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

PROJECT NUMBER: 645496

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

D5.1 – Operating system and documentation

Due date of deliverable: 31st January 2017
Actual Submission: 14th February 2017 (agreed extended date)

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

Lead contractor for the deliverable: EVI

Revision: See file name in document footer.

Project co-founded by the European Commission within the HORIZON FRAMEWORK PROGRAMME (2020)	
Dissemination Level: ~~CO~~ (promoted to) 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

Ver.#	**Date**	**Author**	**Organiz.**	**Change History**
0.1	25.01.2017	Paolo Gai, Davide Catani	EVI/SECO	Initial version from the Google doc
0.2	27.01.2017	Xavier Martorell	BSC	First Revision
0.3	27.01.2017	Paolo Gai, Davide Catani	EVI/SECO	Updated typos and text
0.4	29.01.2017	Paolo Gai	EVI	Typos
0.5	04.02.2017	Paolo Gai	EVI	Final release for last review.
0.6	25.10.2017	Paolo Gai	EVI	Integration of SECO and EVI updates
0.7	03.11.2017	Davide Catani	SECO	Updated descriptions
0.8	10.11.2017	Stefano Garzarella	EVI	Added appendix with text manuals
0.9	24.11.2017	Davide Catani	SECO	Updated section 4
0.91	10.12.2017	Davide Catani	SECO	Updated section 4.3 adding detail bout Arduino software support
0.92	23/12/2017	Davide Catani	SECO	Updated table 3 and Figure3
1.0	13/2/2018	Davide Catani, Paolo Gai	SECO, EVI	Integrated comments of the reviewers

Release Approval

Name	**Role**	**Date**
Paolo Gai	WP Leader	14.02.2018
Roberto Giorgi	Project Coordinator for formal deliverable	14.02.2018

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

Davide Catani, Davide Cardillo
R&D Department
SECO
Paolo Gai, Stefano Garzarella
R&D Department
Evidence– AXIOM

TABLE OF CONTENTS

TABLE OF FIGURES

GLOSSARY

aarch64 – ARM 64-bit instruct set
ALU – Arithmetic Logic Unit
API – Application Programming Interface
ARM – Instruction set architecture developed by ARM Holdings Ltd.
AVR8 – a RISC microcontroller developed by Atmel, implemented as open source IP for FPGAs
AXI – Advanced Extensible Interface
BRAM – Block RAM
BSP – Board Support Package
CPU – Central Processing Unit
CSI – Camera Serial Interface
DoA - Description of Action (acronym set by the European Commission)
eMMC – Embedded Multi Media Card
FPGA – Field Programmable Gate Array
GPU – Graphical Processing Unit
ICMP - Internet Control Message Protocol
IDE – Integrated Development Environment
IOCTL – System call for device-specific input/output operations
IP – Intellectual property
ISR – Interrupt Service Routine
LVDS – Low voltage differential signals
Mali – A GPU microarchitecture developed by ARM Holdings Ltd.
Mercurium – OmpSs compiler
MPSoC – Multiprocessor System-on-Chip
N.C. – Not connected
Nanos++ – OmpSs runtime
NIC – Network Interface Controller
OS – Operating system
PL – Programmable logic
POSIX - Portable Operating System Interface for UNIX
PS – Processing system
QEMU – Open source machine emulator and virtualizer
RDMA – Remote Direct Memory Access
RISC – Reduced Instruction Set Computer
SDK – Software Development Kit
SoC – System on chip
TCP – Transmission Control Protocol
TTL – Transistor-transistor logic
UART – Universal Asynchronous Receiver-Transmitter serial port
microSD – Micro SD card

1 Executive summary

Please note: THIS DELIVERABLE IS UN UPDATED VERSION OF THE INTIAL D5.1 AND IT WAS INITIALLY CONFIDENTIAL. IN AGREEMENT WITH THE INTERESTED PARTNERS D5.1 IS NOW MADE PUBLIC.

This document is the final version of deliverable D5.1, and it is meant to provide a description on the work done on the following components:

- The AXIOM Linux Distribution, which is the outcome of the build system for the AXIOM board based on PetaLinux;
- The Soft-IP Arduino compatibility layer (including the "Arduino-UNO" interface), which is a soft IP enabling users to program Arduino applications on top of the AXIOM board;
- The Kernel Driver and corresponding user libraries for controlling the AXIOM NIC (the soft-IP has been developed in T6.5 since month m6);
- The AXIOM User Space utilities, needed to initialize and use the AXIOM-link.

Finally, this document covers a short description of the initial AXIOM-link network configuration (including the discovery algorithm used to discover Node IDs).

1.1 *Main changes since the D5.1 preview document at m24*

In order to simplify the reading of this deliverable compared to the preview version, we highlight below the main changes:

- Figure 4 has been slightly changed;
- The AXIOM NIC short messages have now a payload of 248 bytes;
- Section 5.3, a tunnel for Ethernet packets on top of the AXIOM NIC has been implemented;
- Section 6.3, The axiom-ethtap daemon has been added;
- Appendix B and C have been added.

1.2 Relation to other deliverables

This document can be thought as the basis on top of which the following deliverable depends:

- D5.2 Remote memory Access;

This deliverable also has a strong relationship with the following deliverables:

- D6.1 Technical specification of the AXIOM board;
- D6.2 Hardware design production documents;
- D6.3 System prototype, BSP testing and documentation;
- D6.4 Soft-IPs for the FPGA and documentation;
- D6.5 Interconnect API document.

1.3 Tasks involved in this deliverable

Task 5.1 (month 1 - 18): Operating system

What:

This task will deal with the integration/configuration of a Linux distribution containing the drivers for the peripherals, the libraries needed by applications and the development tools for the specific reference platform.

How:

• Partner SECO will be in charge of the porting and configuration of the base Linux distribution;

• Partner SECO will develop/integrate kernel drivers and soft IPs for the on-board devices, as well as the Arduino UNO interface and the display controller;

• Partner EVI will design and develop the kernel driver for the high-speed B2B interconnect;

• Partner EVI will integrate any additional library needed by the use-cases.

Expected output:

- Source code of boot-loader and Linux kernel;

- Pre-compiled binary images of firmware (i.e., boot-loader, Linux kernel and filesystem) for the reference platform;

- Development tools (e.g., compiler, IDE) and scripts for configuring and building new firmware images.

D5.1: Operating system and documentation

This deliverable will consist in a preliminary version of the Linux distribution with drivers for a subset of the hardware peripherals of the reference platform. The delivered package will also contain the development tools (e.g., compiler, debugger, etc.) with a user manual explaining their installation and usage

2 Introduction

The AXIOM Linux Distribution includes all the components, described later on this section, that are necessary to build a basic OS image to be used on the AXIOM board[16], [42].

Standard tools from Xilinx have been modified in order to keep using the IDE provided for the Zynq UltraScale+ MPSoC (which is necessary to integrate FPGA IPs at the system level) to operate on remote repositories for custom bootloader/kernel source code and specific versions of the Ubuntu filesystem.

The BSP is now in its final version, and it has been tested on the final AXIOM boards. Before using the final AXIOM board, the software has also been developed by using the AXIOM simulator (WP7), Xilinx emulators and other prototyping boards (Udoo boards, Trenz boards, Xilinx ZCU102) that were acquired or made available by partners.

This document includes also a description of the architecture of the kernel drivers that manage the AXIOM NIC. In addition to that, a set of software libraries have been developed to provide a messaging interface in user space. On top of the kernel drivers and libraries, a set of AXIOM User Space utilities have been developed, taking care of the cluster initialization through the implementation of a discovery and of a routing table computation algorithm.

2.1 Document structure

Section 3 provides details on the current status of the Linux distribution for the AXIOM boards. After that, Section 4 provides a description of the Soft-IP providing Arduino capabilities, which is integrated in the FPGA fabric to control the Arduino UNO R3 pinout on the AXIOM board.

Next, the AXIOM NIC kernel drivers and user libraries are described in Section 5. Section 6 describes the set of user space applications available to the user, whereas Section 7 and 8 provides a description of the AXIOM-link network configuration and the possible future extensions to the current implementation.

Finally, Appendix A shortly describes the network discovery algorithm used in the cluster, Appendixes B and C shortly refers to the Software Package, User Manual and documentation produced for the partners allowing them to use the AXIOM boards.

3 AXIOM Linux Distribution

In order to generate an OS image to be flashed on the AXIOM boards, all the necessary tools, source code and packages are included in a suite (AXIOM SDK), which is derived from the one originally distributed by Xilinx, enabling software development for Zynq UltraScale+ based projects.

The custom BSP is based on source code distributed by Xilinx, which has been customized to support the AXIOM board.

The AXIOM Consortium agreed that an Ubuntu 16.04 based distribution is the preferred choice since it provides enough flexibility and ease of use compared to the previous choice Ubuntu 14.04. Two variants, based on Ubuntu 16.04, are available.

- Minimal version:
    - Light;
    - Reconfigurable;
    - aarch64;
    - Mali 400-r5p1-01rel0 kernel driver and libraries for X-Window and fbdev.
- Desktop version:
  - Minimal version plus MATE Window Manager.

A list of the packages, which have been included in the filesystem, is available on the AXIOM wiki, please refer to [6].

3.1 AXIOM SDK Components

Build system:

- PetaLinux SDK v2016.3: Xilinx tool to manage the Embedded Linux system;
- XGENIMAGE: command line tool for the customization and building of a complete system image flashable in a bootable SD card.

Source Code:

- linux-4.6.0-AXIOM-v2016.3: Linux kernel 4.6.0 based on Xilinx kernel and customized by SECO;
- u-boot-2016.07-AXIOM-v2016.3: U-Boot version 2016.07 based on Xilinx kernel and customized by SECO.

Template BSP project:

- AXIOM-ZU9EG-2016.3: PetaLinux project for the AXIOM board;
- AXIOM_XCZU9EG_template_v1.8x_2016.3: starting point to configure the PL and PS systems via Vivado 2016.3.

Filesystem Binaries:

- Ubuntu 16.04 (Xenial) without graphical interface (minimal version);
- Ubuntu 16.04 (Xenial) with graphical interface (desktop version).

The XGENIMAGE tool allows to automatically generate, starting from a pre-built BSP, a bootable image to flash a microSD card to boot AXIOM system.

3.2 Driver support for AXIOM board peripherals

The Linux kernel originally available for Zynq UltraScale+ has been modified in order to support the specific hardware configuration implemented by the AXIOM board, whose structure is outlined in Figure 1.

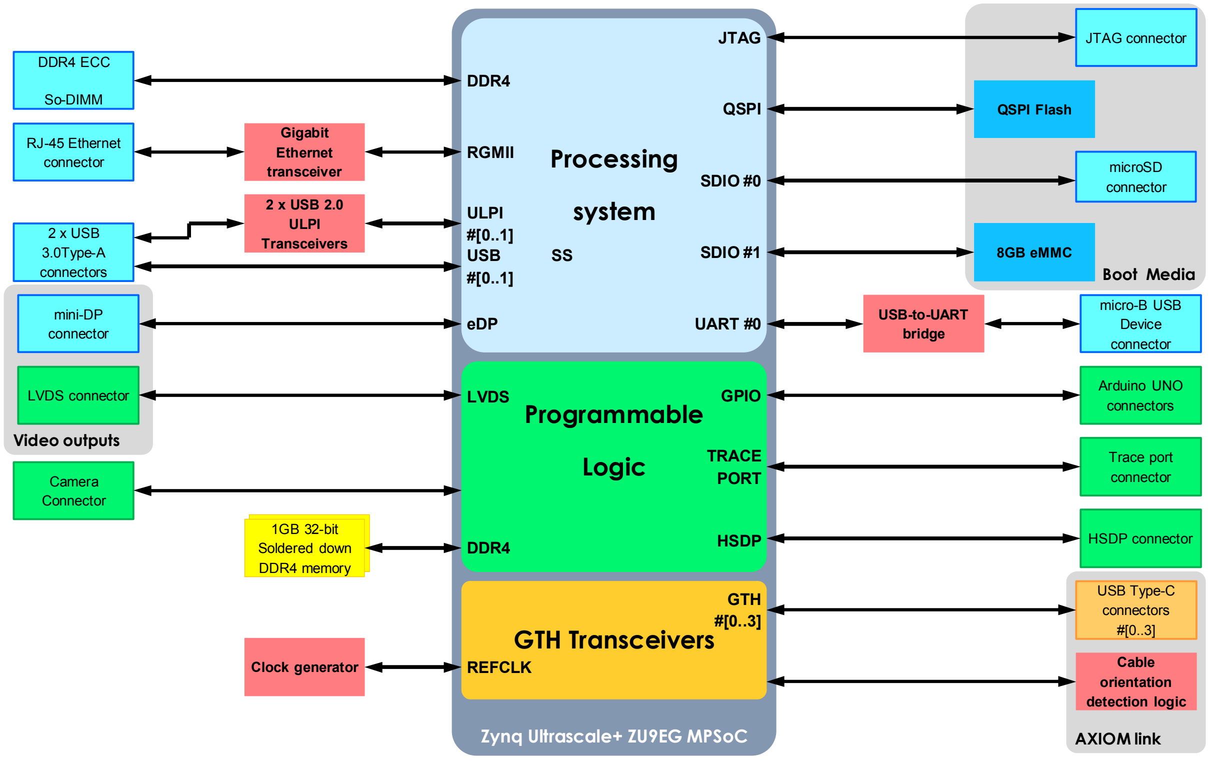


Figure 1 – AXIOM-9EG board, block diagram

All the on-board peripherals are supported by the final version of the Linux kernel. Additional hardware level details are provided in Table 1.

Table 1 – AXIOM board's peripherals.

Device	**P/N**	**Board schematic reference**	**Notes**
eMMC	MTFC8GAKAJCN-1M WT	U15	
microSD	-	CN25	
QSPI Flash	MT25QU512ABB1EW9	U5/U6	
Watchdog timer/ uP supervisor	APX823-29W5	U24	AXIOM custom driver – Watchdog configurable from user space
User LED	-	LED3	
USB 3.0	-	CN9	2 x Host ports
Debug UART	-		CN17
TTL UART	-		
Ethernet	KSZ9031 PHY	CN8	
AXIOM-link	-	CN3, CN4 CN5, CN6	
DisplayPort	-	CN7	
LVDS	-	J1	Optional
MIPI CSI	-	CN14	Optional
Power monitors	INA219A	U[42..49]	AXIOM custom driver for the power monitoring at user space level
Clock generator	Si5341	U22	AXIOM custom driver – Clock providers used also by DRM driver for video output
Mali GPU	-	-	
FPGA Soft-IPs	-	-	Custom drivers to manage: Arduino and AXIOM NIC Soft IPs and application specific accelerators inferred through OmpSs@FPGA

Beyond the customized source code for bootloader and Linux kernel and dedicated filesystem versions, the BSP for the AXIOM board relies on a modified version of the PetaLinux build system, originally distributed by Xilinx.

