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

WP2 New accelerator designs exploiting mixed precision

D2.3 AI Accelerator with mixed-precision including Posit, 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: D2.3 AI Accelerator with mixed-precision including Posit, 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	D2.3
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

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:	D2.3					
Deliverable title:	AI Accelerator with mixed-precision including Posit, part 2					
Due date:	M30					
Actual submission date:	02/12/2023					
Editor:	Sergio Saponara					
Authors:	F. Rossi, S. Saponara					
Work package:	WP2					
Dissemination Level:	Public					
No. pages:	29					
Authorized (date):	30/11/2023					
Responsible person:	Sergio Saponara					
Status:	Plan	Draft	Working	Final	Submitted	Approved

Revision history:

Version	Date	Author	Comment
0.1	2023-11-01	S. Saponara	Draft structure
0.2	2023-11-10	F. Rossi, S. Saponara	First version completed
0.3	2023-11-12	S. Saponara	Consolidated version complete and send to internal review
1.0	2023-11-29	F. Rossi, S. Saponara	Updated after internal review

Quality Control:

Checking process	Who	Date
Checked by internal reviewer	Paolo Palazzari	November 30th,2023
Checked by WP Leader	Sergio Saponara	November 30th,2023
Checked by Project Coordinator	Massimo Celino	December 1st, 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
CINI	Consorzio Interuniversitario Nazionale per l'Informatica
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
EPI	European Processor Initiative
FIR	Floating-point intermediate format
FP32	Floating Point 32 bit
FPGA	Field Programmable Gate Array
GEMM	Generic Matrix multiplication
GF	Global Foundry
HDL	Hardware Description Language
HW	Hardware
HPC	High-Performance-Computing
INFN	Istituto Nazionale di Fisica Nucleare
IP	Intellectual Property
IPR	Intellectual Property Rights
ISA	Instruction Set Architecture
PMB	Project Management Board
PPU	Posit Processing Unit
PQC	Post Quantum Cryptography
RISC	Reduced Instruction Set Computer
SIMD	Single Instruction Multiple Data
SW	Software
TSMC	Taiwan Semiconductor Manufacturing Company

UART	Universal Asynchronous Receiver Transmitter interface
VPU	Vector Processor Unit

Executive Summary

This document reports the activities done by TEXTAROSSA partner CINI (UNIPISA), with reference to design, verification, synthesis, and characterization in FPGA technology of an accelerator IP in WP2 called Full PPU (Posit Processing Unit).

The design has been done using SystemVerilog HDL language, verification using HDL simulation and python to define a golden reference mode, synthesis in VIVADO Xilinx environment and test on Xilinx FPGA devices.

The proposed IP acts as AI accelerator by giving full hardware support to the main operations of a new arithmetic type called Posit, plus translation between Posit and conventional formats like IEEE-754 Floating point 32 bits.

The Full PPU IP has been integrated with an open-source RISC-V core, implemented in FPGA technology and tested on benchmarks related to machine learning computation.

The Full PPU AI accelerator IP is designed according to the specifications defined in D2.1 [1].

This part 2 D2.3 deliverable completes the part 1 D2.2 [7] deliverable by completing the SystemVerilog Design of the IP, its verification by both simulation and test on FPGA, verification of its integration with RISC-V open cores.

1. Introduction

This document D2.3 reports the activities done by TEXTAROSSA partner CINI (UNIPISA) in WP2.

D2.3 deals with the complete HDL design, using SystemVerilog, verification, synthesis, and characterization in FPGA technology of an accelerator IP in WP2 called Full PPU (Posit Processing unit).

The Full PPU AI accelerator IP is designed according to the specifications defined in D2.1 [1].

This part 2 D2.3 deliverable completes the part 1 D2.2 [7] deliverable by completing the SystemVerilog design of the IP, that was preliminary in D2.2 [7], its verification by both SystemVerilog simulation and test on FPGA, verification of its integration with RISC-V open cores.

Another part in D2.3 new vs. D2.2 is the definition of a Python gold model of the IP.

The main innovation of the D2.2 and D2.3 is in the hardware support of a new arithmetic format called Posit, suitable for acceleration of DNN computation. Moreover, the IP supports the translation between Posit and conventional formats like IEEE-754 Floating point 32 bits.

The Full PPU IP has been integrated with an open-source RISC-V core, implemented in FPGA technology and tested on benchmarks related to machine learning computation.

Please note that the theory of Posit arithmetic, the structure of Posit numbers and their benefits vs. classic integer and floating-point formats have been already discussed in D2.2 and by us in published works such as [2, 3]. Therefore, the goal of the D2.2 (preliminary) and D2.3 (final) deliverables is on the design and verification of digital IPs supporting Posit.

To this aim, Section 2 highlights the main difference of D2.3 vs D2.2

Section 3 presents a brief description of posit numbers, of their main properties and elaborates on the logical implementation of posit operations. A focus on the different aspects of division algorithms, that is the most complex operation to be supported by the Full PPU IP core, has been discussed in D2.2.

Section 4 shows the design at register transfer level using SystemVerilog of the Full PPU.

Section 5 describes how to interface the Full PPU to a RISC-V open core [5, 6]: this is achieved by exploiting the possibility of Instruction Set Architecture (ISA) extension of RISC-V with new opcodes that are introduced to support Posit operations. The SW support for compilation is also discussed.

In Section 6 validation is done considering both a simulation testbench (using a golden model in Python to generate input stimuli and expected results) and then an emulation testbench carried out using a prototype in FPGA technology.

To this aim, Section 7 presents the final characterization of the Full PPU, integrated with open RISC-V core, when implemented on Xilinx Alveo U280 FPGA.

The KPI used for validation are computation accuracy of the Full PPU vs FP32 operation for some benchmark functions, area occupation, clock cycles needed to implement some neural networks operations, and power consumption.

Tests are carried out considering some linear algebra tasks of interest for Convolutional and Deep Neural Network operations such as convolution, GEMM (Generic Matrix multiplication) and average pooling.

Discussions and Conclusions about possible impact are given in the last Section 8 where the link to the GitHub repository containing the Full PPU IP data base, in the final version, is also reported.

References are reported in Section 9.

2. D2.3 difference vs D2.2

D2.3 is the description of final version of D2.2, so that the structure is similar while the status of IP is more advanced being finalized in terms of design and verification:

- In the SystemVerilog design new features have been added such as: Fused Multiply&ADD operation beside single Add or Multiply operations and complete support of Float to Posit and Posit to Float operations.
- Enhancing of the Python Golden Model with support of new Fused Multiply&ADD operation and more test cases.
- Finalization of the verification with a more structured verification environment.

