2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)

Slow and Steady Wins the Race? A Comparison of Ultra-Low-Power RISC-V Cores for Internet-of-Things Applications

Pasquale Davide Schiavone, Francesco Conti, Davide Rossi, Michael Gautschi,
Antonio Pullini, Eric Flamand and Luca Benini

***Abstract*—Achieving a power envelope of few milliwatts combined with tight performance constraints is emerging as one of the key challenges for battery-powered and low cost *Internet-of-things* (IoT) end-nodes. IoT devices have to cope with highly time-varying workloads, characterized by intermittent “race-to-sleep” bursts of compute-intensive operations mingled with long periods of low activity. Architectural heterogeneity provides a possible solution to harmonize these competing constraints; the availability of diverse cores optimized for diverse tasks, but able to run the same code is advantageous for IoT devices. In this paper, we introduce *Zero-riscy* and *Micro-riscy*, two novel RISC-V cores targeting mixed arithmetic/control applications and control-oriented tasks respectively. We compare them with the DSP-enhanced open-source *Riscy* core [1]. *Zero-riscy* is 2.2× smaller than *Riscy* and provides a 2× energy boost for mixed control/arithmetic code with limited DSP. *Micro-riscy* is 1.6× smaller than *Zero-riscy* (∼11.6 kgates in UMC 65nm), has a power envelope of just 100µW at 160MHz and it is 1.4× more energy efficient than *Zero-riscy* on pure control code.**

***Index Terms*—RISC-V, energy efficiency, area optimized, ultra-low-power, open-source, core, microprocessor, internet-of-things.**

I. Introduction

IoT end-nodes couple the requirement for high computational capabilities, extreme energy efficiency and low cost. These devices need to cover a wide range of applications, such as processing, elaboration and transmission of biometrical signals such as ECG, EEG or EMG (e-health), signals coming from low-power imagers or cameras (smart surveillance), vibrations or audio signals in the factories (smart industry), or just low-bandwidth information such as temperature or humidity (smart monitoring). To cover such a large set of use-cases, IoT devices must be programmable and tend to look more and more like complete systems-on-node including sensors, microprocessors, specialized hardware, memories, and wireless transceivers, capable of operating autonomously for several years [2].

To this end, IoT end-nodes usually operate with a highly time-varying behavior: for a majority of the time, they stay in sleep mode but they must be sensitive to external events and wake up. When this happens, they typically have to execute very heterogeneous tasks such as managing the interfaces to acquire data from environmental sensors, storing it into volatile or non-volatile memories, and transmitting it via radio; often, they also have to perform relatively heavy digital signal processing to perform semantic extraction of relevant information from sensor data [3].

Due to the breadth of computational requirements found in the applications workload, different kinds of cores might be better suited for different parts of the workload: from very small, ultra-low-power cores with limited performance, but very low area and power cost; to cores enhanced with *Digital Signal Processing* (DSP) capabilities targeting near-sensor data processing. For this reason, several core vendors provide a wide range of different architectures to cover the large variety of energy, performance, power and area constraints. For example ARM provides cores for low power and low area budgets, as well as superscalar cores for high performance.

To cope with different kinds of application workloads, heterogeneous multi-core architectures have recently been proposed in commercial systems for IoT both from the industry and the research communities [4], [5]. Another emerging trend is parallel near-threshold computing [6]. Significant energy efficiency can be achieved by leveraging tightly coupled clusters of DSP processors operating at low voltage, improving energy efficiency thanks to the quadratic dependency of dynamic power with supply voltage, while recovering performance degradation due to low-voltage operations by using multiple processor cores as exploited in [7]. In the context of parallel ultra-low-power computing platforms, significant benefits can be achieved by selectively activating the computational resources of the system, leveraging a single processor during data acquisition, storage or transmission, and activating the multi-core clusters during heavy signal processing phases as proposed in [8]. In addition to pure-control code, a single processor can be leveraged to compute some pre-processing tasks that wake up the accelerator only when something relevant has been detected.

In this paper we explore architectural heterogeneity at the core level for the fast-growing open-source RISC-V *Instruction-Set Architecture* (ISA) [9]. We describe micro-architectural design choices and analyze cost, performance, power and energy efficiency. Two area-optimized cores, *Zero-riscy* and *Micro-riscy*, are presented and compared with the one proposed by Gautschi et al. in [1] (*Riscy*). While *Riscy* has been optimized to target DSP and to be integrated in a cluster of processors, *Zero-riscy* and *Micro-riscy* have been optimized to target arithmetic-control mixed and pure-control applications respectively [1]. We show that thanks to an area-aware design with small data-path, *Zero-riscy* is more than 2× smaller than *Riscy*, consumes 2× less energy and it has only only 1.3× longer execution time in *Coremark*. *Micro-riscy* is an even smaller core and it consumes 3.5× less energy than *Riscy* when executing pure control-oriented code. In the remainder of this paper we first review related work, then we describe the micro-

[1] Both the *Zero-riscy* and *Micro-riscy* cores will be released open-source in summer 2017.

978-1-5090-6462-5/17/$31.00 ©2017 IEEE

architectural design for *Zero-* and *Micro-riscy*. We report an in-depth-comparative analysis of the three cores in terms of area, power, performance and energy efficiency, with a wide range of synthesis constraints and workloads. Finally, we analyze their energy consumption in an always-on context, where the cores are waiting for an event to start the computation and finally go back to sleep. We show that when the interval time between events is long enough, the contribution of the leakage power becomes crucial, hence small cores overwhelm the fast cores in energy efficiency.

II. Related Work

Uses of a core in a deeply embedded systems such as IoT end-nodes range from simple control and interaction with peripherals, to the execution of light-weight processing, all the way up to the computation of complex signal processing algorithms. ARM provides a set of micro-controller-level cores in its Cortex-M series; for example, the Cortex-M0 [10], which is meant for running applications that need minimal power and area or the Cortex-M4 [11], which provides an instruction set extension to target signal processing algorithms, and many more. The ARM Cortex-M0 is a single-issue in-order core with three pipeline stages and optional single-cycle 32b multiplier unit, whereas the ARM Cortex-M4 is a single-issue in-order core with three pipeline stages and a branch speculation engine, single-cycle 32b *Multiply-and-Accumulate* (MAC) unit, 8/16b *Single-Instruction Multiple-Data* (SIMD) instructions and an optional floating-point unit. Synopsys provides the DesignWare ARC configurable processors [12], based on the ARCv2 ISA to target power- and area-constrained embedded systems, as well as the ARC EM DSPs for ultra low-power embedded applications requiring advanced signal processing capabilities. The latter support an extended ARCv2DSP ISA and provide MAC, fixed-point and SIMD instructions. Recently, Minima Processor proposed a RVC32IMA core that consumes 16μW at 0.5V, 15MHz [13].