The custom PetaLinux version can be used to generate the AXIOM board following the workflow outlined below and summarized in Figure 2:

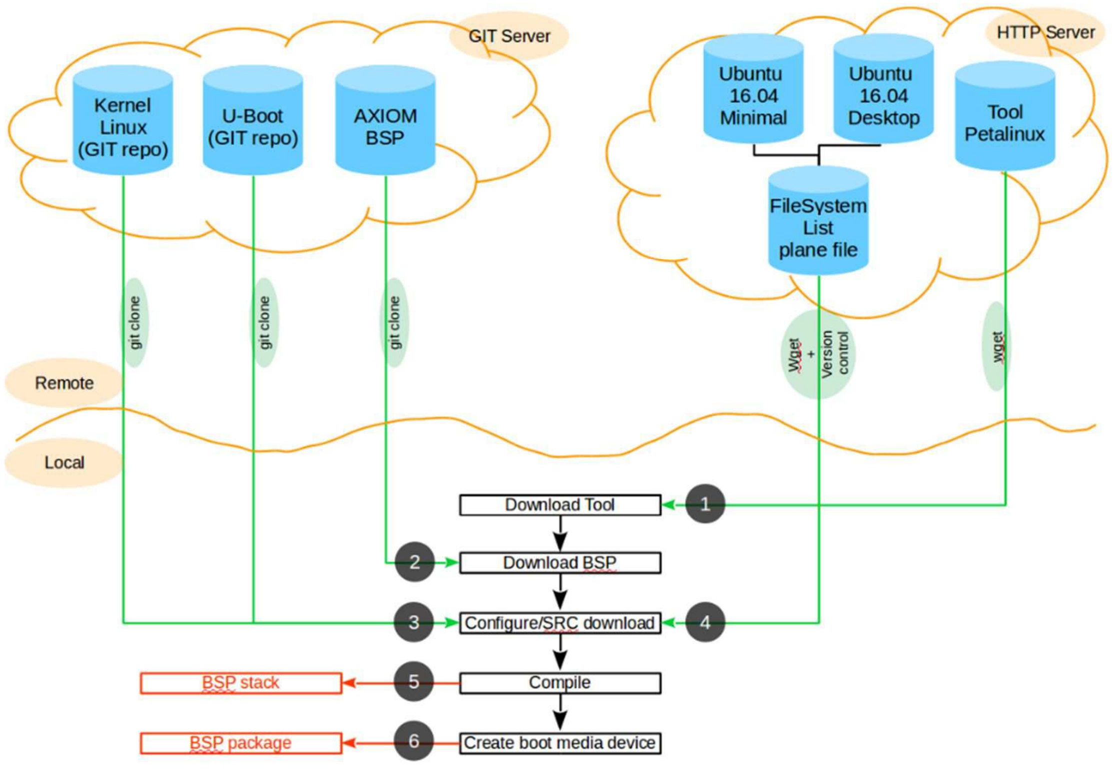


Figure 2 – Workflow for the custom PetaLinux distribution.

1. Download the original PetaLinux tool (from a dedicated server managed by Xilinx)
2. Download the template project to generate PetaLinux BSP for the AXIOM board, which is stored into an AXIOM Git repository [8].
3. Check out bootloader and kernel source code. Local source code trees are no longer used, remote Git repositories replace them.
4. Generate the Filesystem using the custom procedure that has been set up:
    - Download the list of available filesystems;
    - Check the filesystem selection and download of the relative archive via http, after old file cleaning;
    - Create basic target Fsfolder for rootfs.
5. Follow the standard PetaLinux compilation procedure as described by Xilinx's applicable documents.
6. Install the custom target FileSystem: this imply to select one of the available filesystem binary and include specific application packages through PetaLinux build system.

In the case of the usage of OmpSs@FPGA accelerators, autoVivado (explained in D4.3) will generate a hardware design with the baseline AXIOM board integrated with the accelerators. The hardware description file of this design can be used in the standard Petalinux compilation procedure.

4 Soft-IP for Arduino compatibility

The AXIOM board can interface Arduino shields, allowing developers to leverage a plenty of available sensors and actuators to enhance system's functionalities. A dedicated driver takes care about initialization of the device (which is embedded in the FPGA fabric) and provides communication mechanisms between the CPU complex and the Arduino Soft core. A set of command line interfaces (sysfs) and programming command interfaces (ioctl syscalls) has been developed.

For these purposes, the major activities performed are:

- Soft core selection;
- Implementation details analysis;
- Programmability from the PS;
- Communication between PS and AVR8 Soft-IP.

To allow compatibility and Arduino-like programmability, the starting point was the AVR8 Soft-Processor developed by Ruslan Lepetenok. Further information about the original AVR core is provided by [5].

The Atmega103 is an 8-bit microcontroller based on the AVR RISC architecture, which combines a rich instruction set with 32 general-purpose working registers directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle.

The AVR uses a Harvard architecture concept with separate memories and buses for program and data.

Lepetenok's Soft-IP version has been modified by SECO, as shown by Figure 3, in order to add the required infrastructures to interface the CPU complex integrated within the Zynq UltraScale+ device.

The Soft-IP provides the following peripherals:

- 8-bit data;
- Clock Divider;
- 4 Kbytes Data Memory;
- 16 Kbytes Program Memory;
- 8 Ports (for a total amount of 48 GPIO, more than enough to cover the requirements of the Arduino socket);
- 4 Timer/Counters, for implementing PWM outputs;
- UART;
- SPI;
- An interface to the analog to digital converter integrated in Zynq's PL (referred as system monitor);
- I2C.

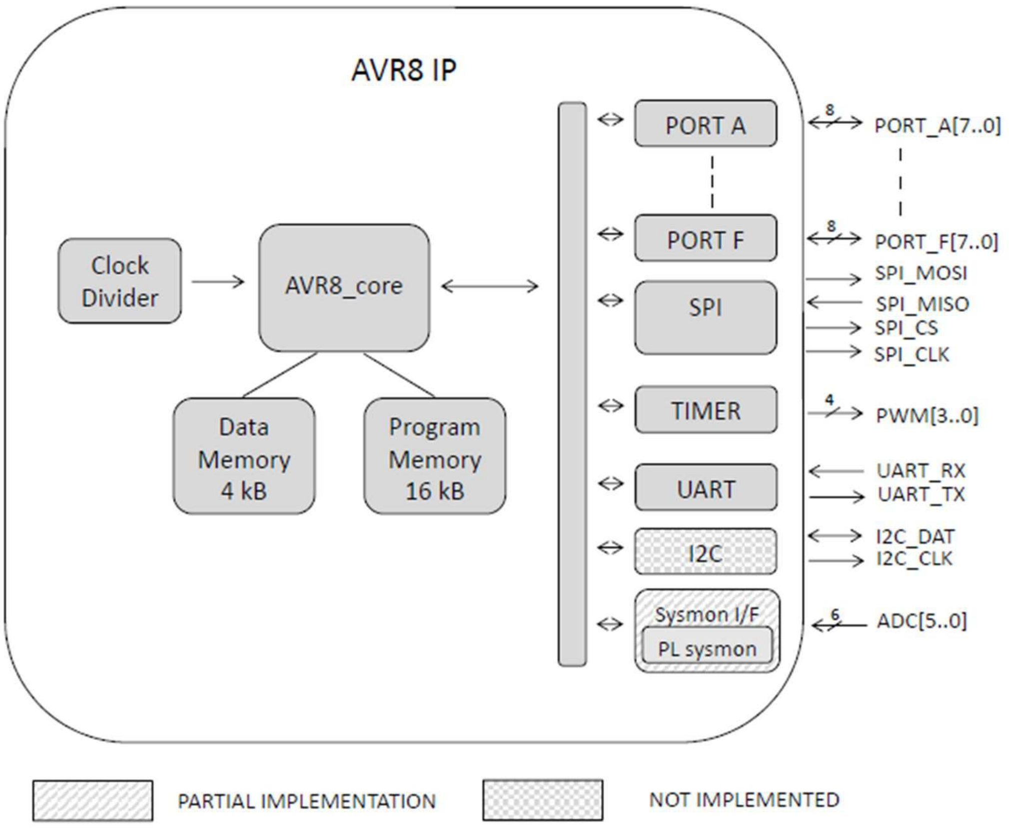


Figure 3 – AVR8 Soft-IP structure.

The Original soft core has been almost preserved in the AXIOM implementation, basically adding Analog-to-digital conversion functionality and I2C connectivity (I2C was only partially completed within the project's framework).

The AVR8 Soft-IP mapping on the Arduino UNO connector is outlined by Table 2, further details are provided by deliverables D6.1 (Technical specification of AXIOM boards) and D6.2 (Hardware design production documents).

Table 2 – Arduino connector pin mapping.

Arduino Uno connector		
Arduino UNO	**AVR Soft-IP core pin**	**XCZU9EG-1FFVC900 Zynq UltraScale+ ball # or Backplane**
PD0	UART_RX	H13
PD1	UART_TX	H14
PD2	PORT_A[2]	J14
PD3	PORT_A[3]	K14
PD4	PORT_A[4]	J15
PD5	PORT_A[5]	K15
PD6	PORT_A[6]	G14
PD7	PORT_A[7]	G15
PB0	PORT_B[0]	F13
PB1	PORT_B[1]	G13
PB2	PORT_B[2]	E15
PB3	PORT_B[3]	F15
PB4	PORT_B[4]	E13
PB5	PORT_B[5]	E14
GND	N/A	connected to board's power planes
AREF	N/A	N.C
PC4	I2C_DAT	AB3
PC5	I2C_CLK	AC3
ADC0	NOT CURRENTLY SUPPORTED	U7
ADC1	NOT CURRENTLY SUPPORTED	U11
ADC2	NOT CURRENTLY SUPPORTED	W11
ADC3	NOT CURRENTLY SUPPORTED	U9
ADC4	ADC[4]	T11
ADC5	NOT CURRENTLY SUPPORTED	R10
VIN	N/A	N.C.
GND	N/A	connected to board's power planes
GND	N/A	connected to board's power planes
5V	N/A	connected to board's power planes
3V3	N/A	connected to board's power planes
PC6	PORT_A[1]	D14
IOREF	N/A	N.C.
N.C.	N/A	N.C.

4.1 Arduino Soft Core programming

The `BRAM_prog_driver` allows mapping the Soft-core program hex file in RAM blocks (for instance the program memory has been mapped to `/dev/memX`). We can use a 32 bit AXI interface that allow, through AXI_BRAM_controller, to initiate write/read transactions from ARM-core to the BRAM of the AVR8 implemented in PL (see Figure 4).

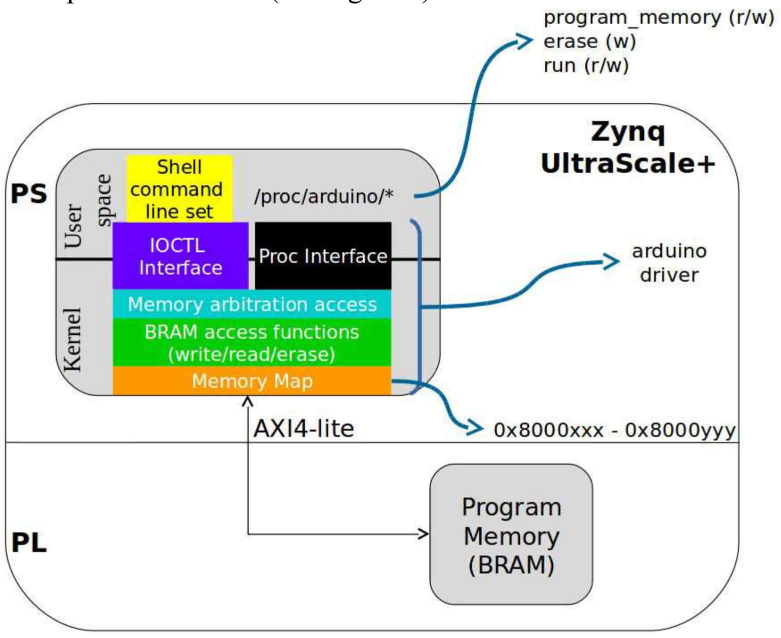


Figure 4 – Architecture of the Arduino Soft Core programming.

In Figure 4, the main components involved on the programming of the Soft Core are colored in the following way:

● Linux standard interface through which the user space can interact with the Soft-IP designed in PL.

● Set of commands to manage the FIFO controllers in order to:

- write/read to/from the program memory of the Arduino soft core
- start/stop program execution on the soft core

● Linux standard interface used to communicate with the Soft-IP implemented in the PL.

● High level function set to manage BRAM access accordingly to the Soft-IP core's status. For example, program execution is stopped when updating the program memory. Moreover it performs `.hex` program file parsing and checksum verification.

● Set of functions used to interact with the BRAM controller.

● The access to the Soft-IP takes place through a reserved memory range, through dedicated memory mapped registers.

The AVR soft-IP is included in the integrated Xilinx Vivado project:

`https://git-private.axiom-project.eu/AXIOM-ZU9EG-2016.3/`

The Arduino IDE is included in the official SD-IMAGE of the AXIOM-board (for convenience):

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

4.2 Arduino Soft Core to Processing System communication

Beyond AVR processor programming, it is useful to provide developers a mechanism to allow communication between the CPU complex integrated in the PS (i.e., the ARM cores) and the AVR Soft Core synthetized in the PL (Figure 5).

The communication between user applications and Arduino Soft Core is provided in a standard way, with a kernel driver and a set of APIs. The first one (yellow block in Figure 4) has the double purpose to implement the communication interface between the Kernel and the Soft Core and the communication interface between the kernel space and the user space, provided as a set of primitives. The second one (blue block in Figure 4) is a dynamic library, which uses these primitives and implements a higher set of functions that can be called by user applications.

There are two main interfaces: 1) user space/kernel space and 2) kernel/Soft Core. Both interfaces are asynchronous, so the driver implements a synchronization mechanism that queues the tasks to perform. The priority can be assigned at run-time or in a static way, based on the specific operation.

As described throughout this section, the Processing System within Zynq UltraScale+ SoC controls, acting as the communication master, the status of the Soft-IP bridge towards the AVR core, which is acting like a communication slave. In order to use the Soft Core as slave device and allow it to notify events to the master, the driver handles an interrupt routine (ISR) in which the slave notification is decoded and the message (from the slave) is passed to the user space via kernel event system. In this way, an asynchronous communication from the slave to the user application is provided.

