Design Space Exploration of Microcontroller Memory Architectures for Intermittent Computing at the Edge

Theo Soriano

THESIS TO OBTAIN THE DEGREE OF DOCTOR
OF THE UNIVERSITY OF MONTPELLIER

In SyAM - Automatic and Microelectronic Systems

Doctoral School: Information, Structures and Systems sciences (I2S)

Research Unit: LIRMM

Design Space Exploration of Microcontroller Memory Architectures for Intermittent Computing at the Edge

Presented by Theo Soriano
December 15, 2022

Under the supervision of:
Pascal Benoit and David Novo

In front of the jury composed of:

Lorena Anghel, Professor, University of Grenoble-Alpes, INP	Reporter
Pascal Benoit, Associate professor, University of Montpellier, LIRMM	Thesis Director
Francky Catthoor, Professor, KU Leuven, IMEC	Examiner
Gregory Di Pendina, Research Engineer, CNRS, Spintec	Examiner
Laurent Latorre, Professor, University of Montpellier, LIRMM	Examiner, President
David Novo, Research Scientist, CNRS, LIRMM	Thesis Co-supervisor
Jean-Michel Portal, Professor, University of Aix-Marseille, IM2NP	Reporter
Lionel Torres, Professor, University of Montpellier, LIRMM	Invited

UNIVERSITÉ DE MONTPELLIER

Abstract

Since the emergence of wireless sensor networks in the early 2000s, there has been a growing interest in many domains, including environment, agriculture, health, energy, or industry, with diverse sensing / actuating and wireless communications approaches. The ever-increasing amount of data generated by these end-node devices led to the distribution of data processing across the network, especially for performance, energy and security requirements (*i.e.*, edge computing). This distribution increases sensor nodes' processing and storage requirements. A node can, for example, carry out statistics on the data collected and determine the relevance of transmitting the information. Today, it seems possible to go as far as the integration of more sophisticated algorithms, such as embedded machine learning inference. Thus, our objective is to optimise sensor nodes energy efficiency to enable more computing and storage in these devices. To achieve this goal, we address three challenges.

The first challenge is to model the execution of an application in a sensor node in order to be able to evaluate the energy of its components to guide future optimisation. For this, we study the architecture and operation of a sensor node to build an activity-based energy model. These devices embed a Microcontroller Unit (MCU) responsible for the processing and storage. After studying the architecture of these MCUs and how non-volatile memories enable energy-saving strategies, we build an activity-based energy model for MCU, enabling us to evaluate the energy of these different parts allowing architecture exploration.

The second challenge is to find a solution for energy evaluation and architecture exploration at both the sensor node and MCU levels. To address it, we propose an FPGA-based sensor node prototype that can embed a wide range of applications. An MCU architecture with a RISC-V CPU core, memories and peripherals is implemented on an FPGA target, which is connected to a sensing and radio unit to build a real-life operating sensor node. The architecture embeds a monitoring unit that traces the activity of the different components at both levels. Thus, these activity traces allow us to evaluate the energy consumed by these different components. We propose two applications, a temperature-like sensor node application and an intelligent node-like

one for keyword detection based on machine learning inference. With this platform, we show that intelligent radio use allows us to reduce the node's power consumption. In this context, the consumption of the MCU and more particularly during inactive phases becomes a significant part of the energy consumed by the node.

The final challenge is to find in this context how the integration of emerging Non-Volatile Memory (NVM) in MCU architecture can help increase sensor nodes' energy efficiency. To this end, we propose MemCork, an MCU memory exploration tool. Based on the platform's output, the tool automatically explores different memory technologies, sizing, and data mapping to find the most energy-efficient solutions for a given application. We then use this tool on our two applications to find how emerging memory technologies can help reduce MCU energy.

In summary, this thesis explores sensor nodes' communication/processing trade-offs. It explains how emerging NVM can be integrated into MCU architecture to increase sensor node energy efficiency. We believe our exploration method can be used to perform further application-architecture co-optimisation beyond the ones covered in this manuscript. We hope our results will enable the design energy efficient smart sensor nodes.

Résumé de la Thèse

Depuis l'émergence des réseaux de nœuds capteurs au début des années 2000, on constate un intérêt croissant dans de nombreux domaines, notamment l'environnement, l'agriculture, la santé, l'énergie ou l'industrie, avec diverses approches de détection / actionnement et de communication sans fil. La quantité croissante de données générées par ces nœuds a conduit à la distribution du traitement des données dans le réseau, en particulier pour des raisons de performance, d'énergie et de sécurité. Cette distribution augmente les besoins en traitement et stockage des données dans ces nœuds. Un nœud peut, par exemple, analyser les données collectées et déterminer la pertinence de transmettre l'information. Aujourd'hui, il semble possible d'aller jusqu'à l'intégration d'algorithmes plus sophistiqués, par exemple basés sur l'intelligence artificielle. Ainsi, notre objectif est d'optimiser l'efficacité énergétique des nœuds pour pouvoir augmenter leurs capacités de calcul et de stockage. Pour atteindre cet objectif, nous relevons trois défis.

Le premier défi est de modéliser l'exécution d'une application dans un nœud de capteurs afin d'évaluer l'énergie de ses composants pour guider les futures optimisations. Pour cela, nous étudions l'architecture et le fonctionnement d'un nœud de capteurs afin de construire un modèle énergétique basé sur l'activité. Ces dispositifs embarquent un microcontrôleur (MCU) responsable du traitement et du stockage. Après avoir étudié l'architecture de ces MCU et la façon dont les mémoires non volatiles permettent diverses stratégies d'économie d'énergie, nous construisons un modèle énergétique basé sur l'activité du MCU permettant d'évaluer l'énergie de ses différentes parties pour l'exploration.

Le deuxième défi est de trouver une solution pour l'évaluation de l'énergie et les explorations de l'architecture à la fois au niveau des nœuds capteurs et des MCU. Nous proposons un prototype de nœud capteur basé sur un FPGA qui peut intégrer une large gamme d'applications. Une architecture de MCU avec un cœur RISC-V, des mémoires et des périphériques est implémentée sur un FPGA, qui est connecté à un capteur et une radio pour construire un nœud capteur fonctionnant en conditions réelles. L'architecture intègre un moniteur qui collecte l'activité des différents composants, qui

par la suite est combinée aux modèles énergétiques pour évaluer leur consommation. Nous proposons deux applications, une application simple de type capteur de température et une application de type nœud intelligent basée sur l'intelligence artificielle. Avec cette plateforme, nous montrons que l'augmentation du traitement de données dans le nœud minimise sa consommation pour la communication, entrainant une augmentation de l'énergie pour le traitement et le stockage des données.

Le dernier défi est de trouver dans ce contexte comment l'intégration de mémoires non volatiles (NVM) émergentes dans l'architecture des MCU peut aider à augmenter l'efficacité énergétique des nœuds capteurs. Nous proposons MemCork, un outil qui explore automatiquement les différentes technologies de mémoire, le dimensionnement et le positionnement des données afin de trouver les solutions les plus efficaces sur le plan énergétique pour une application donnée. Nous utilisons ensuite cet outil sur nos deux applications pour découvrir comment les technologies de mémoire émergentes peuvent aider à réduire la consommation des MCU.

En résumé, cette thèse explore les compromis des applications de nœuds capteurs et comment les NVM émergentes peuvent être intégrées dans l'architecture des MCU pour augmenter l'efficacité énergétique des nœuds capteurs. Nous pensons que la méthode d'exploration proposée peut être utilisée pour réaliser d'autres optimisations des applications et des architectures au-delà de celles couvertes dans ce manuscrit. Nous espérons que nos résultats permettront de concevoir des nœuds capteurs intelligents et économes en énergie.

Acknowledgements

First, I express my deepest gratitude to my advisors, Pascal Benoit and David Novo. I want to thank my thesis director, Pascal Benoit, for his support, guidance, and mentorship throughout the last three years. His expertise and priceless advice have been essential in guiding my research and helping me achieve my goals. I also want to thank my co-supervisor, David Novo, for his listening and invaluable help. His encouragement and ideas have significantly contributed to the success of this work. I am deeply grateful for everything they have done for me, especially their dedication and support throughout my PhD journey.

I would also like to thank the members of my thesis jury for evaluating my work. As reporters, Lorena Anghel and Jean Michel Portal provided insightful feedback and valuable suggestions. As examiners, Gregory Di Pendina and Francky Catthoor meticulously evaluated my work and provided valuable insights. And last but not least, Laurent Latorre, as president of the jury, played a crucial role in guiding me throughout my studies since my arrival at Polytech Montpellier. His advice, encouragement and expertise were precious to me.

I am also thankful to the NV-APROC project members, Gregory Di Pendina, Jean-Frederic Christmann, Guillaume Prenat and Chadi Al Khatib for their valuable contributions and support. I want to thank my colleagues from the laboratory: Guillaume Patrigeon, Loïc Dalmasso, Yohan Boyer, Quentin Huppert, Loïc France, Paul Leloup, Jonathan Miquel and Paul Delestrac for their support and helpful discussions. I am also thankful to the interns involved in this work: Axel Legué, Romain Pessato, Victoria Bial, Reda Arifi, Remy Baillet, Aurelien Bouchot, Bruno Lovison Franco, and Aymen Romdhane.

Finally, I want to acknowledge the unwavering support of my friends and family, especially my parents and my partner Margot, who have been my constant source of love and encouragement throughout my PhD journey. Their belief in me has been a continuous source of motivation, and I am forever grateful for their love and support.