Among open-source and academic cores targeted at the Internet-of-Things, those based on the RISC-V ISA have recently enjoyed increasing levels of attention and success. While some RISC-V cores are targeted at high-performance platforms beyond our scope (e.g. Boom [14], Rocket [15]), many others are oriented at IoT applications. An example is Z-Scale, a small 32b three stages single-issue in-order core, designed by the original authors of the RISC-V ISA [16]. Another example is PULPino [17], which is a RISC-V micro-controller that integrates the *Riscy* core [1]. PicoRV32 is a small configurable RISC-V core with high frequency but also high cycle-per-instructions and it can be found open-source in [18]. Duran et al. presented the mRISC, a RISC-V RV32IM core that consumes in 130nm ∼97 μW/MHz and has an area and maximum frequency equal to 0.12 mm^2 and 100MHz respectively [19]. The open-source and typically parametric design of these cores allows them to be adapted to the varying scenarios described above.

Systems where a single core is used for both control and signal processing operations are often found: Benatti et al. [20] propose an EMG gesture recognition application based on an ARM Cortex-M4 micro-controller, which is used to extract salient features out of the EMG signals and classify them with a support-vector-machine (SVM); Imtiaz et al. [21] propose a system to detect epileptic seizures using an ultra-low-power Texas Instrument (TI) MSP430 micro-controller [22] as the main computation device. Another common scenario is that of a simple central core coordinating and controlling several peripherals in a more complex system, where other devices are used as the main computation engines. Konijnenburg et al. [23] propose a multi-sensor acquisition system where the ARM Cortex-M0 is used to interact with a sensor readout chip and a system of hardware accelerators; another example is found in Ickes et al. [24], where a 16b core controls memory-mapped accelerators for FIR and FFT.

To cope with the diversity in tasks needed for the IoT, *heterogeneous* systems, which include several different types of cores optimized for different tasks, have been recently introduced to the market. For example, the TI CC2650 [4] has one ARM Cortex-M3 to control the system and execute applications and one dedicated ARM Cortex-M0 to control only the wireless transmission. Another example of heterogeneous system is the NXP LPC54100 platform [5], with a tiny ARM Cortex-M0 processor managing control tasks such as acquisition, storage and transmission of data, while a more powerful Cortex-M4 is responsible for the signal processing tasks.

For more data-intensive algorithms, which require higher performance or energy efficiency, multi-core architectures or vector machines have also been proposed to accelerate computational-intensive tasks. Benatti et al. [25] use a multi-core cluster to extract complex features from EEG data to detect seizures. Keller et al. [26] propose a heterogeneous RISC-V vector-processor platform where the Rocket core is used as a controller for the vector-machine and Z-Scale is used as a power management unit. *Fulmine* [27] is a heterogeneous multi-core platform equipped with 4 DSP-enhanced OpenRISC cores and two HW accelerators for convolution and cryptography algorithms is proposed. The 4 cores can execute general-purpose applications and are optimized for signal-processing algorithms, whereas for convolution or cryptography kernels, the energy consumption can be reduced using the dedicated HW accelerators. In GAP-8 [28], a single-core is responsible for handling the peripherals and compute light-processing, whereas a cluster of 8 enhanced DSP RISC-V cores is leveraged to accelerate complex computations.

Many systems require several of the tasks described above. Heterogeneous and asymmetric cores can be integrated in a multi-core system to handle different tasks according to their features. For control-code, small and ultra-low-power cores can be built using 8b and 16b architectures [29], whereas high-performance cores benefit more from 32b and 64b architectures. However, in heterogeneous systems sharing or partially sharing the ISA brings benefit in terms of bus interconnections and interfaces as well as flexibility in the SW tool-chain that allows different cores with the same or partially the same ISA to share the infrastructure.

In this paper we present two new cores: *Zero-riscy* optimized for arithmetic-control mixed applications and *Micro-riscy* optimized for control-code tasks. *Micro-riscy* is optimized to have a very small area and power; it is meant to work in an always-on voltage domain, interacting with peripherals and controlling the activity of bigger accelerators off-loading tasks and implementing power-management policies. *Zero-riscy* is optimized to have a small area and power, but higher computation performance than *Micro-riscy*. It is meant to work as the main actor in a complex system, controlling peripherals and accelerators but also performing general-purpose computations. They implement the RISC-V ISA [2] supporting compressed instructions. Together with the *Riscy* core presented in [1], the three cores will provide an open-source family of heterogeneous cores ready to be used in different contexts. Furthermore, to target high energy-efficiency and ultra-low power in battery-powered IoT devices, the cores are evaluated in Near-Threshold (NT), where the transistors achieve their maximum energy efficiency [30]. As the targeted domain of the proposed cores is close to the one of the Cortex-M cores, Table I shows an area comparison between the proposed RISC-V cores and their corresponding Cortex-M family implementations.

TABLE I
COMPARISON BETWEEN THE PROPOSED RISC-V CORES AND ARM CORTEX-M FAMILY IMPLEMENTATIONS.

Core	*Area [KGE]*	*Core*	*Area [KGE]*
Riscy	40.7	Cortex-M4/Cortex-M3	53.0/37.9
Zero-riscy/Micro-riscy	18.9/11.6	Cortex-M0/M0+	13.3/12.5

Cortex-M0 area from a synthesis in UMC 65nm with 32 cycles mult, 16 interrupts, no wake-up and debug controllers. Cortex-M0+, M3, and M4 area estimated from publicly available information [31].

III. RISC-V MICROARCHITECTURES

In this section, the RISC-V architectures proposed in this work are described.