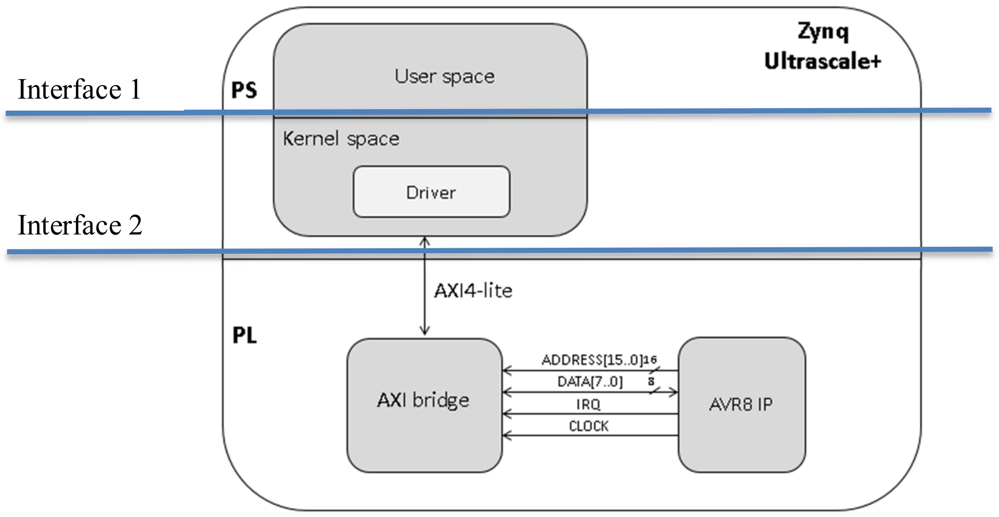


Figure 5 – Asynchronous communication between PS and the Arduino Soft Core.

4.3 Arduino IDE for programming and testing Arduino Soft Core

Arduino IDE version 1.5.2 has been included in the BSP for the AXIOM board. A dedicated version of the Arduino library has been developed and has been used to test the kernel drivers described by Sections 0 and 4.2.

The test setup consists of the AXIOM-board accomodating some of the most popular Arduino shields, like the 'Arduino 4 relays shield' (further information is provided by [4]) as shown in Figure 6, as well as prototyping hardware (like EEPROM, potentiometers, etc) which is a common usage for Arduino.

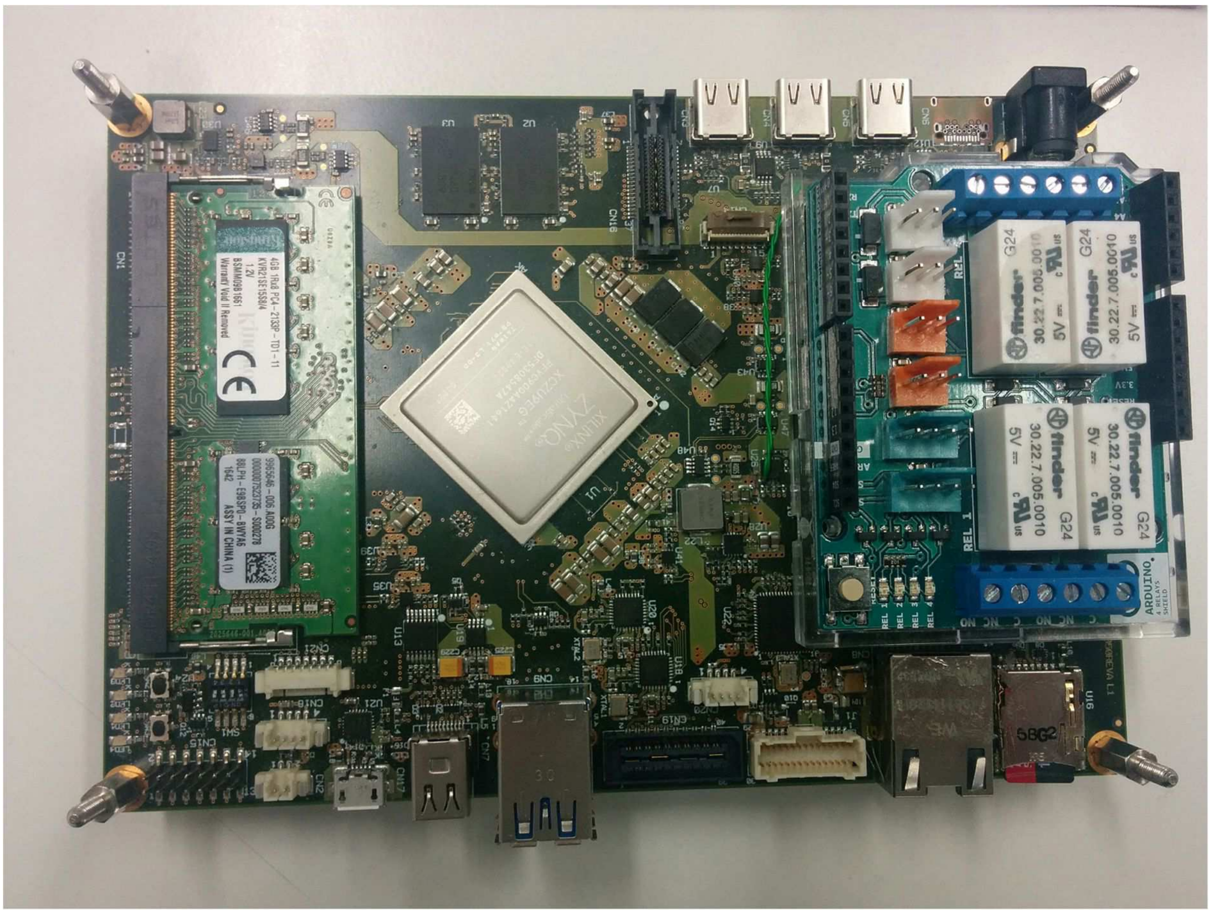
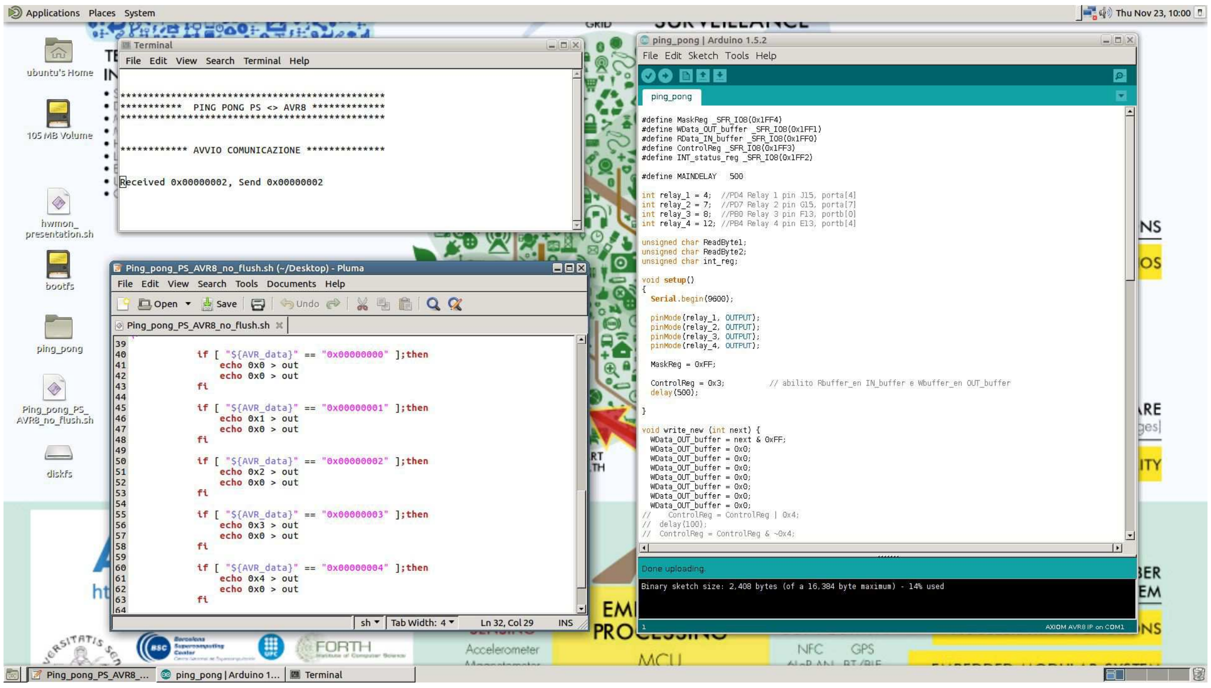


Figure 6 – Arduino 4 relays shield plugged on the AXIOM-9EG board

For testing purposes simple programs for the Arduino Soft Core, usually referred in the Arduino community as 'sketches', were developed and included in the AXIOM BSP. Figure 7 shows a screenshot of such a sketch (programmed on the AVR core's program memory through the Arduino IDE) and a Bash script (which runs on the PS) exchanging numeric values in a ping-pong fashion.


Applications Places System
Terminal
File Edit View Search Terminal Help
Ping_pong_PS_AVR8_no_flush.sh (~/Desktop) - Pluma
File Edit View Search Tools Documents Help
Open Save Undo
Ping_pong_PS_AVR8_no_flush.sh
sh Tab Width: 4 Ln 32, Col 29 INS
ping_pong | Arduino 1.5.2
File Edit Sketch Tools Help
ping_pong
Done uploading.
AXIOM AVR8 IP on COM1
Thu Nov 23, 10:00

```
************************************************
***********  PING PONG PS <> AVR8 *************
************************************************


************ AVVIO COMUNICAZIONE **************


Received 0x00000002, Send 0x00000002
```

```

        if [ "${AVR_data}" == "0x00000000" ];then
            echo 0x0 > out
            echo 0x0 > out
        fi

        if [ "${AVR_data}" == "0x00000001" ];then
            echo 0x1 > out
            echo 0x0 > out
        fi

        if [ "${AVR_data}" == "0x00000002" ];then
            echo 0x2 > out
            echo 0x0 > out
        fi

        if [ "${AVR_data}" == "0x00000003" ];then
            echo 0x3 > out
            echo 0x0 > out
        fi

        if [ "${AVR_data}" == "0x00000004" ];then
            echo 0x4 > out
            echo 0x0 > out
        fi

```

```
#define MaskReg _SFR_IO8(0x1FF4)
#define WData_OUT_buffer _SFR_IO8(0x1FF1)
#define RData_IN_buffer _SFR_IO8(0x1FF0)
#define ControlReg _SFR_IO8(0x1FF3)
#define INT_status_reg _SFR_IO8(0x1FF2)

#define MAINDELAY   500

int relay_1 = 4;  //PD4 Relay 1 pin J15, porta[4]
int relay_2 = 7;  //PD7 Relay 2 pin G15, porta[7]
int relay_3 = 8;  //PB0 Relay 3 pin F13, portb[0]
int relay_4 = 12; //PB4 Relay 4 pin E13, portb[4]

unsigned char ReadByte1;
unsigned char ReadByte2;
unsigned char int_reg;

void setup()
{
  Serial.begin(9600);

  pinMode(relay_1, OUTPUT);
  pinMode(relay_2, OUTPUT);
  pinMode(relay_3, OUTPUT);
  pinMode(relay_4, OUTPUT);

  MaskReg = 0xFF;

  ControlReg = 0x3;          // abilito Rbuffer_en IN_buffer e Wbuffer_en OUT_buffer
  delay(500);

}

void write_new (int next) {
  WData_OUT_buffer = next & 0xFF;
  WData_OUT_buffer = 0x0;
  WData_OUT_buffer = 0x0;
  WData_OUT_buffer = 0x0;
  WData_OUT_buffer = 0x0;
  WData_OUT_buffer = 0x0;
  WData_OUT_buffer = 0x0;
  WData_OUT_buffer = 0x0;
//    ControlReg = ControlReg | 0x4;
//  delay(100);
//  ControlReg = ControlReg & ~0x4;
```

```
Binary sketch size: 2,408 bytes (of a 16,384 byte maximum) - 14% used
```

Figure 7 – Arduino IDE running on the AXIOM-9EG board.

Table 3 outlines the current support of functions described by the Arduino API reference.

Table 3 – Arduino supported APIs in AXIOM BSP.

FUNCTION'S NAME	**CURRENT SUPPORT STATUS**	**NOTES**
	Digital I/O	
digitalRead()	Supported	
digitalWrite()	Supported	
pinMode()	Supported	
	Analog I/O	
analogRead()	Supported	Currently just one analog input is supported. Extension to support six inputs is in progress
analogReference()	Not supported	AREF pin is not connected, analogReference() is fixed to DEFAULT value. Might be extended in future releases of the AXIOM board
analogWrite()	Supported	
	Advanced I/O	
noTone()	Supported	
pulseIn()	Supported	
pulseInLong()	Supported	
shiftIn()	Supported	
shiftOut()	Supported	
tone()	Supported	
	Time	
delay()	Supported	
delayMicroseconds()	Supported	
micros()	Supported	
millis()	Supported	

Math		
abs()	Supported	
constrain()	Supported	
map()	Supported	
max()	Supported	
min()	Supported	
pow()	Supported	
sq()	Supported	
sqrt()	Supported	
Trigonometry		
cos()	Supported	
sin()	Supported	
tan()	Supported	
Characters		
isAlpha()	Supported	
isAlphaNumeric()	Supported	
isAscii()	Supported	
isControl()	Supported	
isDigit()	Supported	
isGraph()	Supported	
isHexadecimalDigit()	Supported	
isLowerCase()	Supported	
isPrintable()	Supported	
isPunct()	Supported	
isSpace()	Supported	
isUpperCase()	Supported	
isWhitespace()	Supported	
Random Numbers		
random()	Supported	
randomSeed()	Supported	
Bits and Bytes		
bit()	Supported	
bitClear()	Supported	
bitRead()	Supported	
bitSet()	Supported	
bitWrite()	Supported	
highByte()	Supported	
lowByte()	Supported	
External Interrupts		
attachInterrupt()	Supported	
detachInterrupt()	Supported	
Interrupts		
interrupts()	Supported	
noInterrupts()	Supported	
Communication		
serial	Supported	
stream	Supported	

5 Kernel driver and User Libraries for the High-speed board-to-board interconnect

One of the parts of the AXIOM Project is related to the design and implementation of a fast board-to-board connection (named AXIOM-link) which is able to provide a high-performance connection between boards capable of RDMA. This section is dedicated to the description of the main components of the hardware and software architecture related to the AXIOM-link interface. This interconnection provides the basis for the implementation of the transport layer that is used by the AXIOM Allocator and by the GASNet Conduits described in D5.2.

5.1 High level overview of the AXIOM-link architecture

The main components of the AXIOM-link infrastructure are the following (see Figure 8):

- **AXIOM NIC**
  Implemented in the FPGA by partner FORTH. The AXIOM NIC exports a set of registers to the computing part of the FPGA. These registers are described in the Datasheet [1].
- **AXIOM NIC Device Driver**
  Implements the kernel-related part of the infrastructure, which is responsible to:
    - properly handling the AXIOM NIC registers;
    - provide abstractions for communication ports;
    - provide an IOCTL interface to the user space;
    - extend the hardware buffering space in memory, thanks to the usage of kernel threads.
- **AXIOM User Libraries**
  The AXIOM User Libraries are responsible for providing a comprehensive C API, which can be used by the AXIOM applications to interact with the network interface in a simpler way.

