RISC-V Posit arithmetic and the Pisa PPU

Federico Rossi

PhD Workshop on Hardware accelerators for AI and HPC applications
Pisa, Italy

30th November, 2020

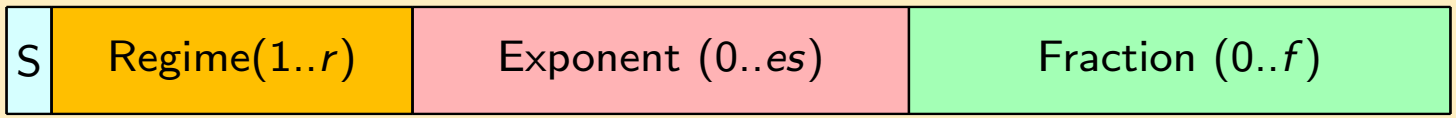
1. Posits: quick recap
2. RISC-V and the Xposit extension
3. The Posit Processing Unit (PPU)
4. Conclusions

Section 1

Posits: quick recap

Format overview

Structure



Format parameters

1. Overall number of bits (*nbits*)
2. Number of exponent bits (*es*)

Represented value

$$x = (-1)^{\text{sign}} \cdot \text{useed}^{k} \cdot 2^{e} \cdot (1 + F)$$

$$\text{useed} = 2^{2^{es}}$$

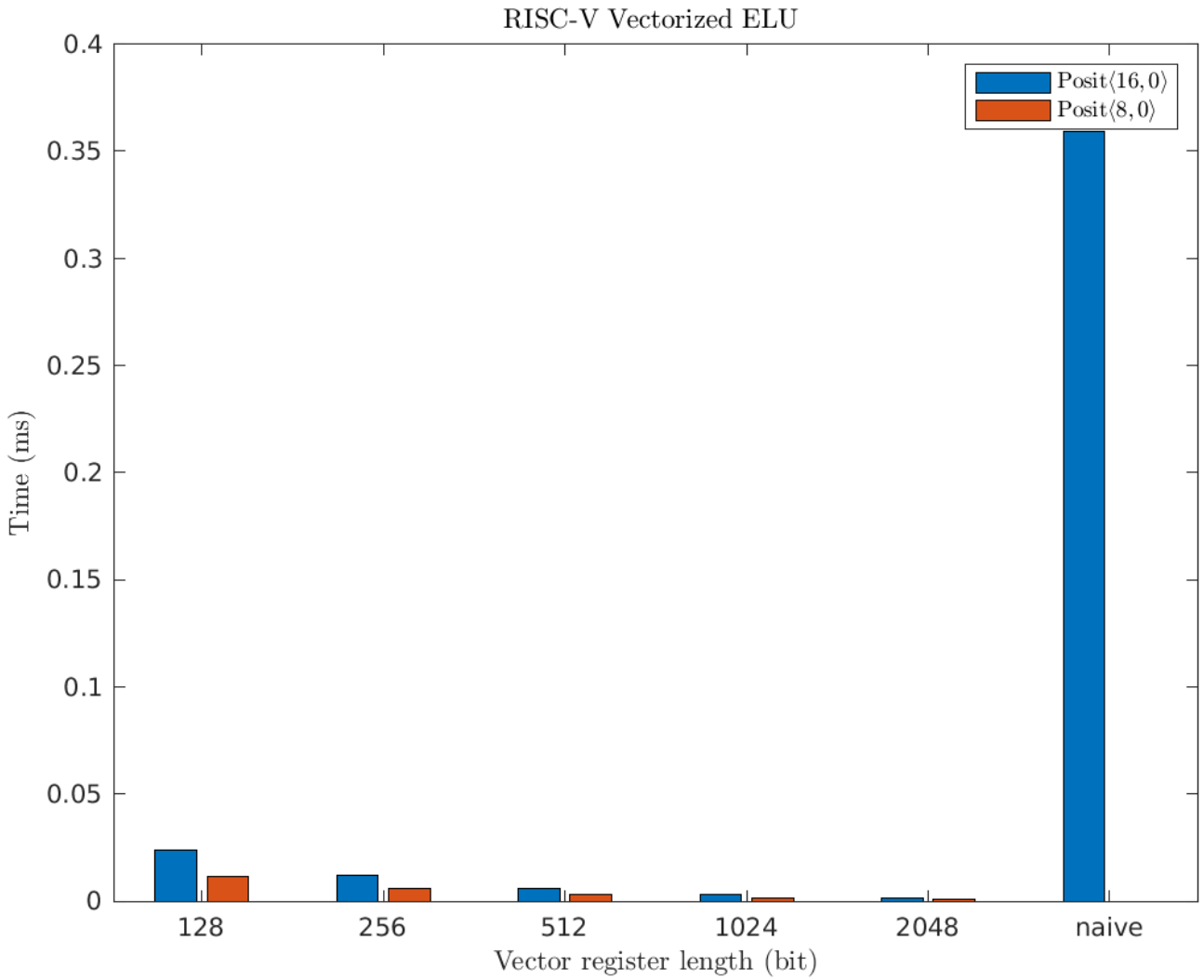
Posits and IEEE Floats

	IEEE Floats	**Posits**
Precision	Flat	Tapered
Format	Fixed	Configurable
Wasted bit patterns	More than 8 millions	No

