A 0.80 pJ/flop, 1.24 Tflop/sW 8-to-64 bit Transprecision Floating-Point Unit for a 64 bit RISC-V Processor in 22 nm FD-SOI

Stefan Mach*, Fabian Schuiki*, Florian Zaruba*, and Luca Benini*†
Email: {smach,fschuiki,zarubaf,benini}@iis.ee.ethz.ch
*Integrated Systems Laboratory (IIS), Swiss Federal Institute of Technology, Zurich, Switzerland
†Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna, Bologna, Italy

***Abstract*—The crisis of Moore's law and new dominant Machine Learning workloads require a paradigm shift towards finely tunable-precision (a.k.a. transprecision) computing. More specifically, we need floating-point circuits that are capable to operate on many formats with high flexibility. We present the first silicon implementation of a 64-bit transprecision floating-point unit. It fully supports the standard double, single, and half precision, alongside custom bfloat and 8 bit formats. Operations occur on scalars or 2, 4, or 8-way SIMD vectors. We have integrated the 247 kGE unit into a 64 bit application-class RISC-V processor core, where the added transprecision support accounts for an energy and area overhead of merely 11% and 9%, respectively; yet achieving speedups and per-datum energy gains of 7.3x and 7.94x. We implemented the design in a 22 nm FD-SOI technology. The unit achieves energy efficiencies between 75 Gflop/sW and 1.24 Tflop/sW, and a performance between 1.85 Gflop/s and 14.83 Gflop/s, across formats.**

I. Introduction

The demand for new computing architectures and systems has grown exponentially in the last decade. The crisis of Moore's law marks the end of an era of CMOS technology driven constant exponential improvement of computing efficiency. The power wall requires a shift towards computing paradigms where energy efficiency is the de-facto metric by which hardware designs are measured. Meanwhile new workloads such as Machine Learning dominate the industry's focus and ask for ever increasing compute capabilities both at the Internet of Things (IoT) and High Performance Computing (HPC) scale. Designers are pushed to develop new architectures and circuits for numerical computation where precision can be fine-tuned with great agility to match application requirements and minimize the energy cost per operation. This new trend has been called "transprecision computing" [1]. Such precision modulation in modern CPUs and GPUs has been limited to the "double" and "float" formats specified in IEEE-754 thus far. However, the recent "Cambrian explosion" of numerical formats, e.g. Intel Nervana's Flexpoint [2], Microsoft Brainwave's 9 bit floats [3], the Google TPU's 16 bit "bfloats" [4], or NVIDIA's Tensor Cores [5], shows that new architectures with extreme transprecision flexibility are needed for floating-point (FP) computation.

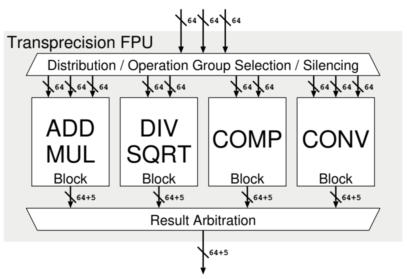

Fig. 1. Top level datapath of the transprecision FPU.

In this paper we present the first silicon implementation of a transprecision floating-point unit (FPU). Our architecture supports a wide range of data formats including IEEE-754 double (FP64), single (FP32), and half precision floats (FP16), as well as 16 bit bfloats (FP16alt) and a custom 8 bit format (FP8), introduced in [6]. The unit fully supports single instruction multiple data (SIMD) vectorization and agile data conversion and vector packing. Fully layout in a 22 nm FD-SOI technology, our unit achieves a compute performance between 1.85 Gflop/s to 14.83 Gflop/s and energy efficiencies between 75 Gflop/s W to 1245 Gflop/s W (at nominal voltage) across its range of formats.