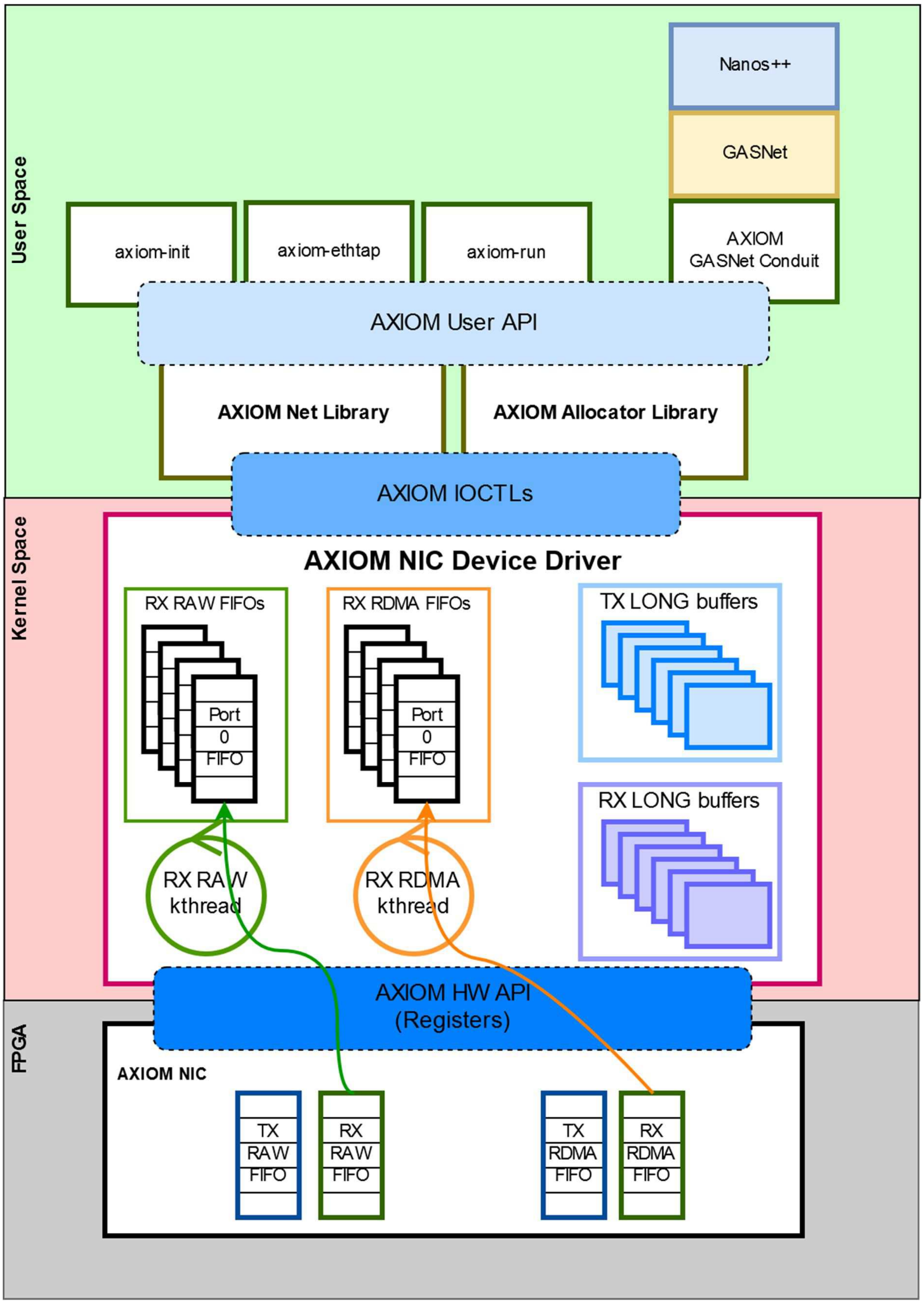


Figure 8 – The AXIOM NIC Architecture highlighting the kernel drivers and the user libraries.

5.2 AXIOM NIC features

In this subsection, we recall the main features of the AXIOM NIC, which are useful to understand how the higher software layers use them to implement the various features provided. A complete reference document regarding the AXIOM NIC is available in the provided archive (see Appendix B).

The main features we want to highlight in this step are the following:

- **Multiple queues of descriptors.**
  Each kind of message is mapped for reasons of efficiency to separate message queues.
- **Interrupt moderation.**
  The AXIOM NIC only generates interrupts when the queue status changes (from empty to not empty, or from full to not full). This is used to limit the interrupt overhead compared to a configuration where an interrupt is sent at each message arrival. The driving idea is that, once the first interrupt is launched, the device driver will read all the NIC messages at once, with higher efficiency.
- **Link status monitoring.**
  The AXIOM NIC is able to provide information about the fact that a specific interface of the board is connected or not to another board. This feature is particularly useful during the discovery algorithm.
- **Routing decision based on Routing Tables.**
  The routing of messages in the AXIOM NIC is based on a store & forward technique, where each node maintains a routing table holding the information about the interface to be used to reach a specific node. Please note that the routing tables are computed by software (in particular by the `axiom-init` application, see Section 6.1), and stored in hardware (one register for each node, see Section 5, Routing Registers, of [1]).
- **Multiple type of messages.**
  The network interface is able to concurrently send two types of messages, small and big messages. Small messages are RAW messages with a payload (embedded in the descriptor) of up to 248 bytes (theoretically 255 but limited to guarantee 8-byte alignment). RAW messages can be directed to a specific node, or to a neighbor interface (this to implement the discovery algorithm). Big messages are used for RDMA read/writes, as well as for LONG messages, which are messages using the RDMA subsystem, with a destination address taken from a pool of buffers provided by the receiver node.
- **Multiple queues available.**
  The AXIOM NIC provides 2 set of queues (one for small and one for big messages, see Figure 9). The descriptor of each message has the possibility to specify 8 ports, which will be processed inside each node to provide a bind mechanism similar to the one available in the TCP protocol.

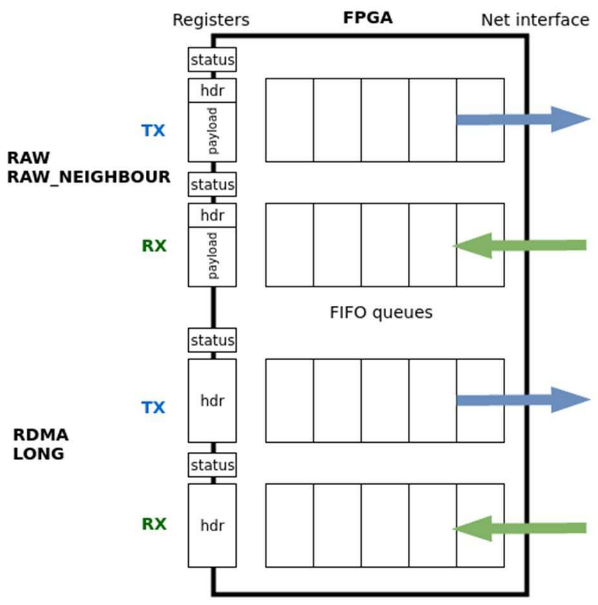


Figure 9 – Multiple queues available for the different kind of messages in the AXIOM NIC.

5.3 General guidelines used for the design of the Linux Device driver and library architecture

During the design of the network infrastructure, we took special care about defining a complete system that could be efficiently mapped on the parallel programming libraries running in user space. Moreover, we took inspiration from some of the latest works in the networking area, such as Netmap [2], to try to implement an architecture that could efficiently use the power of the AXIOM NIC and of the underlying high-speed transceivers in the FPGA. For this reason, we followed these guidelines during the implementation of the AXIOM Drivers and during the design of the AXIOM NIC registers:

- **Network management in user space.**
  Compared to the standard TCP/IP Linux stack, we tried to move various features and network support in user space, limiting the AXIOM NIC Device Driver to its main task of handling the delivery of the packets between the user space applications and the AXIOM NIC registers. This include the support for control packets (like the ICMP ping), the discovery algorithm and routing calculation, and so on. This has the advantage of limiting the amount of kernel code, while still maintaining good performance.
- **Possibility to support memory mapped registers exposed to the user space.**
  We left open the possibility to use memory mapped registers to avoid the use of IOCTL on each packet transfer, and to reduce memory copies. On the other hand, this requires the availability of separate per-process network queues to limit the overhead that is linked to mutual exclusion on the NIC register sets.
- **Possibility to use the NIC as tunnel for Ethernet packets.**
  Although not included in the DoA, we added the possibility to have a virtual Ethernet device mapped on top of the AXIOM NIC as a replacement for a standard Linux network interface. The idea is that the LONG messages will be used as datagrams, giving the possibility for TCP/IP packets to be sent on top of the AXIOM NIC layers. This has the advantage to reduce cabling between the boards (only the AXIOM-link is needed), allowing legacy applications based on TCP/IP to run unmodified on top of the AXIOM-link network.

5.4 AXIOM-link APIs at different levels

Based on the main components of the AXIOM-link architecture (NIC, Device Driver, User Libraries), we defined three levels of API:

- AXIOM NIC register API, documented in [1], which is a datasheet-like description of the registers of the AXIOM NIC, and which has been specified in collaboration between partners FORTH and EVI.
- AXIOM IOCTL API, an internal API documented in [3], which is used to handle user/kernel space data passing and notifications.
- AXIOM User API, documented in [3], which is an API implemented in a set of dynamic libraries, which is used to provide a simple interface for the AXIOM applications for the IOCTL API.

5.5 AXIOM NIC Device Driver and User Library internal structure

The AXIOM NIC registers are handled internally in the Linux Kernel by a dedicated driver, named "`axiom_netdev.ko`". The AXIOM NIC Device driver is responsible for interfacing the AXIOM NIC registers with the user space.

The internal architecture (depicted in Figure 8) can be summarized in the following main components:

- The driver is divided in two parts, the RX and the TX part.
- The RX path (green box in the kernel part of the Figure 8), shows a kernel thread, which is responsible for fetching packets from the hardware queue, and for dispatching them in the per port queues. After that, the IOCTL implementation only polls from the available per-port queues.
- The TX path (blue box in the kernel part of Figure 8), pushes directly the messages in the transmission queue, or blocks waiting for an empty descriptor.

The AXIOM User Library provides a set of primitives that can be used to directly interact with the AXIOM Device Driver exported IOCTLs. In addition to the AXIOM RAW messages, the system also offers the possibility to perform RDMA transactions.

The RDMA transactions that can be issued by the AXIOM User Library allowing the setup of an RDMA transaction on a specific RDMA enabled memory region. The size of the region must be multiple of 8 bytes, and its maximum size of 512 Kbytes was determined together by the partners EVI, BSC and UNISI to allow the transmission in a single transfer of most of the RDMA operations.

In addition to the RDMA transfers, the API allows the sending of datagram messages using the LONG messages of the AXIOM NIC. In this case, LONG messages have a maximum payload of 4Kbytes (enough to store a standard Ethernet packet), supports a maximum of 8 ports, and works on top of a set of preallocated buffers that are constantly refurbished by the kernel driver. Currently there is only one of such message pools in the AXIOM NIC, forcing in this way a memory copy between the kernel and the user space (this memory copy could be removed if there could be the possibility of a per-port pool of messages). The pool of messages is allocated at the beginning of the RDMA memory zone, before the initialization of the Level 1 allocator (see D5.2), as shown in Figure 10.

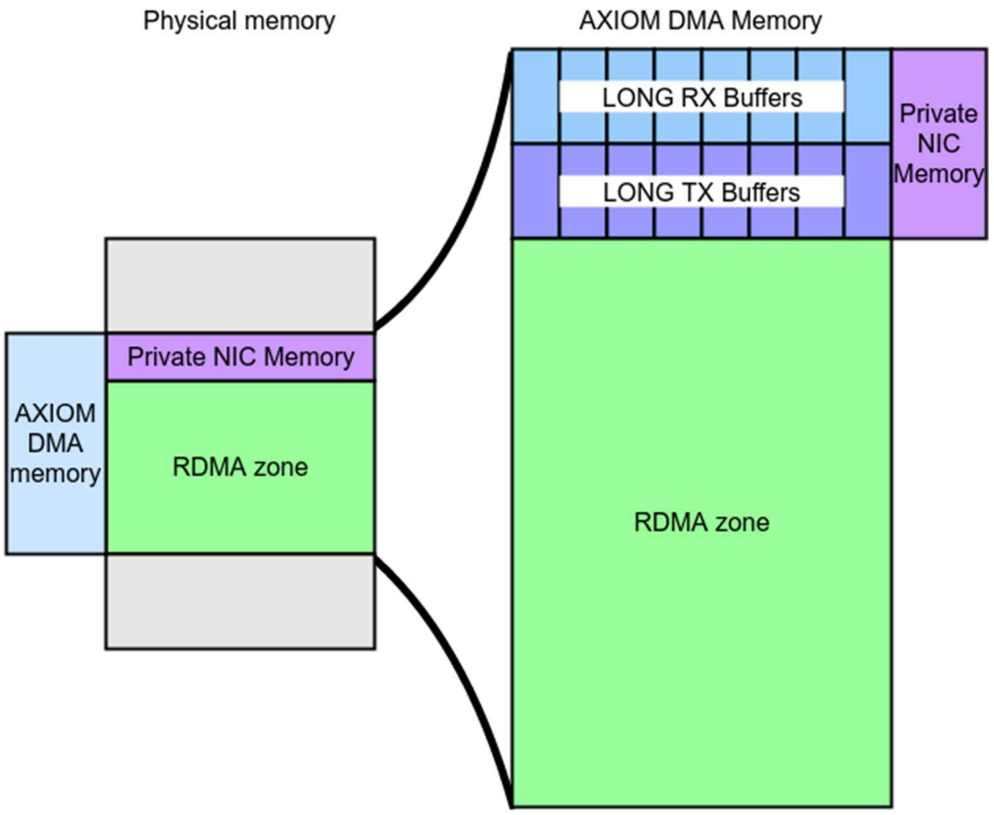


Figure 10 – Allocation of the pool of LONG messages in the RDMA zone of the AXIOM NIC.

6 AXIOM User Space utilities

Together with the AXIOM Device Driver and the AXIOM User Libraries, we developed a set of accompanying applications. Some of them implement a needed subset of functionalities that have been "moved" to userspace, such as the network discovery. Other utilities are command line applications useful for testing and scripting functionalities. All utilities have a built-in help that is printed by using the `-h` command line parameter. The following is a short description of these utilities.

6.1 *`axiom-init`*

`axiom-init` is the main daemon running on all nodes of the cluster. It is responsible for the following features:

- The network startup and first configuration (see Appendix A), in particular:
  - the discovery protocol, which is a recursive algorithm used to traverse all nodes and assign the node IDs;
  - the routing table computation on the master node and the consequent distribution to all nodes;
- The handling of control messages (which in a typical TCP/IP implementation are handled by the network stack in the kernel, as for example the ICMP messages);

The `axiom-init` daemon can be started in master or slave mode. Only one node can be started in the network as master node: in fact, the master node is responsible for starting the discovery algorithm as well as the routing table calculation and distribution.

6.2 `axiom-info`

This application prints a set of information about the AXIOM NIC, including:

- node ID;
- status of the board interfaces;
- number of nodes on the network;
- routing table for the node;
- status and control registers values.

6.3 `axiom-ethtap`

This application is a daemon which can run on each node in the cluster to map Ethernet frames over AXIOM-link messages. The daemon creates a virtual Ethernet device (TAP) on top of the AXIOM NIC. The AXIOM Node ID is encoded in the MAC address of the virtual Ethernet device. In this way it is easy to map each Ethernet frame on top of AXIOM LONG messages, where the destination Node ID is decoded starting from the destination MAC address contained in the Ethernet frame.