3. Brief description of Posit numbers and operations

A posit number is represented by a signed integer on 2's complement. 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 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,
iv) fraction with a variable length.

An example of a posit number instance is shown in Fig. 1.
The associated real value to the shown Posit is: $+16^{0}\times2^{0}\times(1+512/2048) = 1.25$.

Note that, if the regime fields 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 zeroes to the right of the exponent bits in the format.

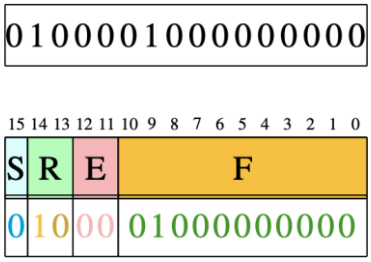

Figure 1: example of a Posit number in the configuration of **N=16** bits and **ES=2** exponent.

The length l of the regime (**excluding the stopping-bit**, 0 in the example in Figure 1) corresponds to the number of identical bits following the sign bit:

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

Depending on the value of the bit b, the regime value k will be computed as follows:

$$k := \begin{cases} l-1 & \text{if} \quad b=1 \\ -l & \text{if} \quad b=0 \end{cases} \tag{2}$$

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 where ES is the maximum number of exponent bits specified in the posit configuration (useed is 2^4=16 in Fig. 1, since **ES=2**),

$$\text{useed} := 2^{2^{ES}} \tag{3}$$

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:

$$r := \begin{cases} 0 & \text{if } P = 0 \\ \text{NaR (Not a Real)} & \text{if } P = -2^{N-1} \\ (-1)^s \times \text{useed}^k \times 2^e \times \left(1 + \frac{f}{2^F}\right) & \text{otherwise} \end{cases} \quad (4)$$

The value F is the length of the fraction field, while the value f is the integer represented by the F bits of the fraction length. In Figure 1 we have **F=11** bits for the fraction and the integer represented by these 11 bits are **f=512.**

Being in Fig. 1 s=0, k=0, useed=16, e=0, f=512, F=11 then the associated real value to the shown Posit is: $+16^0 \times 2^0 \times (1+512/2048) = 1.25$.

The benefits of Posits vs. classic integer and floating-point formats when computing DNN and CNN have been already discussed in D2.2 and in published works such as [2-4].

Concerning Posit arithmetic operations the process involves decoding and conditioning the posit operands into their sign, regime, exponent, and significand fields. During the decoding phase, certain decisions are made, such as handling special cases like when one of the operands is equal to 0 or NaR.

Once the decoding and conditioning phase is completed, each posit operand is transformed into a general floating-point intermediate format (FIR) with the format ⟨sf, te, f⟩, where sf is the sign, te is the total exponent without bias, and f is the fractional part of the significand.

$$p = (-1)^{sf} \times 2^{te} \times (1.f) \quad (5)$$

The total exponent, te, is introduced as $te = (((2^{ES}) * k) + e)$, in agreement with above Equation (4), where k can be positive, zero, or negative, and e is a non-negative integer. In Figure 1 the total exponent is: $2^2 * 0 + 0 = 0$
This intermediate representation is used to compute the intermediate result of operations between two posits. Then, the intermediate result is normalized, transforming the FIR result into a proper posit. During this last phase, the posit rounding mechanism comes into play.

Addition and subtraction operations share several portions of arithmetic and logic parts. The goal is to obtain a FIR-represented posit pout such that the sum or difference of the two input posits is correct and rounded to the nearest representable posit. By supposing that $|p1| \geq |p2|$, a scale factor b = (te1 − te2) ≥ 0 is extracted from the exponents, and the resulting equation is normalized to obtain the final 1.fout value, fout being the fractional part only (fout = $\{f_0, f_{-1}, f_{-2}, ...\}$; $f_n \in \{0, 1\}$).

For addition, it is necessary to ensure that the sum is strictly lower than 2 to be in a valid format. If fsum overflows 2, it is scaled back by one position, summing 1 to the total exponent te1.
In contrast, with subtraction, the result is normalized by subtracting the number of leading zeroes in fsum from te1 and shifting the result of the same number of positions to the left. At the end of addition, when fsum is shifted right by one position, the last bit is discarded. If the discarded bit is 1, this information is signaled outside to preserve it for the final rounding phase.

For multiplication, the goal is to find the FIR tuple ⟨sout, teout, fout⟩ such that the product of the two input posits is correct and rounded to the nearest representable posit. The resulting equation is simplified to obtain the final 1.fout value, which is the integer multiplication of the two fractions readjusted for normalization, along with the possible increment of teout.

Finally, for division, the goal is to find the tuple ⟨sout, teout, fout⟩ such that the quotient of the two input posits is correct and rounded to the nearest representable posit. The equation is simplified to obtain the final 1.fout value, which is the result of an integer division. The sign and total exponent of the output posit are calculated from the sign and exponents of the input posits.

In summary, this section describes the implementation of posit arithmetic operations in detail, emphasizing their numerical rounding and arithmetical correctness. The process involves decoding and conditioning the posit operands, transforming them into a general floating-point intermediate format, computing the intermediate result of operations between two posits, normalizing the intermediate result, and transforming it into a proper posit, while considering the posit rounding mechanism.

4. Multi-stage Full Posit Processing IP architecture.

This section presents the design of the Full PPU.
Similarly, to the previous section, we identify 3 main stages of execution inside the processing unit:

i) decoding and input conditioning.

ii) actual computation on 2 operands with Posit operations like ADD (addition), SUB (subtraction), Mul (Multiply), Div (Division). In D2.3 also a combined operation FMA: Fused Multiply-Add has been included.

iii) normalization and encoding of the result.

iv) Float to Posit (via cascade of Float to FIR and FIR to Posit operations) and vice versa, so that if the proposed PPU is used in microprocessors where the Floating-point Processing Unit is missing then Floating-point operations can be still supported by converting Floating-point numbers to Posit, making operations in the Posit domain via the FPU; and finally reconverting Posit results to Floating-point results.

To limit the length of pure combinatorial paths the FPPU has 3 pipelined stages corresponding to the 3 main stages of execution, with the computation phase split into two to consider for the longer path in the division logic.