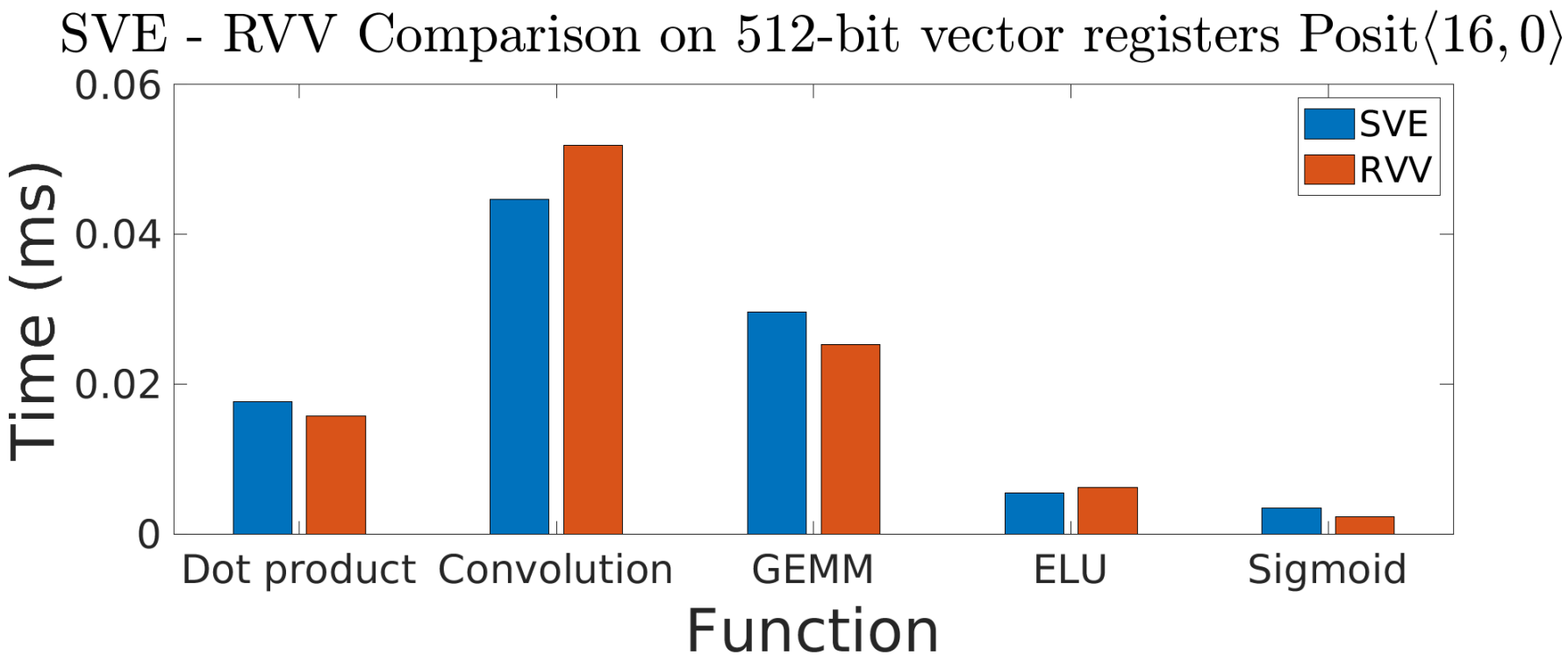
Recent work on posit SW support

- Vectorization support for core machine learning operations (convolution, dot product and matrix-matrix multiplication) and functions (Sigmoid, Tanh and ELU)
  - ARM Scalable Vector Extension
  - RISCV "V" Vector Extension
- Focus on vectorized posit encoding/decoding performance to accelerate compression/decompression of information.
- Benchmarks on simulated platforms using vector instruction emulators.
  - ARM Instruction Emulator (armie) $\rightarrow$ Cray/HPE Apollo 80 (Fujitsu A64FX) in the upcoming months @ UNIPI
  - RISC-V Spike simulator with V 0.8 support

