ARCHIVIO ISTITUZIONALE DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

A Transprecision Floating-Point Architecture for Energy-Efficient Embedded Computing

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Mach, S., Rossi, D., Tagliavini, G., Marongiu, A., Benini, L. (2018). A Transprecision Floating-Point Architecture for Energy-Efficient Embedded Computing. Institute of Electrical and Electronics Engineers Inc. [10.1109/ISCAS.2018.8351816].

Availability:
This version is available at: https://hdl.handle.net/11585/652240 since: 2019-01-29

Published:

DOI: http://doi.org/10.1109/ISCAS.2018.8351816

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

14 October 2026

This is the post peer-review accepted manuscript of:

S. Mach, D. Rossi, G. Tagliavini, A. Marongiu and L. Benini, "A Transprecision Floating-Point Architecture for Energy-Efficient Embedded Computing", 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, 2018, pp. 1-5. doi: 10.1109/ISCAS.2018.8351816

The published version is available online at: https://doi.org/10.1109/ISCAS.2018.8351816

© 2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

A Transprecision Floating-Point Architecture for Energy-Efficient Embedded Computing

Stefan Mach*, Davide Rossi†*, Giuseppe Tagliavini†, Andrea Marongiu‡, and Luca Benini*†

*Integrated Systems Laboratory, ETH Zurich, Switzerland, {mach, benini}@iis.ee.ethz.ch
†DEI, University of Bologna, Italy, {davide.rossi, giuseppe.tagliavini}@unibo.it
‡DISI, University of Bologna, Italy, a.marongiu@unibo.it

***Abstract*—Ultra-low power computing is a key enabler of deeply embedded platforms used in domains such as distributed sensing, internet of things, wearable computing. The rising computational demands and high dynamic of target algorithms often call for hardware support of floating-point (FP) arithmetic and high system energy efficiency. In light of transprecision computing, where accuracy of data is consciously changed during the execution of applications, custom FP types are being used to optimize a wide range of problems. We support two such custom types - one 16 bit and one 8 bit wide - together with IEEE *binary16* as a set of "smallFloat" formats. We present an FP arithmetic unit capable of performing basic operations on smallFloat formats as well as conversions. To boost performance and energy efficiency, the smallFloat unit is extended with SIMD-style vectorization support to operate on a conventional word width of 32 bit. Finally, it is added into the execution stage of a low-power 32-bit RISC-V processor core and integrated as part of an SoC in a 65nm process. We show that the energy efficiency for processing smallFloat data in this amended system is 18% higher than the *binary32* baseline, thus enabling hardware-supported power savings for applications making use of transprecision.**

I. Introduction

An increasing amount of deeply embedded applications such as monitoring and processing of vital signs, building health profiles, and audio processing algorithms require extreme energy efficiency and complex, highly dynamic numerical computations involving double-precision (64) or single-precision (bynary32) floating-point (FP) operations [11], defined by the IEEE 754 standard. In many of these FP intensive applications, the execution of FP operations and the related memory transfers emerge as the main bottleneck for energy efficiency consuming up to 50% of the overall system power [9]. The most traditional approach to optimize energy consumption of applications requiring high dynamic range and precision in power-constrained platforms is to shift to fixed-point implementations, and adjust the dynamics and precision of operands according to the requirements of the processing chain. However, this approach is often highly intrusive, requiring in-depth understanding of the target algorithms. To trade energy for dynamic range and precision of FP operations, IEEE 754-2008 introduced a 16-bit format referred to as half-precision (*binary16*).

Several recent works propose the design of energy-efficient approximate and variable precision FP units, such as the design presented by Gautschi et al. [3] exploiting Logarithmic Number System (LNS) to approximate FP computations reducing by 4x the power consumption, Tong et al. [10] that explored an iterative, precision tunable (digit-serial) FP multiplier, Kaul et al. [6], that implemented a variable-precision FP multiply-add unit where each operand carries a 5-bit certainty field to implement automatic precision tracking. As opposed to traditional approximate computing approaches aimed at relaxing the precise-computing abstraction [1][7][5][2], a new paradigm of *transprecision computing* is emerging [8], aiming at explicitly designing systems that just deliver the required precision for intermediate computations rather than tolerating errors, creating opportunities for larger energy savings. A step towards *transprecision computing* has been proposed by Tagliavini et. al. [9], who designed a software library and an automated methodology to tune the precision and dynamic of FP operations according to the precision requirements of FP applications in programmable SoC, also exploited in this work.

