An FPGA-based Emulation Platform for Edge Computing Node Design Exploration

Theo Soriano, David Novo, Pascal Benoit

An FPGA-based Emulation Platform for Edge Computing Node Design Exploration

Theo Soriano, David Novo, and Pascal Benoit
LIRMM, University of Montpellier, CNRS
Montpellier, France
Email: {firstname}.{lastname}@lirmm.fr

Abstract—**Recent advances in machine learning have made it possible to consider the implementation of smart applications in constrained systems at the edge of the network. These memory and Central Processing Unit (CPU) intensive applications may require specific exploration methodologies to design efficient node computing devices. To better guide and validate these explorations, we need to perform energy and performance evaluations of the system. Software-based evaluation tools are application-oriented and do not consider real-time and hardware constraints. Alternatively, hardware prototyping allows an accurate and real-time evaluation but offers limited flexibility and does not allow agile design exploration of the microcontroller unit (MCU). In this work, we propose a Field Programmable Gate Arrays (FPGA) based edge computing node emulation platform. Our solution combines the flexibility and the real-time capability of programmable logic with hardware prototype evaluation. We present an open-source microcontroller architecture for design exploration which integrates an activity monitor to collect traces at run-time. These activity traces are then used to profile the energy consumption of different components in the edge computing node. Importantly, our FPGA is connected to real sensors and communication modules to enable interactions with the environment during the node evaluation and exploration.**

I. Introduction

In recent years, the concept of *edge computing* has emerged as an efficient solution to reduce the amount of data generated by IoT nodes and thus reduce the energy needed to transport and store this data [1] [2]. In opposition to cloud computing, in edge computing, the data is processed locally (*i.e.,* in the node) and only relevant information is sent over the network. The edge computing approach leads to more complex applications running on the node, resulting in more computations and memory accesses, which may have an impact on the energy consumed by the device. Still, end nodes are highly constrained devices, their limited size and access to energy make energy efficiency a critical issue in these systems.

Ultra-low-power nodes, operating for years on small batteries, are limited to low bandwidth sensors (*e.g.,* temperature, humidity, pressure). However, edge computing based devices will need to process richer sensor data (*e.g.,* images, audio, motions), applying strong constraints on the MCU regarding energy management. The study of the energy distribution (*i.e.,* component-level energy breakdown) is a key enabling tool to optimise the energy efficiency of these objects. These evaluations allow identifying the components that consume the most and therefore guide optimisation. Depending on the application, the critical components may be different. Accordingly, our goal is to propose a solution for evaluating edge computing node performance and energy consumption to enable an application-architecture co-optimisation of the node. When considering evaluating a system, the first option is to use a simple instrumented prototype node. We can easily measure the energy of the devices while the application is running. It is also possible to evaluate the performance of the microcontrollers using the monitors that some of them integrate. However, the main drawback of this solution is its lack of flexibility. Indeed, if this solution is suitable for the evaluation, it is not suitable for architecture exploration. This is where Field Programmable Gate Arrays (FPGA) come into play. FPGA emulation allows real-time interaction with the environment with a high degree of flexibility thanks to its reconfigurability. In addition, this method allows for easy evaluation of the architecture's performance through customised monitors. The use of programmable logic allows the evaluation and exploration phases to be performed on the same target architecture. In this paper, we present an FPGA-based modular emulation platform for edge computing applications. This platform combines FPGA emulation for microcontroller architecture evaluation and exploration, hardware prototype for node-level energy evaluation and a parametric energy model for application exploration.

The rest of the paper is organised as follows. Section II enumerates the state of the art of performance and energy evaluation solutions. Section III describes our FPGA-based emulation platform which is a combination of an open-source instrumented microcontroller architecture [3], a prototype node around an FPGA target and a parametric model for fast energy and performance evaluation. Then, Section IV presents an example of evaluation and explorations made with our platform. Finally, Section V concludes this paper.

II. Related works

Performance and energy evaluation of end nodes is paramount when developing complex applications in edge computing systems. We can classify existing evaluation techniques into three main categories: (1) software simulation; (2) real system (either fabricating one or using an existing one); and (3) FPGA emulation.

