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

WP2 New accelerator designs exploiting mixed precision

D2.7 IP with data compression, part 2

V1.0

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: D2.7 IP with data compression, part 2

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	D2.7
WP No:	WP2
WP Leader:	CINI-UNIPI
Due date:	M30
Delivery date:	30/11/2023

Dissemination Level:

PU	Public	X
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)	

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

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:	D2.6					
Deliverable title:	IP with data compression, part 2					
Due date:	M30					
Actual submission date:	02/12/2023					
Editor:	Sergio Saponara					
Authors:	S. Saponara, F. Rossi					
Work package:	WP2					
Dissemination Level:	Public					
No. pages:	23					
Authorized (date):	30/11/2023					
Responsible person:	Sergio Saponara					
Status:	Plan	Draft	Working	Final	Submitted	Approved

Revision history:

Version	Date	Author	Comment
0.1	2023-10-31	S. Saponara	Draft structure
0.2	2023-11-13	F. Rossi	First version completed
0.3	2023-11-14	S. Saponara	Sent to internal review
1.0	2023-11-21	S. Saponara	Integrated comments review from internal review (E. Boella)

Quality Control:

Checking process	Who	Date
Checked by internal reviewer	Elisabetta Boella	2023-11-16
Checked by WP Leader	Sergio Saponara	2023-11-21
Checked by Project Coordinator	Massimo Celino	2023-12-01

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
AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
CINI	Consorzio Interuniversitario Nazionale per l'Informatica
CPU	Central Processing Unit
DNN	Deep Neural Network
FFT	Fast Fourier Transform
FP32	Floating Point 32 bit
FPGA	Field Programmable Gate Array
GF	Global Foundry
GTSRB	German Traffic Road Sign Benchmark
HDL	Hardware Description Language
HW	Hardware
HPC	High-Performance-Computing
IP	Intellectual Property
IPR	Intellectual Property Rights
ISA	Instruction Set Architecture
KPI	Key Performance Indicator
LUT	Look-up table
ML	Machine Learning
PDE	Partial Differential Equation
POLIMI	Politecnico Milano
PPU	Posit Processing Unit
RISC	Reduced Instruction Set Computer
SOC	System on Chip
SW	Software
TAFFO	Tuning Assistant for Floating-Point to Fixed-Point Optimization
TSMC	Taiwan Semiconductor Manufacturing Company
UART	Universal Asynchronous Receiver Transmitter interface

VPU	Vector Processor Unit

Executive Summary

The document D2.7 reports the activities done by Textarossa partner CINI (UNIPISA), with reference to the HDL design, verification, and synthesis of accelerator macrocells in WP2 for IP with data compression.

The HDL design has been carried out in SystemVerilog, while verification has been performed via simulations in a SystemVerilog test environment using a Python model as golden reference model, and via emulation in FPGA platforms (synthesis by Vivado on Xilinx FPGA devices).

The IP with data compression, called Light PPU (Posit Processing Unit) according to what foreseen in D2.1, has been implemented in FPGA technology.

Moreover, it has been integrated with a RISC-V open core like the Ariane RISC-V 64 bits to demonstrate the feasibility of integrating the Light PPU within a SoC based on RISC-V.

The IP with data compression in D2.7 is designed according to the specifications defined in D2.1 [1], and completes the design and verification done at preliminary level in D2.6 [2].

The main difference with D2.6 is the extension of verification and presentation of more results (not only ML and AI but also FFT, Sobel filters, Black-Scholes PDE), and the comparison with other data arithmetic techniques such as Double float and BFloat.

The IP can be accessed at:

https://bitbucket.org/federicorossifr/ppu_public/src/master/

As an innovation with respect to D2.6, the SW library CppPosit has been made available as open library in https://github.com/federicorossifr/cppposit

The proposed work has been also released to project partners, like CINI-POLIMI, to be used for mixed-precision compiler tools like TAFFO (https://taffo-org.github.io/)

.

1. Introduction

This document D2.7 reports on the activities done by Textarossa partner CINI (UNIPISA) in WP2.

In D2.7 the HDL design of an IP for data compression that exploits the data compression capability of Posits has been carried out in SystemVerilog, while verification has been done via simulations in a SystemVerilog test environment using a Python model as golden reference model, and via emulation in FPGA platforms (synthesis by Vivado on Xilinx FPGA devices).

D2.7 follows the revised D2.1 specifications [1] and completes the preliminary design in D2.6 [2].

The main innovation vs. what is already available in the state of art and in other EuroHPC projects like EPI is represented by the hardware support of a new arithmetic format called Posit that, particularly for ML and DNN applications, has been proved in recent literature to have a high compression effect [3-5].

