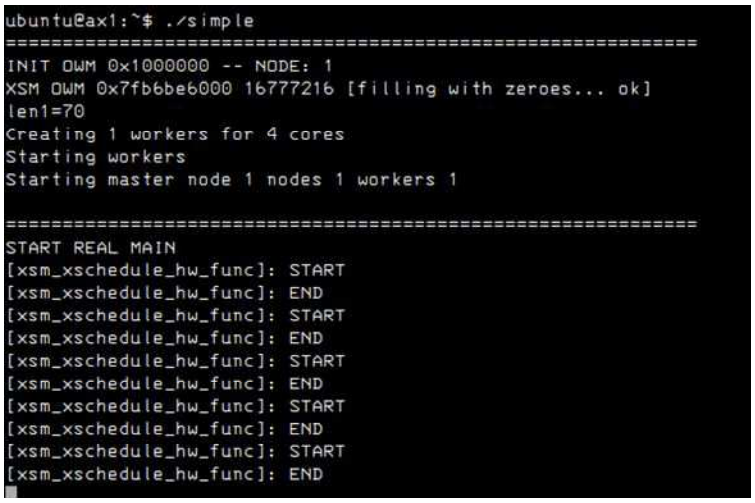
Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	118
FIFO	-	-	-	-
Instance	0	-	1704	3064
Memory	2	-	0	0
Multiplexer	-	-	-	825
Register	-	-	730	-
Total	2	0	2434	4007
Available	1824	2520	548160	274080
Utilization (%)	~0	0	~0	1

Vivado project post-implementation resource utilization

Name	LUT	LUT RAM	FF	BRAM	IO	BUGF
Total	6334	575	6672	9.50	8	1
Available	274080	144000	548160	912	204	404
Utilization (%)	2.31	0.40	1.22	1.04	3.92	0.25

Figure 13 – Results of the synthesis of the XSMLL module when including the XSCHEDULE command.

Once the architecture had been implemented and synthesized, we tested it in combination with the hardware components. For example, after some analysis about the possible method to allocate fixed chunks of memory (frames), within the XSCHEDULE operation, we preferred to migrate the functionality to software instead of having it in hardware.



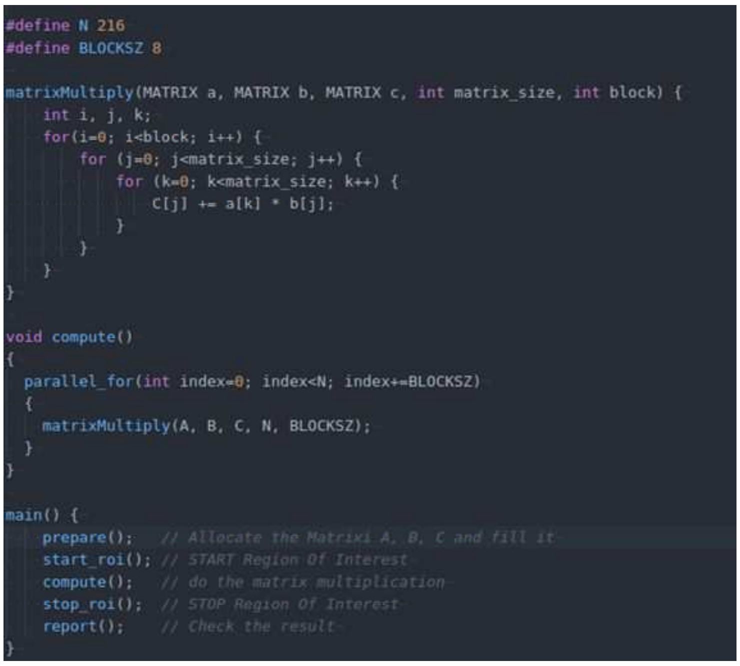
```
ubuntu@ax1:~$ ./simple
==========================================================
INIT OWM 0x1000000 -- NODE: 1
XSM OWM 0x7fb6be6000 16777216 [filling with zeroes... ok]
len1=70
Creating 1 workers for 4 cores
Starting workers
Starting master node 1 nodes 1 workers 1

==========================================================
START REAL MAIN
[xsm_xschedule_hw_func]: START
[xsm_xschedule_hw_func]: END
[xsm_xschedule_hw_func]: START
[xsm_xschedule_hw_func]: END
[xsm_xschedule_hw_func]: START
[xsm_xschedule_hw_func]: END
[xsm_xschedule_hw_func]: START
[xsm_xschedule_hw_func]: END
[xsm_xschedule_hw_func]: START
[xsm_xschedule_hw_func]: END
```

Figure 14 – Testing at the application level of the XSCHEDULE operation as implemented in the FPGA.

2.5 Performance evaluation of the XSMLL

We used the following algorithm (Matrix-Multiplication with block size multiplication of N/B stripes where N is the size of the matrix and B is the block size). In our tests, we used the size of 8 elements as we did not notice major difference when using sizes of 4, 16, 32 elements for the block.

In Figure 15, we reported a C-like pseudo-code of the multiplication algorithm that we have used, where we show also the point of measurement "`start_roi()`" and "`end_roi()`" so that the measurements precisely relate to the Region of Interest (ROI) and exclude initial preparation and reporting phases (as normally done in performance evaluation methodologies).



```
#define N 216
#define BLOCKSZ 8

matrixMultiply(MATRIX a, MATRIX b, MATRIX c, int matrix_size, int block) {
    int i, j, k;
    for(i=0; i<block; i++) {
        for (j=0; j<matrix_size; j++) {
            for (k=0; k<matrix_size; k++) {
                C[j] += a[k] * b[j];
            }
        }
    }
}

void compute()
{
  parallel_for(int index=0; index<N; index+=BLOCKSZ)
  {
    matrixMultiply(A, B, C, N, BLOCKSZ);
  }
}

main() {
    prepare();   // Allocate the Matrixi A, B, C and fill it
    start_roi(); // START Region Of Interest
    compute();   // do the matrix multiplication
    stop_roi();  // STOP Region Of Interest
    report();    // Check the result
}
```

Figure 15 – C-like pseudo-code of the multiplication algorithm that we have used.

By using the simulator, we have obtained the following result of the evaluation, regarding the execution times, percentage of kernel time over total time, L2- and L3-miss rate (Figure 16).

AXIOM DF-THREADS

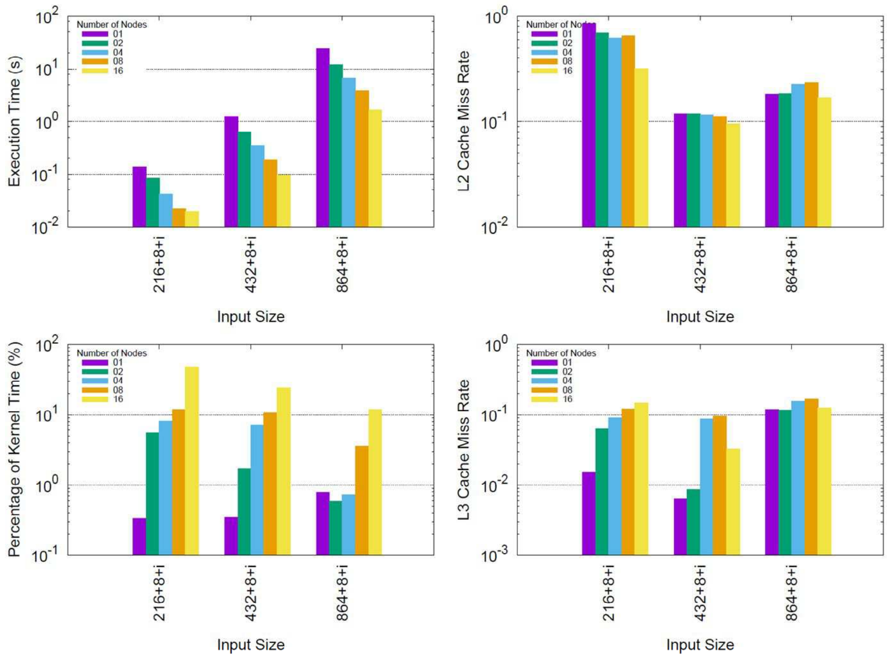


Figure 16 – Execution times, percentage of kernel time over total time, L2- and L3-miss rate of XSMLL/DF-Threads. The benchmark is Matrix-Multiplication. The input size labels represent <matrix_size>+<block_size>+<data_type>, e.g., 216x216 for the matrix size, 8 for the block size, "I" for integer on 64 bits.

These results had been compared with the OpenMPI [64], Cilk, and JUMP (DSM) [41], [63]. We discuss here a comparison with OpenMPI (Figure 17).

OPENMPI

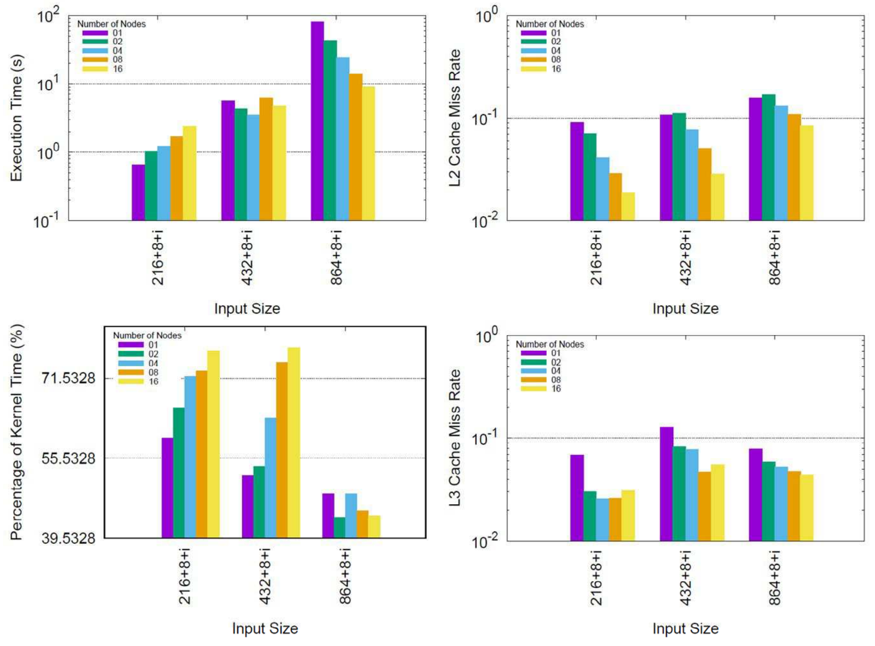


Figure 17 – Execution times, percentage of kernel time over total time, L2- and L3-miss rate of OpenMPI. The benchmark is Matrix-Multiplication. The input size labels represent <matrix_size>+<block_size>+<data_type>, e.g., 216x216 for the matrix size, 8 for the block size, "I" for integer on 64 bits.

In these experiments, we used the XSMLL/DF-Threads version 103 and the OpenMPI version 3.0. As we can see from Figure 16 and Figure 17 the execution time achieved with XSMLL/DF-Threads are about 1 order of magnitude better than those achieved in OpenMPI. We still see a major opportunity of improvement of XSMLL since the miss rate in L2 and L3 is – on the other side – about 1 order of magnitude worse than in OpenMPI. In fact, other tests (not shown here) where we just increase the cache size show that XSMLL further improves its performance while OpenMPI does not.

Another observation is related to the kernel activity, which is conditioning the performance of OpenMPI for smaller input sizes (e.g., 216): in this case the kernel activity ranges from about 60 to 75% of the time and this may represent some difficulty in the connection through the Ethernet and TCP/IP stack when using the simulator. On the other side the results for larger sizes (e.g. 864) confirm quite well the expected behavior.

In conclusion, these results confirm the capability of XSMLL/DF-Threads of performing much better than standard programming models for distributed computing such as MPI, while maintaining the capability of providing a transparent solution to distribute the execution that can be interfaced to the applications. In our case, we showed in the previous reporting period (see D4.2) how XSMLL can be interfaced directly and transparently to OmpSs.

2.6 Comparing the simulator and the AXIOM-board performance

As observed at the beginning of the project (see D7.1) "previous studies discovered that the performance and power studies of ARM and x86 do not depend on the specific ISA that is used [17] but rather they depend on the type and quantity of resources that are included in each platform (i.e., bus-widths, cache size, cache organization, instruction window size and similar)". In particular, other researchers in [17]: "We find that ARM and x86 processors are simply engineering design points optimized for different levels of performance, and there is nothing fundamentally more energy efficient in one ISA class or the other. The ISA being RISC or CISC seems irrelevant". While AMD is planning to extend their processors to ARM [18], [19], we present below some actual data coming from the AXIOM-board that is confirming the x86_64 simulator and the ARM-board can be tuned to perform quite closely (Figure 18).

As we can see from Figure 18, despite some architectural differences, the results of the simulator match quite closely the performance trends (and values), confirming our performance predictions.

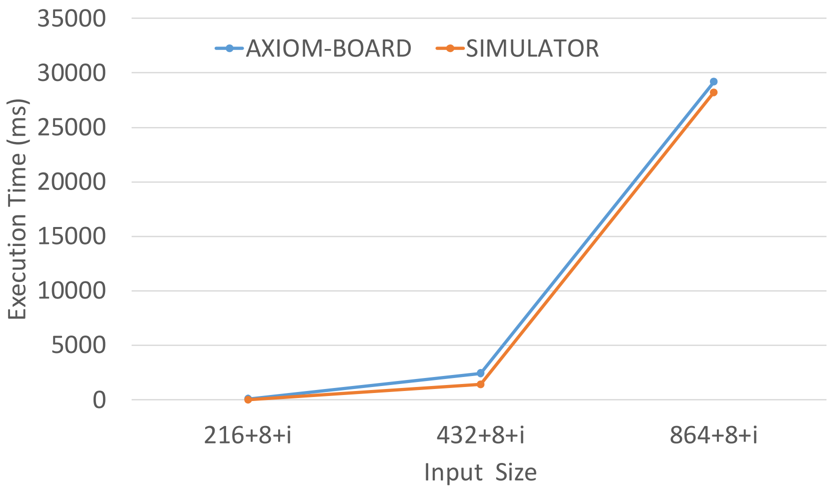


Figure 18 – Comparison of execution time (sequential execution) between the simulator (COTSon) and the actual AXIOM-board (revision B). The benchmark is Matrix-Multiplication. The input size labels represent <matrix_size>+<block_size>+<data_type>, e.g., 216x216 for the matrix size, 8 for the block size, "I" for integer on 64 bits.

3 Optimizing the AXIOM Network Interface for higher performance and flexibility

In D6.5, Partner FORTH completed the design of the basic functionalities of the AXIOM Network Interface (also called AXIOM-link or simply NIC or NI); this was the priority to enable the other Partners by month 30 of the project (July 2017) in order to make tests and evaluations. As a further step of better integration, Partner FORTH greatly supported the other Partners in order to seamlessly integrate their NIC design with the other soft-IP. Moreover, in conjunction with Partner UNISI, the design has been integrated and tested together with the XSMLL soft IPs (Section 2) and the AVR soft-IP (see D5.1, D6.4).

Even more, during FORTH has constantly monitored the performance capabilities of the NIC and in the last months was able to improve NIC features towards enhancing network throughput and latency.

For convenience, Figure 19 provides the NIC implementation, including the local router. In short, a set of memory-mapped FIFOs allow the local OS to post message descriptors for raw data / neighbor data transfers, or RDMA / long data requests. The NI control and status registers allow the local OS to configure the NI / check the interconnect status respectively, as described in D6.5.

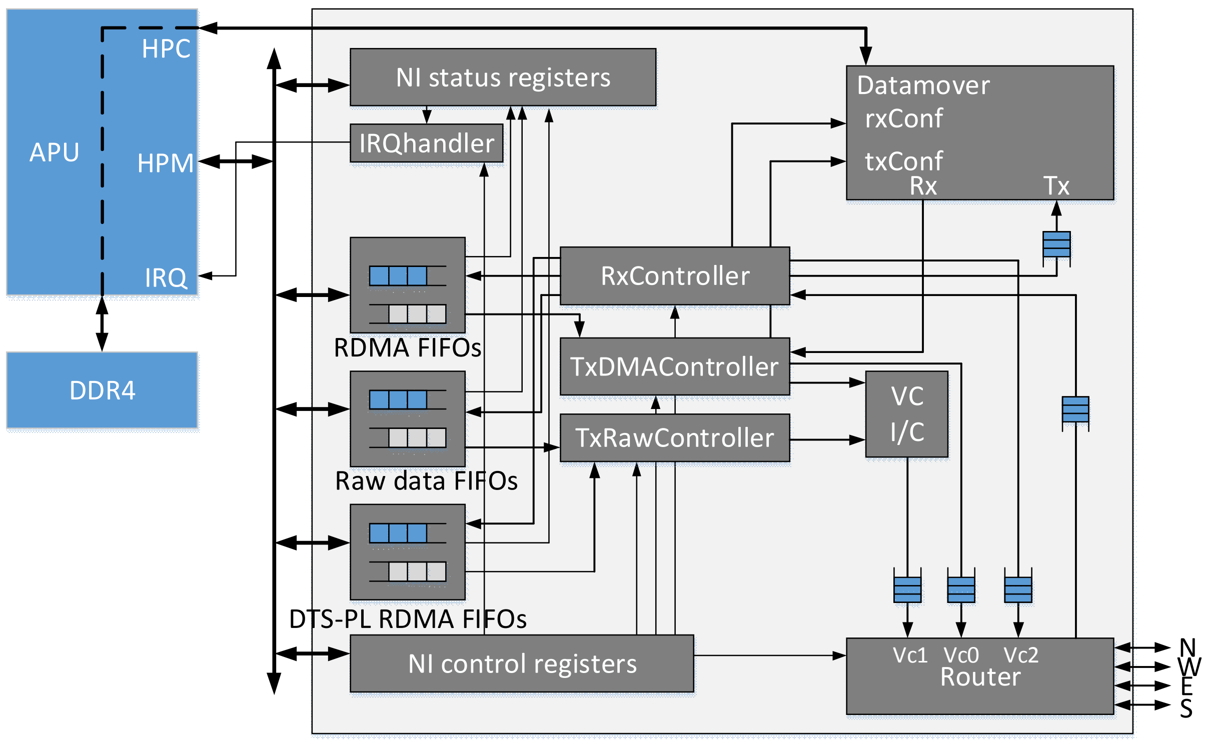


Figure 19 - NIC module implementation, including the local router.

The TxRawController and TxDMAController modules are responsible for parsing posted raw data and RDMA descriptors. Raw data are packetized and directly forwarded to the router; RDMA write / long data requests are parsed by the TxDMAController to configure the local Datamover for reading all requested data from the local DDR4 memory. Data are packetized and transmitted to the local router. RDMA reads generate the corresponding packet that will ask the destination node to send back all requested data. Finally, the IRQ handler notifies the local OS when the FIFOs state changes, in order to trigger the corresponding software routine for reading any pending descriptors.

3.1 Summary of the most important improvements of the NIC

Below we provide the additional NIC features that were implemented towards enhancing network throughput and latency:

- **Bi-directional links**

As stated in D6.5, we used the Aurora PHY IP [65] to connect the NIC logic with the FPGA hardware high-speed serial transceivers. The first NIC version supported single-direction data transmission, whereas Xon/Xoff-related buffer status was transmitted using the opposite link direction. In this version, we used the Aurora sideband channels for transmitting Xon/Xoff-related buffer status, thus allowing the implementation of bi-directional data links. Overall, the latest NIC version supports data transmission over the main channel, and Xon/Xoff-related buffer status over the sideband channels. Furthermore, with bidirectional links, only one connector is needed to connect two nodes, allowing for more complex topologies. It should be noted that this feature does not reduce throughput.