6.4 `axiom-netperf`

This application can be used to estimate the throughput between two nodes. The idea of this tool is to create a flood of packets (RAW, RDMA or LONG) with a given payload size to a given destination node. The tool returns the transmission speed and the number of transmitted packets.

6.5 `axiom-ping`

This application is used to estimate the message round-trip time between two nodes. It works in a way similar to the `ping` application on UNIX systems. AXIOM ping packets are handled on the receiving node through `axiom-init`.

6.6 `axiom-rdma`

This application is used to issue a RDMA operation on a node. The application can be used as an example, and is capable of dumping the content of the RDMA enabled zone.

6.7 `axiom-recv`

This application is used to issue a receive request for a RAW or LONG message on a port. The request is selectable to be blocking or non-blocking.

6.8 `axiom-run`

This is the AXIOM process spawner (see deliverable D5.2), which is used to initialize the AXIOM allocator and to start GASNet and OmpSs application on the nodes composing the cluster.

6.9 `axiom-send`

This application can be used to send a RAW or LONG message to a specific node.

6.10 `axiom-traceroute`

This application is used to provide a trace of the routing path between the current node and a remote node. The application uses the services provided by the `axiom-init` daemon.

6.11 `axiom-utility`

This application can be used to flush stale messages remained unread in the AXIOM NIC message queues.

6.12 `axiom-whoami`

This application prints the current node ID, set after the discovery algorithm has run.

7 Initial AXIOM-link network configuration

The AXIOM NIC interface and applications described so far in the document allow to send and receive packets between applications running in a cluster. It is assumed that each node has a unique ID in the network (this is guaranteed by the discovery algorithm, see later), and that the network connections supported should be both regular topologies (like a ring or a 2D mesh), or irregular (that is, the boards are connected in a "pseudo-random" way).

We expect irregular topologies to be common in the "makers" community, where we cannot expect users to have a precise knowledge about ring, mesh or other topologies. On the other hand, we can also expect that irregular topologies will be the simpler to wire together, as they will typically require less wires (as an example, consider two nodes: a ring topology will require two wires, whereas an irregular topology can be obtained with just one wire).

For these reasons, we designed the system to allow an automatic configuration of the node IDs. At startup, we suppose that all the boards are connected "somehow" each other using AXIOM-link wires. One of the nodes is also connected to the outside world using Keyboard/Mouse/Video or Ethernet or WiFi; that specific node where commands are launched first is then marked to be the "master" node. The master node starts the discovery algorithm, and as a result, the system is configured with unique node IDs. After that, the master node is responsible to compute and distribute the routing tables to all nodes. All these steps are implemented inside the `axiom-init` daemon.

As a final step of network configuration, the `axiom-ethtap` application is run and proper MAC and IP addresses will be assigned to each node (based on the AXIOM node ID). Having Ethernet addresses configured this way also allows starting a remote filesystem, which could be resident in the master node. This in turn would simplify the distribution of the AXIOM applications on the various nodes, as the AXIOM applications needs to be copied only on the master node shared filesystem.

8 Possible extensions

We envision a set of improvements to be implemented on top of the AXIOM-link:

- Usage of memory mapped registers directly exposed to the user space. In this way, it will be possible to avoid IOCTL system calls for each packet transfer, also reducing memory copies. This requires sharing message queue descriptors between user and kernel spaces, and special care will have to be done in order to provide mutual exclusion and synchronization between different users.
- Usage of multiple hardware queues in order to enhance the parallelism on the same node. In this way, it will be possible to statically allocate a queue to an application or to a core, which will access the register set in an atomic and dedicated way without the need of synchronization.

9 Confirmation of DoA objectives and Conclusions

This document provides the final version of deliverable D5.1, and is an improved version of the preview delivered at month 24.

Other related publications of this project can be found in the references [9]-[47].

All the infrastructure has been tested on the real board and all the goals of the task T5.1 has been met.

Appendix A

This appendix shortly describes the nodes discovery algorithm and the testing environment designed by EVI to test the algorithm on several network connections scenarios.

The initial status of the discovery is that all nodes are connected with bi-directional point-to-point connections, forming an unknown topology. Nodes ID has not been assigned yet. Each node has some interfaces, numbered starting from 0. The user connects to a node by standard means (e.g., console, WiFi) to a first node, which will be called "master" (note that potentially any node can be the master).

From the master node, the discovery algorithm is started by the `axiom-init` daemon run in "master" mode. First, the master node will auto-assign the first available node ID, which is ID 1. The node then initiates a recursive algorithm (summarized in pseudo-code in Figure 11) on all its interfaces, which works as follows:

- If the neighbor node on the other side of a given interface already has an ID, skip it and go to the next interface.
- If the neighbor node on the other side of a given interface does not have an ID, assign it the next available ID, then start the discovery algorithm recursively on it.
- When all interfaces have been processed, return the next available ID to the calling node.

At the end of its execution, the discovery algorithm assigns the node IDs to all nodes of the network, and also produces a table representing the topology of the network. Each row of the table represents a point-to-point connection, which contains a pair of tuples (ID, interface) representing both endpoints of the connection.

```
/* AXIOM Recursive Discovery Algorithm */
int ax_discovery(node, next_id)
{
  node.my_id = next_id++;
  for <each neighbor> {
    if <neighbor node already has an ID> {
      <skip it >
    } else {
      next_id = ax_discovery(neighbor, next_id);
    }
  }
  return next_id;
}

/* start the discovery algorithm on the master node */
next_id = 1;
ax_discovery(master, next_id);
```

Figure 11 – High-level description of the Nodes Discovery Algorithm.

The table, which is produced as a result of the discovery algorithm, is then used by the routing table computation phase. In particular, the system computes, for each node, a local routing table useful to reach each node. The local routing table contains, for each node in the cluster, the interface to which a packet directed to that node should be forwarded. The algorithm is based on the shortest path routing and provides only one possible forwarding interface for each node.

In order to test the correctness of the discovery algorithm implementation a wrapper of the AXIOM RAW messaging API on top of Linux has been implemented. In this way, it has been possible to test various topologies and analyze the correctness of the discovery algorithm and of the routing table computation. The source code of this POSIX wrapper is available in the AXIOM Git repository [7]. This wrapper was developed before the availability of the QEMU infrastructure described in Deliverable D4.2, and has been deprecated after the QEMU setup was working.

Appendix B

The software package, which is part of this delivery, contains the code and the documentation. The software package can be downloaded at the following address:

https://download.axiom-project.eu/?dir=RUNTIME

The content of the software package is described in the README.txt inside the software package. We show it below:

```
                        AXIOM D5.1 Software release
===============================================================================

This archive contains software and documentations relative to D5.1 deliverable.
It is organized with the following directories:

* docs
  This directory contains the documentation of AXIOM BSP and AXIOM
  Software-Stack and it is organized with the following sub-folders:

    * BSP
      This folder contains the documentation of AXIOM Board and BSP:
        * WP6 - ARCHITECTURE IMPLEMENTATION_AXIOM board - AXIOM Wiki.html
          This page describes the AXIOM board. It is based on the Zynq®
          UltraScale+™ MPSoC XCZU9EG-1FFC900.

        * WP6 - ARCHITECTURE IMPLEMENTATION_BSP_ComponentsAndSources - AXIOM Wiki.html
          This page describes the tools and the components of the AXIOM BSP.

        * WP6 - ARCHITECTURE IMPLEMENTATION_BSP_FirstBoot - AXIOM Wiki.html
          This page describes how to use and create bootable SD with
          pre-compiled images and FileStystem.

        * WP6 - ARCHITECTURE IMPLEMENTATION_BSP_KERNEL_PowerMonitor - AXIOM Wiki.html
          This page explains how to monitor the power consumption using SYSFS
          and IOCTL interfaces.

        * WP6 - ARCHITECTURE IMPLEMENTATION_BSP_PetalinuxTool - AXIOM Wiki.html
          This page provides the desctiption of a sub-set of the PetaLinux's
          commands, useful to generate the AXIOM BSP.

        * WP6 - ARCHITECTURE IMPLEMENTATION_BSP_PMT - AXIOM_WIKI.html
          This page describes how to use a simple set of bash scripts to acquire
          data about power consumption.

        * WP6 - ARCHITECTURE IMPLEMENTATION_BSP_XGENIMAGE - AXIOM Wiki.html
          This page describes the xgenimage tool. It is a tool to generate OS
          image for the AXIOM Board.

    * SoftwareStack
      This folder contains the documentation of AXIOM Software Stack:
        * WP5 - AXIOM Software Stack - AXIOM Wiki.html
          This page contains instruction on how to get, install, and run the
          AXIOM NIC drivers and user-space applications developed for the
          AXIOM board.
          We added the following files that are linked in the wiki page above:
            * axiom-evi_README-v1.0.txt
              How to compile the AXIOM Software Stack and generate DEBs package
              for the AXIOM board.
            * axiom-evi-apps_README-v1.0.txt
              How to run AXIOM applications and utilities
            * axiom_nic_datasheet-v1.0.pdf
              Datasheet of the real AXIOM NIC peripheral on the FPGA
            * axiom-v1.0-doxygen-html.tgz
              Doxygen documentation HTML of the AXIOM Software Stack
            * axiom-evi-apps_tests_README-v1.0.txt
            How to run the AXIOM regression tests
          * axiom-evi_tests_gasnet_README-v1.0.txt
            How to run the GASNet regression tests
          * axiom-evi_tests_ompss_README-v1.0.txt
            How to run the OmpSS regression tests

* software
    This directory contains the source code developed and it is organized with
    the following sub-folders:

    * BSP
      This directory contains the Petalinux 2016.3 project. It is the main
      component of the Xilinx processing systems and it includes both hardware
      and software setting for the AXIOM board.
        * AXIOM-ZU9EG-2016.3_7948670.tar.gz

    * BSP_Component
      This directory contains the source code of Linux and U-Boot. The Linux
      kernel 4.6.0 and U-Boot 2016.07 are based on Xilinx kernel and customized
      for the AXIOM board.
        * linux-4.6.0-AXIOM-v2016.3_aa575e7.tar.gz
        * u-boot-2016.07-AXIOM-v2016.3_16eb184.tar.gz

    * FileSystem
      This directory contains File System with Ubuntu 16.04.
      The 'minimal' version contains only tools needed for the basic board usage.
      The 'desktop' version is built on the minimal version, adding Mate Desktop
      environment and the video graphical acceleration driver.
        * zynq-ubuntu-desktop_16.04-xenial_v1.3.tar.gz
        * zynq-ubuntu-minimal_16.04-xenial_v1.3.tar.gz

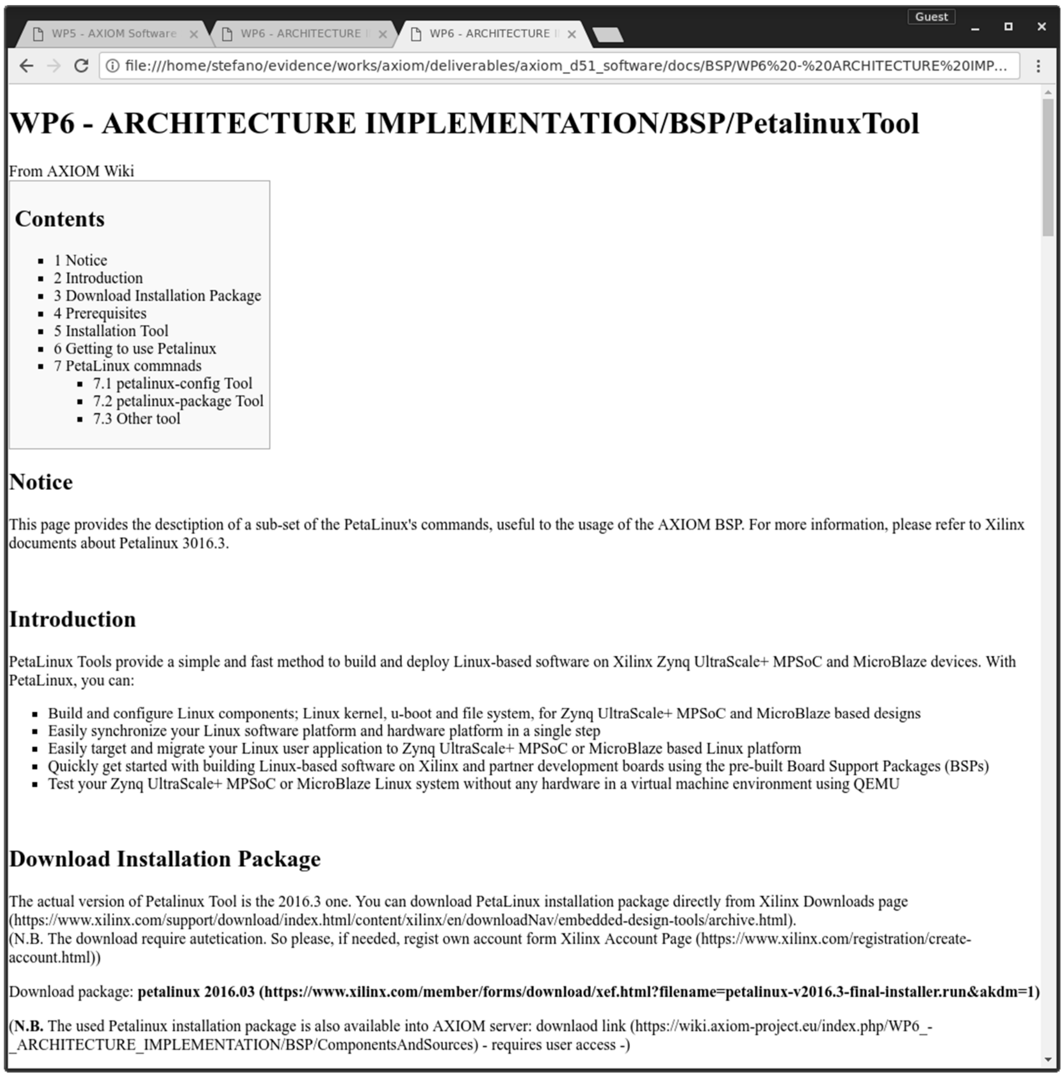
    * SoftwareStack
      This directory contains the archives of the source and DEBs of AXIOM Software
      Stack v1.0. It includes AXIOM drivers (NIC and MemoryDevice), AXIOM UserSpace
      libraries, AXIOM application, GASNet with AXIOM Conduit, Nanos and
      Mercurium with AXIOM support and several regression tests.
        * axiom-evi-software_stack_src_v1.0.tar.gz
        * axiom-evi-debs-v1.0.7z

    * Tools
      This directory contains the tool used to create a Ubuntu 16.04 based
      FileSystem. As output of the execution of the tool there is a FileSystem
      like the pre-build desktop one.
        * xgenimage_6c40bc3.tar.gz
```

Appendix C

The User Manual of the AXIOM BSP and boards has been included in the form of Wiki pages in the internal AXIOM Wiki, in particular at the following pages:

- https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTA-TION/BSP/PetalinuxTool
  For the instructions on how to use the PetaLinux Tool.