Posit data format can ensure a compression gain for the same quality of floating point 32 (FP32), where quality is measured, as example, for a detection or classification task as accuracy (i.e. ratio between correct detection/classification and the total number of detection/classification done).

The main difference between D2.7 and D2.6 is the extension of the verification and presentation of more results and comparison with other data arithmetic techniques (see Section 5 of this report).

This deliverable is organized as follows:

Section 2 discusses Posit numbers and a SW library implementing them called CppPosit, focusing on their data compression properties vs floating point numbers.

As an innovation with respect to D2.6, the SW library CppPosit has been made available as open library in https://github.com/federicorossifr/cppposit

Section 3 discusses in Subsection 3.1 the RISC-V Posit Instruction Set Architecture design and implementation to support, thanks to the Light PPU IP, at minimal overhead the translation from/to Posits and Floats, and from/to Posits and integers. Achieving this data-arithmetic transformation capability ensures that the computation of a DNN can be done using conventional ALU and FPU, but the storage of the DNN can be done more effectively in Posit format.

Subsection 3.2 presents hardware results obtained implementing the IP called Light PPU in FPGA technology according to the specifications defined in D2.1, and then verified and integrated with a Ariane RISC-V 64 bits processor core.

In Section 4 functional benchmarks for several examples are shown and are compared to other arithmetics at the state of art.

Conclusions are drawn in Section 5, where also the IP repository is listed.

2. Posits and CppPosit SW Library

A posit number is represented by a signed integer on 2's complement (see Fig. 2.1). It can be configured with the total number of bits N and maximum number of exponent bits ES. We define such a posit as Posit ⟨N, ES⟩.
The Posit format can have at most four fields: i) sign on 1-bit, ii) regime with a variable size (run-length encoded), iii) exponent with at most ES bits and iv) fraction with a variable length.
An example of a posit number instance is shown in Fig. 2.1. Note that, if the regime field is large enough, it is possible that the exponent field has less bits available than ES. In this case the actual exponent value is computed by padding zeros to the right of the exponent bits in the format.

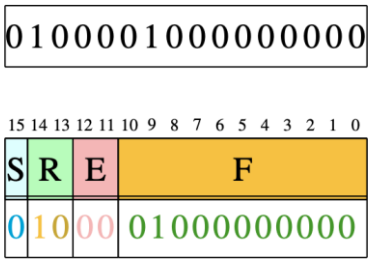

Figure 2.1: example of a Posit number

The length l of the regime corresponds to the number of identical bits following the sign bit:

$$s, \underbrace{b_1, b_2, b_3, \ldots, b_l}_{=b}, \underbrace{b_{l+1}}_{=\bar{b}}, \cdots$$

The regime length is l. Depending on the value of the bit b, the regime value k will be computed as follows:

$$k := \begin{cases} l-1 & \text{if} \quad b = 1 \\ -l & \text{if} \quad b = 0 \end{cases}$$

The regime value is a scale factor for a special constant that depends on the posit configuration, called useed. The useed value is computed as follows

$$\text{useed} := 2^{2^{ES}}$$

Hence, the real value r associated with a posit represented by the integer P on two's complement (with sign s) is computed as in the following Equation:

$$r := \begin{cases} 0 & \text{if } P = 0 \\ \text{NaR (Not a Real)} & \text{if } P = -2^{N-1} \\ (-1)^s \times \text{useed}^k \times 2^e \times \left(1 + \frac{f}{2^F}\right) & \text{otherwise} \end{cases}$$

The value F is the length of the fraction field.
Note that there will always be an implicit one in front of the fraction (i.e. 1.f1, f2, . . . , fF), without any subnormal number differently from IEEE binary32 numbers.

Software support for posits is enabled by our cppPosit library , developed in Pisa and maintained by the Pisa authors, see here the public and open source part https://github.com/federicorossifr/cppposit

The library uses templatization to define different posit configurations during compilation. The posit operations are put into four different levels L1-L4 with increasing computational complexity.

The main features of the cppPosit library are:

- Template-based posit definition and configuration
- Compile-time selection of different backends (software or hardware based)
- Use of operator overloading to hide backends implementation detail to end-user
- Ability to seamlessly target hardware accelerators or co-processors with operator overloading directly on the application code.

Thanks to this library we can transparently use the lightweight PPU in a neural network framework without changing the application code; indeed, it is sufficient to re-compile the application with a proper flag to enable the use of newly inserted RISC-V instructions.
The SW library CppPosit has been made available as an open library in https://github.com/federicorossifr/cppposit and its use is described in the following publication:

E. Ruffaldi, F. Rossi, M. Coccocioni, S. Saponara, “cppPosit: a C++ template-based library implementing Posit arithmetic ready for machine learning applications”, submitted to JMRL (Journal of Machine Learning Research)

3. Light PPU IP, interfacing to RISC-V and HW FPGA implementation results

3.1 Light PPU IP design

The goal of this work is the design of an IP core for lightweight PPU to be connected to a 64b RISC-V processor in the form of a co-processor with an extension of the Instruction Set Architecture.

