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**

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

PROJECT NUMBER: 645496

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

D7.2 – Design Space Exploration (DSE) on the prototype for the AXIOM

Due date of deliverable: 31st January 2017
Actual Submission: 07th February 2017 (agreed due date extension)

Start date of the project: 1st February 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**
0.1	02.02.2017	Roberto Giorgi	UNISI	v0.1
0.2	06.02.2017	Roberto Giorgi	UNISI	v0.1

Release Approval

Name	**Role**	**Date**
Roberto Giorgi	WP Leader	06.02.2017
Roberto Giorgi	Project Coordinator for formal deliverable	07.02.2017

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

Roberto Giorgi
Department of Information Engineering and Mathematics
University of Siena, Italy – (UNISI)

Stefano Viola
SECO S.r.L.

TABLE OF CONTENTS –

TABLE OF FIGURES –

GLOSSARY

ACP – Accelerated Coherency Port: an ARM AXI bus for (one-way) coherent operations
AEP – AXIOM Evaluation Platform
AXI – a proprietary protocol for buses introduced by ARM Ltd
AXIOM-acc – an FPGA accelerated system that performs a given function
AXIOM-arch – the architecture of an AXIOM (module or) board
AXIOM-core – the cores where the computations run in an AXIOM board
AXIOM-fpga – the programmable logic part in an AXIOM board
AXIOM-link – the interconnects that permits board-to-board communication in AXIOM
Bitstream – the binary code used for configuring the PL
BIOS-ROM – the Firmware of the system, i.e., the startup code when the machine is powered up
BRAM – Block-RAM: a fast RAM that is available in the FPGA slices (in smaller blocks)
BSD -- BroadSword Document – In this context, a file that contains the SimNow machine description for a given Virtual Machine
BSP – Board Support Package: the necessary software to run the system
CUDA – NVIDIA programming model for GPUs
Conduit – A software stub that connects GASNet to a given network protocol or programming model
Device – in this context: The physical system runs a 'device-tagged' part of the code
DoA – Description of Action (acronym set by the European Commission)
DTS -- Distributed Thread Scheduler
DMA – Direct Memory Access: a separate master that can take over local memory transfers
DSE – Design Space Exploration
DSM – Distributed Shared Memory
eMMC – Embedded Multi Media Card
FPGA – Field Programmable Gate Array
FPGA-device – a specific accelerator that is implemented on the FPGA
GASNet – Global Address Space over Network: is a language-independent, low-level networking layer that provides network independent communication primitives
Infiniband – a high-performance (costly) NI
IMGPKG – Image-Package: a package that contains the hard-drive image, system room, BSD snapshots and other relevant files to start up a computer board or a virtual machine
IP – Intellectual Property system (either hardware or software)
Mercurium – the OmpSs compiler
Nanos++ -- the OmpSs runtime
MGT – Multi-Gigabit Transceiver
MPI – Message Passing Interface: library for writing portable message-passing programs
PL – Programmable Logic: the purely FPGA part of a SoC like ZYNQ
PS – Processing System: the hardwired IPs of a FPGA-hybrid SoC like ZYNQ
MPSoC – Multi-Processor SoC
NI – Network Interface
OpenCL – Khronos group programming model for heterogeneous architectures
OmpSs – Extension of OpenMP programming model to support task dataflow programming
OmpSs@FPGA – FPGA extension of OmpSs
OmpSs@Cluster – Cluster extension of OmpSs
PCIe – PCI Express – standard for peripherals interconnection
PHY – the physical implementation of the network interface
QEMU – An open-source package for system virtualization
QSPI – Quad Serial Peripheral Interface
RDMA – Remote DMA: a DMA that can work from one computer to another computer
ROI – Region of Interest
SoC – System on Chip
USB OTG – Universal Serial Bus On The Go
XSMLL – (pronounced "X-SMALL") eXtended Shared-Memory Low-Level API
X-Thread – a self-contained thread that can be distributed across boards through XSMLL
ZYNQ -- A System-on-Chip commercialized by XILINX, which includes FPGA and CPUs

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 second year of the AXIOM project, we 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 main challenge for this reporting period was to switch from the 32-bit to the 64-bit platform (exceeding the initial goals of the project). This had a manageable but deep impact on several tasks of the project, including the initially defined AEP: we needed two other prototyping boards based on the 64-bit Zynq Ultrascale+ (not available until late July 2016).

The AEP still encompasses several tools, namely:

- The HP-Labs COTSon simulator, which is a powerful virtual platform that permits to run off-the-shelf operating systems like Linux and totally decouples the functional modeling and the timing (architectural) modeling; in particular, we are interested to develop the timing models for novel AXIOM components and decide the appropriate partitioning of functionalities between software and hardware; the XSMLL API specification and deployment shows here such capabilities;
- The DSE tools, that are the necessary glue to properly manage the experiments, parallelize them, collect and visualize the results;
- The prototyping and development boards from the FPGA vendor; our choice is to use Xilinx boards in this project, although our platform is in principle portable to other vendor boards.

This document briefly illustrates the evolution of such tools, the new challenges, the integration of various methodologies and the connection with several other workpackages.

We show in particular how those tools permitted to derive early in the Design process useful information to focus the design efforts toward optimizing L2 cache usage and the interconnects.

Therefore we believe that the objectives of the second year have been meet or exceeded (e.g. with the experiment involving the impact of the Operating System and of several popular Linux distributions).

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, MYINSTALL, MYDSE, ESTRAI, RISGRAPH. The COTSon and MYDSE are by far the most complex ones. We do not describe here the Xilinx XC-706 platform (our reference platform) for which documentation is available from the vendor [42].

In this reporting period we extended all those tools and introduced a new tool partly dictated by the need of having the flexibility in changing the reference Operating System (and related packages and distribution, and perform further measurement on the influence of the Operating System itself. We believe that the new tool permits us to move more quickly on more sophisticated measurements and to deploy more benchmarks in the near future.

We show how our concept ideas have been translated into practical tools and results. In D5.2, we show some initial results comparing our XSMLL model (only possible at the time being on the simulation platform) with other programming models such as Cilk and OpenMPI.

1.1 Document structure

This document is organized as follows:

- In Section 2, we briefly recall the AXIOM platform high-level architecture and we highlight the major challenges in the Reporting Period 2 (RP2);
- In Section 3, we illustrate the work flow of some of the most important tools behind the AXIOM Evaluation Platform;
- In Section 4, we highlight the main options that we considered for the power estimation;
- In Section 5, we present a test case based on the matrix multiplication benchmark that uses the XSMLL execution model. In particular, we highlight the impact of the Operating System on the global performance and how such impact varies with the OS distribution.

1.2 Relation to other deliverables

Deliverables D3.1, D3.2 refer to the selection of initial kernels and benchmarks to be used for the initial analysis (this is also detailed in this deliverable).

Deliverables D4.1, D4.2 refer to the programming model (OmpSs) and other activities to build a coherent software and hardware stack in coordination with WP4, WP5, WP6 (and this Workpackage).

Deliverables D5.1, D5.2, D5.3 refer to the support for the OS, the runtime and the remote communication mechanisms.

Deliverable D6.1 describes the initial architecture for the first AXIOM board prototype (specified at month 9), D6.2 describes the choices of the new board and D6.3 is the new board itself.

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.

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

- Task T4.4 (month 13-24) Code Generation for the AXIOM environment.
- Task T5.1 Operating System (month 1-27(extended)).
- Task T5.2 (month 1 -24(extended)) Remote Memory Access (UNISI, BSC, EVI and FORTH).

2 The AXIOM platform

For easier reading, we briefly recall here how the AXIOM platform is architected (cf. D7.1). The architecture is based on the following pillars (Figure 1), and see also Deliverables D6.1, D6.2:

P1 FPGA [42][43], i.e. large Programmable Logic for acceleration of functions, soft-IPs, implementing specific AXIOM support for interconnects and scaling,

P2 General Purpose Cores, to support the OS and for running parts that make little sense on the other accelerators,

P3 High-Speed, Inexpensive Interconnects (similar to [24]) to permit scalability and deverticalise the technology, e.g., for toolchains,

P4 Open-Source Software Stack,

P5 Lower-Speed Interface for the Cyber-Physical world, such as Arduino [22] connectors, USB, Ethernet, WiFi like, e.g., in the UDOO [24].

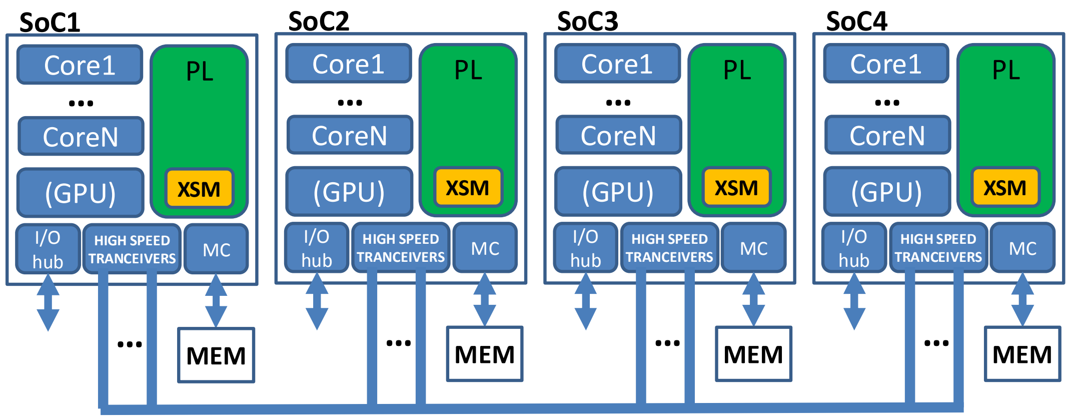


Figure 1: AXIOM Scalable Architecture. An instance consisting of four boards, each one based on the same System-on-Chip (SoC). GPU is an optional component (MC=Memory Controller. PL=Programmable Logic. XSM=eXtended Shared Memory).

During the second year of the project (April 2016), we faced a major challenge as we got the news that the long-awaited 64-bit platform from major FPGA providers like Xilinx would have been available in a reasonable time. In particular, both at the Embedded World exposition in Nurnberg and at the DATE conference in Dresden, we received information and brochures about the soon to be re-leased Xilinx ZYNQ-Ultrascale+ ZU9EG, which would have been much more ideal for the needs of the AXIOM project.

As a consequence of this information we had several internal meetings to evaluate the feasibility of the change to 64-bits and we agreed with our Project Officer to take this challenge and move our architecture to the "64-bit based" design rather than the "32-bit design". We all believe that this choice can put us in a much more competitive position as many embedded boards (including, e.g., the well-known Raspberry-Pi platform) moved to 64 bits. Xilinx guarantees support for the ZYNQ-Ultrascale+ until 2030.

In the WP7, platform agnostic given that recent studies [17] have considered not so important the difference between CISC-like (e.g., X86_64 [18]) and RISC processors (e.g. ARM [19]).

Partner SECO also had the possibility to access documentation and boards for the ZYNQ-Ultrascale+ [21] [23] earlier, thanks to special agreements with Xilinx. Partners BSC, FORTH, UNISI got the first ZYNQ-Ultrascale+ boards between July and September 2016, although they were generic designs which do not allowed us an easy mapping to the AXIOM architecture outlined above but helped us for the initial tests on 64-bits. Some first experiments were anyway done as documented in other deliverables (D4.2 and D5.2).

Partner BSC, EVI, FORTH, SECO, and UNISI had to migrate the new design, tools, drivers, board-support packages to the 64-bit case. This step had quite a lot of impact on our planned tasks.

In particular, UNISI, with the help of the involved partners (BSC, EVI, FORTH, SECO) reviewed the existing tools (like myinstall.sh, mydse.sh under the DSETOOLS umbrella, cf. list of versioned software and description of tools below), extended some of them and created further tools.

One other important consequence was the need to adapt the new board to more recent Operating System distributions, also because of possible problems in the filesystem support: partner SECO asked to move the supported Operating System distribution "Ubuntu 16.04 LTS (Long Term Support)". Please note that in task T7.1 all partners agreed to use "Ubuntu 14.04 LTS", therefore this was another (smaller but with deep impact) challenge that we had to face. Therefore, in August 2016, we had a consultation among all partners and finally in September we ratified this move to the Ubuntu 16.04 LTS. This choice had a large impact because both Xilinx tools, simulation tools at any level and from several vendors and very importantly, our internal tools had to be adapted (sometimes with unpredicted bugs and consequences).