This work is dedicated
to Gisèle and Jean Soriano

Contents

List of Figures

List of Tables

List of Acronyms

ADC	Analog-to-Digital Converter
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BLE	Bluetooth Low Energy
BRAM	Block RAM
CAN	Controller Area Network
CMOS	Complementary Metal-Oxide-Semiconductor
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSR	Control and Status Register
DAC	Digital-to-Analog Converter
DBPSK	Binary Phase-Shift Keying
DDR-DIMM	Double Data Rate Dual In-line Memory Module
DMA	Direct Memory Access
DRAM	Dynamic RAM
ECM	Electret Condenser Microphone
EEPROM	Electrically Erasable Programmable Read-only Memory
EMU	Energy Management Unit
ENVM	Emerging Non-Volatile Memory
FF	Flip-Flop
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FRAM	Ferroelectric RAM
GPIO	General-Purpose Input/Output
GPU	Graphics Processing Unit
HDL	Hardware Description Language
I2C	Inter-Integrated Circuit
ICOBS	Ibex Core Based System

ID	Instruction-Decode
IF	Instruction Fetch
IMU	Inertial Measurement Unit
IoT	Internet of Things
ISA	Instruction Set Architecture
ISS	Instruction Set Simulator
ITU	International Telecommunication Union
LPWAN	Low-Power Wide-Area Network
LSU	Load-Store Unit
MCU	Microcontroller Unit
MEMS	Micro-Electro-Mechanical System
MIPS	Microprocessor without Interlocked Pipelined Stages
ML	Machine Learning
MOS	Metal-Oxide-Semiconductor
MPU	Microprocessor Unit
MRAM	Magnetoresistive RAM
MTJ	Magnetic Tunnel Junction
NV	Non-Volatile
NVFF	Non-Volatile Flip-Flop
NVM	Non-Volatile Memory
NV-MCU	Non-Volatile Microcontroller Unit
OBI	Open Bus Interface
OTA	Over The Air
PCRAM	Phase-Change RAM
PLL	Phase-Locked Loop
PMU	Performance Monitoring Unit
PPS	Peripheral Pin Select
RAM	Random Access Memory
RCC	Reset and Clock Control
RF	Radio-Frequency
RISC	Reduced Instruction Set Computer
RNG	Random Number Generator
RRAM	Resistive RAM
RTC	Real Time Clock
RTL	Register Transfer Level
SOT	Spin-Orbit Torque MRAM
SPI	Serial Peripheral Interface
SRAM	Static RAM
SSD	Solid-State Drive
STT	Spin-Transfer Torque MRAM

TAS	Thermally Assisted MRAM
UART	Universal Asynchronous Receiver-Transmitter
ULP	Ultra Low Power
UNB	Ultra Narrow Band
USB	Universal Serial Bus
VM	Volatile Memory
WAN	Wide-Area Network
WSN	Wireless Sensor Network
WUC	Wake Up Controller

I

Introduction

Contents

1.1 Context

The first connected object was a vending machine in the early 1980s. This machine could communicate its contents over a network, allowing people to remotely obtain the current number of drinks left [1]. Light sensors were placed near the machine's indicator lights. These sensors were connected to a gateway, which was connected to the ARPANET, the precursor of today's Internet. In 1990, John Romkey designed the first object connected to the Internet, a toaster which can be triggered from a computer [2].

With the improvement of communication technologies and especially wireless networks, today's connected objects are omnipresent around us, and their interconnect network forms the so-called Internet of things (IoT). *IoT* has been defined by the International Telecommunication Union (ITU) as "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies" [3].

1.1.1 Wireless sensor nodes in the IoT network

Figure 1.1 shows the typical infrastructure of the IoT. Connected objects may be deployed anywhere: as sensors or actuators in buildings, fields, and streets, or directly on living beings as wearables, for instance. They integrate a wireless communication interface to exchange data with the network composed of gateways and routers. On this network, there are servers to collect data and provide services. The IoT includes all connected objects, which can be smart devices, autonomous agricultural or industrial equipment, systems for air quality monitoring in cities, or even home automation systems. As it can be seen in Figure 1.2, according to an estimate made in 2022, the number of IoT devices deployed is constantly increasing, and the number of connected objects in 2030 is estimated to be around 29.4 billion.

In this thesis, the focus is on wireless sensor nodes, which play an important part in IoT. These devices are deployed in an environment to collect data that can then be used to provide a service [5]. In 1998, as part of the Smart Dust project, the first millimetre-sized sensor node capable of measuring information using sensors and communicating it to the outside world was created [6]. Today, sensor nodes can measure many physical phenomena such as temperature, humidity, pollution levels, light, motion, image or sound. These devices are widely used in different sectors [7]:

- Environmental monitoring: for air pollution monitoring [8] or for natural disaster prevention (*e.g.*, early warning on geohazards [9])

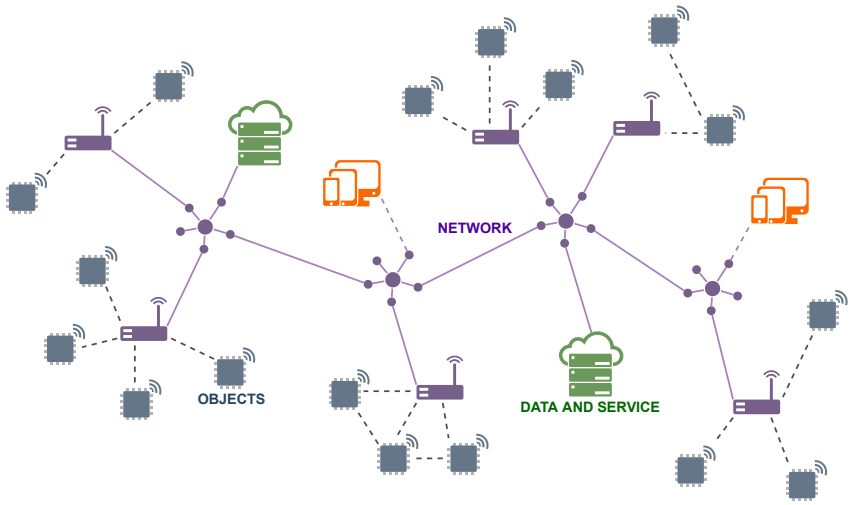


Figure 1.1: *Internet of Things (IoT) infrastructure*

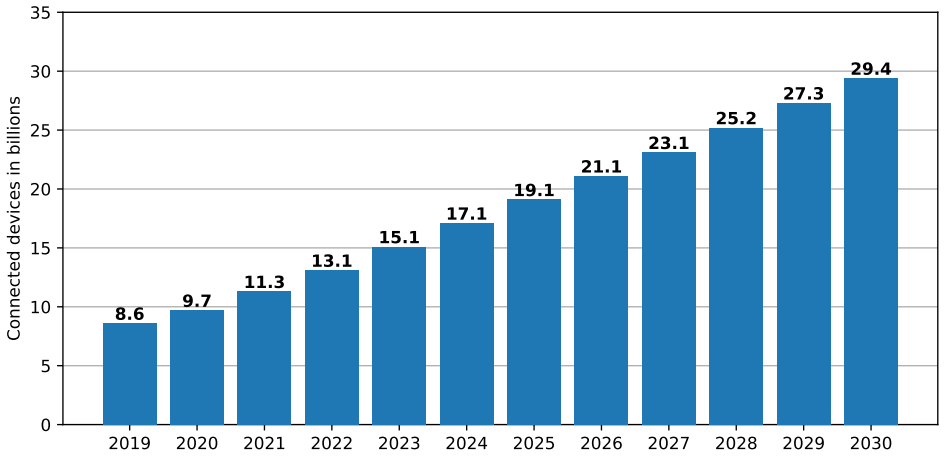


Figure 1.2: *Number of Internet of Things (IoT) connected objects worldwide from 2019 to 2021, with forecasts from 2022 to 2030 [4]*

- Manufacturing: for the identification and location of tools, merchandise and people or for production optimisation [10].
- Military: for the localisation and tracking of soldiers and vehicles, for the detection of chemical substances or for the detection of biological and explosive weapons [11]
- Health: for physical condition monitoring, fall detection, medicine stock monitoring [12] or even to recognise and observe the COVID-19 patients [13].
- Smart cities: To monitor big infrastructures (*e.g.*, Malaysia Pahang University campus [14]) or for real-time monitoring of city installations (*e.g.*, bridge safety monitoring [15])
- Agriculture: To monitor and control the environment of a greenhouse in order to manage the climate at an optimal level for crop growth to increase production [16] or for smart water management [17].

This thesis mainly considers resource-constrained wireless devices that are energy self-sufficient using a battery or an energy harvesting system. As depicted in Figure 1.3, sensor nodes are composed of three main elements: one or more sensing units, which retrieve physical data; a radio unit, which sends the data to the network; and a Microcontroller Unit (MCU), which embeds and runs the application. Indeed, the MCU acts as the conductor. Thanks to its peripherals, embedded intelligence and memories, it collects data from the sensing unit and exchanges data with the network through the radio unit. These controllers allow a high degree of integration, low power consumption and, finally, a reduced cost which is not negligible in the implementation of networks comprising a large number of nodes. Sensor nodes may be deployed in any environment (*e.g.*, human body [18]). Their size is minimised to integrate them into their environment, leaving limited space for energy storage. This is why the scientific community is working to reduce the power consumption of the different components of the node in order to increase their autonomy and avoid too frequent maintenance.

1.1.2 Computing in the IoT network

Figure 1.3 describes the organisation of an IoT network. Sensor nodes are deployed in an environment in which they collect data through their sensing unit. They communicate through their radio unit with a gateway which itself communicates with servers via a network to provide a service. This service can be based on simple data collection and storage, but there is often a need to process the data. Data processing can be done at three different levels:

- **Cloud computing**: The data processing is done at the cloud level, and the data collected by the sensor nodes are routed to the server which processes them.
- **Fog computing**: The processing is distributed to the devices responsible for data routing between sensor nodes and the server (*e.g.*, gateways and routers)
- **Edge computing**: Also known as near-sensor computing, the processing is done directly in the sensor nodes and/or in the gateways.

