

Florian: Developing a Low-power RISC-V Multicore Processor with a Shared Lightweight FPU

Jina Park^{†*}, Kyuseung Han^{†*}, Eunjin Choi[†], Sukho Lee[‡], Jae-Jin Lee[‡], Woojoo Lee^{†#}, and Massoud Pedram[§]

[†] School of Electrical and Electronics Engineering, Chung-Ang University, Korea

[‡] AI SoC Research Division, Electronics and Telecommunications Research Institute (ETRI), Korea

[§] Department of Electrical and Computer Engineering, University of Southern California, USA

Abstract—As applications running on lightweight RISC-V processors become increasingly diverse and complex, the need for multicore processors supporting floating-point units (FPUs) is rising, making processor designs using existing open-source RISC-V cores challenging. With the exception of a very few, most open lightweight RISC-V cores are integer cores without FPUs, which greatly reduces the design exploration space, making it impossible to design a processor optimized for each application. For example, most of these applications mainly perform integer operations, but occasionally perform floating-point operations. For them, a multicore processor with FPU per core is overkill and wastes power, which is a critical problem for processors where low-power design is paramount. To address the problem, we propose an external lightweight FPU that can be attached to any RISC-V integer core and a low-power multicore architecture using the designed FPU. For verification, we designed a RISC-V processor that implements all the proposed technologies, prototyped it on an FPGA device, and finally fabricated it as a System-on-Chip. Through experiments, it was confirmed that the proposed technology can cut energy consumption energy by up to 23%.

I. INTRODUCTION

Explosive interest in RISC-V from academia and industry has driven dozens of free and open core releases built on this instruction set architecture [1], [2], [3]. Among them, especially lightweight RISC-V cores are finding increasing use in processors specialized in IoT, wearable, and embedded system applications [4], [5], [6]. These applications are becoming more diverse and complex, and as a result, the number of the applications that require a multicore processor supporting a floating-point unit (FPU) is growing exponentially [7], [8], [9], [10].

Unfortunately, it is very hard to develop a multicore processor supporting FPUs optimized for a target application by utilizing currently available lightweight RISC-V cores. This is due to the fact that most cores are integer cores except for very few cores with FPUs. Even if there is an integer core in the pool that best fits the target application, it is practically impossible to add an FPU to this core. This because the RTL

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grants funded by the Korea government(MSIT) (No. 2022-0-00957, Distributed on-chip memory-processor model PIM semiconductor technology development for edge applications, and No. 2022-0-00971, Logic Synthesis for NVM-based PIM Computing Architecture).

*Jina Park and Kyuseung Han contributed equally to this work.

#Corresponding authors: Woojoo Lee (space@cau.ac.kr)

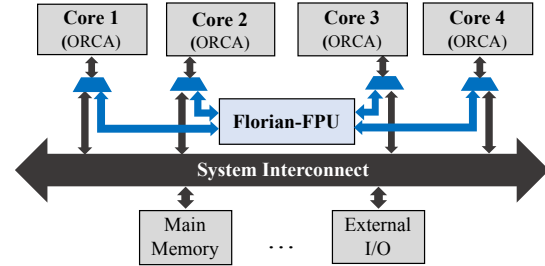


Fig. 1. Concept of the Florian quadcore processor architecture.

code for that core may not be disclosed, and even if it is, adding FPU to an already completed core is as difficult as designing a new core [11], [12].

As a consequence, there are only a handful of lightweight RISC-V cores supporting a FPU to choose from, namely Rocket [13] and PULP [14], whose inflexibility poses an immediate limit to the development of new multicore processors using them. More specifically, Rocket core is difficult to use for new platform development due to its high platform dependencies. Rocket cores are provided as chip platforms, because of which it is very difficult from an optimization design perspective to create a new processor by extracting only Rocket cores. For example, we compared the performance of a newly created processor by extracting only the Rocket core from the Rocket-Chip platform and a processor made with the ORCA [15] core, which is much lighter than the Rocket core. When each FPGA prototype processor ran a EEMBC Coremark-Pro [16] benchmark program that only performs integer operations, although the Rocket-based processor used about 4 times more FPGA resources than the ORCA-based processor, the execution time was 630 and 372 sec, respectively, it can be seen that serious performance degradation has occurred in the Rocket-based processor.