We are currently quite happy with the migration move although it required a substantial effort. In particular, this required us to better analyze the impact of the Operating System distribution (cf. Section 5) and to provide tools (useful for much greater flexibility and independence from the Operating System distribution): we created therefore the following tools:

- BOOTSTRAP to easy the installation on different platforms of the simulation tools, possibly resolving the installation of the necessary dependent packages on the host
- BSDBUILDER to script the creating of the "virtual machine" that reflected the simulated system (e.g., by flexibly choosing the number of cores in each node from 1 to 8); this script also

creates the snapshots for the DSE in order to restrict the analysis to the Region Of Interest (ROI) (cf. D7.1) [15] [16].

- GENIMAGE to generate the "main storage" hd-image related to a given OS distribution (in our tests we are now able to automatically generate such images for 4 different OS distributions including the initially chosen Ubuntu-14.04 –for reference- and Ubuntu-16.04); details are discussed in Section 3.5.
- GEN-IMGPKG and INST-IMGPKG respectively create a package with BSDs, HD-image, BIOS-ROM ready to be deployed on the simulated system and install it in the proper location of the simulation host (this operation was quite time-consuming (e.g. 30 minutes each time per host since the files are several GiBs). This operation also has to be repeated on each simulation hosts and we wanted to be able to replicate the same environment at any partner location and in the future for any researcher who needs that. Although this is a less frequent operation, it has to be performed in a repeatable and well-automatized way to avoid any misinterpretation of simulation results[1].

3 The Importance of Tools

Tools are fundamental for designing a complex system like AXIOM. In WP7 all the partners contributed to make available a number of important tools for making the system work and to properly design and test it in particular in the early stages, when the actual platform might not be available.

The DSE tools developed in the AXIOM project aim at providing the necessary substrate to implement a proper scientific methodology for experimentation. An important aspect is the open–access [36] [37] and reproducibility of the experiments [38] [39] [40].

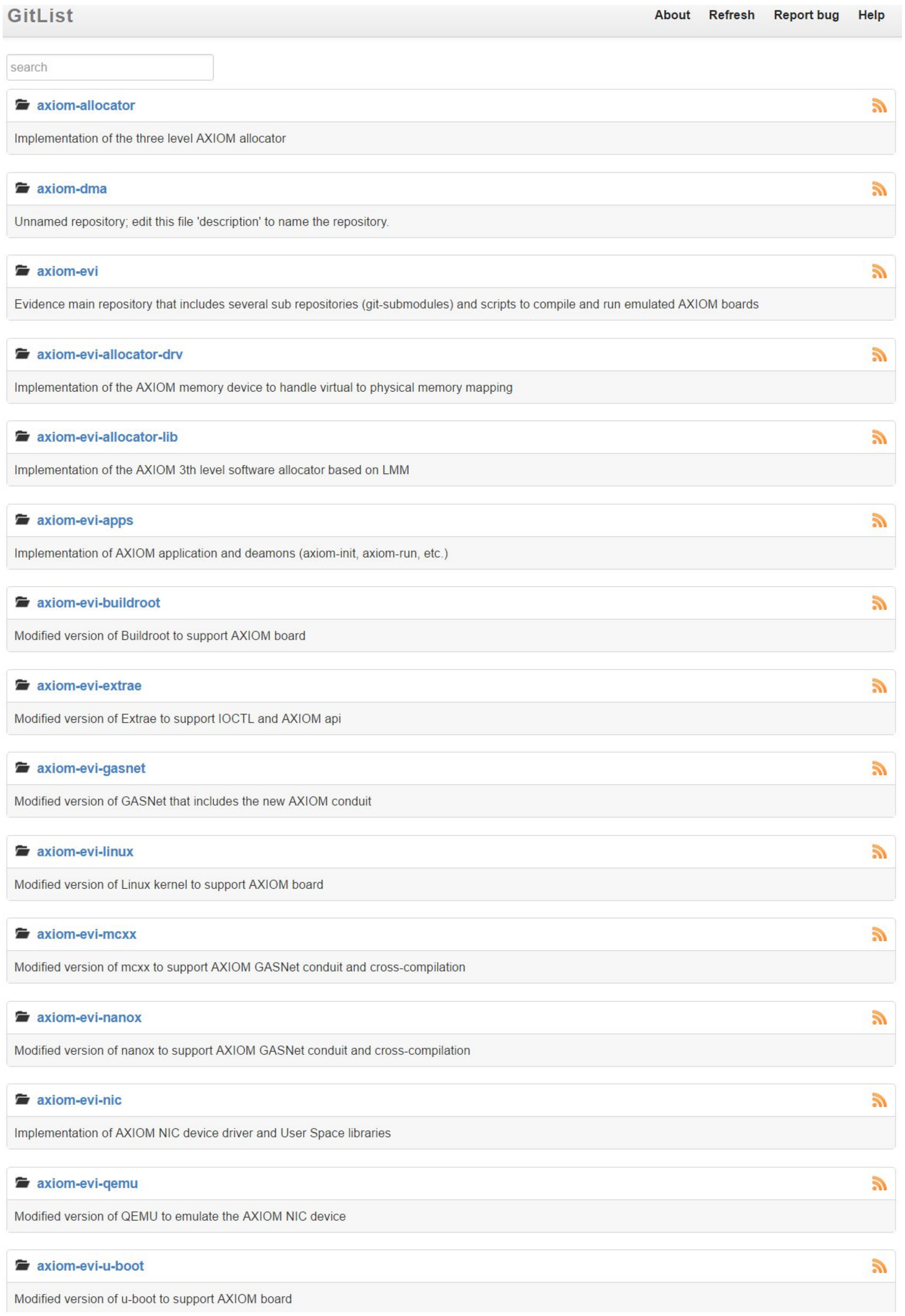
This is even more important considering that the first prototype board was planned to be available at the end of month m24 of the project. Therefore, our prototypes are based on commodity hardware, different FPGA prototype boards and more importantly on different simulation tools in particular based on our reference simulator COTSon and on QEMU (cf. D5.3 and D4.2).

The tools are documented through README files and in many cases through the DOXYGEN [61] documentation system.

3.1 The internal versioning repositories of the tools, software, IPs

Below we report a snapshot of the current list of tools, software, IPs that are available in our internal repositories (Figure 2). Some of the listed software packages are also described more in detail in D5.3, therefore we discuss here some examples of the ones used more closely in WP7.

[1] E.g., it was noted that the results may change depending on the version of the glibc (GNU standard C-library) in the system; for that we reason our methodology for comparisons normally includes to generate the static binaries on the oldest machine and then deploy the exact same binary on every guest.



axiom-allocator
Implementation of the three level AXIOM allocator

axiom-dma
Unnamed repository; edit this file 'description' to name the repository.

axiom-evi
Evidence main repository that includes several sub repositories (git-submodules) and scripts to compile and run emulated AXIOM boards

axiom-evi-allocator-drv
Implementation of the AXIOM memory device to handle virtual to physical memory mapping

axiom-evi-allocator-lib
Implementation of the AXIOM 3th level software allocator based on LMM

axiom-evi-apps
Implementation of AXIOM application and deamons (axiom-init, axiom-run, etc.)

axiom-evi-buildroot
Modified version of Buildroot to support AXIOM board

axiom-evi-extrae
Modified version of Extrae to support IOCTL and AXIOM api

axiom-evi-gasnet
Modified version of GASNet that includes the new AXIOM conduit

axiom-evi-linux
Modified version of Linux kernel to support AXIOM board

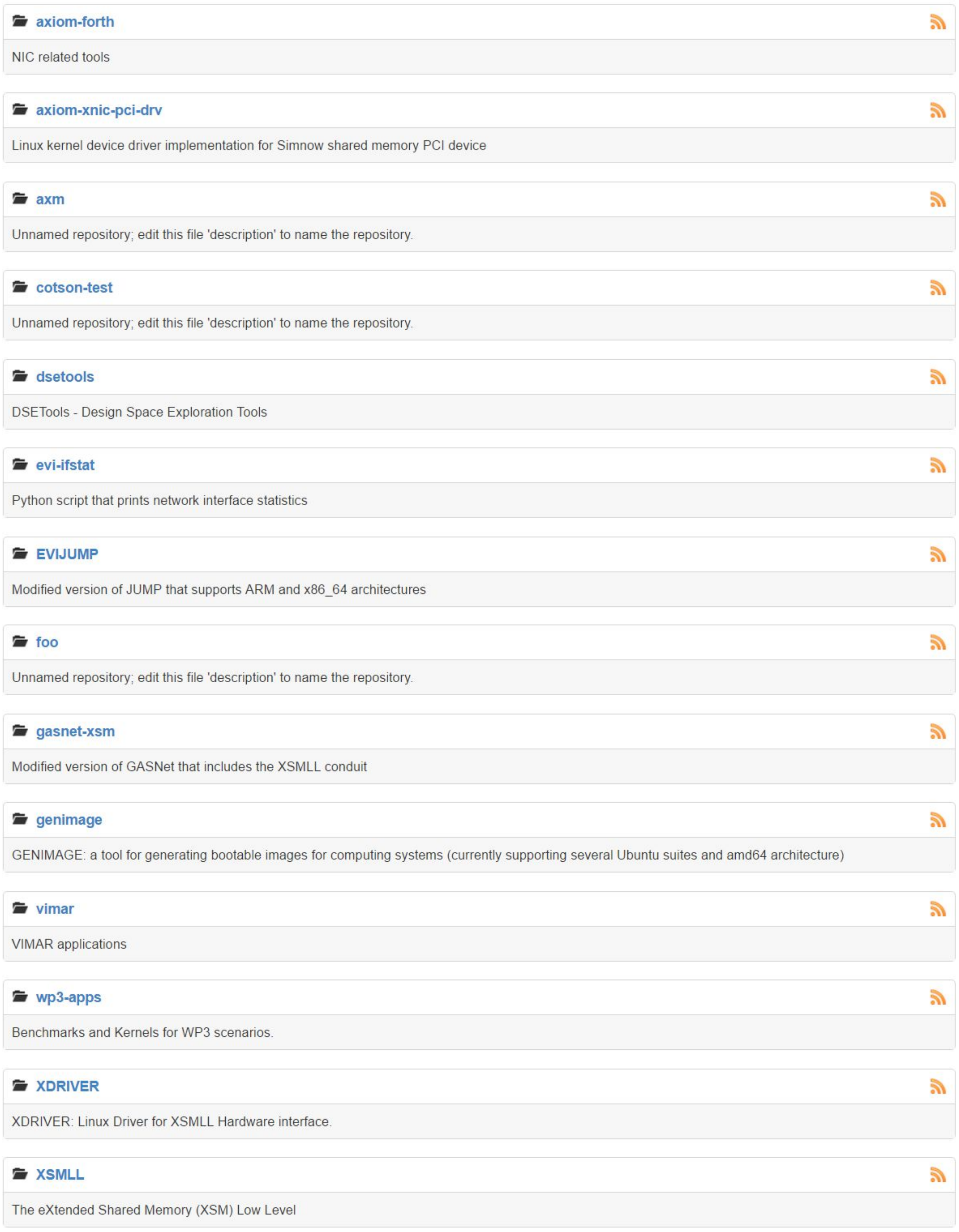
axiom-evi-mcxx
Modified version of mcxx to support AXIOM GASNet conduit and cross-compilation

axiom-evi-nanox
Modified version of nanox to support AXIOM GASNet conduit and cross-compilation

axiom-evi-nic
Implementation of AXIOM NIC device driver and User Space libraries

axiom-evi-qemu
Modified version of QEMU to emulate the AXIOM NIC device

axiom-evi-u-boot
Modified version of u-boot to support AXIOM board



axiom-forth
NIC related tools

axiom-xnic-pci-drv
Linux kernel device driver implementation for Simnow shared memory PCI device

axm
Unnamed repository; edit this file 'description' to name the repository.

cotson-test
Unnamed repository; edit this file 'description' to name the repository.

dsetools
DSETools - Design Space Exploration Tools

evi-ifstat
Python script that prints network interface statistics

EVIJUMP
Modified version of JUMP that supports ARM and x86_64 architectures

foo
Unnamed repository; edit this file 'description' to name the repository.

gasnet-xsm
Modified version of GASNet that includes the XSMLL conduit

genimage
GENIMAGE: a tool for generating bootable images for computing systems (currently supporting several Ubuntu suites and amd64 architecture)

vimar
VIMAR applications

wp3-apps
Benchmarks and Kernels for WP3 scenarios.

XDRIVER
XDRIVER: Linux Driver for XSMLL Hardware interface.

XSMLL
The eXtended Shared Memory (XSM) Low Level

Figure 2: AXIOM internal repositories for the various tools and software used in this project. For each package, in blue we indicate its name and in black a brief explanation.