II. Architecture

A. Floating-Point Unit for Transprecision

To achieve high energy proportionality, the architecture is based on strong clock- and data-gating of mutually exclusive sub-blocks, as opposed to fine-grained sharing of arithmetic building blocks. The top level block diagram of the transprecision FPU is shown in Fig. 1. Up to three 64 bit input operands are directed towards one of four operation groups which are each dedicated to a class of instructions. These include fused multiply-add (FMA), division/square root, comparisons and bit manipulations, and conversions between formats and integers. Each group is either divided into format-specific slices (parallel) or acts as a multi-format block (merged), as shown in Fig. 2. A merged slice can lower area cost by sharing datapath hardware among formats, which may come at potential energy and latency overheads due to all formats

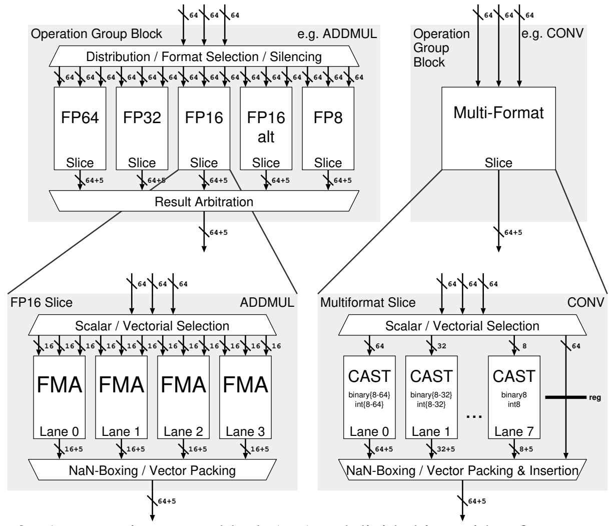

Fig. 2. An operation group block (top), subdivided into either format-specific slices (left, in the ADDMUL block for FP16) or merged multi-format slices (right, in the CONV block for all formats)

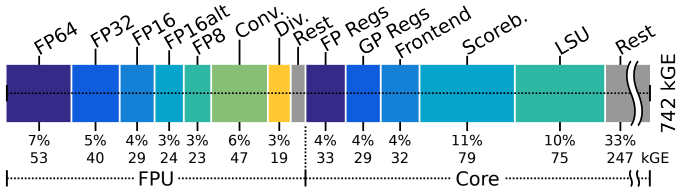

Fig. 3. Area distribution of the entire RISC-V core, excluding cache memories (in kGE, 1 GE $\approx$ 0.199 μm^2). We show the contributions of each format to the FPU together with significant core contributors. The total FPU area is approximately 33% of the overall core area.

sharing one pipeline. We opted for a parallel implementation of the ADDMUL block to allow for different per-format latencies, avoiding unnecessary overheads for narrow formats. As only one slice can receive new data in any given clock cycle, we use clock and datapath gating to silence unused branches of the FPU. Slices are further partitioned into vector lanes to facilitate SIMD execution. Depending on whether the current operation is scalar or vectored, either one or all lanes are used to compute the result while unused lanes are silenced. The operational units within a slice can either be fully pipelined or use a blocking (i.e. iterative) implementation.

In the ADDMUL block, a set of pipelined FMA units implemented using a single-path architecture [7], [8] are used for each format. They are pipelined using 4, 3, 3, 3, and 2 register stages for FP64, FP32, FP16, FP16alt, and FP8, respectively. Divisions and square roots are computed using an iterative non-restoring divider unit shared by all formats (DIVSQRT), computing 3 mantissa bits of the result per cycle. Latency is format-dependent and both division and square root take 21, 11, 7, 6, and 4 clock cycles, which is acceptable due to the relative rarity of the operation in performance-optimized code. SIMD is not supported in this unit to conserve area. The COMP block performs comparisons, sign manipulation and FP classification with 1 cycle of latency for all formats.

Flexible conversions amongst FP types are crucial for efficient on-the-fly precision adjustment [9]. To this end the merged conversion block converts between any two FP formats, as well as from and to integer formats.

While only one datum per cycle may *enter* the FPU, multiple results may be produced simultaneously due to different sub-block latencies. Thus, the order of FPU outputs is determined using fair round-robin arbitration. Fine-grained clock gating based on handshakes can occur within individual pipeline stages, silencing unused parts and allowing pipeline bubbles to "pop" by allowing data to catch up to the stalled head of a pipeline. Coarse-grained clock gating is used to disable operation groups or the entire FPU if no valid data is present in the pipeline.

B. Ariane with Transprecision FPU

Ariane is an open-source 64 bit, six stage, partially in-order RISC-V processor. It has full hardware support for running an operating system as well as parameterizable private instruction and data caches. To speed up sequential code it features a return address stack, a branch history table and a branch target buffer [10]. Besides the official RISC-V "F" and "D" ISA extensions for FP32 and FP64 we also implement the additional "Xf16", "Xf16alt", and "Xf8" extensions proposed in [11]. We furthermore support the "Xfvec" extension that adds SIMD operations for all formats narrower than 64 bit. We extend the processor's instruction decoder accordingly and to handle these extensions. These modifications have negligible timing and area cost.