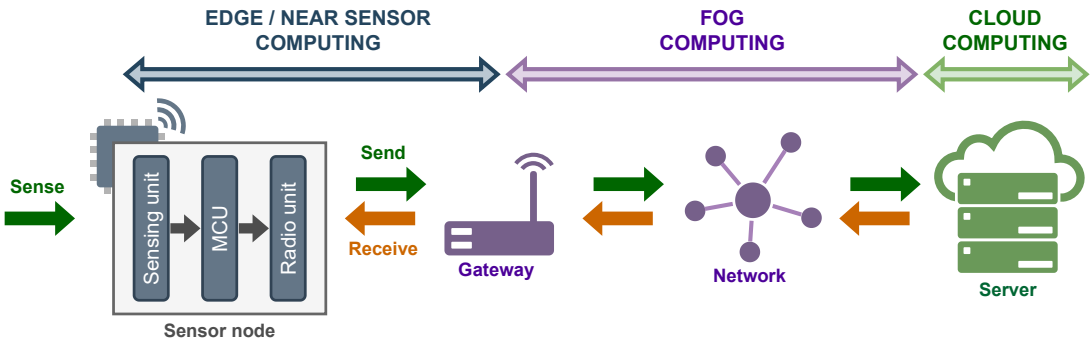


Figure 1.3: *Internet of Things network organisation*

The traditional approach is cloud computing because of the low computing and storage capacity of sensor nodes. However, this approach has many drawbacks. The communication of raw data over the network results in very high bandwidth usage, especially if there are numerous nodes. The communication of a large amount of data through many electronic devices results in high activity and, therefore, high energy consumption. In addition, the circulation of raw data, which may be private, on the network can cause security issues. This is where the fog and edge computing approaches come into play. The distribution of data processing across the network allows for the circulation of meaningful information only. Edge computing reduces the traffic on the network and, therefore, the consumption of the devices involved. In addition, it can reduce the security issue related to the circulation of private data on the network [19].

1.2 Challenges in edge computing

In the edge computing approach, end-nodes such as sensor nodes can, for example, perform statistics, filtering or encryption on the collected data and determine the relevance of the information in order to communicate only useful and secure data. It is even possible to integrate algorithms based on machine learning (*e.g.*, for real-time fall detection [20]). This approach leads to an increase in the complexity of the MCU's operations and its storage needs, resulting in an increase in its energy consumption. However, as explained above, these nodes have a very limited energy supply, which directly conflicts with edge computing needs. In this thesis, we are particularly interested in the role and optimisation of resource-constrained end-node devices in this context. Thus, **the main challenge is to improve the energy efficiency of sensor nodes to enable further computing and storage at the edge.**

To address this challenge, we first need to be able to model the energy consumption of a sensor node. Furthermore, it is necessary to take into account all the different components of a sensor node in order to identify those that are critical in terms of consumption. The modelling must be able to isolate the contribution of each different component. Thus, **we need a holistic approach to model the consumption of a sensor node and its different components during the execution of an application.**

We need to be able to evaluate the energy and performance of a sensor node to explore various optimisations. For this, it is fundamental to consider the intermittent nature of the operation of these objects. Indeed, sensor nodes perform sensing, processing and communication, but these functions are only executed periodically or following a trigger event, meaning that the device remains inactive for the rest of the

time. Events can be (1) periodic (*e.g.*, timer interrupts, periodic data received by a sensor or the radio module) or (2) aperiodic (*e.g.*, interruption due to sensor threshold exceeding or sudden radio reception). Thus, sensor nodes operate intermittently alternating between active and inactive state. The event-based nature of these applications makes the behaviour of the node highly dependent on its environment. In the case of edge computing, this goes further as the possible data transmission can be conditioned by the information collected in the environment. Thus, **we need a solution to evaluate sensor node performance and energy consumption in real-life conditions while enabling application-architecture co-exploration.**

Wireless sensor nodes can sometimes remain inactive for long periods. Indeed, they often operate in the normally-off computing paradigm [21]. The system is programmed to be inactive by default, waiting for a specific event to wake up and perform some operations before returning to the inactive state. Energy savings are usually achieved by low duty-cycling the system, meaning that the active phases represent a minimal amount of time compared to the inactive phases. In other words, the system is inactive most of the time. Therefore, in such nodes, the energy consumption during the inactive phase becomes critical. For low duty-cycled applications, even if the power consumption of the system during inactive phases can be greatly reduced, the energy consumed during these phases can still represent the majority of the energy consumed by the system resulting in standby power critical applications [22]. Emerging non-volatile memory technologies such as FRAM, RRAM or MRAM have grown in maturity. As a result, their integration into programmable computer architectures has been the subject of a vast amount of research [23]. Emerging non-volatile memory technologies are also disrupting the industrial landscape. Key market players like Everspin, Global Foundries, ARM, and Samsung have been selling MRAM-enabled industrial products and applications since 2016. However, integrating such memory technologies into wireless sensor nodes raises several scientific issues related to the technology itself and the benefits expected at the application level. Some of these memories have SRAM-like performance while having the capability to hold stored data even if the power is turned off. Accordingly, they open new opportunities in the design of ultra-low-power sensor nodes for normally-off applications and battery-less energy harvesting-based systems. Emerging NVMs are used to provide instant on/off capabilities to MCUs with near-zero power consumption during the inactive phases [24, 25]. **Our final challenge is to determine how the integration of MRAM into sensor nodes can help to improve their energy efficiency, allowing them to perform more complex operations with the same amount of energy.**

1.3 Main objective and contributions

The main objective is to improve the energy efficiency of sensor nodes to enable further computing and storage at the edge by using emerging non-volatile memories. To this end, we make the following contributions.

In Chapter 2, we propose a parametric model for sensor node energy evaluation. First, we describe the general structure of an IoT with a three-layer model. Then, we describe the basic architecture of a sensor node as well as each of the components it embeds. After studying their operation, we propose a sensor node application model. This model is based on a set of operating phases common to all sensor nodes. Taking into account several operating modes of the different components that compose a node, it is possible to build an equivalent energy model. Based on several application-dependent parameters, the energy model allows us to estimate the consumption of all the components of a sensor node and, thus, its total consumption and the share of each component for any application following our application model. In Chapter 3, we look in more detail at the architecture of the MCUs responsible for processing and storage in the sensor nodes. We study their memory architecture and how the integration of emerging non-volatile memories in this architecture can enable energy-saving strategies. Finally, based on this study, we propose an energy model of the different components of MCUs with a special focus on their memories.

In Chapter 4, we propose a method for the evaluation of sensor node energy and performance to enable an application-architecture co-optimisation of the node. In a previous work of our team, Patrigeon et al. proposed Flex Node, an FPGA-based node prototype [26]. This prototype node consists of a custom MCU architecture implemented on an FPGA target connected to a real-life sensor and a radio unit to build a sensor node. The proposed MCU architecture has been instrumented with a monitor peripheral to perform event-based energy evaluation. Based on this previous work, we propose an improved FPGA-based node prototype. The CPU architecture is now based on a 32b RISC-V CPU core. In addition, it is now capable of storing a wide range of applications, including computing-intensive ones. For monitoring, we implement a separate unit capable of tracing various events in real-time during the execution of the application while being truly non-intrusive (*i.e.*, does not affect the execution of the application). This monitoring unit allows to retrieve the application run-time parameters used by the energy model defined in the previous chapter. We then propose two different reference applications representing the wide spectrum of software applications that sensor nodes can embed. The first application is a simple temperature-like sensor node that represents low-end nodes. The second application represents a smart sensor node embedding machine learning algorithms. Finally, we run these two appli-

cations on the node prototype in real conditions while varying their parameters such as their wake-up interval to evaluate the energy consumption and distribution in the end-nodes.

In Chapter 5, we propose MemCork, a tool for memory exploration based on the prototyping platform described in the previous chapter. Based on size-dependent energy models of volatile and non-volatile memories and a fined-grained evaluation of the memory activity, our tool exhaustively explores the possible data mapping and memory architecture combinations and evaluates them. The tool allows to find the most energy efficient memory architecture and data mapping for a given application. We use this tool on two applications to assess how emerging memory technologies can help reduce the memory energy consumption of the MCU. Then, we study the effect of MCU memory optimisation on total end-node energy.

Finally, the conclusions and future work directions are discussed in the Chapter 6.

This work was part of the NV-APROC ANR project (ANR-19-CE24-0017), which targets the design of a Non-Volatile Asynchronous PROCessor [27], in collaboration with Spintec [28] (main coordinator) and CEA-Leti [29]. NV-APROC aims to combine the asynchronous design technique and MRAM emerging non-volatile technologies to decrease the power consumption of sensor nodes significantly.

II

End-node functional analysis

Contents

The so-called "connected objects" have become widespread in recent years and constitute the visible part of what is known as the Internet of Things. Behind this term, we can find very basic devices, such as simple connected temperature sensors, to much more complex devices such as smartphones, which can also be classified in this category of connected objects. These objects, or end-nodes, are the most visible part of the network, as they are directly in contact with the environment, on which they can act (we talk about actuator), collect information (we talk about sensor), or both. In this thesis, we are particularly interested in sensor nodes with very limited computing and energy resources: this is why increasing their energy efficiency is a crucial challenge.

To address this challenge, we first need to understand how sensor nodes work from their embedded application software to their hardware architecture. Then, we have to model the consumption of a sensor node and its different components during the execution of an application.