Recent work on posit SW support



Recent work on posit SW support



[1] *M. Cococcioni, F. Rossi, E. Ruffaldi, and S. Saponara.* **Fast deep neural networks for image processing using posits and ARM scalable vector extension.** Journal of Real-Time Image Processing 17

[2] *M. Cococcioni, F. Rossi, E. Ruffaldi, and S. Saponara.* **Vectorizing Posit Operations on RISC-V for faster Deep Neural Networks: Experiments and Comparison with ARM SVE**, [under review]

Section 2

RISC-V and the Xposit extension

Preliminary concepts

- Possibility to extend the original RISC-V instruction set architecture (ISA)
- Instruction encoding suffix `'b0001011` reserved for custom RISC-V ISA extension
- Goal: provide a set of posit instructions for compression and decompression of IEEE Floats and fixed point format
- Requirements:
  - compiler independent
  - compliant with RISC-V minimalism

The Xposit extension

- ALU registers as posit holders $\rightarrow$ no dedicated posit registers
- Support for *posit* $\langle 16, 1 \rangle$, *posit* $\langle 16, 0 \rangle$, *posit* $\langle 8, 0 \rangle$ configurations
- Posit-to-posit conversions
- Posit-to-float and posit-to-fixed conversions
- Standard instruction naming (e.g. for float-posit conv.):
  - `FCVT.S.P8/FCVT.P8.S`:
    Float to/from posit$\langle 8, 0 \rangle$ conversion
  - `FCVT.S.P16.0/FCVT.P16.0.S`:
    Float to/from posit$\langle 16, 0 \rangle$ conversion
  - `FCVT.S.P16.1/FCVT.P16.1.S`:
    Float to/from posit$\langle 16, 1 \rangle$

ISA list

RV64Xposit Posit Ext. Instruction Set

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

Table: Instruction listing for RISC-V RVXposit extension

Compiler support and cppPosit integration

Intrinsics header file

- From instruction listings → instrinsics list using a script
  - Each intrinsic uses inline assembly code to emit instruction bytecode
  - Use of `register` directive → compiler autonomously chooses right registers
- All intrinsics in a single header-only file → just include it and compile

cppPosit integration

- cppPosit is our made-in-Pisa C++ posit library supporting full emulation of posits via ALU,FPU or tabulation
- Added "hardware" support for the posit processing unit in type construction and conversions
  - Just pass the `RISCV_PPU` flag to the compiler to use the hardware-accelerated conversion functions

Spike ISA simulator

Spike in a nutshell

- "Official" RISC-V simulator from the RISC-V organization
- High-level C++ implementation of RISC-V instruction set (RV64-GCV)

Xposit extension in Spike

- Posit instructions implemented using our cppPosit library fully emulating a posit processing unit
- Validated comparing results from cppPosit when using the `RISCV_PPU` flag for compilation or not.

Simulated comparison on neural network inference

	w/ Emulated PPU (ms)	w/o Emulated PPU (ms)
posit$\langle 8, 0 \rangle$	92	533
posit$\langle 16, 0 \rangle$	105	312
posit$\langle 16, 1 \rangle$	132	492

Table: Timing performance on a 10-layer convolutional neural network with and without the emulated PPU support in Spike for the cppPosit library.

Section 3

The Posit Processing Unit (PPU)

Summary

- Independent Verilog module designed from scratch (no high-level synthesis from cppPosit)
- Support for *posit* $\langle 16, 1 \rangle$, *posit* $\langle 16, 0 \rangle$, *posit* $\langle 8, 0 \rangle$ configurations
- Posit-to-posit conversions
- Posit-to-float and posit-to-fixed conversions
- Less generalizations $\rightarrow$ more minimisation