3.2 Tools for the Design Space Exploration (DSE)

Below we present the main functionalities of some of the tools developed within WP7, trying to highlight also the integration with other software and components that have been realized in the AXIOM project. Design Space Exploration and its automation is an important part of modern performance evaluation and power estimation methodologies [11] [12] [13] [14] [15] [16]. Recent projects like MULTICUBE [12] have proposed systematic approaches for DSE.

3.3 Modelling Environment installation: MYINSTALL tool

The MYINSTALL tool takes care of preparing the whole installation environment. Some of the packages (e.g., AMD SimNOW) needs to be downloaded separately to respect the licensing terms but once downloaded the installation process takes place with a single command. The tool has been improved and progressively adapted to the new needs.

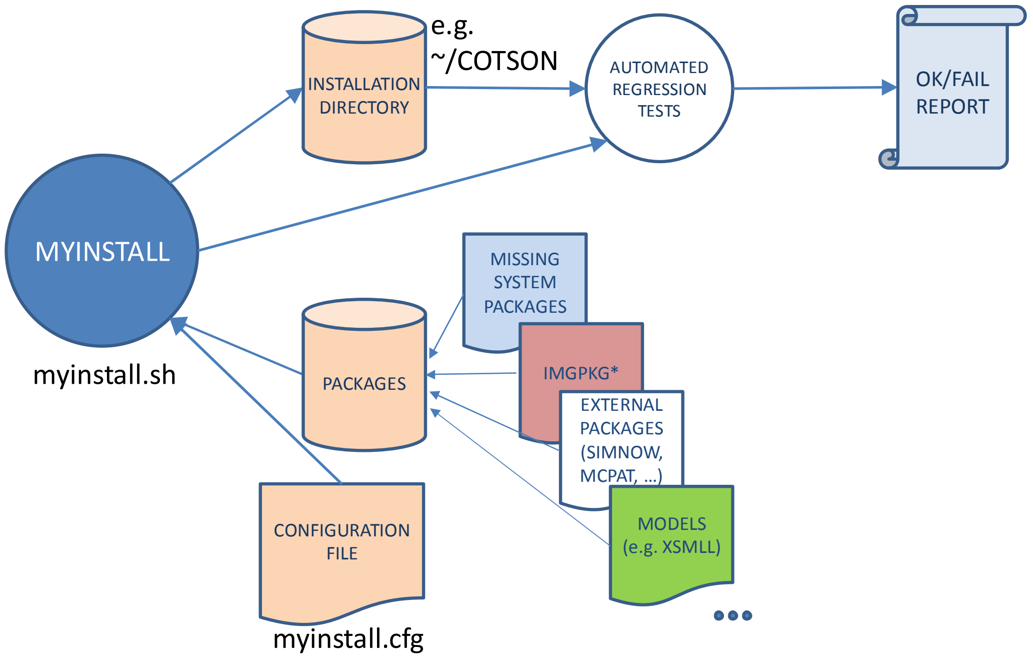


Figure 3: TOOLFLOW for the MYINSTALL tool. MYINSTALL prepares the whole environment for simulation-based Design Space Exploration with a single command. (*IMGPKG is described separately.)

This is currently a 50KiB Bash script, which allows the user to install a reference development environment in particular based on the HPLabs COTSon framework in about 10-15 minutes. In this RP2 we introduced the possibility to add additional models as part of the installation process (this was needed for example for the XSMLL [31] [32] [36] – cf. D5.2) and the possibility to install so-called Image-Packages (IMGPKG for short) that is explained in Section 3.4 below.

3.4 Managing Image-Packages: GEN-IMGPKG, INST-IMGPKG

These two smaller tools are needed for generating a single package (GEN-IMGPKG.SH) and deploy the package (INST-IMGPKG.SH). The package is generated by simply using files generated by another more complex tool (GENIMAGE.SH, which is described in Section 3.5 below).

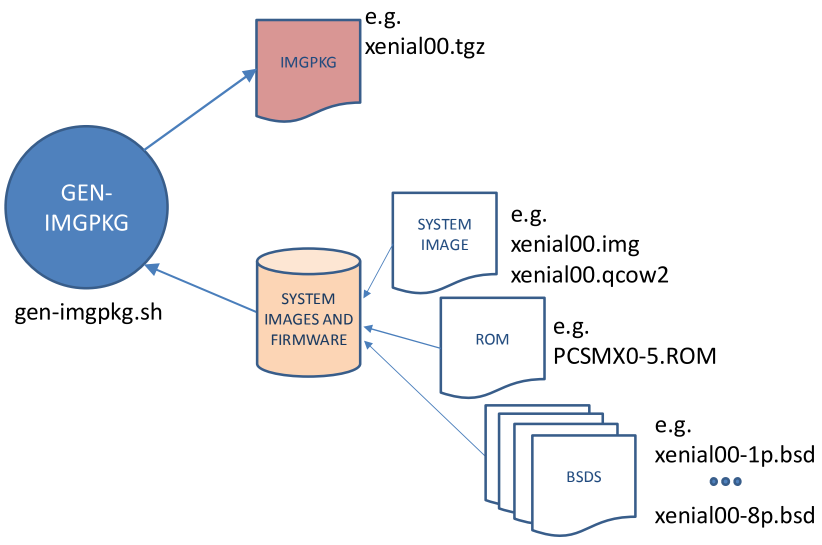


Figure 4: TOOLFLOW for the GEN-IMGPKG tool. The tool groups is single package several system file.

The IMGPKG (Image-Package) contains in a single file what is needed (Figure 4) to run a new platform setup in either a physical system or a virtualized platform, including:

- a description of the hardware (BSDs files)
- the BIOS-ROM
- the hard disk image (typically several GiBs) with the Operating System, System software and packages, libraries, etc. (i.e., the basic Ubuntu 16.04LTS distribution, the additional software, the user applications)

The INST-IMGPKG takes care of installing in the proper location in the filesystem (e.g. /opt/cotson) such files that are contained in the IMGPKG for running the other tool MYDSE for the Design Space Exploration. Please note that during the evolution of the project we needed to work on several IMGPKGs, so we needed to develop a systematic approach to manage the many system files. Therefore, we have modified the MYDSE tool (see Section 3.6) in order to manage different experiments on different IMGPKGs and keep separate and at the same time being able to compare several output files.

Please also note that at any time the two tools GEN-IMPKG and IMGPKG can be invoked without the need to repeat an installation from scratch with the MYINSTALL tool.

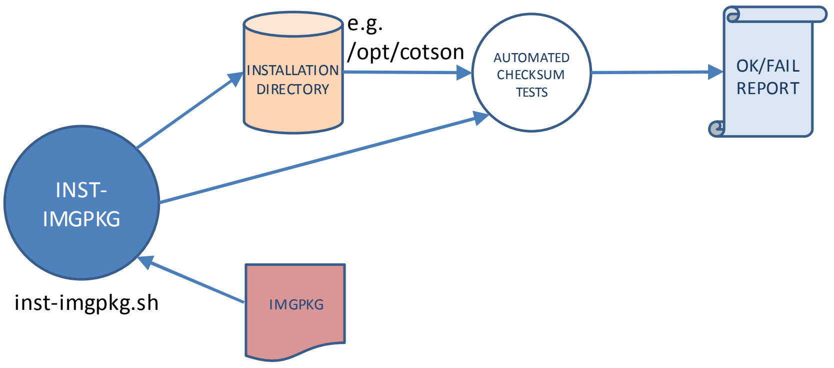


Figure 5: TOOLFLOW for the INST-IMGPKG tool. The tool installs a given set of system files (e.g., xenial00) in the appropriate location and verifies that those files are properly installed (e.g., by verifying checksums).

3.5 Generating Images and BSDs: GENIMAGE tool

The GENIMAGE tool is the key component for generating the hard-drive image for the AXIOM machine. This is based in turn on slightly modified version of Linux packages like VMBuilder [48] and Debootstrap [49]. In particular, we agreed with partner SECO to generate images based on the Debootstrap tool so we can more easily synchronize the necessary packages provided with the board (cf. D6.3, Board Support Package or BSP). Some other software components like the Drivers, Runtime components generated by partner EVI can also be integrated and updated at any time in the GENIMAGE tool workflow. In particular, this step involves a verification of the actual boot of the virtualized system, in order to be sure that all components are in place and that a performance/energy measurement can take place after the system has booted and can execute the user applications. The user prompt is recognized through an OCR recognition process based on the Tesseract package [50].

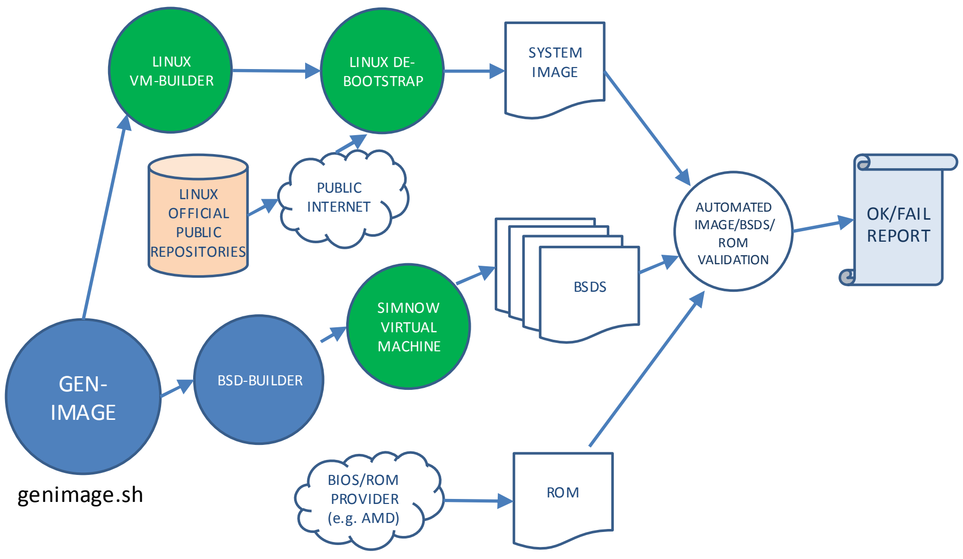
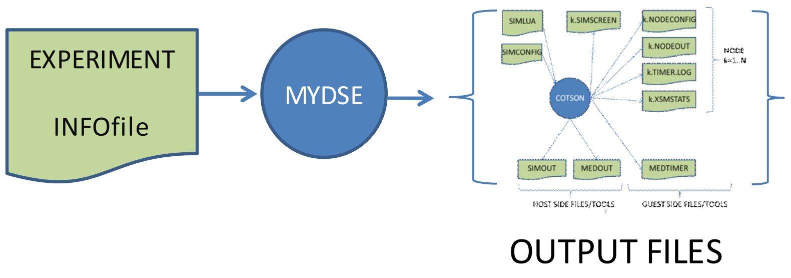


Figure 6: TOOLFLOW for the GENIMAGE tool. The system image (hard-drive image) and BSDs are generated based on the project (or user) requirements. The virtualized system is booted and stopped at the root prompt. An automated validation through Tesseract image recognition package of the screen is used to validate that generated images, BSDs, ROM are working properly.

3.6 Performing Design Space Exploration Experiments MYDSE tool

The MYDSE tool has been already described in D7.1. We briefly recall here its main purpose for easier reading. The MYDSE tool is the most complex of our tools (about 220 KiB of Bash scripting without counting several additional library scripts) and serves for managing the experiments of multiple design points on a cluster of machines. MYDSE manages the underlying simulator (our improved version of the HPLabs COTSon framework) [5] [9] [10].



Example
(changing no. of cores, no. of nodes, cache configuration):

```
$ mydse.sh T301-ncc
```

Figure 7: TOOLFLOW for the MYDSE tool. The tools manages an experiment involving many design space points and allows distributing the simulations on a cluster of machine for reducing simulation time.

Tens of files are generated for each simulation point (Figure 7) and therefore they need to be carefully stored to avoid confusion and in such a way that we can later mine the data to extract useful information. In particular, each simulation is tagged based on the date of the experiment, the host machine where the simulation has been performed and the user that launched the simulation besides the possible parameters that can be modified in the DSE. For example:

```
usgiorgi_madelta_rr751_da1701131757_moxsml1_apmmx_si128+8+i_no04_co04_wo04_tisimple
+3M_hdtrusty00_l132kB_l2256kB_l34096kB_ts1+6+3+100+10_cptimer0_ni10000M+0.1.timer.l
og
```

This indicates that the user ('us') 'giorgi' has launched a simulation on the machine ('ma') 'delta' with the revision of the framework ('rr') number '751' on the date ('da') 20170131 hours 17:51. The rest of the parameters in the filename identify the simulation point with a similar syntax, which is derived from the parameters in the INFOFILE (illustrated below) in an automated way. In the following figure (Figure 8), such filename is indicated with "SIMULATION". Raw value of the metrics of interest are stored in files which end with ".timer.log" and other information for validation of such metrics and other output messages are stored in the ".out" files.