Please note that the theory of Posit arithmetic, the structure of Posit numbers and their compression data benefit vs. classic integer and floating-point formats, particularly for DNN computation, have been already discussed in published works such as [3-5]. Therefore, the goal of this deliverable is describing the design and verification of digital IPs supporting data compression for DNN thanks to the light support of Posits.

We focus on the compression abilities of posits by providing a co-processor with only a conversion in mind, called light PPU (as in Figure 3.1 below).

The difference between D2.6/D2.7 vs D2.2/D2.3 is that D2.2 and D2.3 present an AI accelerator IP giving full hardware support to posits number and hence using the IP in D2.2 and D2.3 the FPU of a RISC-V processor can be replaced by the Full PPU.

Instead, in D2.6/D2.7 the idea is minimizing the circuit complexity overhead and hence the proposed IP supports only the data compression using Posits with Float to/from Posit translation, but operations have to be done still in the FPU.

We can convert binary32 floating point numbers to posit numbers with 16 and 8 total bits (and a variable number for the exponent since Posit 16,0 and Posit16,1 is considered).

In order to provide a circuit design for our light-PPU we considered several key points to simplify the final logic design:

1. IEEE floating point values are encoded in a module and sign-like representation while posits are encoded using 2's complement representation. Therefore, when converting from IEEE floats we just ignore the sign and build the positive posit. Then we use the sign to apply the 2's complement to the result if negative.
2. Given a Posit<16,0> the size of the regime spans from a minimum of 2 to a maximum of 15 bits. As a result, the mantissa size spans from a minimum of 0 to a maximum of 12 bits. This means that, given a 23-bit mantissa IEEE Float, the 8 least significant bits of the float are set to 0. The same concepts hold for Posit<8,0>.
3. We can build the Posit<X,0> regime arithmetically shifting an appropriate value by the X least significant bits of the FP32 normalized exponent. For Posit<16,0> we shift the signed integer represented by 2^{15} or $(8000)_{16}$ in hexadecimal notation. For Posit <8,0> we shift the signed integer represented by 2^{7}.
4. Decoding the regime is particularly interesting since we need to employ a *find first set* module (or *find first unset)* to evaluate the regime length. The output of the *find first set* module is the index *i* of the highest set bit (discarding the sign if present). For Posit<16,0> the regime length is actually computed as *l=14-i.*

Figures 3.1, 3.2 and 3.3 show simplified versions of the light-PPU architecture.