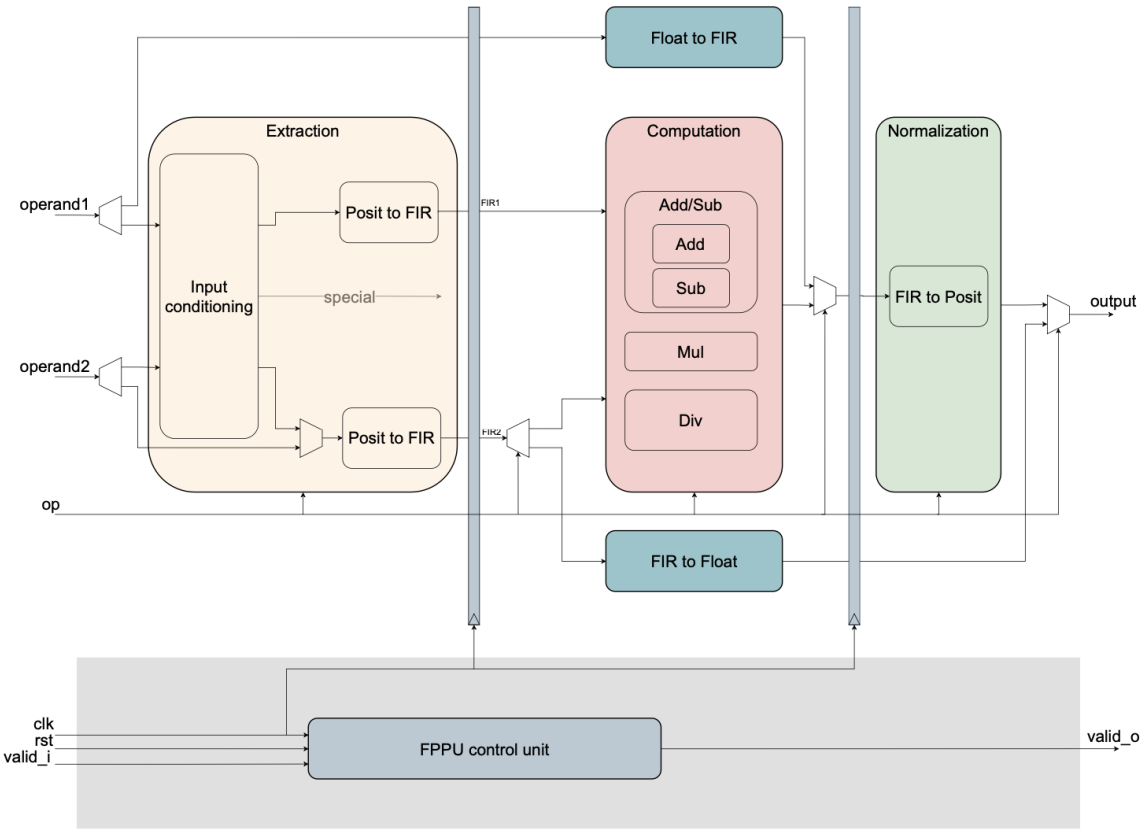

Figure 2: 3-stage Full PPU with control unit.

The Figure 2 shows a top-level schematic of the Full PPU.
The Full PPU control unit has the task of signaling whether the current output is valid (i.e. 4 stages have been traversed by an instruction) or not.
Figure 5 shows an example of interaction with the Full PPU with the submission of an operation identified by OP and the two operands P1, P2. When the operands are ready, valid_in is set to active and after 3 cycles the Full PPU produces a valid result PO signaled by valid_out.

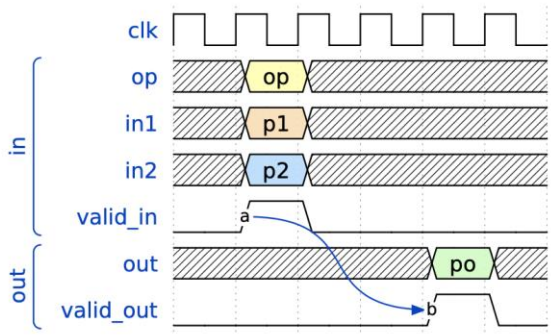

Figure 3: Example of interaction with the Full PPU.

In Table I we compare the accuracy of our solution to the solution of PACoGen [8], that employs a pre-computed look-up table for reciprocal approximation, and as input of a round of Newton-Raphson.
In the Table I, the IN column is the number of fraction bits from the posits used to index the LUT, while OUT is the number of bits of the reciprocal approximation of the fraction; NR indicates the number of Newton-Raphson rounds used and wrong [%] states the error percentage of the results when compared to a software golden model for Posit computation. In Table I the focus is on division operations which is the most complex to implement.
The results in Table I show that the proposed PPU has better accuracy than [8].

TABLE I PERCENTAGES OF POSITS P⟨N, ES⟩ INEXACT DIVISION RESULTS.
PACOGEN VERSION [10] VS PROPOSED.

		PACoGen				proposed	
N	ES	IN	OUT	NR	wrong [%]	NR	wrong [%]
8	0	8	9	0	4.8	1	**1.4**
8	1	8	9	0	5.4	1	**1.2**
8	2	8	9	0	9.3	1	**2.1**
8	3	8	9	0	13.5	1	**4.2**
8	4	8	9	0	16.4	1	**7.5**
16	0	8	9	1	10.0	1	**1.5**
16	1	8	9	1	10.0	1	**0.6**
16	2	8	9	1	8.8	1	**0.5**
16	3	8	9	1	9.0	1	**0.1**

5. Full PPU interfacing to RISC-V: ISA extension and compiler support

This section briefly presents the RISC-V ISA extension for Posits and the related compiler support.
We decided to reuse the existing RISC-V registers (i.e. x1...x31).
Since posits can be treated as signed integers inside the architecture, we followed the RV32I base integer instruction set from the RISC-V standard for arithmetic operations. ADD, SUB, MUL, DIV and FMA (Fused MUL&ADD) posit operations are encoded as R-type instructions (see [9]). In Table II we reported the listing of posit instructions used for the RISC-V ISA extension proposed in this deliverable D2.3. With respect to D2.2 in Table II we highlight the new supported operation.