A. Riscy

Riscy is an open-source 32b RISC-V core, in-order with four pipeline stages [1]. It implements the full RVC32IM ISA and has been extended to enhance performance, reduce the code size and increase the energy efficiency of signal processing algorithms. *General Purpose* extensions to *Riscy* include hardware loops to handle automatically up to two nested finite-loops, auto-increment load/store instructions which fuse the memory operation with the address-update and bit-manipulation instructions to handle bit fields in registers. *DSP*-oriented extensions include MAC instructions, fixed-point addition, subtraction and multiplications instructions, Packed-SIMD instructions as addition, subtraction, comparison and manipulation of vectors of four 8b elements or two 16b elements and finally a dot-product-unit which compute the dot-product multiplication between two vectors. The core has been designed and optimized to work in a multi-core cluster, where the cores share the L1 scratch-pad memory and the instruction cache as for example the *Parallel Ultra-Low Power Platform* (PULP) [32] to provide high performance in power-constraints embedded-systems.

B. Zero-riscy

Zero-riscy is an area-optimized RISC-V core implementing the RVC32IM instruction set architecture; Figure 1 shows a simplified version of its micro-architecture. It has two pipeline stages that are the *Instruction Fetch* (IF) and *Instruction Decode and Execute* (IDE) stage. Similarly to *Riscy*, the IF stage interacts with the instruction memory subsystem. It contains a prefetch-buffer that collects data from the instruction memory and it handles compressed instructions. In addition, it generates the instruction address and the program counter value (which can be different due to misalignment given by compressed instructions), and it contains a FIFO to store instructions also when the next stage is not ready to process them. The IDE stage decodes the instructions, reads the operands from the register file, prepares the operands for the *Arithmetic Logic Unit* (ALU) and the multiplier unit and executes the instructions. The register file is a 2-read-1-write latch-based register file. The ALU contains the minimal hardware resources to implement the RVC32IM ISA: one 32b adder, one 32b shifter and the logic unit. The 32b adder is used to compute additions, subtractions and comparisons and it is shared with the data address generation unit, the branch engine and the divider. Branch, multiplication, division, load and store instructions are computed iteratively, stalling the next instruction until they have finished.

Figure 2 shows the simplified ALU architecture. The top multiplexer selects the addition operands, whereas the shifter and the logic unit always compute their function on operands coming from the decoder or register-file. Branch instructions, which need to compute both the comparison and the branch target, use the adder in two different cycles: the first cycle is used to compute if the branch is taken or not. In case the branch is taken, in the next cycle the adder is used to add to the program counter the offset, so the target address. Load and store instructions are executed in two cycles: the first cycle is used to calculate the address and the second cycle to receive the data from memory. The multiplier unit contains one MAC unit which is able to sequentially multiply two 16b operands and accumulate the result in a 32b register. Divisions are implemented with minimum resources with the unsigned serial division algorithm using the adder in the ALU in all the steps. No forwards-path and data-dependencies logic is present in the core, which allows to save control-logic, multiplexers and comparators. *Zero-riscy* implements a minimum set of control-status registers defined by the privileged RISC-V 1.9 spec and it supports debug and up to 32 interrupt requests.

C. Micro-riscy

Micro-riscy is further optimized for area with respect to *Zero-riscy* by removing the RVM RISC-V extensions. *Micro-riscy* does not have any hardware (HW) support for multiplications and divisions. To further reduce the area footprint, it implements the RVE RISC-V specification which allows to use only 16 general-purpose registers.

IV. EXPERIMENTAL RESULTS

In this section, the area, performance, power and energy consumption of the three cores at different operating points running different workloads is analyzed. Each core has been synthesized, placed and routed targeting the UMC 65nm technology. How the energy consumption changes with respect to different workloads and operating frequencies and voltages is also discussed in this section.

A. Area

Figure 3 shows the area distribution of the three cores in terms of kgates-equivalent. The *Riscy* area is 40.7 *KGE*[3],

[2] *Zero-riscy* implements the RVC32IM ISA and *Micro-riscy* implements the RVC32E subset ISA

[3] Equivalent minimum-size NAND2 gate area. In UMC 65nm, one gate equivalent (GE) is 1.44 μm^2.

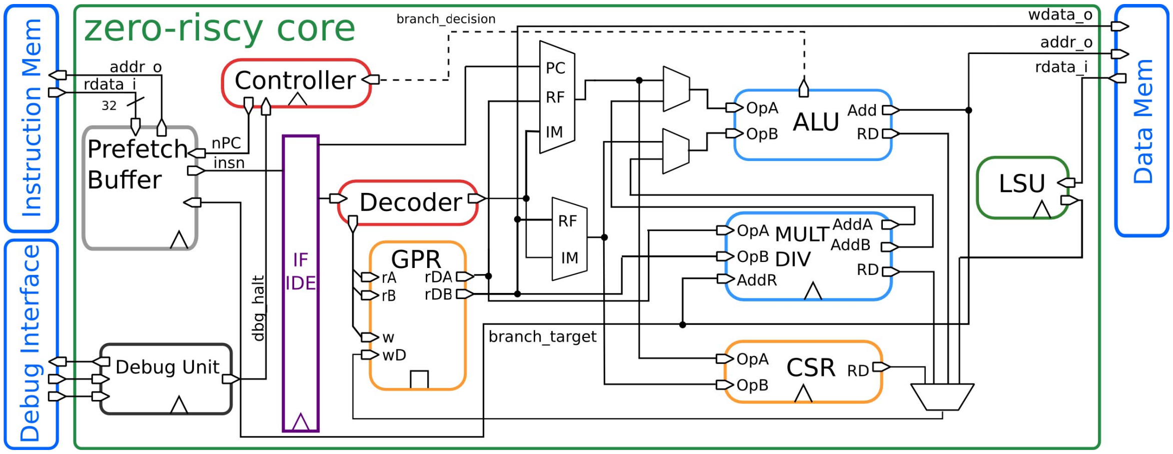

Fig. 1. Simplified block diagram of *Zero-riscy*. The critical-path is along the branch-address from the IF-IDE pipeline stage to the ALU, all the way along to the branch-target in the prefetch-buffer and finally to the instruction memory. It is about ∼280 FO4 NAND2 gate in UMC 65nm.

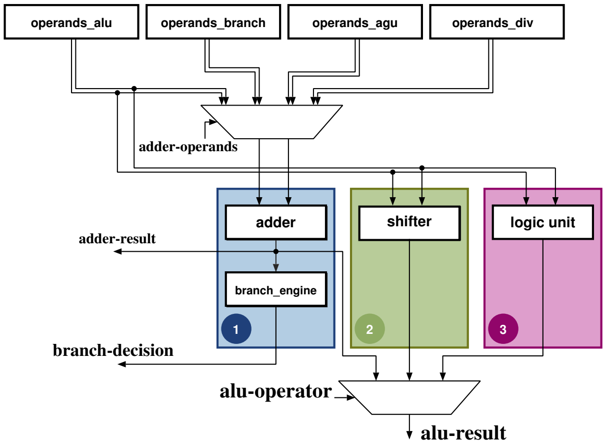

Fig. 2. Simplified block diagram of the *Zero-riscy* ALU. The adder is shared between additions/subtractions, branches, address generation for load/store operations and divisions.