Since each core has an FPU, there is a limit to designing a low-power multicore processor using PULP. Most IoT, wearables, and embedded system applications that require FPU actually perform integer operations mainly, but only infrequently perform floating-point operations. For these applications, a multicore processor with one FPU per core is over-spec, which wastes power. This is a fatal drawback for lightweight processors, where low-power design is the most important design concern.

In this paper, we present a method to develop the FPU-

enabled low-power multicore processors using existing open RISC-V integer cores, which is called *Florian (FPU librarian)* method. The proposed Florian includes an external lightweight FPU that can be attached to any RISC-V integer core and a low-power multicore architecture using the designed FPU. Fig. 1 is an example of a quadcore processor architecture that shows the concept of Florian, where four RISC-V integer cores (ORCA cores) share an out-of-core lightweight FPU, called *Florian-FPU*, to handle each floating-point operation. In addition to Florian-FPU, we have designed and implemented an architecture that allows each core to share the FPU efficiently, performed RTL simulation, and completed functional verification through FPGA prototyping. And finally, we have fabricated the processor in Fig. 1 into a chip using the 110 nm process technology and utilized the chip for performance evaluation. Through intensive experiments, we have confirmed that the proposed method can improve energy efficiency by 10.9%, 19.2%, and 23.1% in dualcore, quadcore, and octacore processors, respectively.

II. FLORIAN

The proposed Florian method utilizes an external lightweight FPU, which is deployed independently of the core, so that it is possible to design a processor supporting FPU using existing open cores without changing their structures at all. This method excels especially in the development of multicore processors, enabling energy-optimized designs. This is due to the fact that applications running on recent light-weight processors require floating-point operations, but they account for a small portion of the entire program [17]. Moreover, in the case of AI applications for state-of-the-art edge computing, it is a pragmatic development trend to convert floating-point operations to integer operations as much as possible to reduce processor power consumption [18]. Of course, there are still some floating-point operations left in these applications, which can be covered by a single, lightweight FPU, namely Florian-FPU. Thus, Florian allows processor designers to choose the most customized open core for their applications and design the most energy-efficient processors consuming minimal FPU power.

Fig. 2 shows the structure of the proposed Florian architecture. As shown in the figure, this architecture consists of two main blocks, the aforementioned Florian-FPU, and the Florian-Arbiter. To briefly explain the blocks prior to the detailed description of each, first, Florian-FPU is connected to each core by an independent path without going through the system interconnect. In order to reduce the design difficulty that may arise from the method of devising a new external port on the core for that connection and connecting the entire signal through it, Florian-FPU is designed to easily connect the core using Memory Mapped Input Output (MMIO). In addition, since high-performance communication is not required, the advanced peripheral bus (APB) protocol is simply used. Next, Florian-Arbiter plays a key role in allowing multi-cores to share and use a single FPU. Florian-Arbiter has a FIFO to prevent signal collisions that may occur in communication between cores and the FPU and to sequentially designate the FPU occupancy order of cores for continuous utilization of the

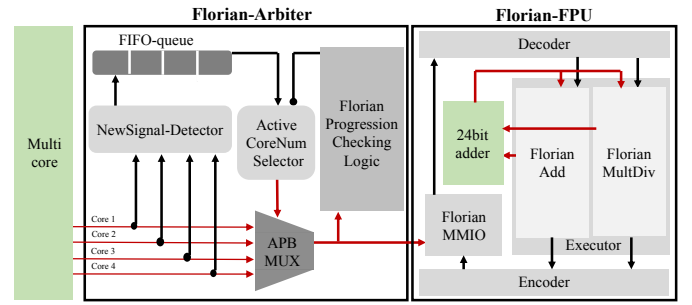


Fig. 2. Block diagram of the proposed Florian architecture.

FPU. More detailed explanations on the structure and design of Florian-FPU and Florian-Arbiter are described through the following subsections.

A. Florian-FPU

The left part of Fig. 3 shows the detailed internal structure of Florian-FPU, consisting of major blocks of MMIO, decoder, encoder, and Executor. Since the Florian MMIO module manages all inputs and outputs of Florian, it can define whether a given signal is an operational command, an operand transfer, or a request for a calculation result. When a signal is defined as an operand, it is converted from a 32-bit IEEE754 [19] format to a 48-bit denormalized format through the decoder, resulting in a structure suitable for floating-point calculations. When the operation is completed through the Executor, the generated denormalized result is properly converted into IEEE754 format through the encoder and then sent back to the core along with the corresponding APB signal.