In this chapter, we will begin by clarifying the notions of Internet of Things and Edge Computing introduced in Chapter 1, focusing specifically on the current trend of bringing computing closer to data capture and the potential benefits of this approach. We will see the consequences of such evolution for sensor nodes regarding computation and energy consumption. To better target the optimisations required for greater energy efficiency, we will analyse the general operation of these sensor nodes, as well as their hardware architecture. We will finally propose a high-level energy modelling of a generic sensor node which will be used as a basis for the contributions proposed in the following chapters.

2.1 IoT and edge computing

The vision of a network with many energy-autonomous sensor nodes that collaborate with each other to collect and route information has been the motivation behind many research works [30]. Twenty years ago, we were talking about Wireless Sensor Networks (WSNs), a term that evolved in the early 2010s into IoT, as we had gone from dedicated networks with very specialised nodes, to a globalised network with heterogeneous devices.

Wireless sensor nodes play an important part in IoT. They can be used in many new application domains such as environmental monitoring, manufacturing, smart cities or even health. Today, the deployment of sub-millimetre-scale autonomous computing and sensing platforms remains a challenge in many sectors. These devices embedded in the world, from cities to our bodies, give us greater visibility into the environment around us [31]. In this section, we describe the standard architecture of IoT and sen-

sor nodes. Then, we show how IoT applications have evolved and how intelligence has migrated in the network. Finally, we show how these evolutions generate new constraints on the energy efficiency of sensor nodes.

2.1.1 IoT network

Figure 2.1 describes the typical structure of a sensor node and how it interfaces with an IoT network. IoT devices are distributed over three different layers: service, network, and perception layers [32]. The devices in each layer and their interconnection work together to provide a service to a user. For example, the Libelium company proposes sensor nodes for parking monitoring. The sensor nodes are deployed at each parking spot and connected to an IoT network. Thus, the proposed service allows users to know in real-time which parking spots are free or occupied [33]. Next, we describe each layer in further detail.

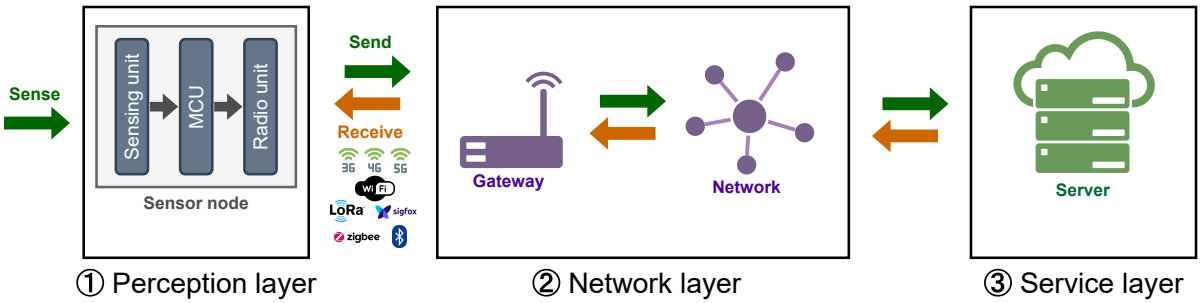


Figure 2.1: *IoT network organisation*

Perception layer: The sensor nodes are located in this layer. The typical tasks of such devices consist in collecting data through a sensing unit (*i.e.*, sensing task S), communicating with the network layer through a radio unit (*i.e.*, communication task C) and eventually performing some processing on collected or received data (*i.e.*, processing task P). These tasks are further discussed in Section 2.2. Sensor nodes are autonomous devices (*i.e.*, they do not have a continuous energy source). They use one or more energy accumulators (*e.g.*, battery or super-capacitor) and sometimes one or more energy harvesting systems. Some of them, especially in the case of a wearable or implantable device (*e.g.*, glucose or blood pressure sensors), must also satisfy strict size constraints to be as non-intrusive as possible. This last constraint limits the size of the energy accumulators and energy harvesters, which makes energy management even more critical. This is why energy management is a major issue when designing such devices. The objective is to ensure a long autonomy (in years) to avoid too regular and expensive maintenance, which would quickly increase the total cost of the service. For example, the French company Ineo-Sense offers a battery-powered sensor node dedicated to industrial and environmental measurement. This

node measures humidity and temperature. With a measurement made every 20s and a report (Min/Max/Average) sent by radio every hour, the node has an autonomy of 6 years [34]. The sensor nodes are designed around a Microcontroller Unit (MCU). The MCU is the conductor of the node. It embeds the application software and controls both sensor and radio units. MCU are small, low cost, offer low power consumption and embed the necessary storage capacity, thus avoiding the addition of extra components which, most of the time, would increase the footprint, the cost and/or the power consumption of the final product [35].

Network layer: Based on gateways or base stations, this layer ensures the interconnection between the perception and service layers. Indeed, it is necessary to use gateways capable of receiving data from the sensor nodes (which use the same radio protocol) and transmitting it to the service layer. Thus, these gateways integrate a radio unit connected to an MCU or, in some cases, to a low-power Microprocessor Unit (MPU) capable of interfacing with the internet network via an IP interface. These gateways are generally compact; even if they are usually powered via the electrical network, they are still designed to have low power consumption as they may be deployed in large numbers. For example, the maximum power consumption of the LoraWAN base station used by Libelium [36] is limited to 15 watts.

Service layer: Based on servers, this layer is responsible for ensuring user access to the service. In this layer, the data sent by the other layers are stored and can be accessed by the user. These servers are high-performance computers often located in big data centers. They are capable of storing and processing large amounts of data, thanks to several powerful MPUs and/or Graphics Processing Units (GPUs). They also embed high-speed network interfaces and mass storage systems (several TB). They consume a high amount of energy (hundreds of watts for a single unit) but are able to provide multiple services at the same time. For example, the Libelium cloud provides access to servers that provides data storage for historical queries (up to 13 months) and a graphical display of data collected by sensors to the user [37].

2.1.2 Evolution and trends of end-node computing

We have described the nature of the devices that are involved in an IoT network and their interconnection to provide the user with a service. We are now going to focus on the workload of each of these devices. Indeed, the routing of the raw data is often not sufficient. Thus, IoT devices must often perform statistics, filtering, and encryption on the collected data. They also may have to make decisions. For this, many of these decisions rely on inference performed by previously trained machine learning models. As we can see in Figure 2.2, the computing and storage capacity are not uniformly

distributed on the network. Indeed, the devices of the service layer embed the most computing and storage capacity. In contrast, the devices of the peripheral layers (sensor nodes and gateway) embed very low computing and storage capacity because of their high energy constraints.

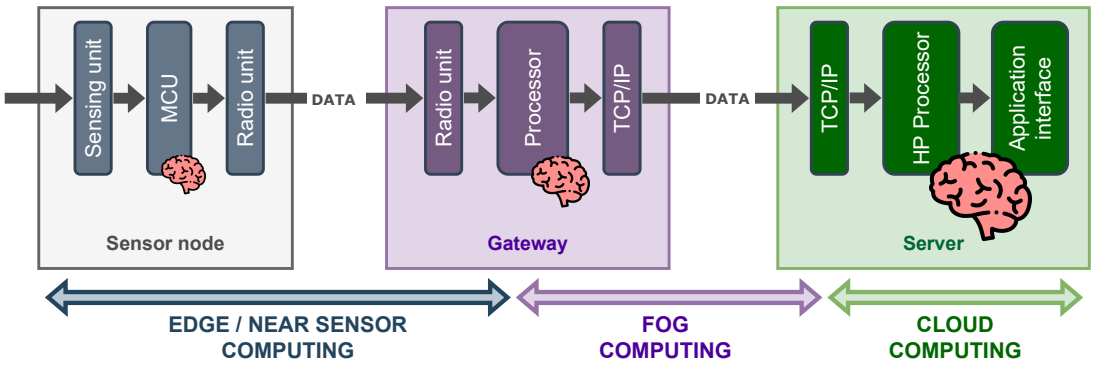


Figure 2.2: *Computing in IoT network*

For a given service, the data processing can be done at different levels (see Figure 2.2). The most common approach is cloud computing. The raw data collected by the sensor nodes are directly routed through the gateway to the server which processes them. This approach allows the perception layer devices to embed very simple software performing simple tasks with no or little data computing and storage. The workload of end-nodes and gateways is minimal, indeed, all the data processing is done by the servers of the service layer. Service layer devices can be subject to a heavy workload in order to process the flow of data, which is more or less important depending on the application (*e.g.*, Google API for speech-to-text conversion in the cloud [38]). However, the cloud computing approach has several drawbacks. First, this can lead to a large amount of data being communicated by all nodes, resulting in very high traffic on the network. This can be problematic in the case of radio protocols with low throughput and where the collision probability becomes too high. Second, a high amount of energy is involved in data transport. This can be very problematic for devices where energy resources are very limited (*e.g.*, sensor nodes or gateways). Finally, this approach poses security issues as it often leads to the circulation of private data over the network.

In recent years, the concepts of fog computing and edge computing have emerged as an alternative to cloud computing [39, 40, 19]. Fog computing is based on the distribution of computing in devices responsible for data routing (*e.g.*, gateways and routers). Edge computing is also a distributed computing paradigm that brings computation in the peripheral layers (perception and network layers). The distribution of storage and processing across the IoT network allows the communication of more meaningful information. Thus these paradigms minimise data exchange, improving the energy efficiency of network use, management, and security, as well as data storage. To that end, many recent works target the integration of machine learning as close

as possible to the data source [41, 42]. Furthermore, with the emergence of Tiny Machine Learning (TinyML), it becomes possible to implement smart applications directly in end-nodes [43]. TinyML is based on lightweight libraries, including ML models designed to run on resource-constrained devices.