TABLE II INSTRUCTION LISTING FOR POSIT ARITHMETIC ISA EXTENSION

funct7	rs2	rs1	funct3	rd	opcode	R-TYPE
1100000	rs2	rs1	000	rd	0001011	PADD
1101010	rs2	rs1	001	rd	0001011	PSUB
1100000	rs2	rs1	010	rd	0001011	PMUL
1100000	rs2	rs1	100	rd	0001011	PDIV
rs3 — 00	rs2	rs1	000	rd	0101011	PFMADD

Besides the posit arithmetic operations we also added conversion instructions between posits and binary32 numbers, so that we can enable the use of binary32 numbers as frontend while maintaining posit computation as backend in HW. To enable the use of the novel instructions in SW, we provided a set of intrinsic functions to map high-level C/C++ function calls to the underlying machine code. Doing this, we do not need to change the RISC-V compiler, but we generate the correct assembly for the posit ISA at compile time. We report an example of intrinsic and relative call from C code in Listing 1. The register keyword suggests the compiler to put the values of the input operands and the result in three registers, since the instruction type is R-type. At lines 6,7,8 we set up the opcode, funct3, funct7 parameters for the specific operation (this is the only part that varies between different operations).

```
1  posit_t padd(long a, long b) {
2    register int p1 asm("a1") = a;
3    register int p2 asm("a2") = b;
4    register int result asm("a0");
5    __asm__(
6    ".set op,0xb\n"
7    ".set opf1,0\n"
8    ".set opf2,0x6A\n"
9    ".byte  op|((r%[result]&1) <<7),"
10   "((r%[result]>>1)&0xF)|
11   (opf1<<4)|((r%1 &1) <<7),"
12   "((r%2&0xF) << 4) | ((r%1>>1)&0xF),"
13   "((r%2>>4)&0x1)|(opf2 << 1)\n"
14   : [result] "=r"(result)
15   : "r"(p1), "r"(p2), "[result]"(result));
16   return result;
17 }
```

Listing 1. Example of intrinsic for a posit addition operation.

A more complex example is shown in Listing 2 and 3 with the implementation of a simple square matrix multiplication and a 3 × 3 convolution.

```
1   void gemm(posit_t *a,
2             posit_t *b,
3             posit_t *c,
4             int n) {
5    for (int i = 0; i < n; i++) {
6     for (int j = 0; j < n; j++) {
7      posit_t sum = 0;
8      for (int k = 0; k < n; k++) {
9       sum = padd(sum, pmul(a[i*n+k], b[k*n+j]));
10     }
11     c[i * n + j] = sum;
12    }
13   }
14  }
```

Listing 2. Square matrix-matrix multiplication using posit intrinsic.

```
1   void conv3x3(posit_t *a,
2                posit_t *f,
3                posit_t *c,
4                int n) {
5    for (int i = 0; i < n; i++) {
6     for (int j = 0; j < n; j++) {
7      posit_t sum = 0;
8      for (int k = 0; k < 3; k++) {
9       for (int l = 0; l < 3; l++) {
10       sum = padd(sum, pmul(a[(i+k)*n+j+l],f[k*3+l]));
11      }
12     }
13     c[i * n + j] = sum;
14    }
15   }
16  }
```

Listing 3. 3×3 convolution using posit intrinsic.

To assess the easy integration of the proposed FPPU (Full Posit Processing Unit) with RISC-V based cores, the FPPU has been integrated with an Ibex Core, which is a 32-bit RISC-V core with a 2-stage pipeline.
Ibex Core supports the Integer (I) or Embedded (E), Integer Multiplication and Division (M), Compressed (C), and B (Bit Manipulation) extensions.
The Ibex Core has been selected since the Ibex Core does not support real number arithmetic; hence it was an ideal candidate to test the impact of adding such operations using the FPPU.
The FPPU was added alongside the ALU (Arithmetic Logic Unit) inside the execution stage of the pipeline, and the decoder module was modified by adding the instructions designed in the previous Section.

6. Full PPU (integrated with RISC-V) validation

To validate the integration, the authors used RTL simulation with binaries compiled for the RV32IM architecture. The instruction tracer inside the Ibex core was used to dump all the instructions executed by the core during the RTL simulation, including the newly added posit instructions. The output of the tracer was fed to a trace parser to evaluate the output of each posit instruction and compare it to the same software golden model used for validating the standalone FPPU. The authors also used the trace parser to assess the accuracy of each operation by comparing the result to the IEEE binary32 correspondent operation result.

To test the compliance of the FPPU HDL design vs. a posit golden model, the authors tested the overall system with different DNN (Deep Neural Network) kernels on 8 × 8 matrices in D2.2:

1. Matrix-matrix multiplication kernel involving pairs of 8 × 8 matrices.
2. Convolution kernel using a 3 × 3 image filter on 8 × 8 matrix inputs.
3. Average pooling using a 4 × 4 window on 8 × 8 matrices.

In D2.3 we tested the overall system with different DNN kernels on 32 × 32 matrices (i.e. size of images for MNIST and CIFAR10 datasets). In detail, we tested a matrix-matrix multiplication, a 3×3 convolution and a 4×4 average pooling. Each test was run on the 8-bit FPPU and on the 16-bit FPPU.

Fig. 4 shows the workflow diagram used for compiling, simulating, and validating the instruction set added to the RISC-V instruction set thanks to the HW support offered by the Full Posit Processing Unit (PPU).

