**Towards EXtreme scale Technologies and Accelerators for euROhpc hw/Sw Supercomputing Applications for exascale**

# WP2 New accelerator designs exploiting mixed precision

## D2.1 Consolidated specs of accelerators IPs

Revised version

http://textarossa.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme, EuroHPC JU, grant agreement No 956831

# TEXTAROSSA

## Towards EXtreme scale Technologies and Accelerators for euROhpc hw/Sw Supercomputing Applications for exascale

**Grant Agreement No.: 956831**

**Deliverable: D8.4 External Advisory Board reports**

**Project Start Date**: 01/04/2021 **Duration**: 36 months
**Coordinator**: *AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE, L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE - ENEA, Italy*.

| | |
|---|---|
| **Deliverable No** | D8.4 |
| **WP No:** | WP8 |
| **WP Leader:** | ENEA |
| **Due date:** | M24 (March 31, 2023) |
| **Delivery date:** | 21/05/2023 |

**Dissemination Level:**

| | | |
|---|---|---|
| PU | Public | |
| PP | Restricted to other programme participants (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) | X |

DOCUMENT SUMMARY INFORMATION

| | |
|---|---|
| **Project title:** | Towards EXtreme scale Technologies and Accelerators for euROhpc hw/Sw Supercomputing Applications for exascale |
| **Short project name:** | TEXTAROSSA |
| **Project No:** | 956831 |
| **Call Identifier:** | H2020-JTI-EuroHPC-2019-1 |
| **Unit:** | EuroHPC |
| **Type of Action:** | EuroHPC - Research and Innovation Action (RIA) |
| **Start date of the project:** | 01/04/2021 |
| **Duration of the project:** | 36 months |
| **Project website:** | textarossa.eu |

## WP2 New accelerator designs exploiting mixed precision

| | | | | | | |
|---|---|---|---|---|---|---|
| **Deliverable number:** | D2.1 | | | | | |
| **Deliverable title:** | Consolidated specs of accelerators IPs | | | | | |
| **Due date:** | M12 | | | | | |
| **Actual submission date:** | 21/05/2023 | | | | | |
| **Editor:** | Sergio Saponara | | | | | |
| **Authors:** | S. Saponara, C. Alvarez, D. Jimenez, A. Lonardo, F. Lo Cicero, P. Cretaro, S. Di Matteo, F. Rossi, M. Cococcioni, M. Lo Gerfo | | | | | |
| **Work package:** | WP2 | | | | | |
| **Dissemination Level:** | Public | | | | | |
| **No. pages:** | 30 (37 revised) | | | | | |
| **Authorized (date):** | 31/05/2022 (03/05/2023 revised) | | | | | |
| **Responsible person:** | Sergio Saponara | | | | | |
| **Status:** | Plan | Draft | Working | Final | [✓ Submitted] | Approved |

**Revision history:**

| Version | Date | Author | Comment |
|---|---|---|---|
| **0.1** | 2022-05-10 | S. Saponara | Draft structure |
| **0.2** | 2022-05-27 | S. Saponara, C. Alvarez, D. Jimenez, A. Lonardo, F. Lo Cicero, P. Cretaro, S. Di Matteo, F. Rossi, M. Cococcioni, M. Lo Gerfo | Added contribution of CINI; INFN and BSC |
| **0.3** | 2022-05-28 | S. Saponara | Revised draft |
| **0.4** | 2022-05-30 | S. Saponara | Final |
| **0.5** | 2023-04-24 | F. Lo Cicero, A. Lonardo, C. Alvarez | Updated according to reviewers observations |
| **0.6** | 2023-04-30 | S. Saponara | Updated according to reviewers observations |

**Quality Control:**

| Checking process | Who | Date |
|---|---|---|
| **Checked by internal reviewer** | F. Magugliani | May 30th, 2022 |
| **Revised Checked by internal reviewer** | F. Magugliani (revised) | May 3rd, 2023 |
| **Checked by Tasks Leaders** | S. Saponara, A. Lonardo, C. Alvarez | May 31st, 2022 |
| **Revised Checked by Tasks Leaders** | S. Saponara, A. Lonardo, C. Alvarez (revised) | May 3rd, 2023 |
| **Checked by WP Leader** | S. Saponara | May 31st, 2022 |
| **Revised Checked by WP Leader** | S. Saponara (revised) | May 3rd, 2023 |
| **Checked by Project Coordinator** | M. Celino | May 31st, 2022 |
| **Revised Checked by Project Coordinator** | M. Celino (revised) | May 20th, 2023 |

## COPYRIGHT

## ACKNOWLEDGEMENTS

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement no 956831. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Italy, Germany, France, Spain, Poland.
Please see http://textarossa.eu for more information on the TEXTAROSSA project.
The partners in the project are AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE, L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE (ENEA), FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V. (FHG), CONSORZIO INTERUNIVERSITARIO NAZIONALE PER L'INFORMATICA (CINI), INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE (INRIA), BULL SAS (BULL), E4 COMPUTER ENGINEERING SPA (E4), BARCELONA SUPERCOMPUTING CENTER-CENTRO NACIONAL DE SUPERCOMPUTACION (BSC), INSTYTUT CHEMII BIOORGANICZNEJ POLSKIEJ AKADEMII NAUK (PSNC), ISTITUTO NAZIONALE DI FISICA NUCLEARE (INFN), CONSIGLIO NAZIONALE DELLE RICERCHE (CNR), IN QUATTRO SRL (in4). Linked third parties of CINI are POLITECNICO DI MILANO (CINI-POLIMI), Università di Torino (CINI-UNITO) and Università di Pisa (CINI-UNIPI); linked third party of INRIA is Université de Bordeaux; in-kind third party of ENEA is Consorzio CINECA (CINECA); in-kind third party of BSC is Universitat Politècnica de Catalunya (UPC).
The content of this document is the result of extensive discussions within the TEXTAROSSA © Consortium as a whole.

## DISCLAIMER

# Table of contents

# List of Acronyms

| Acronym | Definition |
|---|---|
| ALU | Arithmetic Logic Unit |
| ASIC | Application Specific Integrated Circuit |
| BSC | Barcelona Supercomputing Center |
| BIST | Built In Self Test |
| CINI | Consorzio Interuniversitario Nazionale per l'Informatica |
| CPU | Central Processing Unit |
| CTO | Chief Technical Officer |
| DNN | Deep Neural Network |
| EPI | European Processor Initiative |
| FP32 | Floating Point 32 bit |
| FPGA | Field Programmable Gate Array |
| FTS | Fast Task Scheduler |
| HE | Homomorphic Encryption |
| HLS | High Level Synthesis |
| HW | Hardware |
| HPC | High-Performance-Computing |
| IDV-E | Integrated Development Environment- E4 |
| INFN | Istituto Nazionale di Fisica Nucleare |
| IP | Intellectual Property |
| IPR | Intellectual Property Rights |
| KPI | Key Performance Indicators |
| XOF | eXtendable Output Function |
| PMB | Project Management Board |
| PPU | Posit Processing Unit |
| PQC | Post Quantum Cryptography |
| RISC | Reduced Instruction Set Computer |

| | |
|---|---|
| RLWE | Ring Learning With Errors |
| SEAL | Simple Encrypted Arithmetic Library |
| SW | Software |
| RLWE | Ring Learning With Errors |
| UART | Universal Asynchronous Receiver Transmitter interface |
| VHDL | VHSIC Hardware Description Language |
| VPU | Vector Processor Unit |

# Executive Summary

This document reports the activities done by TEXTAROSSA partners CINI (UNIPISA), INFN and BSC with reference to consolidated specifications of accelerator IPs in WP2, revised according to the comments received by the reviewers at the mid-term (M18) project review.

In order to assess the feasibility of some specifications, for the target IP preliminary architectures are also discussed and some preliminary design activities and synthesis, mainly in FPGA technology, have been carried out. This will allow also having an exploration of the design and performance space.

# 1. Introduction

This document reports the activities done by TEXTAROSSA partners CINI (UNIPISA), INFN and BSC with reference to consolidated specifications of accelerator IPs in WP2, revised according to the comments received by the reviewers at the mid-term (M18) project review.

In order to assess the feasibility of some specifications, preliminary architectures of the IPs are also shown and some preliminary design activities and synthesis, mainly in FPGA technology, have been done.

Particularly, Section 2 deals with specification and with preliminary design space and performance analysis of accelerator IPs using alternative data arithmetic vs. conventional floating point 32 bit (FP32). These accelerators aim at achieving data compression and efficient computation of DNN (Deep Neural Network).

Section 3 discusses the specifications of accelerators for innovative security services based on Post Quantum Cryptographic (PQC) techniques, that may be useful also for future evolution towards homomorphic encryption.

Sections 2 and 3 show how the proposed accelerators can be integrated with RISC-V computing core like the RSC-V 64B Ariane IP and the RISC-V with support of the Vector extension.

Section 4 refers to INFN Communication IP cores for low-latency inter-core and intra-core communications between HLS kernels.

Section 5 refers to BSC Fast Task Scheduling IP for high performance computing systems and OmpSs@FPGA framework.

Conclusions are drawn in Section 6.

# 2. Accelerators with alternative data arithmetic for AI computing and data compression

## 2.1. Specifications on IP macrocells with data arithmetic alternative to IEEE-754 FP

This Section focuses on the specifications about IP macrocells having an alternative data arithmetic vs. classic IEEE-754 [16] Floating Point (FP) types and complementary to other solutions already investigated in EPI1.

Moreover, this Section will refer to specifications on Posits arithmetic [14] considering also the relevant industrial exploitation impact.

The section will cover specifications on IP cores design and on IP core verification procedures, and also the KPIs will be highlighted.