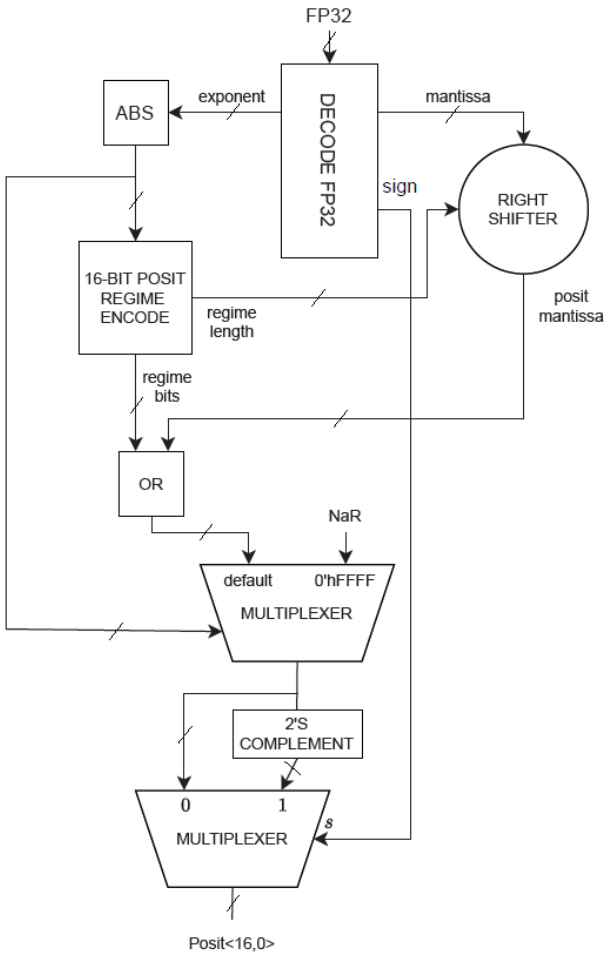

Figure 3.1: Logical circuit for the 32-bit floating point to posit16,0 converter

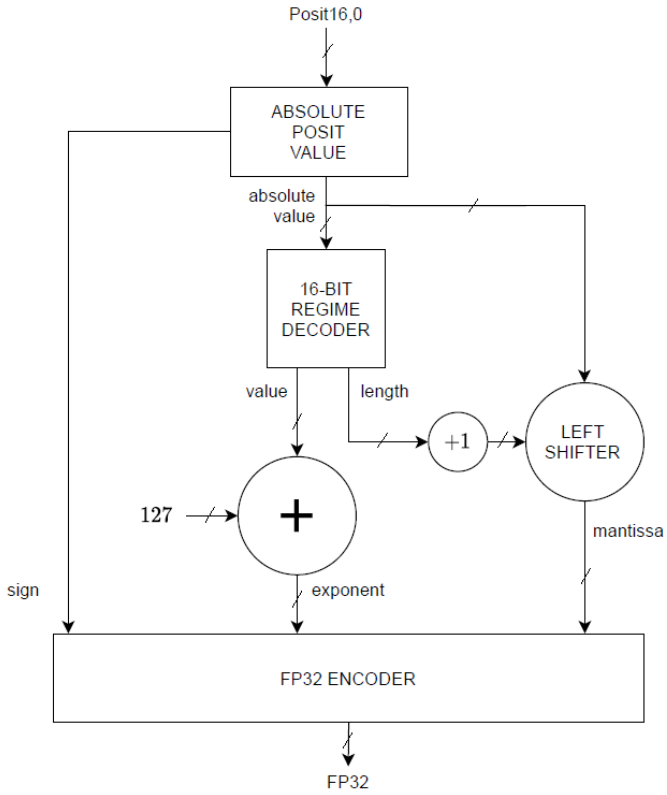

Figure 3.2: Logical circuit for the posit16,0 to 32-bit floating point converter.

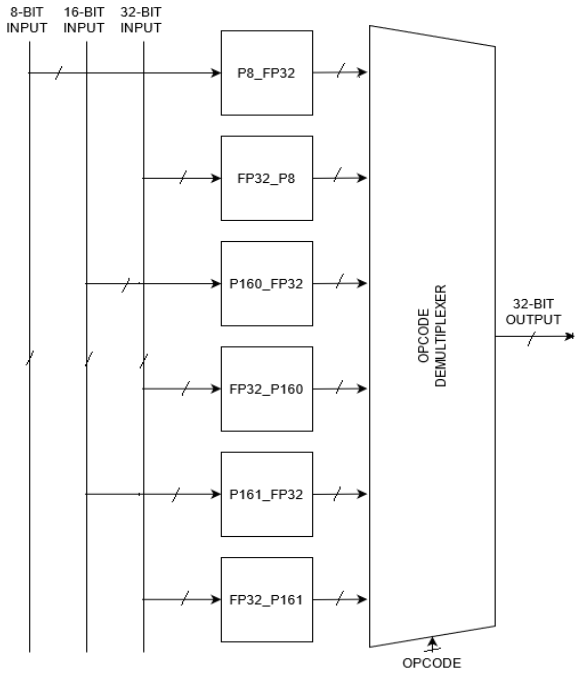

Figure 3.3: Overall architecture for the light PPU

3.2 Integration with RISC-V CPUs Ariane 64bits and FPGA HW implementation results

As shown in Figure 3.4, the proposed co-processor can be paired with a RISC-V core that already has a floating-point unit (e.g., the Ariane 64b RISC-V) without interrupting the existing pipeline.
On the other hand, we can use this unit to enable ALU computation of posit numbers with the posit-to-fixed conversion modules on a RISC-V core that does not support floating-point:

1. If the RISC-V processor embeds an FPU the light PPU can be used as a wrapper, providing a data compression by a factor up to 4, with little accuracy degradation. The cost of compression and decompression is the cost of converting a posit to a float and vice-versa.
2. If the RISC-V processor does not embed an FPU or we want to exploit only the ALU, the light PPU can function as a wrapper of fixed-point representation. Indeed, once we have converted between posit and fixed-point, the basic arithmetic operations can be computed just with the ALU. Also note that, for half of the posit domain, that is the [-1,1] range, the conversion between posit and fixed point is a simple left shift of 2 positions followed by a 0 padding on the most significant bits to reach desired fixed-point size.

We investigated the first use-case by outfitting a CVA6 core with our PPU co-processor and synthesizing it for a Xilinx Genesys 2 FPGA, resulting in a working RISC-V core capable of running a general-purpose Linux distribution.
The integration within the RISC-V core (Ariane CVA6 code, [7, 8]) can be done using the possibility to customize the instruction set.
The posit-based compression IP proposed in D2.6 can be integrated in addition to the Ariane integer ALU and the Ariane FPU.