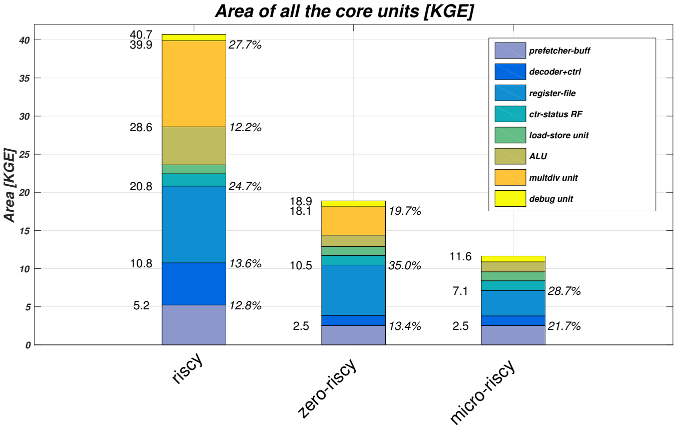

Fig. 3. Area distribution of the different cores. Values are taken from the post-synthesis slow-netlist instances.

whereas the *Zero-riscy* and *Micro-riscy* areas are 18.9 and 11.6 *KGE* respectively when all the cores are synthesized at relaxed timing-constraints. The smaller and less complex prefetch-buffer saves 2.7 *KGE* overall in both optimized cores. Thanks to the non-extended ISA and less complex pipelines, the decoder and controller are also much smaller (∼7*KGE*). In addition, the ALU has been optimized to use as few resources as possible, preferring time-multiplexing for instructions which need the same data-path concurrently. The *Zero-riscy* multiplier has been also optimized to use multi-cycle implementations of the product instructions and it does not contain any dot-product unit nor fixed-point support. *Riscy* has a costly 3-read-2-write ports latch-based register-file (10 *KGE*), almost as much as the entire *Micro-riscy* core. By substituting it with a 2-read-1-write register-file, *Zero-riscy* saves 3.4 *KGE*; furthermore, *Micro-riscy* reduces its size to only 16 entries, down to only 3.3 *KGE* of footprint. Finally, the simpler special-purpose registers reduces the area of the control-status register-file from 1.5 *KGE* to 1.2 *KGE*. The major contribution to the *Riscy* area footprint is the multiplier and division unit (27.7%), due to the complex dot-product unit and fixed-point multipliers. In *Zero-riscy*, the core area is dominated by the register-file (35%), as the multiplier unit implements only the RVM specifications in a multi-cycle fashion. Even if the number of entries has been reduced to 16, in *Micro-riscy* the top contribution to the total area is again the register-file (28.7%).

B. Workloads

We chose three micro-benchmarks selected from a large set as representative of three different classes of workloads.

The *2D-Convolution* is a common kernel in image and signal processing applications. This kernel has been chosen to benchmark signal processing capabilities of the three cores. We chose to implement a 5x5 filter and consider a 32x32 input image. Both the filter and the input and output images are 16b fixed-point. Furthermore, before storing the result, the output is saturated between -1 and 1. In case of *Riscy*, the code has been optimized to use Packed-SIMD, dot-product, fixed-point and vector manipulation instructions.

The second micro-benchmark is Coremark, a system-independent benchmark from the Embedded Microprocessor Benchmark Consortium (EMBC) used to test embedded system cores [33]. It contains both integer arithmetic and

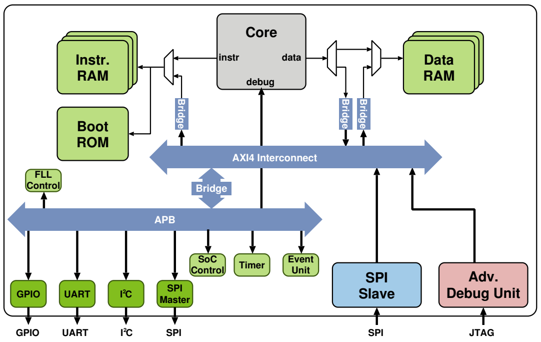

Fig. 4. PULPino architecture overview. Software and RTL are open-source and available at www.pulp-platform.org

control code operations. This kernel has been chosen to show arithmetic-control mixed processing capabilities of the cores.

Finally, the *Runtime* workload implements a simple set of embedded-system Runtime functions, as e.g. a non-preemptive task-scheduler and a driver to interact with peripherals. The functions are taken from the PULP Runtime.

C. Performance

Experiments have been conducted by integrating the three cores within three separate instances of the open-source PULPino micro-controller platform [17]. The micro-controller has one core, one data and one instruction memory, an event-unit for handling interrupts and common peripherals as SPI, UART, I2C, GPIO and timers. In Figure 4 we show an overview of the PULPino architecture.