**Specifications on alternative IEEE-754 FP data arithmetic and TEXTAROSSA complementarity to EPI**

The specifications about the research and development activities of accelerator IP cores having an alternative data arithmetic vs. classic IEEE-754 floating-point types are dictated by the following considerations:

- Growing constraints on memory utilization, power consumption, and I/O throughput have increasingly become limiting factors to the advancement of high performance computing (HPC) platforms and of the edge computing units needed, in a digital continuum scenario, by the users to access Cloud server and HPC applications. IEEE-754 floating-point types (32 bit for single-precision and 64 bits for double-precision) have been the de facto standard for floating-point number systems for decades, but the drawbacks of this numerical representation [14] leave much to be desired. Alternative representations are gaining traction, both in HPC and machine learning environments and many group in academia and industry are investigating alternatives.
- For example, commercial accelerators used in HPC applications like the GPUs from NVIDIA, support classic FP64 and FP32 data arithmetic plus alternative arithmetic: in the A100 GPU there is the support also for BF16 and FP16 (scaled versions on 16 bits), TF32, a custom format which uses the same 10-bit mantissa as FP16 to ensure accuracy while sporting the same range as FP32, thanks to using an 8-bit exponent, and INT8 and INT4 (i.e. integers at 8 and at 4 bits, respectively). In https://www.nvidia.com/en-us/data-center/a100/, for HPC applications formats like TF32 are suggested to be used, since TF32 can target similar accuracy than FP32 but is optimized for speed for NVIDIA GPUs.
- A mixed precision approach similar to what is proposed by NVIDIA is also adopted in the accelerators for the EPI SGA1:
  Bfloat16 and its conversion to integer (INT) is supported in https://www.european-processor-initiative.eu/dissemination-material/data-movement-reduction-for-dnn-accelerators-enabling-dynamic-quantization-through-an-efpga/; FP32 and FP64 are supported in the STX

accelerator, see https://www.european-processor-initiative.eu/wp-content/uploads/2022/06/Matheus-Cavalcante-@-RISC-V-Week.pdf

- The TEXTAROSSA project, according to the project proposal, aims at finding innovative solutions to computing problems and complementary to the research carried out in EPI SGA1 so that the results of TEXTAROSSA may enrich the output of EPI SGA1 and be an input for further development in EPI SGA2.

Summarizing the above points, one key specification for the IP cores to be developed in WP2 of TEXTAROSSA is *SPEC1: researching alternative data arithmetic to FP32, that is not limited to looking to FP16 (or BF16) or INT8/4.*

## Specifications on Posits and relevant industrial exploitation impact

To address the above SPEC1, a new data format that has been recently proposed in literature as a drop-in replacement for the IEEE-754 floating-point representation, but is still not adopted in EPI SGA1 accelerators like Atrevido/Avispado form Semidynamics, embedded FPGA from Menta or STX from Fraunhofer/ETHZ, is the Posit format.

To be noted that, at the state of art, the interest on Posit format has not been limited only to academia but involves also industry:

In Europe the French company Kalray has shown Interest on Posits.
In the paper https://link.springer.com/chapter/10.1007/978-3-031-09779-9_2 "A Posit8 Decompression Operator for Deep Neural Network Inference", Springer LNCS, volume 13253, July 2022, the CTO of the company B. De Dinechin et al. proposes a hardware operator to decompress Posit8 representations with exponent sizes 0, 1, 2, 3 to the IEEE 754 FP16 representation with the motivation to leverage the tensor units of the Kalray manycore processor which already supports FP16/FP32 matrix multiply-accumulate operations for deep learning inference. According to the paper, adding instructions to decompress Posit8 into FP16 numbers allows to further reduce the footprint of DNN parameters with an acceptable loss of accuracy or precision.

In Asia the company VividSparks has shown the industrial interest on Posits as reported in https://www.vivid-sparks.com/supports Posits as a promising alternative to traditional floating-point systems, both as a stand-alone replacement and combined in a mixed-precision environment.

A first standardization of Posits has been recently released, see https://www.posithub.org/posit_standard4.12.pdf

Development of the Posit type and its support in hardware is still ongoing, and research continues to explore the application of Posits in different domains, how to best implement the type in hardware, and where the type fit with other numerical representations.

Therefore, from the above considerations we have defined the specifications summarized in Table 2.1 for the research in TEXTAROSSA on IPs with innovative data arithmetic:

|  |  |
|---|---|
| SPEC1 | Researching alternative data arithmetic to FP32, which is not limited to looking at FP16 (or BF16) or INT8/4. |
| SPEC2 | Developing solutions based on Posit 8 and Posit 16, the most promising from already industrial interest from companies in Europe |
| SPEC3 | Focusing on computing problems including machine learning and DNN, mainly for inference |
| SPEC4 | Aiming at a computing accuracy similar to FP32 |
| SPEC5 | Design of an IP core for light support of Posit, i.e. capable with a limited complexity overhead to add the support of Posit to computing cores, compliant to RISC-V instruction set and already equipped with an integer ALU and a FPU. This IP will be called Light PPU (Posit Processing Unit) and will be developed in WP2-Task 2.3 |
| SPEC6 | Design of an IP core to give full support in hardware to all basic Posits arithmetic operations between at least 2 operands (addition, subtraction, multiplication, division); this IP should be interfaceable to RISC-V instruction set processors already equipped with an integer ALU. This IP will be called Full PPU (Posit Processing Unit) will be developed in WP2-Task 2.1 |
| SPEC7 | Since the complexity overhead of the Full PPU is expected to be higher than the Light PPU, as consequence the Full PPU should be integrated also in RISC-V processors without adding an FPU. Hence, to support compatibility with float computation the Full PPU should support also conversion from float to posit, computing in posit format and viceversa computing from posit to float. |
| SPEC8 | Light PPU in SPEC5 and Full PPU in SPEC6 should be interfaceable with RISC-V well known open cores like CVA6 (Ariane), that is a 64bit RISC-V core available from https://github.com/openhwgroup/cva6, and Ibex, that is a 32-bit RISC-V core available from https://github.com/lowRISC/ibex by using the possibility of custom opcodes offered by the RISC-V instruction set. Hence, Light and Full PPU operations will represent an extension of the instruction set rather than a separate coprocessor on the AXI bus. |
| SPEC9 | Textarossa aim at providing IP cores that implement a single computing lane working on 2 operands. A vectorized multi-lane, i.e. multi-operands, version of the Light and FULL PPU IPs is out of Textarossa scope, and may be achieved in future projects having the results of Textrossa WP2s as input |

Table 2.1: Specifications for accelerator IP cores with alternative data arithmetic vs. IEEE-754 FP types

### Specifications on IP cores verification procedures and performance KPIs

In order to verify the functionalities of the developed IPs, namely the Light PPU and the Full PPUs, TEXTAROSSA envisages to implement the hierarchical approach shown in Table 2.2 considering the following key performance indicators (KPI):

- Achievement of a computing accuracy similar to the use of FP32 but with a data compression factor (in storage and data transfer) higher than 2 for Light PPU vs a state-of-art FPU;

- Achievement of a computing accuracy similar to the use of FP32 and both a data compression factor (in storage and data transfer) higher than 2 and an increase efficiency computing speed/complexity-overhead vs. state-of-the-art FPU.

| | |
|---|---|
| Computing kernel | Implementation of basic computing kernels typical of DNNs: e.g. weighted multiplications and accumulation in GEMM: GEneral Matrix to Matrix Multiplication, non-linear activation function. |
| Mini-apps | Implementation of reference Neural Networks (e.g. LeNet) on reference benchmark data sets (e.g. MNIST, CIFAR, GTRSB) |
| Complete application | Collaborating with other Textarossapartners (Fraunhofer IISB) to test via an FPGA platform the use of Posit vs. Float 32 for a complex algorithm, like the reverse time equation in the Oil&Gas Fraunhofer application case. |

Table 2.2: Specifications for verification procedures for the IP cores in WP2 of Textarossa

## 2.2. Preliminary design space exploration for specification consolidation

To consolidate the specifications in Table 1 we,

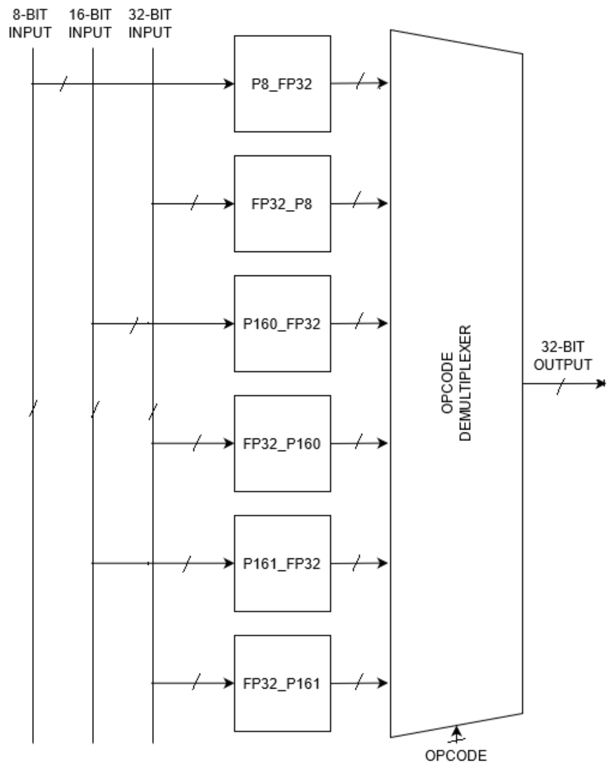


Figure 2.1: Light PPU co-processor with only conversions FP32 to Posit8/16 and viceversa

The Light PPU will be developed focusing only on the compression abilities of posits by providing a co-processor with only the following conversions as objectives:

- Conversion from IEEE 32-bit float (a.k.a binary32) to posit16/8 and vice versa (see Figure 2.1)

- Conversion from fixed point (various sizes according to posit) to posit16/8 and vice versa

By exploiting the possibility of customizing the opcodes foreseen in a RISC-V instruction set, the selection of the specific conversion can be done by the new opcode itself used as control input of a multiplexer.

As reported in Figure 2.2, top, this light PPU co-processor can be paired with a RISC-V core that already has a floating-point unit without disrupting the already present pipeline: e.g. CVA6 Ariane 64b RISC-V available https://github.com/openhwgroup/cva6.
Hence, as a specification for the FPGA implementation activity in WP2, we have integrated a CVA6 Ariane 64b RISC-V core (with its original ALU and FPU) to our Light PPU co-processor, synthesizing it targeting a Xilinx Genesys 2 FPGA [15], thus obtaining a functional RISC-V core that supports INT, FP and Posits and that could run a general-purpose Linux distribution.
Since the Posit computation in the case of Figure 2.2 is done via translation in FPU domain and use of the FPU, then Posits can be useful for data compression vs. float but not to increase computation efficiency.

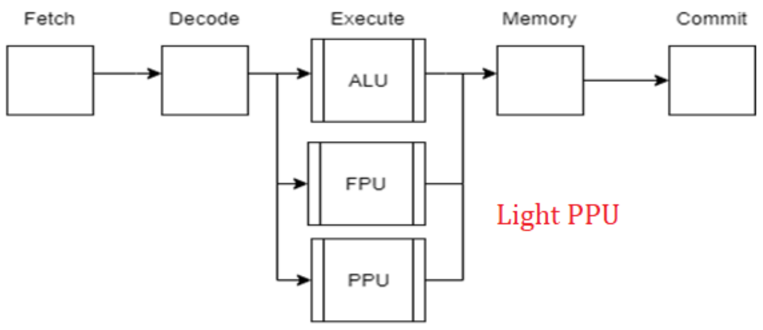


Figure 2.2: Light PPU possible integration mode within the RISC-V instruction set (with ALU and FPU)

On the other hand, with the Full PPU IP core that gives full support to all arithmetic operation to Posits we can equip a RISC-V core with the ALU but without the FPU. Indeed, the FPU may be replaced by the FULL PPU, see Figure 2.3.
Hence, to support compatibility with algorithms having float computation the Full PPU should support also conversion from float to posit, computing in Posit format and vice versa computing from posit to float. Hence, the Full PPU gives both Posit and (via translation) Float computation support.