Florian-FPU is designed to support only addition, subtraction, multiplication and division operations for light weight, and *Executor* is in charge of these arithmetic parts. The right part of Fig. 3 shows the detailed structure of the Executor architecture, which consists of the *Florian-Add* and *Florian-MultDiv* modules. The modules in Executor receive 24-bit *adder_result* and *core_ALU_result* signals, the former from a 24-bit adder located outside the Executor, and the latter from *Concatenator* via the core. In other words, the signals are not generated internally, but are generated externally, which allows us to design a low-power FPU using minimal resources.

To elaborate on the low-power FPU design, for floating-point addition, a 24-bit adder is needed to add two 24-bit mantissa values. Also, since the multiplication and division require enough significant digits during the calculation of the two 24-bit mantissa values, 48-bit multiplier and divider are required to obtain accurate results. Plus, the floating-point multiplication/division requires an 8-bit adder to add or sub the two exponents. When all these calculation modules are built into the FPU, significant power consumption is inevitable. To address this issue, we noted that all the aforementioned calculations are only integer operations, so they can all be performed by the core as well. That is, we devised a method to handle 48-bit multiplication and division using the integer ALU in the core instead of processing it inside the FPU, and ported a 24-bit adder inside the FPU and shared it with *Florian_MultDiv* for the 8-bit exponential calculation. As a result, although the execution time inevitably increased slightly

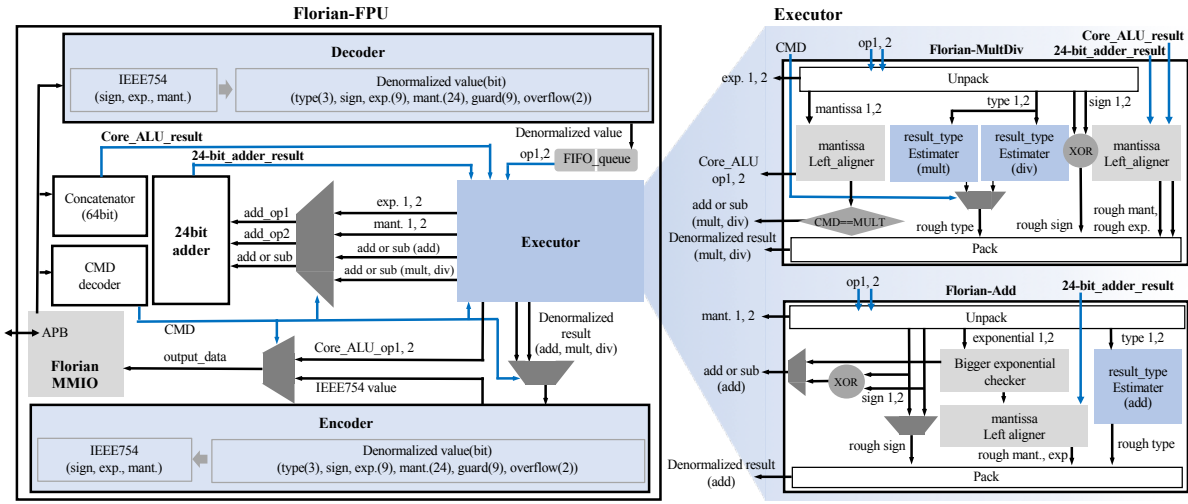


Fig. 3. Structure of the Florian-FPU architecture.

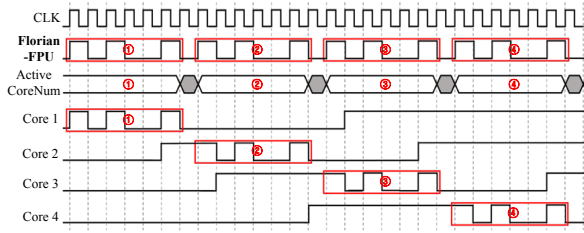


Fig. 4. Waveform of access between Florian-FPU and cores

as additional communication between the core and the FPU was required, energy savings was achieved because the effect of reducing power consumption due to the FPU resource savings was greater than that.

B. Florian-Arbiter

The low power effect of Florian is that multiple cores share the Florian-FPU. To do this, as shown in Fig. 4, only one core must be granted access to the FPU at a time, and the other cores must wait in sequence until the currently executing core completes its task and returns the privilege. This is different from normal bus operation and requires a dedicated arbiter, Florian-Arbiter.