To illustrate the difference between traditional cloud computing and edge computing, we take an intrusion detection application as an example. In this case, we assume an application that warns the user when a movement is detected and a person is present at a given location. For this purpose, end-nodes equipped with motion sensors and cameras are deployed in the place to be monitored. These cameras are connected to a WiFi router capable of transferring data to the server. The server is responsible for notifying the user by email in case of intrusion detection.

(a) *Cloud computing*

(b) *Edge computing*

Figure 2.3: *Intrusion detection application, cloud computing versus edge computing approach*

Figure 2.3a describes the traditional edge computing approach. When a motion is detected, the node captures an image and sends it to the gateway, which then transmits it to the server, where it is analysed to detect if a person is present on it or not. Finally, the server can notify the user if an intrusion has been detected. The problem with this approach is that raw images representing several kB each are circulating on the network: indeed, each time a movement is detected, an image is transmitted on the network. This solution induces a lot of energy dedicated to the transport and storage of these data. In addition, the circulation of a raw image on the network raises obvious security concerns.

In contrast, Figure 2.3b describes a possible edge computing approach for the same application. The main difference is that the image is analysed directly at the end-node level. In case of detection, the alert is sent from the node to the gateway and then to the server, which notifies the user. This approach reduces communication on the network because communication is only required when the detection of a person occurs and not every time a movement is detected. Much fewer data are exchanged on the network, as a simple alarm replaces sending a complete image. This approach allows for reducing the energy for the transport and storage of images in the network and service layer. Another advantage is also that it reduces the circulation of private data on the network

Table 2.1: *Energy over network vs workload*

	Service	Network	Perception
Energy availability	High	Med	Low
Computing power availability	High	Low	Low
Computing power trend	↘	↗	↗

and, therefore, greatly improves security. However, the main drawback of this solution is that increasing the complexity of end-nodes tasks increases their processing and memory requirements, which enormously challenges the design of these constrained devices. Thus, the edge computing paradigm applies new constraints on end-nodes, particularly on the energy management of these devices [44].

2.1.3 Energy efficiency of sensor nodes

Edge computing aims at the distribution of intelligence across the network to increase its efficiency and security, which in turn increases the computing and storage requirements of the peripheral layers. As discussed in Section 2.1.1 and shown in Table 2.1, energy availability is not equally distributed across layers. The service and network layer devices are generally powered directly by the electrical network. The continuity of service on these devices is not affected by their energy consumption as they rely on stable power sources. Concerning the perception layer devices, sensor nodes do not have a continuous energy source. They must necessarily use one or more energy accumulators (*e.g.*, battery or super-capacitor) and sometimes one or more energy harvesting systems. These solutions are generally capable of storing a limited amount of energy. However, for some applications, the increasing size constraints on such end-nodes also imply the use of small energy harvesting and storage systems, further increasing energy constraints. To summarise, edge computing requires increasing computing and storage in edge devices which are the ones with the highest energy constraints.

In this context, the most important metric for designing such computing devices is energy efficiency. Energy efficiency is the energy required to perform a task, and it is usually measured in performance[1] per watt. A higher energy efficiency implies that the computing system is able to perform more operations using the same amount of energy or perform the same operations using less energy. As said previously, sensor nodes embed MCU for computation and storage. These small devices are low-cost, embed storage and consume little energy. Even if they generally offer weak computing performances, their low power consumption makes them very energy-efficient.

Figure 2.4 depicts general qualitative differences between MCUs and other process-

[1]Performance is often measured in Instructions per second (IPS) [45] or floating point operations per second (FLOPS) [46]

ing units such as Microprocessor Units (MPUs), Graphics Processing Units (GPUs) or Field Programmable Gate Arrays (FPGAs) according to their consumption and computing performances. All points in the diagonal lines have the same energy efficiency. The red bar in the graph represents the maximum main unit power that can be considered for an end-node due to its high energy constraints. This level depends on the application but is typically in the order of a few mW. MPU and GPU offer very high computing performances but are not used for end-nodes because they consume too much power (hundreds of W). Moreover, unlike MCUs, they do not integrate storage. It is therefore required to complete them with additional memories, such as DDR DIMMs and SSD drives, resulting in higher energy consumption. FPGAs embed reprogrammable logic allowing them to provide good computing performances. In addition, their high flexibility allows them to be perfectly adapted to any application, resulting in higher energy efficiency than MPU. However, these circuits are costly, and most consume too much energy to be integrated into ultra-low-power devices such as sensor nodes.

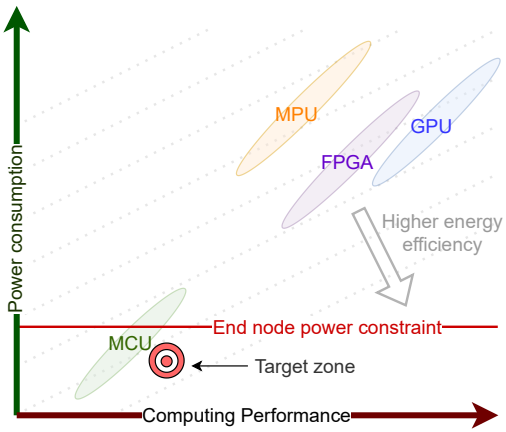


Figure 2.4: *Processing unit energy efficiency*

One of the main challenges in edge computing is the increase in energy efficiency required by the nodes to accommodate more local processing while complying with their growing energy constraints. In Figure 2.4, this results in moving existing MCU to the right while keeping them below the horizontal red bar (denoted as the target zone). In order to reach this area, two steps are necessary:

- It is necessary to increase computing performance. This results in a horizontal right shift in the figure. To do this, there are different techniques, such as the addition of more computing units for parallelisation or the addition of dedicated accelerators.
- It is also necessary to reduce the average energy per operation, thus increasing energy efficiency. This results in a diagonal shift towards the lower right corner of the figure. For this purpose, there are several energy-saving strategies, such as power gating, clock gating or frequency scaling.

2.2 Sensor node application execution

Sensor nodes represent a small fraction of the energy consumed by the entire network as their instantaneous power consumption is generally a few mW compared to hundreds of W for a server. However, the improvement of their energy efficiency is very important in the context of edge computing, as shown in Section 2.1. Indeed, due to their high energy constraints, an increase in their workload without an increase in their energy efficiency results in a major decrease in their autonomy and can therefore have an effect on the service. This is why in our work, we focus on these energy-constrained devices. In this section, we first provide a description of the main tasks performed by sensor node application software. Then, we explain how these tasks are scheduled during application execution.

2.2.1 Main tasks description

As mentioned earlier, the application software is embedded and executed in sensor nodes by the MCU, which acts as the conductor. We identify the three main tasks of sensor nodes as the sensing task (S), the processing task (P) and the communication task (C) (see Figure 2.5).

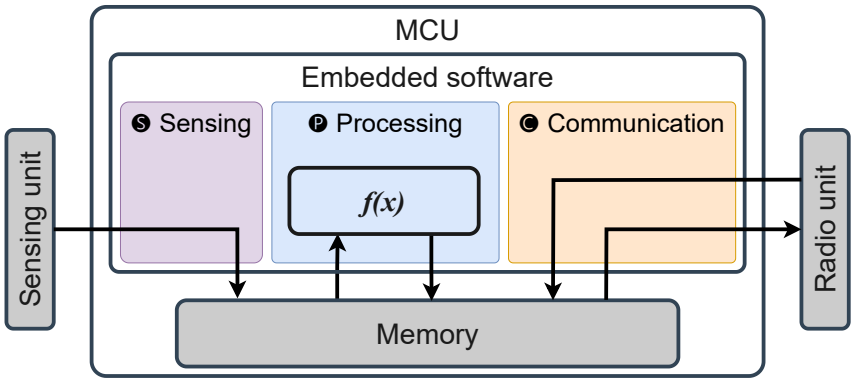


Figure 2.5: *Main tasks of sensor nodes*

The **sensing task** (S) consists of collecting and storing data extracted from the environment through the sensing unit. We can characterise the sensing task with a quantity of data generated and stored over a certain time. The **processing task** (P) can be seen as a function with input and output data. Three different processing cases can be distinguished, depending on the size of the input and output data. First case, the input data size is larger than output data size (*e.g.*, analysis or compression). Second, input and output data sizes are identical (*e.g.*, pre-processing or encryption). Finally, in some rare case, the output data size can be larger than the input data size. In all these cases, the processing task can be characterised by a duration and a data ratio which corre-

sponds to the size of the output data over the size of the input data. In the first case, the data ratio is below 1; in the second case, the data ratio is equal to 1; and finally, in the last case, the data ratio is above 1. Concerning the **communication task** C, the node can send and receive data from the radio unit. The reception may be due to a re-configuration of the node by the application, for example, in the case of an Over The Air (OTA) update [47]. The node can also receive data from other nodes in the case of mesh topology [48], or for the required firmware/software updates. However, in most applications, the node will be mainly transmitting data, so we are mainly interested in data transmission in the communication task. Thus, in the same way, as for the sensing task, we can characterise the communication task with a quantity of data sent over a certain time.

2.2.2 Tasks scheduling

For many applications, end-nodes operate intermittently. Indeed, sensor nodes perform sensing, processing and communication, but these tasks are only executed periodically or following a trigger event, meaning that the device remains inactive for the rest of the time. Thus, sensor nodes alternate between active and inactive states. In this way, software applications are programmed in the normally-off computing paradigm. The system is programmed to be inactive by default, waiting for a specific event to wake up and perform some operations before going back to the inactive state. Energy savings are usually achieved by low duty-cycling the system, meaning that the active phases represent a very small amount of time compared to the inactive phases. Indeed, thanks to power gating and clock gating mechanisms, the power consumption of the whole node is reduced to a minimum during the long inactive phases [49]. When an event is detected, the system wakes up to perform the required tasks and then returns to standby mode.