- **Reversible USB type-C based connectors**

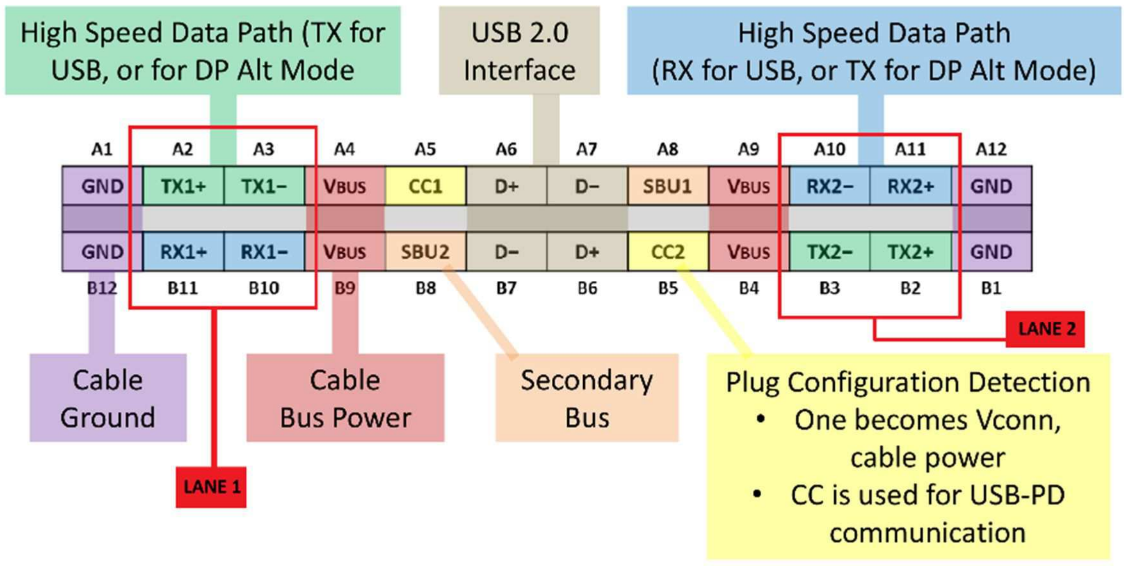

High Speed Data Path (TX for USB, or for DP Alt Mode
USB 2.0 Interface
High Speed Data Path (RX for USB, or TX for DP Alt Mode)
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12
GND TX1+ TX1− VBUS CC1 D+ D− SBU1 VBUS RX2− RX2+ GND
GND RX1+ RX1− VBUS SBU2 D− D+ CC2 VBUS TX2− TX2+ GND
B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1
LANE 2
LANE 1
Cable Ground
Cable Bus Power
Secondary Bus
Plug Configuration Detection
One becomes Vconn, cable power
CC is used for USB-PD communication

Figure 20 - USB type-C pins layout.

The latest version of the NIC supports the full USB Type C connector functionality, including **reversibility**, allowing the user to connect AXIOM boards easily. To implement reversibility, we use the orientation pins from each interface (CC1 – A5 and CC2 – B5, shown in Figure 20), and reconstruct the packets depending on the way the lanes of the transmitter and receiver are connected.

Figure 21 shows two AXIOM boards interconnected over a single bi-directional link.

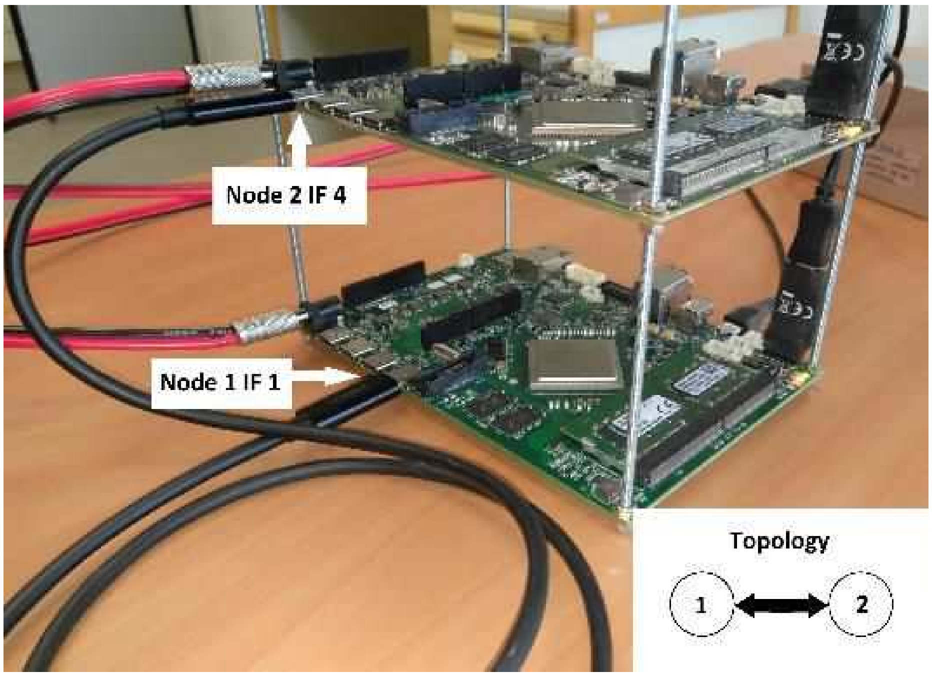


Figure 21 - AXIOM boards interconnected with one bi-directional link.

- **Minimal reconfigurable resource utilization**

Table 1 - NIC resource utilization to an xczu9eg MPSoC.

Resource primitive	% utilization to xczu9eg MPSoC
LUTs	15.61
FFs	10.67
GTH transceivers	33.33
BRAMs	20

Table 1 provides the NIC resource utilization on the xczu9eg MPSoC, used in all AXIOM boards. As observed, the NIC requires less than 16% of the available programmable resources. This allows system developers to instantiate it even to low-range MPSoCs, facilitating the deployment of inexpensive multi-CPS processing platforms.

- **64bit / 128bit interconnect links**

As stated, each USB type-C cable includes two 10Gbps lanes. In the latest NIC version we used both available 10Gbps lanes of the connector to establish a 20Gbps link between two AXIOM boards. Internally, the NIC uses 128bit AXI4 Stream signals to interface the Aurora PHY IP, hence effectively doubling the throughput. To do that we pair both available GTH transceivers per USB Type C connector to a single channel consisting of two lanes.

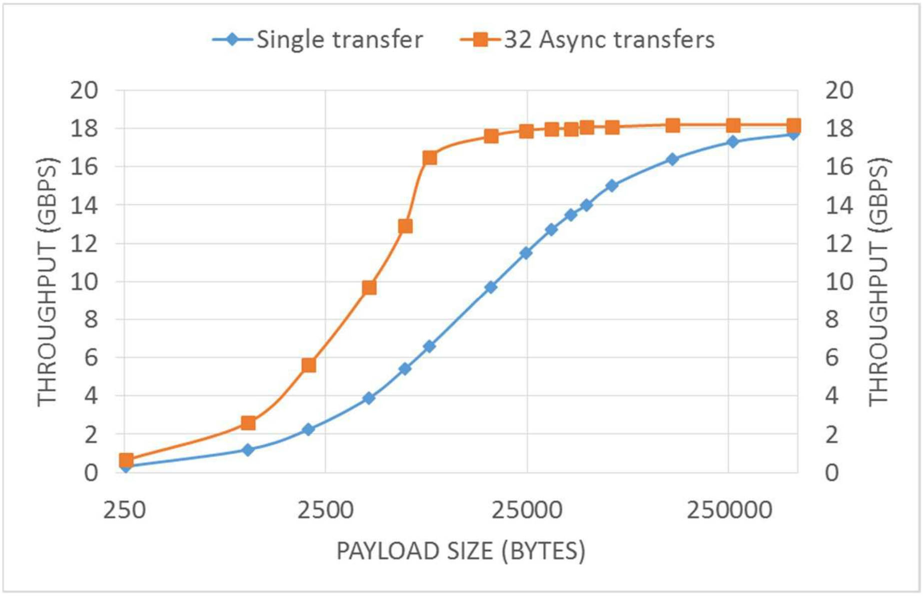


Figure 22 - RDMA write / read throughput between two nodes for a single and 32 transfers.

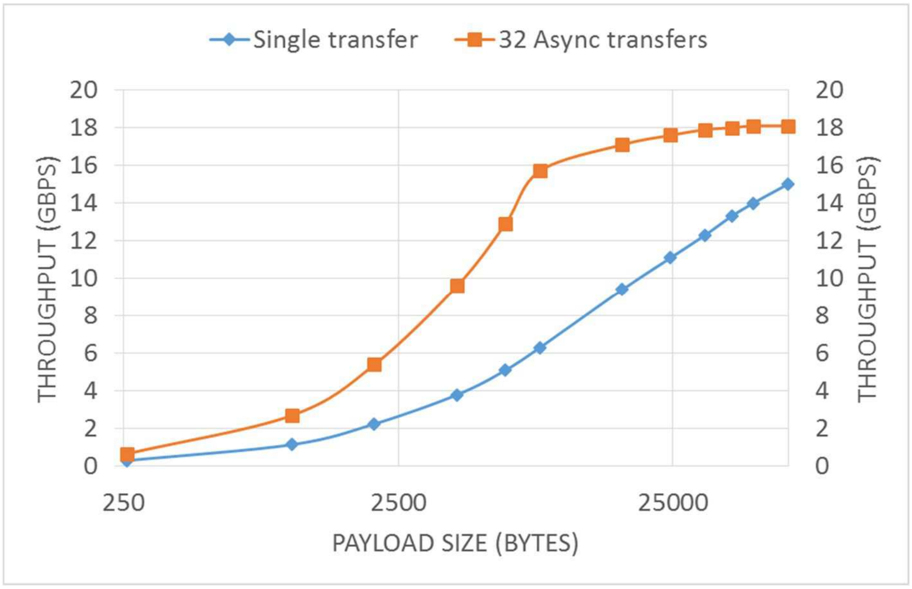


Figure 23 – Long data throughput between two nodes for a single and 32 transfers.

To isolate any OS-related overheads, all experiments were done using a bare-metal software configuration that executed single and 32-asynchronous transfers. D6.5 has already provided experimental results when using a 64bit link. Now, Figure 22 and Figure 23 provide the measured throughput of RDMA write/read and long data descriptors respectively when using a 128bit link. As observed, as the payload size increases, the link pipeline utilization is also increased, leading to a throughput of up to 18Gbps. In other words, the link effective utilization is close to 90%.

Our implementation is similar to the way the new USB 3.2 Type C specification. Unfortunately this means that not all Type C SuperSpeed cables will work with the 128bit design. For this reason, AXIOM boards will also support 64bit links with a maximum bandwidth of 10 Gbps, supporting though most currently available USB type-C cables in the market.

Finally, in Figure 24, we show a partial snapshot of the schematic of the NIC to five an idea of the actual complexity of the design.

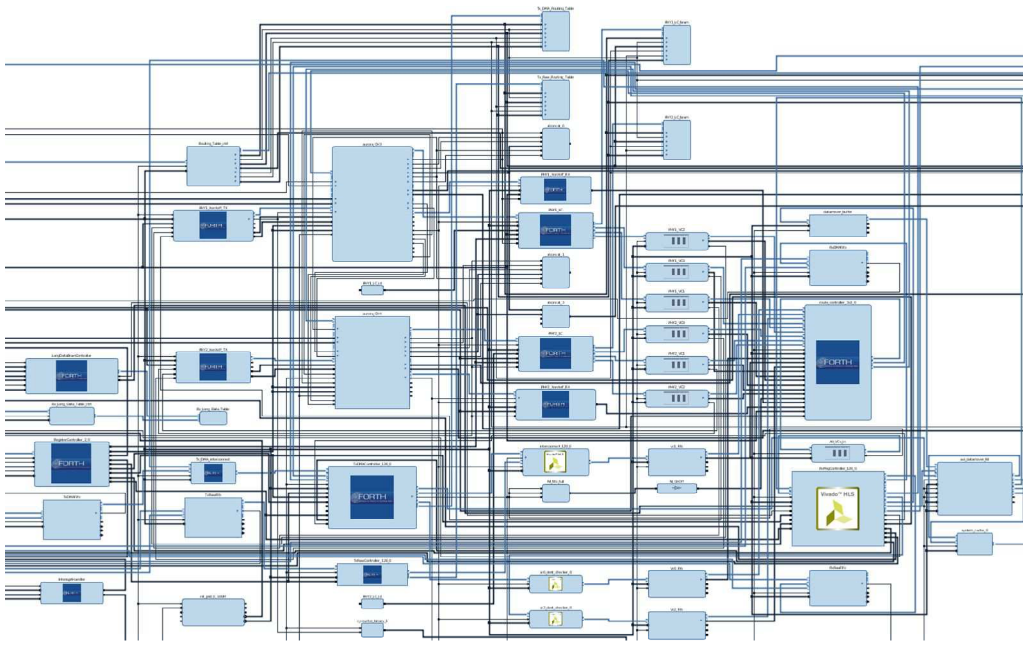


Figure 24 – (Partial) snapshot of the schematic of the NIC inside the AXIOM project in the Xilinx Vivado tools.

3.2 Impact of the achievement on project/WP

Table 2 provides the impact of the NIC achievements on the project.

Table 2 - AXIOM NIC achievements.

Achievement	Impact
Support of raw data	Raw data allow AXIOM nodes to exchange low-latency short messages.
Facilitating RDMA requests	Facilitating RDMA requests at hardware level, boosts the performance of applications that require large memory space (memory-bound applications), distributed over different AXIOM nodes.
Low resource utilization	The AXIOM NIC can be implemented even to low-range MPSoCs with constrained amount of programmable logic resources, allowing the deployment of cost-effective multi-CPS platforms.
Bi-directional links	Bi-directional links allow the deployment of efficient AXIOM networks with irregular topologies.
Reversible USB type-C based connectors	Reversible connectors allow users to easily connect multiple AXIOM nodes.
64bit / 128bit interconnect links	A 64bit NIC version allow users to cascade multiple AXIOM nodes with inexpensive cables, whereas a 128bit version will support node-to-node bandwidths up to 20 Gbps, over SuperSpeed USB type-C 3.2 cables.

4 FPGA Design integration

One important goal of this workpackage is to integrate the contribution of the Partners developing separately some of the soft-IPs (Figure 25). In particular, we arrived to the conclusion to use the git versioning system to keep track and easily integrate the contributions from the several Partners in a single project. In Figure 26 and Figure 27, we show a snapshot of the schematic of the XSMLL (see Section 2) and the AVR (see D5.1 and D6.4 – AVR is the processor that supports the Arduino socket [22], [25], [54], [61]) to give an idea of the soft-IP parts. The corresponding schematic for the NIC is in Figure 24.

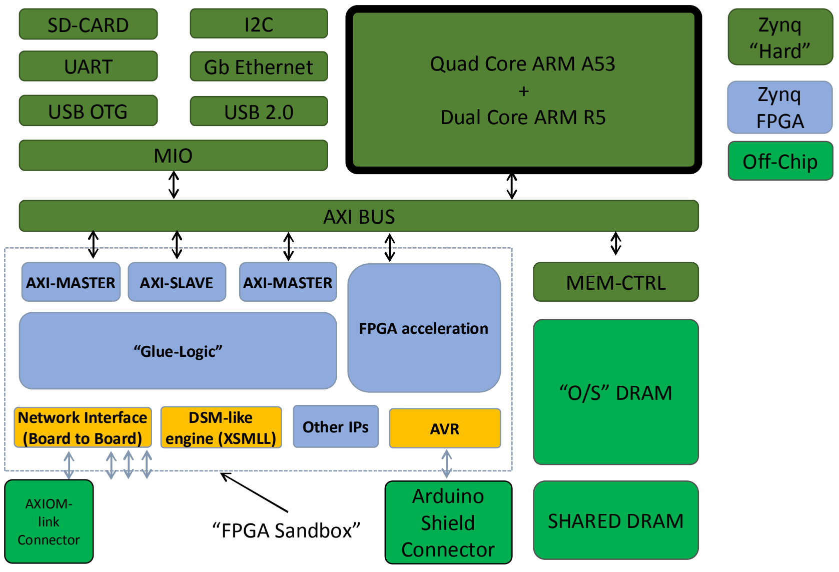


Figure 25 – Overall view of the AXIOM-board SoC (Zynq UltraScale+). AXIOM soft-IPs are highlighted in orange.

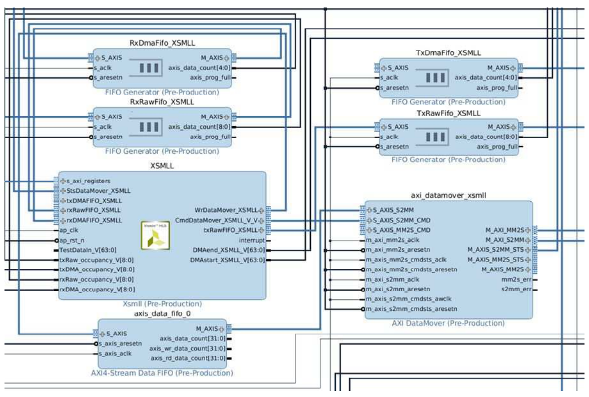


Figure 26 – XSMLL related soft-IP.

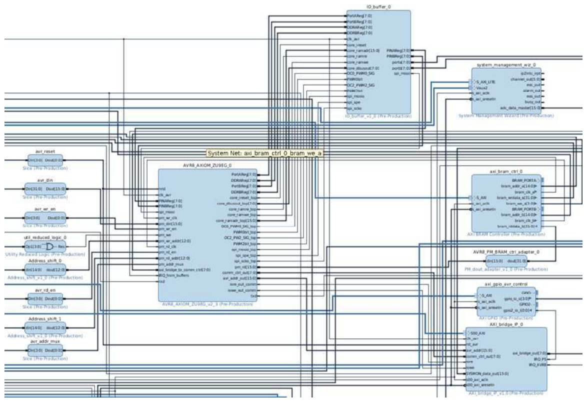


Figure 27 – AVR related soft IP.

Those soft-IPs have been made available to the Partners through the following git repository (this software is only accessible internally to the AXIOM Consortium and EU Commission services):

WEB: https://git3.axiom-project.eu/AXIOM_VIVADO/

GIT: git clone https://git3.axiom-project.eu/AXIOM_VIVADO/

In Figure 28, we show a partial snapshot of the overall integration of the several soft-IPs. In order to make easier to the AXIOM Partners to integrate the IPs, we decided to have a single Vivado project and rely on the excellent capabilities of the git versioning system.