Fig. 5 shows the internal structure of the Florian-Arbiter. The APB signal from each core enters the arbiter, allowing the arbiter to check all signals and select the core to occupy the FPU via the *APB-MUX* and the selected signal *ActiveCoreNum*. *ActiveCoreNum* represents the index of the current Florian occupied core, defined by two variables: *NextCoreNum*, the index of the highest priority core in the queue, and *is_florian_calculating*, the activation signal of the *ActiveCoreNum-Selector*. When *NewSignal-Detector* detects the receipt of a new signal, the core ID of the detected signal is sent to the *FIFO-queue* for updates. Of the list of core IDs waiting in the queue, the one with the highest priority is assigned as *NextCoreNum* and gets a chance to be assigned as *ActiveCoreNum*.

NextCoreNum is not always immediately assigned to *ActiveCoreNum*. The Florian-FPU needs to interact with

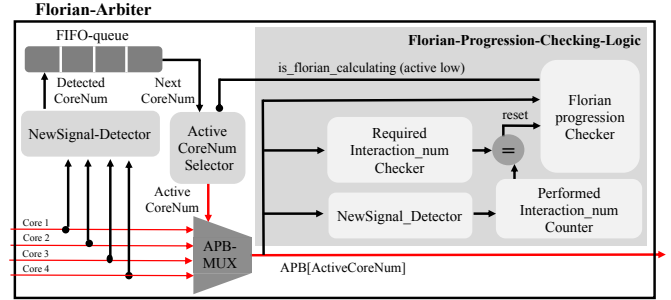


Fig. 5. Structure of the Florian-Arbiter architecture

the core at least three times to complete all floating point operations. So if a signal from another core gets into the FPU before it fills in the required amount of interactivity, it will result in an error. We introduced *Florian-Progression-Checking-Logic* to prevent this error, which serves to check that the currently active cores have communicated sufficiently as needed. After all necessary interactions have been performed, *Florian-Progression-Checking-Logic* can change the register *is_florian_calculating* to 0 to enable *ActiveCoreNum-Selector* and finally update *ActiveCoreNum* to *NextCoreNum*. Arbitrators designed in this way help each core successfully utilize the Florian-FPU, ensuring that all cores use it fairly.

C. Florian operational process

Fig. 6 shows the operation process of floating-point addition (*Florian_ADD*) and multiplication (*Florian_MULT*). First, for *Florian_ADD*, as soon as the FPU receives the ADD command (cf. 1st line), the next two operands are converted to denormalized form and sent to the *Florian-Add* module in *Executor* for execution.

Florian_MULT requires a slightly more complex procedure. As aforementioned, multiplication in Florian-FPU requires additional interaction with the core due to integer arithmetic. To obtain the result of a 48-bit multiplication of two given mantissas, we use the MUL and MULHU instructions

Function : <Florian_ADD>			
1 :	sw	cmd_add,	florian_mmio_input_addr
2 :	sw	operand1,	florian_mmio_input_addr
3 :	sw	operand2,	florian_mmio_input_addr
4 :	lw	result,	florian_mmio_output_addr

Function : <Florian_MULT>			
1 :	sw	cmd_mult,	florian_mmio_input_addr
2 :	sw	operand1,	florian_mmio_input_addr
3 :	sw	operand2,	florian_mmio_input_addr
4 :	lw	multiplcand,	florian_mmio_output_addr
5 :	lw	multiplier,	florian_mmio_output_addr
ALU-1 :	mul	product_lower32bit,	multiplcand, multiplier
ALU-2 :	mulhu	product_upper32bit,	multiplcand, multiplier
6 :	sw	product_lower32bit,	florian_mmio_input_addr
7 :	sw	product_upper32bit,	florian_mmio_input_addr
8 :	lw	result,	florian_mmio_output_addr

Fig. 6. Operation process of addition and multiplication of Florian-FPU.

to produce the 64-bit result, then return the value to the FPU so that the multiplication operation in progress can complete. Therefore, a total of 8 interactions between the core and the FPU are required.

In Florian-FPU, floating-point division is performed in a similar way to the multiplication, as are other floating-point operations.