WP5 - AXIOM Software
WP6 - ARCHITECTURE I
WP6 - ARCHITECTURE I
Guest
file:///home/stefano/evidence/works/axiom/deliverables/axiom_d51_software/docs/BSP/WP6%20-%20ARCHITECTURE%20IMP...

WP6 - ARCHITECTURE IMPLEMENTATION/BSP/PetalinuxTool

From AXIOM Wiki

Contents

Notice

This page provides the desctiption of a sub-set of the PetaLinux's commands, useful to the usage of the AXIOM BSP. For more information, please refer to Xilinx documents about Petalinux 3016.3.

Introduction

PetaLinux Tools provide a simple and fast method to build and deploy Linux-based software on Xilinx Zynq UltraScale+ MPSoC and MicroBlaze devices. With PetaLinux, you can:

- Build and configure Linux components; Linux kernel, u-boot and file system, for Zynq UltraScale+ MPSoC and MicroBlaze based designs
- Easily synchronize your Linux software platform and hardware platform in a single step
- Easily target and migrate your Linux user application to Zynq UltraScale+ MPSoC or MicroBlaze based Linux platform
- Quickly get started with building Linux-based software on Xilinx and partner development boards using the pre-built Board Support Packages (BSPs)
- Test your Zynq UltraScale+ MPSoC or MicroBlaze Linux system without any hardware in a virtual machine environment using QEMU

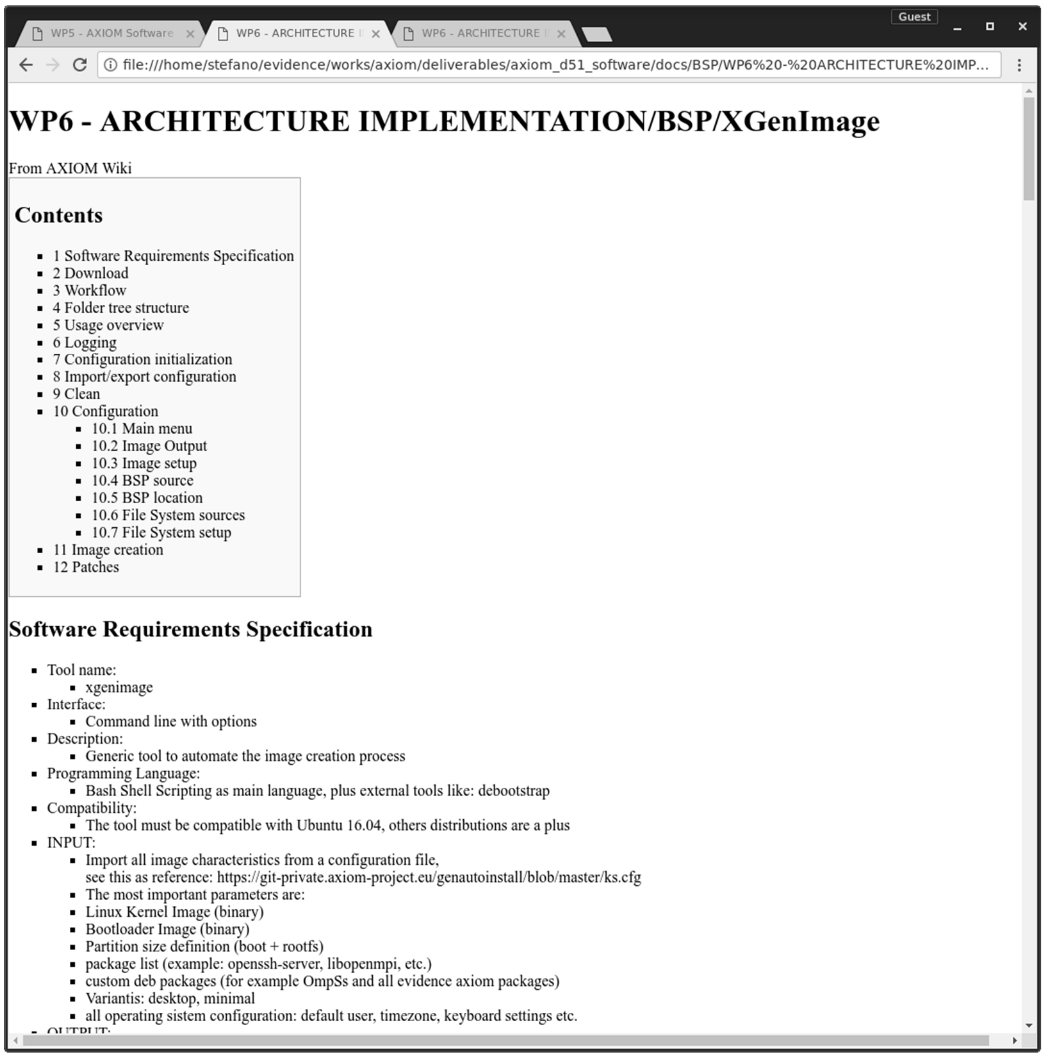
Download Installation Package

The actual version of Petalinux Tool is the 2016.3 one. You can download PetaLinux installation package directly from Xilinx Downloads page (https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/archive.html).
(N.B. The download require autetication. So please, if needed, regist own account form Xilinx Account Page (https://www.xilinx.com/registration/create-account.html))

Download package: **petalinux 2016.03 (https://www.xilinx.com/member/forms/download/xef.html?filename=petalinux-v2016.3-final-installer.run&akdm=1)**

(**N.B.** The used Petalinux installation package is also available into AXIOM server: downlaod link (https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTATION/BSP/ComponentsAndSources) - requires user access -)

- https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTA-TION/BSP/XGenImage
  For the instructions on how to use the XGENIMAGE Tool


WP5 - AXIOM Software
WP6 - ARCHITECTURE
WP6 - ARCHITECTURE
Guest
file:///home/stefano/evidence/works/axiom/deliverables/axiom_d51_software/docs/BSP/WP6%20-%20ARCHITECTURE%20IMP...

WP6 - ARCHITECTURE IMPLEMENTATION/BSP/XGenImage

From AXIOM Wiki

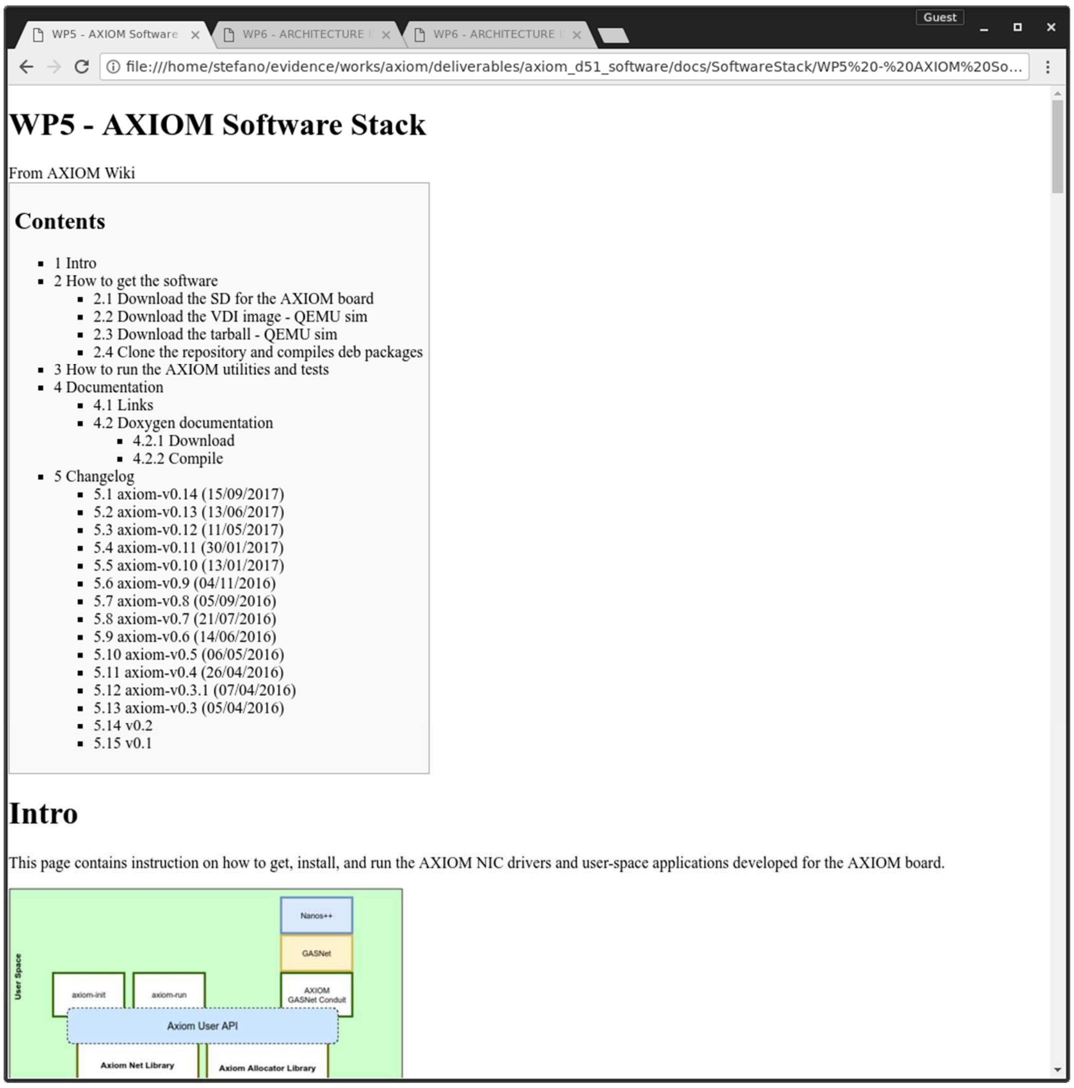
Contents

- 1 Software Requirements Specification
- 2 Download
- 3 Workflow
- 4 Folder tree structure
- 5 Usage overview
- 6 Logging
- 7 Configuration initialization
- 8 Import/export configuration
- 9 Clean
- 10 Configuration
  - 10.1 Main menu
  - 10.2 Image Output
  - 10.3 Image setup
  - 10.4 BSP source
  - 10.5 BSP location
  - 10.6 File System sources
  - 10.7 File System setup
- 11 Image creation
- 12 Patches

Software Requirements Specification

- Tool name:
  - xgenimage
- Interface:
  - Command line with options
- Description:
  - Generic tool to automate the image creation process
- Programming Language:
  - Bash Shell Scripting as main language, plus external tools like: debootstrap
- Compatibility:
  - The tool must be compatible with Ubuntu 16.04, others distributions are a plus
- INPUT:
  - Import all image characteristics from a configuration file, see this as reference: https://git-private.axiom-project.eu/genautoinstall/blob/master/ks.cfg
  - The most important parameters are:
  - Linux Kernel Image (binary)
  - Bootloader Image (binary)
  - Partition size definition (boot + rootfs)
  - package list (example: openssh-server, libopenmpi, etc.)
  - custom deb packages (for example OmpSs and all evidence axiom packages)
  - Variantis: desktop, minimal
  - all operating sistem configuration: default user, timezone, keyboard settings etc.

- https://wiki.axiom-project.eu/index.php/WP5_-_AXIOM_Software_Stack
  For the instructions on how to use the AXIOM software stack.


WP5 - AXIOM Software
WP6 - ARCHITECTURE
WP6 - ARCHITECTURE
Guest
file:///home/stefano/evidence/works/axiom/deliverables/axiom_d51_software/docs/SoftwareStack/WP5%20-%20AXIOM%20So...

WP5 - AXIOM Software Stack

From AXIOM Wiki

Contents

- 1 Intro
- 2 How to get the software
  - 2.1 Download the SD for the AXIOM board
  - 2.2 Download the VDI image - QEMU sim
  - 2.3 Download the tarball - QEMU sim
  - 2.4 Clone the repository and compiles deb packages
- 3 How to run the AXIOM utilities and tests
- 4 Documentation
  - 4.1 Links
  - 4.2 Doxygen documentation
    - 4.2.1 Download
    - 4.2.2 Compile
- 5 Changelog
  - 5.1 axiom-v0.14 (15/09/2017)
  - 5.2 axiom-v0.13 (13/06/2017)
  - 5.3 axiom-v0.12 (11/05/2017)
  - 5.4 axiom-v0.11 (30/01/2017)
  - 5.5 axiom-v0.10 (13/01/2017)
  - 5.6 axiom-v0.9 (04/11/2016)
  - 5.7 axiom-v0.8 (05/09/2016)
  - 5.8 axiom-v0.7 (21/07/2016)
  - 5.9 axiom-v0.6 (14/06/2016)
  - 5.10 axiom-v0.5 (06/05/2016)
  - 5.11 axiom-v0.4 (26/04/2016)
  - 5.12 axiom-v0.3.1 (07/04/2016)
  - 5.13 axiom-v0.3 (05/04/2016)
  - 5.14 v0.2
  - 5.15 v0.1

Intro

This page contains instruction on how to get, install, and run the AXIOM NIC drivers and user-space applications developed for the AXIOM board.

A copy of the main Wiki documents has been included in the software package which is part of this delivery.

We show below the most important files available in text form. These files are written in Markdown language to be shown in a web-based Git repository manager. Instead the Wiki pages previously described are available in HTML form in the software package to be easily consulted.