As a specification for the FPGA implementation activity in WP2, we have integrated an Ibex RISC-V core (with its original ALU, but missing the FPU) to our Full PPU co-processor, synthesizing it targeting an Alveo U280 Xilinx FPGA, the same foreseen in the IDV-E TEXTAROSSA WP5, thus obtaining a functional RISC-V core that supports INT, FP and Posits and that could run a general-purpose Linux distribution.

Since the Posit computation in the case of Figure 2.3 is done directly while the computation of Floats is done via translation in Posit domain and use of the Full PPU, then Posits can be useful for both data compression and increased computation efficiency vs. float. Obviously, the Float calculation according to the scheme in Figure 2.3 is less efficient than in platforms where there is also a dedicated hardware FPU.
As a third possibility, the scheme in Figure 2.4 foresees the integration of RISC-V with ALU and FPU plus the Full PPU. In such case there will be a higher hardware complexity that in the cases in Figures 2.2. and 2.3 but all computation domains will be optimized: integer on the ALU, FP type on the FPU

and Posits on the Full PPU: Format conversion will be still possible thanks to the support foreseen in the Full PPU.

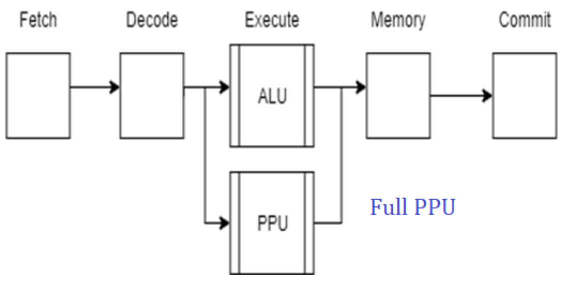


Figure 2.3: Full PPU possible integration mode within the RISC-V instruction set (with ALU, but replacing the FPU)

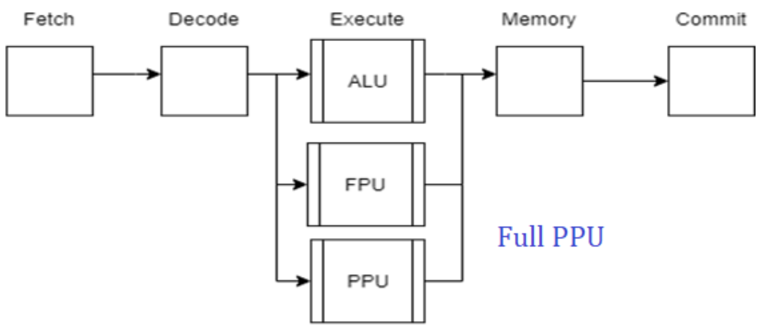


Figure 2.4: Full PPU possible integration mode within a RISC-V processor with ALU and FPU

# 3. eXtreme Secure Crypto IP

## 3.1. Specifications of advanced secure services

This chapter focuses on the specification of advanced security features to be supported by dedicated IP macrocells to be developed in WP2, Task 2.2.

As already specified in Section 2, TEXTAROSSA aims at being complementary to EPI SGA1.

In EPI SGA1 UNIPI has developed and delivered to SiPearl, in order to be integrated in Rhea1, a suite of cryptographic accelerators covering symmetric encryption (AES 128 bits and 256 bits), Asymmetric encryption (ECC-based schemes like ECDH.ECIES, ECDSA), hashing functions like SHA2 and SHA3 from 256 bits to 512 bits, True Random Number Generator (TRNG) and a Cryptographic Secure Pseudo RNG (CSPRNG). Each cryptographic accelerator can be connected to an Arm or a RISC-V processor via an AXI4 interface.

Details on Cryptographic IPs developed in EPI 1 are public and can be found at the following link:

https://www.european-processor-initiative.eu/dissemination-material/crypto-tile-factsheet/

In order to be complementary to the security developments in EPI1 and to cover new security features of interest, the activities in WP2 of TEXTAROSSA about security have been specified to cover the following cutting-edge cryptography specifications (SPEC_SEC1-4):

SPEC-SEC1: TEXTAROSSA will develop an IP macrocell accelerating the computing kernel of Homomorphic Encryption (HE) algorithm, which in Cloud and HPC server applications are needed when to ensure the data privacy of different users the Server is considered untrusted.

Indeed, in standard cryptographic system only the communication edge-cloud (i.e. client-server) is encrypted and the data of different users are stored in plain text mode in the server. Instead, in HE the data on the server are kept encrypted but HE ensures the property that operations data in the server on the encrypted data of the suers give the same results (from here the term Homomorphic where “homo” means “the same”) as doing the operations on the decrypted data.

Since WP2 of TEXTAROSSA aims at developing IPs accelerating wide-spread adopted algorithms, then:

SPEC-SEC2: we will accelerate one of the first and well-known SW implementation of the HE approach, the SEAL library from Microsoft. The latter is supported also by INTEL on its AVX512 instruction-set and by the NVIDIA GPU AI framework called CLARA.

Moreover,

SPEC-SEC3: TEXTAROSSA will develop an IP core implementing in HW the eXtendable Output Functions (XOF) SHAKE 128/256, which is a new hashing function vs. the SHA2 and SHA3 accelerators developed in EPI1.

To be noted that SHAKE is already adopted by algorithms like Crystals-Dilithium for digital signature using Lattice LWE (Learning With Errors) Codes that have been recently standardized for Post-quantum crypto applications.

Hence, this will realize a

SPEC_SEC4: TEXTAROSSA IP will be a first step towards Post-quantum Cryptographic applications.

These specifications will be further discussed in Sections 3.2 and 3.3.

Before proceeding to the IP macrocells design, a preliminary benchmarking campaign has been carried out to assess the performance on different CPU architectures of the aforementioned cryptographic functions.

This analysis aims at: i) consolidating the specification of designing IP macrocells (using SystemVerilog Language and synthetizing the IP macrocell in Xilinx FPGA technology); ii) defining the main limits and bottlenecks of such algorithms; iii) defining at the beginning of the design process possible HW/SW strategies and specifications to improve performance in terms of computation time and energy efficiency.

## 3.2. Homomorphic Encryption: SEAL-Embedded Library

Homomorphic Encryption (HE) is a specialized type of encryption that allows specific computations on the encrypted data and generates a cyphertext that, once decrypted, matches the result of operations performed on the plaintext data. HE is nowadays considered a strong privacy-preserving solution that allows users to share data with clouds or any non-secure server.
However, HE requires high computational resources and memory consumption, which limits its use in resource-constrained IoT devices. Different HE libraries exist, and the main ones are: Microsoft SEAL [1], PALISADE [2], and HELib [3]. Nevertheless, all of them are not specifically designed for resources-constrained devices. The SEAL-Embedded library [4] is the first HE library targeted for embedded devices that employ several optimizations to perform the encoding and encryption of data, featuring the CKKS Homomorphic Encryption scheme.
An assessment of the SEAL-Embedded library has been carried out to evaluate its performance on different CPUs, and the results will be presented in the next section.

### Benchmark on RISC-V CPUs

The source code of the SEAL-Embedded library can be found in [5]. Two different RISC-V processors have been selected for the benchmark campaign, and two different environments have been implemented on the FPGA Board Zynq UltraScale+ MPSoC ZCU106 equipped with the System-on-Chip (SoC) XCZU7EV-2FFVC1156.
Figure 3-1 shows the proposed hardware systems running the benchmark. The selected RISC-V processors are:

- The 32-bit RISC-V RISCY, whose HDL code can be downloaded in [6]. The left side of Figure 3-1 shows the complete system implemented in the target FPGA which encompasses the RISCY CPU, 256 kB of on-chip memory, and AXI4 peripherals (i.e. JTAG and serial UART interface).
- The 64-bit RISC-V CVA6, whose HDL code can be downloaded in [7]. The right side of Figure 3-1 shows the complete system implemented in the target FPGA which includes the CVA6 CPU, 512MB of memory (i.e. onboard DDR4), and AXI4 peripherals (i.e. JTAG and serial UART interface).

Both systems run at 100 MHz of frequency on the target FPGA.

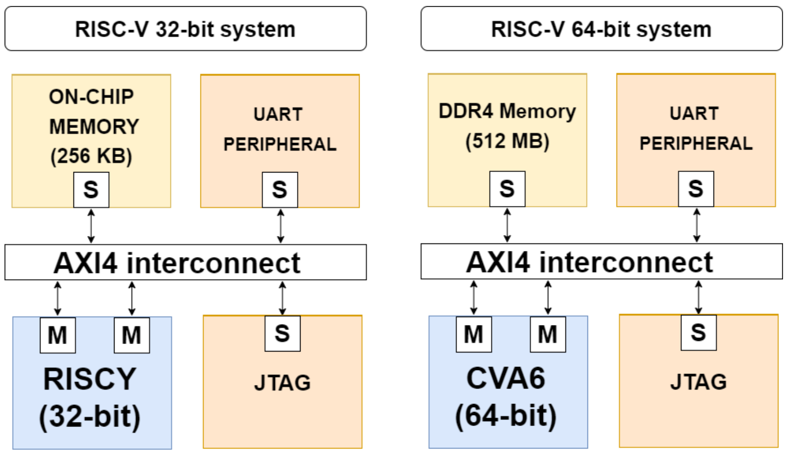


Figure 3.1: RISCV-based systems for benchmarking the SEAL-Embedded library.