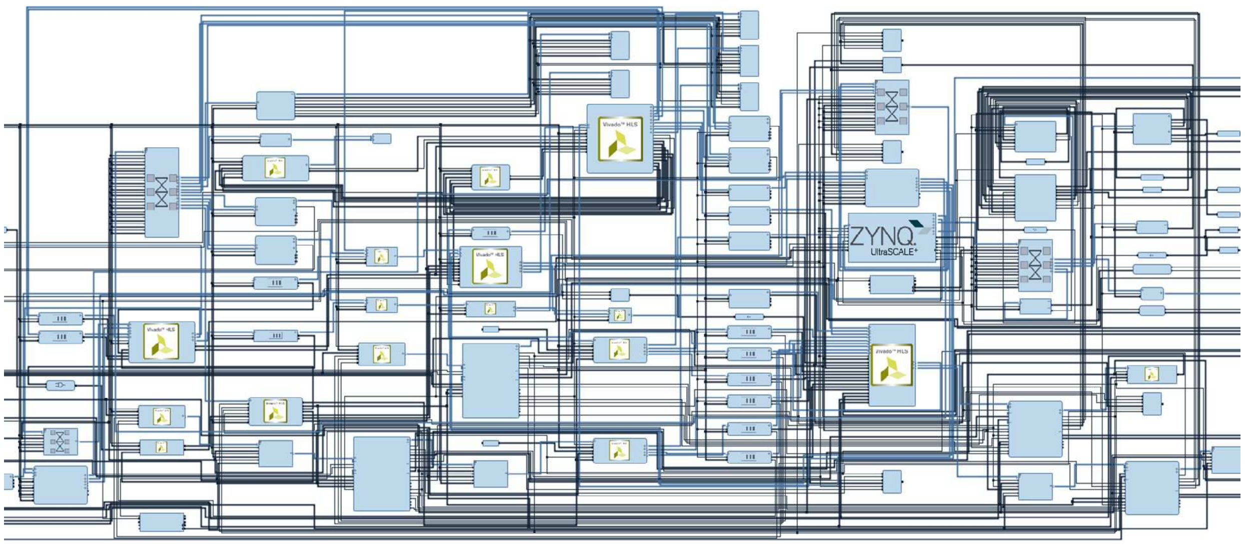


Figure 28 – (Partial) snapshot of the overall integration of the AXIOM soft-IPs.

This choice has enabled a much better workflow between the contributing AXIOM Partners.

As a final conclusion, as outlined in Figure 2, through the XGENIMAGE tool developed in collaboration between Partner SECO and UNISI (see D5.4) we generated automatically an “”SD-IMAGE”. XGENIMAGE is an extension of the GENIMAGE tool, that was designed to provide and “SD-IMAGE” for the COTSon full-system simulator. This enables the final integration of the software stack, the bit-streams of the soft-IPs, the kernel, drivers and everything to immediately use the AXIOM-board. The official AXIOM SD-image is publicly available here:

https://download.axiom-project.eu/IMAGES/XOS_v0.8_20180112.tar.bz2

The instructions on how to run the system has been illustrated in D2.3 and made public here:

http://www.axiom-project.eu/get-started/

5 Enabling AXIOM board power-consumption measurements

Partner SECO designed the AXIOM-board by following the request of the AXIOM Consortium to have a reliable and accurate method for monitoring the power consumption of the system. This requirement is crucial to enable optimizations towards a reduction of the power consumption. Therefore, we also preferred to focus on measurements of power on the actual board instead of those on simulator [20].

More specifically, eight INA219 power monitor integrated circuits have been adopted as reported by Table 3 to monitor power consumption on its most important power supply rails.

Those power monitoring ICs are connected with processing system within Zynq UltraScale+ MPSoC on the AXIOM board using the I2C bus, for additional information on INA219 refer to [66]

AXIOM board’s Linux distribution allows power consumption monitoring using INA219s through SYSFS and IOCTL interfaces. On top of that, monitoring AXIOM board’s power consumption, split for each supply rail, is quite straightforward as shown by [67]. This approach allows live monitoring of power consumption of the main system’s components while running applications.

Table 3 - AXIOM board's power supply rail adopting dedicated power monitors

Power supply rail	Label	Nominal Voltage [V]	System ID[1]	Description
0.85V_INTFP	PM_VCC_INTFP	0.85	0	PS full-power domain supply voltage
0.85V_VCCINT	PM_VCC_INT	0.85	1	PL internal power supply
12V_ALW	PM_VIN	12	2	VIN power supply
0.85V_INTFP_DDR	PM_INTFP_DDR	0.85	3	PS DDR controller and PHY supply voltage
1V2_DDR_PS	PM_1V2_DDR_PS	1.2	4	PS DDR supply
1.2V_DDR_PL	PM_1V2_DDR_PL	1.2	5	PL DDR supply
MGTAVCC	PM_MGTAVCC	0.9	6	Analog supply voltage for GTH transceiver
1.2V_MGTAVTT	MGTAVTT	1.2	7	Analog supply voltage for GTH transmitter and receiver termination circuits

To demonstrate power monitoring capabilities of the AXIOM board, this section describes measurements that were performed while transmitting and receiving data through AXIOM NIC. For this purpose two AXIOM boards were used, interconnecting them through USB type-C connectors as shown in Figure 29.

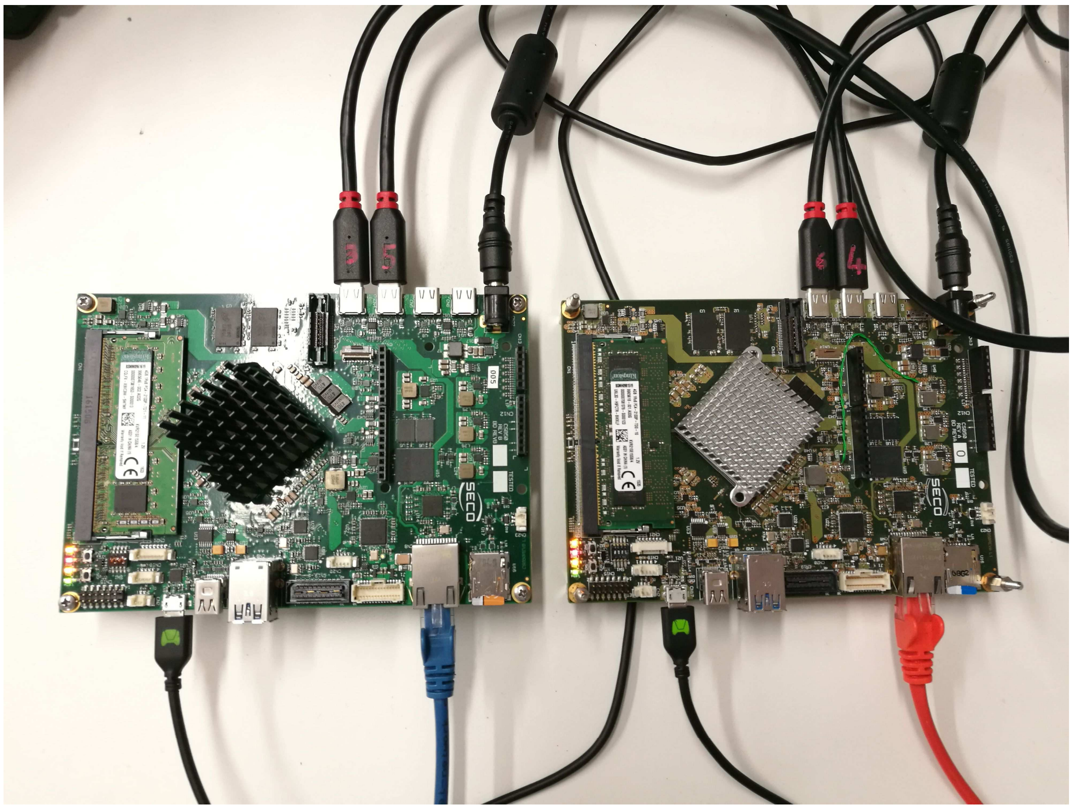
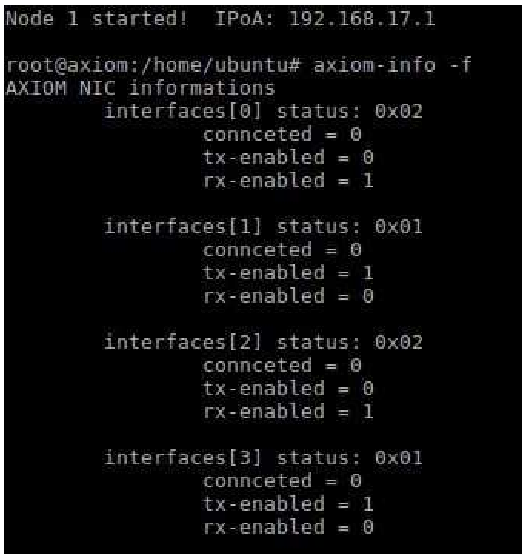


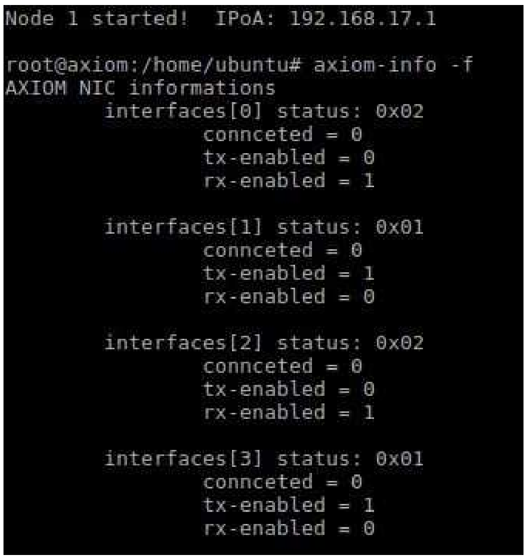
Figure 29 – Power measurement system configuration

[1] The System ID is a conventional zero-based number used to uniquely identify the chip. This number is used also as suffix of the device file descriptor assigned at each driver instance.

Power measurements has been performed while running *netperf* utility on AXIOM boards, configuring it in server mode in one of the board and in client mode on the other one, as shown in Figure 30.



```
Node 1 started!  IPoA: 192.168.17.1

root@axiom:/home/ubuntu# axiom-info -f
AXIOM NIC informations
        interfaces[0] status: 0x02
                connceted = 0
                tx-enabled = 0
                rx-enabled = 1

        interfaces[1] status: 0x01
                connceted = 0
                tx-enabled = 1
                rx-enabled = 0

        interfaces[2] status: 0x02
                connceted = 0
                tx-enabled = 0
                rx-enabled = 1

        interfaces[3] status: 0x01
                connceted = 0
                tx-enabled = 1
                rx-enabled = 0
```



```
Node 1 started!  IPoA: 192.168.17.1

root@axiom:/home/ubuntu# axiom-info -f
AXIOM NIC informations
        interfaces[0] status: 0x02
                connceted = 0
                tx-enabled = 0
                rx-enabled = 1

        interfaces[1] status: 0x01
                connceted = 0
                tx-enabled = 1
                rx-enabled = 0

        interfaces[2] status: 0x02
                connceted = 0
                tx-enabled = 0
                rx-enabled = 1

        interfaces[3] status: 0x01
                connceted = 0
                tx-enabled = 1
                rx-enabled = 0
```

Figure 30 - Client settings (left) and Server settings (right).

The power-data acquiring tool is available on the Desktop of the AXIOM system to any user as part of our distribution and it is named:

`hwmon_scan_all.sh`

This tool makes IOCTL calls to the INA219s' driver that return the present value of current and power absorbed to each monitored rail. These values are stored into a plane file, in tabular form and are sampled with parameters reported in Table 4. So, each performed test has as output a couple of file (one for client and one for server), through which the graphs shown below were created.

5.1 TESTS on the power consumption while sending packets through the AXIOM-link

Three different tests were performed to understand the power consumption of NI:

- **TEST1**: client netperf adopting LONG type, length 1000M, performing 10 cycles;
- **TEST2**: client netperf adopting RAW type, length 1000M, performing 10 cycles;
- **TEST3:** client netperf adopting RDMA type, length 1000M, performing 10 cycles.

Table 4 - Power measurement test setup

OS Image:		`XOS_v0.3_20171113`
SW test:		`hwmon_scan_all.sh` (refer to [66])
Test parameters	**Duration:**	240 s
	Sample time:	0.2 s

The following subsections provide power measurements on one of the most important power supply rail: VCC_INT, related to the FPGA portion within Zynq UltraScale+ (also referred as PL) in Figure 31, Figure 32, Figure 33, Figure 34, Figure 35, Figure 36. Data from INA219 power monitors can be used to derive useful information like the difference between minimum and maximum power consumption during a specific operation (next figures).

As reported by the pictures in the following subsections, the maximum and minimum power consumption is almost not affected by the specific package type used in the test setup. The most noticeable difference in power consumption (referred as 'delta power') is, as expected, related to the MultiGb transceivers dynamic power consumption (MGTAVTT, used to supply transceiver termination circuits).

- ## TEST1 power consumption (LONG type data packets)

Client:

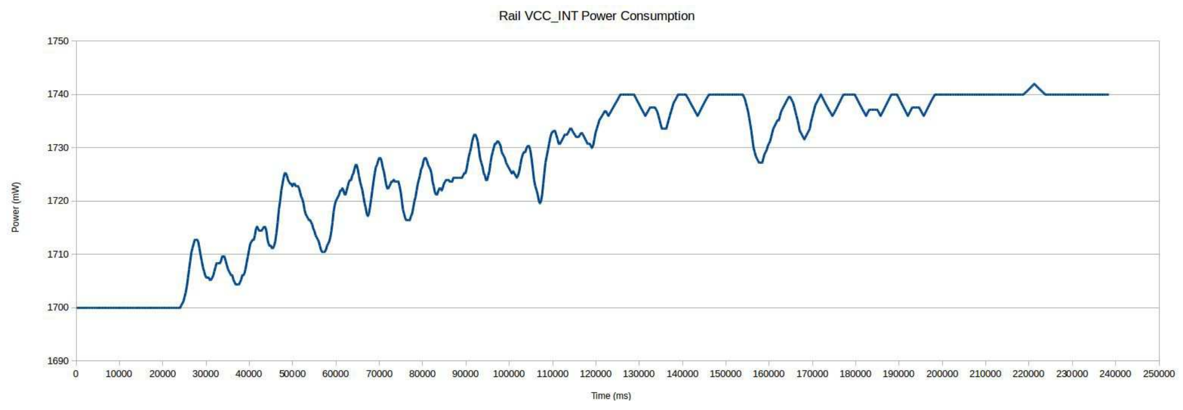


Figure 31 - Power Consumption for Client during TEST1 on VCC_INT power supply rail

Server:

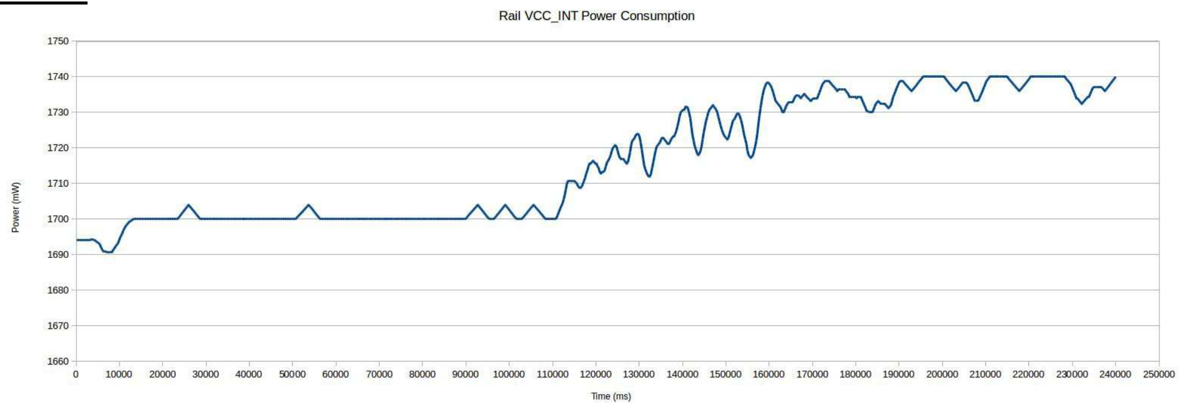


Figure 32 - Power Consumption for Server during TEST1 on VCC_INT power supply rail

- ## TEST2 power consumption (RAW type data packets)

Client:

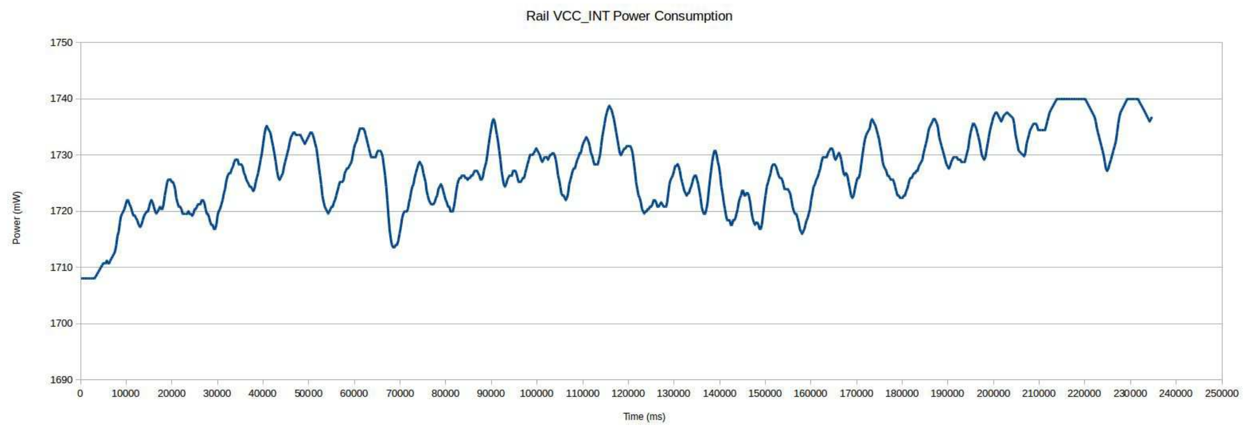


Figure 33 - Power Consumption for Client during TEST2 on VCC_INT power supply rail

Server:

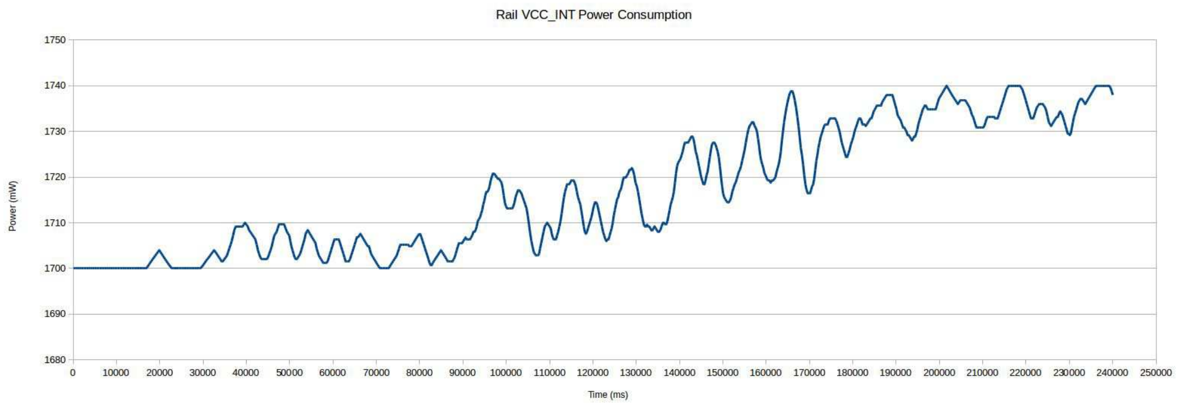


Figure 34 - Power Consumption for Server during TEST2 on VCC_INT power supply rail

- **TEST3 power consumption (RDMA type data packets)**

Client:

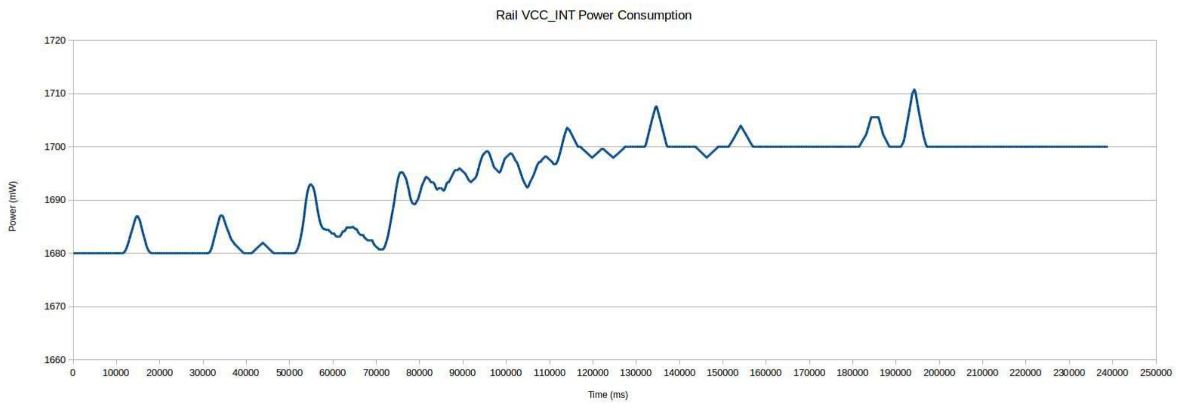


Figure 35 - Power Consumption for Client during TEST3 on VCC_INT power supply rail

Server:

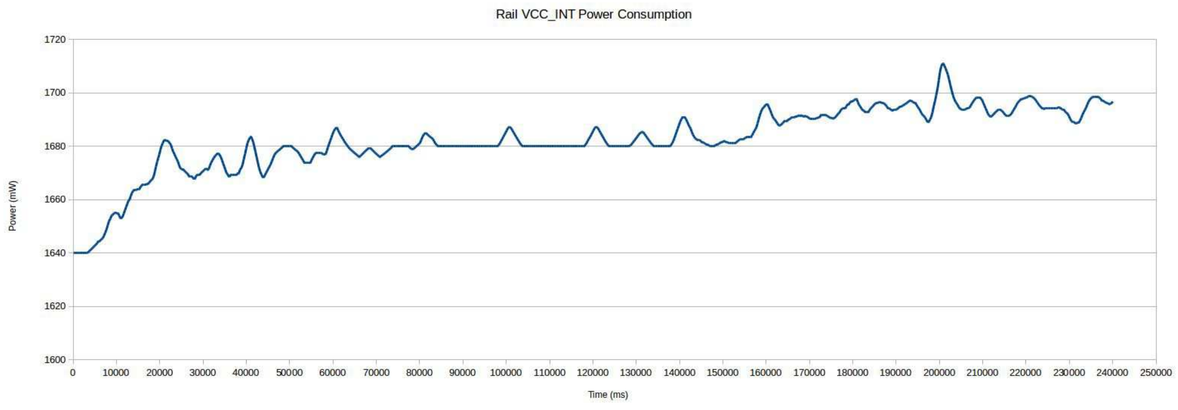


Figure 36 - Power Consumption for Server during TEST3 on VCC_INT power supply rail

- ## TEST1 delta power consumption (LONG type data packets)

Here we also summarize the variation of the power for other parts of the AXIOM-board. Essentially we remain between about 1.4W and 2W (Figure 37,Figure 38,Figure 39,Figure 40,Figure 41,Figure 42).

Client

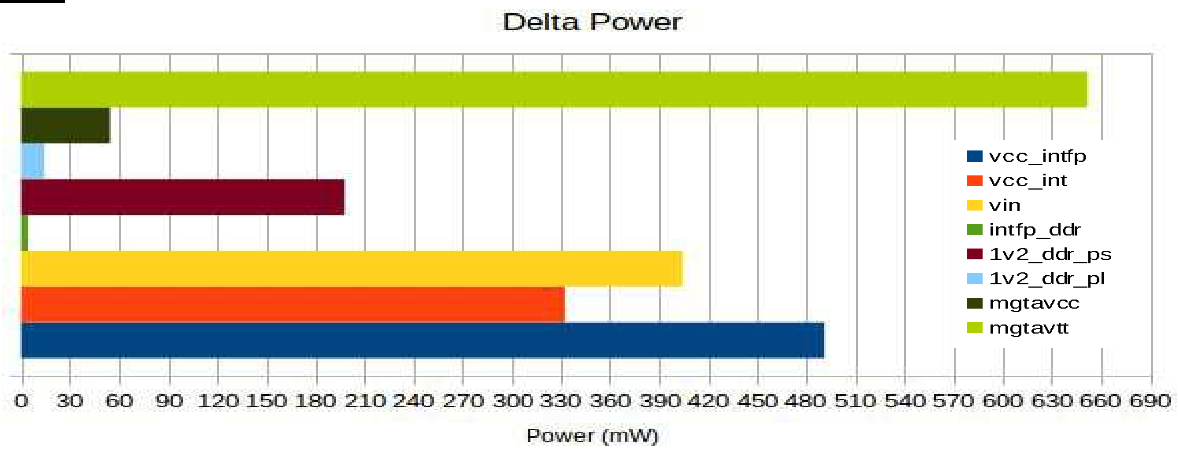


Figure 37 - Maximum Variation in Power consumption for Client during TEST1

Server:

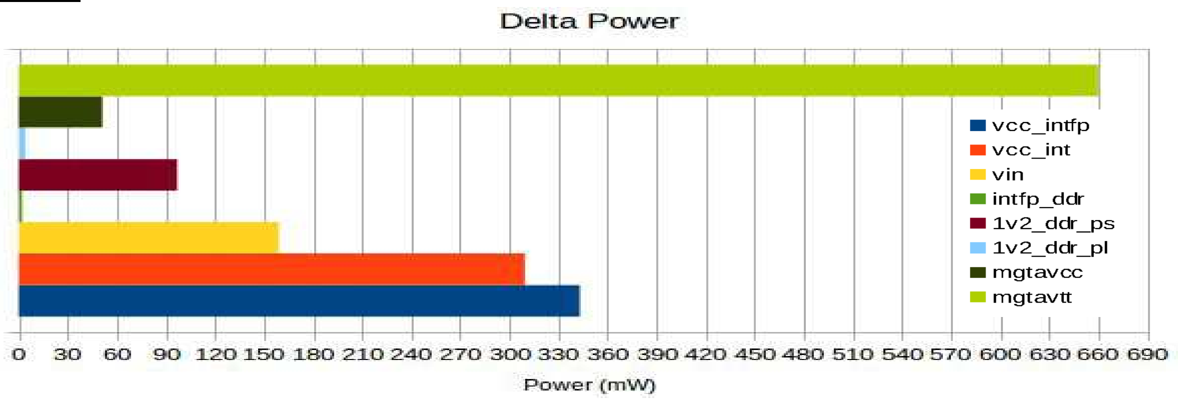


Figure 38 -Maximum Variation in Power consumption for Server during TEST1

- ## TEST2 delta power consumption (RAW type data packets)

Client:

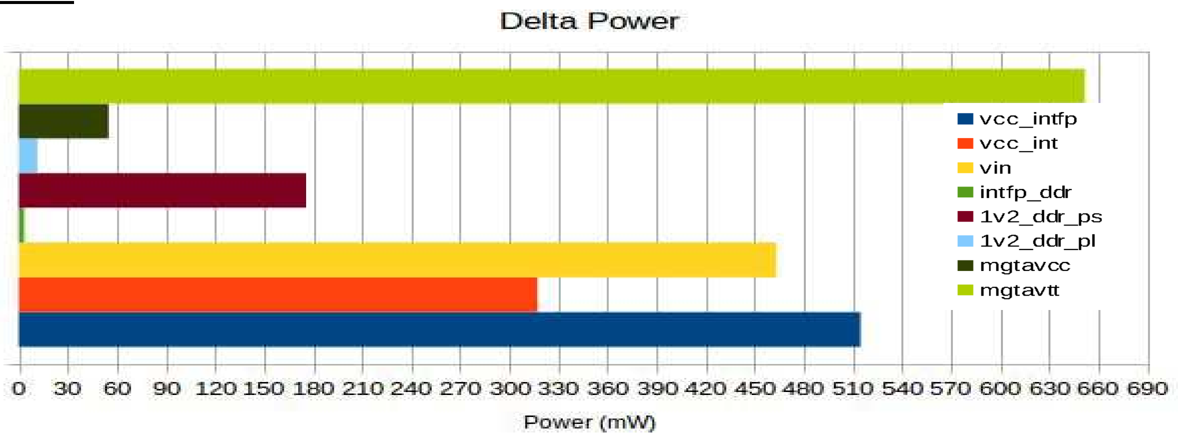


Figure 39 -Maximum Variation in Power consumption for Client during TEST2

Server:

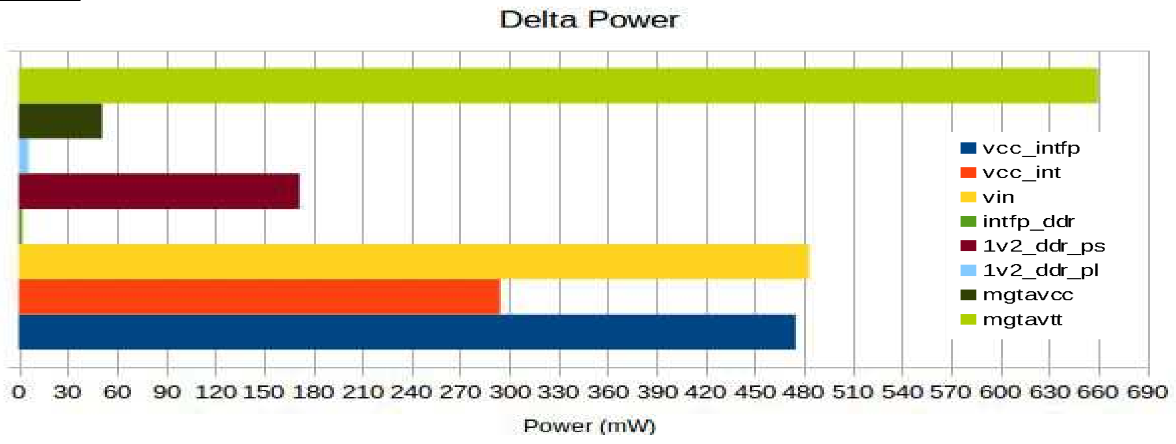


Figure 40 - Maximum Variation in Power consumption for Server during TEST2

- **TEST3 delta power consumption (RDMA type data packets)**

Client:

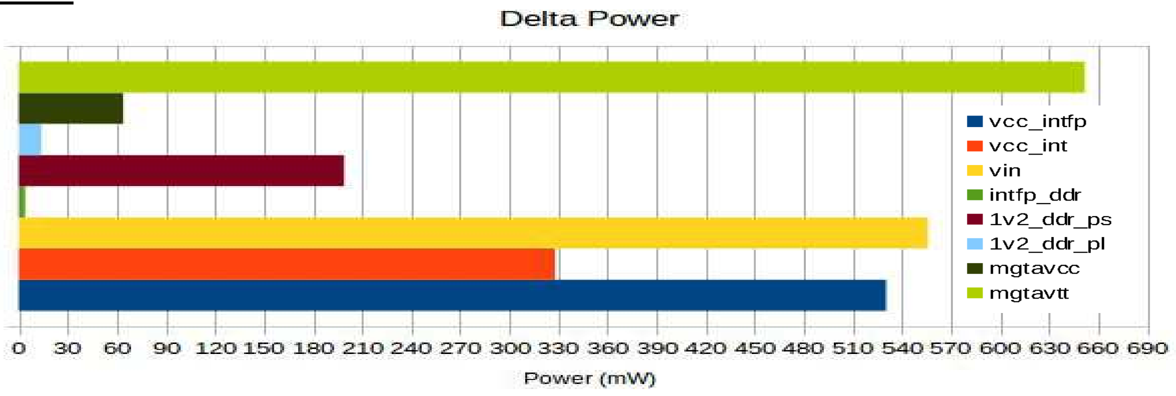


Figure 41 - Maximum Variation in Power consumption for Client during TEST3

Server:

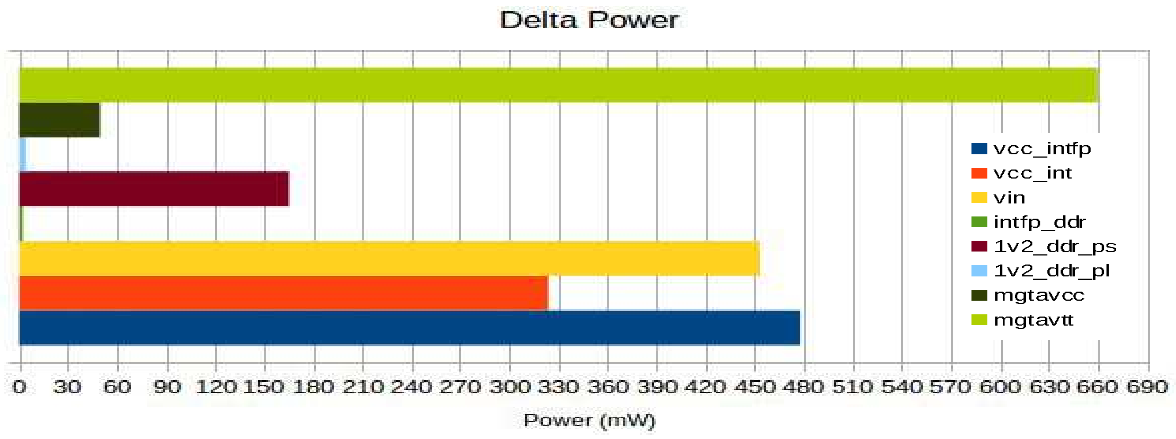


Figure 42 -Maximum Variation in Power consumption for Server during TEST3

6 Power consumption of matrix multiplication with OmpSs@FPGA

We have measured the power consumption of matrix multiplication on the AXIOM board. The algorithm is described in the AXIOM Deliverable D4.3 [68]. The matrix multiplication works on a matrix of 2048x2048 single precision floating point elements, on blocks of 128x128 elements. This version based on 128x128 blocks allows to experiment with using 1 to 3 FPGA IP cores.

We have evaluated the power consumption of matrix multiplication when using the following runtime alternatives:

- 1 to 4 CPU cores, that is, using only the SMP host cores, monitoring the PS cluster and the total power consumption.
- 1 to 3 FPGA IPs, with 2 helper threads and 4 / 16 pending tasks, that is using only the FPGA accelerators, with the helper threads support, monitoring the PS and PL clusters, and the total power consumption.
- 2 SMP cores and 1 to 3 FPGA IPs with 2 helper threads, that is using the full set of resources available, also monitoring the PS and PL clusters and the total power consumption.

6.1 Power consumption on SMP host cores

Figure 43 shows the power consumption of the SMP of the matrix multiplication benchmark, running on 1 to 4 SMP cores. As it can be observed, the baseline power consumption when the 4 cores are idle is a little bit over 1W that is 1,140 W. Each SMP core adds up to 120 mW.

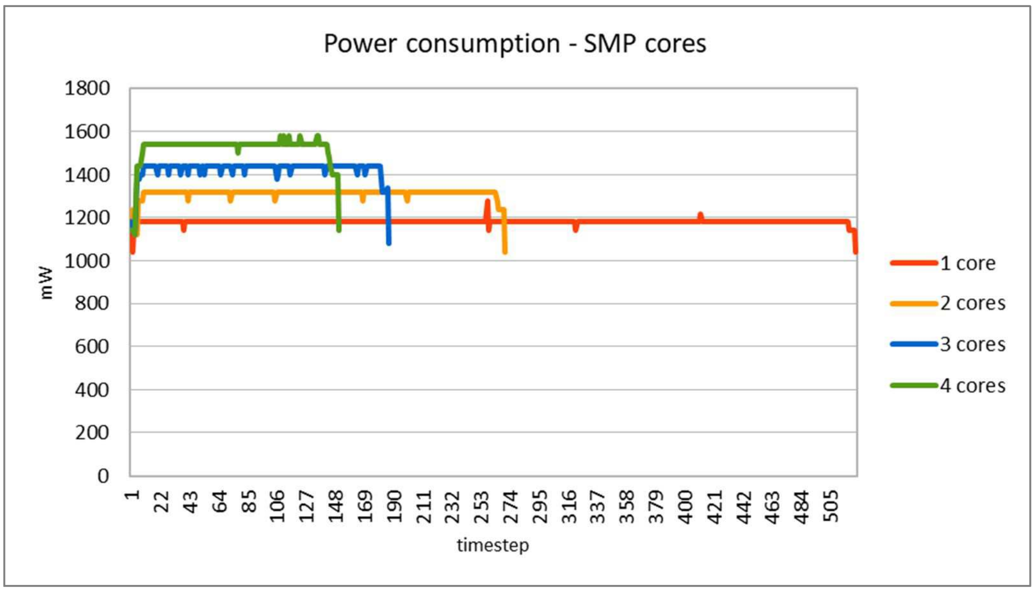


Figure 43 - PS SMP cores power consumption of matrix multiplication running on the SMP cores

Regarding the power consumption of the PS DDR, we have obtained that 4 cores add up to 100 mW (see Figure 44), from 360 mW when running on 1 core to 460 mW when running on 4 cores. It is noticeable the peak on the DDR power consumption when the application starts, reaching up to 800 mW of instantaneous power consumption when running on 3 and 4 cores.

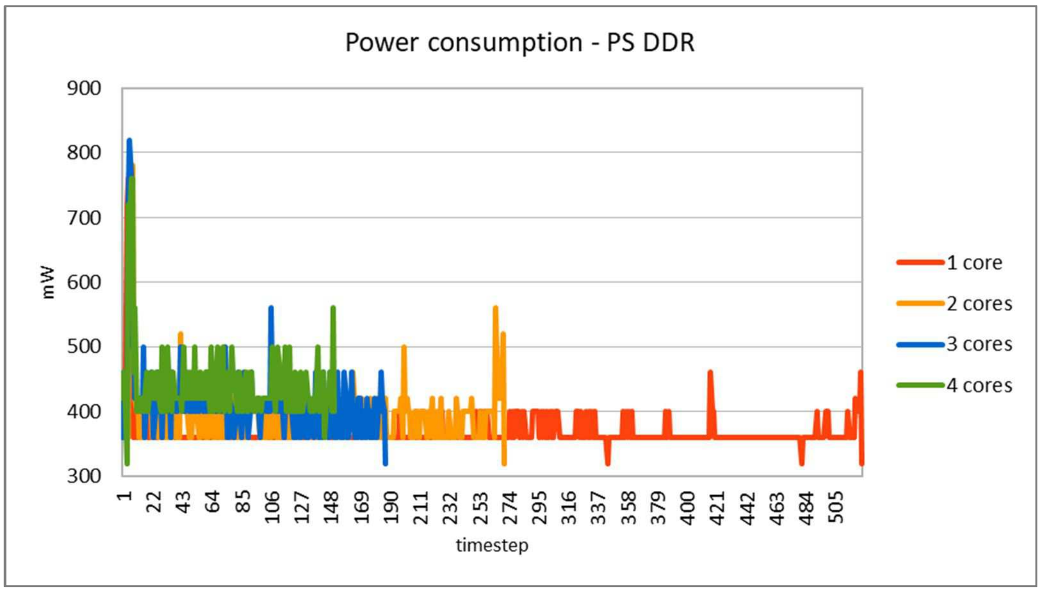


Figure 44 - PS DDR power consumption of matrix multiplication running on the SMP cores. Observe that it is based on 300 mW, as the details can be better appreciated

The overall power consumed by these experiments is shown Figure 45. This includes all the AXIOM board. The minimum values on idle are 12400 mW. In particular, the idle FPGA bitstream has a baseline power consumption of 5640 mW. The power spent follows the same shape of the PS (cores and DDR), as it is expected.