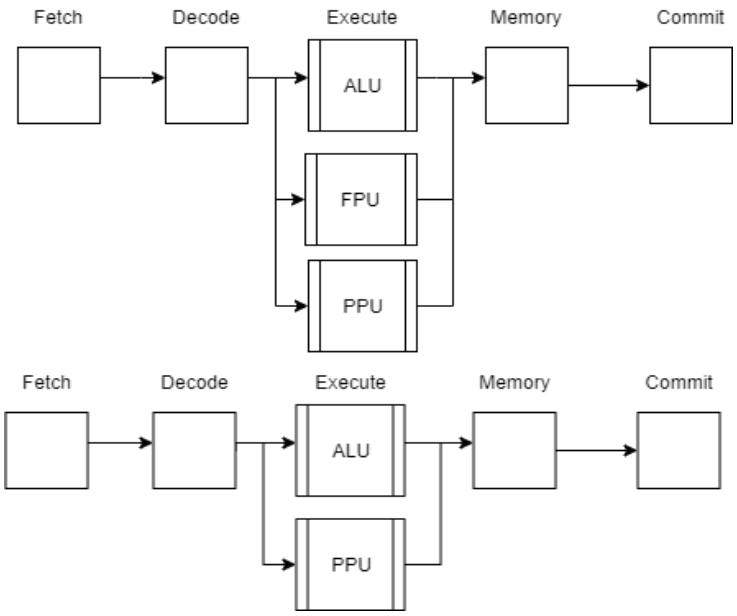

Figure 3.4: PPU possible integration modes within the RISC-V instruction set (with/without the FPU)

Table 3.1 Shows the implemented RISC-V instructions and opcodes for the new light PPU instructions. Note that we are leveraging the custom opcode space in the 6 least significant bits **0x0b.**

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
1100000		00010	rs1	000	rd	0001011	FCVT.S.P8
1100000		00011	rs1	000	rd	0001011	FCVT.S.P16.0
1100000		00011	rs1	010	rd	0001011	FCVT.S.P16.1
1101000		00010	rs1	000	rd	0001011	FCVT.P8.S
1101000		00011	rs1	000	rd	0001011	FCVT.P16.0.S
1101000		00011	rs1	010	rd	0001011	FCVT.P16.1.S
1100000		00010	rs1	001	rd	0001011	FXCVT.H.P8
1100000		00011	rs1	001	rd	0001011	FXCVT.W.P16.0
1100000		00011	rs1	011	rd	0001011	FXCVT.L.P16.1
1101000		00010	rs1	001	rd	0001011	FXCVT.P8.H
1101000		00011	rs1	001	rd	0001011	FXCVT.P16.0.W
1101000		00011	rs1	011	rd	0001011	FXCVT.P16.1.L
1101000		00010	rs1	001	rd	0001011	FXCVT.P8.H
1101000		00011	rs1	001	rd	0001011	FXCVT.P16.0.W
1101000		00011	rs1	011	rd	0001011	FXCVT.P16.1.L
1100000		00010	rs1	100	rd	0001011	FCVT.P8.P16.0
1100000		00011	rs1	100	rd	0001011	FCVT.P16.0.P8
1101000		00011	rs1	111	rd	0001011	FCVT.P16.1.P16.0
1101000		00010	rs1	101	rd	0001011	FCVT.P16.1.P8
1100000		00011	rs1	110	rd	0001011	FCVT.P8.P16.1
1101000		00011	rs1	101	rd	0001011	FCVT.P16.0.P16.1

Table 3.1: ISA extension for the light PPU with conversions between 16-bit/8-bit posits and FP32/Fixed

For the HW implementation and characterization in FPGA technology of the compressor IP, we chose the Xilinx Genesys 2 board which is equipped with a Kintex 7 XC7K325T-2FFG900C FPGA component.
We chose this board to reduce the work required to construct our PPU inside a RISC-V core. We did, in fact, use the Ariane RISC-V core that was originally built for this board.
The resulting design was then implemented on the same board.
This approach of using the same prototyping board also facilitates a fair comparison of the RISC-V without the Light PPU and the RISC-V with the Light PPU.
The Genesys 2 board can be connected to a host controller for configuration and diagnosis via an UART interface.

For the PPU component, we performed power, circuit complexity, and propagation delay (worst case combinatorial propagation delay of the PPU) reports:
1. Look-up table (LUT) utilization: 747/203800 (0.36%) LUTs used.
2. Component latency: 6.332ns (worst propagation delay).