Software simulators (*e.g.,* Instruction Set Simulator (ISS) or Cycle Accurate Model (CAM)) are cheap and flexible. However, ISS barely consider the hardware target as they only model instruction executions and are therefore not accurate [4]. Instead, CAM simulators allow building, configuring and simulating a computer architecture using behavioural models for each component which makes them a suitable solution for architecture evaluation and exploration. Their main limitation in our context is that they are computer architecture oriented and do not integrate models for commonly used cores for ULP microcontroller. Furthermore, these simulations do not allow for real-time evaluation and usually focus on the processing unit and memories, hardly modelling the interactions with other components in the node and the environment.

Using a real system provides the higher accuracy. Commercial ultra-low power (ULP) microcontrollers integrate trace modules or Performance Monitoring Units (PMU). This method allows real-time and accurate performance and power evaluations [5] [6]. The monitor output can be used to evaluate architecture bottlenecks during applications execution (*e.g.,* miss rates). Their main limitation is the lack of flexibility, which severely limits design space exploration.

FPGA emulation is our preferred evaluation option. Since the microcontrollers used in IoT nodes usually operate at low frequencies, FPGA emulation allows combining the flexibility of simulation tools and the accuracy and speed of a physical circuit. In addition, it also allows interactions with real-life radio and sensor modules. Enabling real-time evaluation thanks to an FPGA-based prototype. This solution is based on a microcontroller architecture described in a hardware description language (HDL). This makes it easy to add components such as a monitor. In addition, the FPGA allows complete freedom in the choice of the activity to be monitored. We can adapt the monitoring and strategically place the set of probes depending on the architecture parts to be evaluated. For example, Lenormand et al. [7] propose an FPGA implementation integrating a monitor to evaluate and optimize a matrix multiplication accelerator. The monitor allows them to measure the performance of an accelerator, to find its optimal settings and to identify its bottlenecks. However, their work focuses only on accelerator performance and the tested architecture does not integrate peripherals commonly used in IoT applications. In another related work, Ho et al. [8] present an implementation of a PMU in an FPGA-based LEON3 platform, but the presented solution focuses on the processor performance. Their architecture is a complex multi-core architecture that does not correspond to the ULP SoC architecture. Moreover, their PMUs are located in each core and their counters only focus on the core performance (including L1 miss rates).

Open-source HDL designs of ULP SoCs are more directly related to our work. They target the development of prototypes and applications for evaluation and design exploration. For instance, the PULP platform [9] is an open-source low-power RISC-V architecture. PULPissimo [10] is the microcontroller architecture of the more recent PULP chips. This single-core architecture is configurable and modular, it includes peripherals and a high-performance accelerator for edge computing applications. The main issue with this platform is its complexity. It directly targets complex edge computing devices and does not allow to build this complexity gradually. As a result, it also requires a lot of FPGA resources for emulation. Alternatively, Patrigeon et al. presented Flexnode, a reconfigurable prototyping platform used for SoC architecture exploration and real-time application evaluation [11]. The proposed architecture is based on a Cortex-M0 and integrates a memory-transactions dedicated monitor for memory technology exploration. The main limitation of this platform is the simplicity of the architecture (single master bus with memories and peripherals) and the use of a proprietary core developed by Arm, which makes it extremely difficult to monitor the internal events of the microarchitecture. Instead, our proposed solution, which is presented in detail in the next section, can be used to evaluate the edge computing node performance and energy consumption and enable an efficient node application-architecture co-exploration.

III. FPGA-BASED EVALUATION AND EXPLORATION

In this section, we describe our method to evaluate the performance and energy of an edge node while running an application. Our method includes a platform that allows for architecture exploration using programmable logic, but also for application-level exploration. To estimate the node's energy consumption, we measure the system activity at both node and architecture level. These measurements are then used to perform activation-based estimation estimation using architecture and device energy models. Fig. 1 presents the complete method from the IoT service (i.e., the application distributed over the IoT network) to the performance and energy consumption estimation that enables node-level application-architecture co-exploration. Based on one or a set of connected node devices, the purpose of an IoT network is to provide a service, whose computation is distributed over the nodes and the cloud. Once the IoT service is defined, we first define the node-level application constraints and then the node-level hardware constraints (*i.e.,* processing and storage capacity, communication module bandwidth). These constraints guide the design of the application and the reference architecture. The microcontroller architecture is instrumented with an activity monitor and emulated on an FPGA target. During application execution, the FPGA is connected to a sensor and radio board, the traces generated by the monitor are fed to the linear parametric model, which evaluates the performance and the energy consumption of the system based on precomputed power profiles. Finally, we use these evaluations to guide the application and architecture co-exploration.