Regime encoding logic

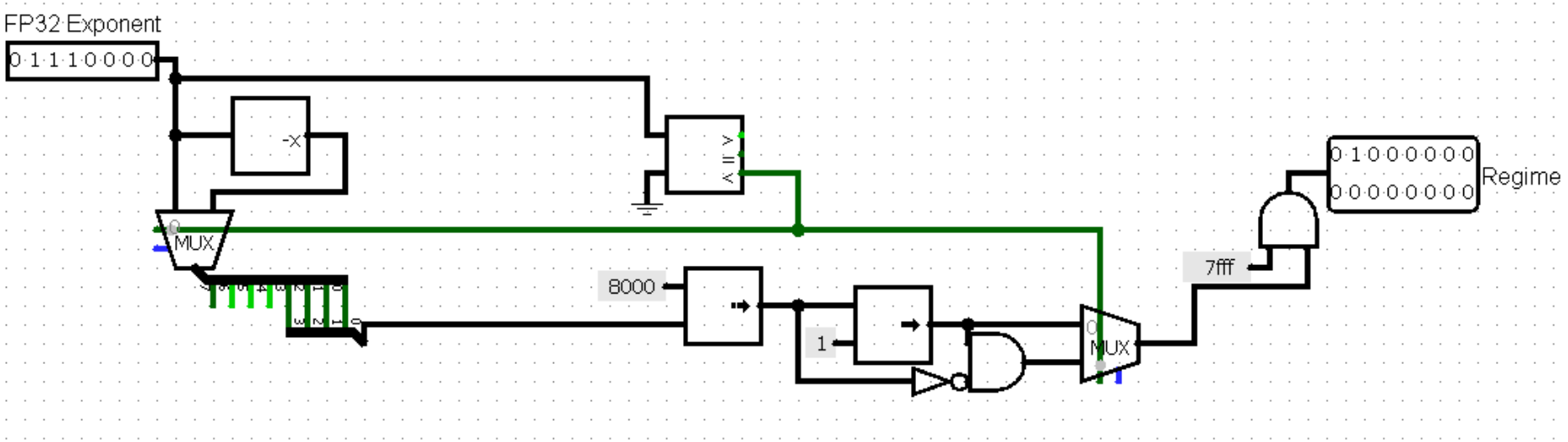


Figure: Logic circuit for the 16-bit posit regime encoder

Regime decoding logic

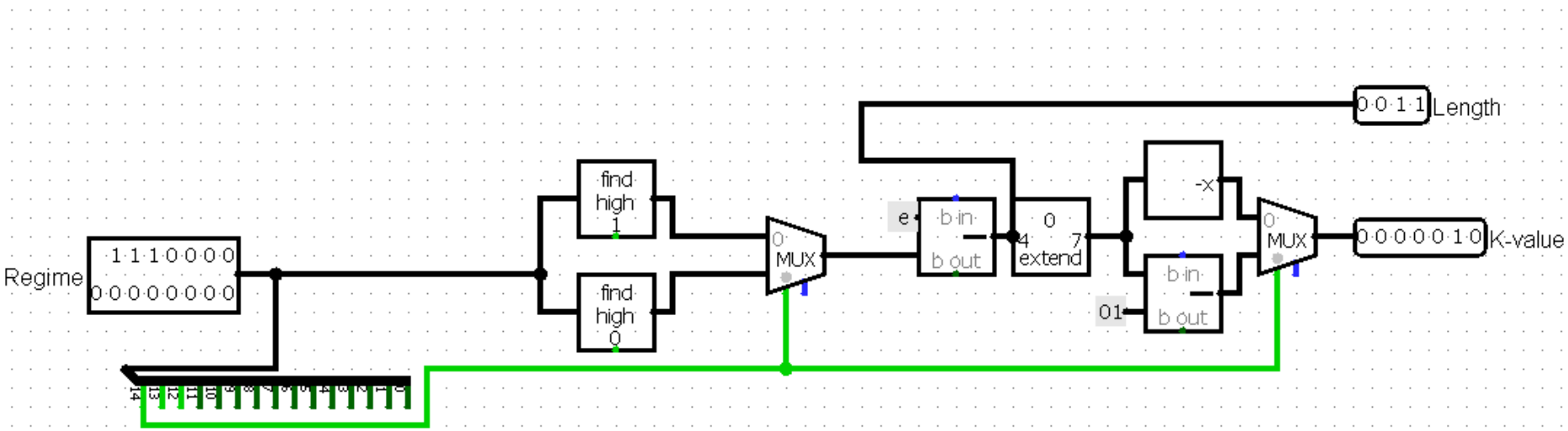


Figure: Logic circuit for the 16-bit posit regime decoder

Integration in a real RISC-V core

- PPU integration in the Ariane 6-stage RISC-V core
  - PPU as a functional unit alongside the ALU in the execution stage
  - RISC-V posit instructions added to the original Ariane ISA
  - Simulated using Verilator
- Synthesis and implementation on the Xilinx Genesys 2 Fpga
  - Power: $2.056W$
  - Space: 63805/203800 (31.31%) LUTs used.

Section 4

Conclusions

Conclusions

- We designed an instruction set extension for RISC-V supporting posit numbers
- We integrated the new ISA inside the Spike simulator, obtaining a substantial speed-up in neural network inferencing
- We designed the PPU logic circuits and synthesized them on an FPGA
- Future works:
  - Test thoroughly the FPGA performance in neural networks
  - Add support for more posit operations in hardware

federico.rossi@ing.unipi.it