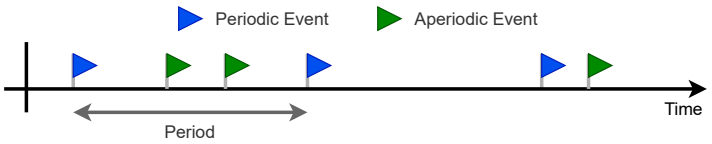


Figure 2.6: *Periodic and aperiodic events*

As depicted in Figure 2.6, these events can be periodic or aperiodic. In the case of periodic events, they can be generated by a dedicated component such as a Real Time Clock (RTC) or a timer. In the case of asynchronous events, it is not possible to predict their occurrence. These events can be generated by sensors thanks to interrupt generators that they can embed, as it is the case in inertial measurement units for the detection of movement or fall [50]. These events can also be generated by certain radio modules [51]. Whatever the type of wake-up and the component that generates it, the

MCU usually integrates a component capable of waking it up following an internal or external interruption (*e.g.*, Wake Up Controller).

As illustrated in Figure 2.7, we can model these applications as a simple alternation between two states, the active and the inactive states. When talking about intermittent computing, the key parameter of these applications is the activity rate. During a certain duration, the activity rate corresponds to the total time spent by the system in active mode (T_{act}) over the total duration.

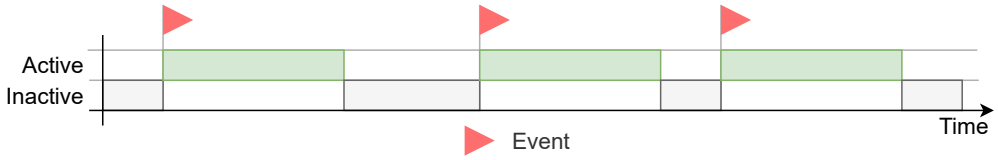


Figure 2.7: *Intermittent task execution*

In general, during each activity phase, the node performs all or a subset of the previously defined tasks. Indeed, in some cases, there is no processing and/or communication required (*e.g.*, raw sensing data communication or processing-dependent communication). Figure 2.8 illustrates the typical end-node task scheduling after each event. As each task represents a certain time, this model contains three new temporal parameters T_{sens}, T_{proc} and T_{com} with:

$$T_{act} = T_{sens} + T_{proc} + T_{com} \tag{2.1}$$

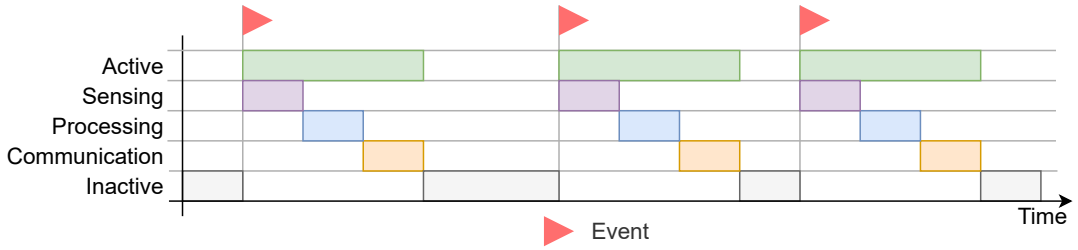


Figure 2.8: *Task organisation*

To illustrate the proposed model, we take the example of two sensor nodes developed by Ineo-Sense: the ESG-Logger-Hygro [34] and the TRK-Slim [52]. The ESG-Logger-Hygro is a good example of a system based on periodical events. The system is based on two events with different periods. A first periodic event wakes up the system to collect temperature and humidity through a sensing unit, thus the system performs a sensing task S. A second periodic event with a different period wakes up the system to send a report on the last collected data. Then, in this case, it performs a processing task and a communication task P C. The TRK-Slim is a good example of a sensor node based on aperiodic events. This sensor node is dedicated to isolated worker protection. The system is able to communicate an alarm in case of button presses or in case

of no motion detection (dead man detection). In this case, the aperiodic events are the button press and an interrupt generated by the inertial measurement unit. Following one of these events, the system only performs a communication task C.

2.3 Sensor node architecture

In the previous section, we described the main tasks performed by a typical sensor node and how they are scheduled. In this section, we describe in detail a typical wireless sensor node architecture with a particular focus on its three major components: (1) the MCU, (2) the sensing unit and (3) the radio unit.

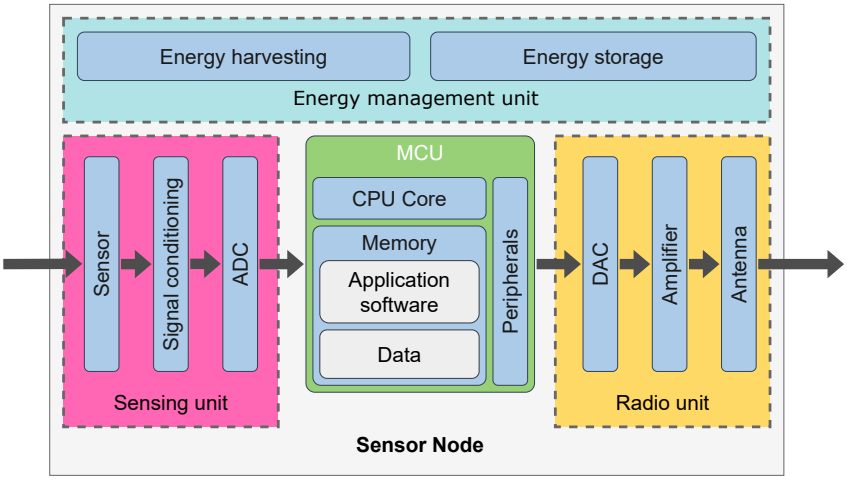


Figure 2.9: *Wireless sensor node architecture*

The main role of a sensor node is to collect data from the environment. Figure 2.9 illustrates the typical architecture of an energy-autonomous wireless sensor node [53]. These devices are powered by an energy storage system such as a battery or supercapacity. These energy accumulators can have a wide range of specifications: their nominal voltage, the amount of charge (energy), the maximum charge and discharge current, their internal impedance, their size, their degradation depending on their use, their inherent energy loss, and many other metrics. The node also can embed one or more energy harvesting systems (*e.g.*, solar, thermal, RF or piezoelectric). Energy harvesting systems also include a wide range of specifications: their nominal voltage, average power generated, maximum output current or even stability. The requirements for energy storage and generation and therefore the specification of the systems responsible for it depend directly on the needs of the application and the energy efficiency of the node. The energy is directly distributed by an Energy Management Unit (EMU) responsible for regulating and supplying energy to all the components of the node. If an energy harvesting system is used, the EMU is responsible for the transfer of the harvested energy into the energy accumulator.

As said previously, sensor nodes embed an MCU responsible for the storage and execution of the whole application, the computing and the data storage in embedded memories. These MCU also embed General Purpose I/O (GPIO) and communication peripherals (UART, SPI, I2C) to control and exchange data with other units. More details on the architecture of MCUs will be given in Chapter 3 and Section 4.3.

The sensing unit can be divided into three stages. The sensor inputs a certain physical quantity, property or condition (such as acceleration, resistance, pressure, temperature or light) and outputs an electrical signal in response. This response comes as a voltage, current or charge that can be further characterised in terms of amplitude, polarity, frequency or phase. For example, photo-transistors are light-sensitive. The current flowing between emitter and collector depends on the level of light it receives. Thus the output signal is a current amplitude that depends on the light received by the photo-transistor. These signals cannot be interpreted directly by the processing unit. Also, sensors can produce low-level analog signals that need to be amplified, filtered and/or shifted before digital conversion, which can be performed by a signal conditioning circuit [54]. Finally, this analog output needs to be translated into a digital signal by the ADC. Sensors with different interfaces are available:

- Raw sensors, which must be used in a signal conditioning circuit to produce a voltage signal which is then used as input of an ADC. The ADC can be integrated into the processing unit or external to it. In this case, its output can be read through protocols such as I2C or SPI.
- Analog sensors generally embed the signal conditioning circuit and can be directly interfaced through an ADC.
- Digital sensors embed both the signal conditioning circuit and ADC.

Furthermore, in some cases, there are different sensor technologies for the same physical quantity. For example, microphones are used to capture ambient sound. There are a large number of acoustic sensors, the most commonly used in embedded systems being the Micro-Electro-Mechanical System (MEMS) microphones or the Electret Condenser Microphones (ECM).

Each value read from the ADC is called a sample. Sample size S_{size} may vary from a few bits for basic ADC to hundreds of kB for richer sensors. The larger the sample size, the higher is the resolution and the closer is the value to reality. We can then define T_{sens} as the time required to read one sample and store it in memory. It depends on the sample size and on the communication speed between the sensor or the ADC and the processing unit. Samples can be obtained periodically or sporadically (based on event detection) depending on the application. If the sampling is periodic, a sampling frequency f_s can be defined. Furthermore, the sampling frequency depends on the scales, the context, the application needs and the environment. For example, high

sampling frequency temperature sensors can be used in processors, and low sampling frequency sensors can be used to measure temperature in an environment where it depends only on the weather (outdoor). The direct proportion between sample size and sample frequency is called throughput. The amount of data generated by the sensing unit corresponds directly to its throughput. For example, a microphone with a two bytes sample size and a 16kHz sampling frequency generates 32kB/s. In comparison, an Inertial Measurement Unit (IMU) can produce a 24 bytes sample with a 1kHz sampling frequency generating 24kB/s. Finally, a camera sensor with a 1280 x 720 matrix of RGB pixels generates 2.7MB samples. With a sampling rate of 20Hz, its generates 54MB/s. Generally, the physical quantity to measure, the required resolution and the minimum sampling frequency are enforced by the application. It remains for the designer to choose the sensing unit and to set up the communication with the MCU to obtain the most energy-efficient solution.