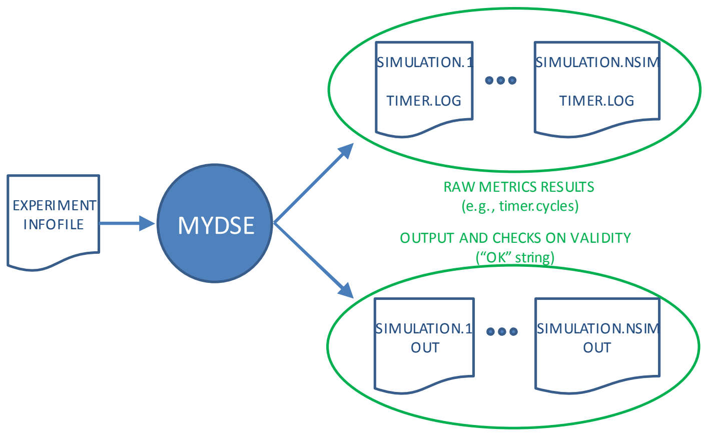


Figure 8: TOOLFLOW for the MYDSE tool. Several output files are generated from the basic tool, but they are re-ordered in such a way that can be easily retrieved and parsed for a mining of the results.

3.7 Experiment description: the INFOFILE

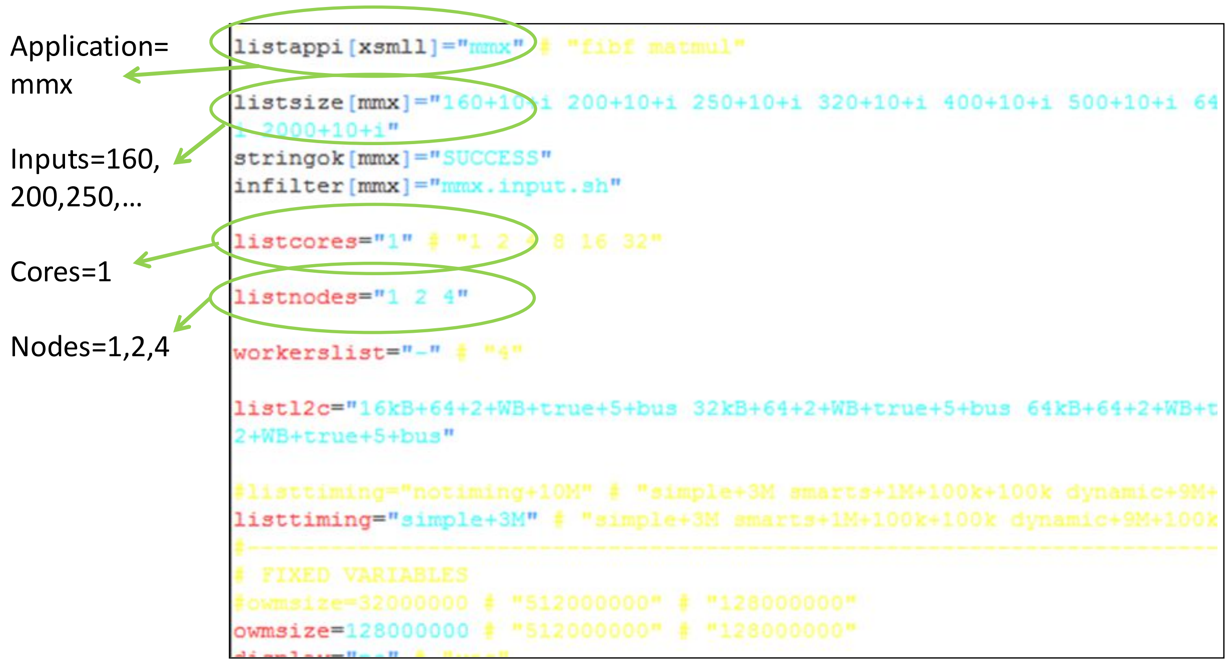
Here we show a sample of a configuration file for the Design Space Exploration. Our aim is to make the DSE easy to manage with a clear identification of the Design Space. To this end, we describe the experiment through a simple file that uses "Bash syntax" [51] : **`<variable>="<string>"`**. Each variable starting with the prefix "**`list`**" is used for defining a DSE variable, while **`<string>`** defines a set of values (space-separated values, i.e. **`<string>::=[<value>]['  '<value>]`***). For example:

```
listcores="1 2 4 8"
```

This means that during the DSE the variable 'cores' assumes the values cores=1, cores=2, cores=4, cores=8. Please also note that if the <value> represents multiple element separated by space (e.g., the command line arguments of the application, we use the character '+' to separate them (i.e., "160+10+i" represent matrix_size=160, block_size=10, element_type=integer).

All necessary context is generated and managed as requested by the "Correct Methodology" (CM) (configuration files, names, paths …) [38] [39] [40] as illustrated in D7.1 Section 6.

Figure 9 illustrates a sample INFOFILE for an experiment using the model XSMLL, the "Matrix Multiplication" application/benchmark ('mmx'), for various inputs and different number of nodes/SoCs.



```
listappi[xsmll]="mmx" # "fibf matmul"
listsize[mmx]="160+10+i 200+10+i 250+10+i 320+10+i 400+10+i 500+10+i 64
i 2000+10+i"
stringok[mmx]="SUCCESS"
infilter[mmx]="mmx.input.sh"

listcores="1" # "1 2 4 8 16 32"

listnodes="1 2 4"

workerslist="-" # "4"

listl2c="16kB+64+2+WB+true+5+bus 32kB+64+2+WB+true+5+bus 64kB+64+2+WB+t
2+WB+true+5+bus"

#listtiming="notiming+10M" # "simple+3M smarts+1M+100k+100k dynamic+9M+
listtiming="simple+3M" # "simple+3M smarts+1M+100k+100k dynamic+9M+100k
#--------------------------------------------------------------------
# FIXED VARIABLES
#owmsize=32000000 # "512000000" # "128000000"
owmsize=128000000 # "512000000" # "128000000"
```

Figure 9: Sample INFOFILE, which describes a Design Space Exploration experiment.

3.8 Defining an Architecture in the INFOFILE

One more possibility is to explore also different organizations for the architecture. This can serve not only to model different chips that are available in the commodity market and provided by different vendors, but also to test which features may matter mostly for a given application [52][53][54][55][56][57][58][59].

For example (see Figure 10), we can define the architecture under study by connecting the high-level architectural blocks as depicted in the picture with a **`<value>`** as defined in the previous section. In this case, the **`<value>`** describes the links between the architectural blocks, i.e. "**`l3-bus`**" is a link between the "**`L2 cache`**" and the "**`bus`**" (the '**`main`**' block is always present as the "father" of all blocks); the '**`+`**' character separates the different links. The '**`.`**' character separates a first part which represents a single instance of the implicitly defined architectural blocks and a second part which represents multiple instances of the implicitly defined architectural blocks (by default the number of such instances is equal to the specified cores in the design space point).

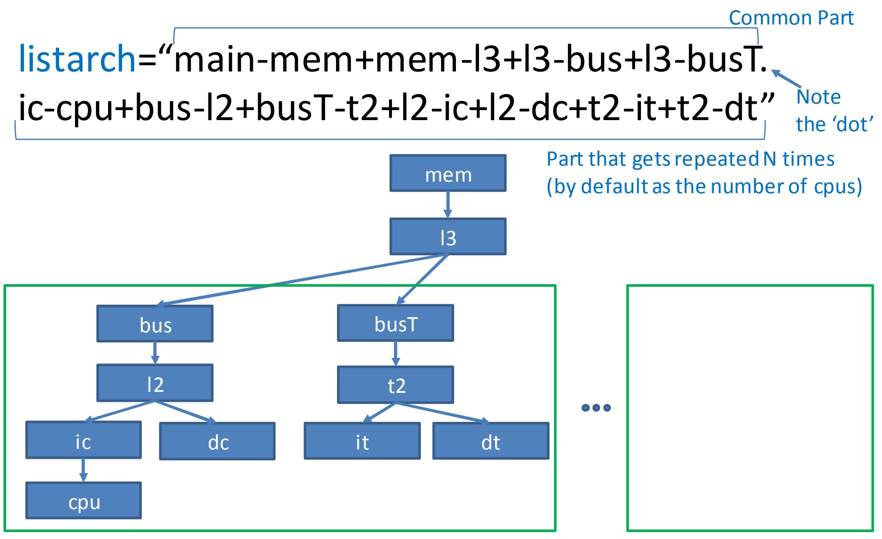

Common Part
listarch="main-mem+mem-l3+l3-bus+l3-busT.
ic-cpu+bus-l2+busT-t2+l2-ic+l2-dc+t2-it+t2-dt"
Note
the 'dot'
Part that gets repeated N times
(by default as the number of cpus)
mem
l3
bus
busT
l2
t2
ic
dc
it
dt
cpu
•••

Figure 10: Defining an architecture in the MYDSE tool. The architecture is then translated in statements for the COTSon framework.

This modeling syntax does not have the ambition to be a general description language but it is a simple and rapid prototyping way to explore most common combinations that we may want to explore.

Of course, like for the other list-type variables we can specific multiple values in the listarch variable so that the MYDSE tool generates automatically all COTSon framework input files that are necessary.

In a similar way, we can describe an FPGA block (under definition) as illustrated below (Figure 11):

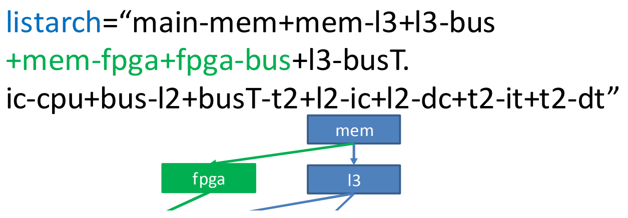

listarch="main-mem+mem-l3+l3-bus
+mem-fpga+fpga-bus+l3-busT.
ic-cpu+bus-l2+busT-t2+l2-ic+l2-dc+t2-it+t2-dt"
mem
fpga
l3

Figure 11: Introducing an FPGA block between memory and the bus in the 'listarch' parameter.

3.9 TRACING capabilities

Another feature that is available in the MYDSE tool, through the listarch variable, is the possibility to insert a "tracing probe". Similar to an actual probe, the user can insert it in the point where the tracing is needed. In the following example (Figure 12), a trace module is inserted between memory and the last level cache.

listarch="main-mem+mem-trace+trace-l3+l3-bus+l3-busT.ic-cpu+bus-l2+busT-t2+l2-ic+l2-dc+t2-it+t2-dt"

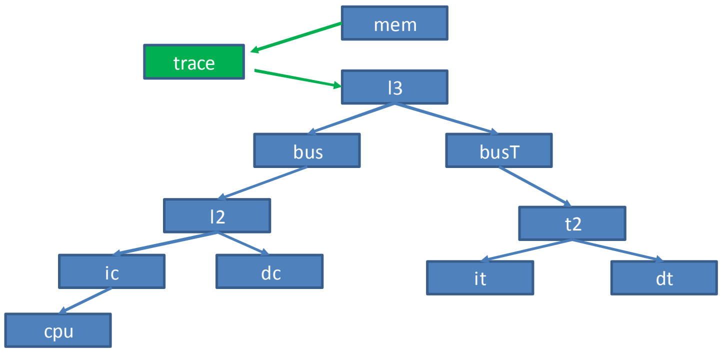
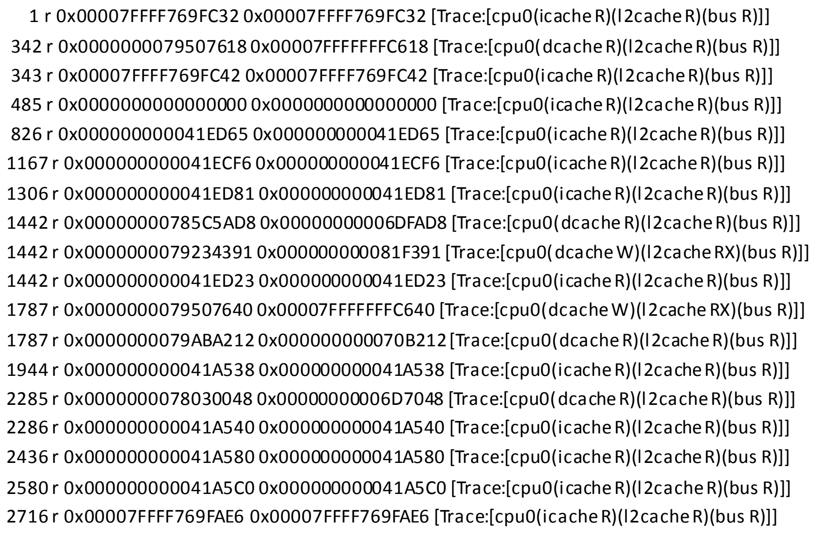


Figure 12: Introducing a software "tracing probe" between the memory and the L3 cache.