Finally, the new instruction set architecture was merged into the Ariane RISC-V core and synthesized for the Xilinx Genesys 2.
The following KPIs were obtained:

1. Clock frequency: 125MHz (this corresponds to the original Ariane design on the same board, so adding the Light PPU does not change the maximum achievable frequency)

2. Total power on FPGA components (Kintex 7): 2.056W of which the contribution of the Light PPU is less than 1%.

3. Look-up table (LUT) utilization: 63805/203800 (31.54%) LUTs used, of which only 0.36% is due to the added Light PPU.

To be noted that the target of Textarossa is the implementation of the IP on FPGA, not on a ASIC, and hence the KPI of keeping the same frequency of the original Ariane design in the range of hundreds of MHz for FPGA is a good result.

An ASIC implementation can be part of future evolutions of this IP design but is out of scope for Textarossa. To this aim, the output of Textarossa can be useful to other research projects like EPI where the design and implementation of ASICs in nanoscale technologies like GF12nm or TSMC 6 nm is foreseen.

Moreover, the work on Textarossa refers to single line posit processing units. A vectorization of the Posit operators could be evaluated as a future extension of this work to become compliant with vectorized RISC-V instructions thus creating a Vector Processing Unit (VPU).

We validated the new ISA assuring the coherency between the two versions of cppPosit, compiled with or without the light PPU support. In the former case, the ISA instructions were emulated in the Spike simulator and the operation results were compared to the latter case in which posit operations were implemented completely in cppPosit.
We developed a test-bench using the same HDL language used for the light PPU unit to test the functionality of the newly introduced components. We tried every combination for the inputs and tested the outputs of conversion against our cppPosit software library. This means that we have also verified the correct handling of all the corner cases, such as the redundant negative zero and the redundant representations for the Infinity and Not-A-Number in IEEE 32-bit floats.

4. Benchmarks and verification results

To assess the variation of accuracy with compressed formats, we tested the IP using:

- two well-known datasets for AI applications: the MNIST dataset and the GTSRB datasets on the networks (Figure 4.1); these datasets are often used in literature for benchmarking AI applications [2, 5, 6 10]
- classic operators like Sobel filters, Fast Fourier Transform (FFT), and Black-Scholes (a Partial Differential Equation used in economics) for other signal processing applications.

For the AI application test, we used the neural network in Fig. 4.1 (LeNet-5) as benchmark base for weight compression.

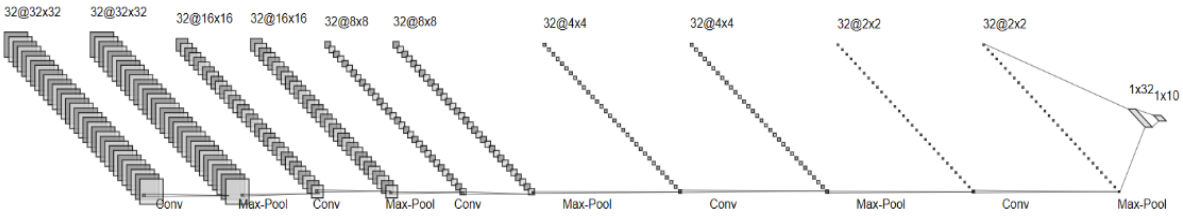

Figure 4.1: example neural network used for test of the light PPU data compression IP

We also report accuracy results on this neural network in Table 4.1 utilizing both posit and binary32 integers with varying workloads for completeness. This result was reached by using posit compression for the weights, according to the IP described in previous Sections, and computing using binary32 (fp32) format. We measured the relative speedup to a software-based posit compression. As a result, we investigated the overall system compression times with the weights of a tiny LeNet-5 neural network, yielding the result displayed in Table 4.1.

LeNet		
	MNIST	**GTRSB**
FP32	98.83%	91.8%
posit(16, 1)	98.83%	91.8%
posit(16, 0)	98.50%	90.5%
posit(8, 0)	98.34%	90.4%

Table 4.1: Accuracy performance of Posit vs FP32 for different benchmark data sets (MNIST [6] and the German Traffic Road Sign Benchmark [7])

To assess the performance of the customized core we ran the same benchmarks used in the simulation phase on the Ariane RISC-V core, equipped with the OpenPiton 12 Linux distribution (based on the Ariane Linux 4.2).
As before, timing performance was measured using C++ internal software *chrono* directives. This approach involves the conversion between posits and floats at each operation (e.g. sum or multiplication). As a result, for each operation, we are performing two more instructions for type conversion. Table 4.2 and Table 4.3 summarize the results obtained with the evaluation of this trade-

off. Note that the value obtained with IEEE FP32 is totally independent from the presence of the light PPU.

	w/ PPU (s)	wo/ PPU (s)	Speedup
posit(8,0)	5.4	58.87	10.90
posit(16, 0)	11.6	64.54	5.56

Table 4.2: Real HW (FPGA) timing performance on a 10-layer convolutional neural network with and without the synthesized hardware PPU light support for the cppPosit library.