The proposed systems permitted to find out the main bottleneck of the SEAL-Embedded library, which relies on the encryption function based on Ring Learning With Errors (RLWE) computation.
Table 3-1 shows the benchmark results. The first column indicates the selected polynomial degree for the RLWE encryption. This parameter impacts both the message size (i.e. the size of the message blocks that must be provided to the encryption function, reported in the Msg size column) and the security strength of the RLWE encryption. Further details about the SEAL-Embedded library can be found in [4]. Despite the SEAL-Embedded being targeted for resource-constrained devices, it cannot be successfully executed on the RISCY CPU for Poly-Degree higher than 4096 (256 kB of memory are not enough). In addition, the latency for the encryption process is extremely high: around 3 seconds are required to encrypt 8 kB with 4096 Poly-Degree.

| Poly-Degree | Msg size | CVA6 (64-bit) | RISCY (32-bit) |
|---|---|---|---|
| 1024 | 2048 B | 17.19 ms | 207.10 ms |
| 2048 | 4096 B | 37.09 ms | 444.22 ms |
| 4096 | 8192 B | 273.80 ms | 2806.43 ms |
| 8192 | 16384 B | 1184.19 ms | -- |
| 16384 | 32768 B | 5861.02 ms | -- |

Table 3.1: Benchmark results for the encryption function of the SEAL-Embedded Library. Column 1 indicates the selected polynomial degree for the RLWE encryption, column 2 indicates the message size in Bytes, column 3 shows the results for the CVA6 processor and column 4 the results for the RISCY processor. Both CPUs run at 100 MHz.

### Hardware accelerator specifications definition

Table 3.2 summarizes desired specifications of the hardware accelerator for the SEAL-Embedded library.

| Spec | Target | Further Specification | Condition | Comments |
|---|---|---|---|---|
| SPEC-SEC1 | IP macrocell accelerating the computing- |  | Must Have |  |

| | | | | |
|---|---|---|---|---|
| | intensive kernel of HE algorithm | | | |
| SPEC-SEC2 | IP macrocell the SEAL library from Microsoft | Starting from SEAL Embedded library version | Must Have | |
| SPEC-SEC5 | Communication Interface | AXI4 - Memory Mapped (MM) | Must Have | Standard AXI4 Slave MM interface for the communication with CPU. |
| | | AXI4 – Direct Memory Access (DMA) | Nice to Have | DMA for High-Throughput data exchange from/to memory. Depending on the needs could be implemented. |
| SPEC-SEC6 | Supported parameters | Poly-degree for the RLWE symmetric encryption. Hardware support for all the parameters (i.e. from 1024 to 16384). | Must Have | |
| SPEC-SEC7 | Latency for encryption | Desired latency for encryption of 8 kB (with the polynomial degree of 4096) could be hundreds of milliseconds. | Must Have | Depending on the needs the performance can be improved respect to the defined specification. |

Table 3.2: Specification definition of the hardware accelerator for the SEAL-Embedded library.

**Testing procedure for the IP macrocell and KPIs**

Please note that to test the IP under development the idea is repeating tests like those in Table 3.1 in the target FPGA platform (e.g. an FPGA Board like the Zynq UltraScale+ MPSoC ZCU106 equipped with the XCZU7EV-2FFVC1156) but with the presence of the new accelerated IP.
The KPIs to analyze the achieved implementation results will be:

- The functional equivalence of the accelerated algorithm (HW-SW) with the original pure SW algorithm,
- The complexity overhead of the accelerator in FPGA technology,
- The computation speed-up thanks to the sue of the accelerator IP macrocell vs a non-accelerated (pure SW) implementation.

## 3.3. eXtendable Output Functions (XOF) SHAKE-128/256

An eXtendable Output Function (XOF) is a variable-length HASH function in which the length of the output can be chosen to meet the requirements of individual applications. The XOFs can be specialized to hash functions or used in a variety of other applications. The reference standard for the XOF is the NIST FIPS 202 [8], where two XOFs are specified: SHAKE-128 and SHAKE-256. Several NIST Post-Quantum finalists for both Key Encapsulation Mechanism (KEM) and Digital Signature (DS) adopt the XOF functions SHAKE 128/256: CRYSTALS-Kyber (KEM) and Dilithium (DS), Classic McEliece (KEM), NTRU (KEM), Saber (KEM) and Falcon (DS). In particular, in DS algorithms the hardware acceleration of XOFs becomes crucial since they are employed to HASH messages of any size. Some IoT applications, for instance Over-The-Air update, requires verifying the DS of large messages (e.g. up to Gigabytes) with low latency. Next section will show the benchmark results of the DS algorithms CRYSTALS-Dilithium and Falcon running on both RISC-V CVA6 and ARM-A53 CPUs, aiming to identify how the message size affects the computation time.

## Performance evaluation in Post-Quantum Digital Signature Algorithms

The source code of the Crystals-Dilithium and Falcon algorithms can be downloaded at the NIST official page for the PQC competition: https://pq-crystals.org/, https://falcon-sign.info/. In this case, we selected the CPUs RISC-V CVA6 and ARM-A53 because they can be reasonably used for IoT applications. Two different environments have been implemented on the FPGA Board UltraScale+ MPSoC ZCU106 equipped with the System-on-Chip (SoC) XCZU7EV-2FFVC1156:

- A RISC-V CVA6-based system, the one reported on the right side of Figure 3-1. In this case, the entire system is implemented on the target FPGA at 100 MHz of frequency.
- An ARM-A53-based hard-core system running at 1.2 GHz of frequency. The processor is connected to 2 GB of DDR4 memory.

Table 3-3 reports the results for the DS verification function of CRYSTALS-Dilithium and Falcon algorithms with different message lengths (i.e. from 10 kB to 100 MB) on the RISC-V CVA6 CPU, while Table 3-4 reports similar results on the ARM-A53 CPU (in this case the message length varies from 10 kB to 1 GB).

| **Message length[byte]** | **VERIFICATION FUNCTION – RISC-V CVA6 processor** | | | |
|---|---|---|---|---|
| | **Dilithium-2** | **Dilithium-5** | **Falcon - 512** | **Falcon - 1024** |
| 10k | 30,27 ms | 69,21 ms | 14,11 ms | 16,91 ms |
| 100K | 104,85 ms | 143,60 ms | 83,99 ms | 86,88 ms |
| 1M | 865,77 ms | 904,37 ms | 799,24 ms | 802,12 ms |
| 10M | 8455,38 ms | 8492,71 ms | 7.933,16 ms | 7.936,13 ms |
| 100M | 84351,33 ms | 84375,76 ms | 79.273,83 ms | 79.275,83 ms |

Table 3.3: Computation time for the DS verification function of CRYSTALS-Dilithium and Falcon algorithms on the RISC-V CVA6 CPU.

| **Message length [byte]** | **VERIFICATION FUNCTION – ARM-A53 processor** | | | |
|---|---|---|---|---|
| | **Dilithium-2** | **Dilithium-5** | **Falcon - 512** | **Falcon - 1024** |
| 10K | 4,65 ms | 10,37 ms | 4,6 ms | 6,26 ms |
| 100K | 13,80 ms | 19,52 ms | 33,08 ms | 34,71 ms |
| 1M | 107,77 ms | 113,49 ms | 325,00 ms | 326,63 ms |
| 10M | 1.045,22 ms | 1.050,89 ms | 3.236,75 ms | 3.238,38 ms |
| 100M | 10.419,43 ms | 10.424,82 ms | 32.354,11 ms | 32.355,73 ms |
| 1G | 104.161,53 ms | 104.167,26 ms | 323.527,44 ms | 323.529,06 ms |

Table 3.4: Computation time for the DS verification function of CRYSTALS-Dilithium and Falcon algorithms on the ARM-A53 CPU.

## Hardware accelerator specifications definition

Table 3-5 summarizes the desired specifications of the hardware accelerator for the SHAKE-128/256 functions.

| | Target | Specification | Condition | Comments |
|---|---|---|---|---|
| SPEC_SEC3 | implementing in HW the eXtendable Output Functions (XOF) SHAKE | SHAKE 128 and SHAKE 256 | Must Have | |
| SPEC_SEC4 | Compliant with use in Post-quantum Cryptographic applications. | - | Nice to Have | SHAKE used as hashing function in recently standardized PQC signature algorithms (e.g. Dilithium) |
| SPEC_SEC8 | Communication Interface | AXI4 - Memory Mapped (MM) | Must Have | Standard AXI4 Slave MM interface for the communication with CPU. |
| | | AXI4 – Direct Memory Access (DMA) | Nice to Have | DMA for High-Throughput data exchange from/to memory. Depending on the needs could be implemented. |
| SPEC_SEC9 | Supported parameters | Support both SHAKE-128 and SHAKE-256 functions | Must Have | |
| SPEC_SEC10 | Latency/throughput | To be further investigated | TBD | Depending on the needs, desired latency and throughput shall be identified. Data exchange via DMA can significantly improve performance. |

Table 3.5: Specification definition of the hardware accelerator for the SHAKE-128/256 functions.

**Testing procedure for the IP macrocell and KPIs**

Please note that to test the IP under development the idea is repeating tests like those in Tables 3-3 and 3-4 in FPGA platform (e.g. an FPGA Board like the Zynq UltraScale+ MPSoC ZCU106) but with the presence of the new Shake hardware IP.
The KPIs to analyze the achieved implementation results will be:

- The functional equivalence of the accelerated algorithm (HW-SW) with the original pure SW algorithm
- The complexity overhead of the accelerator in FPGA technology
- The computation speed-up thanks to the sue of the accelerator IP macrocell vs a non-accelerated (pure SW) implementation

# 4. IPs for low-latency intra-node and inter-node communication links

The INFN Communication IPs implement a n-D Torus direct network for FPGA accelerators, allowing low-latency data transfer between processing tasks deployed on the same FPGA (IntraNode communication) and on different FPGAs (InterNode communication), see Figure 2-1.

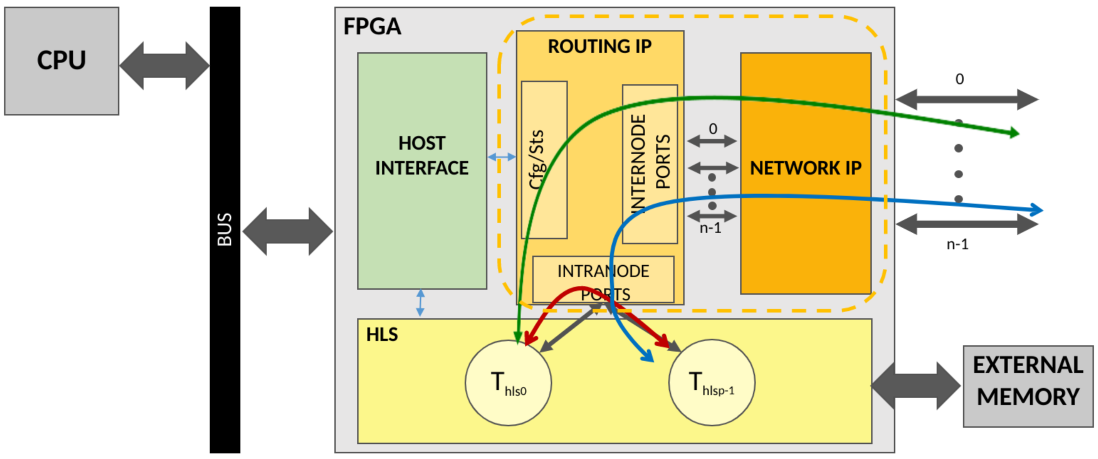


Figure 4.1 : Example of IntraNode (red) and InterNode (green and blue) data transfers between tasks

The hardware block structure, depicted in Figure 4.2, can be split into a **Network_IP** and a **Routing_IP**, described in more detail in the next sections.