The tracing module in the INFOFILE variable listarch triggers the generation of the corresponding hooks in the underlying simulation framework (COTSon in this case) and the COTSon module has in turn a customizable output. For example, the trace of Figure 13 is generated by the following method in the Tracer class. Tracing is a relevant feature for performance evaluation [6].

```
MemState Tracer::read(const Access& m, uint64_t tstamp, Trace& mt, MOESI_state ms)
{
        stringstream s;
        s << format("%10llu r 0x%016X 0x%016X") % tstamp % m.phys % m.virt;
        s << " [" << mt << "]\n";
        gz.as_text(s.str());
        return next->read(m,tstamp,mt,ms);
}
```



```
   1 r 0x00007FFFF769FC32 0x00007FFFF769FC32 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
 342 r 0x0000000079507618 0x00007FFFFFFFC618 [Trace:[cpu0(dcache R)(l2cache R)(bus R)]]
 343 r 0x00007FFFF769FC42 0x00007FFFF769FC42 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
 485 r 0x0000000000000000 0x0000000000000000 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
 826 r 0x000000000041ED65 0x000000000041ED65 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
1167 r 0x000000000041ECF6 0x000000000041ECF6 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
1306 r 0x000000000041ED81 0x000000000041ED81 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
1442 r 0x00000000785C5AD8 0x00000000006DFAD8 [Trace:[cpu0(dcache R)(l2cache R)(bus R)]]
1442 r 0x0000000079234391 0x000000000081F391 [Trace:[cpu0(dcache W)(l2cache RX)(bus R)]]
1442 r 0x000000000041ED23 0x000000000041ED23 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
1787 r 0x0000000079507640 0x00007FFFFFFFC640 [Trace:[cpu0(dcache W)(l2cache RX)(bus R)]]
1787 r 0x0000000079ABA212 0x000000000070B212 [Trace:[cpu0(dcache R)(l2cache R)(bus R)]]
1944 r 0x000000000041A538 0x000000000041A538 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
2285 r 0x0000000078030048 0x00000000006D7048 [Trace:[cpu0(dcache R)(l2cache R)(bus R)]]
2286 r 0x000000000041A540 0x000000000041A540 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
2436 r 0x000000000041A580 0x000000000041A580 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
2580 r 0x000000000041A5C0 0x000000000041A5C0 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
2716 r 0x00007FFFF769FAE6 0x00007FFFF769FAE6 [Trace:[cpu0(icache R)(l2cache R)(bus R)]]
```

Figure 13: Sample output that is produced by the tracer module. The trace represents in each row: the timestamp of the operation, the type of the bus-transaction (r=read), the physical and virtual address, which core has issued the operation if it is instruction-fetch or data-read or data-write, the operation on the l2 cache and on the bus (again).

3.10 Collecting the data of the experiments: the GTCOLLECT tool

We have evolved the previous ESTRAI tool which was initially limited (cf. Deliverable D7.1) to a few DSE parameters so that it can now exactly match the same design point that are specified in the INFOFILE (description of the Experiment cf. 3.7). The tool has been renamed GTCOLLECT because it is also parsing a so-called LAYOUTFILE to identify the relevant output metrics that the user wishes to collect after the experiment (GT stand for Graphic Table – see the simple format in Figure 14).

One important benefit of decoupling this phase from the MYDSE phase is that the user can now run only once (or at least 5 times to have the classical "statistically stable" number of measurements). However, after the experiment has been performed, the raw metrics remain stored in the output files and the user can mine the produced data in order to calculate various formulas based on the raw metrics.

For example, we take the maximum value across a set of cores and a set of nodes for the executed cycles, but we need to accumulate the value of the executed instruction to understand how much work or energy had been consumed by the system. In other cases we need to calculate the miss rate based on the total number of read and write operations that hit the cache and the total number of reference (all those values are produced separately and for each core and for each level of the caches).

Here is an example of how we define the instruction latency that is seen by the processor:

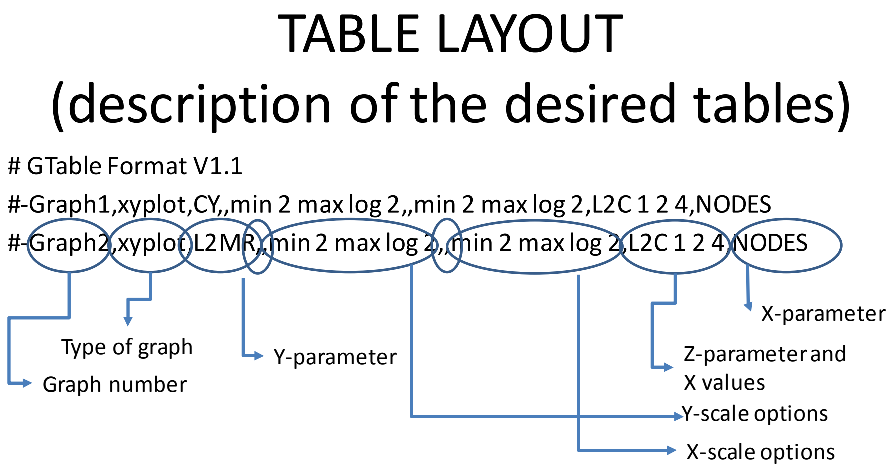
```
((tmem*l3mr+th3)*l2mr+th2)*icmr+tih1,ilat,a,f
```

In this case we define the instruction latency seen by the processor (ilat) as the average 'a' of floating point values ('f') and the formula that is used add the hit-time of L1 cache (tih1) and adds the latency of the rest ((tmem*l3mr+th3)*l2mr+th2) multiplied by the miss probability for instructions (icmr).

In this way, the user can exactly run MYDSE and GTCOLLECT tools:

```
$ mydse.sh       T305-multihd

$ gtcollect.sh   T305-multihd
```



```
# GTable Format V1.1
#-Graph1,xyplot,CY,,min 2 max log 2,,min 2 max log 2,L2C 1 2 4,NODES
#-Graph2,xyplot,L2MR,,min 2 max log 2,,min 2 max log 2,L2C 1 2 4,NODES
```

Figure 14: GTable 1.1 format. The X-scale and Y-scale options mean the following: 'min' and 'max' are automatically determined as the minimum and maximum value among the data respectively; 'log 2' means that the scale is of logarithmic type in base two.

3.11 Massive Experiment Test

As a test of the possibilities of the MYDSE tool we launched a massive experiment (ideal month was August, when any machine is less loaded) that used 12 machines for parallelizing the simulation points. The experiment used the following INFOFILE:

```
listsize[mmx]="160+10+i   200+10+i   250+10+i   320+10+i   400+10+i   500+10+i
640+10+i 800+10+i 1000+10+i 1280+10+i 1600+10+i"
listcpu="timer0 timer1 timerdep bandwidth+2+128 random"
listcores="1 2 4"
listnodes="1 2 4 8"
listhd="karmic64 trusty-axmv3 tfxv4"
listl2c="16kB 32kB 64kB 128kB 256kB"
listtiming="simple+3M smarts+1M+100k+100k dynamic+9M+100k+100k+90+120"
```

As can be easily calculated, although the INFOFILE is simple, the number of simulation points is quite high:

11x5x3x4x3x5x3 = **29700** simulation points!

The experiment was quite successfully performed (we did not expect all simulation point to successfully complete) in a couple of weeks in a fully automated way and was and interesting proof of concept of the capabilities of the MYDSE tool.

3.12 Preparing the dependent packages: the BOOTSTRAP, CONFIGURE, ADD-IMAGE tools

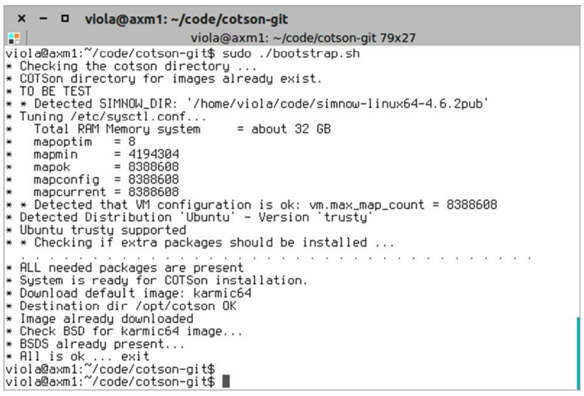
The BOOTSTRAP tool serves for taking care of the dependencies that are needed by the various packages (missing packages needed by the various tools). Again, although the user can go through the several documentation files and try to install package by package we realized this as a time-consuming activity also sometimes system administrators get confused on how to setup properly our host machines. Moreover, the tools may need special tuning of some kernel parameters such as the number of host memory pages or disabling the address space randomization.

This information is methodically encapsulated in several scripts so that we have a solid reference for system administrators and we save time whenever we add one more simulation host or a new dedicated workstation to our cluster of simulation computers.

The BOOTSTRAP tool extends the COTSon framework and it has been published openly (MIT license) in the public repository of COTSon on the SourceForge repository (one of the main Open-Source repositories). The BOOTSTRAP tool is typically run once per machine by system administrators.

The CONFIGURE tool (Figure 16) is a new variant of the same tool originally available in the COTSon framework. The motivation of this modification is to enable multiple users on the same host to run a configuration of their own simulation setup without the need of system administration intervention. The tool can be run multiple times to configure or re-configure the development environment for any system model under development, totally in user-space (i.e., without the need of root-access system changes).

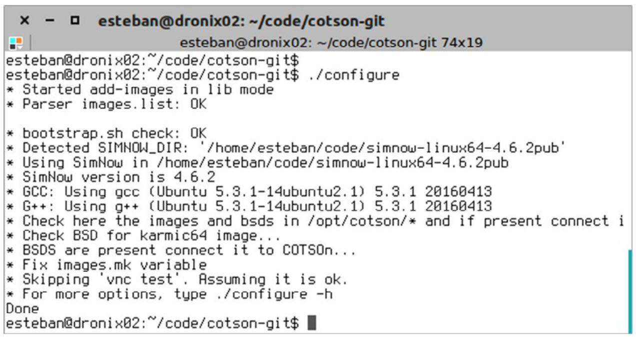
Command: ./bootstrap.sh (as root)



```
viola@axm1: ~/code/cotson-git
viola@axm1: ~/code/cotson-git 79x27
viola@axm1:~/code/cotson-git$ sudo ./bootstrap.sh
* Checking the cotson directory ...
* COTSon directory for images already exist.
* TO BE TEST
* * Detected SIMNOW_DIR: '/home/viola/code/simnow-linux64-4.6.2pub'
* Tuning /etc/sysctl.conf...
*   Total RAM Memory system     = about 32 GB
*   mapoptim   = 8
*   mapmin     = 4194304
*   mapok      = 8388608
*   mapconfig  = 8388608
*   mapcurrent = 8388608
* * Detected that VM configuration is ok: vm.max_map_count = 8388608
* Detected Distribution 'Ubuntu' - Version 'trusty'
* Ubuntu trusty supported
* * Checking if extra packages should be installed ...
  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
* ALL needed packages are present
* System is ready for COTSon installation.
* Download default image: karmic64
* Destination dir /opt/cotson OK
* Image already downloaded
* Check BSD for karmic64 image...
* BSDS already present...
* All is ok ... exit
viola@axm1:~/code/cotson-git$
viola@axm1:~/code/cotson-git$ █
```

Figure 15: Sample run of the BOOTSTRAP tool. The tool is typically run once when we add new machine to our DSE cluster and checks and installs the missing packages and fine tunes host kernel parameters and other necessary system settings.