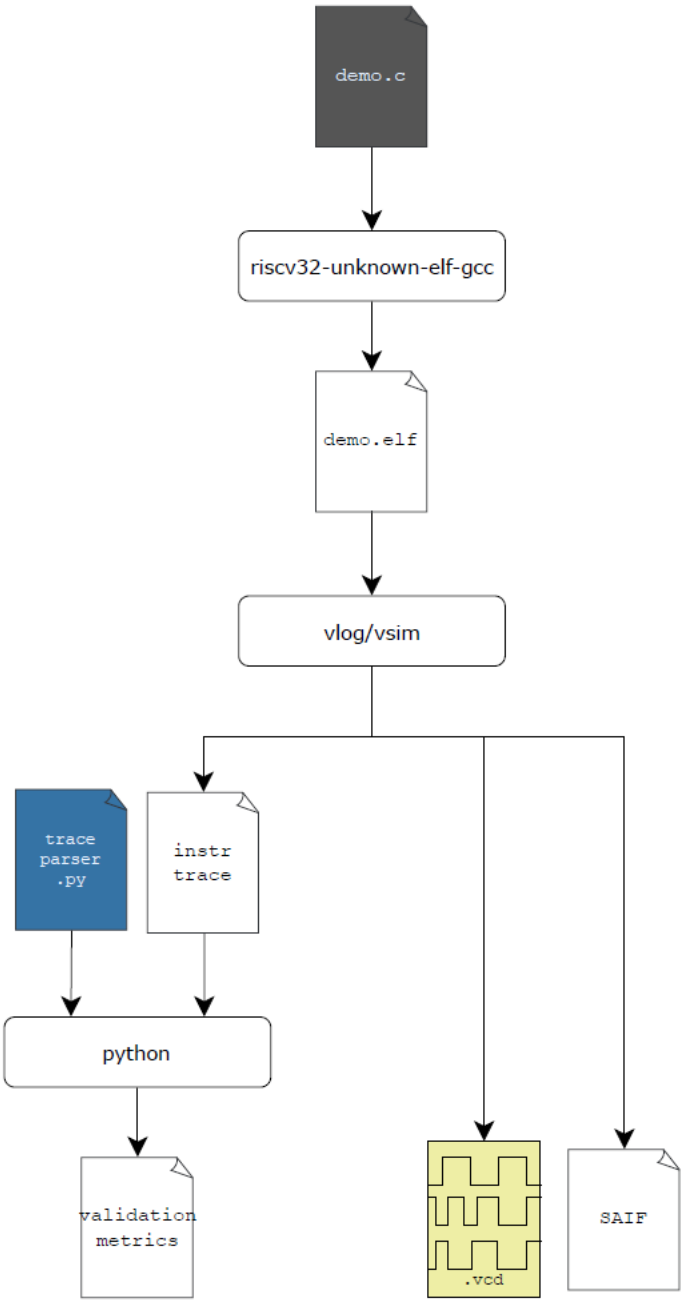

Fig. 4: Workflow diagram for compiling, simulating, and validating the instruction set added to the RISC-V ISA thanks to the HW support offered by the Full Posit Processing Unit (PPU).

The DNN kernel used are just simplified examples of the ones that can be found in bigger DNNs. We employed these kernels to understand the behavior of single posit instructions on the core operations that can be found in different DNNs.
Each test was run on both the 8-bit and 16-bit FPPUs inside the Ibex core, and the instruction traces. were collected and fed to the trace parser. Two sets of results were collected: the accuracy with respect to the posit golden model and the normalized mean error with respect to the same operation executed with binary32 format. The accuracy w.r.t the posit golden model of both 8-bit and 16-bit FPPUs inside the Ibex core on different DNN kernels is shown in Table III.
Table IV shows the normalized mean error of posit operations compared to the corresponding binary32 ones. Further accuracy comparisons regarding the use of Posits for more complex Deep Neural Networks has been done by us and published in literature, see as example [2,10].

TABLE III
FUNCTIONAL TEST OF THE FPPU UNIT WITH IBEX AGAINST THE POSIT GOLDEN MODEL IN DIFFERENT LINEAR ALGEBRA TASKS.

	Conv 3x3		GEMM		Average Pooling 3x3	
	$p\langle 8,0\rangle$	$p\langle 16,2\rangle$	$p\langle 8,0\rangle$	$p\langle 16,2\rangle$	$p\langle 8,0\rangle$	$p\langle 16,2\rangle$
p.mul	100%	100%	100%	100%	-	-
p.add	100%	100%	100%	100%	100%	100%
p.div	-	-	-	-	100%	100%

TABLE IV
NORMALIZED MEAN ERROR OF FPPU OPERATIONS VS. CORRESPONDENT BINARY32 OPERATION IN SEVERAL LINEAR ALGEBRA TASKS.

	Conv 3x3		GEMM		Average Pooling 4x4	
	$p\langle 8,0\rangle$	$p\langle 16,2\rangle$	$p\langle 8,0\rangle$	$p\langle 16,2\rangle$	$p\langle 8,0\rangle$	$p\langle 16,2\rangle$
p.mul	0.042	0.004	0.019	0.003	-	-
p.add	0.025	0.0004	0.016	0.0007	0.019	0.0002
p.div	-	-	-	-	0.002	0

To prove the effectiveness of the proposed PPU with real Neural Networks, Fig. 5 shows the accuracy comparison between 8-bit posit and 16-bit posit formats, bfloat16 and 32-bit IEEE binary32 with a LeNet-5 convolutional neural network.
The considered data base is 3 well-known data bases used in literature for NN benchmarking: CIFAR10, MNIST and GTRSB (German Traffic Road Sign Benchmark).

Fig. 6 shows an accuracy comparison between 16-bit formats (posit and bfloat16) and 32-bit IEEE binary32 on complex DNN tasks: SSD300model (based on SSD: Single Shot MultiBox Detector) with VOC2007 data base, used as benchmark in state of art, see [11], and EfficientNetB0 (EffNetB0 in Fig. 6) on version2 of the ImageNet [12] benchmark.

The achieved results show that the Posit processing supported by the proposed IP outperforms BFloat16 format in terms of processing accuracy and is a good alternative to 32 classic data formats.

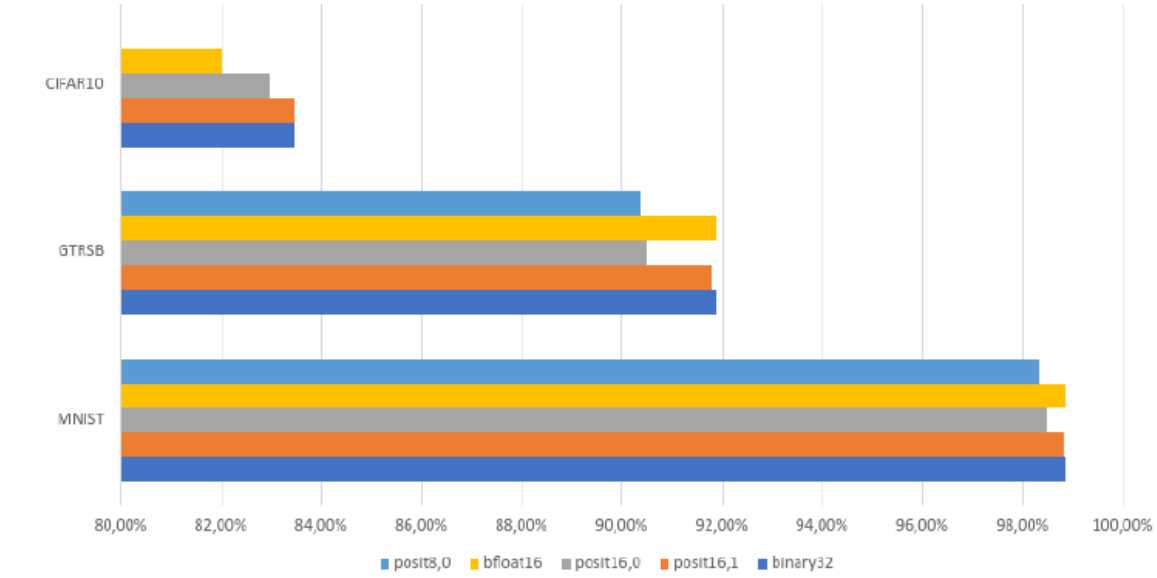

Fig. 5: Accuracy comparison between 8-bit, 16-bit posit formats and 32-bit IEEE binary32 on the LeNet-5 convolutional neural network.