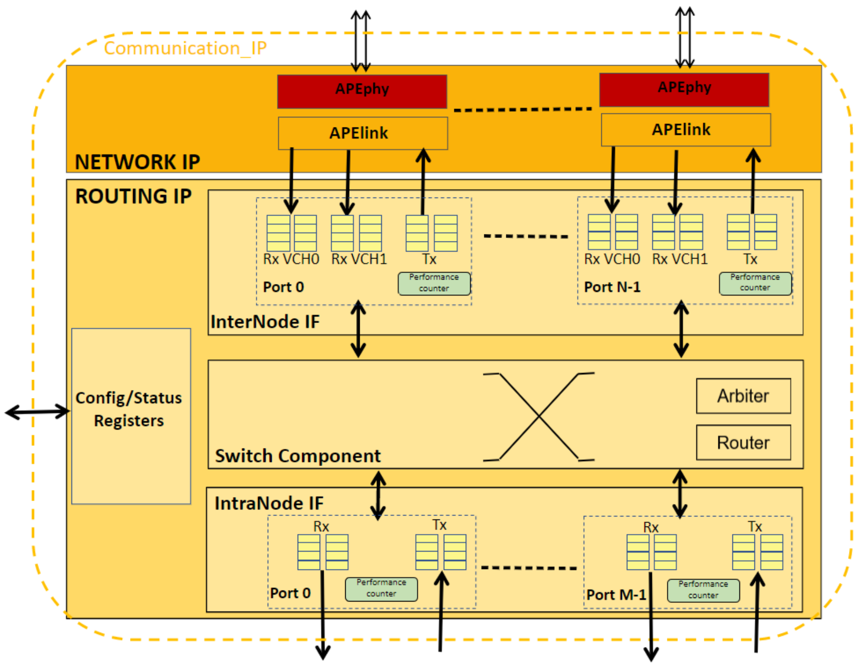
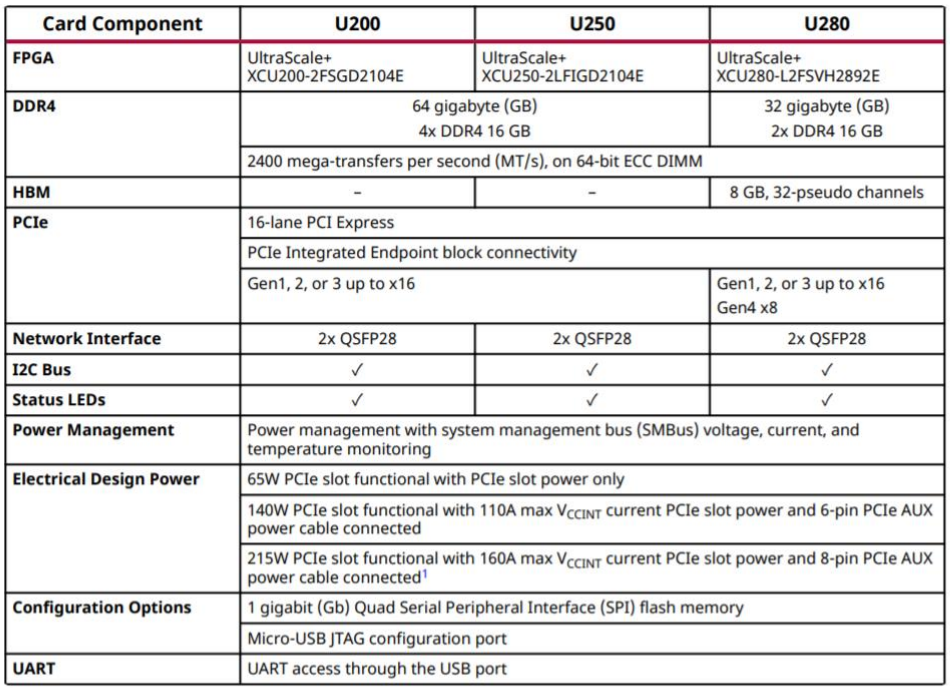


Figure 4.2: Architectural partition of Communication IPs

The INFN Communication IPs, developed in VHDL, will be implemented as RTL-kernel in Vitis, a framework which allows to develop, debug, and optimize accelerated applications using standard programming languages for both software and hardware components.

In TEXTAROSSA project, target platforms are both Xilinx Alveo U200 and U280 cards, featuring the Xilinx UltraScale+ technology

Table 4.1 collects the features of the Alveo family of boards.



| Card Component | U200 | U250 | U280 |
|---|---|---|---|
| **FPGA** | UltraScale+ XCU200-2FSGD2104E | UltraScale+ XCU250-2LFIGD2104E | UltraScale+ XCU280-L2FSVH2892E |
| **DDR4** | 64 gigabyte (GB) 4x DDR4 16 GB | | 32 gigabyte (GB) 2x DDR4 16 GB |
| | 2400 mega-transfers per second (MT/s), on 64-bit ECC DIMM | | |
| **HBM** | - | - | 8 GB, 32-pseudo channels |
| **PCIe** | 16-lane PCI Express | | |
| | PCIe Integrated Endpoint block connectivity | | |
| | Gen1, 2, or 3 up to x16 | | Gen1, 2, or 3 up to x16 Gen4 x8 |
| **Network Interface** | 2x QSFP28 | 2x QSFP28 | 2x QSFP28 |
| **I2C Bus** | ✓ | ✓ | ✓ |
| **Status LEDs** | ✓ | ✓ | ✓ |
| **Power Management** | Power management with system management bus (SMBus) voltage, current, and temperature monitoring | | |
| **Electrical Design Power** | 65W PCIe slot functional with PCIe slot power only | | |
| | 140W PCIe slot functional with 110A max $V_{CCINT}$ current PCIe slot power and 6-pin PCIe AUX power cable connected | | |
| | 215W PCIe slot functional with 160A max $V_{CCINT}$ current PCIe slot power and 8-pin PCIe AUX power cable connected[1] | | |
| **Configuration Options** | 1 gigabit (Gb) Quad Serial Peripheral Interface (SPI) flash memory | | |
| | Micro-USB JTAG configuration port | | |
| **UART** | UART access through the USB port | | |

Table 4.1 – Features of the Alveo family of boards

These cards provide a PCI Express interface to allow communication between the host processor and the network and are equipped with two 4-lane QSFP28 ports capable of 100 Gbps each.
QSFP+ ports available allow the connection, using point-to-point, bi-directional, full-duplex communication channels, of each board with its two neighbors in a 1-D torus network topology (a ring). The relevant KPIs for the Communication IP are a) communication latency, with the aim of reaching sub-microsecond figures and, b) communication bandwidth, with the aim of exploiting most of the raw bandwidth offered by the different kind of communication channels that will be implemented (see section 4.2 below).

The development of the Communication IP addresses the following project objectives:

Objective 1 - Energy efficiency. Implementing a direct communication between FPGAs, it avoids the usage of bounce buffers and the involvement of the CPUs and system bus resources in data transfers improving the energy efficiency of the multi-FPGA execution platform. The achievement of this objective will be assessed both directly (depending on the availability of the power modelling tools developed in Task 4.5) through measurement of the energy cost for data movements and indirectly by measuring the processed Events/J obtained by the RAIDER application, and comparing it with what will be obtained running the same processing pipeline both on a CPU only and on a CPU+GPU systems.

Objective 2 - Sustained application performance. The sustained application performance of distributed dataflow applications, such as the RAIDER use case, are strongly affected by the performance of the network system. Implementing a direct FPGA to FPGA interconnect and bypassing the host network stack, allows to keep the communication latency in the sub-microsecond range and to increase the bandwidth for small messages. The achievement of this objective will be assessed both directly, measuring one-way communication latency and communication bandwidth, and indirectly, measuring the processed Events/s obtained by the RAIDER application, and comparing it with what will be obtained running the same processing pipeline both on a CPU only and on a CPU+GPU systems.

Objective 3 - Seamless integration of reconfigurable accelerators. The IP enables the deployment of distributed dataflow applications over a multi-FPGA execution platform.

Objective 4 - Development of new IPs. The INFN Communication IP is the key enabling technology behind the APEIRON framework, allowing direct low-latency intra/inter FPGA communications between HLS kernels.

## 4.1. Routing IP

The **Routing_IP** defines the switching technique and routing algorithm, dynamically interconnecting all IP's ports and solving contentions for shared resources.
The transmission is packet-based, in the sense that Communication IP sends, receives, and routes packets with header (shown in figure 4.3), a variable size payload and a footer.

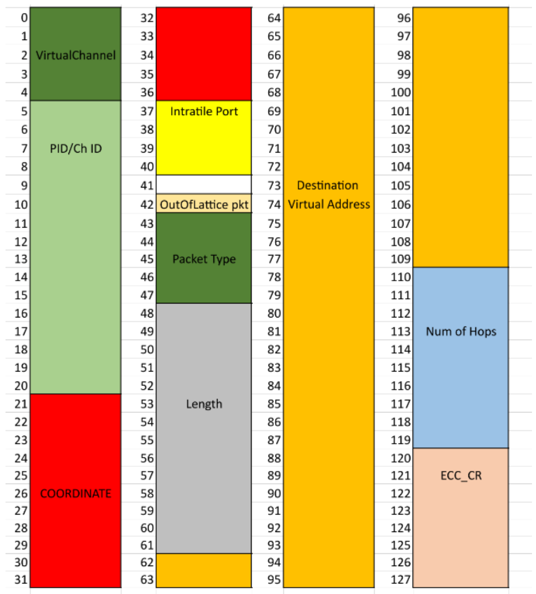


Figure 4.3: Packet's header format

The two sets of interfaces exposed, i.e. **IntraNode** and **InterNode**, are composed by a number of ports (M and N) that can be customized at design time.
The **IntraNode IF** manages data flow to (RX) and from (TX) local tasks; each port consists of two FIFOs for each direction, so that header/footer and data use a specific FIFO.
For interNode communications the routing policy applied is the dimension-order one (DOR): it consists in reducing the coordinates offset between current and destination node to zero while routing the packet, considering one dimension at time in an inverse lexicographic order (e.g. ZYX).
The deadlock-avoidance of DOR routing is guaranteed by the implementation in the I**nterNode IF** of two virtual channels for each physical channel [9].
The employed switching technique — i.e., when, and how messages are transferred — is Virtual Cut-Through (VCT) [10]: the router starts forwarding the packet as soon as the routing algorithm has picked a direction and the receiving buffer has enough space to store the full packet.