III. EXPERIMENTAL WORK

First, to verify the functional correctness of the Florian-based processor, we designed a test processor with the structure shown in Fig. 1 consisting of four ORCA cores and Florian-FPU/Arbiter and prototyped it on a Xilinx Kintex Ultrascale+ FPGA board [20]. We confirmed that the processor operates normally by running applications that include floating-point operations on the prototype processor. In the FPGA prototyping process, the FPGA resource consumption of each major component was compared, which is shown in Table I. As can be seen from the table, the Florian-FPU is lightweight, with only 37.1% of the resources consumed by the single ORCA core and 32.2% of the FPU in Rocket. Moreover, this result is a comparison of only single cores, and if the number of cores in the processor increases further, it is obvious that the difference between the resource consumption of FPU in multicore with FPU per core and multicore using only single Florian FPU will be even greater, which means that the low-power effect of Florian is more pronounced.

Next, we designed and fabricated a prototype processor on a chip. The chip was fabricated in the size of $4500 \times 4500 \mu m^2$ using the $0.11 \mu m$ CMOS logic/mixed signal process technology, and the die photo and detailed specifications of the chip are shown in Fig. 7. The power consumption of the ORCA core, Florian-FPU and Arbiter measured from the chip are reported in Table II. All the power values are normalized based on the power value of a single ORCA core. As seen in the table, Florian-FPU and Arbiter consume only 0.3 times and 0.04 times the power of the ORCA Core, respectively, it can be confirmed that the proposed Florian hardware was successfully designed for low power consumption.

We then conducted experiments to evaluate the energy saving of Florian on several multicores with different numbers of cores. To this end, we needed to design a single, dual, quad and

TABLE I
COMPARISON OF FPGA RESOURCE CONSUMPTION.

	Rocket		ORCA	Florian-	
	Core	FPU	core	FPU	Arbiter
LUTs	11711	5456	3230	1546	144
FFs	6653	1775	3056	558	89

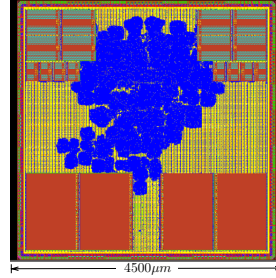
	Technology:	0.11μm CMOS Logic/ Mixed Signal Process
	Chip Size:	4500 × 4500 μm ²
	Gate Count:	778K GE
	Memory	SRAM: 294 kbytes
	Supply Voltage	1.2V / 3.3V
	Target Freq.	50MHz
	Temp. Range	from -40°C to 125 °C
	Pads	Signal: 116, Power & ground: 32
	Package	168 FBGA

Fig. 7. Die photo & spec. of the Florian prototyping chip.

octa core processor with one Florian-FPU shared, each core has a Florian-FPU, or no FPU, i.e., a total of 11 processors were necessary. For reference, processors without an FPU use soft-float. At this time, manufacturing all processors into chips requires excessive time, effort, and cost, so the execution time and energy consumption of each processor were accurately estimated using the experimental results of the fabricated chip of the quad core processor, execution time of applications in each FPGA prototype processor, and the synthesis results using the $110 nm$ CMOS technology through Synopsys design compiler. In addition, to evaluate the energy efficiency of the processor with the proposed method depending on the workload of the application's floating point computation, i.e., the lower the floating point workload, the more suitable it is to apply Florian, we combined the benchmark programs of EEMBC Coremark-Pro [16] to create 11 testbench applications where the ratio of floating point computation and integer computation varies from 1:1 to 1:20, respectively. Table III lists the created testbench applications.

Derived from the experiments we performed, Table IV shows the results of running the application on a quad-core processor with no FPU (Case-I), a Florian-FPU on every core (Case-II), and a shared Florian-FPU (Case-III). At this time, the exp10 testbench was used for the application, which was executed simultaneously for all cores. As a result, the execution time of Case-II was reduced by 28% compared to Case-I, indicating that Florian effectively handles floating-point operations. On the other hand, it can be confirmed that the execution time of Case-III is increased than that of Case-II. This is because all cores perform the same application, so all cores perform floating-point operations at the same time,

TABLE II
POWER CONSUMPTION (NORMALIZED BASED ON ORCA).