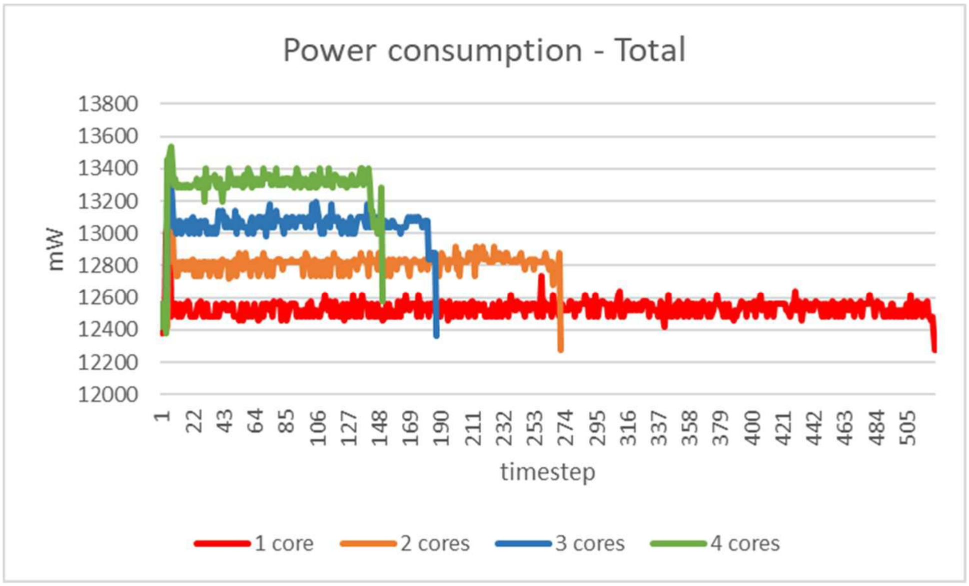


Figure 45- Total power consumption of matrix multiplication when running on 1 to 4 SMP cores. Observe that it is based at 12000 mW, as the details can be better appreciated

6.2 Power consumption on FPGA IPs

The next set of results show the power consumption of matrix multiplication when running on the 1 to 3 128x128 block IPs available on the FPGA. Figure 46 shows the power consumption of the main thread and 2 helper threads in the 4 experiments presented. Each experiment uses 1 to 3 FPGA IPs, and is done with 4 or 16 pending tasks. The shape is similar to Figure 45, with much more variability, as the tasks done by each thread are dedicated to the management of the FPGA IPs. At the end of each execution the power consumption increases because of the verification phase of the matrix multiplication.

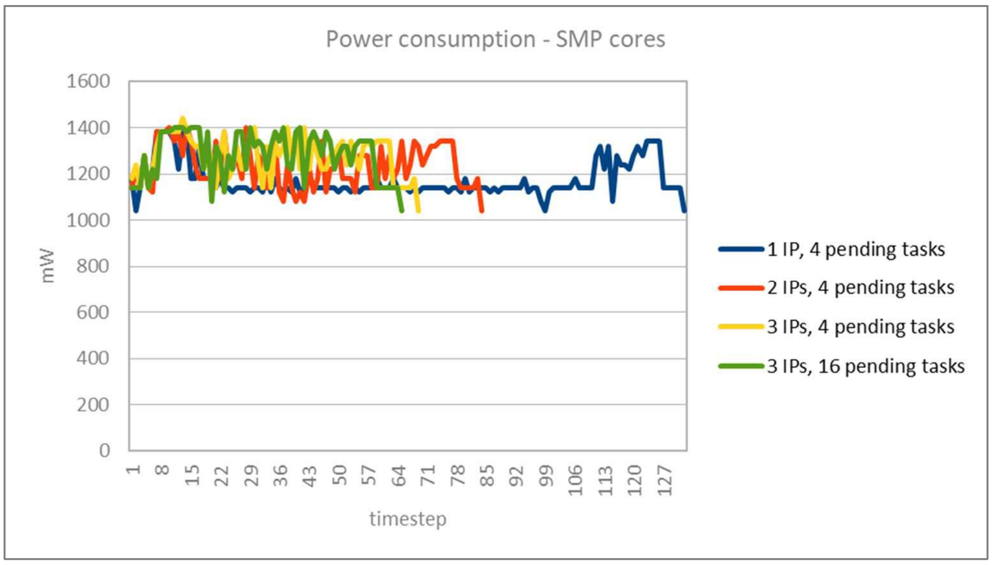


Figure 46 - PS SMP cores power consumption of matrix multiplication when running with 1 to 3 IP cores

Figure 47 shows the power consumption of the FPGA, when running 1 to 3 IPs (with up to 4 pending tasks), and the fourth experiment with 3 IPs and up to 16 pending tasks. It shows that the IPs consumption increases from 5640 mW to 6260 mW. Given that the idle bitstream is constantly consuming 5640 mW. We attribute the additional 620 mW to the movement of data through the AXI interconnects and inside the FPGA IPs. The addition of each IP increases the power consumed by around 150-300 mW, including the fact of increasing the number of pending tasks from 4 to 16, which also increases performance, and power consumption.

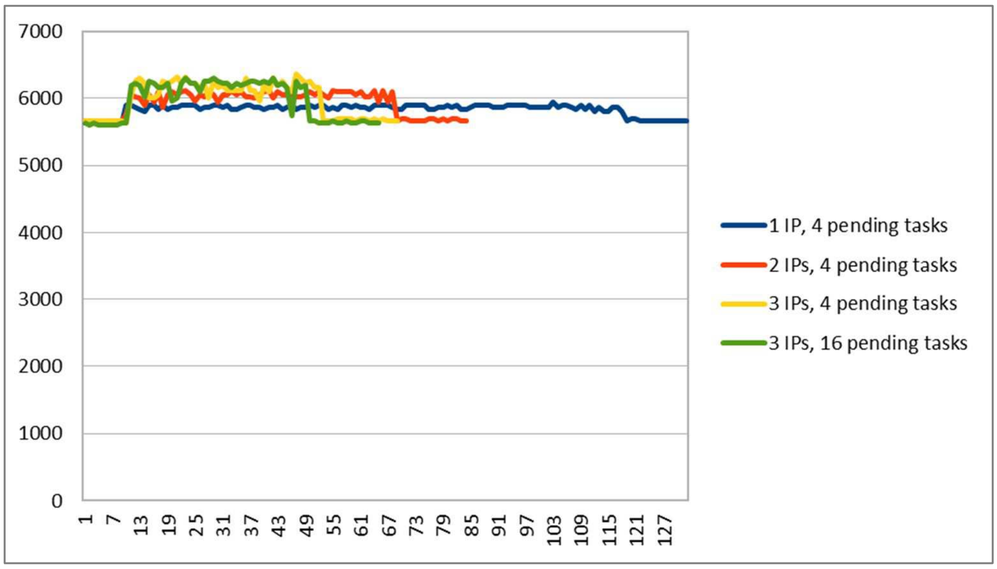


Figure 47 - PL power consumption while executing matrix multiplication using 1 to 3 IP blocks

The last measurement on this experiment is also the total power consumption from the AXIOM board. Figure 48 shows these results. The shapes of the graphs resemble the addition of the previous Figure 46 and Figure 47. Compared to the execution on SMP cores only, we can observe that both of them have minimum values at 12200 mW. But this one with work on the IPs, it reaches a peaks of power consumption of up to 14600 mW, while running on SMP cores only, reaches a flat maximum of 13400 mW (see Figure 45).

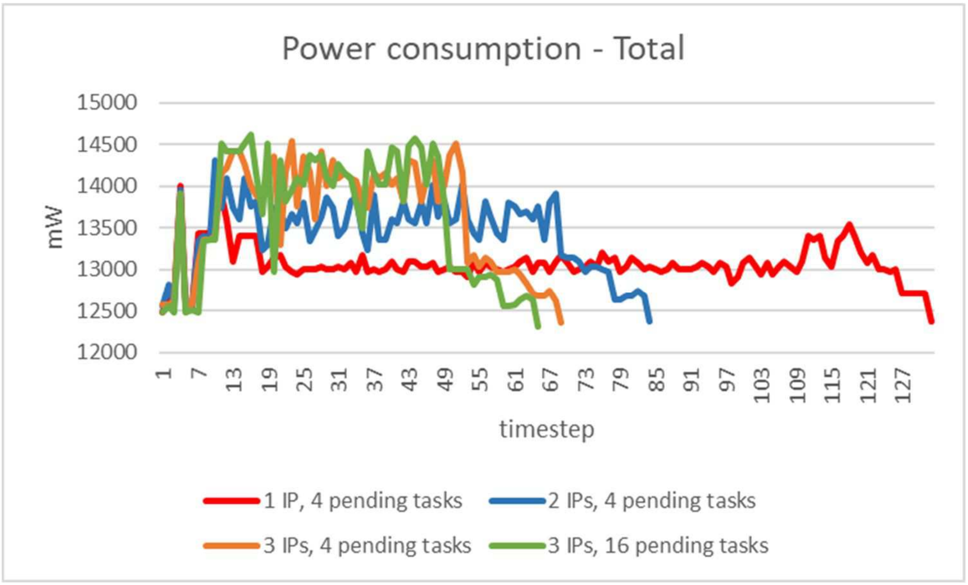


Figure 48 - Power consumption of matrix multiplication when using 1 to 3 IPs on the FPGA. Observe that it is based at 12000 mW, as the details can be better appreciated

6.3 Power consumption on SMP and FPGA

The third set of experiments measures the power consumption when running matrix multiplication using the master thread, one worker thread, and 2 hybrid helper threads on the 4 SMP cores, and the 3 IPs on the FPGA.

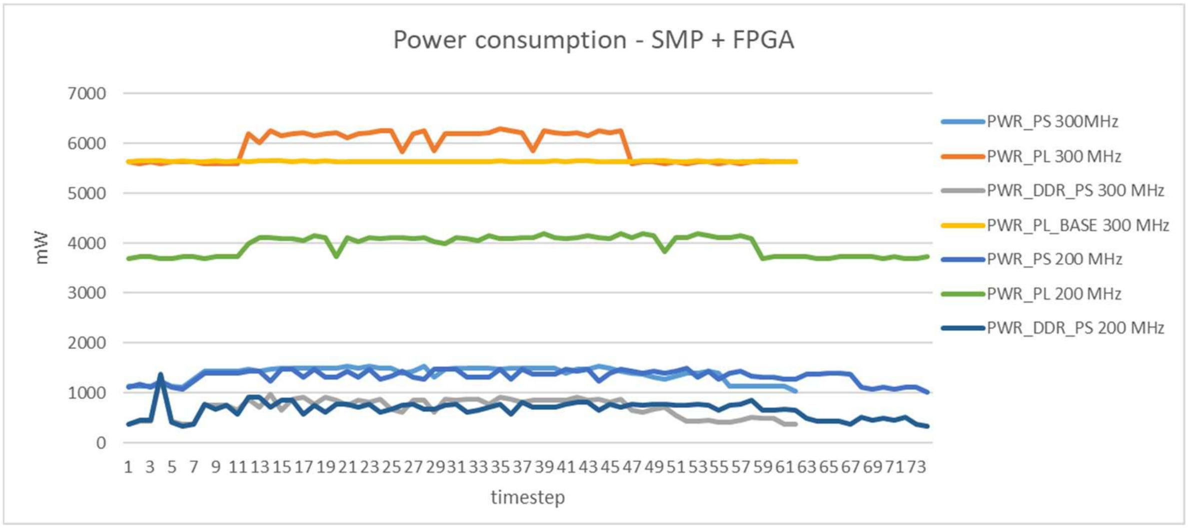


Figure 49 - Power consumption of matrix multiplication when using 4 SMP cores (master, worker, 2 helper threads), and 3 IPs on the FPGA. It includes data obtained when running on 300 MHz. and 200 MHz.

We can observe that there is an important increase of power consumption – from 4000 mW to 6200 mW when increasing the frequency of the FPGA hardware from 200 to 300 MHz. At the same time, there is also an important increase in the performance. The power consumption on the PS ARM cores, and the DDR memory is nearly the same regarding the FPGA frequency.

Finally, Figure 49 shows the total power consumption when running matrix multiplication when running on the SMP and FPGA resources. We can observe the difference on consumption between the 200 MHz and the 300 MHz implementation. The higher frequency version nearly reaches 15 W, while the execution at 200 MHz barely goes up to 11 W.

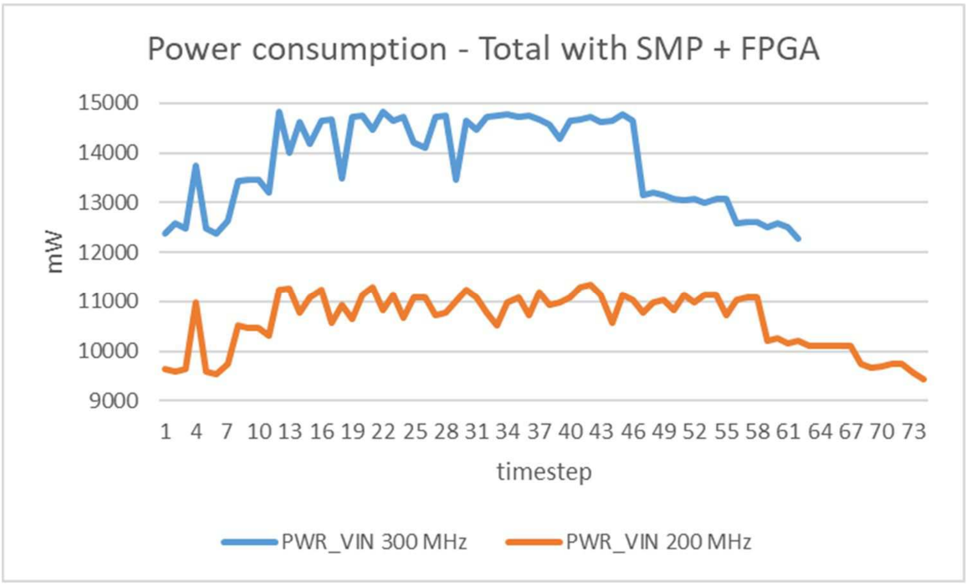


Figure 50 - Total power consumption of matrix multiplication including SMP and FPGA execution

7 Smart Video Surveillance (SVS) application performance and power evaluation

We extensively used the AXIOM-board to run the SVS application developed by Partner HERTA:

- The PL domain has an LBP face detector accelerator synthesized on the reconfigurable logic, and performs face detection from the internal 4 Megabyte BlockRAM (32 Megabits). The classifiers are also stored in the BlockRAM. As such, the image chunks of a given input video frame are transferred from the external DRAM memory to the on-die local memory of the PL block to evaluate the classifiers, and then look for faces.
- The PS domain (ARM Cortex A53 cores) executes our Qt-based WP3 application logic, performs video H.264 demuxing/decoding and also includes all the required glue code for the XDMA.
- The Mali GPU runs all the required GLSL pixel and vertex shaders, and deals with the OpenGL ES 2.0 textures for the video buffers. They are basically vector-parallel operations for texture resizing/downscaling, and for performing coordinate transformations. The GLSL code is compiled by the ARM Mali driver using JIT compilation, and thus generating the SIMD-like GPU assembly code. This is automatically done when our application is started.

To sum up, we are leveraging the heterogeneous architecture of the UltraScale+ chip (ARM CPUs + FPGA + GPU) to show the potential of the AXIOM board.

More in detail, the power/performance of the Smart Video Surveillance (SVS) use case application was evaluated running on the Xilinx UltraScale+ ZU9EG MPSoC included on the AXIOM board. As it was described in Deliverable D3.3, the SVS application implements a heterogeneous video face analytics pipeline, and thus heavily exploits both the PL and PS clusters.

In order to determine how the application would behave in stressed out scenarios, we carefully selected two different pre-recorded H.264 1080p videos (siena.mp4 and crowd_run.mp4), which greatly varied throughout their frames both the dimensions and the amount of simultaneous faces. Table 5 included below summarizes the main properties of the selected videos:

Table 5 - Selected videos for the SVS application evaluation on the AXIOM board.

Video Name	Codec Properties	Total Frames
`siena.mp4`	1920x1080 H.264/AVC 4559 Kbps	150
`crowd_run.mp4`	1920x1080 H.264/AVC 36728 Kbps	500

The first video (siena.mp4) was recorded by UNISI in their premises and featured three simultaneous faces approaching to the camera. The idea behind this test was to analyze faces at the first stages of the multi-scale Gaussian pyramid (i.e. the bigger ones) with the aim of increasing the number of potential sliding windows during the LBP face detection process. As the histogram in Figure 51 shows, most detected faces -before applying non-maximum suppression- were found at the first 4 scales of the image pyramid. This fact means that they were localized within the bigger scales of the Gaussian image pyramid, which are the most computational expensive as they contain the highest amount 48x48 candidate windows.

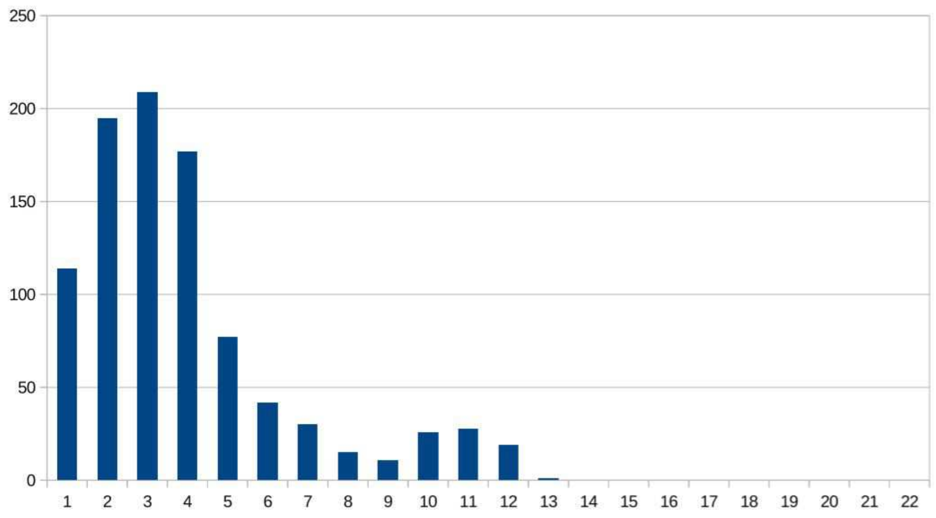


Figure 51 - Number of detected faces for each scale of the 22-scale Gaussian image pyramid (`siena.mp4`).

On the other hand, the second video (crowd_run.mp4) featured approximately up to 60 simultaneous faces per frame after having applied non-maximum suppression (NMS). It should be noted that the NMS process is required for selecting the best detected face from the set of detections returned from the LBP classifier from a given 48x48 sliding window.

This video can be found in the SVT HD multi-format test set[2], and it shows runners participating in a marathon. The amount of total detected faces per frame (before applying NMS) ranged between 370 and 650 (see Figure 52 for more details).