### 4.1.1 HLS Communication Adaptor

Task-side, input/output channels' interface is decoupled from the Routing IP one, so that the user doesn't have to care about the network protocol. A task should only implement a generic stream interface for each communication channel, based on the AXI4-Stream protocol, as follows:

```
void example_task(
            [optional kernel-specific list of parameters],message_stream_t
message_data_in[N_INPUT_CHANNELS],message_stream_t message_data_out[N_OUTPUT_CHANNELS])
```

Where the message_stream_t type is defined as:

```
typedef hls::stream<uint128_t> message_stream_t;
```

The Communication Library leverages AXI4-Stream Side-Channels to encode all the information needed to forge the packet header.
Adaptation toward/from IntraNode ports of the Routing IP is done by two IPs: Aggregator and Dispatcher. The Dispatcher receives incoming packets from the Routing IP and forwards them to the right input channel, according to the relevant fields of the header. The Aggregator receives outgoing packets from the task and forges the packet header, filling then the header/data FIFOs of the Routing IP.

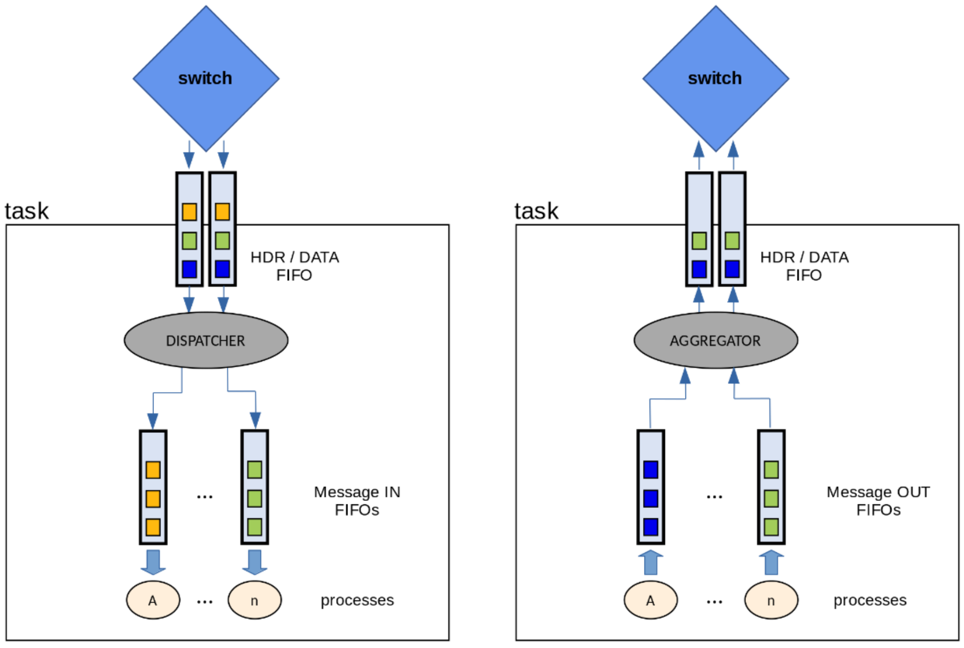


Figure 4.4: Schematic view of incoming and outgoing communication flow

## 4.2. Network IP

The **Network_IP** block, oversees managing data flow over the serial links between FPGAs.
In the first Communication IPs release, to transfer data between each node with its neighbors we will use Xilinx Aurora 64B/66B cores for the serialization of the messages over the cable, and INFN APElink IP [11] to guarantee reliable communication, performing error detection and correction.
In the final version we will implement also 10/25 Gbps and 100 Gbps Ethernet.

## 4.3. Communication IP Specifications

In Table 4.2 we report the specification of the Communication IP, for both the intermediate release (M18) and for the final release (M30).

| | Intermediate Release (M18) | Final Release (M30) |
|---|---|---|
| **Number of IntraNode ports** | 1, 2 | 1,2,3,4 |
| **Number of InterNode ports** | 2 | 2 |
| **Number of lanes/transceivers for an InterNode Channel** | 2 | 4 |
| **Raw Bandwidth of an InterNode Channel** | 20 Gbps | 40 Gbps |
| **Internal Router Datapath Width** | 128 bit | 256 bit |
| **Raw Bandwidth of an IntraNode Channel** | 12.5 Gbps | 37.5 Gbps |
| **IntraNode one-way latency** | <= 500 ns | <= 500 ns |
| **InterNode one-way latency** | <= 1 us | <= 1 us |
| **Operating clock frequency** | 100 MHz | 150MHz |

Table 4.2 – Specifications for the INFN Communication IP (intermediate and final releases)

The number of IntraNode ports reported shows the configuration that we tested and validated (or we are going to test with the final revision). The number of InterNode ports is limited by the two QSFP28 ports available in the U200 and U280 Alveo cards. We plan to increase for the Final Release the bandwidth of the external channels by increasing the number of lanes used.
In the Final Release we also foresee to increase the internal datapath width (from 128 to 256 bit) and the clock frequency, thus enhancing communication bandwidth.

## 4.4. Test and performance evaluation strategies

We will implement a BIST mechanism to validate the functionalities of the Intra/Internode communications. Both IntraNode and InterNode IFs will be provided with a self-test mechanism to measure the latency and bandwidth achieved.

Furthermore, relevant KPIs for the Communication IP, i.e. communication latency and bandwidth between HLS Kernels (both IntraNode an InterNode) will be assessed thanks to a dedicated design integrating the Communication IP with ad-hoc developed HLS kernels.

# 5. IP for fast task scheduling

The objective of task 2.5 is the development of a HW IP, compatible with a RISC-V core available in BSC, for fast task management compliant at tool chain-level with OmpSs approach that in TEXTAROSSA is then linked to the Vitis HLS tool to ensure that reconfigurable units are integrated in the tool chain. The use of tasks to interface with the accelerators will allow the runtime to integrate both tasks exploiting the "Kahn channel" abstraction and standard OmpSs (OpenMP) tasks which will improve the scope of applications targeted by the project. Preliminary results show that using a HW manager to schedule tasks significantly improves the performance of the application. This IP will be a service IP since it interfaces with the cores and other IPs in the project. This section explains the high-level view, functionality and interface of the IP, while all the details are reported in Deliverable 2.10.

As explained in Deliverable 2.10, the fast task scheduling IP is related to the following project objectives and strategic goals as stated in the DoA:

- Objective 1 - Energy efficiency. The IP reported in this deliverable is designed to be integrated in an FPGA or attached as a runtime accelerator to a manycore system. It provides two ways of increasing energy efficiency: a first-order effect by improving the energy efficiency of the task-based runtime and, a second-order effect that is achieved by improving the efficiency of the application being executed using the runtime.
- Objective 2 - Sustained application performance. As with Objective 1, the IP reported in this deliverable contributes to sustained application performance: by improving the performance of the task-based runtime and also, by improving the performance of the application being executed using the runtime. As a fast task scheduling effectively increases application available parallelism, this second effect improvement is expected to be significant.
- Objective 3 - Fine-tuned thermal policies integrated with an innovative cooling technology. The Fast Task Scheduling IP is expected to be able to work with the software part of the runtime by either, providing it with information about the power consumption of the tasks and/or enabling the thermal control system (in software) to actuate over the accelerators if necessary.
- Objective 4 - Seamless integration of reconfigurable accelerators. OmpSs@FPGA runtime allows for seamless integration of reconfigurable accelerators (as detailed in Deliverable 4.1 and Deliverable 1.4). As an integral part of the framework the IP should allow for scheduling of tasks that are either specific to an accelerator or destined to be executed in a general-purpose unit.
- Objective 5 - Development of new IPs. This deliverable reports the development of a new IP dedicate to scheduling tasks, so it directly tackles objective 5.
- Objective 6 - Integrated Development Platform. As part of the OmpSs@FPGA runtime the IP reported in this deliverable will be used in applications executing on the IDV-E platform. It is important to highlight that IDV-E features a host CPU (ARM based) that has never before been used to drive computation in a PCIe attached FPGA. Developing the system in a way that is compatible with new different CPUs helps ensuring new host CPUs (like EPI CPUs) will be able to drive this kind of computations in the future.

- Strategic Goal #1: Alignment with the European Processor Initiative (EPI). The Fast Task Scheduling IP will be aligned with EPI in its both application fields. On one side, along with OmpSs@FPGA, it will provide a system that can use an EPI processor to drive computations in a cluster of FPGA PCIe attached accelerators. Also, the second field of application of the Fast Task Scheduler is to accelerate a manycore system. We are doing work (to be reported in Deliverable 2.11) to integrate it with a manycore system to accelerate parallel computations in such systems. The preliminary results are promising and have been sent to be considered for publication.

- Strategic Goal #2: Supporting the objectives of EuroHPC as reported in ETP4HPC's Strategic Research Agenda (SRA) for open HW and SW architecture. The Fast Task Scheduler IP is developed following the open HW model and is freely available as a standalone IP or as part of the whole OmpSs@FPGA framework.
- Strategic Goal #3: Opening of new usage domains. The IP reported in this deliverable addresses a problem that affects current manycore platforms: their inability to exploit large-scale parallelism when each of the pieces of work involved (tasks) is small. In this sense, although the IP itself doesn't address any specific usage domain removing this system bottleneck opens the possibility of executing efficiently new applications on the objective platforms.

## 5.1 Basic design and functionality

This document describes the basic design and functionality of the IP for fast task scheduling (from now on FTS or Fast Task Scheduler). A first diagram is shown in Figure 5.1.1.

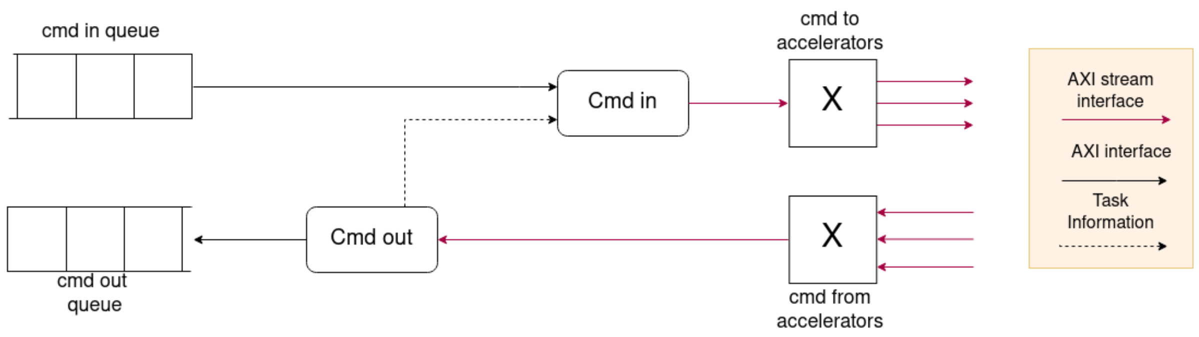


Figure 5.1.1 IP for fast task scheduling diagram (FTS).