A. ICOBS: Ibex core based system

When designing a flexible emulation platform for edge computing node design exploration, it is important to choose an adequate reference core architecture. There are many cores in the IoT industry and more specifically in the embedded systems industry. Amongst the largest manufacturers, there is Arm, which offers the Cortex-M series, designed for ultra-low power systems. These proprietary cores are sold as hard

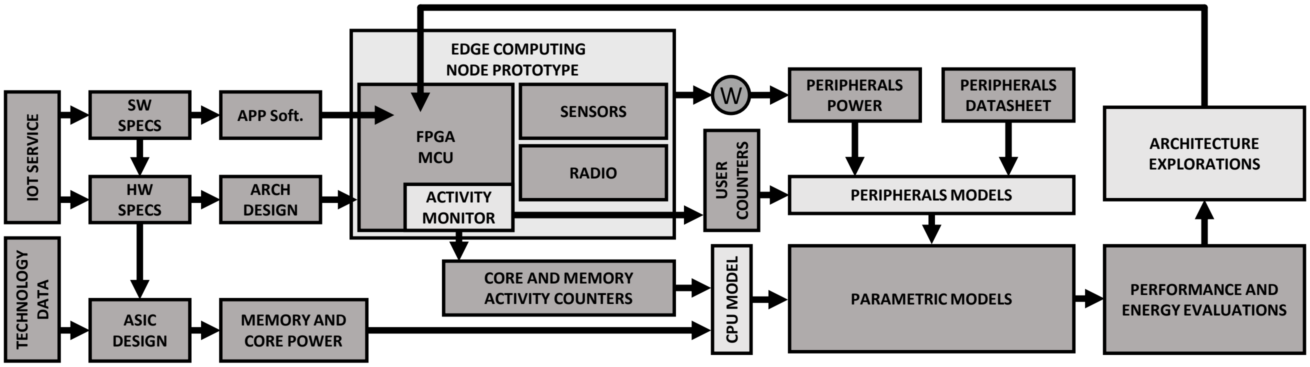

Fig. 1. FPGA-based evaluation flow

macros. The core is described in a netlist optimised for power, area or timing and silicon tested. However, when integrated into a design, a hard macro core is not modifiable, we can only access the I/O interface. To gain access to the internal description of the core and to be able to reach every internal signal, we decided to use an open-source softcore. A softcore is an HDL processor implementation. Fortunately, there is a large number of open-source softcores based on the RISC-V instruction set architecture (ISA). We started using the PULP platform to evaluate two RISC-V softcore: RISCY and Ibex. Because of its simplicity and small size, we decided to use the Ibex core. Ibex is a tiny production-quality open-source 32-bit RISC-V CPU core written in SystemVerilog. This core is heavily parametrizable and well suited for embedded IoT applications. Originally developed as part of the PULP platform [9] under the name ”Zero-riscy”, Ibex is now maintained and developed by lowRISC [12]. It is currently under active development.

The open-source microcontroller architecture we propose is illustrated in Fig. 2. We connect the Ibex core and some memories to an AXI crossbar. The Ibex core integrates two different memory interfaces. Connected to the Instruction Fetch (IF) stage of the core, the IF interface is responsible for fetching instructions from memory to the Instruction-Decode (ID) stage. Externally, the IF interface only performs word-aligned instruction fetches. Instead, The Load-Store Unit (LSU) interface of the core is responsible for accessing the data memory. Loads and stores of words (32 bit), half words (16 bit) and bytes (8 bit) are supported. The IF and LSU interfaces are connected via dedicated bridges to the AXI crossbar as masters. The use of a crossbar allows the integration of multiple masters on the bus, enabling them to potentially access two different memories simultaneously. Of course, the data flow through the crossbar causes latency, but this solution has the advantage of being very flexible and allows easy integration of memories and peripherals. For memory-intensive applications such as machine learning, the size of the memory required by the application and the model can represent several hundred kB. On the FPGA, memories will be implemented with block RAM (BRAM). We used a Nexys video board, this board integrates a Xilinx XC7A200T FPGA which contains 1,625 kB of BRAM. We chose to use an 8 kB memory that contains the startup code and two 512 kB memories that will incorporate the application code and data section. Note, however, that the memory distribution can easily be changed depending on the target application. The 1024 kB are normally sufficient to store both code and data section of tiny machine learning applications. Still, our platform could easily be ported to a larger FPGA if necessary. To interact with other components to build a fully functional IoT node, we connect the AXI crossbar to a single master peripheral bus based on the AHB-Lite protocol. Connected via a bridge, this bus integrates a set of peripherals commonly used in IoT nodes (UART, SPI, I2C, TIMER). We also integrate a reset and clock control module (RCC), this device controls the clock and reset signals of other peripherals and can also perform a soft reset. Finally, we include an activity monitor which is described in more detail in Section III.