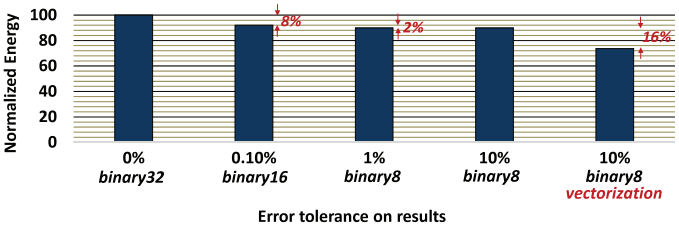


Fig. 1 Energy consumption of the KNN application for three precision requirements, normalized to *binary32* baseline

In this work we present a full SoC architecture for ultra-low-power transprecision computing. The proposed hardware architecture extends the PULPino open source SoC[1] with a transprecision FP unit integrated into the RI5CY processor core [4]. Tagliavini et. al. [9] demonstrated that significant energy savings can be achieved leveraging sub-32-bit FP formats on top of standard IEEE *binary32* and *binary16*. The proposed FP unit thus supports – along with *binary32* and *binary16* – two non-standard formats, namely *binary16alt* and *binary8*, collectively referred to as "smallFloat" formats. To fully exploit the benefit of reduced precision formats, the proposed FP unit implements Single Instruction Multiple Data (SIMD) operations on smaller-than-32-bit formats, further increasing the performance and energy efficiency of the SoC. This concept is summarized in Fig. 1, that shows the energy saving that can be achieved on a sample application (k-nearest neighbors) by shifting 32-bit operations to reduced precision FP operations while constraining the precision requirements of the whole kernel with the methodology described in [9]. It can be noted that significant energy savings can be achieved exploiting SIMD operations that, on top of the lower energy cost of reduced precision instructions, also reduce the execution time of applications leveraging data-level parallelism (column 5 of Fig. 1).

We characterize the energy consumption of the smallFloat instructions on the post place-&-route (P&R) implementation of the proposed transprecision SoC, and evaluate the execution of a set of signal processing benchmarks to assess the energy saving when tuning the precision of the operations for predefined accuracy targets. Special attention has been paid to the

[1] http://www.pulp-platform.org/

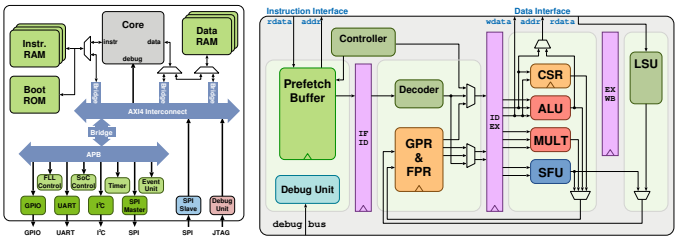


Fig. 2 Simplified architectural overview of the PULPino SoC (left) and the RI5CY core (right). The new smallFloat unit (labelled *SFU*) was implemented in the execution stage of the core.

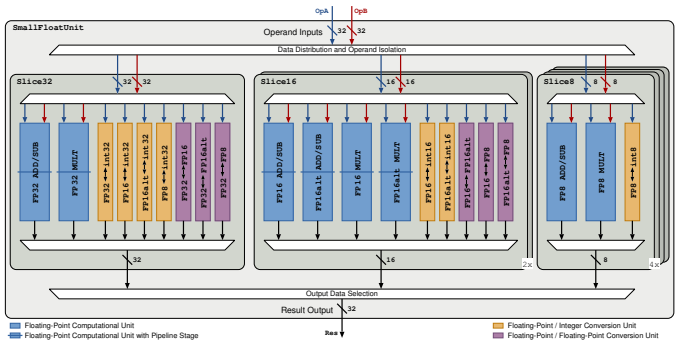
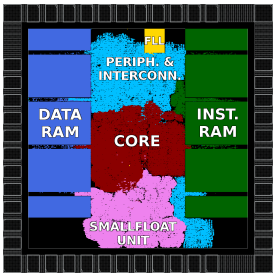


Fig. 3 Sliced architecture used for the smallFloat unit.