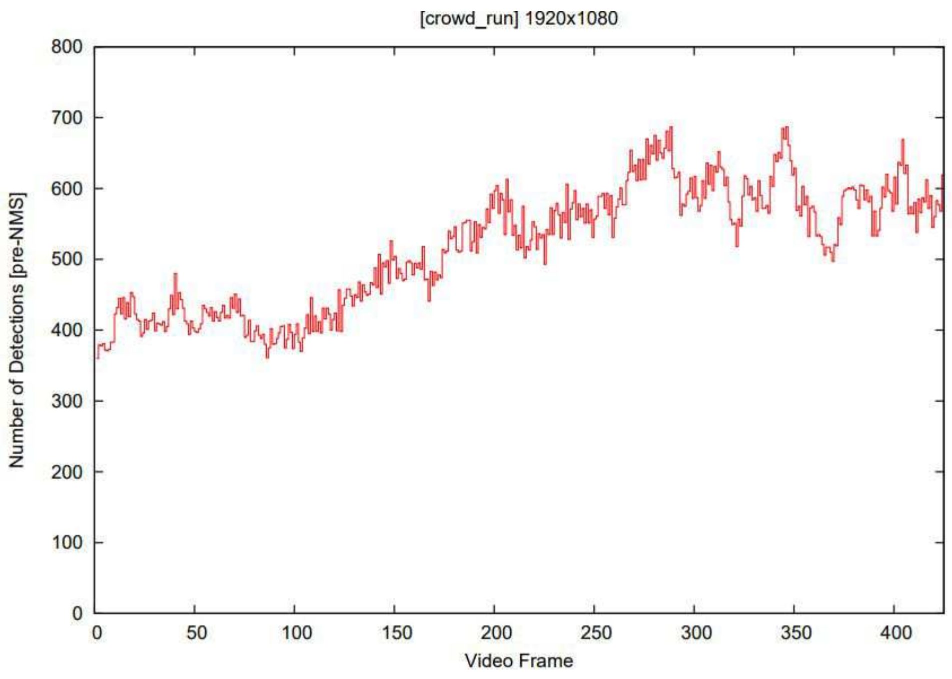


Figure 52 - Number of detected faces per frame before NMS (crowd_run.mp4).

7.1 Power measurements of the SVS application

Taking both videos as an input of the SVS application, we measured the power consumption of the whole SVS application using three of the eight Texas Instruments INA219 chips included on the AXIOM board. More particularly, we monitored the PS (PM_VCC_INTFP), PL (PM_VCC_INT), and VIN (PM_VIN) voltage domains of the UltraScale+ MPSoC. The main purpose was to determine the combined current and power consumption of the CPU cores, GPU, and FPGA reconfigurable logic when running the software.

The power experiments were carried out using a sampling rate of 300 ms by gathering the current and power metrics from the Linux kernel `sysfs` interface developed by SECO. These measures were stored in the RRDtool[3] round-robin database for further analysis. The obtained results (see Figure 53 and Figure 54) showed that the total power consumption of the SVS application when running on the board (VIN power rail) was less than 8 W, while drawing an average current of 600 mA in the VIN shunt resistor. However, the most relevant part was to determine the detailed power consumption of both PL and PS clusters of the UltraScale+ while analyzing the selected videos.

[2] ftp://vqeg.its.bldrdoc.gov/HDTV/
[3] https://oss.oetiker.ch/rrdtool/

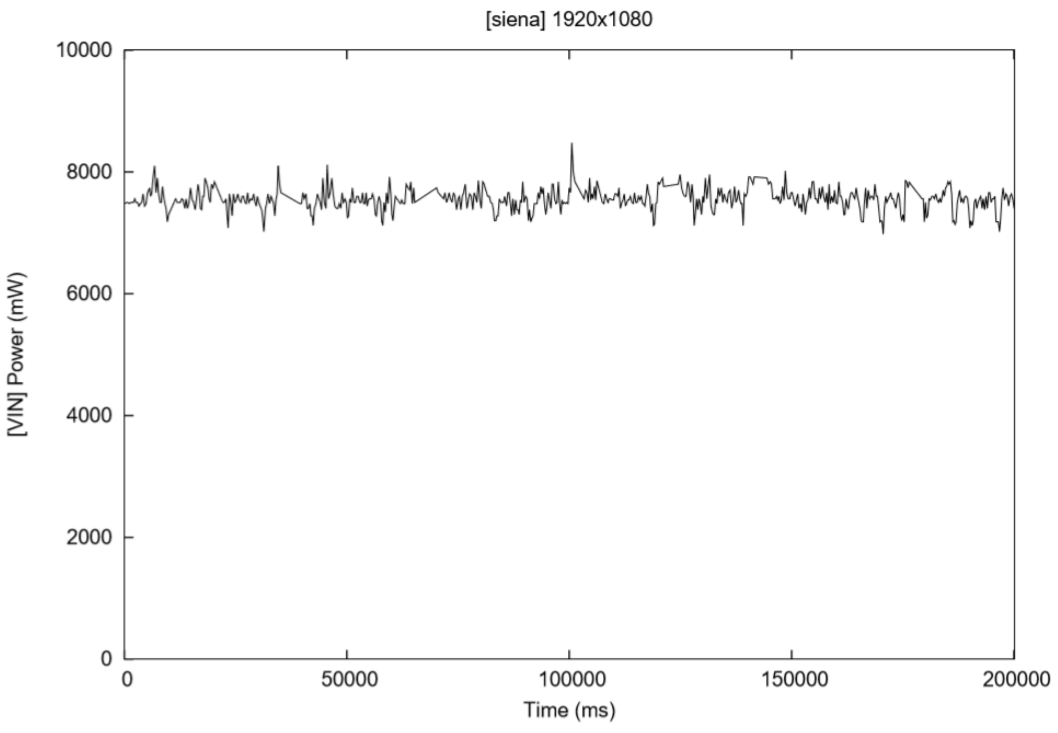


Figure 53 - Total power consumption of the AXIOM board (siena.mp4).

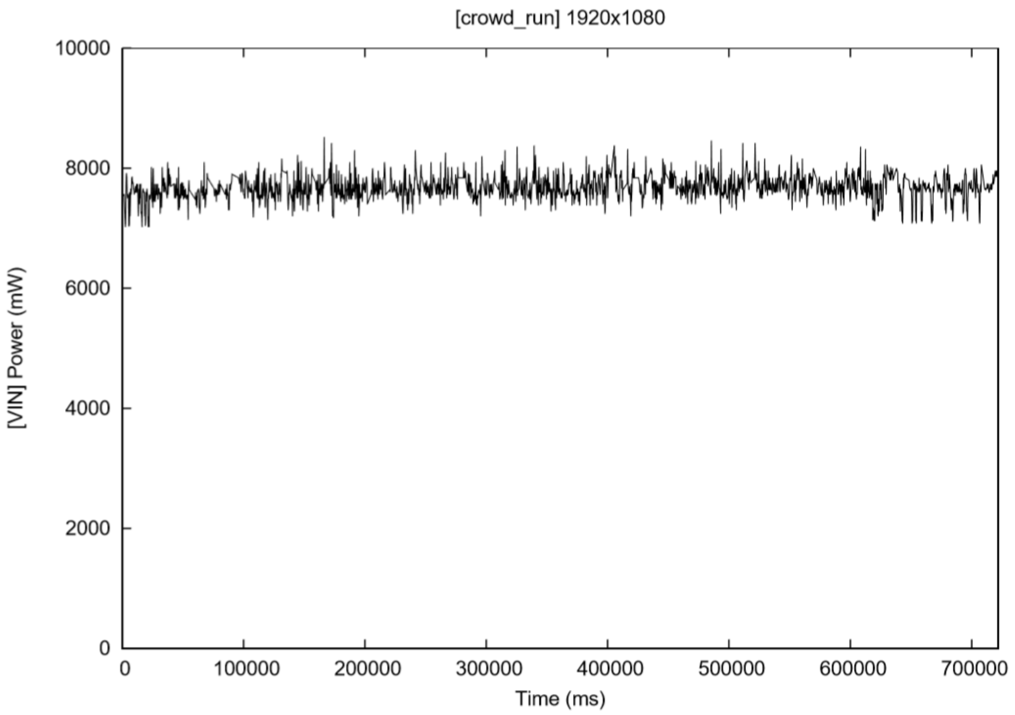


Figure 54 - Total power consumption of the AXIOM board (crowd_run.mp4).

Since the PS cluster deals with H.264 video decoding, performs the required CPU/FPGA memory transfers, and also renders the UI on the GPU; we expected variations in the power consumption depending on the load factor of the ARM Cortex A53 CPU cores due to the DVFS techniques implemented in the

MPSoC. As the Figure 55 enclosed below shows, the obtained power consumption of the PS cluster was less than 1 W before/after the SVS video application started/stopped analyzing the video (highlighted vertical red lines). However, when a given frame is decoded, analyzed, and mapped into an OpenGL ES texture, the power consumption peaks on average to 1.3 W.

Similarly, the power consumption also returns to 1 W while the CPU cores are idle waiting for the completion of I/O operations from the miniSD card or CPU/FPGA memory transfers. More interestingly, the PS power trace also show regions in which the power consumption is halved or cut by a factor of three from the peak values. This means that up to three ARM CPU cores are power down when they are idle. Unfortunately, there was no way to determine the power consumption attributed to the ARM Mali 400 GPU from the aggregated PS power traces.

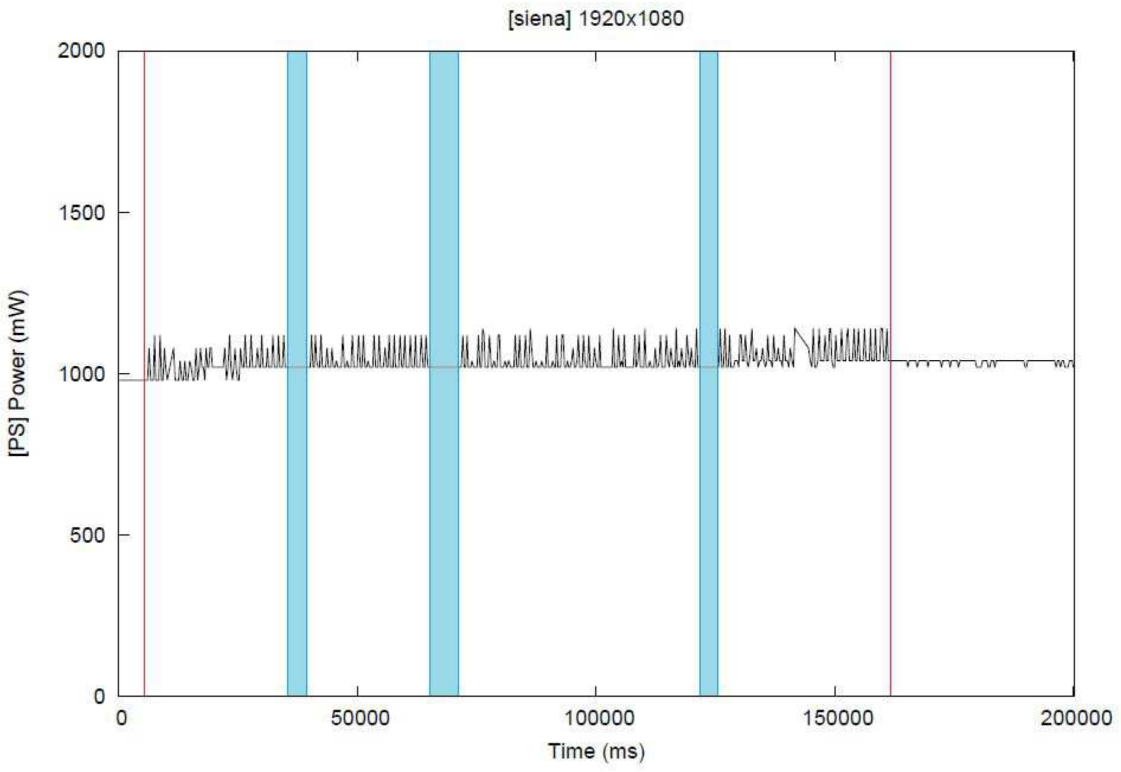


Figure 55 - PS cluster power trace running the SVS application (siena.mp4).

Finally, there are also areas shaded in blue which may correspond to delays attributed to longer I/O operations, as the PS cluster power consumption is very far from peak values. Unfortunately, were not able to profile the exact lines of the SVS source code in which such delays were originated while capturing values from the INA219 power measurements.

As Figure 56 shows, the PS domain power consumption values obtained for the crowd_run.mp4 video closely resembled the ones obtained from the siena.mp4 video. The power consumption also was cut down substantially before and after the input video was processed (vertical red lines).

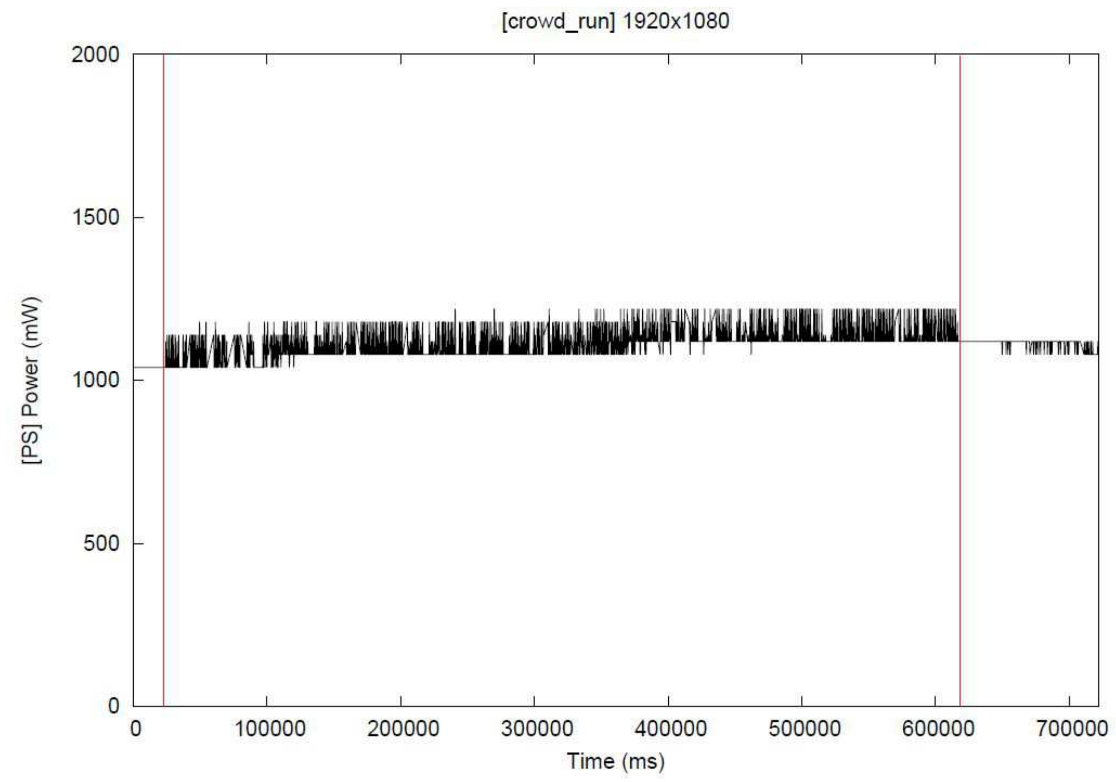


Figure 56 - PS cluster power trace running the SVS application (crowd_run.mp4).

Arguably, the most interesting part of the SVS application porting to the AXIOM board was the offloading of the LBP face classifier to the reconfigurable logic using the OmpSs@FPGA programming model. This particular kernel was called by the SVS application for each video frame. It used as an input image chunks from the multi-scale Gaussian image pyramid, which were cached in the on-die BlockRAM memory on the fly to avoid excessive off-die DDR4 memory accesses.

The bitstream of this accelerator was synthesized into the PL cluster together with the boosted cascade of classifiers, which were also cached in BlockRAM in order to increase the classifier memory fetch bandwidth. The obtained PL cluster power measurements of this FPGA accelerator are shown in Figure 57 and Figure 58, and consumed on average less than 1.5 W. Unfortunately, we were not able to attribute the power consumption spikes to any particular computation, as we did not have the capability to trace cycle by cycle the FPGA power consumption on the AXIOM board at the lab premises where we tested the SVS application (in the next Section we will show how this was done in the SHL application). However, the minor reductions in power consumption on the PL cluster could be attributed to lower activity of the DSP units used by the LBP kernel. This lower activity may be related to delays in which some ALUs included in the DSPs are idle waiting from data requested from the BlockRAM memory or pending DDR4 memory transfers from or to the off-die DRAM memory.

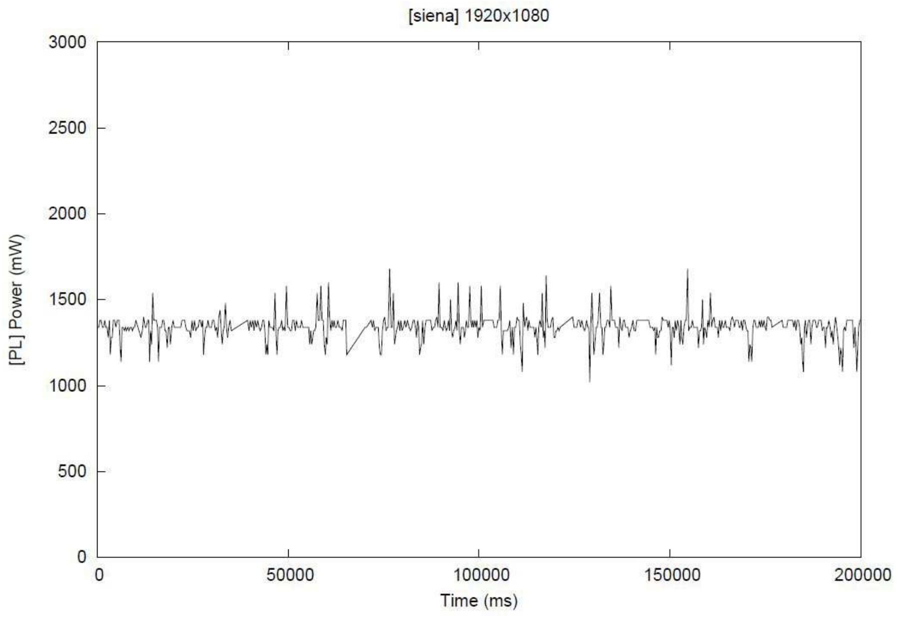


Figure 57 - PL cluster power trace running the SVS application (siena.mp4).

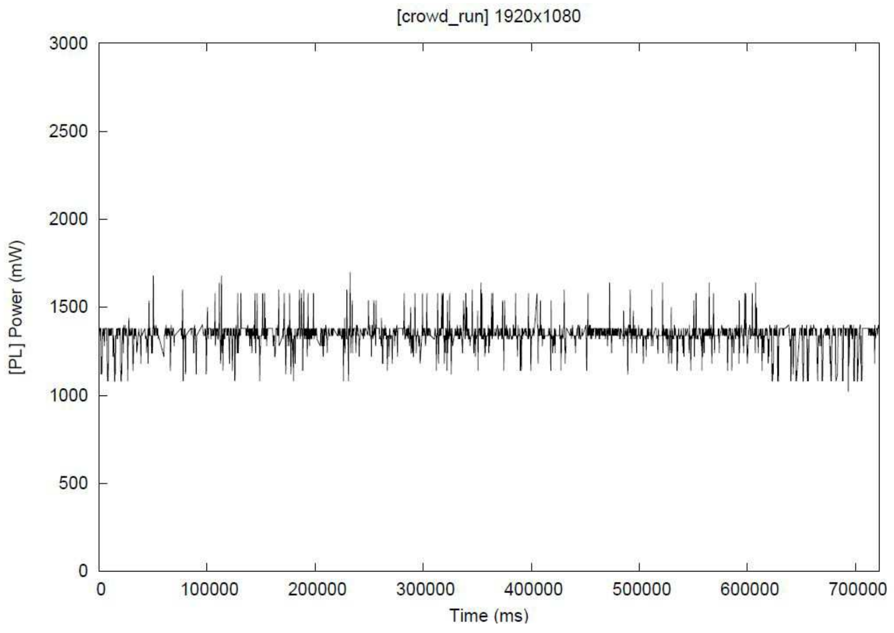


Figure 58 - PL cluster power consumption running the SVS application (crowd_run.mp4).