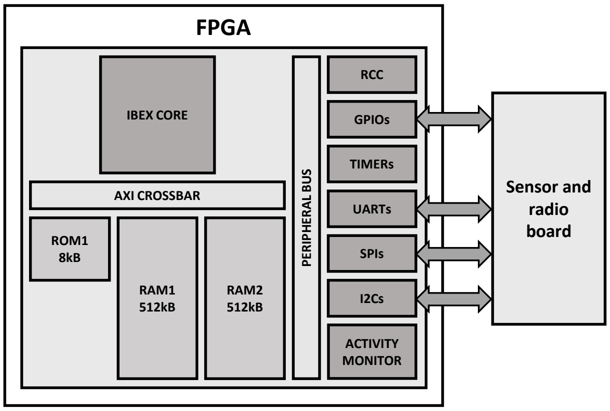

Fig. 2. ICOBS architecture

Fig. 3 shows our node infrastructure. The Nexys video FPGA board is connected to our sensor and radio board via its PMOD connectors. The sensor and radio board integrates a power management unit for battery voltage regulation, several sensors (inertial measurement unit, microphone, temperature, humidity, and camera) and two radio modules (Bluetooth low energy and LoRa).

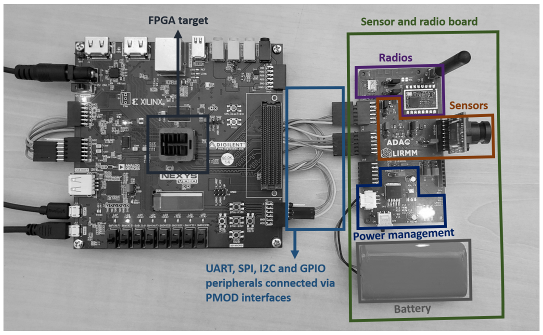

Fig. 3. FPGA-based node prototype

B. Application prototyping

Our node platform runs applications in bare metal (i.e., applications execute instructions directly on logic hardware without an intervening operating system). This is a quite common practice in embedded systems, indeed, for a given application, in most cases, a bare-metal implementation will be faster, use less memory and therefore be more energy efficient. To facilitate application development, our platform includes a firmware with all the drivers and macros that ensure a convenient interface between hardware architecture and application software. Thanks to a custom linker script and startup file, we define the memory areas that will contain the application code and those that will contain data. By default, the code is located in RAM1, and the data is in RAM2. ROM1 contains the boot code (bootloader). At boot time, the bootloader will load the application binary from the UART to RAM1. Once the entire binary is retrieved and copied into the memory, the RCC module will perform a soft reset. Thus, the system restarts from the beginning of RAM1. In that way, the bootloader conveniently allows the application to be modified without generating a bitstream each time. This technique allows fast application development and debugging.

Once the system was fully operational, we developed a machine learning based benchmark for our architecture. For this purpose, we adapted a TensorFlow LITE image classification application based on the MobileNet model for our system. In conclusion, we have a functional reference architecture capable of running wide range of IoT applications from basic sensor node applications to complex machine learning algorithms.

C. Activity monitor

To evaluate our IoT applications, we instrumented our architecture with an activity monitor. As in Patrigeon et al. [11], we first integrated a set of counters that keep count of the events in targeted modules, such as the memories, to make estimations based on the activation (*i.e.,* each activation has a corresponding energy consumption associated; thus, we can evaluate the total energy by counting the activations [13]). We choose to position a probe at each master interface of the crossbar. As all transactions are initiated by the masters, we can see all transactions regardless of the number of slaves in our system. With a basic decoder, we can differentiate each type of access in each component of our architecture. This way we can easily monitor the overall activity of each memory. The activity of a memory is defined by the number of reads and writes on 8, 16 and 32 bits. In the edge computing context, the applications apply high constraints on the CPU and the memories, so we decided to add the core activity monitoring in our solution. To measure the activity of the core, we simply use an output signal from the core indicating its operating mode (run or sleep) so we can easily measure the number of cycles spent in each mode.