Command: ./configure



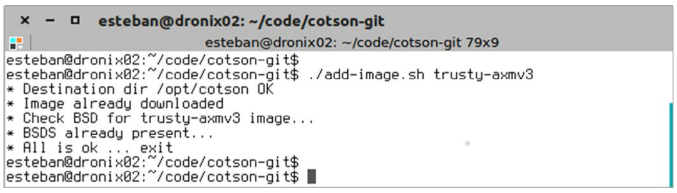
```
esteban@dronix02: ~/code/cotson-git
esteban@dronix02: ~/code/cotson-git 74x19
esteban@dronix02:~/code/cotson-git$
esteban@dronix02:~/code/cotson-git$ ./configure
* Started add-images in lib mode
* Parser images.list: OK

* bootstrap.sh check: OK
* Detected SIMNOW_DIR: '/home/esteban/code/simnow-linux64-4.6.2pub'
* Using SimNow in /home/esteban/code/simnow-linux64-4.6.2pub
* SimNow version is 4.6.2
* GCC: Using gcc (Ubuntu 5.3.1-14ubuntu2.1) 5.3.1 20160413
* G++: Using g++ (Ubuntu 5.3.1-14ubuntu2.1) 5.3.1 20160413
* Check here the images and bsds in /opt/cotson/* and if present connect i
* Check BSD for karmic64 image...
* BSDS are present connect it to COTSOn...
* Fix images.mk variable
* Skipping 'vnc test'. Assuming it is ok.
* For more options, type ./configure -h
Done
esteban@dronix02:~/code/cotson-git$ █
```

Figure 16: Sample run of the CONFIGURE tool. The tool permits the local per-user configuration of development software, i.e. any component model that is defined within the COTSon framework.

Finally, we describe briefly here another tool that enables a selective inclusion of a given drive-image and related BSDs files. This tool (Figure 17) is propaedeutic to the INST-IMGPKG tool and it is published currently as part of the COTSon framework. Additionally this tool can perform the download of the drive images: some of them as now provided on the download public portion of the AXIOM website http://download.axiom-project.eu .

Command: ./add-image.sh trusty-axmv3



```
esteban@dronix02: ~/code/cotson-git
esteban@dronix02: ~/code/cotson-git 79x9
esteban@dronix02:~/code/cotson-git$
esteban@dronix02:~/code/cotson-git$ ./add-image.sh trusty-axmv3
* Destination dir /opt/cotson OK
* Image already downloaded
* Check BSD for trusty-axmv3 image...
* BSDS already present...
* All is ok ... exit
esteban@dronix02:~/code/cotson-git$
esteban@dronix02:~/code/cotson-git$ █
```

Figure 17: Sample run of the ADD-IMAGE tool that shows the various operations performed by the tool.

4 Power Estimation

4.1 Power Estimation in the simulation tools

Xilinx provides a tool called "Xilinx Power Estimator (XPE)" this is a tool based on Spreadsheets to roughly estimate the power consumption [60]. For a power-estimation related to the real application that runs on the system it is necessary to provide on-the-field measurements or use other power estimation tools. At the current stage in the project, we considered such options but we preferred to use another methodology, which is provided by our simulation tools based on the HPLabs McPAT tool [20]. Further details on power estimation with XPE are in Table 1 of D6.2,

The McPAT tool is widely used in research and in the literature [20][27][44][45][46][47]. The McPAT tool serves for power and area estimation (http://www.hpl.hp.com/research/mcpat/). McPAT can model power, area, and timing simultaneously. It also supports processor configurations ranging from 90nm to 22nm and beyond. McPAT includes models for the components of a complete chip multiprocessor, in-order and out-of-order processor cores, and networks-on-chip, shared caches, and integrated memory controllers. It also models timing, area, and dynamic, short-circuit, and leakage power for each of the device types forecast in the ITRS roadmap including bulk CMOS, SOI, and double-gate transistors.

McPAT needs and XML input that results from the activity on the application. The MYDSE tool generates the experiment results through an additional feature called 'heartbeat'. For every heartbeat, a dump of the full simulation statistics is produced. At this point, another tool called COTSON2MCPAT (a Perl script) which provides the translation into data that it is readable by the McPAT tool (XML format). The tool flow is represented in Figure 18.

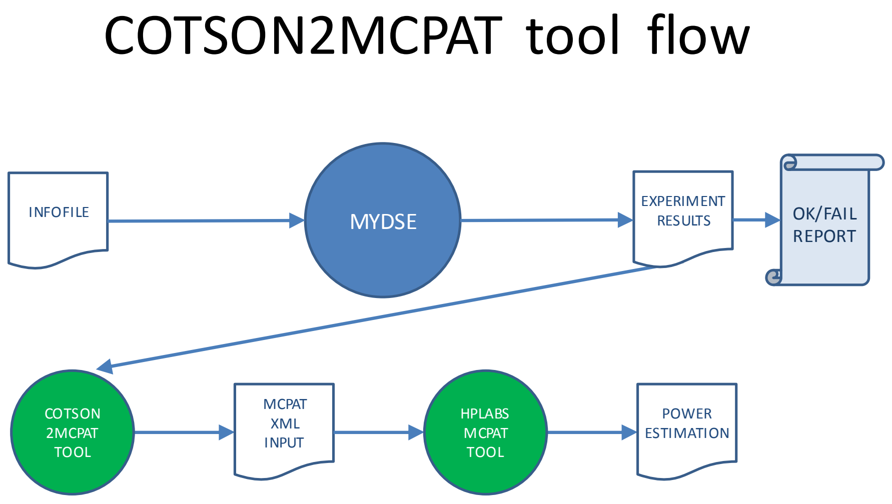


Figure 18: The COTSON2MCPAT and MCPAT tool workflow integrated in the MYDSE (the MCPAT is launched offline).

The COTSON2MCPAT tool generates a McPAT input containing the relevant event information (instruction counts, memory accesses, etc.) for the power analysis tools, so that such tool can produce an estimate of the power consumption.

Code and examples are available at http://cotson.svn.sourceforge.net/viewvc/cotson/trunk/src/mcpat/

This tool can always serve as a reference basis to discard some of the Design Space points that may generate a too large power consumption without the need to buy components or build a system.

4.2 Power Estimation of commodity boards that were available by July 2016

As of July 2016, the first Xilinx ZU9EG were available on the market by some system integrator, therefore we considered to start using them in order to become more familiar with such chips, the related Development Kits, initiate to test early designs. We have received the Xilinx ZU9EG on a module (see Figure 19) and only the test carrier boards that are shown in Figure 19, which did not include easy flashing capabilities. We needed two boards in order to have a basic environment to test the communication between two boards, as required by our project goals. We needed to buy an additional bench power supply (shown in the same figure) in order to provide a regulated power supply (RIGOL DP711), necessary to power up such boards.

Moreover, we made sure that any partner could have used those boards for initial testing through remote connection to one of our server and eventual assistance from one of our skilled lab technicians.

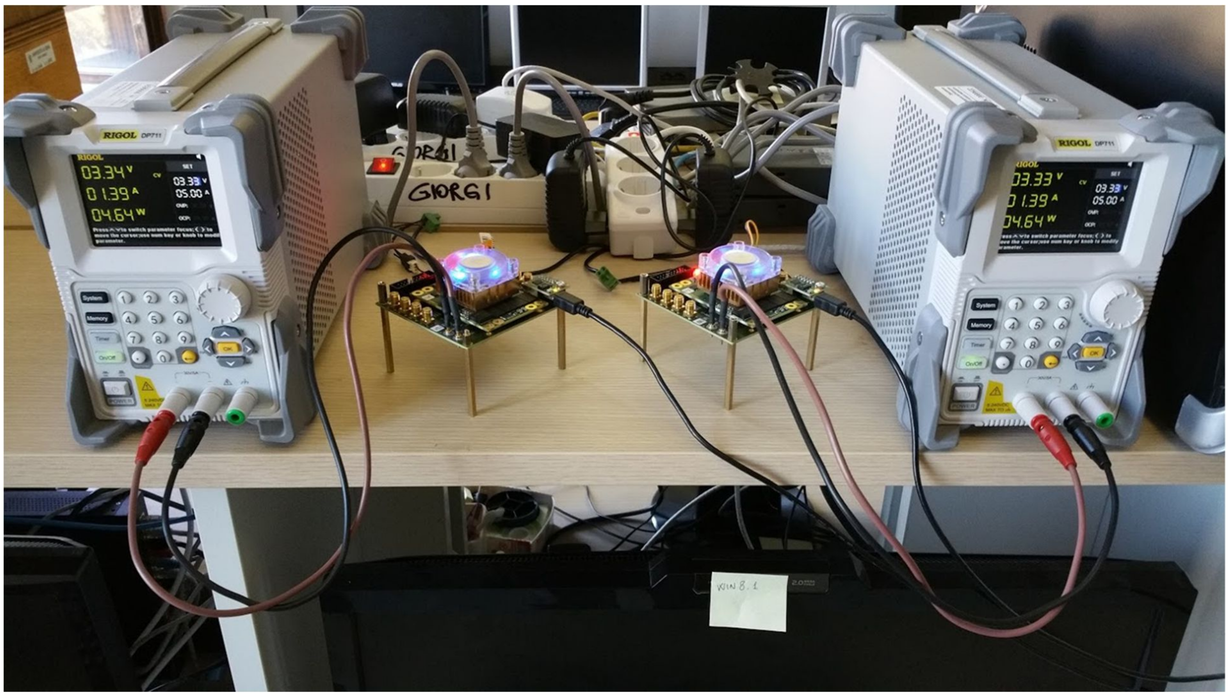


Figure 19: Two Xilinx ZU9EGmounted on a test carrier board (without power regulation).

We also considered that in case of risk in the production of our boards, these additional boards could have provided a good backup. Moreover, we can eventually try to test different conditions to be compared to our board that are planned for initial release in early February 2017 (as planned for Task T6.6 in the next reporting period).

4.3 Power Estimation on the AXIOM board

In Figure 20 we show the facility that had been included for the power measurements in the actual AXIOM-board (cf. D6.2 Figure 4). The example shows the Zynq SoC but it is also possible to measure the power consumption of other components such us the DRAM (cf. D6.2).

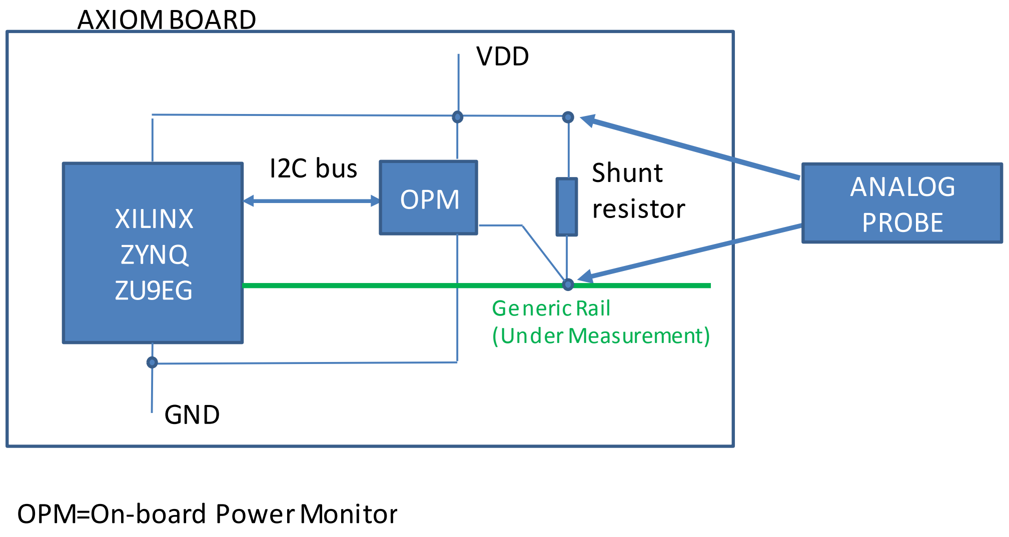


Figure 20: Power measurement facility on the AXIOM board (cf. deliverable D6.2). The Power can be read using a power monitor and an analog probe.

This will permit us to have a more precise reading for the power consumption in actual use case and it is planned for a later stage in this project, depending on the actual needs.

5 Test case experiments: beyond benchmarks – studying the impact of the Operating System

From WP3 we got indication about some important benchmarks derived from the two main use cases (SVS – Smart Video Surveillance and SHL – Smart Home and Living – cf. D3.1, D3.2). We report here the data relevant for the reference experiments regarding the benchmarks selected in WP3.

5.1 Benchmarks for the use case SVS (Smart Video Surveillance)

Table 1 reports for each benchmark a brief description of the purpose, what kind of inputs are expected, the actual values for the inputs that are used in the reference cases, the reference output that is expected and the dissemination level (public or private).

Table 1: Selected Smart Video Surveillance benchmarks selected.