	$P_{dynamic}$	$P_{leakage}$	P_{total}
ORCA	0.96	0.04	1
Florian_FPU	0.299	0.014	0.31
Florian_Arbiter	0.035	0.0016	0.04

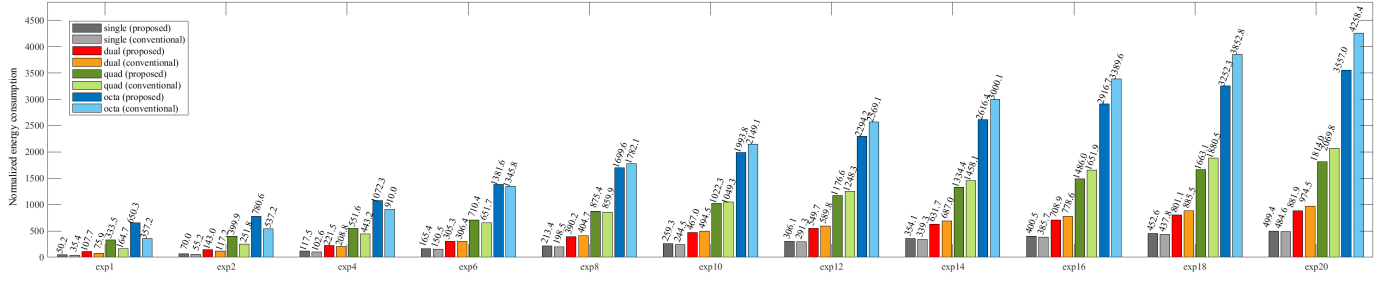


Fig. 8. Energy consumption results of the proposed and conventional single, dual, quad, and octa core processors for 11 applications (each core in the processor runs the same application concurrently).

TABLE III
THE 11 TESTBENCH APPLICATIONS USED IN THE EXPERIMENT.

App.	Workload ratio (FP : Others)	Included EEMBC programs
exp1	1 : 1	linear_alg-mid-100x100-sp, cjpeg-rose7-preset
exp2	1 : 2	linear_alg-mid-100x100-sp, cjpeg-rose7-preset, sha-test
exp4	1 : 4	linear_alg-mid-100x100-sp, sha-test
exp6	1 : 6	linear_alg-mid-100x100-sp, cjpeg-rose7-preset, sha-test
exp8	1 : 8	linear_alg-mid-100x100-sp, cjpeg-rose7-preset, sha-test
exp10	1 : 10	linear_alg-mid-100x100-sp, sha-test
exp12	1 : 12	linear_alg-mid-100x100-sp, sha-test
exp14	1 : 14	linear_alg-mid-100x100-sp, sha-test
exp16	1 : 16	linear_alg-mid-100x100-sp, sha-test
exp18	1 : 18	linear_alg-mid-100x100-sp, sha-test
exp20	1 : 20	linear_alg-mid-100x100-sp, sha-test

TABLE IV
ENERGY CONSUMPTION COMPARISON OF THE QUAD CORE PROCESSORS.
POWER VALUES ARE NORMALIZED BASED ON A SINGLE ORCA
PROCESSOR.

	Case-I : ORCA × 4	Case-II : (ORCA + Florian) × 4	Case-III : ORCA × 4 + Florian
Power	3.64	5.28	4.10
Exec. Time (sec)	279	199	249
Energy	-	1049	1022

causing a bottleneck in the FPU. However, Case-III can still save 2.57% energy compared to Case-II, because the power consumption can be reduced by sharing the FPU.

Finally, to clearly demonstrate that using the proposed Florian is more beneficial in terms of low-power design than using cores with FPU per core in the multicore development, we report energy comparisons between processors with Florian (denoted as *proposed*) and processors with FPU per core (denoted as *conventional*). For the conventional processors, since the ORCA core does not have an FPU, we assume that each core is equipped with Florian-FPU as a conservative approach. To this end, we manually calculated the application execution time on the conventional processors based on the communication overhead between Florian and Core obtained through RTL simulations.

Fig. 8 shows the energy consumption results for each processor and each application when all cores in the processor simultaneously execute the same application. First, looking at the results according to the change in the number of cores of the processors, the proposed processors can reduce energy consumption by 9.5%, 12.4%, and 16.5% in dual, quad, and octa cores, respectively, which, as expected, increase the