	Time (s)	DNN size (bytes)	Compression
IEEE FP32	2.1	224894	-
posit(16, 0)	11.6	112874	1.99
posit(8, 0)	5.4	56864	3.95

Table 4.3: Tradeoff between processing time and compression factor of Posit vs FP32 for DNN.

We may instead take an entire posit network and convert it beforehand to IEEE FP32 in order to exploit compression as much as possible without slowing down actual image processing. This use case is relevant if we think about resource constrained environments where volatile memory is scarce (e.g. embedded or automotive systems) or when frequent transfer of network models are done (e.g. smartphones with recognition software). Moreover, these systems often use an adaptive approach where, depending on the surrounding environment (e.g. snow, night-time, off-road etc.), different machine learning models need to be loaded. This means that having multiple models on volatile storage can highly benefit from compression even when they are not actually loaded into main memory. In this case, we need to decompress the network into IEEE FP32 format only once at the beginning of the execution. This will lead to a much slower start but a faster computation time (that is the same as IEEE FP32 in Table 4.3). Table 4.4 summarizes the results of this evaluation.

	Time (s)	DNN size (bytes)	Compression
IEEE FP32		224894	
posit(16, 0)	51.5	112874	1.99
posit(8, 0)	49.8	56864	3.95

Table 4.4: Trade-off between compression time of the network in Figure 4.1 and compression factor. Processing time was obtained from the real hardware implementation.

Other benchmarks have been carried out together with the partner POLIMI using the TAFFO mixed-precision compiler, targeting the Posit instruction offered by the posit processing unit. Exploiting TAFFO we targeted 8-bit and 16-bit posits for several benchmarks in linear algebra. Figure 4.2 shows some results on using said benchmarks with compressed real arithmetic formats such as 8-bit and 16-bit posits.
Figure 4.2 reports relative error vs. double floating point numbers (64 bits).
These results show that:
- using Posit16 the error vs 64-bit doubles is 1% or below for 7 cases out of 11 cases considered and in the worst case is anyway below 10%.
- using Posit16 the error vs 64-bit is always lower than using Bfloat16.
- using Posit32 the error vs 64-bit is always lower than 1% and can be as low as 10^{-7} for FFT 2048 points.

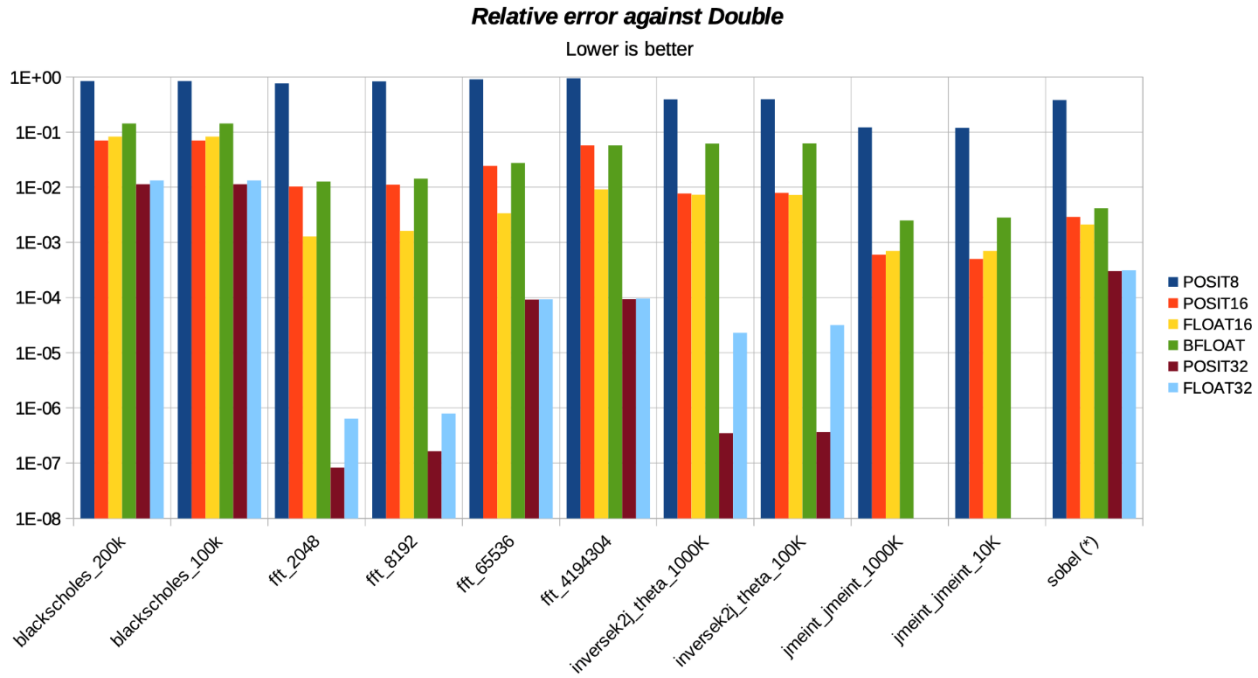

Figure 4.2 Relative error of different precision formats against the 64-bit double counterpart.