Chip area excluding pad instances.

16-bit Pipelined?	No	Yes
Total [mm^2]	0.906	0.872
RI5CY Core	13.5%	13.7%
smallFloat Unit	12.0%	9.0%
Data Memory	29.9%	31.1%
Instr. Memory	30.7%	31.8%
Periph. & Interc.	12.5%	13.1%
FLL	1.3%	1.4%

Fig. 4 Full layout of the SoC (left), highlighting significant blocks. The area breakdown (right) shows both pipelined and non-pipelined configurations.

design and analysis of the *binary16* and *binary16alt* datapath (i.e., pipelined vs. non-pipelined), which are widely used in most of the analyzed transprecision applications. The results of the exploration show that the pipelined 16-bit units are more energy efficient, even through the non-pipelined solution execute faster (i.e., less cycles, no stalls), mainly due to the power inflation caused by the high timing pressure on the non-pipelined datapaths. The proposed transprecision SoC improves system performance by 15% to 25% and the energy efficiency by 14% to 18% over a traditional 32-bit FP SoC on the analyzed applications, when constrained to match a maximum accuracy loss of 10% compared to the *binary32* baseline.

II. System Architecture

The transprecision SoC proposed in this work is based on the PULPino open source architecture, and shown in Fig. 2. The processor core is based on an implementation of the RISC-V instruction set architecture optimized for energy-efficient digital signal processing including custom extensions such as hardware loops, load and store with address pre- and post-increment to speed-up pointer arithmetic and lightweight support for fixed-point computations, including small SIMD instructions and saturation instructions [4]. The SoC features 4 KiB of data memory, 4 KiB of instruction memory and a bootup ROM, tightly coupled to the processor, as well as a standard peripheral set which includes SPI, I2C, UART, timers and interrupt controller.

In this work we extend the PULPino SoC with a transprecision FP unit supporting vectorization of reduced-precision operations. The hardware unit, pictured in Fig. 3, consists of three slices featuring a width of 32 bit, 16 bit and 8 bit, respectively, that support additions, subtractions and multiplications as well as conversion operations. *Binary8* is an 8-bit format featuring 3 bit of mantissa, and 5 bit of exponent, while *binary16alt* is a 16-bit format complementary to the IEEE featuring 8 bit of exponent and 8 bit of mantissa. To enable SIMD sub-word parallelism inside the unit, the narrower slices are replicated such that two 16-bit or four 8-bit FP operations can be executed simultaneously. Individual operation blocks are instantiated as Synopsys DesignWare FP Datapath IPs. Operand isolation logic is employed at the inputs of every data path in order to save dynamic power of unused subunits.

To meet the timing requirements of the SoC, 32-bit FP arithmetic operations are pipelined with one stage. Arithmetic operations in *binary8* as well as all conversion operations complete in one clock cycle. One design parameter explored in this work concerns the option of pipelining the 16-bit arithmetic operations: Although the timing requirements are met with both options, this parameter imposes a trade-off between the energy cost of 16-bit FP instructions and the number of cycles required to run applications on the system, analyzed in Section IV.

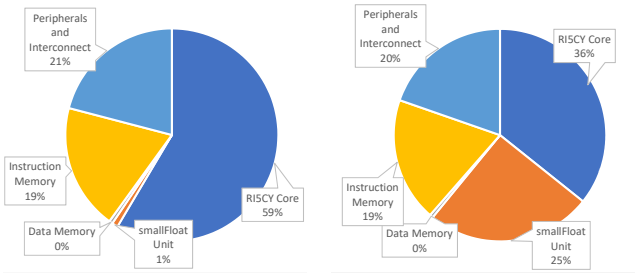
In addition to the integration of the smallFloat unit (SFU) into the execution stage of the RI5CY core, the decoder was extended with a custom set of smallFloat instructions that bear similiarity to standard RISC-V FP instructions. *Binary32* operations utilize the floating-point register file while smallFloat values are stored in the general purpose register file to make them visible to the vector shuffling hardware already present in the RI5CY core.