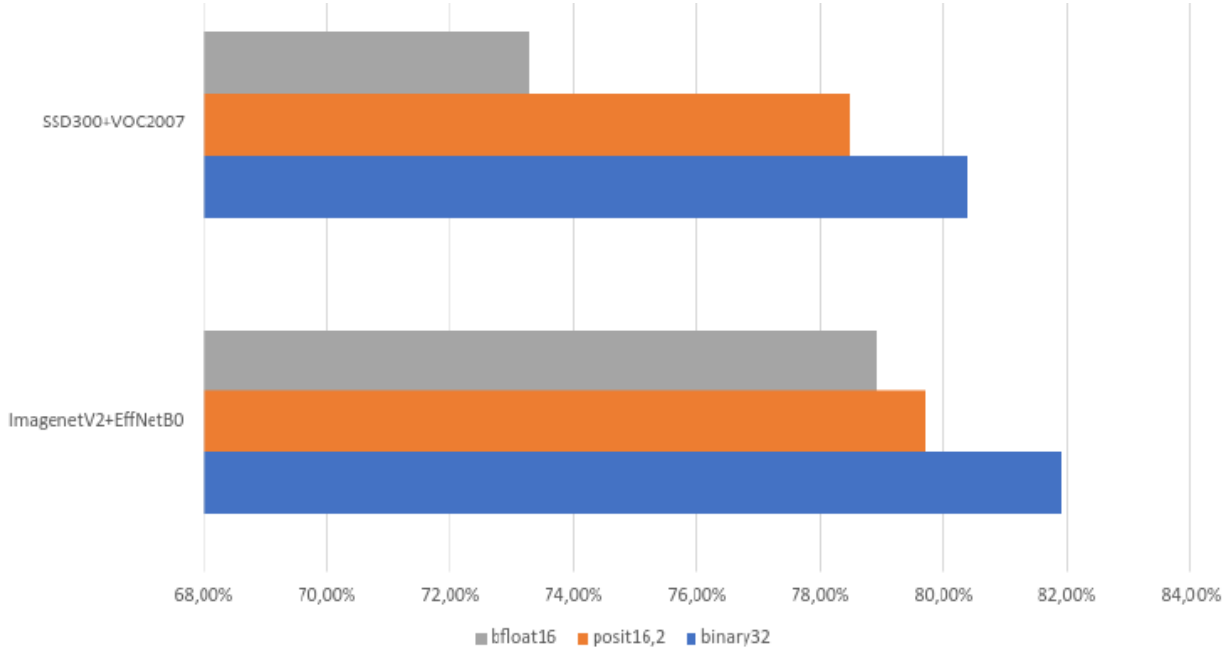

Fig. 6: Accuracy comparison between 16-bit formats (posit, bfloat16) and 32-bit IEEE binary32 on complex DNN task.

7. Full PPU characterization in FPGA technology

This section discusses the characterization of the FPPU, either alone or integrated with the Ibex RISC-V core, targeting Xilinx Alveo U280.
Since Xilinx Alveo U280 is useful for big data and for server applications, this characterization is significant for the aim of Textarossa.
Through the synthesis process, area and power metrics were extracted, and the CV32E40P RISC-V open core was instantiated to compare the area occupation of the operation implementations in both solutions.
The area occupation of the arithmetic logic unit (ALU) and the FPPU were shown in relation to the total area of the Ibex RISC-V core, see Fig. 7, and it was found that an 8-bit PPU provides real number arithmetic capabilities to the core with an area cost that is less than the cost of the original Ibex ALU.
The comparison between FPPUs and FPU in terms of area occupation of the logic related to ADD, MUL, and DIV operations is presented in Fig. 8, showing that using half of the bits for the representation results in an area cost that is less than half of the area for the 32-bit counterpart.

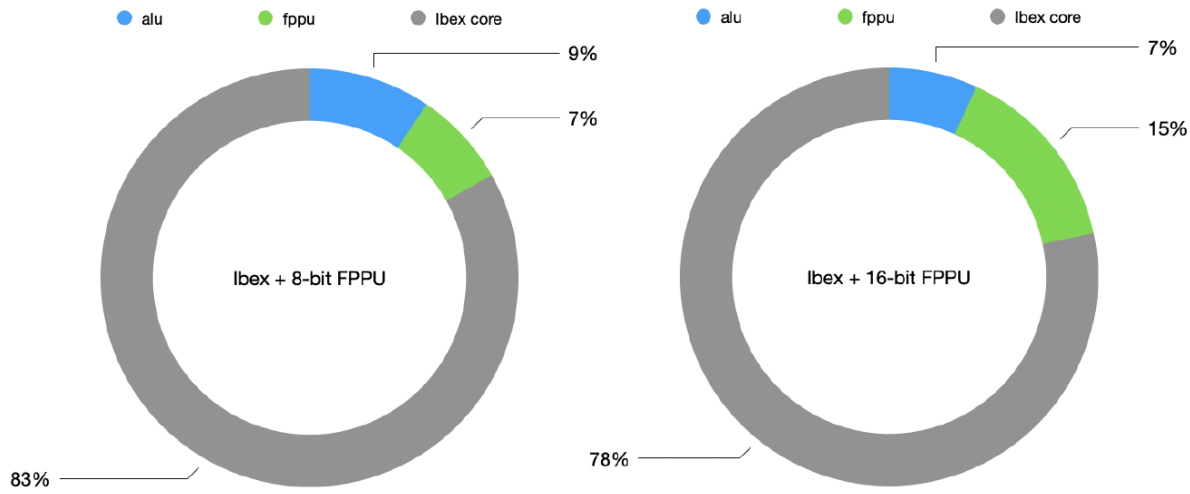

Figure 7: Percent Area utilization (LUTs) of the FPPU with Posit ⟨8, 2⟩ (left) and Posit ⟨16, 2⟩ (right) and the other components of Ibex.