After having studied carefully the obtained power traces, we estimated that the SVS application running on the Xilinx ZU9EG MPSoC consumed on average less than 3 W while performing face analysis on two challenging video sets.

The obtained total power consumption of the MPSoC was obtained simply by aggregating the power consumption of both PS and PL power domains. The SVS application power consumption was consistent with the values estimated by the Xilinx Power Estimator spreadsheet [60].

8 Smart Home and Living (SHL) application performance and power evaluation

In this Section, we discuss how the OmpSs execution environment has been evaluated on the Smart Home/Living (SHL) application. The environment was tested with the AXIOM evaluation platform (32-bit Xilinx ZC706 evaluation kit) and with the AXIOM platform (64-bit Zynq UltraScale+ ZU9EG).

In the SHL scenario, several applications manage the comfort, the security and the energy consumption of the smart home. In the context of the AXIOM project it has been implemented an application to identify the user by the iris code and the voice features extracted by the video and audio captured from the video door entry system. The application has been explored with the aim to validate and define the advantage of the heterogeneous architecture and the execution environments.

The *Axiom_Bio_Identification* is the application developed by VIMAR, and it is based on several open-source libraries to process and analyze audio and video data. The description of the application is presented on the D3.3. The video processing task and the video analysis task are the more time-consuming tasks and their workflow is briefly described in the following paragraphs.

The video task starts with the identification and extraction of the minimal box that incorporates an eye inside the frames record from the input video recorded by the video door entry system. That minimal box is named Region of Interest (ROI). ROIs are processed to extract the metrics required for determining the image quality; in the case they violate image quality thresholds they are discarded, otherwise the ROIs are processed to generate the iris code.

To extract the iris code, a segmentation step is executed in which the inner and the output boundaries of iris are detected. Thereafter, the iris annulus is transformed into a size-invariant strip, following Daugman's rubber-sheet method [71]. This new image is filtered to extract the iris features, and finally, an iris code is generated as a selection of the complete set of features. To conclude, a matching phase is performed to find the distance from the processed iris code to the codes saved in the enrolled subject's database.

Anisotropic smoothing is a task required in the segmentation steps of iris recognition module and it is a very time-consuming workload task. This task aims at filtering the ROIs recognition in frames to retrieve the iris contours. These tasks are sequentially executed four times for each ROI to obtain the information required by the algorithm.

To accelerate the execution of the *Axiom_Bio_Identification* application, the above-mentioned algorithm was exhaustively analyzed to precisely determine the data flow and its corresponding dependences and define the tasks that could be mapped in the SMP and the FPGA resources. Finally, it was annotated with OmpSs directives to define the tasks and the target devices of each task. The code of the tasks with a FPGA target has been also annotated with the HLS directives to optimize the implementation of the FPGA accelerators.

The structure of the *Axiom_Bio_Identification* application is outlined below:

- The application takes the input frames from the camera using the GStreamer framework;
- The eyes (ROIs) contained in a single input frame are extracted using the OpenCV library;
- Two ROI_Tasks are created, each task takes as argument the reference to the ROI containing the single eye;
- The ROI_Tasks run on SMP, each one creates 4 *Anisotropic_Tasks* that must be executed sequentially.

Given the current structure of the SHL application we experimented acceleration possibilities on the single node (for more nodes, we need more careful management of the data dependencies, so we focused on the FPGA and multicore acceleration in first instance).

8.1 Performance Results of SHL application

The analysis done on the algorithms in the *Axiom_Bio_Identification* application showed that the data of each ROI is independent from each other, thus the ROIs can be processed using different parallel tasks. In collaboration with partner BSC, the OmpSs directives have been introduced in the source code to transform the sequential processing of the video frames into several tasks, which will be identified with the name of ROI_Tasks, which can be scheduled by the Nanox++ runtime system. The complexity of the code involved in the processing of the ROIs does not enable to map efficiently all the work done by these tasks into the FPGA PL resources, so the target device for such tasks has been set equal to SMP with OmpSs directives. However, the ROI_Tasks incorporate the segmentation step, which contain the anisotropic smoothing tasks, which are fit to be executed in FPGA. In order to take advantage of the FPGA resources, OmpSs annotations were added to the anisotropic smoothing code to define OmpSs@FPGA tasks. These tasks are named Anisotropic_Tasks and they include all the operations of the anisotropic smoothing algorithm. Further low-level PL optimizations were explored by annotating the code with HLS directives. On the tests done, the target device for the Anisotropic_Tasks were set to SMP in some experiments and to FPGA in other experiments. In the FPGA experiments the pragmas of OmpSs@FPGA that allows us to instantiate multiple-accelerators were explored with the introduction of 1 and then 2 accelerators.

In the segmentation step, the anisotropic smoothing task is executed four times but the data flow in this step does not allow the parallel execution of the four anisotropic smoothing calls. However, the tasks creation and execution of the iris recognition module enable the parallel processing of several ROI_Tasks, and allows the usage of design in which the FPGA accelerators are mapped several times in PL fabric of the MPSoC.

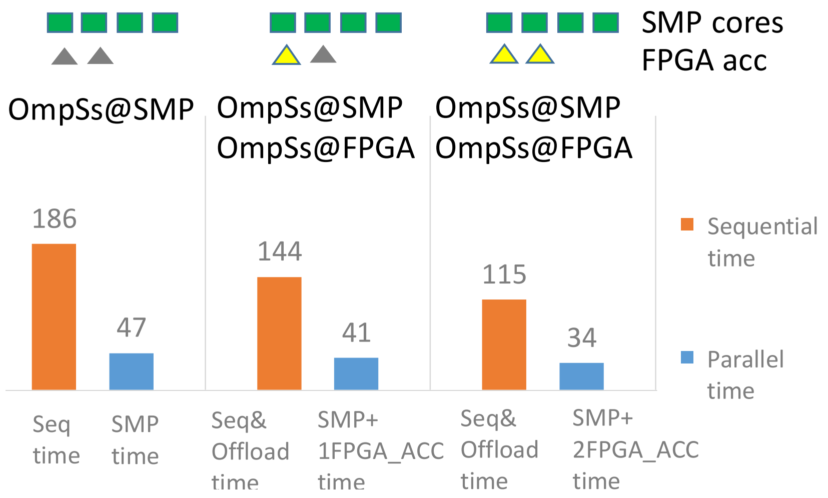


Figure 59 - Sequential time and the parallel time of the Axiom_Bio_Identification application in the AXIOM board. The sequential time is the sum of time spent in each SMP resources in user-mode and kernel-mode within the VIMAR application, the parallel time is wall-clock time.

Figure 59 shows the time execution of the Axiom_Bio_Identification while processing a video of 60 frames a with resolution 720x578 pixels in the AXIOM board. The columns show the results of the following experiments:

1. Parallel execution with only the SMP tasks that process the ROIs data in parallel in 283 OmpSs tasks in the 4 SMP resources of the AXIOM board.
2. Parallel execution of the application where the ROI_Tasks are mapped to the SMP resources and the *Anisotropic_Tasks* are mapped to one FPGA accelerator. In this configuration, 285 OmpSs tasks are created.
3. Parallel execution of the application where the ROI_Tasks are mapped to the SMP resources and the *Anisotropic_Tasks* are mapped to two FPGA accelerators. In this configuration, 299 OmpSs tasks are created.

Figure 59 shows that both the sequential and the parallel time decrease when the FPGA accelerators are added due to the offload of some parts on the accelerators. Moreover, with one and two accelerators, the execution time for the *Axiom_Bio_Identification* application decreases, and as well there are free CPU resources that can be used to perform additional tasks.

8.2 Energy Consumption Measurements of SHL application

In collaboration with the EVIDENCE Partner, the energy consumption of the execution of the application was measured. The setup used included the Lauterbach Power Trace tools, and the Lauterbach Analog Probe which has been connected to three rails (CPU, FPGA Fabric and DRAM). In order to do that, the AXIOM board shunt resistors have been wired out to a connector as shown in Figure 60. We connected the probes to these resistors to measure the energy consumption of different power domain.

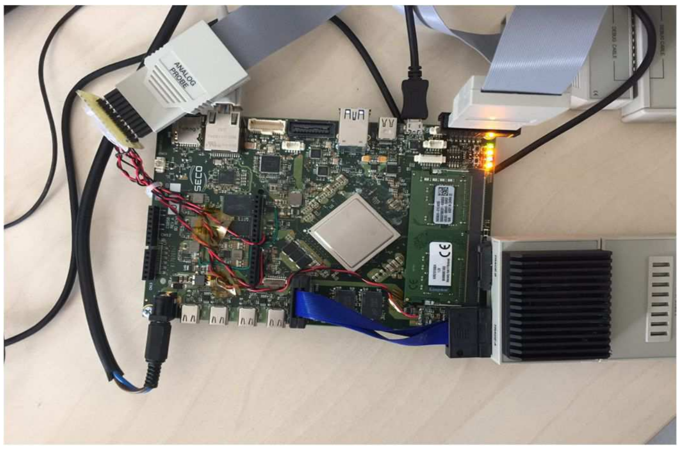


Figure 60 - The shunt resistors on the power rails have been wired to an external connector and connected to the Lauterbach Analog probe.

The traces of the power consumption of the executions of the four experiments presented in the preview paragraph were captured by sampling the current flow in the power lines of the AXIOM board described in the Table 1.

Table 6 Power lines under test

Bus	Label	Nominal Voltage (V)	Description	Trace Color
0.85V_INTFP	PM_VCC_INTFP	0.85	PS full-power domain supply	green
0.85V_VCCINT	PM_VCC_INT	0.85	PL internal power supply	red
1V2_DDR_PS	PM_1V2_DDR_PS	1.20	PS DDR supply	blue

We performed a set of experiments related to the four version of the SHL application described in the previous paragraphs, sampling the power consumption of the three rails of interest. As a result, it is possible to identify the various phases of execution, in particular:

- It is possible to identify when the SMP CPUs are running, highlighting 5 power consumption levels (0 to 4 CPUs running);
- It is possible to identify the phases of the FPGA activation, identifying when there are one or two FPGA accelerators running
- It is possible to identify when there is a substantial load on the DDR memory subsystem.

All power traces show a static power (around 0.9 W) consumed by the FPGA where no accelerators are running.

In particular, Figure 61 shows th*e* power trace of the sequential version (experiment 1) of the SHL application. In this case, the application runs on only one CPU core.

Figure 62 shows that the application is using all the four CPU cores (experiment 2), and for that reason the PS power consumption (green line) is greater than the sequential version. As it can be seen from Figure 62 also the RAM power consumption (blue line) is greater than the experiment 1, because all the cores use the memory during the computation.

The experiment 3 uses also one accelerator implemented in the FPGA. This is visible in Figure 64, where the PL power supply (red line) increases when the accelerator is working.

Finally, in Figure 63 the SHL application takes advantage of the parallel execution of the application on 4 SMP resources and 2 FPGA accelerators. The red line (PL power supply) shows clearly when one or two accelerators are working or not.

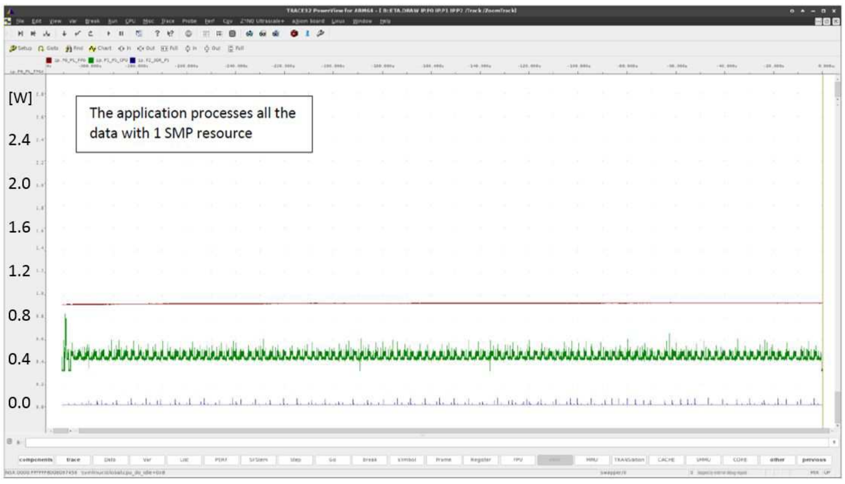


Figure 61 - Power trace sampled when running experiment number 1 (sequential execution). A single processor is constantly loaded for all the execution time.

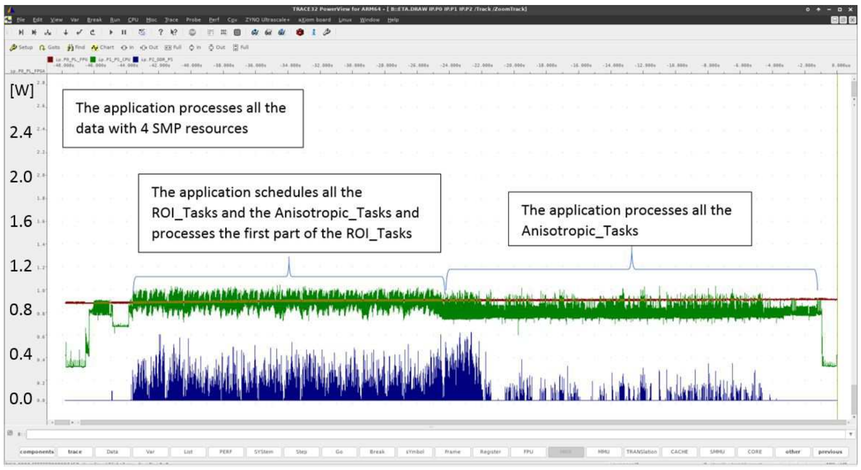


Figure 62 - Power trace sampled when running experiment 2 (Parallel execution only on SMP). In this case, all four cores are running, and a substantial access to the DRAM is also performed.

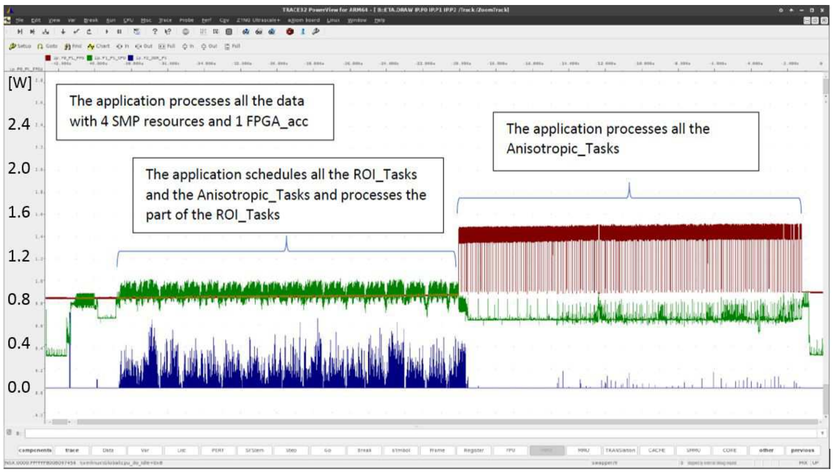


Figure 63- Power trace sampled when running experiment 3 (Parallel execution on SMP and 1 FPGA accelerator). As it can be seen, during the FPGA execution, the CPU is basically idle, with limited usage of the DDR memory.

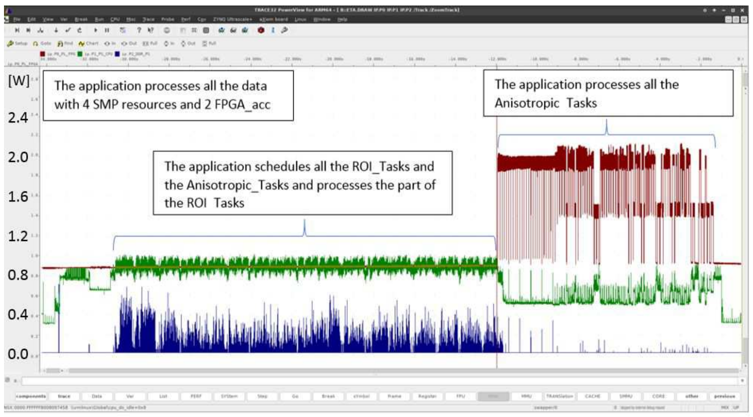


Figure 64- Power trace sampled when running experiment 4 (Parallel execution on SMP and 2 FPGA accelerator). In this case, the two power levels of the FPGA (1 and 2 accelerators running) are clearly visible.

The Lauterbach TRACE32 software is also able to perform the computation of the total energy consumed for the computation (in Joule), by integrating the three power line under test during the execution of the application Figure 65 shows the profile of the total energy spent over time on the experiment number 4 (shown in Figure 63).

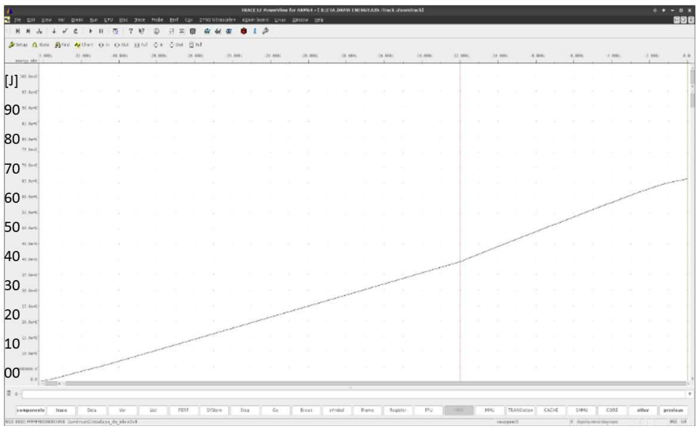


Figure 65- Trace of energy consumed by the 3 power lines under test in the experiment 4 (Parallel execution on SMP and 2 FPGA accelerator).

Figure 66 shows the total energy consumed by the four experiments on the AXIOM board. It is clear that the usage of the FPGA acceleration allowed a reduction of both the execution time and the energy consumption.

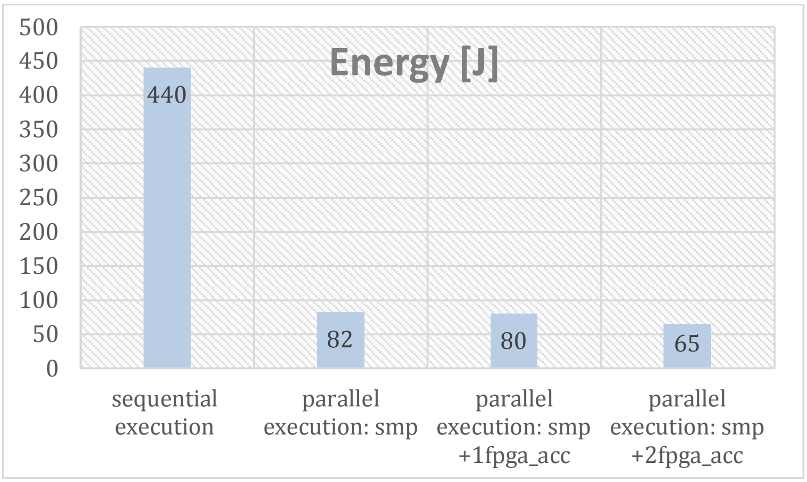


Figure 66- Energy consumption to process the application with different version of parallelism.