As it can be seen in Figure 5.1.1 the system is composed of two command queues, one for input coming from the CPU/exterior of the FPGA ("cmd in queue") and another going to the CPU/exterior of the FPGA ("cmd out queue"), two control modules ("Cmd in" and "Cmd out") and two interconnection multiplexers/demultiplexers. Tasks are sent from the host CPU to the fast task scheduler by using commands that are temporarily stored in the "cmd in queue". These commands are processed in order by the "Cmd in" module and, depending on the accelerators availability, are sent to the appropriate accelerator. Commands are sent through the "cmd to accelerators" demultiplexer through an AXI stream interface, and only when accelerators are available (ready) in order to avoid interface contention and starvation.

Once the task has been processed by the corresponding accelerator, it informs the FTS through an output AXI stream interface that is multiplexed to reach the "Cmd out" module with a "Finished Task" command. "Finished Task" command is expected to be processed in very few cycles (tens of cycles at most). Therefore, although some contention can be expected when several accelerators finish at the same time submitting this command, no significant performance drop is expected in this case.

The "Cmd out" module is in charge of processing the "Finished Task" packet by forwarding it with the adequate format to the "cmd out queue" and to notify the "Cmd in" module about the new ready state of the accelerator in order for the FTS to forward a new task to it.

## Tasks and Periodic Tasks

In order to accomplish fast task scheduling, the prototype IP is going to be able to schedule both tasks and periodic tasks [12].

Periodic systems (i.e., recurrent workloads) are a common workload in industrial environments and real-time applications. Those workloads use the task concept to define the different activities that must be executed periodically (after some amount of time). Thereby, task-based parallel programming models are great candidates to support recurrent workloads. We propose extending the current syntax of task-based parallel programming models to define the main recurrent task parameters. Therefore, modeling recurrent workloads can be accomplished efficiently in terms of code lines, and with all parallel capabilities of baseline programming models. Also, we propose using the reconfigurable heterogeneous platforms to efficiently manage these recurrent workloads. These platforms will provide an efficient management of recurrent tasks, keeping the great programmability provided by the parallel programming models.

Periodic tasks are defined as tasks that repeat themselves a number of times. This repetition can be set a number of times (as soon as possible) or at a certain time interval defined by the user (provided that the task is executed in less time than the defined trigger interval). These kinds of tasks have demonstrated to be very powerful to address the problems of industrial environments [5.1]. The support for periodic tasks would also be incorporated in the programming model support in task 4.2.

The periodic tasks syntax includes two clauses: period(N) and num_repetitions(K). An example for a periodic and a regular task definition is shown in Figure 5.1.2.

```
# pragma omp task inout ([10] array ) num_repetitions(reps) period (1000000)
void periodic_task ( int * array , const int reps );
# pragma omp task inout ([10] array )
void regular_task ( int * array );
int main (...) {
        int array [10];
        regular_task ( array );
        periodic _task ( array , 100);
        regular_task ( array );
        # pragma omp taskwait
}
```

Figure 5.1.2 Periodic and regular task definition example.

The main function calls the regular task, then the periodic task, and finally the regular, creating a chain of three task instances due to its data dependence. The periodic task has the num_repetitions clause, which defines that the task body will be executed reps times (in this case, the argument value is 100), and the period clause, which defines that the task will begin the execution every 1 second (1000000 microseconds). The first regular task becomes ready when created as its data dependences are satisfied. In contrast, the other two are postponed. The recurrent task is postponed until the first regular task finishes, and the second regular task is postponed until the 100 repetitions of the recurrent task have been executed.

## 5.2 Queues and Commands information

The following section describes the structure of memories used to communicate the Host (usually using libxtasks) with the FTS.

**Command in and Command out queues**

Each queue has 1024 elements (uint64_t type) and it is divided into 16 subqueues of 64 elements. Each subqueue corresponds to one accelerator, starting from accelerator 0 (positions [0,63]) to accelerator 15 (positions [960,1023]) as shown in Figure 5.2.1.

```
 1023                  64  63              0
+----------------------------------------+
| | | | |    ...     | | |    ...    | | |
+----------------------------------------+
<--- 1 subqueue --->
<------------ 1024 positions ----------->
```

Figure 5.2.1 Command in and Command out queues.

Each command uses a dynamic number of slots in the queue. The number of slots depends on the command. The odd command codes make the accelerator become busy (no further commands will be sent to the accelerator until it returns the command out response) and the even command codes do not. The information is structured as shown in Figure 5.2.2.

```
 63                                                         0
+------------------------------------------------------------+
| Valid |                Command Arguments              | Code  |
+------------------------------------------------------------+
|                      Command payload                       |
|                           ...                              |
+------------------------------------------------------------+
<-------------------- 64 bits - 8 bytes --------------------->
```

- [7  :0  ] Command code
  * 0x01 - Execute task cmd
  * 0x03 - Finished task cmd
  * 0x05 - Execute period task cmd
- [55 :8  ] Command arguments
- [63 :56 ] Valid Entry
  * 0x00 - Invalid
  * 0x80 - Valid
- [  :  ] Command payload

Figure 5.2.2 Commands format.

As it can be seen the commands use the first position to indicate the command and the next positions as a payload (actual command information).

**Commands format**

We have defined three initial commands in the FTS, a Execute task command, a Finished task command notification and a Execute periodic task command. The Execute task and Execute periodic task commands follow the structure shown in Figure 5.2.3.

```
 63                                                             0
+----------------------------------------------------------------+
| Valid |       | DesID | CompF |                | #Args | Code  |
+----------------------------------------------------------------+
| 0x00  |                  Task Identifier                       |
+----------------------------------------------------------------+
|                     Parent Task Identifier                     |
+----------------------------------------------------------------+
|             Period             |        Num. repetitions       |
+----------------------------------------------------------------+ Λ
|           ArgumentID           |                       | Flags | |
+----------------------------------------------------------------+ | 1 arg
|                            Argument                            | |
+----------------------------------------------------------------+ v
|                         Other arguments                        |
|                               ...                              |
+----------------------------------------------------------------+
<--------------------- 64 bits - 8 bytes ----------------------->
```

- [7  :0  ] Command code
  *  0x01 - Execute task cmd
  *  0x05 - Execute periodic task cmd
- [15 :8  ] Number of arguments
- [31 :16 ]
- [39 :32 ] Compute flag
  *  0x00 - Compute disabled
  *  0x01 - Compute enabled
- [47 :40 ] Destination ID where the accelerator will send the 'complete' signal
  *  0x1F - Processing System (PS)
- [55 :48 ]
- [63 :56 ] Valid Entry
  *  0x00 - Invalid
  *  0x80 - Valid
- [119:64 ] Task identifier
- [127:120] 0x00 constant. This field is used to identify task commands created externally
- [191:128] Parent Task identifier. This field is ignored by the FTS and the accelerators, it is maintained to match the format of the internal command queue and the format expected by the accelerators.
- [223:192] Number of times that task body will be executed (Execute periodic task cmd only)
- [255:224] Time (us) between task body launches (Execute periodic task cmd only)

Each argument is:
- [7  :0  ] Flags
  *  0x00 - BRAM
  *  0x01 - Private
  *  0x02 - Global
  *  0x10 - Enable input copy to wrapper BRAM
  *  0x20 - Enable output copy from wrapper BRAM
  * bit7 is internally used by cmd In module to store whether the input copy has been optimized or not.
- [31 :8  ]
- [63 :32 ] Argument ID
- [127:64 ] Argument value

Figure 5.2.3 Execute task and Execute periodic task commands format.

Finally, Figure 5.2.4 shows the Finished task command format. As it can be seen this command is simpler as it only sends the task identifier information. This information is used by the task creator (runtime running in the host) to keep track of possible dependencies and could also be used by the FTS to identify the accelerator that has finished.

```
 63                                                              0
+-----------------------------------------------------------------+
| Valid |                                                | Code   |
+-----------------------------------------------------------------+
|                      Task Identifier                            |
+-----------------------------------------------------------------+
<---------------------- 64 bits - 8 bytes ----------------------->
```

- [7 :0 ] Command code (value fixed to `0x03`)
- [55 :8 ]
- [63 :56 ] Valid Entry
  * 0x00 - Invalid
  * 0x80 - Valid
- [127:64 ] Task identifier sent to the accelerator in the execute task command

Figure 5.2.4 Finished task command format.

## 5.3 Data reuse optimizations

In order to reduce the amount of data to be accessed by the accelerators, FTS includes an automatic detection of data reuse among tasks that are waiting in the command in queue. FTS can detect if two consecutive tasks in the command in queue are re-using the same input data. In that case, it can deactivate the copy flag of the argument to be reused of the second task before this task is issued to the accelerator. Therefore, the accelerator will only need to copy data that is not already in its local memory.

Figure 5.3.1 shows two execution traces of an application when data reuse is deactivated or activated. This application has been annotated with FPGA tasks using OmpSs@FPGA and has been cross-compiled for and executed on a Zynq 7000 family board (two Cortex-a9 at 666MHz + FPGA running at 100Mhz) as a proof of concept. This is using two different versions of the FTS mentioned above to coordinate the two accelerators (IPs) and the software running on the two cores in the SMP, showing that FTS IP will be able to interface with cores and other IPs in the project. Horizontal lines in the trace show the states (different colors) on the SMP threads (two lines on the top of each execution trace) and two accelerators (two lines on the bottom of each execution trace), along the time.

Task execution in an accelerator has, with no optimizations, three states (colors): copy in data (first starting with a flag - olive green), kernel execution (second - dark olive green) and the last one copy out data (brown).