TABLE V
ENERGY SAVING (ES) RESULTS WHEN EACH CORE RUNS THE SAME
APPLICATION SIMULTANEOUSLY

App.	$ES_{single}(\%)$	$ES_{dual}(\%)$	$ES_{quad}(\%)$	$ES_{octa}(\%)$
exp1	-41.9	-41.9	-102.5	-82
exp2	-26.8	-22.1	-58.8	-45.3
exp4	-14.4	-6.1	-24.5	-17.8
exp6	-9.84	0.36	-9	-2.66
exp8	-7.46	3.6	-1.8	4.63
exp10	-6.06	5.56	2.57	7.22
exp12	-5.09	6.81	5.74	10.7
exp14	-4.37	8.05	8.48	12.8
exp16	-3.84	8.95	10	14
exp18	-3.38	9.52	11.6	15.6
exp20	-3.06	9.49	12.4	16.5

benefit as more cores share Florian-FPU. For single cores, no gain occurs in all processors, as we conservatively assumed that ORCA core has a built-in Florian-FPU, i.e., the conventional processors is assumed to already have the advantage of introducing the lightweight FPU, and since the FPU is built in, communication overhead is very low compared to external ones. In other words, the benefits of Florian's lightweight FPU are excluded from the results here. Therefore, if it were not for this conservative assumption, it is possible to expect that Florian could have made some gains even in a single core, and it is clear that the gains in dual, quad, and octa cores would have increased further.

Looking at Table V, which reports the change in energy saving (ES) factor of Florian as the floating-point operation workload varies by application, it can be seen that the smaller the proportion of floating-point operations, the greater ES . In the case of exp20, in contrast to the maximum gain in dual, quad, and octa cores, in exp1, losses occur at -41.9%, -102.5%, and -82.0%, respectively. This is because, as the cores share the FPU, if many cores request the use of the FPU at the same time, a bottleneck occurs, inducing delays. Since all of the lightweight applications targeted in this paper have a very small proportion of floating-point arithmetic, it is obvious that there must be benefits through Florian, but if not, it would be better to design a processor with an FPU for each core.

Fig. 8 and Table V are the results of all cores in the processor performing floating point operations at the same time, which may be too far from the actual behavior of the multicore. In other words, it is very likely that the timing required for floating-point operation is different for each core in multicore in general. To reflect this, we modified the test-

TABLE VI
ENERGY CONSUMPTION AND SAVINGS RESULTS OF THE PROPOSED AND CONVENTIONAL PROCESSORS FOR 11 TESTBENCH APPLICATIONS
(APPLICATIONS ARE SET TO DIFFERENTIATE THE TIMING OF PERFORMING FLOATING-POINT OPERATIONS PER CORE).

App.	Dual core			Quad core			Octa core		
	$E_{proposed}$	$E_{conv.}$	ES (%)	$E_{proposed}$	$E_{conv.}$	ES (%)	$E_{proposed}$	$E_{conv.}$	ES (%)
exp1	99.5	71.7	-38.7	295.7	156.2	-89.3	620.6	345.9	-79.4
exp2	128.5	111.8	-14.9	330	245.6	-34.3	725.8	525	-38.2
exp4	211.5	208.3	-1.5	424.7	442.8	4.1	914.8	913.2	-0.2
exp6	294.2	304.9	3.5	574.2	652.6	12	1199.7	1346.2	10.9
exp8	377.3	401.6	6	732.9	855.8	14.4	1439.2	1759.7	18.2
exp10	456.1	493.5	7.6	883.8	1051.2	15.9	1734.4	2156.4	19.6
exp12	538.3	589.2	8.6	1042.3	1253.4	16.8	2020.1	2576.1	21.6
exp14	619.4	683.7	9.4	1197.5	1455.9	17.7	2344.4	3012.4	22.2
exp16	699	776.2	9.9	1350.2	1653.4	18.3	2638.8	3408.3	22.6
exp18	789.8	882.4	10.5	1526.5	1879.5	18.8	2968.4	3847.8	22.9
exp20	869.1	975	10.9	1679.2	2079.5	19.2	3286.3	4271.2	23.1

bench applications so that the timing of performing floating-point operations is different for each core, and conducted experiments based on this, and the results are reported in Table VI. The results of this table confirm that Florian's energy saving effect has increased compared to those of the previous Table V because the bottleneck of FPU has been resolved. In the previous result, the gains of the quad and octa core occur from exp10 and exp8, respectively, whereas in this result, they occur from exp4 and exp6, respectively. In other words, in the case of the quad core, it can be seen that Florian has gain unless the floating-point operation is too frequent as the proportion of floating-point operation and other operations is 1:2 or less. Finally, the maximum ES appears when exp20 is performed on the octa core, and the result reaches 23.1%.