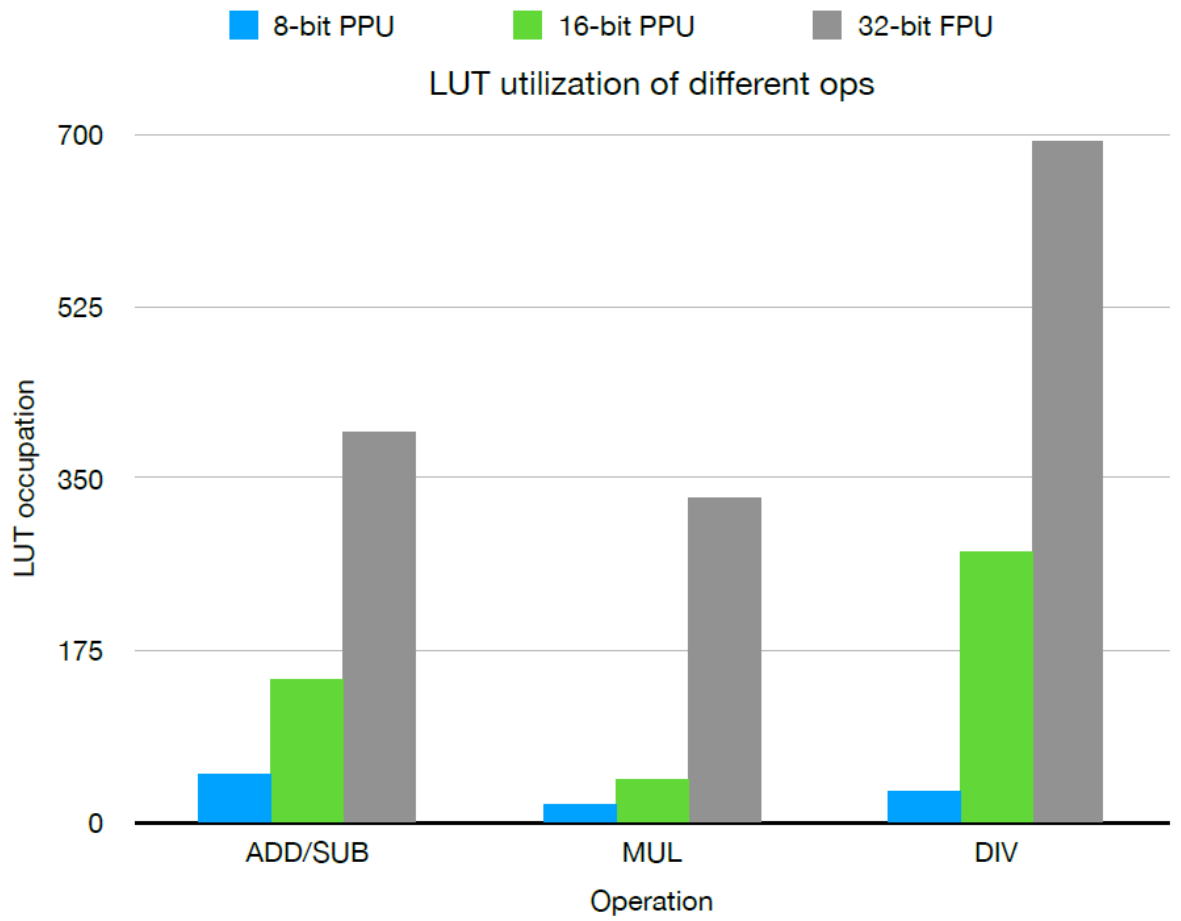

Figure 8: Comparison between absolute area utilization (LUTs) of 8-bit, 16-bit FPPU and 32-bit FPU operations

The dynamic power of the FPPU component was carefully estimated in collaboration with the Politecnico of Milano partner using the methodology published in [14] for implementation on different Xilinx FPGAs (Artix7, Spartan7 and Kintex7).
In this deliverable the results are reported with reference to dynamic power consumption after a synthesis on the Alveo U280 FPGA device.
The results achieved show that for a Full PPU configured for Posit-8 the average power consumption when implementing multiplication and addition/subtraction operations is around 0.5 mW.
Such dynamic power consumption tests were done at a frequency of 100 MHz, and we can perform up to 1 Posit operation per clock cycle, hence the average energy cost of a single Posit-8 operation is 0.5mW/100MHz= 5 pJ.
If we analyze worst case values, then the peak power consumption for the Full PPU configured for Posit-8 is obtained with testbench performing division operations, for which the peak power consumption is 1 mW.
For a Full PPU configured for Posit-16 the average power consumption is doubled vs. the power consumption of a Full PPU configured for Posit-8, in the same conditions.
These values are aligned with the specification of Alveo U280 device in [15] which claims 24.5 TOPs for a peak power consumption of 225 W (including memory), thus resulting in 225W/24.5 TOPs=9.2 pJ per operation (logic and memory).

The above power consumption and energy cost results can be lowered moving from the FPGA implementation, foreseen in Textarossa, to an ASIC implementation in scaled technologies like

GF12nm and/or TSMC 6 nm as foreseen for the RISC-V based accelerator and for the general purpose processor in the EPI SGA2 project.

To be noted that when synthesizing the Full PPU on Xilinx FPGA technology the maximum frequency, also reported in [14], is 121 MHz. For the FPGA application this value is acceptable since the RISC-V Ibex core with which the IP is used has, on the same device, a frequency of 20 MHz.

The maximum peak throughput at this frequency for the Full PPU, if the input is feed at enough rate, is 120 MOps/s, i.e. 1 MOps/MHz. To be noted that for the Full PPU integrated with the Ibex RISC-V core, since the Ibex cannot enable the Full PPU each clock cycle to start a new computation, but a new computation is started only when the previous one has been completed, the sustained throughput is 0.25 MOps/MHz for the FPPU component with four pipeline stages.

The above frequency values can be increased by one order of magnitude moving from the FPGA implementation, foreseen in Textarossa, to an ASIC implementation in scaled technologies like GF12nm and/or TSMC 6 nm as foreseen in the EPI SGA2 project where indeed target frequencies are in the order of some GHz.