Additionally, our solution allows us to evaluate the application by tracking the different phases of the application in order to estimate the energy sharing between the microcontroller and the possible radio modules and sensors. For this purpose, the monitor integrates eight user counters, which count the cycles when they are enabled. This enables us to know how long each module in the system has been in use. The use of these counters has negligible impact on the execution of the application (one single instruction for start and stop) unlike the use of basic timers. For example, to evaluate the consumption of the radio in our application, we can use the user counter to know the duration of use of the communication module in each of its operating modes. Thus, it is enough to know the consumption of our radio in each application phase to estimate the energy consumed by this component.

The activity monitor is integrated in our peripheral bus and contains a set of basic counter registers connected to our probes. The monitoring infrastructure is accessible from the application code. We developed a simple library for reading and writing the monitor registers. Fig. 4 shows how the monitor is integrated into a sample application. Once the boot procedure is finished, the startup phase of the application starts. When the content of the memories is ready, the monitor can be launched. To do this, we simply start the monitor, reset its counters to make sure they are empty and enable them. To measure the duration of each phase of the application, it is easy to use the user counters. Finally, once the application is finished, the first thing to do is to stop the counters by disabling them, then the register values are sent to the computer via the UART.

D. Parametric model

To evaluate the energy consumption of the node, we established a linear parametric model that allows us to model the total energy consumption of the system while executing a particular application. Thus, for each module, we define a succession of operating phases with a duration and a power consumption based on reference documents of the device or on basic energy models (see Fig. 5).

We define the energy of a particular module as:

$$E = \sum_{i=1}^{n} T_i \times P_i, \tag{1}$$

where T_i and P_i are the average power and the duration time of phase i.

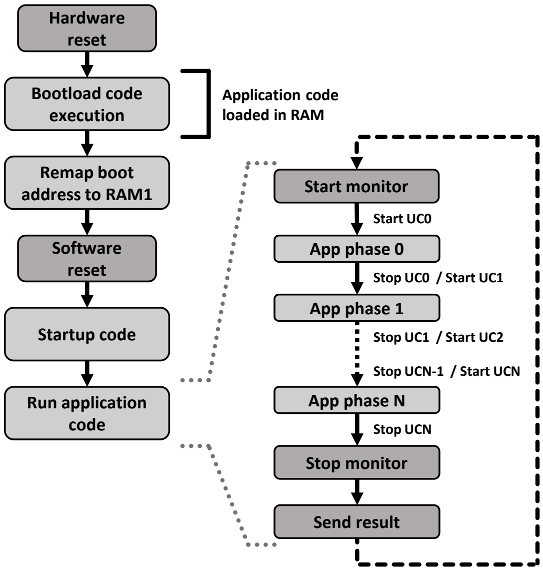

Fig. 4. Monitor utilisation procedure

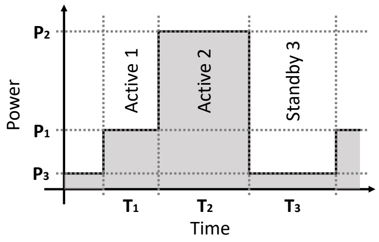

Fig. 5. Example of component power profile

To implement the parametric model, we build a dedicated python application. Fig. 6 shows a simplified view of the model's behaviour. The model takes as input the output of the activity monitor which corresponds to the activation of the core and memories and to the duration of each phase. The parametric model integrates the power profile of the peripherals and the core. In addition, it integrates the energy of each type of memory activation. The parametric model can compute the energy of each component during application execution. Based on their power profiles, the energy of each module during each phase is defined by the corresponding user counter and the corresponding power.

The main advantage of this model is that it allows real-time evaluation and rapid exploration at the application level. Indeed, we can proceed to post-execution explorations by modifying application-specific parameters. For example, in the case of a periodic application, the parameters can be the standby time or the duty cycle; for sensors and communication modules, it can be the amount of data to be collected from the sensor and the amount of data to be sent to the radio. We study the share of each module in the total consumption at node-level and at architecture-level (*i.e.,* we can study the energy of each memory and the core). We can then identify critical modules and phases at the energetic level for each application.