IV. CONCLUSION

The open lightweight RISC-V cores are rapidly increasing in use in processors specialized in IoT, wearable, and embedded system applications. As these applications become more diverse and complex, more and more applications require floating-point operations along with the need for multicore processors, which is a major limitation in designing processors using the existing open cores. Because most lightweight RISC-V cores are integer cores without FPU, the design space is greatly narrowed to very few lightweight cores that provide FPU together, making it impossible to design a processor optimized for each application. To tackle this problem, we proposed Florian that includes an external lightweight FPU that can be attached to any RISC-V integer core and a low-power multi-core architecture using the designed FPU. To verify the effectiveness and performance of the proposed Florian, we designed RISC-V processors that implement all the proposed technologies, made prototypes with FPGAs, and finally fabricated a quad core processor chip using 0.11 μ m CMOS technology. We conducted intensive experiments on single, dual, quad, and octa core processors with 11 different testbenches, and confirmed that the proposed Florian achieves energy savings of up to 23.1%.

REFERENCES

[1] K. Han *et al.*, "Tip: A temperature effect inversion-aware ultra-low power system-on-chip platform," in *2019 IEEE/ACM Int'l Symposium on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–6.

[2] C. Heinz, Y. Lavan, J. Hofmann, and A. Koch, "A catalog and in-hardware evaluation of open-source drop-in compatible RISC-V software processors," in *Int'l Conf. on ReConfigurable Computing and FPGAs (ReConFig)*, 2019, pp. 1–8.

[3] H. Jang *et al.*, "Developing a multicore platform utilizing open RISC-V cores," *IEEE Access*, vol. 9, pp. 120 010–120 023, 2021.

[4] N. Bruschi *et al.*, "GVSoC: A highly configurable, fast and accurate full-platform simulator for RISC-V based IoT processors," in *IEEE Int'l Conf. on Computer Design (ICCD)*, 2021, pp. 409–416.

[5] K. Han *et al.*, "Developing TEI-aware ultralow-power SoC platforms for IoT end nodes," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4642–4656, 2021.

[6] J. Park *et al.*, "Developing an ultra-low power RISC-V processor for anomaly detection," in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2023, pp. 1–2.

[7] J. Hormigo and J. Villalba, "Hub floating point for improving fpga implementations of dsp applications," *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 64, no. 3, pp. 319–323, 2017.

[8] W. Lee *et al.*, "K-means clustering-specific lightweight RISC-V processor," in *International SoC Design Conference (ISOC)*, 2021, pp. 391–392.

[9] M. Franceschi, A. Nannarelli, and M. Valle, "Tunable floating-point for artificial neural networks," in *IEEE Int'l Conf. on Electronics, Circuits and Systems (ICECS)*, 2018, pp. 289–292.

[10] Y. Tortorella *et al.*, "RedMulE: A compact FP16 matrix-multiplication accelerator for adaptive deep learning on RISC-V-based ultra-low-power SoCs," in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2022, pp. 1099–1102.

[11] S. Mach, F. Schuiki, F. Zaruba, and L. Benini, "FPnew: An open-source multiformat floating-point unit architecture for energy-proportional transprecision computing," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 774–787, 2021.

[12] Z. Lei, F. Cai, J. Zhou, and Z. Guo, "A floating-point unit architecture based on SweRV EH1 core," in *IEEE Int'l Conf. on Anti-counterfeiting, Security, and Identification (ASID)*, 2022, pp. 1–5.

[13] SiFive, <https://github.com/chipsalliance/rocket-chip>, accessed 19 March. 2022.

[14] PULP, <https://pulp-platform.org/>, accessed 19 March. 2022.

[15] Vectorblox, <https://github.com/riscveval/orca-1>, accessed 19 March. 2022.

[16] EEMBC, <https://www.eembc.org/coremark-pro/>, accessed 19 March. 2022.

[17] A. Pullini *et al.*, "Mr.Wolf: An energy-precision scalable parallel ultra low power SoC for IoT edge processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019.

[18] T. Iwashita, K. Suzuki, and T. Fukaya, "An integer arithmetic-based sparse linear solver using a gmres method and iterative refinement," in *2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, 2020, pp. 1–8.

[19] IEEE, "Iso/iec/ieee int'l standard - floating-point arithmetic," *ISO/IEC 60559:2020(E) IEEE Std 754-2019*, pp. 1–86, 2020.

[20] Xilinx, <https://www.xilinx.com/products/silicon-devices/fpga/kintex-ultrascale-plus.html>, accessed 19 March. 2022.