5. Conclusions, IP repository and dissemination

In this work we dealt with the finalized HDL design, using SystemVerilog, verification and synthesis of an IP for data compression exploiting posit arithmetic.

The work in D2.7 completes, particularly in terms of verification and comparison to state of art, the work done at preliminary level in D2.6.

The key novelty is the hardware support for a new arithmetic format called Posit, which has been shown to have a good compression effect, notably for ML and DNN applications: up to 4x compression for the same quality of floating point.

In D2.7 the verification has been extended also to other classic operations such as FFT (from 2K to 4M points), Sobel filtering, Black-Scholes equation: in such cases, a compression factor up to 2x vs. double format is achieved with a relative error from 10^{-7} to 10^{-2} while a compression factor up to 4x vs. double format is achieved with a relative error from $7*10^{-4}$ to $8*10^{-2}$.

We designed the data compression IP using FPGA technology targeting the Xilinx Genesys 2 FPGA platform. Furthermore, we tested the IP design against a golden-model software library for posit arithmetic.

The data compression IP has been implemented in different Xilinx FPGA devices and it has been designed according to the specifications defined in D2.1, aiming to demonstrate its platform independence.

Finally, we integrated the data compression IP within the Ariane 64-bits 6-stage RISC-V core.

IP repository and dissemination

The IP can be accessed: https://bitbucket.org/federicorossifr/ppu_public/src/master/

Moreover, as an innovation vs. D2.6, the SW library CppPosit has been made available as open library in https://github.com/federicorossifr/cppposit

The proposed work has been also submitted to the review of the scientific community and has been accepted on

Marco Cococcioni; Federico Rossi; Emanuele Ruffaldi; Sergio Saponara, “A Lightweight Posit Processing Unit for RISC-V Processors in Deep Neural Network Applications”, IEEE Transactions on Emerging Topics in Computing, 2022, vol. 10, n. 4

The SW library has been also presented in the submitted paper (still under review)

Emanuele Ruffaldi, Federico Rossi, Marco Cococcioni, Sergio Saponara, “cppPosit: a C++ template-based library implementing Posit arithmetic ready for machine learning applications”, submitted to JMRL (Journal of Machine Learning Research)

The proposed work has been also released to project partners, like POLIMI, to be used for mixed-precision compiler tools like TAFFO (Prof. G. Agosta), https://taffo-org.github.io/

6. References

[1] D21, “Consolidated specs of accelerators IPs”, Textarossa project, May 2022, Revised May 2023
[2] D26, “IP with data compression, part 2”, Textarossa project, May 2022, Revised May 2023
[3] 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
[4] Marco Cococcioni, Federico Rossi, Emanuele Ruffaldi, Sergio Saponara, A Novel Posit-based Fast Approximation of ELU Activation Function for Deep Neural Networks, 2020 IEEE International Conference on Smart Computing (SMARTCOMP)
[5] Marco Cococcioni, Federico Rossi, Emanuele Ruffaldi, & Sergio Saponara. (2021). A Lightweight Posit Processing Unit for RISC-V Processors in Deep Neural Network Applications. IEEE Trans on Emerging topics in Computing, https://zenodo.org/record/7128760#.Y0ZfpC8QNbU
[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998.
[7] J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition”, Neural Networks, Available online 20 February 2012, ISSN 0893-6080, 10.1016/j.neunet.2012.02.016.
(http://www.sciencedirect.com/science/article/pii/S0893608012000457) Keywords: Traffic sign recognition; Machine learning; Convolutional neural networks; Benchmarking
[8] F. Zaruba, and L. Benini, “The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-ready 1.7GHz 64bit RISC-V Core in 22nm FDSOI Technology”, arXiv e-prints, 2019.
[9] https://github.com/openhwgroup/cva6.
[10] Marco Cococcioni; Federico Rossi; Emanuele Ruffaldi; Sergio Saponara, A Lightweight Posit processing Unit for RISC-V Processors in Deep Neural Network Applications, IEEE Transactions on Emerging Topics in Computing. 2022 | Volume: 10, Issue: 4 |
[11]
https://benchmark.ini.rub.de/gtsdb_dataset.html#:~:text=The%20German%20Traffic%20Sign%20Detection%20Benchmarkis%20a%20single-image,It%20is%20introduced%20on%20the%20IEEE%20International%20Joint