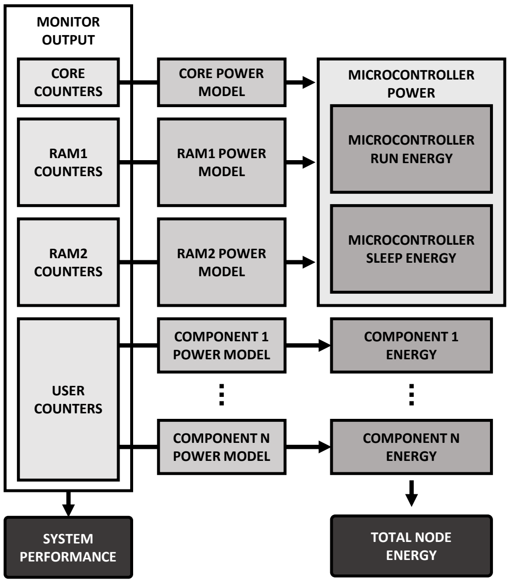

Fig. 6. Parametric model evaluation flow

IV. Experimental results

To illustrate our method, we propose an example based on an edge computing oriented IoT service for intrusion detection. In concrete, the application follows this sequence: the system will capture an image, execute a classification algorithm based on a machine learning model (MobileNet) and send the result on the network. The sequence is activated periodically or based on an event interruption. The application uses a camera to capture images and a Bluetooth module for communications with the network. We use a Nexys video card to emulate the microcontroller architecture described in Section III-A clocked at 42 MHz. The architecture is implemented on the Xilinx XC7A200T FPGA target and uses approximately 6% of the look up tables an 71% of the Block RAM. Table I shows the energy model used in our parametric model. Concerning the power of the microcontroller, we used as energy model of the core, an old version of Ibex (Zero-riscy) [14] and concerning the memories we used a basic SRAM model [15]. The energy model of the SRAM comes from a memory implementation in 28-nm FD-SOI from STMicroelectronics.

Table II shows the output of the activity monitor after an iteration of the application code execution. The first counter represents the number of cycles in which our core was in sleep mode during the monitor listening window. In this case, the monitor only focuses on the main sequence of the application. Therefore, it is normal for this counter to be equal to 0. However, it can be useful to characterise the average duty cycle of an application where the active sequences are unpredictable. The second counter shows us the number of cycles the core has been in run mode. We can see that the execution took about 4.37 billion cycles or about 104.16 seconds. Regarding the memories, the monitor reports the number of reads and writes in RAM1 and RAM2. RAM1 is used as a read-only memory (no writes) and stores the application code section during

TABLE I
POWER MODELS

Microcontroller (ICOBS)		
Core	Run [µW/MHz]	2.08
	Sleep [µW/MHz]	0.73
Memory	Idle [µW]	49.2
	8 bits read [mJ]	7.65e-9
	16 bits read [mJ]	1.53e-8
	32 bits read [mJ]	3.06e-8
	8 bits write [mJ]	5.85e-9
	16 bits write [mJ]	1.17e-8
	32 bits write [mJ]	2.34e-8
Camera (OV2640)		
Shutdown [mW]		0
Capture [mW]		125
Bluetooth (RN4871U)		
Standby [µW]		9.57
TX [mW]		30

TABLE II
MONITOR OUTPUT

Counter	Value
CSCNT	0
CRCNT	4374703489
RAM1RBCR	7072803
RAM1RHCR	62
RAM1RWCR	503581082
RAM2RBCR	7084297
RAM2RHCR	694
RAM2RWCR	123259328
RAM2WBCR	259627
RAM2WHCR	184
RAM2WWCR	6184311
ML SETUP	24102792
GET IMAGE	3963645
INFERENCE RUN	4336268541
SEND RESULT	24897

application execution. RAM2 contains the data section of the application including the stack. Concerning the user counters, the first one (ML SETUP in Table II) is configured to report the application initialisation time, while the second one (GET IMAGE) is configured to report the time to fetch the image from the camera. Once the image is stored in memory, the inference process starts. The third user counter (INFERENCE RUN) indicates the execution time of the inference. This phase represents more than 99% of a sequence run of the application. Finally, the fourth counter describes the time needed to send the results via the Bluetooth module. This time is short because in our case, the result is the label and the probability for both 'person' and 'not person' categories which only represents a few dozen bytes. Once these counters have been integrated into our parametric model, we can start evaluating the node. The estimated energy for the execution of a sequence run of the application is 51.6 mJ. Fig. 7 shows how this energy is distributed between the three main components (microcontroller, camera and Bluetooth module). This information allows us to identify the critical components of the system. In our case, we observe that more than three quarters of the energy is consumed by the microcontroller. This result is consistent with the CPU and memory intensive nature of most edge applications.