III. SoC Implementation

The SoC was synthesized and implemented (i.e., full layout) in the UMC 65 nm technology with Design Compiler 2015.6 and Cadence Innovus 15.2 using worst case libraries (slow-slow, 1.08 V, 125 °C) constraining the design to match the same target frequency of the original PULPino SoC (i.e., 350 MHz) [4]. Two versions of the SoC were implemented, one employing pipelining only for 32-bit operators, the other resorting to pipelining for both 32-bit and 16-bit arithmetic operations. The layout of the system is shown in Fig. 4, together with a breakdown of the area utilization of the different blocks.

The total area of the transprecision SoC is 0.906 mm^2, with the largest contributor to chip area being the data and instruction memory instances. The smallFloat unit – supporting various operations on four FP formats – makes up a significant part of the new core, filling 47% of the core area in the baseline configuration with single-cycle 16-bit operations. In the pipelined scenario, the SFU shrinks to 40% of core area since the strong timing pressure on the 16-bit FP arithmetic operations can be alleviated.

To provide an accurate estimation of the power consumption of the transprecision SoC and characterize the system-level power consumption of both original integer instructions and the new FP instructions, we conduced post-place-&-route power simulations in the typical corner (typycal-typical, 1.20 V, 25 °C). To this end, the Value Change Dump (VCD) traces of the system executing the various instructions have been generated with Mentor Modelsim 10.5c_3 and passed to Cadence Innovus to extract the power numbers. Fig. 5 shows the power breakdown of the baseline transprecision system



Integer multiplication `mul`. *Binary32* multiplication `fmul.s`.

Fig. 5 Power distribution in the baseline SoC for multiply instructions.

TABLE I Average energy per operation extracted from post-layout simulation, for various instruction groups on the two configurations of the transprecision SoC.

Format	Operation	Instruction	16-bit pipelined? No	Yes
	Idle Cycle	`nop`*	62.2 pJ	62.9 pJ
int32	Data movement	`lw,sw`*	94.4 pJ	94.6 pJ
	Arithmetic	`add, mul`*	106.4 pJ	102.4 pJ
binary32	Arithmetic	`f{add,mul}.s`*	106.8 pJ	102.4 pJ
	Conversions	e.g. `fcvt.w.s`*	79.7 pJ	78.1 pJ
binary16	Arithmetic	`f{add,mul}.h`†	98.8 pJ	82.0 pJ
	Conversions	e.g. `fcvt.h.s`†	74.7 pJ	74.6 pJ
	Vector Arithmetic	`vf{add,mul}.h`†	132.6 pJ	93.9 pJ
	Vector Conversions	e.g. `vfcvt.x.h`†	86.4 pJ	77.6 pJ
binary16alt	Arithmetic	`f{add,mul}.ah`†	87.2 pJ	83.2 pJ
	Conversions	e.g. `fcvt.ah.s`†	73.5 pJ	73.7 pJ
	Vector Arithmetic	`vf{add,mul}.ah`†	108.9 pJ	92.7 pJ
	Vector Conversions	e.g. `vfcvt.x.ah`†	79.5 pJ	74.3 pJ
binary8	Arithmetic	`f{add,mul}.b`†	74.0 pJ	75.5 pJ
	Conversions	e.g. `fcvt.b.s`†	72.5 pJ	72.8 pJ
	Vector Arithmetic	`vf{add,mul}.b`†	95.2 pJ	94.1 pJ
	Vector Conversions	`vfcvt.x.b`†	77.8 pJ	74.0 pJ

* RISC-V mnemonic
† Custom smallFloat mnemonic, based on RISC-V mnemonic

in the non-pipelined configuration, while the energy cost of pipelined and non-pipelined instructions is shown in Table I.

Fig. 5 outlines the power consumption of the major blocks in the system when running multiplication instructions on either intergers or 32-bit FP values. It should be noted that data memory is unused when executing an arithmetic instruction in isolation and thus has negligible impact on system power. When an integer multiplication is performed nearly 60% of the system power is used inside the ALU, while the SFU is isolated and clock gated, thus only contributing insignificant static power. During FP multiplications, the SFU consumes 25% of the system power while the ALU power consumption is reduced by approximately the same amount, indicating that the power consumption of our FP multiplier is similar to its integer counterpart.