- `axiom-evi_README-v0.14.txt`

This file explains how to compile the AXIOM Software Stack and generate DEBs package for the AXIOM board.

```
                            AXIOM Software Stack
===============================================================================

This repository contains the following git sub-modules:

 * axiom-allocator
    + Implementation of the three level AXIOM allocator
 * axiom-evi-allocator-drv
    + Implementation of the memory device to handle virtual to physical memory mapping
 * axiom-evi-allocator-lib
    + Implementation of 3rd level software allocator based on LMM
 * axiom-evi-apps
```

~~~
      + Implementation of AXIOM application and daemons (axiom-init, axiom-run, etc.)
   * 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-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

## 1. Prerequisite to build the Software Stack

For the last release of the requested software see Axiom wiki.
```
https://wiki.axiom-project.eu/index.php/WP5_-_AXIOM_Software_Stack
https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTATION/BSP/Compo-
nentsAndSources
```

### 1.1 Axiom file-system

You must have a copy of the Axiom file-system.
```
$ wget -c https://upload.axiom-project.eu/uploads/WP5.1/filesystems/aarch64/zynq-ubuntu-
minimal_16.04-xenial_v1.3.tar.gz
$ mkdir zynq-ubuntu-minimal_16.04-xenial_v1.3
$ ln -s zynq-ubuntu-minimal_16.04-xenial_v1.3 zynq-rootfs
$ cd zynq-ubuntu-minimal_16.04-xenial_v1.3
$ tar xvf ../zynq-ubuntu-minimal_16.04-xenial_v1.3.tar.gz
```

### 1.2  Linaro

You must have the Linaro toolchain for AARCH64.
The directory of the Linaro toolchain must have a specific structure and must have
some link as show below.
You should use these linaro packages:
  * gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz
  * runtime-gcc-linaro-5.3.1-2016.05-aarch64-linux-gnu.tar.xz
  * sysroot-glibc-linaro-2.21-2016.05-aarch64-linux-gnu.tar.xz

```
$ mkdir linaro
$ cd linaro
$ wget -c  https://releases.linaro.org/components/toolchain/binaries/5.3-2016.05/aarch64-
linux-gnu/gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz
$ wget -c  https://releases.linaro.org/components/toolchain/binaries/5.3-2016.05/aarch64-
linux-gnu/runtime-gcc-linaro-5.3.1-2016.05-aarch64-linux-gnu.tar.xz
$ wget -c  https://releases.linaro.org/components/toolchain/binaries/5.3-2016.05/aarch64-
linux-gnu/sysroot-glibc-linaro-2.21-2016.05-aarch64-linux-gnu.tar.xz
$ tar xJvf gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz
$ tar xJvf runtime-gcc-linaro-5.3.1-2016.05-aarch64-linux-gnu.tar.xz
$ tar xJvf sysroot-glibc-linaro-2.21-2016.05-aarch64-linux-gnu.tar.xz
$ ln -s gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu host
$ ln -s sysroot-glibc-linaro-2.21-2016.05-aarch64-linux-gnu sysroot
$ ln -s . host/usr
$ ln -s /usr/bin/pkg-config host/bin/aarch64-linux-gnu-pkg-config
$ cd ..
```

### 1.3 Petalinux

You should have installed Petalinux.
```
$ wget -c https://upload.axiom-project.eu/uploads/WP5.1/tools/petalinux-v2016.3-final-
installer.run
$ ./petalinux-v2016.3-final-installer.run
$ ln -s petalinux-v2016.3-final petalinux

```

### 1.4 Axiom BSP
~~~

~~~
You must have installed the last Axiom BSP.
```
$ mkdir bsp
$ cd bsp
$ wget -c https://upload.axiom-project.eu/uploads/WP5.1/BSP_pre-built/AXIOM-ZU9EG-BSP-
2016.3_v1.6.bsp
$ mv AXIOM-ZU9EG-BSP-2016.3_v1.6.bsp AXIOM-ZU9EG-BSP-2016.3_v1.6.bsp.tar.gz
$ tar xzvf  AXIOM-ZU9EG-BSP-2016.3_v1.6.bsp.tar.gz
$ mv AXIOM-ZU9EG-2016.3 AXIOM-ZU9EG-BSP-2016.3_v1.6
$ ln -s AXIOM-ZU9EG-BSP-2016.3_v1.6 AXIOM-ZU9EG-2016.3
$ cd ..
```

### 1.5 Compile the kernel using Petalinux

You must compile the kernel.
```
$ source petalinux/settings.sh
$ cd bsp/AXIOM-ZU9EG-2016.3
$ petalinux-config --verbose -c kernel
$ petalinux-build --verbose
$ petalinux-package --boot
```
**WARNING:** sourcing 'petalinux/settings.sh' inserts into the environment a toolchain
that does not work to build the Axiom software stack. You must clean the environment to
build the Axiom software stack.

**WARNING:** petalinux suggest to use bash as default system shell
```
$ sudo ln -sf /bin/bash /bin/sh
```

## 2. Download the code

### 2.1 Clone the repository
```
$ git clone https://git.axiom-project.eu/axiom-evi
$ cd axiom-evi
```
**NOTE:**
Can be useful to add the following line to ~/.ssh/config
```
Host git.axiom-project.eu
     HostName git.axiom-project.eu
     Port 22
     User YOUR_AXIOM_PROJECT_NAME

```
and to add some lines to ~/.gitconfig
```
[user]
        name = YOUR_NAME
        email = YOUR_EMAIL
[http]
        sslverify = false

[credential]
        helper = store

```

### 2.2 Init the sub-modules (may take awhile)
```
$ ./scripts/submodules_update.sh
```

## 3. AXIOM Software Stack compilation

This section explains how to configure and compile the AXIOM sub-modules.

**Requirements:**

Note that some packages required for a build can be missing.
For example into a Ubuntu 16.04 LTS you should have installed the following packages:
```
~~~

~~~
$ sudo apt-get install build-essential git libssl-dev libncurses5-dev gawk bison wget
$ sudo apt-get install flex bc pkg-config libglib2.0-dev libfdt-dev libpixman-1-dev
zlib1g-dev
$ sudo apt-get install linux-tools-generic linux-headers-generic autoconf automake
gfortran libxml2-dev
$ sudo apt-get install libtool-bin libsqlite3-dev gperf debhelper fakeroot fakechroot
qemu-user-static
$ sudo apt-get install automake autoconf libtool

```

### 3.1 Prepare the system

Change the environment variables into ```axiom-evi/settings.sh```.

* PETALINUX
    + You must have installed PetaLinux 2016.3; this must point to the installation
directory
    + (default ```$HOME/petalinux```)

* AXIOMBSP
    + You must have installed SECO Petalinux project BSP; this must point to the BSP
directory
    + (default ```$HOME/bsp/AXIOM-ZU9EG-2016.3```)

* LINARO
    + You must have a linaro toolchain; this must point to the linaro directory
    + (default $HOME/linaro)
    + You should use these linaro packages:
        * gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz
        * runtime-gcc-linaro-5.3.1-2016.05-aarch64-linux-gnu.tar.xz
        * sysroot-glibc-linaro-2.21-2016.05-aarch64-linux-gnu.tar.xz
    + The linaro installation directory must have the following layout:
```
            ${LINARO}/
            ${LINARO}/runtime-gcc-linaro-5.3.1-2016.05-aarch64-linux-gnu
            ${LINARO}/sysroot-glibc-linaro-2.21-2016.05-aarch64-linux-gnu
            ${LINARO}/gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu
            ${LINARO}/host (link to gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu)
            ${LINARO}/sysroot (link to sysroot-glibc-linaro-2.21-2016.05-aarch64-linux-
gnu)
```

    + After the installation you must create a symbolic link "usr" to linaro/host
directory:
```
            cd ${LINARO}/host
            ln -s . usr
```

    + Then you must create a link "aarcha64-linux-gnu-pkg-config" to pkg-config:
```
            cd ${LINARO}/host/usr/bin
            ln -s /usr/bin/pkg-config aarcha64-linux-gnu-pkg-config
```

* ROOTFS
    + You must have extracted the Seco Ubuntu file-system image; this must point to the
installation directory
    + (default $HOME/zynq-rootfs)
    + Usually $HOME/zynq-rootfs is a link to $HOME/zynq-ubuntu-minimal_16.04-xenial_v1.3

    + NOTE: in zynq-rootfs/usr/lib/aarch64-linux-gnu there are .so with absolute link.
    Extrae try to use libdl.so (link to /lib/aarch64-linux-gnu/libdl.so.2)
    To fix this issue:
```
        cd zynq-rootfs/usr/lib/aarch64-linux-gnu
        rm libdl.so
        ln -s ../../../lib/aarch64-linux-gnu/libdl.so.2 libdl.so
```

* ROOTFSARCH
     + You must have a copy of the rootfs archive
     + (default $HOME/zynq-ubuntu-minimal_16.04-xenial_v1.3.tar.gz)
~~~

~~~
### 3.2 Compile the system

#### 3.2.1 Include settings file

To compile the system you must include the settings.sh into your environment, before to
use the Makefile

```
    $ source axiom-evi/settings.sh
```

**Note** that the environment must be clean (i.e. you can not include the petalinux's
settings)

#### 3.2.2 Define ```make``` variables

If you go into the ```scripts``` directory and do the command ```make``` for the first
time, an help will be show:
```
    $ cd scripts
    $ make
    configure.mk:113: *** Command line environment variables not defined!
    You must define some variable on the command line (for example "make MODE=aarch64"):
    * MODE (required)
       aarch64: use linaro to cross-compile for arm 64bit
       x86: use native x86 kernel on rootfs
    * DISABLE_INSTR (optional, used during libraries compilation)
       0: compile EXTRAE and nic user library with instrumentation support (default)
       1: disable EXTRAE compilation and nic user library instrumentation
    * KVERSION (optional, used to compile X86 in a chroot)
       you must defined this if you are compiling for X86 from a chroot and must be
       set to the kernel version (otherwise a "uname -r" is used using the chroot host
       kernel version)
    * DFLAGS (optional, other gcc parameters to add during compilation
    Note that:
    MODE and DISABLE_INSTR must defined during "make configure" and never changed.
    KVERSION should be defined only for X86 chroot compilation during "make configure".
).  Stop.

```

#### 3.2.3 Configure the system

```
    $ make MODE=aarch64 DISABLE_INSTR=0 configure
```

you can change parameters as specified in the 3.2.2 section

#### 3.2.4 Compile the system

```
    $ make all
```


#### 3.2.5 Generate DEB packages

```
    $ make deb
```
The debian packages are generated into the ```axiom-evi/debs``` directory.

```
    $ make deb-tests
```
Generate the tests debian packages.


### 3.3 Handle temp overlay rootfs

During the compilation, we need some overlay rootfs to build the packages.
To umount these directories:

```
~~~

~~~~
    $ make umount-temp
```

After the first compilation of all packages, if you want to compile in the sub-module
directory,
you must mount these overlay roots:
```
    $ make mount-temp
```
**Note** that a temp overlay rootfs is mounted under ~/axiom-evi/rootfs-tmp and is used
to install the sub-projects applications/libraries; this rootfs must be mounted if you use
the make command from the sub-projects; this can be accomplished using the following
command from the ~/axiom-evi/scripts directory


### 3.4 Update the sub-modules

The best way to compile a new release is to clear the previous compilation to be sure to
have a clean state and update each sub-modules.

```
    $ cd axiom-evi/scripts
    $ make mrproper
    $ ./submodules_pull.sh
    $ make MODE=XXX DISABLE_INSTR=XXX configure
    $ make all
```

### 3.4 Note for x86 compilation

* the user that compile the software must be into the sudoers file (but it is not
necessary to use a "sudo make .....")
* the system shell should be bash, so there must be a link from /bin/sh to bash
* the linux-tools-generic and linux-headers-generic packages must be installed

## 4. Note to compile an AXIOM application

### 4.1 Standalone application

You must link your application using the supplied linker script.
(This script include some entry point requested by the axiom memory manager).
You should add something like this into the makefile:
```
LDFLAGS += -Wl,-T~/axiom-evi/axiom-evi-allocator-lib/ldscript/aarch64/xs_map64.lds
```
or, if you are compiling on the board,
```
LDFLAGS += -Wl,-T/usr/axiom-evi-allocator-lib/xs_map64.lds
```
You could also use pkg-config:
```
pkg-config --libs evi_lmm
```

### 4.2 OMPSS@Cluster application

To compile a OMPSS@Cluster application you should use these flags for mercurium
```
MCCFLAGS := --ompss --cluster
```
during compilation. You must use the following flags during the link phase
```
LDFLAGS += --rdynamic -Wl,-T~/axiom-evi/axiom-evi-allocator-
lib/ldscript/aarch64/xs_map64.lds
```
**NOTE** The ```--rdynamic``` flag is required otherwise the symbols from the supplied
linker script will be discharged.

## 5 Note

### 5.1 OMPSS@Cluster

#### 5.1.1 Command line options
~~~~

~~~~
A new command-line options has been added: ```--use-my-cpus-number``` and can be zero
(default) or one.
It is used to force the use of ```--smp-cpus``` parameters because, otherwise, this
parameter can be set to a value greater than the number of cores of the microprocessor.

Example:
```
$ export NX_ARGS="--num-sockets=1 --cpus-per-socket=4 --smp-cpus=4"
```
this is the default for an AXIOM board if no parameter are passed. OMPSS create 4 physical
thread and assign each to a core of the microprocessor then assign 4 working thread to
these physical threads.

```
$ export NX_ARGS="--num-sockets=1 --cpus-per-socket=5 --smp-cpus=5 --use-my-cpu-number=1"
```
this create one more physical thread that share a core of the microprocessor and use it
for a working thread.

```
$ export NX_ARGS="--num-sockets=1 --cpus-per-socket=5 --smp-cpus=5 --use-my-cpu-number=1 -
-cluster --cluster-network=axiom --cluster-smp-presend=4"
```
this create a physical thread for communication that share a core with 1 of the 4  working
thread on each board.

#### 5.1.2 Scheduling policy

When you run a OMPSS@Cluster application you can define some environment variable to
change
the scheduling policy of the system threads: ```AXIOM_WRK_PARAMS``` (for the working
threads) and ```AXIOM_COM_PARAMS``` (for the communication threads)

The general form of the assignment is:
```
schedpol,par0,par1,par2
```
in which ```par0```, ```par1``` and ```par2``` are optional and depend on  ```schedpol```.

```schedpol``` can be:

* OTHER using the default linux policy
    + ```par0``` is the nice value (default to 0)
* FIFO using the FiFo real-time policy
    + ```par0``` is the priority (default to 1)
* RR using the RoundRobin real-time policy
    + ```par0``` if the priority (default to 1)
* BATCH using Batch policy, no parameters
* IDLE using the Idle policy, no parameters
* DEADLINE using the Deadline scheduling policy
    + ```par0``` is the execution time
    + ```par1``` is the deadline time
    + ```par2``` is the period time

    the time is specified using a units:

    + ```us``` micro-seconds
    + ```ms``` milli-seconds
    +  seconds otherwise

    the ```par0``` can also be specified using ```%``` of the period time

Examples:
```
$ export AXIOM_WRK_PARAMS=RR,2
$ export AXIOM_COM_PARRAMS=DEADLINE,10ms,80ms,100ms
```
or
```
$ export AXIOM_WRK_PARAMS=OTHER,10
$ export AXIOM_COM_PARRAMS=DEADLINE,2%,800us,10ms
```