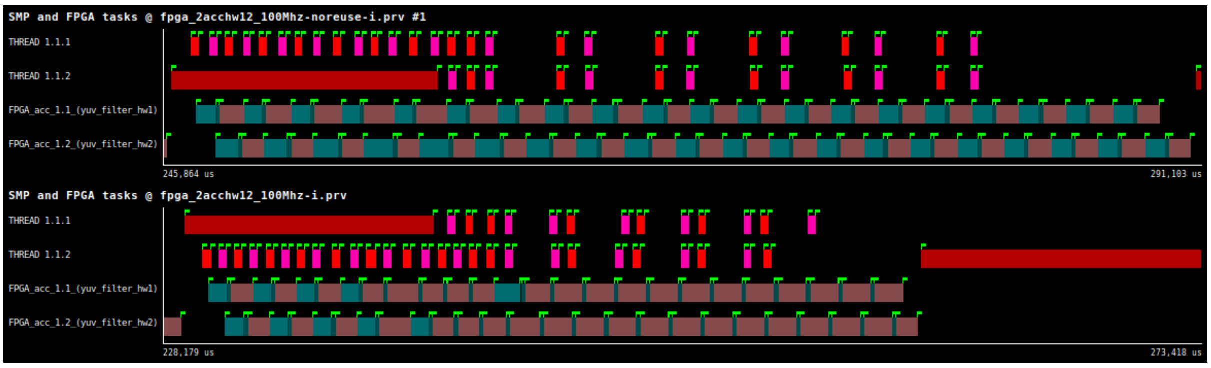
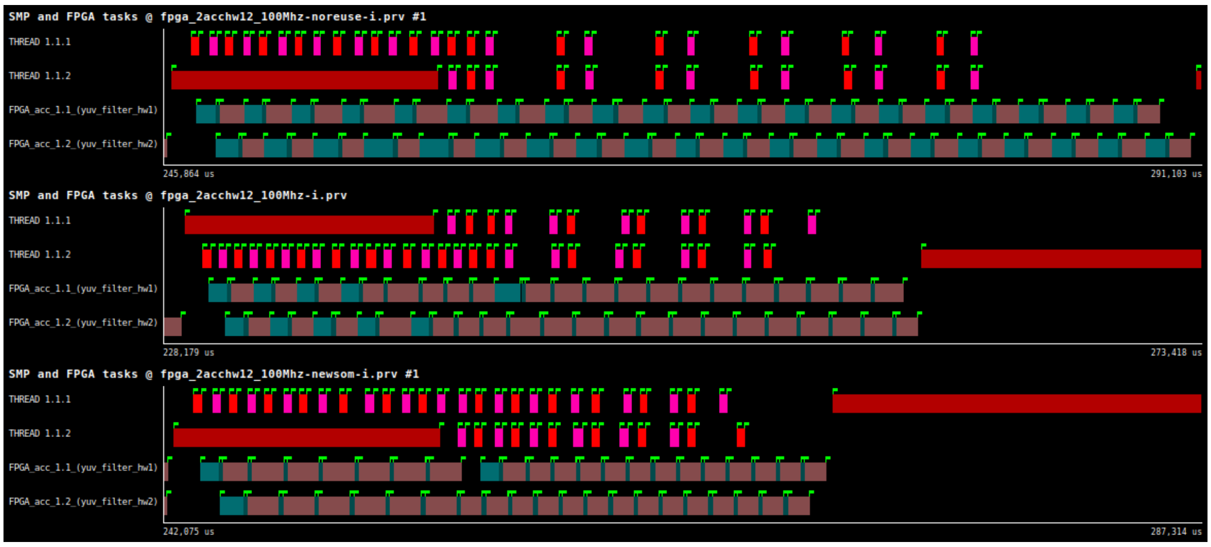


Figure 5.3.1 Execution traces of an application using two accelerators. Execution traces show the same time duration. Top: FTS has data reuse deactivated. Bottom: FTS has data reuse activated for tasks in the Command in queue.

On the execution trace on the top of the figure we can see that there are always three states (different states start and end with flags), which is not ideal. Those tasks are always re-using the same input vector but the accelerator is not conscious about this fact and is copying the input vector all the time. On the other hand, the execution trace on the bottom shows the performance achieved once FTS includes the data reuse feature. In this case FTS can automatically detect data to be reused in an accelerator and help to almost remove all input copies modifying the argument copy flags of the task descriptions.

Note however that there are still tasks in the execution trace on the bottom of Figure 5.3.1 that have three states and no data reuse is detected. This happens because originally data reuse detection among the tasks is only performed among tasks waiting in the Command In queue and no detection is done between a task being executed and tasks that arrive later to the Command In queue. This situation may happen in several applications: a task is submitted (first one) by the runtime, it immediately starts execution in the accelerator, and then, another task is submitted by the runtime. Since the first one has already started, no detection can be done between Command In queues commands. This can be solved by taking care of the task being executed in the accelerator at that moment. FTS has been improved to detect and be able to catch this situation. This can be seen in Figure 5.3.2. The execution trace on the bottom incorporates that feature. Only the first task of all tasks being executed has to copy the data, significantly improving the first FTS version (no data reuse) and allowing first task executing-second task in Command In queue data reuse. The extra-copy seen in the execution trace of the Figure 5.3.2 (bottom) is because the accelerator was completely empty when a new task was submitted. The overall performance improvement with data reuse can be significant as it can be seen in Figure 5.3.2.


SMP and FPGA tasks @ fpga_2acchw12_100Mhz-noreuse-i.prv #1
THREAD 1.1.1
THREAD 1.1.2
FPGA_acc_1.1_(yuv_filter_hw1)
FPGA_acc_1.2_(yuv_filter_hw2)
245,864 us
291,103 us
SMP and FPGA tasks @ fpga_2acchw12_100Mhz-i.prv
THREAD 1.1.1
THREAD 1.1.2
FPGA_acc_1.1_(yuv_filter_hw1)
FPGA_acc_1.2_(yuv_filter_hw2)
228,179 us
273,418 us
SMP and FPGA tasks @ fpga_2acchw12_100Mhz-newsom-i.prv #1
THREAD 1.1.1
THREAD 1.1.2
FPGA_acc_1.1_(yuv_filter_hw1)
FPGA_acc_1.2_(yuv_filter_hw2)
242,075 us
287,314 us
Figure 5.3.2 Execution traces of an application using two accelerators. Execution traces show the same time duration. Top: FTS has data reuse deactivated. Middle: FTS has data reuse activated for tasks in the Command in queue. Bottom: FTS has data reuse activated for tasks in the Command In queue and tasks being executed.

## 5.4 Test and performance evaluation strategies

In order to verify that the Fast Task Scheduling IP complies with the functionality requirements described in this deliverable it has been extensively tested, both as a standalone IP and as a part of the OmpSs@FPGA framework. The IP details are described in Deliverable 2.10 while some of the initial testing as a part of the OmpSs@FPGA framework is described in Deliverable 4.1. It is important to highlight that at the moment of delivering this document the IP is being used in a production-ready environment, so the IP is fully functional and meets the initial requirements.

The future work to be done with the IP will be driven by the performance results obtained in the evaluation and by the framework features that will be developed as the IP needs to be fully compatible with the implementation requirements of these features (like power measurement and control).

# 6. Conclusions

This document reports the activities done by TEXTAROSSA [13] partners CINI (UNIPISA), INFN and BSC with reference to the consolidated specifications of accelerator IPs in WP2 and preliminary design and synthesis results, manly in FPGA technology.

CINI UNIPI in Section 2 has presented the specification and preliminary design results of accelerators with mixed-precision (using fixed, float and posit formats), for data compression and for efficient computation of DNN (Deep Neural Network).

CINI UNIPI in Section 3 has presented the specification and preliminary design results of accelerators for innovative security services based on Post Quantum Cryptographic (PQC) techniques taking into account the NIST standardization effort. The proposed accelerator will be useful also for homomorphic encryption where SW libraries from Microsoft have been proposed already in the market.

In section 4 the specifications for both the intermediate and final releases and preliminary design of the INFN Communication IP  enabling low-latency interFPGA and intraFPGA communications are also presented .

The IP for Fast Task Scheduling has been presented in section 5. The complete documentation of the IP can be found in Deliverable 2.10 with a complete schematic, code and verification of the IP. The information presented in this deliverable summarizes the high-level functionality of the Fast Task Scheduling IP, how it can be used from a host software driver and the advanced data optimizations introduced in the hardware.

The proposed IPs are interesting, also in view of synergies between TEXTAROSSA and the other initiatives like EPI and the European Pilot, since all the proposed accelerators can be integrated with RISC-V computing cores like the RISC-V 64b Ariane IP and the RISC-V with support of the Vector extension in the EPAC (European Processor Accelerator).

# 7. References

**General**

[1] Chen, H., Laine, K., & Player, R. (2017). Simple Encrypted Arithmetic Library - SEAL v2.1. Financial Cryptography Workshops.
[2] PALISADE. https://gitlab.com/palisade. New Jersey Institute of Technology (NJIT).
[3] Halevi, S., & Shoup, V. (2020). HElib design principles. Tech. Rep.
[4] Natarajan, D., & Dai, W. (2021). SEAL-Embedded: A Homomorphic Encryption Library for the Internet of Things. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(3), 756–779. https://doi.org/10.46586/tches.v2021.i3.756-779.
[5] https://github.com/microsoft/SEAL-Embedded.
[6] https://github.com/pulp-platform/pulpino.
[7] https://github.com/openhwgroup/cva6.
[8] Dworkin, M. (2015), SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD.
[9] Deadlock-free message routing in multiprocessor interconnection. Seitz, W. J. Dally, and C. L. 1987. 5, s.l. : Computers, IEEE Transactions on, 1987, Vol. C.36, p. 547–553.
[10] A New Computer Communication Switching Technique. P. Kermani and L. Kleinrock, Virtual Cut-Through: Comput. Networks 3 (1979) 267.
[11] APEnet+ 34 Gbps Data Transmission System and Custom Transmission Logic. Ammendola, R et al JINST 8 C12022
[12] Bosch J., Vidal M., Filgueras A., Jiménez-González D., Álvarez C., Martorell X., Ayguadé E.: Task-Based Programming Models for Heterogeneous Recurrent Workloads. ARC 2021: 108-122
[13] E. Commission, "Grant Agreement 671668 - TEXTAROSSA: exploring Manycore Architectures for Next-GeneratiOn HPC systems." 2021.
[14] Marco Cococcioni, Federico Rossi, Emanuele Ruffaldi, & Sergio Saponara, Benoit De Dinechin, "Novel Arithmetics in Deep Neural Networks Signal Processing for Autonomous Driving: Challenges and Opportunities", IEEE Signal Processing Magazine, Volume: 38, Issue: 1, 2021
[15] https://digilent.com/reference/programmable-logic/genesys-2/start
[16] 754-2019 - IEEE Standard for Floating-Point Arithmetic | IEEE Standard | IEEE Xplore, https://ieeexplore.ieee.org/document/8766229