Table I showcases the significant energy use of data movement which is comparable to arithmetic operations themselves, generally even overshadowing the operations on smallFloat types in the pipelined scenario. Vectorizing arithmetic operations costs up to 35% more energy at system level, albeit at double or quadruple throughput, depending on the format used. Furthermore, vectorization directly reduces the number of load and store operations in the same way, drastically reducing the energy spent on a single value during a load-execute-store cycle. The impact of added timing pressure in the non-pipelined baseline over the pipelined configuration is clearly visible, with an average 7% higher energy consuption, up to 40% in some cases. However, the baseline instructions complete within a single cycle, alleviating the need for stall cycles in difficult-to-schedule applications. Since there is a trade-off to be made between total number of cycles and energy used per cycle which depends on the schedule friendliness of the target application, a set of benchmarks was run in order to explore this design space.

IV. Benchmarking

Benchmarking of the transprecision SoC has been performed on a set of applications which implement algorithms for two domains of ULP systems, near-sensor computing and embedded machine learning. SmallFloat types have been introduced in the source code using the methodology from [9], where a software library (FlexFloat) emulates arbitrary FP types while an external tool (fpPrecisionTuning) selects for each variable the smallest FP type among the supported ones meeting strict constraints on the result accuracy. The accuracy of results is expressed as a value of signal-to-quantization-noise ratio (SQNR) that program outputs must satisfy. In addition, FlexFloat features a detailed run-time report on FP operations, which provides the number of executions classified by FP type, arithmetic operator and class (i.e., scalar, vectorial, cast). The ANSI-C programs have been compiled using GCC 5.2 with a RISC-V backend optimized for PULPino [4], featuring support for single-precision FP types as defined in the RISC-V instruction set architecture (ISA). To perform a wide exploration on large applications that would require long simulation time on the RTL platform, binaries have been executed on the PULPino virtual platform which is cycle accurate and provides detailed statistics. Since the current version of GCC does not include the support for the extended instruction set needed to handle *binary16*, *binary16alt* and *binary8* formats, we have used the *binary32* type to measure the exact number of cycles required by each instruction to execute. This value depends on the ability of the compiler to schedule other classes of operations to fill latency cycles and avoid stalls in the core pipeline, so it is strictly dependent on both application and compiler back-end.

To assess the trade-off between the pipelined and non-pipelined solution for *binary16* units, we have made an analytic exploration varying the percentage of latency slots filled for every application. Fig. 6 depicts the ratio between the energy consumption of the pipelined design (Pipelined Energy) over the energy consumption of the non-pipelined design (Unpipelined Energy) for the analyzed applications. The horizontal axis reports the percentage of instructions that require a latency slot, which varies from 0% (no latency cycles for FP instructions) to 100% (a latency cycle per pipelined FP instruction). The aim of this experiment is to explore a full range of compiler capabilities, to understand the trade-off between the energy/instruction (pipelined is better) and number of execution cycles (non-pipelined is better). This trend may be better explained by examining a breakdown analysis of operation based on FP classes, which is depicted in Fig. 7. The ratio of applications that require a high amount of 16-bit vectorial operations w.r.t. scalar ones (DWT, SVM) is growing with the number of latency slots with a limited slope, since the energy savings of SIMD operation in the pipelined version are relatively higher than in the non-pipelined case. The ratio of applications with a predominance of *binary32* or *binary8* operations (JACOBI, KNN) is nearly constant since these operations are pipelined the same way in both designs. Finally, the ratio of applications that are characterized by a relatively large 16-bit scalar workload (PCA, CONV) is particularly affected by the number of latency slots and their slopes are steep, since scalar operations are heavily penalized by the pipelined design. However, if the compiler were able to reduce the latency slots under 70% using instruction scheduling techniques, the

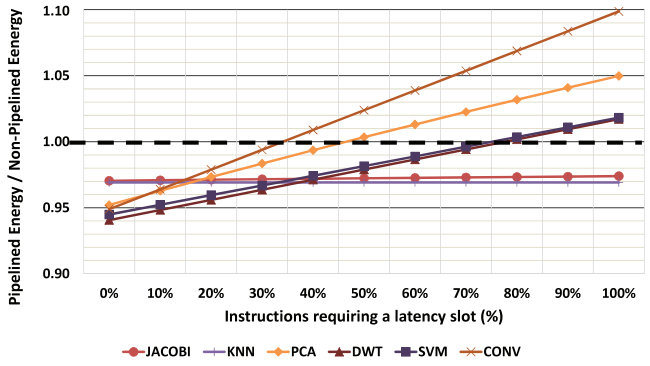


Fig. 6 Energy of applications running on the *binary16* pipelined design normalized to that of non-pipelined design when varying the percentage of latency slots in in *binary16* and *binary32* operations.