## 6 Example of build ARM64 debs on a fresh Ubuntu 16.04 LTS host PC
~~~~

~~~~
```
    # Install missing packages in the host PC

    $ sudo apt-get install build-essential git libssl-dev libncurses5-dev gawk  \
      bison wget flex bc pkg-config libglib2.0-dev libfdt-dev libpixman-1-dev   \
      zlib1g-dev linux-tools-generic linux-headers-generic autoconf automake    \
      gfortran libxml2-dev vim libtool-bin libsqlite3-dev gperf debhelper       \
      fakeroot fakechroot qemu-user-static

    # Petalinux 2016.3 installation

    $ mkdir ~/petalinux
    $ wget --no-check-certificate --user=YOUR_USERNAME --ask-password
https://upload.axiom-project.eu/uploads/WP5.1/tools/petalinux-v2016.3-final-installer.run
    $ chmod +x petalinux-v2016.3-final-installer.run
    $ ./petalinux-v2016.3-final-installer.run ~/petalinux

    # Linaro 5.3.1-2016.05 installation

    $ mkdir ~/linaro
    $ cd ~/linaro
    $ wget https://releases.linaro.org/components/toolchain/binaries/5.3-2016.05/aarch64-
linux-gnu/gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz
    $ wget https://releases.linaro.org/components/toolchain/binaries/5.3-2016.05/aarch64-
linux-gnu/sysroot-glibc-linaro-2.21-2016.05-aarch64-linux-gnu.tar.xz
    $ wget https://releases.linaro.org/components/toolchain/binaries/5.3-2016.05/aarch64-
linux-gnu/runtime-gcc-linaro-5.3.1-2016.05-aarch64-linux-gnu.tar.xz
    $ tar xf https://releases.linaro.org/components/toolchain/binaries/5.3-
2016.05/aarch64-linux-gnu/gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz
    $ tar xf https://releases.linaro.org/components/toolchain/binaries/5.3-
2016.05/aarch64-linux-gnu/sysroot-glibc-linaro-2.21-2016.05-aarch64-linux-gnu.tar.xz
    $ tar xf https://releases.linaro.org/components/toolchain/binaries/5.3-
2016.05/aarch64-linux-gnu/runtime-gcc-linaro-5.3.1-2016.05-aarch64-linux-gnu.tar.xz
    $ ln -s gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu host
    $ ln -s sysroot-glibc-linaro-2.21-2016.05-aarch64-linux-gnu sysroot
    $ cd ~/linaro/host
    $ ln -s . usr
    $ cd ~/linaro/host/usr/bin
    $ ln -s /usr/bin/pkg-config aarcha64-linux-gnu-pkg-config

    # Download AXIOM root file-system

    $ mkdir ~/zynq-rootfs
    $ wget --no-check-certificate --user=YOUR_USERNAME --ask-password
https://upload.axiom-project.eu/uploads/WP5.1/filesystems/aarch64/zynq-ubuntu-
minimal_16.04-xenial_v1.3.tar.gz
    $ tar xfz zynq-ubuntu-minimal_16.04-xenial_v1.3.tar.gz -C zynq-rootfs/
    $ cd zynq-rootfs/usr/lib/aarch64-linux-gnu
    $ rm libdl.so
    $ ln -s ../../../lib/aarch64-linux-gnu/libdl.so.2 libdl.so

    # Download AXIOM Petalinux project BSP

    $ mkdir ~/bsp
    $ cd ~/bsp
    $ git config --global http.sslVerify false
    $ git clone https://git-private.axiom-project.eu/AXIOM-ZU9EG-2016.3/
    $ git checkout axiom_ni    # for now the FORTH bit-stream is not merge in the master
branch
    $ source ~/petalinux/settings.sh
    $ cd ~/bsp/AXIOM-ZU9EG-2016.3/
    $ petalinux-build

    # AXIOM Software Stack

    $ # note: you must use another shell or clean the setting of petalinux
    $ su - $(whoami)
    $ git clone https://git.axiom-project.eu/axiom-evi
    $ cd axiom-evi
    $ ./scripts/submodules_update.sh
    $ source settings.sh
    $ cd scripts/
    $ make MODE=aarch64 DISABLE_INSTR=0 configure
    $ make all
    $ make deb
~~~~

~~~
```
~~~

- `axiom-evi-apps_README-v0.14.txt`
  This file explains how to run AXIOM applications and utilities useful for the cluster setup and network troubleshooting.

~~~
                            AXIOM NIC applications
================================================================================

This repository contains the following user space application:

 * axiom-init
    + axiom-init is the main daemon running on all nodes of the cluster.
    It runs the discovery algorithm, sets the routing table and handles control messages.
    It will be run automatically by the system or through axiom-startup.sh script.
```
        example:

        # in n-1 nodes
        axiom-init &

        # in the master node
        axiom-init -m &
```
 * axiom-ethtap
    + axiom-ethtap is another deamon that can run on each nodes in the cluster
    to map the ethernet frames over AXIOM messages.
    It creates a virtual ethernet device on top of AXIOM device.
    It will be run automatically by the system or through axiom-startup.sh script.
```
        example:
        # start axiom-ethtap with 4 threads
        axiom-ethtap -n 4
```
 * axiom-info
    + print informations (node-id, interfaces, routing, etc.) about AXIOM NIC
```
        example:
        # print all information
        axiom-info
```
```
        # print nodeid
        axiom-info -n
```
```
        # print routing table
        axiom-info -r
```
 * axiom-whoami
     + print the node-id set after the discovery phase
```
        example:
        axiom-whoami
```
 * axiom-ping
    + estimate the round trip time (RTT) between two nodes
```
        example:
        # Estimate RTT with target node 2. Send a ping message every 0.5 seconds.
        axiom-ping -d 2 -i 0.5
```
```
        # Estimante RTT with target node 2. Send 10 ping message every 0.2 seconds.
        axiom-ping -d 2 -c 10 -i 0.2
```
 * axiom-netperf
    + estimate the throughput between two nodes
```
        example:
        # start server on one node with 8 threads
        axiom-netperf -s -n 8
~~~

~~~~
        # Estimate the throughput with targat node 3, sending 512 KBytes of data
        # using RAW messages.
        axiom-netperf -d 3 -l 512K -t raw
```
```
        # Estimate the throughput with targat node 3, sending 2 MBytes of data.
        # using LONG messages and 2 sending threads.
        axiom-netperf -d 3 -l 2M -t long -n 2
```
 * axiom-traceroute
    + print the hops needed to reach a specified target node
```
        example:
        # Print the hops needed to reach the node 1
        axiom-traceroute -d 1
```
 * axiom-[send|recv]
     + send/receive Axiom RAW/LONG data to/from a node
```
        examples:
        # receive RAW message
        axiom-recv -t raw
```
```
        # receive LONG message
        axiom-recv -t long
```
```
        # send RAW data to node (node_id=4) with a specified payload
        axiom-send -t raw -d 4   112 43 0 57 33
```
```
        # send RAW data to a neighbour directly connected on a local
        # interface (if_id=1) with a specified payload
        axiom-send -t raw -d 1 --neighbour   67 52 1 0 1 0
```
```
        # send LONG data to node (node_id=0) with a specified payload
        axiom-send -t long -d 0   112 43 0 57 33
```
 * axiom-rdma
     + use RDMA features (remote write, remote read) of Axiom NIC
```
        examples:
        # dump local RDMA zone (starting from offset 0 to 1023)
        axiom-rdma -m d -s 1k
```
```
        # store in the local RDMA zone (starting from offset 1024 to 2043)
        # the bytes ( 1, 2, 3, 4) repeated many times to fill the size
        # specified
        axiom-rdma -m s -o 1k -s 1k 1 2 3 4
```
```
        # write 3072 bytes in the RDMA zone of node 3 (offset 2048) reading
        # from the local RDMA zone (offset 1024)
        axiom-rdma -m w -n 3 -o 1k -O 2k -s 3k
```
```
        # read 4096 bytes in the RDMA zone of node 2 (offset 7168) reading
        # from the local RDMA zone (offset 0)
        axiom-rdma -m r -n 2 -O 7k -s 4k
```
 * axiom-run
    + spawn application on multiple nodes
```
        # run ls on remote node 1 (-n) and redirect (-r) the stdin/stdout/stderr
        axiom-run -n 1 -r ls
```
```
        # run ls on all nodes, redirect (-r) the stdin/stdout/stderr and print
        # the node id in each line (-i)
        axiom-run -r -i ls
```
~~~~

References

1. AXIOM NIC Datasheet, axiom_nic_datasheet-v1.0.pdf (https://download.axiom-project.eu/?dir=RUNTIME)
2. Luigi Rizzo, Revisiting network I/O APIs: The Netmap Framework, Communications of the ACM, March 2012.
3. AXIOM API, Doxygen generated file, axiom-v1.0-doxygen.pdf axiom-v1.0-doxygen-html.tgz (https://download.axiom-project.eu/?dir=RUNTIME)
4. https://store.arduino.cc/arduino-4-relays-shield
5. https://opencores.org/project,avr_core
6. https://wiki.axiom-project.eu/index.php/WP6_-_ARCHITECTURE_IMPLEMENTATION/BSP/ComponentsAndSources
7. POSIX wrapper for the network discovery algorithm: https://git.axiom-project.eu/axiom-evi-apps/tree/master/axiom-init/axiom-discovery/test/
8. AXIOM Git Repository for the PetalLinux BSP: https://git-private.axiom-project.eu/AXIOM-ZU9EG-2016.3/
9. R. Giorgi, “Transactional memory on a dataflow architecture for accelerating Haskell,” WSEAS Trans. Computers, vol. 14, pp. 794–805, 2015.
10. R. Giorgi and P. Faraboschi, “An introduction to DF-Threads and their execution model,” in IEEE MPP, Paris, France, Oct. 2014, pp. 60–65.
11. 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.
12. 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.
13. 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.
14. 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.
15. R. Giorgi, "Exploring Future Many-Core Architectures: The TERAFLUX Evaluation Framework", Elsevier, 2017, pp. 33-72.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. R. Giorgi, "Accelerating Haskell on a Dataflow Architecture: a case study including Transactional Memory", Proc. Int.l Conf. on Computer Engineering and Applications (CEA), Dubai, UAE, Feb. 2015, pp. 91-100.
23. N. Ho, A. Mondelli, A. Scionti, M. Solinas, A. Portero, R. Giorgi, "Enhancing an x86_64 Multi-Core Architecture with Data-Flow Execution Support", ACM Computing Frontiers, May 2015, pp. 1-2.
24. N. Ho, A. Portero, M. Solinas, A. Scionti, A. Mondelli, P. Faraboschi, R. Giorgi, "Simulating a Multi-core x86-64 Architecture with Hardware ISA Extension Supporting a Data-Flow Execution Model", IEEE Proc. AIMS-2014, Madrid, Spain, Nov. 2014, pp. 264-269.
25. A. Portero, A. Scionti, Z. Yu, P. Faraboschi, C. Concatto, L. Carro, A. Garbade, S. Weis, T. Ungerer, R. Giorgi, "Simulating the Future kilo-x86-64 core Processors and their Infrastructure", 45th Annual Simulation Symp. (ANSS12), Orlando, FL, Mar 2012, pp. 62-67.
26. 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.
27. 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.
28. 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.

29. L. Verdoscia, R. Vaccaro, R. Giorgi, "A matrix multiplier case study for an evaluation of a configurable Dataflow-Machine", ACM CF'15 - LP-EMS, May 2015, pp. 1-6. DOI:10.1145/2742854.2747287
30. G. Burresi, R. Giorgi, "A Field Experience for a Vehicle Recognition System using Magnetic Sensors", IEEE MECO 2015, Budva, Montenegro, June 2015, pp. 178-181. DOI:10.1109/MECO.2015.7181897,
31. A. Mondelli, N. Ho, A. Scionti, M. Solinas, A. Portero, R. Giorgi, "Dataflow Support in x86-64 Multicore Architectures through Small Hardware Extensions", IEEE Proc. DSD, August 2015, pp. 526-529.. DOI: 10.1109/DSD.2015.62
32. C. Alvarez, E. Ayguade, J. Bueno, A. Filgueras, D. Jimenez-Gonzalez, X. Martorell, N. Navarro, D. Theodoropoulos, D. Pnevmatikatos, C. Scordino, P. Gai, C. Segura, C. Fernandez, D. Oro, J. Saeta, P. Passera, A. Pomella, A. Rizzo, R. Giorgi, "The AXIOM Software Layers", IEEE Proc. 18th EUROMICRO-DSD, Aug. 2015, pp. 117-124. DOI:10.1109/DSD.2015.52
33. D. Jiménez-González, C. Álvarez, A. Filgueras, X. Martorell, J. Langer, J. Noguera, K. Vissers. “Coarse-grain performance estimator for heterogeneous parallel computing architectures like Zynq all-programmable SoC”. arXiv preprint arXiv:1508.06830.
34. R. Giorgi, A. Scionti, "A scalable thread scheduling co-processor based on data-flow principles", ELSEVIER Future Generation Computer Systems, Amsterdam, Netherlands, vol. 53, Dec. 2015, pp. 100-108. DOI:10.1016/j.future.2014.12.014
35. 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. DOI:10.1145/2903150.2906829
36. C. Alvarez, E. Ayguade, J. Bosch, J. Bueno, A. Cherkashin, A. Filgueras, D. Jiminez-Gonzalez, X. Martorell, N. Navarro, M. Vidal, D. Theodoropoulos, D. Pnevmatikatos, D. Catani, D. Oro, C. Fernandez, C. Segura, J. Rodriguez, J. Hernando, C. Scordino, P. Gai, P. Passera, A. Pomella, N. Bettin, A. Rizzo, R. Giorgi, "The AXIOM Software Layers", ELSEVIER Microprocessors and Microsystems, vol. 47, Part B, 2016, pp. 262-277. DOI:10.1016/j.micpro.2016.07.002
37. 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. DOI:10.1109/DSD.2016.80
38. G. Llort, A. Filgueras, D. Jiménez-González, H. Servat, X. Teruel, E. Mercadal and J. Labarta. “The Secrets of the Accelerators Unveiled: Tracing Heterogeneous Executions Through OMPT”. In International Workshop on OpenMP (pp. 217-236). Springer, Cham. DOI: 10.1007/978-3-319-45550-1_16
39. 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. DOI:10.1007/978-3-319-42304-3_3
40. R. Giorgi, S. Mazumdar, S. Viola, P. Gai, S. Garzarella, B. Morelli, D. Pnevmatikatos, D. Theodoropoulos, C. Alvarez, E. Ayguade, J. Bueno, A. Filgueras, D. 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. ISSN: 1381-6551
41. M. Wagner, G. Llort, A. Filgueras, D. Jiménez-González, H. Servat, X. Teruel, and E. Ayguadé. “Monitoring Heterogeneous Applications with the OpenMP Tools Interface”. In Tools for High Performance Computing 2016 (pp. 41-57). Springer. DOI: 10.1007/978-3-319-56702-0_3
42. D. Theodoropoulos, S. Mazumdar, E. Ayguade, N. Bettin, J. Bueno, S. Ermini, A. Filgueras, D. Jimenez-Gonzalez, C. Alvarez Martinez, 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, 2017. DOI:10.1016/j.micpro.2017.05.018
43. A. Rizzo, F. Montefoschi, M. Caporali, A. Gisondi, G. Burresi, R. Giorgi, "Rapid Prototyping IoT Solutions Based on Machine Learning", Proc. European Conf. on Cognitive Ergonomics 2017, New York, NY, USA, 2017, pp. 4. DOI:10.1145/3121283.3121291
44. D. Theodoropoulos, D. Pnevmatikatos, S. Garzarella, P. Gai, A. Rizzo, R. Giorgi. “AXIOM: enabling parallel processing in cyber-physical systems.” Reconfigurable Computing Workshop, Lausanne, CH. Sep 2016. Pp.1-2.
45. J. Bosch Pons, “Asynchronous runtime for task-based dataflow programming models.” Jul. 2017. Master's Thesis. Universitat Politecnica de Catalunya.
46. S. Mazumdar and R. Giorgi. "A Survey on Hardware and Software Support for Thread Level Parallelism." arXiv preprint arXiv:1603.09274 (2016).
47. C. Scordino and B. Morelli. “Sharing memory in modern distributed applications”. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (pp. 1918-1921) ACM, April 2016.