BENCH MARK	PURPOSE	INPUTS	REFERENCE INPUTS	REFERENCE OUTPUT	DISSEMINATION LEVEL
mm	Matrix Multiply benchmark (C=AxB) used for Convolutional Neural Networks estimation	M=int, N=int, K=int; A will be M*K, B will be K*N	M=64, N=25, K=1944; M=64, N=1600, K=322; M=32, N=25, K=2128; M=64, N=800, K=360	The application must return 0, otherwise an arithmetic error occurred	public
ffmpeg	H264 decoder benchmark	INMP4=H264 input file, OUTRAW=output raw video file, OUTMP3=output audio file	INMP4=input.mp4 OUTRAW=output.raw OUTMP3=output.mp3	md5sum $OUTRAW must return 77b12eda4ce87513a04a023fb93c73fd	public
face-detection	LBP Face Detection Kernel	INIMG=input raw image file, W=$INIMG image width, H=$INIMG image width, OUT=output faces file	INIMG=testImage.raw, W=1920, H=1080, OUT=testImage.output	md5sum $OUT must return 596aaf1ffb2351c8ad7999b11e49099a	private
yuv2rgb	YUV to RGB color conversion	INIMG=test-yuv420p.yuv, W=1920, H=1080, OUT=test-output.ppm	INIMG=test-yuv420p.yuv, W=1920, H=1080, OUT=test-output.ppm	md5sum $OUT must return ace5e75271ae3c523983391c7ac7eeb9	public

5.2 Benchmarks for the use case SHL (Smart Home & Living)

Table 3 reports for each benchmark a brief description of the purpose, what kind of inputs are expected, the actual values for the inputs that are used in the reference cases, the reference output that is expected and the dissemination level (public or private).

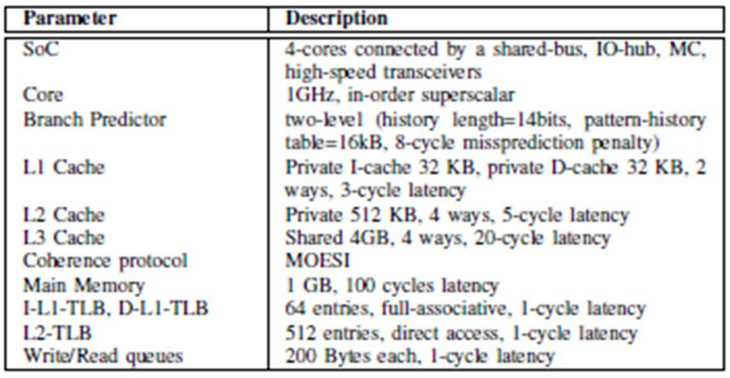
Table 2: Selected Smart Home and Living benchmarks selected.

BENCH MARK	PURPOSE	INPUTS	REFERENCE INPUTS	REFERENCE OUTPUT	DISSEMINATION LEVEL
sfbcep	Feature extraction	30 sec audio file, format PCM, bit depth 16bits, sample rate 16kHz	gstsfbcep_30s.pcm	#ref32bit: gstsfbcep_30s.ref.ARM7.prm #ref64bit: gstsfbcep_30s.ref.x86_64.prm	private
Anisotropic Smoothing	image processing: smoothing filter	INIMG=input bmp image file, W=$INIMG image width, H=$INIMG image width, OUT=output file	INIMG= input_260x260.bmp, W=260, H=260	OUT= output_260x260.ref.bmp	private
Iris Recognition Module	Iris recognition algorithm	INIMG=input jpg image file, W=$INIMG image width, H=$INIMG image width, MODELS=16 iris_code	INIMG= Gray_Image.jpg, W=1920, H=1080	txt results OUT= matching.ref.txt	private

While it is certainly interesting to analyze systematically those benchmarks for the moment in this report, we present some detailed results for one of those benchmarks: Matrix Multiplication (‘mm’). The ported version on the XSMLL execution model that has been developed in Task 5.2 (cf. D5.2) is called ‘mmx’.

In Table 3, we report the key parameters of the modeled cores. The system is implementing the XSMLL execution model [7] [30] [31] [32] [33] [34] [35] that has been developed in Task 5.2 (cf. deliverable D5.2). In the WP4 the compilation toolchain to compile from OmpSs [2] [3] [8] [26] to the XSMLL have been explored (cf. Deliverable D4.2) by partner BSC. We also documented some other efforts to test the possibilities of connecting directly the GASnet [1] substrate (which in turn relies on OpenMPI [4]) to XSMLL in D4.1 and D5.2

Table 3: Architectural parameters of the multicore.



Parameter	Description
SoC	4-cores connected by a shared-bus, IO-hub, MC, high-speed transceivers
Core	1GHz, in-order superscalar
Branch Predictor	two-level (history length=14bits, pattern-history table=16kB, 8-cycle missprediction penalty)
L1 Cache	Private I-cache 32 KB, private D-cache 32 KB, 2 ways, 3-cycle latency
L2 Cache	Private 512 KB, 4 ways, 5-cycle latency
L3 Cache	Shared 4GB, 4 ways, 20-cycle latency
Coherence protocol	MOESI
Main Memory	1 GB, 100 cycles latency
I-L1-TLB, D-L1-TLB	64 entries, full-associative, 1-cycle latency
L2-TLB	512 entries, direct access, 1-cycle latency
Write/Read queues	200 Bytes each, 1-cycle latency

5.3 Experiments highlighting the impact of different OS distributions and of the OS itself

Some experiments have been performed on the COTSon/XSMLL (cf. D5.2) with the following parameters.

Experiment T313:

- Square Matrix size n=512 and block size b=8
- Cores=1
- Different number of Nodes=1,2,4
- Different Linux Distributions:
  - Ubuntu 9.10 (karmic64)
  - Ubuntu 10.10 (tfxv4)
  - Ubuntu 14.04 (trusty-axmv3)
  - Ubuntu 16.04 (xenv0)
- Different L2 cache sizes: 2KiB, 4KiB, 8KiB, 16KiB, 32KiB

As we can see from Figure 21, there is a large variation from between different configuration: in particular, for one node there is a 60% variation from the slowest to the fastest configuration. From the bars on the left in the same figure, we can identify that the best OS distribution appears to be the trusty-axmv3 and the xenv0. This in turn may depend from several different factor including the set of installed packages. We are currently investigating more such complex OS setups with the help of automated tools illustrated in the previous sections. In all cases, a larger cache size is improving the performance, as expected. Once we increase the number of nodes, one interesting effect is also that the total capacity of caches is increasing and thanks do the Dataflow based execution model and the fact that coherency is not needed, we observe a reasonable scaling of performance with the number of nodes.

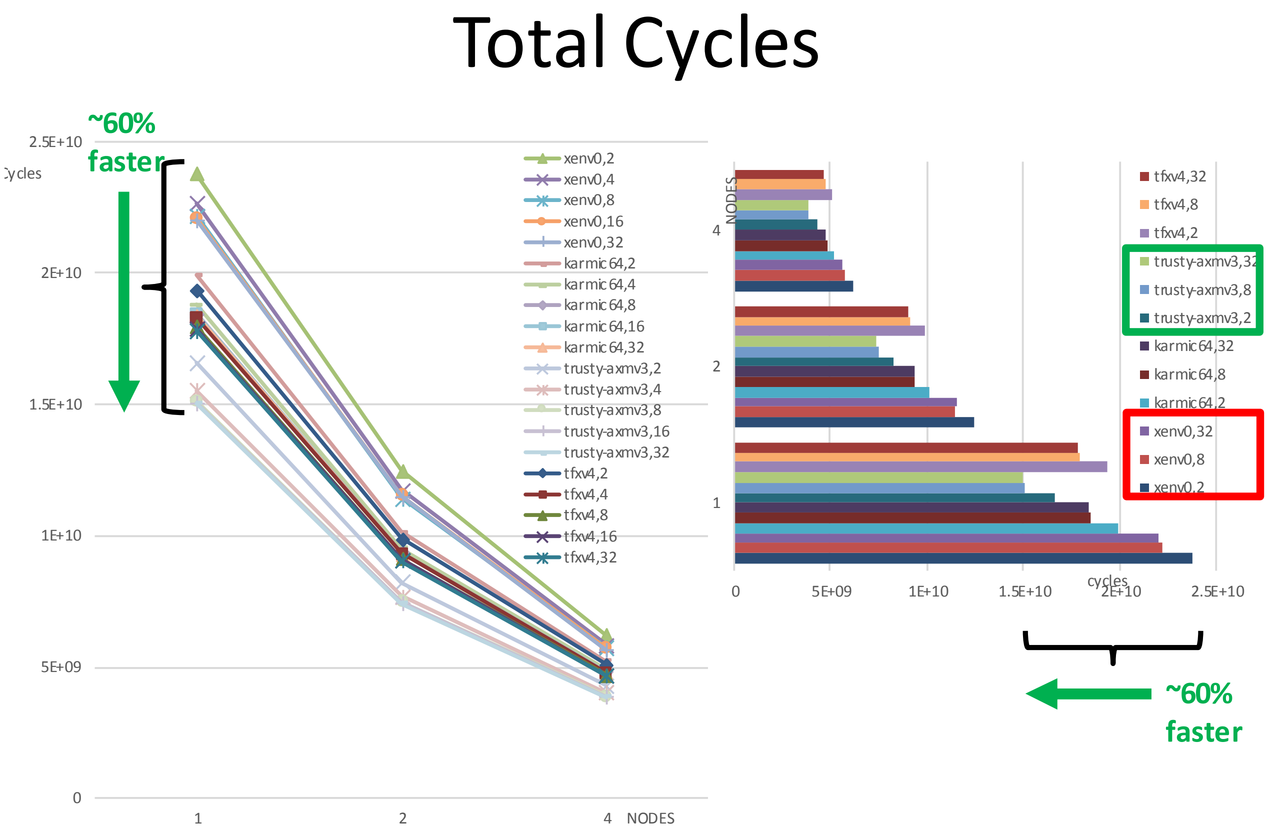


Figure 21: Experiment T313 showing a large variation in performance (cycles) as we vary the cache size and the operating system distribution. In the case of one node, this variation is up to 60%.

In the following experiment T311, we keep fixed the L2 cache size and again we vary the OS distribution while observing the effect of different input size and therefore a different number of generated threads. In the XSMLL model, as we increase the input size we generally observe an increase in the number of available threads and therefore more potential for parallelism.

Experiment T311:

- Square Matrix size n and block size b: n=128,256,512,1024 b=8
- Cores=1
- Nodes=1,2,4
- Different Linux Distributions:
    - Ubuntu 9.10 (karmic64)
    - Ubuntu 10.10 (tfxv4)
    - Ubuntu 14.04 (trusty-axmv3)
    - Ubuntu 16.04 (xenv0)
- L2 cache sizes: 256KiB

As we can see from Figure 22, we have a certain variation in the performance with the OS distribution and trusty-axmv3 seems the best among the tested ones in most cases. The scaling with the input size and with the number of nodes seems also quite good.

Multi-HD Experiment (2)

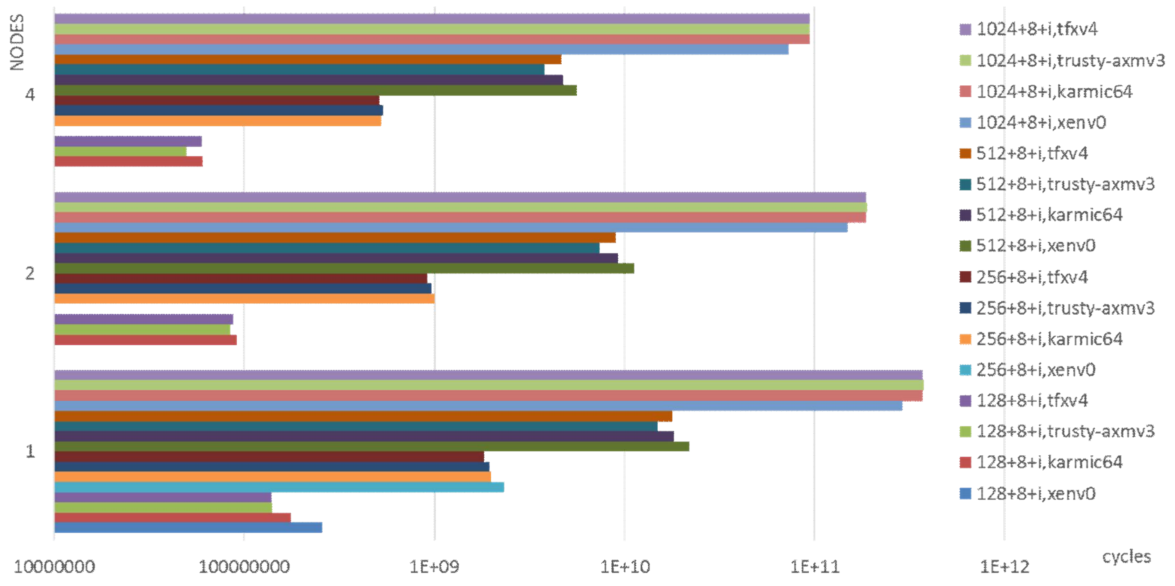


Figure 22: Experiment T311 - some variation in performance (cycles) in the different OS distribution as we vary the input size (matrix size=128, 256, 512, and 1024); the trusty-axmv3 seems to have one of the best performance compared to other distributions.