The total area of the core is 742 kGE (1.4 MGE with caches), as shown in Fig. 3. The FPU accounts for 247 kGE, of which 160 kGE are the FMA, 19 kGE the iterative divider and square root (DIVSQRT), and 47 kGE the conversion unit (CONV). Compared to an Ariane core with support for only scalar FP32 and FP64 (F and D extensions), area and static energy is increased by 9.3% and 11.1%, respectively. The higher increase in energy w.r.t. the added area stems from the FPU utilizing relatively more short-gate cells than the rest of the processor due to areas of high timing pressure.

III. Implementation in 22FDX

We have implemented the architecture outlined in Section II in GlobalFoundries 22FDX technology, a 22 nm FD-SOI node. It contains two Ariane cores enhanced with our FPU, paired with additional memory and debugging infrastructure for testing. Synthesis was performed in Synopsys Design Compiler against a 1 GHz worst-case constraint (SSG, 0.72 V, 125 °C). Place and route was done in Cadence Innovus with a 1 GHz constraint in a Multi-Mode Multi-Corner flow that included all temperature and mask misalignment corners, eight in total. The design was closed in the backend at 0.96 GHz worst (SSG, 0.72 V, $\pm$0.8 V bias, $-40/125$ °C), 1.29 GHz typical (TT, 0.8 V, $\pm$0.45 V bias, 25 °C), and 1.76 GHz best (FFG, 0.88 V, 0 V bias, $-40/125$ °C) case.

We make extensive use of automated clock gate insertion during synthesis ($> 96\%$ of FPU registers gated) and support

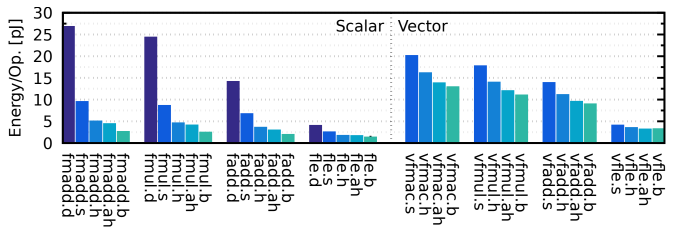

Fig. 4. FPU energy cost per operation for the fully pipelined scalar operations (left), and vectorial operations (right), grouped by FMA, multiply, add, and comparison.

architectural clock gating, as highlighted in the previous section. Ungated registers comprise only handshaking tokens and the finite-state machine controlling division and square root.

IV. Results

A. Evaluation Methodology

A detailed breakdown of energy usage within the FPU while executing individual instructions has been carried out in post-layout simulations. To this end, synthetic applications which specifically stress parts of the FPU are run on Ariane. All operations are executed using randomly distributed legal input values with sufficient constraints to prevent numerical instabilities. These input patterns provide a worst-case scenario for power dissipation as they have high switching activity. To provide fair comparisons, the tested operations are injected back-to-back so pipelines are full. Post-layout power simulations are carried out using typical libraries at nominal conditions (TT, VDD = 0.8 V, 25 °C).

B. FPU Instruction Energy Efficiency and Performance

Fig. 4 shows the average per-operation energy cost within the FPU for arithmetic operations. For the scalar operations, energy proportionality is especially pronounced in the ADDMUL block (first three groups), due to the dominance of the multiplier therein. Executing the FP64 FMA *fmadd.d* consumes 26.7 pJ, where its FP32 counterpart uses 65% less energy. This trend continues with continued energy decreases of 48%, 54%, and 49% for FP16, FP16alt and FP8, respectively. Operations on FP16alt consume 12% less energy than on FP16 due to the reduced number of mantissa bits in the former case. For the vector operations, one would intuitively expect these operations to consume roughly equal amounts of energy as they all operate on 64 bits of data. Nevertheless, the energy proportionality progression from the scalar operations carries over to the vectorial case, enabling lower energy consumption on vectored data if the underlying format is more narrow. The relative energy gains in the vectorial case are 20%, 31%, and 20% for FMA; 21%, 32%, and 21% for multiplication; 20%, 31%, and 19% for addition, and 14%, 23%, and 8% for comparisons. Thus, despite processing the same amount of bits, operations using ever-smaller vectorial formats realize super-proportional energy gains. This scaling would be less easily attainable when using a merged data path.