The three PULPino instances have been fully synthesized, placed and routed at the 1.08V worst-case corner using Synopsys Design Compiler v2015.06 (DC) for the synthesis and Cadence Innovus v15.20 (INNOVUS) for the place and route phase. For each core, two different netlists have been generated to show the timing constraints effects on power consumption, as discussed in the next Section IV-D. The first netlist has been constrained with a relaxed clock period of 10 ns, whereas the second netlist has been synthesized with a tighter clock period of 3 ns. We refer to the "relaxed" design as slow-netlist and to the "tighter" design as fast-netlist. In typical conditions of the targeted technology at 1.2 V, the slow-netlist can reach 185 MHz and the fast-netlist 560 MHz, whereas at 0.8 V, the slow-netlist can reach 55 MHz and the fast-netlist 160 MHz. Table II shows the micro-benchmark [4] execution cycles, the *Instructions-Per-Cycle* (IPC) and code size. Execution cycles and code size have been normalized with respect to *Riscy*. In this experiment all the cores run at the same frequency and can reach almost the same maximum frequency. As expected, *Riscy* is the fastest core in all the kernels thanks to its enhanced ISA and more complex pipeline, whereas *Micro-riscy* is the slowest. When running DSP applications, *Riscy* is 6.1× faster than *Zero-riscy* and 53.4× faster than *Micro-riscy*. Such difference in performance is given by the dot-product and Packed-SIMD extensions. The code size for the DSP kernel changes less than 10%, whereas for *Coremark*, the code size increases by 10% and 30% for *Zero-riscy* and *Micro-riscy* respectively. As expected, in the control-code micro-benchmark case, the code size does not change significantly. *Zero-riscy* is 8.8× faster than *Micro-riscy*. The two cores have the same pipeline architecture but *Zero-riscy* implements the HW multiplication instruction, which is heavily used in the *2D-Convolution* kernel. *Riscy* provides the best performance also on arithmetic-control mixed code (3.19 CoreMark/MHz), but the benefits with respect to *Zero-riscy* (2.44 CoreMark/MHz) are much less significant than the signal processing algorithms. *Coremark* does not contain much DSP code, therefore the difference in performance does not come from the DSP extensions of *Riscy*, but mainly from its deeper pipeline that enables single-cycle data memory access, and from the single-cycle multiplication-accumulation and general-purpose instruction extensions as e.g. bit-manipulation. *Zero-riscy* targets arithmetic-control mixed code: its area is small, but it still has the hardware resources necessary to support multiplications and divisions, which are required for signal processing algorithms. Similarly to what happens in the DSP kernel, *Micro-riscy* is the slowest core (0.91 CoreMark/MHz) especially due to the lack of multiplication instructions, which are emulated in software. The reduced number of registers is only associated to a 3% performance drop. In the Runtime kernel, the three cores show negligible differences in performance: the code implementing these routines do not benefit from multiplications nor from DSP extensions.

In terms of performance and targeted domain, the *Zero-* and *Micro-riscy* are comparable with the ARM Cortex-M0/M0+ (2.33/2.46 Coremark/MHz), whereas *Riscy* is comparable with Cortex-M4 (3.40 Coremark/MHz [1]). The IPC of *Riscy* is the highest for all the micro-benchmarks. As the PULPino data and instruction memories have 1 cycle access, the IPC is 0.8 due to misaligned memory accesses (that require two cycles), data-hazard after load operations and branches/jumps. Furthermore, in the control-oriented kernel the control-status write operations require the core to flush the pipeline. For *Zero-riscy*, the IPC in the DSP micro-benchmark is only 0.5 as the most common instructions are multi-cycle operations as load, store and multiplications. In addition, the multi-cycle operations penalize to 0.7 the IPC also in the other kernels. Finally, *Micro-riscy* has IPC 0.7 in all the kernels.

TABLE II
NUMBER OF CYCLES, IPC AND CODE SIZE FOR EACH KERNEL.

KERNEL	*Riscy*			*Zero-riscy*			*Micro-riscy*		
	Cycles	IPC	Cod. Size	Cycles	IPC	Cod. Size	Cycles	IPC	Cod. Size
2D-Convolution	1.0 (43.1K)	0.82	1.0 (1080B)	6.1	0.52	1.0	53.4	0.66	1.0
Coremark	1.0 (313.5K)	0.79	1.0 (15.3KB)	1.3	0.67	1.1	3.5	0.67	1.3
Runtime	1.0 (37)	0.76	1.0 (232B)	1.0	0.68	1.1	1.0 (36)	0.67	1.1

D. Power estimation

The switching activity for each one of the versions of the platform described in Section IV-C has been extracted by means of simulation on post-layout netlists. The switching activity has been then analyzed using Synopsys PrimeTime v2012.12 (PT) at 1.2 and 0.8 V in typical condition.Table III shows the dynamic power density and the leakage power estimated.

The fast-netlists have higher leakage power and dynamic power density, since it has been synthesized, placed and routed to target higher frequencies. On average, the leakage power of the fast-netlists is 14× the leakage power of the slow-netlists,

[4] The three benchmark kernels have been compiled with GCC5.2 whose back-end has been modified to instantiate the extended instructions for *Riscy* when appropriate.

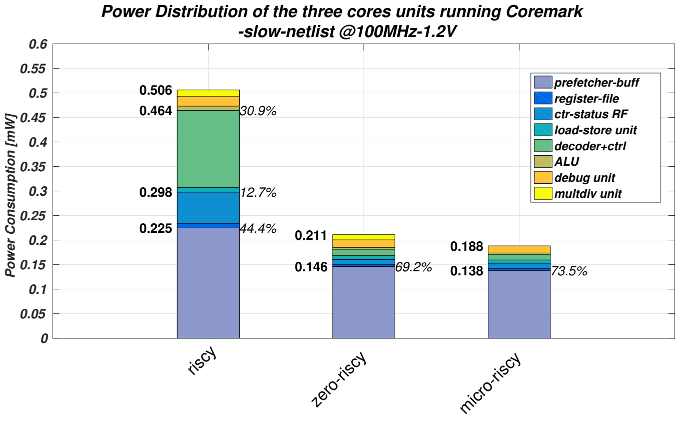

Fig. 5. Power distribution of the different cores units measured from post-layout simulations at 1.2V, 100MHz.

TABLE III
CORE POWER ESTIMATIONS.

Core	PDynD- PLkg @1.2V	PDynD - PLkg @0.8V
Netlist with relaxed timing constraints		
Riscy	5.07 - 1.91	1.40 - 0.22
Zero-riscy	2.08 - 0.73	0.59 - 0.07
Micro-riscy	1.88 - 0.45	0.52 - 0.04
Netlist with tight timing constraints		
Riscy	6.70 - 24.90	2.10 - 2.96
Zero-riscy	2.30 - 11.0	0.68 - 1.24
Micro-riscy	2.03 - 6.25	0.60 - 0.70

Dynamic power density is reported in μW/MHz and leakage power in μW.

while the dynamic power density increases by 1.2× at $1.2\,\mathrm{V}$. At $0.8\,\mathrm{V}$, the leakage power of the fast-netlists increases by 16×, whereas the dynamic power density by 1.3×.

Table IV shows the total power consumption of all the four instances at the maximum frequency achievable at $1.2\,\mathrm{V}$ and $0.8\,\mathrm{V}$ and the ratio between the power consumption at the two voltages. As is visible in the plot, the power at $0.8\,\mathrm{V}$ is up to 12.2× smaller than in $1.2\,\mathrm{V}$, whereas the speed decreases only up to ∼3.5.