Table 2.2: *Comparison of wireless protocols for IoT [55, 56, 57, 58, 59]*

Feature	ZigBee	Wi-Fi	Bluetooth	802.15.4	Sigfox	LoRa	LTE-M1	NB-IoT
Frequency of Operation	868MHz 915MHz 2.4GHz	2.4GHz 5.8GHz	2.4GHz	868MHz 915MHz 2.4GHz	868MHz 915MHz	915MHz	Cellular Band	Cellular Band
Bit Rate	20, 40, 250Kbps	11Mbps up to 7Gbps	1Mbps up to 3Mbps	20, 40, 250Kbps	600bps	50Kbps	1Mbps	250Kbps
Modulation Method	DSSS with DBPSK or OQPSK	DSSS, OFDM	FHSS with GFSK,π/4-DQPSK, or 8DPSK	DSSS with DBPSK or OQPSK	UNB with GFSK, or DBPSK	SS Chirp	OFDM	OFDM
Range (m)	10-100+	200	10-100	10-100	1000+	1000+	1000+	1000+
Power Level	0.5, 1 (typically), 100mW	100mW-1000mW+	100, 2.5, 1, and 10mW for class 1, 2, 3, and BT smart respectively	1mW	10uW-100mW	25mW+	100mW	200mW+
Application	Consumer, Industrial	Consumer, Commercial, Industrial	Consumer	Consumer Industrial	Industrial	Industrial	Industrial	Consumer Industrial
Key Characteristics	Uses IEEE 802.15.4 for PHY and MAC, Short range, low data rate, Low power consumption, License-free spectrum	Short range, high data rate, license-free spectrum	Short range, Low power consumption, Single chip implementation, license-free spectrum	Short range, low data rate, Low power consumption, License-free spectrum, represents the basis for many advanced wireless standards	Long Range, Low data rate, Low power consumption	Long Range, Low data rate, Low power consumption	Long Range, Low power consumption	Long Range, Low data rate, Low power consumption
Network Configuration	Point to Point, Star Topology, Mesh Topology	Point to Point, Star Topology	Point to Point, Piconet with up to 7 Nodes, Star Topology	Point to Point, Star Topology	Star Topology	Star Topology	Star Topology	Star Topology

Regarding communication, due to its range, data rates, reliability, and energy expenditure, RF-based communication is the best fit for most ultra-low power applications [60]. Even if the radio unit can also be divided into three stages, most of the time, the radio unit consists of a single device or module called a radio transceiver connected to an antenna. These components connect the node to the network using radio waves modulation to send and receive data. The medium and protocol impose constraints on the transmission, such as operational range and speed. There are many different technologies using different frequency bands, modulation methods, and pro-

tocols. Table 2.2 summarises and compares the different radio technologies used in IoT. Amongst the most used ones, we find:

- **Wi-fi** offers a data transfer rate that can reach 300Mbps. One of the major advantages of Wi-fi is also its coverage that can reach more than 500m outdoors [61]. On the other hand, the power consumption of Wi-fi makes it highly inefficient for ultra-low power sensor nodes. This is due to some characteristics of the protocol, such as active network scanning. Also, periodic wake-ups during power saving mode to send check-ups to the network have a considerable impact on power consumption [62].
- **LPWAN** is a class of wireless IoT communication standards. They offer solutions with large coverage areas, low transmission data rates with small packet data sizes and low power consumption [63]. SigFox, for example, is a proprietary protocol that uses Ultra Narrow Band (UNB) modulation with Differential Binary Phase-Shift Keying (DBPSK) [64]. The Long Range (LoRa) protocol operates in an unlicensed band below 1 GHz for long-range communication. Six Spreading Factors (SF) can be used in a LoRa transmission. The SF is a mechanism that adapts the power and data rate of the transmission. Data rates are between 300bps and 50kbps, depending upon the spreading factor. LoRa enables long-range transmissions up to 30 km with low power consumption [65]. The protocol also does not require constant communication from the end nodes in order to remain in the network. This allows the radio to be turned off when it is not being utilised, saving considerable amounts of power.
- **Zigbee** protocol is aimed at high reliability, cost-effectiveness and energy efficiency. In perspective, ZigBee technology is designed to run for more than five years on two AA alkaline batteries [66]. This is due to the fact that a node on a ZigBee network does not need to send packages constantly to remain in the network. Using around 1mW or less, transmissions can reach up to about 100 meters outdoors [67, 68, 69]. On the other hand, ZigBee can only reach data rates lower than 250kbps, sitting typically at 25kbps in most cases.
- **Bluetooth** is designed for short-range operations. The technology is available in two forms: Basic Rate (BR) and Low Energy (BLE). The LE systems target lower power consumption devices with lower complexity and lower cost than BR. Data rates of 2.1 Mb/s can be achieved in BR, while BLE can only reach speeds of about 2Mb/s [70]. The range of operation on BLE depends on the implemented chip, the antenna and other parameters such as path loss [71]. For example, an STMicroelectronics BLE IC can reach a communication distance of about 33m.

2.4 Sensor node energy modelling

In the previous sections, we describe the architecture of the sensor nodes and how their tasks are scheduled. In this section, we build a parametric energy model of the end nodes and their different components. Based on the node architecture description provided in Section 2.3 and the software application execution model defined in Section 2.2, we model the energy of the different components of the node during its different application phases. To calculate the consumption and, thus, the autonomy of a sensor node for a given application, we first use a parametric description of the energy source. In our case, we will only take the case of a system with a battery which is simply characterised by an initial charge.

For energy consumption evaluation, we use the temporal behaviour of sensor node task execution described in Section 2.2. This model also allows for estimating the consumption of all the components of the node. As all components can be used in different ways and have different consumption levels during each task, we start by characterising the node consumption during each task: P_{sens}, P_{proc}, P_{com}, P_{inact} (see Figure 2.10).

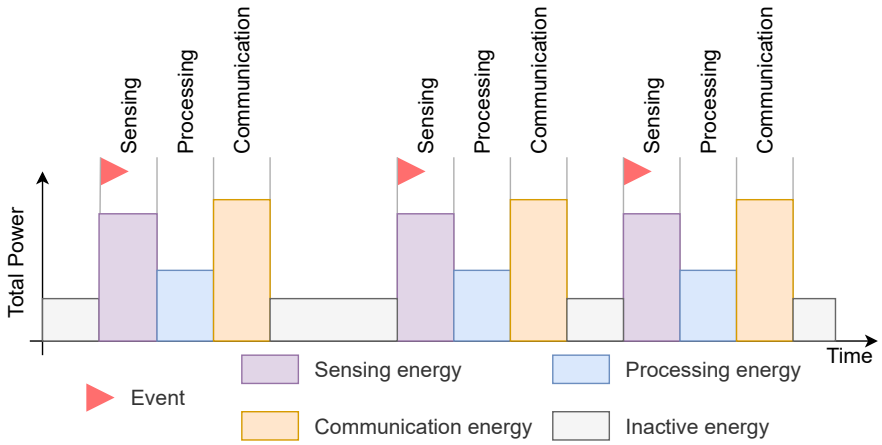


Figure 2.10: *High level energy model*

Thus, we can express the total energy for each task as:

$$\begin{aligned} E_{sens} &= T_{sens} \times P_{sens} \\ E_{proc} &= T_{proc} \times P_{proc} \\ E_{com} &= T_{com} \times P_{com} \\ E_{inact} &= T_{inact} \times P_{inact}. \end{aligned} \tag{2.2}$$

Furthermore, we can define the total energy consumed by the node as:

$$E_{total} = E_{sens} + E_{proc} + E_{com} + E_{inact}. \tag{2.3}$$

To calculate the node consumption during each task, we define the consumption of each embedded component for each task. Thus we build a model for each component [72]. We can define for each component several operating modes resulting in different power consumption. Then we can define for each task the mode in which the component is. We assume a simple use case where each component has two operating modes; they can be in use or not in use. For each component, we consider two possible consumptions P_{act} and P_{stdb}. If we take the temporal model of application execution, we can define the operating mode of each component during each phase as illustrated in Figure 2.11. We make the following assumptions:

- Sensing unit: During the sensing task, we consider the sensing unit as active as the MCU communicates with it to retrieve some data from its environment. During the other phases, the sensing unit is considered inactive. It can either be powered off or perform discrete sensing with possible data buffering if the ADC allows it.
- Radio unit: During the communication task, we consider the sensing unit active. We only consider the transmission operating mode during this phase as it represents the major part of the energy consumed by this unit. During the other phases, the radio unit is considered inactive (powered off).
- MCU: During the sensing, processing and communication, the MCU execute application code. Thus during these tasks, we consider the MCU as active. During inactive phases, we consider the MCU as inactive and only waiting for the next event.

Once we know T_{sens}, T_{proc}, T_{com} and T_{inact}, we calculate the energy corresponding to each task E_{sens}, E_{proc}, E_{com} and E_{inact}, and then, the total consumption of the node E_{total}. However, it is also possible to define the consumption of each component as:

$$\begin{aligned} E_{SEN} &= (T_{sens} \times (P_{act})_{SEN}) + ((T_{com} + T_{proc} + T_{inact}) \times (P_{stdb})_{SEN}) \\ E_{RAD} &= (T_{com} \times (P_{act})_{RAD}) + ((T_{sens} + T_{proc} + T_{inact}) \times (P_{stdb})_{RAD}) \\ E_{MCU} &= ((T_{sens} + T_{proc} + T_{com}) \times (P_{act})_{MCU}) + (T_{inact} \times (P_{stdb})_{MCU}). \end{aligned} \tag{2.4}$$

Thus, we can identify the most consuming component in the node.