Finally, Figure 67 and Figure 68 shows a focus on specific parts of the power trace shown in Figure 63. In particular, we try to characterize the power trace mapping it to the application steps.

By observing the power consumption, in fact, we can identify how many resources are in use on the MPSoC. Figure 67 shows with the addition of dashed lines the number of cores and FPGA accelerators in use during the execution of SHL application (experiment 4).

In addition to that, it is possible to identify the application execution patterns. Figure 68 shows in particular a typical pattern on the experiment 4: in this case, the application executes 5 times the FPGA accelerators (red line) before collecting the results using the CPUs (green line) and the DDR RAM (blue line), that in fact have spikes every 5 FPGA executions.

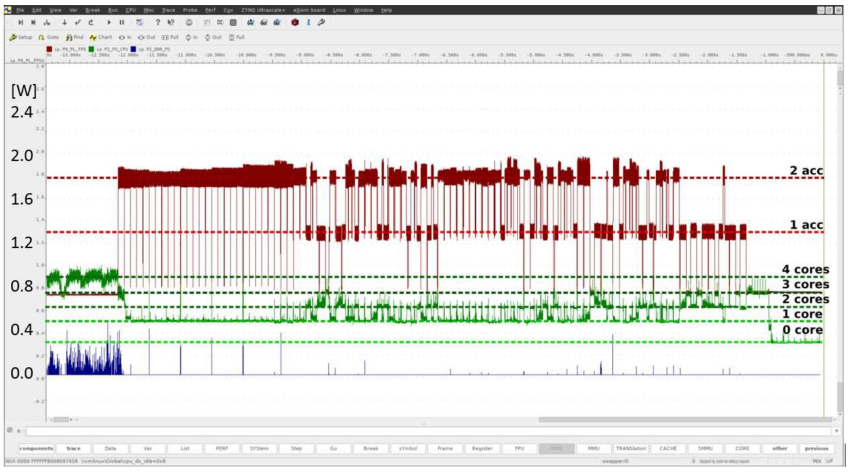


Figure 67- Resources in use on the SoC (experiment 4).

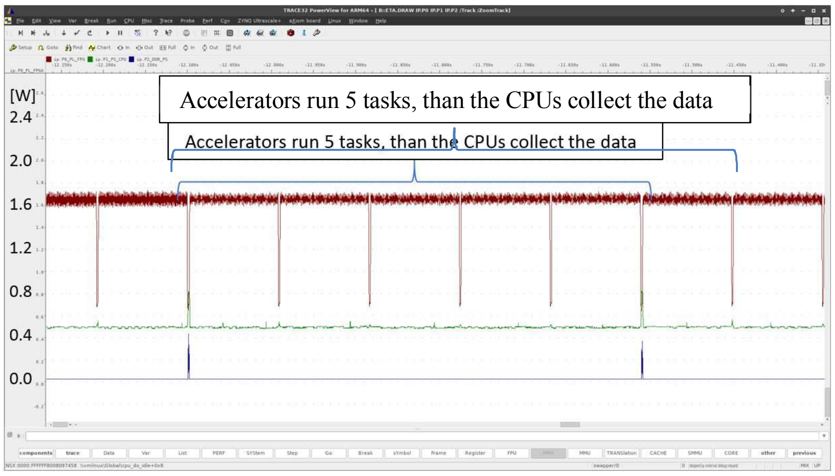


Figure 68- Execution pattern identified on the experiment 4.

9 Confirmation of DoA objectives and Conclusions

Here we describe how the deliverables conform to the DoA stated objectives.

PLANNED	DELIVERED
DELIVERABLE:	
• Performance and Energy Evaluation of AXIOM	Report

We have completed the design of important features of the AXIOM platform thanks to the developed tools and benchmarks. The system is now running. The hardware is fully working and several prototypes boards have been released (about 2 per partner). The software has been completed with the appropriate driver also for peripherals not yet existing at the time we started and not even once the AXIOM-board was available such as in the case of the XSMLL/DF-threads device and drivers.

Once the soft-IPs had been verified we were able to further optimize the performance of the system and we confirmed the initial hypothesis of good scalability.

The two key applications SHL and SVS were properly developed to exploit the capabilities of the AXIOM-board. Further improvements are possible, but we believe that we have met or exceeded all project objectives.

References

1. Dan Bonachea; GASNet Specification, v1.1. Report No. USB/CSD-02-1207. CS Division, EECS Department, University of California, Berkeley; October 2002; http://gasnet.lbl.gov/CSD-02-1207.pdf
2. Javier Bueno, Xavier Martorell, Rosa M. Badia, Eduard Ayguadé, Jesús Labarta; Implementing OmpSs support for regions of data in architectures with multiple address spaces. ICS 2013: 359-368 (2013).
3. Alejandro Duran, Eduard Ayguadé, Rosa M. Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, Judit Planas; OmpSs: a Proposal for Programming Heterogeneous Multi-Core Architectures. Parallel Processing Letters 21(2): 173-193 (2011).
4. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, Version 3.0; September 2012; http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf
5. A. Portero, A. Scionti, Z Yu, P Faraboschi, C Concatto, L Carro, A Garbade; Simulating the Future kilo-x86-64 core Processors and their Infrastructure, Proc. of the 45th Annual Simulation Symposium.
6. Some considerations about passive sharing in shared-memory multiprocessors CA Prete, G Prina, R Giorgi, L Ricciardi IEEE TCCA Newsletter, 34-40.
7. XSMLL API for AXIOM: https://git.axiom-project.eu/?p=XSMLL
8. OmpSs website: http://pm.bsc.es/ompss
9. COTSon website: http://cotson.sourceforge.net/
10. Argollo, E., Falcón, A., Faraboschi, P., Monchiero, M., and Ortega, D. 2009. COTSon: infrastructure for full system simulation. SIGOPS Oper. Syst. Rev. 43, 1 (Jan. 2009), 52-61
11. Palermo, G.; Silvano, C.; Zaccaria, V., "ReSPIR: A Response Surface-Based Pareto Iterative Refinement for Application-Specific Design Space Exploration," in Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.28, no.12, pp.1816-1829, Dec. 2009
12. Silvano, C.; Fornaciari, W.; Palermo, G.; Zaccaria, V.; Castro, F.; Martinez, M.; Bocchio, S.; Zafalon, R.; Avasare, P.; Vanmeerbeeck, G.; Ykman-Couvreur, C.; Wouters, M.; Kavka, C.; Onesti, L.; Turco, A.; Bondi, U.; Mariani, G.; Posadas, H.; Villar, E.; Wu, C.; Fan Dongrui; Zhang Hao; Shibin, T., "MULTICUBE: Multi-objective Design Space Exploration of Multi-core Architectures," in VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on , vol., no., pp.488-493, 5-7 July 2010
13. Giovanni Mariani, Aleksandar Brankovic, Gianluca Palermo, Jovana Jovic, Vittorio Zaccaria, and Cristina Silvano. 2010. A correlation-based design space exploration methodology for multi-processor systems-on-chip. In Proceedings of the 47th Design Automation Conference (DAC '10). ACM, New York, NY, USA, 120-125.
14. Mariani, G.; Palermo, G.; Silvano, C.; Zaccaria, V., "Multi-processor system-on-chip Design Space Exploration based on multi-level modeling techniques," in Systems, Architectures, Modeling, and Simulation, 2009. SAMOS '09. International Symposium, pp.118-124, 20-23 July 2009.
15. R. Giorgi, R. Badia, F. Bodin, A. Cohen, P. Evripidou, P. Faraboschi, B. Fechner, G. Gao, A. Garbade, R. Gayatri, S. Girbal, D. Goodman, B. Khan, S. Koliaï, J. Landwehr, N. Minh, F. Li, M. Lujàn, A. Mendelson, L. Morin, N. Navarro, T. Patejko, A. Pop, P. Trancoso, T. Ungerer, I. Watson, S. Weis, S. Zuckerman, M. Valero, "TERAFLUX: Harnessing dataflow in next generation teradevices ", ELSEVIER Microprocessors and Microsystems, Netherlands, Amsterdam, vol. 38, no. 8, Part B, 2014, pp. 976-990.
16. M. Solinas, M. Badia, F. Bodin, A. Cohen, P. Evripidou, P. Faraboschi, B. Fechner, G. Gao, A. Garbade, S. Girbal, D. Goodman, B. Khan, S. Koliaï, F. Li, M. Lujàn, A. Mendelson, L. Morin, N. Navarro, A. Pop, P. Trancoso, T. Ungerer, M. Valero, S. Weis, S. Zuckerman, R. Giorgi, "The TERAFLUX project: Exploiting the dataflow paradigm in next generation teradevices", IEEE Proc. 16th EUROMICRO-DSD, Santander, Spain, no. 6628287, 2013, pp. 272-279.
17. Blem, E.; Menon, J.; Sankaralingam, K., "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures," in High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on , vol., no., pp.1-12, 23-27 Feb. 2013
18. AMD Opteron A-series: http://www.amd.com/en-us/products/server/opteron-a-series
19. AMD K12 architecture: http://partner.amd.com/Documents/MarketingDownloads/en/AMD-FAD-2015-Codename-Decoder-FINAL.pdf
20. Li, Sheng, et al. "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures." Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture. ACM, 2009.
21. Xilinx Inc., "Zynq Series." [Online]. Available: http://www.xilinx.com/content/xilinx/en/products/silicon-devices/soc/zynq-7000.html
22. M. Banzi, *Getting Started with Arduino*. Sebastopol, CA: Make Books - Imprint of: O'Reilly Media, 2008.
23. Xilinx Inc., "Xilinx UltraScale Architecture." [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp435-Xilinx-UltraScale.pdf
24. S. Lyberis, G. Kalokerinos, M. Lygerakis, V. Papaefstathiou, D. Tsaliagkos, M. Katevenis, D. Pnevmatikatos, and D. Nikolopoulos, "Formic: Cost-efficient and scalable prototyping of manycore architectures," in FCCM, 2012, pp. 61–64.
25. E. Palazzetti, Getting Started with UDOO, Resha Raman, Ed. Packt Publishing, 2015.
26. J. Bueno, L. Martinell, A. Duran, M. Farreras, X. Martorell, R. Badia, E. Ayguade, and J. Labarta, "Productive cluster programming with OmpSs," *Euro-Par 2011 Parallel Processing*, pp. 555–566, 2011.
27. Heirman et. al. - ISPASS Tutorial The SNIPER multi-core simulator
28. Carl J. Mauer et al. Full system timing-first simulation. In SIGMETRICS02, pp. 108-116, June 2002'.
29. Falcon, A.; Faraboschi, P.; Ortega, D., "Combining Simulation and Virtualization through Dynamic Sampling," in Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on , vol., no., pp.72-83, 25-27 April 2007
30. R. Giorgi, "Transactional memory on a dataflow architecture for accelerating Haskell," WSEAS Trans. Computers, vol. 14, pp. 794–805, 2015.
31. R. Giorgi and P. Faraboschi, "An introduction to DF-Threads and their execution model," in IEEE MPP, Paris, France, Oct. 2014, pp. 60–65.
32. R. Giorgi and A. Scionti, "A scalable thread scheduling co-processor based on data-flow principles," ELSEVIER Future Generation Computer Systems, vol. 53, pp. 100–108, July 2015.
33. S. Weis, A. Garbade, B. Fechner, A. Mendelson, R. Giorgi, and T. Ungerer, " Architectural support for fault tolerance in a Teradevice dataflow system," Springer Int.l Journal of Parallel Programming, pp. 1–25, May 2014.
34. D. Theodoropoulos et al., "The AXIOM project (agile, extensible, fast I/O module)," in IEEE Proc. 15th Int.l Conf. on Embedded Computer Systems: Architecture, MOdeling and Simulation, July 2015.
35. R. Giorgi, "Scalable Embedded Systems: Towards the Convergence of High-Performance and Embedded Computing", Proc. 13th IEEE/IFIP Int.l Conf. on Embedded and Ubiquitous Computing (EUC 2015), Oct. 2015.

36. D. Theodoropoulos, S. Mazumdar, E. Ayguade, N. Bettin, J Bueno, S. Ermini, A. Filgueras, D. Jiménez-González, C. Álvarez Martínez, X. Martorell, F. Montefoschi, D. Oro, D. Pnevmatikatos, A. Rizzo, P. Gai, S. Garzarella, B. Morelli, A. Pomella, R. Giorgi, "The AXIOM platform for next-generation cyber physical systems", Microprocessors and Microsystems, v. 52, 2017, pp. 540-555,doi 10.1016/j.micpro.2017.05.018.
37. K. Stavrou *et al*., "Programming Abstractions and Toolchain for Dataflow Multithreading Architectures," *2009 Eighth International Symposium on Parallel and Distributed Computing*, Lisbon, 2009, pp. 107-114.
38. R. Jain, "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling", Wiley, May 1991.
39. D. Lilja, "Measuring Computer Performance: A Practitioner's Guide", Cambridge Univ. Press, 2005.
40. L. Eeckout, "Computer Architecture Performance Evaluation Methods", Morgan & Claypool Publishers, 2010.
41. The JUMP Software DSM System. http://www.snrg.cs.hku.hk/srg/html/jump.htm
42. Zynq-7000 Technical Reference Manual : http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf
43. Zynq UltraScale+ Technical Reference Manual: http://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf
44. S. Wong, A. Brandon, F. Anjam, R. Seedorf, R. Giorgi, Z. Yu, N. Puzovic, S. Mckee, Magnus Sjaelander and Georgios Keramidas, "Early Results from ERA – Embedded Reconfigurable Architectures", 9th IEEE Int.l Conf. on Industrial Informatics (INDIN), Lisbon, Portugal, Jul 2011, pp. 816-822.
45. ERA Benchmark suite: http://www.dii.unisi.it/~giorgi/ebs/
46. S. Wong, L. Carro, S. Kavvadias, G. Keramidas, F. Papariello, C. Scordino, R. Giorgi, S. Kaxiras, "Embedded reconfigurable architectures", ACM Proc. 2012 international conference on Compilers, architectures and synthesis for embedded systems (CASES), New York, NY, USA, 2012, pp. 2.
47. Wong Stephan, Carro Luigi, Rutzig Mateus and Matos Debora Motta, Giorgi Roberto, Puzovic Nikola , Kaxiras Stefanos, Cintra Marcelo, Desoli Giuseppe, Gai Paolo, Mckee Sally A., Zaks Ayal, "ERA - Embedded Reconfigurable Architectures", Springer New York, ISBN:978-1-4614-0061-5, Aug 2011, pp. 239-259.
48. VMBuilder website: https://launchpad.net/vmbuilder (accessed on January 2017).
49. Deboostrap website: https://wiki.debian.org/Debootstrap (accessed on January 2017).
50. Tesseract website information: https://en.wikipedia.org/wiki/Tesseract_(software) (accessed on January 2017).
51. Bash website: https://www.gnu.org/software/bash/ (accessed on January 2017).
52. R. Giorgi, "Exploring Future Many-Core Architectures: The TERAFLUX Evaluation Framework", Elsevier, 2017, pp. 33-72.
53. R. Giorgi, "Exploring Dataflow-based Thread Level Parallelism in Cyber-physical Systems", Proc. ACM Int.l Conf. on Computing Frontiers, New York, NY, USA, 2016, pp. 6.
54. A. Rizzo, G. Burresi, F. Montefoschi, M. Caporali, R. Giorgi, "Making IoT with UDOO", Interaction Design and Architecture(s), vol. 1, no. 30, Dec. 2016, pp. 95-112.
55. L. Verdoscia, R. Giorgi, "A Data-Flow Soft-Core Processor for Accelerating Scientific Calculation on FPGAs", Mathematical Problems in Engineering, vol. 2016, no. 1, Apr. 2016, pp. 1-21.
56. S. Mazumdar, E. Ayguade, N. Bettin, S. Bueno J. and Ermini, A. Filgueras, D. Jimenez-Gonzalez, C. Martinez, X. Martorell, F. Montefoschi, D. Oro, D. Pnevmatikatos, A. Rizzo, D. Theodoropoulos, R. Giorgi, "AXIOM: A Hardware-Software Platform for Cyber Physical Systems", 2016 Euromicro Conf. on Digital System Design (DSD), Aug 2016, pp. 539-546.
57. R. Giorgi, N. Bettin, P. Gai, X. Martorell, A. Rizzo, "AXIOM: A Flexible Platform for the Smart Home", Springer Int.l Publishing, Cham, 2016, pp. 57-74.
58. P. Burgio, C. Alvarez, E. Ayguade, A. Filgueras, D. Jimenez-Gonzalez, X. Martorell, N. Navarro, R. Giorgi, "Simulating next-generation cyber-physical computing platforms", Ada User Journal, vol. 37, no. 1, Mar. 2016, pp. 59-63.
59. R. Giorgi, S. Mazumdar, S. Viola, P. Gai, S. Garzarella, B. Morelli, D. Pnevmatikatos Dionisios and Theodoropoulos, C. Alvarez, E. Ayguade, J. Bueno, D. Filgueras Antonio and Jimenez-Gonzalez, X. Martorell, "Modeling Multi-Board Communication in the AXIOM Cyber-Physical System", Ada User Journal, vol. 37, no. 4, December 2016, pp. 228-235.
60. Xilinx Power Estimation tool website: https://www.xilinx.com/products/technology/power/xpe.html (accessed on January 2017).
61. A Rizzo, F Montefoschi, M Caporali, A Gisondi, G Burresi, R Giorgi, Rapid Prototyping IoT Solutions Based on Machine Learning, European Conference on Cognitive Ergonomics 2017, 184-187
62. .R. Giorgi, "AXIOM: A 64-bit reconfigurable hardware/software platform for scalable embedded computing", 6th Mediterranean Conf. on Embedded Computing (MECO), June 2017, pp. 113-116.
63. B.W.L. Cheung, C.L. Wang, Kai Hwang; "JUMP-DP: A Software DSM System with Low-Latency Communication Support", Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA2000), pp. 445-451, June 2000.
64. E. Gabriel, G.E. Fagg, G. Bosilca, T. Angskun, J.J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R.H. Castain, D.J. Daniel, R.L. Graham, and T.S. Woodall, "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation", In Proc. 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, Sep. 2004
65. Xilinx Inc., "Aurora 64B/66B v11.2", October 2017
66. Texas Instruments, INA219 datasheet
67. AXIOM Wiki, Power Monitor Tool document (https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTATION/BSP/PMT)
68. AXIOM Consortium "D4.3-Evaluation of the compiler and tools infrastructure", January 2018.
69. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug871-vivado-high-level-synthesis-tutorial.pdf (access on January 2018).
70. L. Verdoscia, R. Vaccaro, and R. Giorgi, "A clockless computing system based on the static dataflow paradigm," in Proc. IEEE Int.l Workshop on Data-Flow Execution Models (DFM-2014), Aug. 2014, pp. 30–37.
71. Daugman, J. G., "How iris recognition works", IEEE Transactions on Circuits and Systems for Video Technology, 2004, vol. 14, no. 1, pp. 21-30.