Figure 5 shows how the power is spent in the various components of each core. The main contribution to the whole power consumption is provided by the prefetch-buffer, which interacts with the instruction memory every cycle, produces the instruction fetch address and the program-counter value and stores the instructions in a FIFO. In the *Riscy* core it also handles hardware-loop instruction requests. As the ISA and the pipeline of the two smaller cores have been simplified, the decoder and the controller power consumption has been also reduced significantly.

TABLE IV
CORE TOTAL POWER ESTIMATIONS AT MAXIMUM SPEED.

Core	Ptot @1.2V	Ptot @0.8V	Power Ratio
Netlist with relaxed timing constraints			
	@185 MHz	@55 MHz	
Riscy	940	77	12.2
Zero-riscy	386	33	11.7
Micro-riscy	348	29	12.0
Netlist with tight timing constraints			
	@560 MHz	@160 MHz	
Riscy	3777	339	11.1
Zero-riscy	1299	110	11.8
Micro-riscy	1143	97	11.8

Power is reported in μW.

E. Energy calculation

In this section, the energy is calculated for each combination of core and kernel at different operating frequencies and voltages. In Figure 6, the normalized energy consumption for each core synthesized at the two different clock targets at $1.2\,\mathrm{V}$ is shown. Figure 6a and Figure 6c show the energy consumption of all the slow-netlist instances at two different frequencies, whereas in the right side, Figure 6b and Figure 6d show the energy consumption of all the fast-netlist instances. The energy consumption of each kernel has been normalized with respect to the worst-case using the slow-netlist at $100\,\mathrm{kHz}$. For example, the *2D-Convolution* has been normalized with respect to the energy consumed by *Micro-riscy* as it this the highest. Figures 6a and 6b show the effect of the tighter timing-constraints at a very slow frequency; this underlines the effect of leakage.

As reported in Table III, the fast-netlist has 1.2× the dynamic power density and 14× the leakage power of the slow-netlist. At the same operating frequency, the execution time of the two netlists is identical, whereas the total energy consumption of the fast-netlist increases by ∼10.7×. 97% of this energy consumption is due to leakage in the fast-netlist, whereas it is 76% in the slow-netlist. In Figure 6c and Figure 6d, the slow and the fast netlists are compared at their maximum frequency in typical conditions ($185\,\mathrm{MHz}$ and $560\,\mathrm{MHz}$ respectively). The energy consumption of the fast-netlist is only 1.2× the energy of the slow-netlist due to the combination of the reduced execution time (∼3×) and the higher power consumption (∼3.6×). Leakage energy is negligible in both the netlists.

For DSP applications, the core equipped with DSP instruction extensions consumes the minimum energy with respect to the other two cores in all the operating conditions. The ratio between the execution time of *Riscy* and *Zero-riscy* (6.1 at the same frequency) is smaller than the ratio of powers. For example, in the fast-netlist at $560\,\mathrm{MHz}$, the *Riscy* total power consumption is 2.9× the power of *Zero-riscy*, whereas in the slow-netlist at $185\,\mathrm{MHz}$ it is 2.4×. *Zero-riscy* and *Micro-riscy* consume on average 2.4× and 15.9× the energy of *Riscy*, respectively. For arithmetic-control mixed applications as *Coremark*, *Zero-riscy* is the one which consumes the least energy. The execution time of *Zero-riscy* is 1.3× longer than *Riscy* but it consumes much less power. On the other hand, it is faster than *Micro-riscy* (2.7×) but it consumes more power. *Riscy* and *Micro-riscy* consume on average 1.9× and 2× the energy of *Zero-riscy* respectively. Finally, *Micro-riscy* is the core that consumes the least energy for control-code, as the extra power of the other cores is not compensated by any benefit in performance. Hence, *Zero-riscy* and *Riscy* consume on average 1.4× and 3.5× the energy of *Micro-riscy* respectively. The same experiment has been repeated at $0.8\,\mathrm{V}$ using only the fast-netlist, whose maximum frequency is $160\,\mathrm{MHz}$. At $0.8\,\mathrm{V}$, the cores consume on average 0.3× the energy consumed in the super-threshold region. Figure 7 shows the energy consumption of the cores for each kernel in the

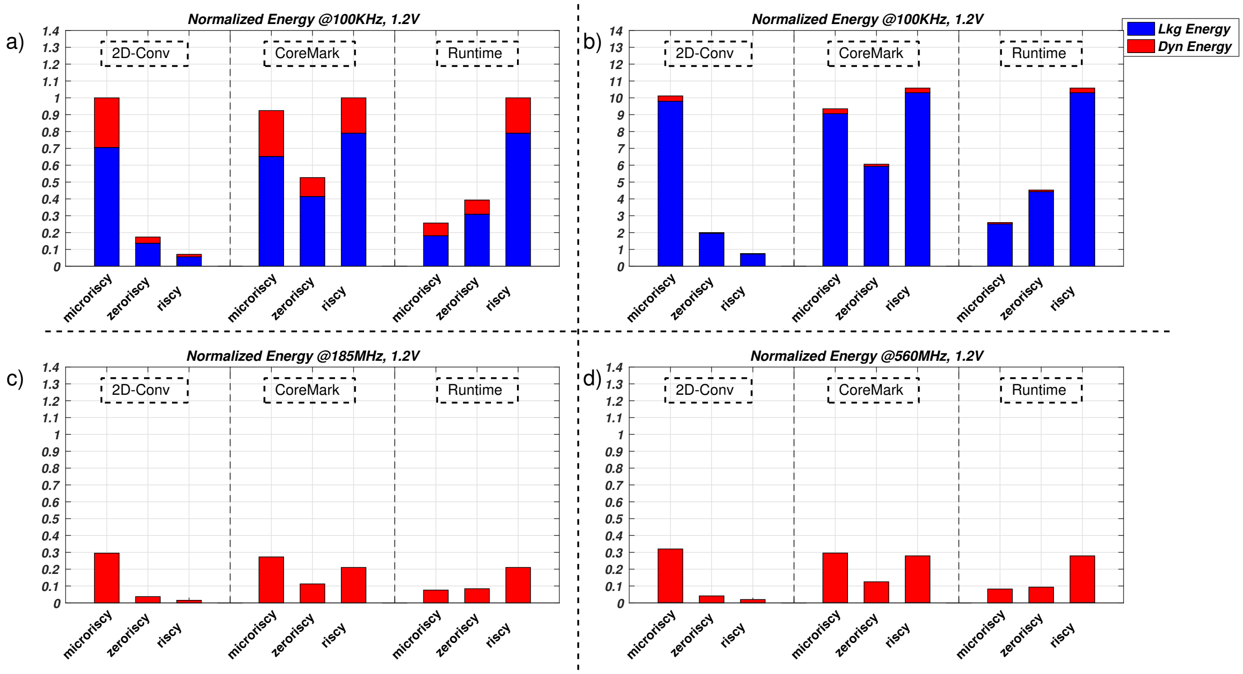

Fig. 6. Energy consumed by the three cores while running different workloads. In blue the leakage energy, in red the dynamic. In the left column, the slow-netlist results are shown (a and c), whereas the fast-netlist is shown in the right column (b and d).