To sum up, scalar FMA operations can be performed for 2.5 pJ to 26.7 pJ while vectorial ones cost 1.6 pJ to 10.0 pJ per data item. At 927 MHz, this equates to a peak performance of 1.85 Gflop/s to 14.83 Gflop/s and energy efficiency of 75 Gflop/s W to 1245 Gflop/s W for the various formats supported.

As such, our divide-and-conquer architecture approach making use of aggressive clock and data gating proves to be effective at ensuring energy proportionality for both scalar and vectorial operations.

C. Example Use of Transprecision in Applications

A common pattern in many applications is the accumulation of point-wise array multiplications. A program loop executing this pattern unrolled to fit the register file consists of three integer additions for updating the source pointers as well as the loop counter, sixteen FP load instructions, eight FMA and a conditional branch. Static parts of the program include the initial accumulator initialization and loading of pointers as well as the final summation of the eight accumulator registers and store. If SIMD is used, the final accumulator must also be unpacked and summed before storing. For an array length of 1024 elements the speedup when using FP32, FP16, and FP8 instead of FP64, is 1.98x, 3.88x, and 7.30x, respectively. Processor energy gains are 2.03x, 4.08x and 7.94x for the above cases. Stall cycles only appear in the final addition of accumulators due to direct data dependencies between instructions, and are reduced for FP8 due to the shorter operation latency. Sizeable speedups can be achieved by employing transprecision across a wide range of application domains as shown in [6].

V. Conclusion and Future Work

We have presented the first silicon implementation of a transprecision FPU in a 22 nm FD-SOI process. It offers rich FP arithmetic and efficient casting and packing operations, across five different data formats, in both scalar and SIMD-vectorized variants, with high energy efficiency and proportionality. This architecture allows for energy efficiencies up to 1.24 Tflop/s W and compute performance up to 14.83 Gflop/s for 8×FP8 SIMD operation. The FPU adds tolerable overhead in terms of static energy (11.1%) and area (9.3%) to the processor. Our results provide the necessary evidence to quantify the gains achieved through algorithmic optimizations using the tools offered in the transprecision framework.

Future work comprises validation of our post-layout results, as well as the execution of more complete transprecision-enabled application kernels on manufactured silicon.

This work has received funding from the European Unions Horizon 2020 research and innovation program under grant agreement No 732631, project “OPRECOMP”. Supported in part by the European Union’s H2020 EPI project under grant agreement number 826647.

References

[1] A. C. I. Malossi, M. Schaffner, A. Molnos, L. Gammaitoni, G. Tagliavini, A. Emerson, A. Tomás, D. S. Nikolopoulos, E. Flamand, and N. Wehn, “The transprecision computing paradigm: Concept, design, and applications,” in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1105–1110.

[2] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof *et al.*, “Flexpoint: An adaptive numerical format for efficient training of deep neural networks,” in *Advances in neural information processing systems*, 2017, pp. 1742–1752.

[3] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman *et al.*, “Serving dnns in real time at datacenter scale with project brainwave,” *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[4] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.

[5] N. Amit, M. Wei, and C.-C. Tu, “Extreme datacenter specialization for planet-scale computing: Asic clouds,” *ACM SIGOPS Operating Systems Review*, vol. 51, no. 1, pp. 96–108, 2018.

[6] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benin, “A transprecision floating-point platform for ultra-low power computing,” in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1051–1056.

[7] R. K. Montoye, E. Hokenek, and S. L. Runyon, “Design of the ibm risc system/6000 floating-point execution unit,” *IBM Journal of research and development*, vol. 34, no. 1, pp. 59–70, 1990.

[8] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres, *Handbook of Floating-Point Arithmetic, 2nd edition*. Birkhäuser Boston, 2018, ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-3-319-76525-9.

[9] S. Mach, D. Rossi, G. Tagliavini, A. Marongiu, and L. Benini, “A transprecision floating-point architecture for energy-efficient embedded computing,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[10] F. Zaruba and L. Benini, “The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–12, 2019.

[11] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini, “Design and evaluation of smallfloat simd extensions to the risc-v isa,” in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, to be published.