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

<table>
<thead>
<tr>
<th>S</th>
<th>Regime(1..r)</th>
<th>Exponent (0..es)</th>
<th>Fraction (0..f)</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>IEEE Floats</th>
<th>Posits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precision</strong></td>
<td>Flat</td>
<td>Tapered</td>
</tr>
<tr>
<td><strong>Format</strong></td>
<td>Fixed</td>
<td>Configurable</td>
</tr>
<tr>
<td><strong>Wasted bit patterns</strong></td>
<td>More than 8 millions</td>
<td>No</td>
</tr>
</tbody>
</table>
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) → 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

![RISC-V Vectorized ELU](image)

- **Posit(16, 0)**
- **Posit(8, 0)**
Recent work on posit SW support

![SVE - RVV Comparison on 512-bit vector registers Posit(16, 0)](image)


Section 2

RISC-V and the Xposit extension
Preliminary concepts

- Possibility to extend the original RISC-V instruction set architecture (ISA)
- Instruction encoding suffix \(\text{b}0001011\) 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 → no dedicated posit registers
- Support for \( \text{posit} \langle 16, 1 \rangle \), \( \text{posit} \langle 16, 0 \rangle \), \( \text{posit} \langle 8, 0 \rangle \) configurations
- Posit-to-posit conversions
- Posit-to-float and posit-to-fixed conversions
- Standard instruction naming (e.g. for float-posit conv.):
  - \text{FCVT.S.P8/FCVT.P8.S}:
    Float to/from posit\( \langle 8, 0 \rangle \) conversion
  - \text{FCVT.S.P16.0/FCVT.P16.0.S}:
    Float to/from posit\( \langle 16, 0 \rangle \) conversion
  - \text{FCVT.S.P16.1/FCVT.P16.1.S}:
    Float to/from posit\( \langle 16, 1 \rangle \)
### RV64Xposit Posit Ext. Instruction Set

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1100000</td>
<td>00010</td>
<td>rs1</td>
<td>000</td>
<td>rd</td>
<td>0001011</td>
</tr>
<tr>
<td>1100000</td>
<td>00011</td>
<td>rs1</td>
<td>000</td>
<td>rd</td>
<td>0001011</td>
</tr>
<tr>
<td>1100000</td>
<td>00011</td>
<td>rs1</td>
<td>010</td>
<td>rd</td>
<td>0001011</td>
</tr>
<tr>
<td>1101000</td>
<td>00010</td>
<td>rs1</td>
<td>000</td>
<td>rd</td>
<td>0001011</td>
</tr>
<tr>
<td>1101000</td>
<td>00011</td>
<td>rs1</td>
<td>000</td>
<td>rd</td>
<td>0001011</td>
</tr>
<tr>
<td>1101000</td>
<td>00011</td>
<td>rs1</td>
<td>010</td>
<td>rd</td>
<td>0001011</td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FCVT.S.P8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FCVT.S.P16.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FCVT.S.P16.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FCVT.P8.S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FCVT.P16.0.S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FCVT.P16.1.S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Instruction listing for RISC-V RVXposit extension
Compiler support and cppPosit integration

Intrinsics header file
- From instruction listings → intrinsics 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

<table>
<thead>
<tr>
<th></th>
<th>w/ Emulated PPU (ms)</th>
<th>w/o Emulated PPU (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>posit\langle8, 0\rangle</td>
<td>92</td>
<td>533</td>
</tr>
<tr>
<td>posit\langle16, 0\rangle</td>
<td>105</td>
<td>312</td>
</tr>
<tr>
<td>posit\langle16, 1\rangle</td>
<td>132</td>
<td>492</td>
</tr>
</tbody>
</table>

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 $\text{posit} \langle 16, 1 \rangle$, $\text{posit} \langle 16, 0 \rangle$, $\text{posit} \langle 8, 0 \rangle$ 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