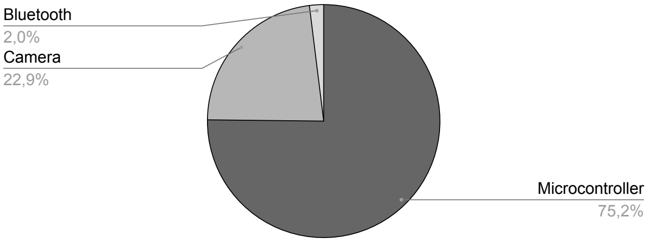

Fig. 7. Energy share in the node after application execution

We should then focus our optimisations on the microcontroller architecture. As mentioned before, for the moment being our method only distinguishes between the energy consumption of the memories and the core. We notice that the memory consumption is about 76% of the microcontroller consumption, consequently we decided to perform memory technology explorations. In the same way we used the SRAM model, we can use models based on other memory technologies. In their work, Patrigeon et al. [15] use a similar model for STT-MRAM technology (see Table III). Fig. 8 presents two different options concerning STT-MRAM integration in our system. We study the integration of STT-MRAM for code section only, then for both code and data section. We observe an important increase in memory consumption when using STT-MRAM. This can be explained by its higher idle power and write energy. This consumption overhead comes from the larger size of its drivers and the properties of the magnetic

TABLE III
STT-MRAM MODEL [15]

Leakage	352 µW
1 bit read energy	0.9 pJ
1 bit write energy	3.0 pJ

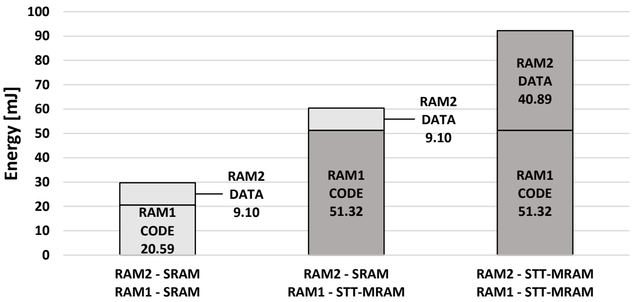

Fig. 8. Memory technology exploration (T=104s)

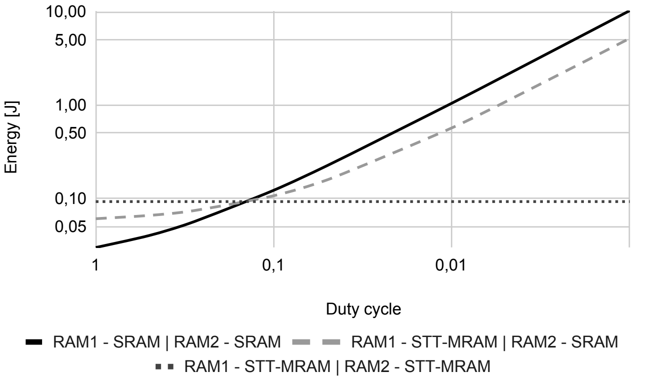

Fig. 9. Example of post-run application exploration: memory consumption as a function of the application duty cycle

tunnel junction. Using STT-MRAM for the code section results in a $2\times$ increase in energy while using STT-MRAM for both the code and data sections results in a $3.1\times$ increase. However, the STT-MRAM technology has the advantage of being non-volatile, so it can be completely powered off and still maintain the data. Now let us consider that our application periodically repeats the sequence presented earlier. Since the duration of the sequence is constant, if the period is longer than the duration of the sequence, then the system can enter in sleep mode until the next sequence begins. With the parametric model, we can easily vary the duration of sleep to model any wake-up period and explore the impact of any duty cycle between run and sleep phases. Fig. 9 presents the total energy of the three memory configurations presented in the Fig. 8 according to the duty cycle between the execution and the sleep phases. The energy consumption of the pure STT-MRAM solution is constant. In fact, since the consumption of these memories during the sleep phase is virtually zero, the duration of the sleep phase has no influence. In the case of SRAM-based solutions, in order not to lose stored data, the memory needs to remain powered even during the sleep phase. Thus, increasing the period with a constant active phase leads to a lower duty cycle. In other words, the duration and thus the proportion of energy consumed by the sleep phase increases due to the consumption of volatile memory for data retention. We can identify that for a duty cycle lower than 10% the STT-MRAM based solutions are more energy-efficient.