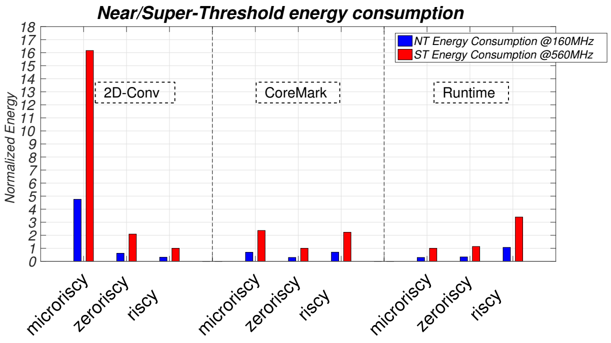

Fig. 7. Energy consumed by the threecores while running different workloads at two different voltages. In blue the energy at 0.8 V, in red at 1.2 V.

super-threshold region 1.2 V, 560 MHz and at near-threshold 0.8 V 160 MHz. For each kernel, the energy consumption has been normalized by the most energy-efficient core for that application. It is possible to notice that *Riscy* is still the most energy efficient core for DSP applications, *Zero-riscy* for arithmetic-control mixed applications and finally *Micro-riscy* for control-oriented code at at NT.

F. Always-On activity

To evaluate the effect of duty-cycled activities in always-on conditions, the final experiment takes into account not only the time frame where the core is active, but also when the core is in idle-mode waiting for an event. In many cases, this happens in an always-on domain where the core cannot be completely switched-off with power gating, thus they always consume leakage power. The longer the time between two events, the bigger the leakage contribution on the total amount of energy consumed.

Figure 8 shows the energy consumption of the three slow-netlist instances at 55 MHz when executing the *2D-Convolution* at 0.8 V, 55 MHz in the y-axis and the time between two events that trigger the computation in x-axis. The three curves show the energy consumption starting from the end of the execution of the benchmark. The energy consumption is normalized with respect to the energy consumed by *Riscy*. The inter-event time is normalized with respect to the execution time of *Riscy* (43.1 Kcycles/55 MHz = 784 µs. As previously discussed, when only the active period is taken into account, *Riscy* is the fastest and the most efficient core for signal-processing applications. However, when the idle period starts being longer, the leakage contribution overwhelms the active energy, thus smaller cores are less penalized. In the second region (from 649 ms to 31 s) *Zero-riscy* consumes less energy than the others, hence it is the most energy efficient core. Such inter-event time is for instance in the EEG processing range, where the processing tasks wait 1 or 2 s before they can start processing the acquired signals [21], [25]. *Micro-riscy* becomes the most energy efficient core when idle periods become extremely long. This analysis shows that leakage-optimized cores are well suited also for sporadic data-intensive applications, if the throughput requirement is sufficiently low.

V. Conclusion

To conclude, in this paper an analysis of three different cores optimized for three different tasks has been carried out to show how the energy consumption changes among core micro-architecture, workload, timing-constraints, operating frequency and voltage. We detail the features of the core that minimize the energy consumption given a certain workload; these results can be used to choose a core implementation based on the set of tasks to be executed. The core with DSP instructions is the most energy efficient in data-intensive kernels but *Zero-riscy* is the most efficient in arithmetic-control mixed and *Micro-riscy* in pure control code thanks to the minimal resources

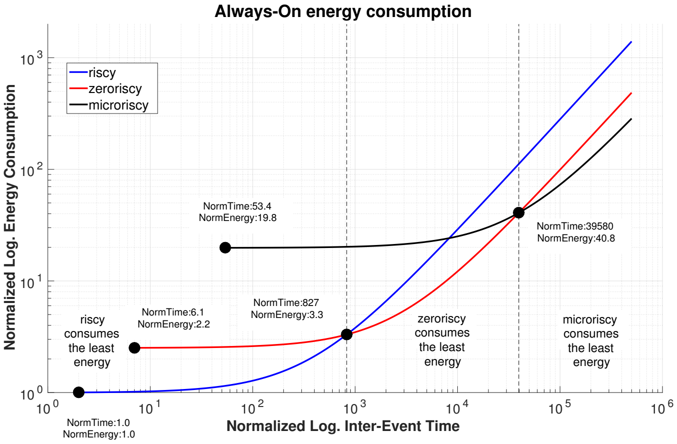

Fig. 8. Always-On normalized energy consumption at 55 MHz 0.8 V: the core is active for a portion of time and sleeps for the rest of the time

implemented in for the targeted ISA. We also showed that tighter timing constrained netlists can achieve high frequency for a limited energy overhead (∼20%), but they are energy-inefficient at low frequency, due to the huge contribution of leakage to the total consumption (>10×). The same results can be obtained both at super-threshold and at near-threshold operating points, where the cores consume on average 3× less energy. Finally, we have shown that for systems that operate in an always-on power domain and execute a task as a consequence of a rare event, the leakage power is the most important contribution to the energy consumption, hence area optimization is crucial.

Acknowledgment

This work has been funded by the Swiss National Science Foundation under grant 162524 (MicroLearn: Micropower Deep Learning). The authors would also thank G. Haugou for the SW and tools support.

References

[1] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gurkaynak, 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*, pp. 1–14, 2017.

[2] M. Alioto, *Enabling the Internet of Things: From Integrated Circuits to Integrated Systems*. Springer International Publishing, 2017.

[3] D. Rossi, I. Loi, A. Pullini, and L. Benini, “Ultra-low-power digital architectures for the internet of things,” in *Enabling the Internet of Things*. Springer International Publishing, 2017, pp. 69–93.

[4] Texas Instruments, “CC2650 SimpleLink^{TM} Multistandard Wireless MCU,” Tech. Rep.