In the next experiment T312, we keep fixed the input size to have enough thread level parallelism; we choose the value of n-512 and again we vary the OS distribution while observing the effect of different L2 cache size but this time we use larger sizes (from 32 to 1024KiB).

Experiment T312:

- Square Matrix size n=64,256,1024 and block size b=8
- Cores=1
- Nodes=1,2,4
- Different Linux Distributions:
  - Ubuntu 9.10 (karmic64)
  - Ubuntu 10.10 (tfxv4)
  - Ubuntu 14.04 (trusty-axmv3)
  - Ubuntu 16.04 (xenv0)

L2 cache sizes: 32KiB, 64KiB, 128KiB, 256KiB, 512KiB, 1024KiB

As we can see from Figure 23 and Figure 24, there is a strong correlation between the data access latency seen by the processor (Figure 23). The miss rate of the L2 cache (Figure 24), meaning that is very likely that the L2 cache size has a strong influence in the performance and therefore it may play a crucial role in the system. For example, optimization techniques or other acceleration features should take care in particular to the L2 cache.

Data Access Latency

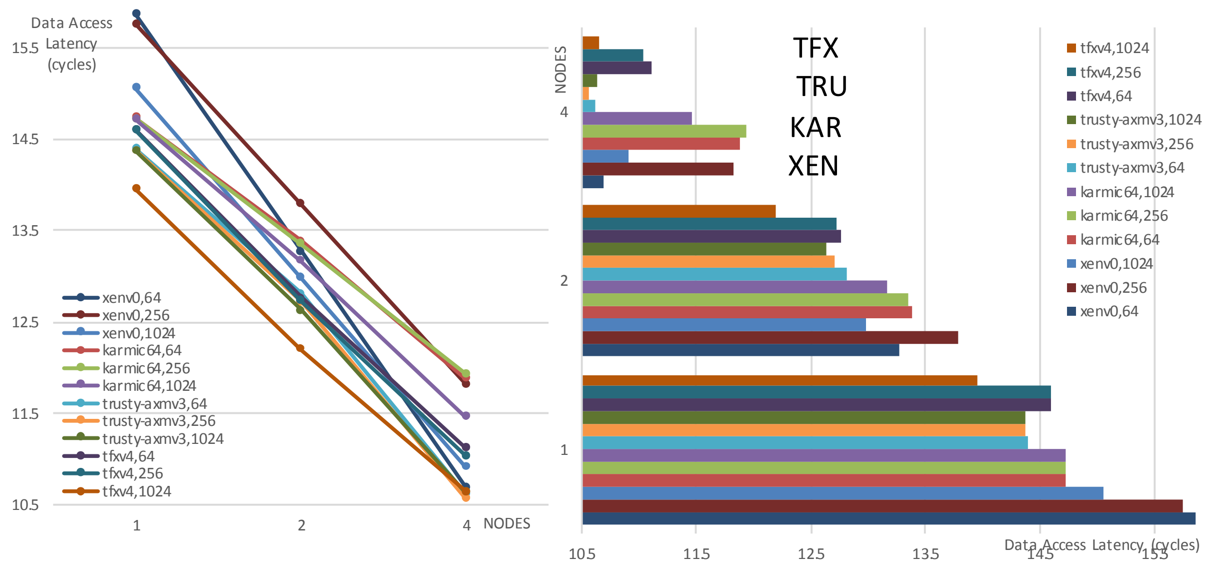


Figure 23: Experiment T312 - varying the input size, the OS distribution and the number of nodes, we observe the data access latency seen by the processor.

L2 Cache Miss Rate

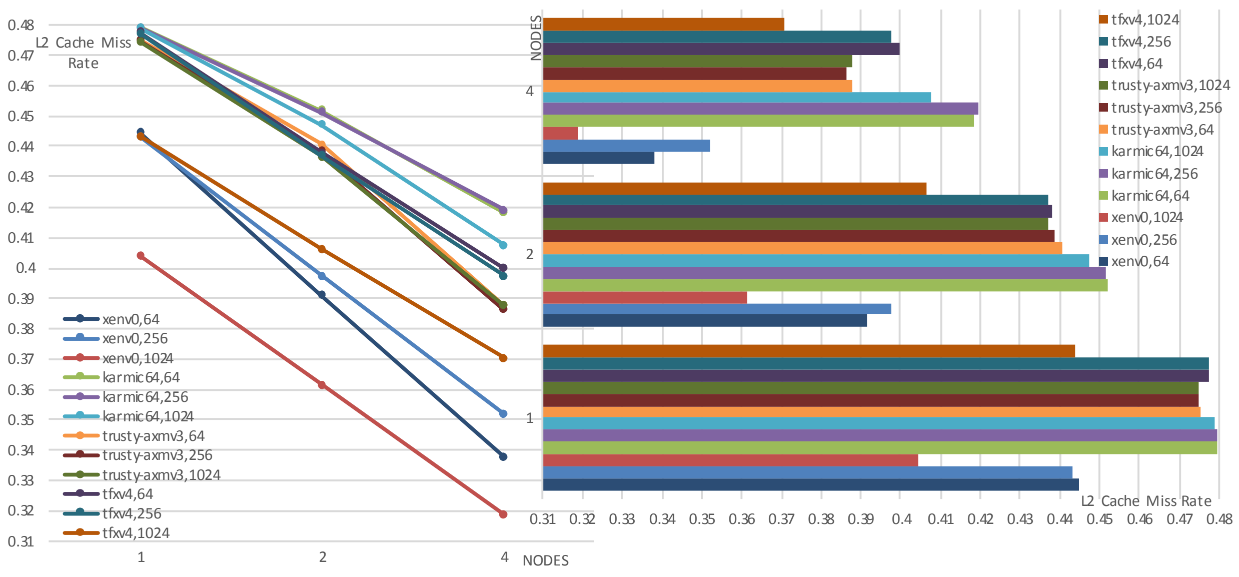


Figure 24: Experiment T312 - varying the input size, the OS distribution and the number of nodes, we observe the L2 cache miss rate: the L2 miss rate seems strongly correlated to the total data access latency.

Finally we observed the influence of the kernel execution time on the total cycles of the application for the experiments T312 (Figure 25) and T313 (Figure 26). In the first case we observe that the kernel impact on the total cycles may range from few percent (1 node) to more than 10% in the case of 4 nodes). In both cases, the influence of the kernel is more important for multi-node configurations. So special care has to be considered in designing the communication substrate between our AXIOM boards. Thanks to the DSE tools, it has been possible to detect this Design requirement more early in the project and before than other software and hardware components has bene released.

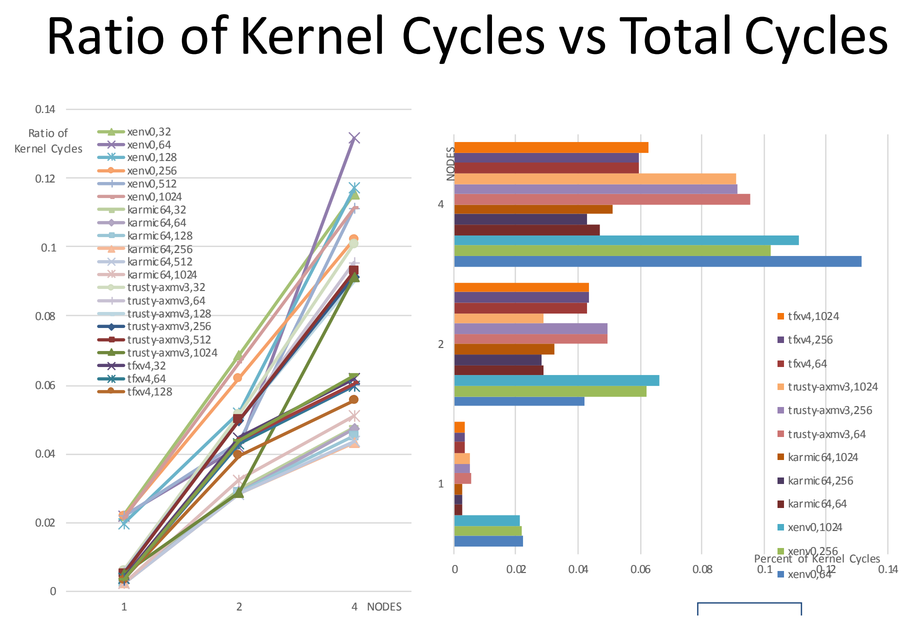


Figure 25: Experiment T312 the ratio of kernel cycles on total cycles. The kernel time may occupy from few percent to more than 13% of the total time in this experiment.

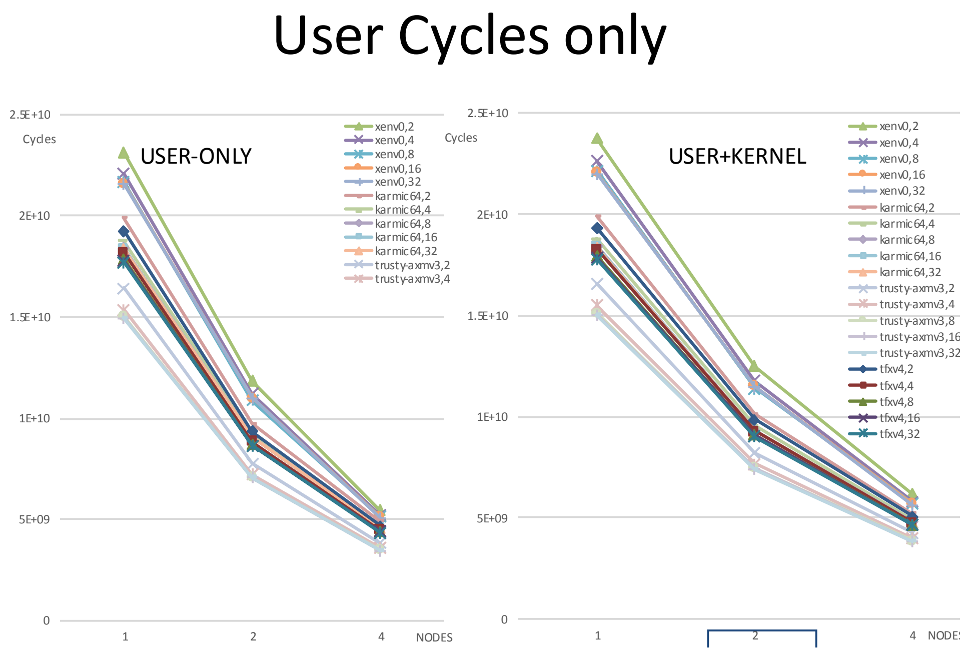


Figure 26: Experiment T313 showing a comparison of the total time and the user-only portion. The user time- can be sometimes misleading especially in configurations with more nodes.

6 Confirmation of DoA objectives and Conclusions

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

PLANNED	**DELIVERED**
DELIVERABLE:	
• Design Space Exploration (DSE) on the prototype for the AXIOM	Report

We have presented the evaluation of tools for the Design Space Exploration in AXIOM including recent experiments to explore factors that influence the performance of the benchmarks such as the matrix multiplication benchmark, which is a key kernel in many relevant applications.

We have shown in particular how those tools permitted to derive early in the Design process useful information to focus the design efforts toward optimizing L2 cache usage and the interconnects.

Therefore the objectives of the second year have been meet or exceeded (e.g. with the experiment involving the impact of the Operating System and of several popular Linux distributions).

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 on , vol., no., 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. "Scientific data: open access to research results will boost Europe's innovation capacity." http://europa.eu/rapid/press-release IP-12-790 en.htm.
37. "EU Open Science and Open Access Policies." http://ec.europa.eu/ research/swafs/index.cfm?pg=policy&lib=science
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. DOXYGEN website: http://www.stack.nl/~dimitri/doxygen/ (accessed January 2017).