Considering that we are under-using the 32-bit registers when employing 8-bit or 16-bit posits, we can think of increasing the number of FPPUs that elaborate in parallel, similarly to a Single Instruction Multiple Data (SIMD) paradigm. This can be done transparently to the instruction caller and with the same opcode.
When using only one FPPU we just need to put the posit arguments in 8 (or 16) least significant bits.
If we want to compute more posit operations in parallel, we will just need to put another three posits in the remaining 24 bits of the register (or another one posit in the remaining 16 bits). This can be easily implemented by reproducing the same FPPU 2 or 4 times (respectively, for 16-bit posits or for 8-bit posits). Then we can feed the same operand, valid, reset and clock inputs to all the FPPUs, while the two operands are constituted by portions of the source registers. The output is then concatenated from all the FPPU outputs into the destination register.
We have verified that this approach would increase the throughput of the FPPU to:
- x4 MOps/s for the same clock frequency for 8-bit posits.
- x2 MOps/s for the same clock frequency for 16-bit posits.

This approach would increase the throughput of the FPPU for both 8-bit posits and for 16-bit posits. However, the main difficulties arise from the software side, where we need to change the C++ source code (or any other programming language source code) to exploit this extended feature, and this can be challenging for complex applications. We need either a compiler supporting automatic vectorization of the code or a partial rewrite of mathematical kernels to explicitly support the SIMD paradigm. Overall, the results suggest that an 8-bit PPU provides a cost-effective solution for implementing real number arithmetic capabilities in the Ibex core and employing a SIMD paradigm can further increase the throughput of the FPPU.

8. Conclusions and IP repository

This deliverable D2.3 presented the final design, verification, and integration of a Full PPU (posit processing unit) inside an Ibex open-source RISC-V core, extending the RISC-V ISA to support posit arithmetic operations.

The Full PPU inside the RISC-V extends the RISC-V ISA to support posit arithmetic operations (add/sub, mul, div, posit-to-float/float-to-posit and FMA) as well as vectorized posit operations in a SIMD configuration of 4 units for 8-bit posits and 2 units for 16-bit posits.

Several integration and validation test against both a posit golden model and a series of baseline binary32 neural network models (LeNet-5, EfficientNet, SSD300) has been done showing at most a degradation of 4% in terms of inference accuracy on the object detection tasks (VOC), while obtaining negligible drops in accuracy for 16-bit posits in the other simpler tasks (MNIST, CIFAR10, GTSRB).
With respect to other works in literature like PaCoGen, this work allows for an improved accuracy.

With reference to an FPGA implementation on Alveo U280 device, the same FPGA that is used in the IDV-E platform of WP5 of TEXTAROSSA, CINI-UNIPI has characterized the whole unit on power and area. Particularly, it has been compared the occupation of the unit in relation to the full-PPU core, showing that the 8-bit posit Full PPU complexity is even lower than the Ibex ALU. This outcome matters in terms of computation potential, in that real numbers' arithmetic can express a much larger range and precision than integer numbers arithmetic.

The increase in area occupation of the Full PPU is 7% for 8-bit posits and 15% for 16-bit posits. When compared to the FPU used by the CV32E40P RISC-V core, the FPPU area occupation is less than half for 16-bit posits (for the same accuracy of binary-32) and 1 order of magnitude lower for 8-bit posits (for an accuracy loss within 4%).

When compared to the Light PPU in D2.6, that gave rise to the scientific work in [4], it must be noted that the Light PPU gives to a RISC-V core the support to Posit with a limited complexity overhead and no speed overhead, but the support is limited to float-to/from-posit conversion.
Hence, [4] is not effective for Posit computation since it requires that posit are converted in float, then processed by the FPU and then the results converted back in posit. Instead, the FPPU gives direct HW support to posit operations (sub, add, mul, div, inversion, conversion) so that computation in posit domain becomes efficient vs floating-point representation.

Since the FPPU replaces in RISC-V the FPU the complexity overhead of the FPPU is lower than the overhead of Light PPU plus the FPU.

The IP repository from which can be accessed the Full PPU is:
https://github.com/federicorossifr/ppu_public

The usability of the IP has been already proved by the fact that the Full PPU IP has been delivered and used by TEXTAROSSA Partners like:

- POLIMI, to be used with their tool for power consumption estimation,
- POLIMI, to be used with their tool TAFFO for mixed-precision and multi-arithmetic data type supported.
- INFN, to be integrated with their Aperion environment for low-latency communication.

Moreover, the results of D.23 have been submitted to the scientific community through the paper, coauthored by Federico Rossi, Francesco Urbani, Marco Cococcioni, Emanuele Ruffaldi and Sergio Saponara entitled “FPPU: Design and Implementation of a Pipelined Full Posit Processing Unit” which in revised version is under review in IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING.
A collaboration with Politecnico di Milano has been done for power consumption analysis and the results published in M. Piccoli, D. Zoni, W. Fornaciari, G. Massari. M. Cococcioni, F. Rossi, S. Saponara, E. Ruffaldi, “Dynamic Power consumption of the Full Posit Processing Unit: Analysis and Experiments”, Proceedings of PARMA-DITAM 2023.

9. References

[1] D2.1, “Consolidated specs of accelerators IPs”, TEXTAROSSA project, revised, May 2023.
[2] 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, pp. 97-110.
[3] 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).
[4] 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
[5] 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.
[6] https://github.com/openhwgroup/cva6.
[7] D2.2, “AI Accelerator with mixed-precision including Posit, part 1”, TEXTAROSSA project, revised version, May 2023.
[8] M. K. Jaiswal and H. K.-H. So, “PACoGen: A hardware posit arithmetic core generator,” IEEE Access, vol. 7, pp. 74 586–74 601, 2019.
[9] “RISC-V ISA Specification,” https://riscv.org/specifications/isa-spec-pdf/,
[10] Cococcioni, M., Rossi, F., Ruffaldi, E., Saponara, S. (2022). Small Reals Representations for Deep Learning at the Edge: A Comparison. In: Gustafson, J., Dimitrov, V. (eds) Next Generation Arithmetic. CoNGA 2022. Lecture Notes in Computer Science, vol 13253. Springer, Cham. https://doi.org/10.1007/978-3-031-09779-9_8
[11] Wei Liu et al., SSD: Single Shot MultiBox Detector, European Conference on Computer Vision 2016, https://link.springer.com/chapter/10.1007/978-3-319-46448-0_2
[12] https://imagenetv2.org/
[13] https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet/EfficientNetB0
[14] M. Piccoli, D. Zoni, W. Fornaciari, G. Massari. M. Cococcioni, F. Rossi, S. Saponara, E. Ruffaldi, “Dynamic Power consumption of the Full Posit Processing Unit: Analysis and Experiments”, Proceedings of PARMA-DITAM 2023
[15] https://www.xilinx.com/products/boards-and-kits/alveo/u280.html#specifications