V. Conclusion and perspectives

We presented an FPGA-emulated platform of a complete edge computing node based on an open-source RISC-V microcontroller architecture [3]. We also introduced an activity monitoring infrastructure designed to estimate node energy consumption while executing the application in real time. This method allows node-level evaluation but also an energy analysis of the energy bottlenecks to guide future exploration of edge computing nodes. Indeed, the parametric model used allows for post-run explorations at the architecture level (*e.g.,* memory technology exploration) but also at the application level (*e.g.,* wake-up period parameter sweep).

In future work, we plan to include domain-specific (*e.g.,* machine learning) accelerators to improve node computational energy efficiency. We also plan to integrate the activity of the architecture's internal peripherals (*i.e.,* UART, SPI, I2C, accelerator) in our monitor and in the parametric model. Currently, the estimation of memory and core consumption is done on the whole monitor window. We would like to extend the monitor to count the memory accesses for each phase of the application. The energy models used will have to be completed and more precise. Finally, we plan to develop further machine learning based applications and benchmarks.

Acknowledgment

The authors acknowledge the support of the French National Research Agency (ANR), under grant ANR-19-CE24-0017 (NV-APROC project).

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the Workshop on Mobile Cloud Computing (MCC)*, 2012.

[2] M. Peng, S. Yan, K. Zhang, and C. Wang, "Fog-computing-based radio access networks: issues and challenges," *IEEE Network*, 2016.

[3] ICOBS, https://gite.lirmm.fr/adac/icobs/hardware/icobs_mk5_project, [Accessed October-2021].

[4] S. Fontaine, L. Filion, and G. Bois, "Exploring iss abstractions for embedded software design," in *Proceedings of the EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, 2008.

[5] J. Bauer and F. Freiling, "Towards cycle-accurate emulation of Cortex-M code to detect timing side channels," in *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*, 2016.

[6] M. El Ahmad, M. Najem, P. Benoit, G. Sassatelli, and L. Torres, "Adaptive power monitoring for self-aware embedded systems," in *Proceedings of the Nordic Circuits and Systems Conference (NorCAS)*, 2015.

[7] E. Lenormand, T. Goubier, L. Cudennec, and H.-P. Charles, "A combined fast/cycle accurate simulation tool for reconfigurable accelerator evaluation: application to distributed data management," in *Proceedings of the International Workshop on Rapid System Prototyping (RSP)*, 2020.

[8] N. Ho, P. Kaufmann, and M. Platzner, "A hardware/software infrastructure for performance monitoring on leon3 multicore platforms," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2014.

[9] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "PULP: A parallel ultra low power platform for next generation IoT applications," in *Proceedings of the IEEE Hot Chips Symposium (HCS)*, 2015.

[10] PULPissimo, https://github.com/pulp-platform/pulpissimo, [Accessed July-2021].

[11] G. Patrigeon, P. Leloup, P. Benoit, and L. Torres, "Flexnode: a reconfigurable internet of things node for design evaluation," in *Proceedings of the IEEE Sensors Applications Symposium (SAS)*, 2019.

[12] LowRISC, https://github.com/lowRISC/ibex, [Accessed July-2021].

[13] G. Contreras and M. Martonosi, "Power prediction for intel XScale processors using performance monitoring unit events," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2005.

[14] P. Davide Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini, "Slow and steady wins the race? a comparison of ultra-low-power RISC-V cores for internet-of-things applications," in *Proceedings of the International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017.

[15] G. Patrigeon, P. Benoit, L. Torres, S. Senni, G. Prenat, and G. Di Pendina, "Design and evaluation of a 28-nm FD-SOI STT-MRAM for ultra-low power microcontrollers," *IEEE Access*, 2019.