[5] NXP, “LPC5410x Datasheet,” Tech. Rep.

[6] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, “Low-power cmos digital design,” *IEICE Transactions on Electronics*, vol. 75, no. 4, pp. 371–382, 1992.

[7] D. Rossi, A. Pullini, M. Gautschi, I. Loi, F. K. Gurkaynak, P. Flatresse, and L. Benini, “A -1.8v to 0.9v body bias, 60 GOPS/w 4-core cluster in low-power 28nm UTBB FD-SOI technology,” in *2015 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. IEEE, oct 2015.

[8] F. Conti, D. Palossi, R. Andri, M. Magno, and L. Benini, “Accelerated visual context classification on a low-power smartwatch,” *IEEE Transactions on Human-Machine Systems*, pp. 1–12, 2016.

[9] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovi, “The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA,” 2011.

[10] ARM, “ARM Cortex M-0 Technical Reference Manual,” Tech. Rep.

[11] ——, “ARM Cortex M-4 Technical Reference Manual,” Tech. Rep.

[12] Synopsys, *DesignWare ARC Processors*. [Online]. Available: https://www.synopsys.com/designware-ip/processor-solutions/arc-processors.html

[13] L. Koskinen, “28 Microwatt RV32IMAC,” https://riscv.org/wp-content/uploads/2017/05/Wed1615-28-Microwatt-RV32IMAC-Koskinen.pdf.

[14] C. Celio, K. Asanovic, and D. Patterson, “The Berkeley Out-of-Order Machine (BOOM!): An Open-source Industry-Competitive, Synthesizable, Parameterized RISC-V Processor,” https://riscv.org/wp-content/uploads/2016/01/Wed1345-RISCV-Workshop-3-BOOM.pdf.

[15] Y. Lee, “RISC-V ”Rocket Chip” SoC Generator in Chisel,” https://riscv.org/wp-content/uploads/2015/02/riscv-rocket-chip-generator-tutorial-hpca2015.pdf.

[16] Y. Lee, A. Ou, and A. Magyar, “Z-scale: Tiny 32-bit RISC-V Systems,” https://riscv.org/wp-content/uploads/2015/06/riscv-zscale-workshop-june2015.pdf.

[17] A. Traber and M. Gautschi, “Pulpino: Datasheet,” ETH Zurich, Tech. Rep., 2016. [Online]. Available: http://www.pulp-platform.org/wp-content/uploads/2017/02/datasheet.pdf

[18] C. Wolf, “Picorv32 - https://github.com/cliffordwolf/picorv32.”

[19] C. Duran, D. L. Rueda, G. Castillo, A. Agudelo, C. Rojas, L. Chaparro, H. Hurtado, J. Romero, W. Ramirez, H. Gomez, J. Ardila, L. Rueda, H. Hernandez, J. Amaya, and E. Roa, “A 32-bit RISC-v AXI4-lite bus-based microcontroller with 10-bit SAR ADC,” in *2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS)*. IEEE, feb 2016.

[20] S. Benatti, F. Casamassima, B. Milosevic, E. Farella, P. Schonle, S. Fateh, T. Burger, Q. Huang, and L. Benini, “A versatile embedded platform for EMG acquisition and gesture recognition,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 9, no. 5, pp. 620–630, oct 2015.

[21] S. A. Imtiaz, L. Logesparan, and E. Rodriguez-Villegas, “Performance-power consumption tradeoff in wearable epilepsy monitoring systems,” *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 3, pp. 1019–1028, may 2015.

[22] Texas Instrument, “http://www.ti.com/ww/en/msp430.html,” Tech. Rep.

[23] M. Konijnenburg, S. Stanzione, L. Yan, D.-W. Jee, J. Pettine, R. van Wegberg, H. Kim, C. van Liempd, R. Fish, J. Schluessler, H. de Groot, C. van Hoof, R. F. Yazicioglu, and N. van Helleputte, “28.4 a battery-powered efficient multi-sensor acquisition system with simultaneous ECG, BIO-z, GSR, and PPG,” in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, jan 2016.

[24] N. Ickes, D. Finchelstein, and A. P. Chandrakasan, “A 10-pJ/instruction, 4-MIPS micropower DSP for sensor applications,” in *2008 IEEE Asian Solid-State Circuits Conference*. IEEE, nov 2008.

[25] S. Benatti, F. Montagna, D. Rossi, and L. Benini, “Scalable EEG seizure detection on an ultra low power multi-core architecture,” in *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, oct 2016.

[26] B. Keller, M. Cochet, B. Zimmer, J. Kwak, A. Puggelli, Y. Lee, M. Blagojevic, S. Bailey, P.-F. Chiu, P. Dabbelt, C. Schmidt, E. Alon, K. Asanovic, and B. Nikolic, “A RISC-v processor SoC with integrated power management at submicrosecond timescales in 28 nm FD-SOI,” *IEEE Journal of Solid-State Circuits*, pp. 1–13, 2017.

[27] F. Conti, R. Schilling, P. D. Schiavone, A. Pullini, D. Rossi, F. K. Gurkaynak, M. Muehlberghuber, M. Gautschi, I. Loi, G. Haugou, S. Mangard, and L. Benini, “An IoT endpoint system-on-chip for secure and energy-efficient near-sensor analytics,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2017.

[28] E. Flamand. (2016) GAP8: A PULP/RISC-V based SoC IoT processor - ORCONF 2016. [YouTube video]. [Online]. Available: https://www.youtube.com/watch?v=PE3myfKA7ik

[29] D. Bol, J. D. Vos, C. Hocquet, F. Botman, F. Durvaux, S. Boyd, D. Flandre, and J.-D. Legat, “SleepWalker: A 25-MHz 0.4-V Sub- mm2 7- μW/MHz Microcontroller in 65-nm LP/GP CMOS for Low-Carbon Wireless Sensor Nodes.”

[30] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, “Near-threshold computing: Reclaiming moores law through energy efficient integrated circuits,” *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, feb 2010.

[31] ARM Limited, “https://www.arm.com/products/processors/cortex-m.”

[32] F. Conti, D. Rossi, A. Pullini, I. Loi, and L. Benini, “Energy-efficient vision on the PULP platform for ultra-low power parallel computing,” in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, oct 2014.

[33] S. Gal-On and M. Levy, “Exploring coremark^{TM} -a benchmark maximizing simplicity and efficacy,” Tech. Rep.