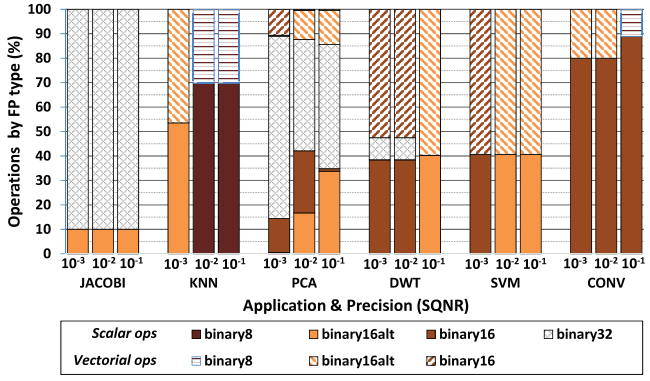


Fig. 7 Breakdown of FP operations for three precision requirements.

negative effect of these cases would be highly mitigated. Compiling the baseline version of applications (which uses *binary32* scalars) we measured a number of latency slots between 50% and 80%, so we conclude that the pipelined design is in general the best solution.

Fig. 8 depicts a groups of bars for each application running on the pipelined architecture, reporting a breakdown of the executions cycles for three precision requirements (SQNR = 10^{-3}, 10^{-2}, 10^{-1}). The bottom contributions take into account the best execution scenario with no stalls due to latency cycles (0% latency), while the gray segment on top considers the worst case, in which each operation involving 16-bit and 32-bit FP types requires a stall (100% latency). The reported values are normalized to the *binary32* version of the application, and operation classes are highlighted with distinct patterns. The performance improvements and energy savings due to the utilization of transprecision operations are mainly due to the vectorization which allows to execute multiple reduced precision operations in parallel and reduce the number of memory accesses, and due to the smaller energy cost of reduced precision FP instructions. The overhead is mainly caused by the cast operations required to dynamically move from one FP format to another. In Fig. 8 we see that on average, the number of cycles is decreased by 15% and 25% for 100% and 0% latency slots, respectively. The number of cycles reported for JACOBI is higher than the original version, since this application only uses a limited number of *binary16alt* variables limited to disjoint program regions, and this behavior introduces a high number of casts. In other benchmarks we can observe that the overhead of cast operations is not relevant.

Fig. 9 reports the energy consumption of each application, normalized to the *binary32* baseline. Each bar contains three contributions, the FP operations (FP ops), the memory accesses (Memory ops) and all the remaining instructions (Other ops). These values are strictly related to the ones shown in Fig. 8,

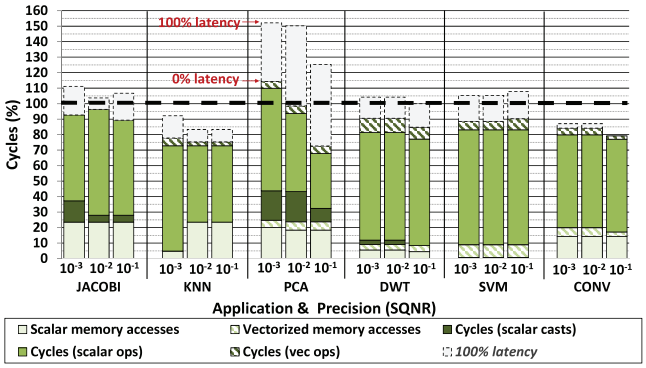


Fig. 8 Execution cycles of applications for three precision requirements and assuming two latency slots conditions (0%, 100%), normalized to *binary32* baseline

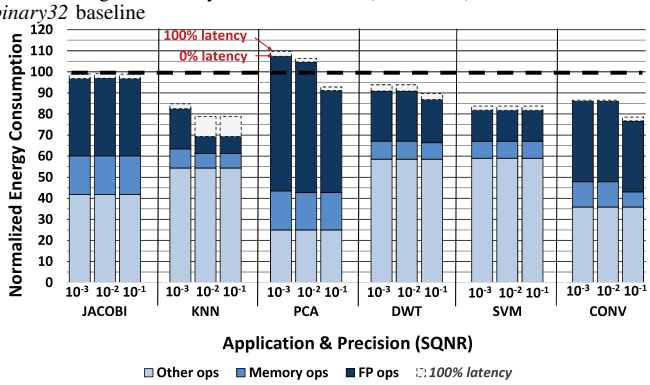


Fig. 9 Energy consumption of applications for three precision requirements and assuming two latency slots conditions (0%, 100%), normalized to *binary32* baseline

and also in this figure the contribution to energy consumption due to stalls is shown on top (100% latency). The energy consumption of PCA is greater than the baseline, due to the high number of casts coupled with a predominant number of scalar operations on *binary32* values. The other applications have average energy savings between 14% and 18% compared to the baseline, with a maximum of 31% measured for KNN.

V. Conclusion

This work we presented a SoC for ultra-low-power transprecision computing extending the PULPino microcontroller architecture with a transprecision floating-point unit integrated into the RI5CY processor core [4]. The FP unit supports, along with IEEE *binary32*, the IEEE *binary16* FP format and two additional formats. To fully exploit the benefit of reduced precision formats, the proposed transprecision floating-point unit implements Single Instruction Multiple Data (SIMD) operations on smaller-than-32-bit formats, further increasing energy efficiency and performance of the SoC. We have characterized the energy consumption of the smallFloat instructions on the post P&R implementation of the proposed transprecision SoC in UMC 65 nm technology, and evaluated the execution of a set of signal processing benchmarks on the proposed system. The results of our exploration show that the introduction of a smallFloat unit improves system performance by 15% to 25% and the energy efficiency by 14% to 18% for 100% and 0% latency slots, respectively, on the analyzed applications when allowing the target precision to be relaxed by 10% compared to a traditional *binary32* baseline.

Acknowledgements

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 732631 (OPRECOMP).

REFERENCES

[1] C. Bekas, A. Curioni, and I. Fedulova. Low-cost data uncertainty quantification. *Concurrency and Computation: Pract. & Exper.*, 24(8):908–920, 2012.

[2] W.-F. Chiang, M. Baranowski, I. Briggs, A. Solovyev, G. Gopalakrishnan, and Z. Rakamarić. Rigorous floating-point mixed-precision tuning. In *Proc. of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, pages 300–315. ACM, 2017.

[3] M. Gautschi, M. Schaffner, F. K. Grkaynak, and L. Benini. An Extended Shared Logarithmic Unit for Nonlinear Function Kernel Acceleration in a 65-nm CMOS Multicore Cluster. *IEEE Journal of Solid-State Circuits*, 52(1):98–112, 2017.

[4] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Grkaynak, and L. Benini. Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2700–2713, Oct 2017.

[5] N.-M. Ho, E. Manogaran, W.-F. Wong, and A. Anoosheh. Efficient floating point precision tuning for approximate computing. In *22nd Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pages 63–68. IEEE, 2017.

[6] H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, F. Sheikh, R. Krishnamurthy, and S. Borkar. A 1.45GHz 52-to-162GFLOPS/W variable-precision floating-point fused multiply-add unit with certainty tracking in 32nm CMOS. In *IEEE Int. Solid-State Circuits Conf.*, pages 182–184, 2012.

[7] P. Klavík, A. C. I. Malossi, C. Bekas, and A. Curioni. Changing computing paradigms towards power efficiency. *Phil. Trans. R. Soc. A*, 372(2018), 2014.

[8] C. Malossi, M. Schaffner, A. Molnos, L. Gammaitoni, G. Tagliavini, A. Emerson, A. Toms, D. S. Nikolopoulos, E. Flamand, and N. Wehn. The Transprecision Computing Paradigm: Concept, Design, and Applications. *DATE 2018*, 2018.

[9] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini. A Transprecision Floating-Point Platform for Ultra-Low Power Computing. *DATE 2018*, 2018.

[10] J. Y. F. Tong, D. Nagle, and R. A. Rutenbar. Reducing power by optimizing the necessary precision/range of floating-point arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):273–286, 2000.

[11] D. Zuras, M. Cowlishaw, A. Aiken, M. Applegate, D. Bailey, S. Bass, D. Bhandarkar, M. Bhat, D. Bindel, S. Boldo, et al. IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.