It is possible to define sub-tasks and to define many different operating modes for each component to increase the precision of the model. For example, for the radio, it is possible to create a more accurate model incorporating many different operating modes, such as different transmission powers and identify several sub-tasks [73]. For now, this model with three tasks and two operating modes per component is sufficient and will be used in the rest of our work. Indeed, we plan to evaluate the consumption of the node and the sharing between its different components to identify bottlenecks.

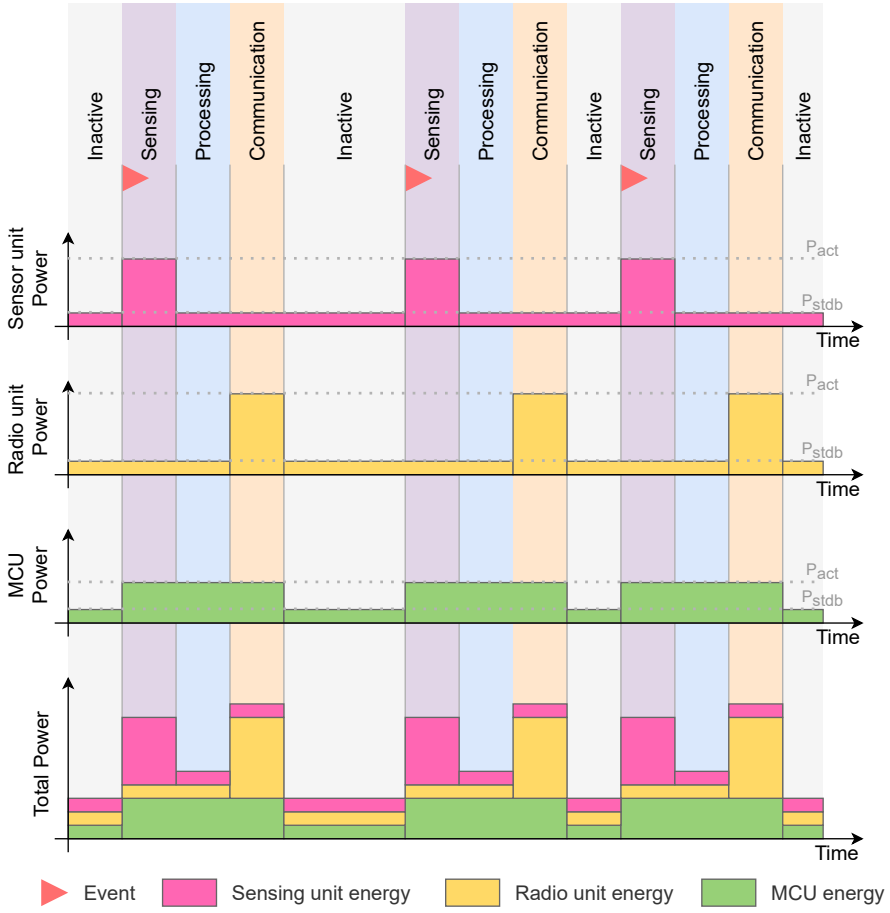


Figure 2.11: *Detailed energy model*

2.5 Summary

In this chapter, we defined IoT networks and showed how edge device workload has evolved over the last few years. We then described the structure of a sensor node and how it schedules its tasks before proposing a parametric energy model for the evaluation of node energy. Based on a general model of the execution of the IoT application in end nodes, we propose a set of temporal parameters that can be combined with simple power models to evaluate their energy and its distribution among the components it embeds. As we plan to explore the impact of the addition of data processing and storage in the end node and, therefore, in the MCU, in the next chapter, we dive into its architecture and, more specifically, into its memory organisation for data processing and storage.

III

MCU memory for normally-off computing

Contents

In the previous chapter, we presented the IoT end node architecture and application context. The MCU memory is becoming a critical energy bottleneck that limits the transition to edge computing. Our goal in this chapter is to understand how to use emerging non-volatile memory technologies to reduce energy consumption in edge computing end nodes. To that end, we describe the MCU memory architecture and technologies, we survey energy-saving strategies based on emerging non-volatile memories, and we propose a generic MCU energy model that allows the exploration of energy-saving strategies based on non-volatile memories.

3.1 Memory architecture and technologies

In this section, we review the memory hierarchy and technologies that are generally used in MCUs. Then we analyse the potential benefits and drawbacks of emerging non-volatile technologies.

3.1.1 MCU memory hierarchy

As shown in Figure 3.1a, the architecture of a microcontroller usually includes a CPU core, memories (used to contain the program, static data, and dynamic data) and peripherals. The logical state of a given digital system, *e.g.* MCU, may be defined as a set of bits (S):

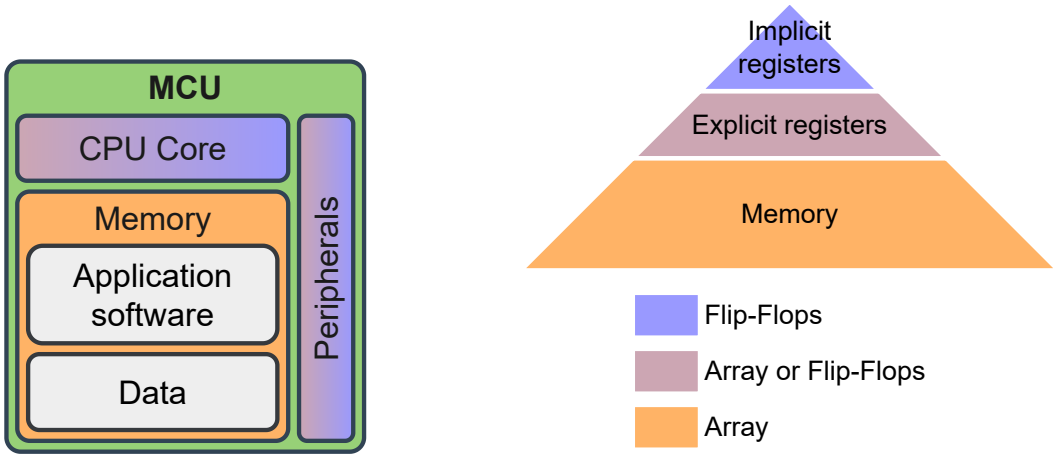
$$\begin{aligned} &S = \{s_1, s_2, ..., s_n\} s_i \in \mathbb{B} \\ &S = S_{regImp} \cup S_{regFile} \cup S_{regPeriph} \cup S_{inst} \cup S_{data}. \end{aligned} \tag{3.1}$$

This set is also equal to the union of several sub-sets distributed over different types of memory elements (Figure 3.1b):

- Memory ($S_{inst} \cup S_{data}$): it is usually divided into program and data memory. The first one contains instructions and constant data, while the data memory is used to store non-constant data. Some MCU use small non-volatile on-chip memories to store the booting program and small portions of the program memory. Alternatively, a more modern trend is to use faster serial interfaces and move the program code from the off-chip Flash memory onto on-chip SRAM memory (*i.e.*, code-shadowing). Both program and data memory are typically volatile.
- Architecture registers ($S_{regFile} \cup S_{regPeriph}$): They are explicit registers that contain

temporary or permanent data (configuration) and are addressable, *i.e.*, accessible from the software. They are present in the core as a register file, which contains all the general-purpose registers of the core (*e.g.*, stack pointer, temporary registers, saved registers). They are also present in the peripherals of the system (peripherals registers). Either flip-flops or memory arrays can be used to implement these registers.

- Microarchitecture registers (S_{regImp}): They are implicit registers, *i.e.*, not accessible from the software. These registers are built at the circuit level by flip-flops directly into the logic (*e.g.*, pipeline registers).



(a) *MCU architecture* **(b)** *MCU memory hierarchy*

Figure 3.1: *The execution context of a microcontroller is distributed across implicit and explicit registers and memory*

3.1.2 Mainstream memory technologies

The technologies used for logical state storage are a decisive factor in the overall energy consumption and performance of a microcontroller. The microarchitecture of a microcontroller combines volatile and non-volatile memory technologies.

Volatile memories are either Dynamic Random Access Memories (DRAM) or Static Random Access Memories (SRAM). DRAM uses tiny capacitors and transistors based on metal-oxide-semiconductor (MOS) technology to store data. In DRAM, the capacitor is used for storing the data. DRAM uses capacitors and very few transistors, enabling high density and allowing to build high capacity memories. The disadvantage of this technology is that the charge in the capacitors gradually leaks out, leading to data loss. These memories need to be refreshed frequently (*e.g.*, every 64ms) to avoid this. DRAM is mostly used for high-capacity off-chip computers' main memory. Due to the required periodic refreshes and high voltage, DRAM technology is not suitable

for MCU. Unlike DRAM, SRAM uses transistors and latches to store data. SRAM uses basic memory cells with built-in feedback mechanisms that retain the stored value as long as the device is powered. SRAM cells are bigger, resulting in lower density than DRAM. This technology allows the design of little memories with very fast accesses where no refreshes are required. SRAMs are mainly used as on-chip memory for CPU cache memories or MCU memories. Indeed, SRAMs are widely used in microcontrollers due to their low voltage and retention capabilities, which allows for saving data with a limited leakage power.

Concerning non-volatile memory, Flash-based technologies are the most mature and widely used in commercial microcontrollers, offering non-volatility and high density. Electrically Erasable Programmable Read-only Memory (EEPROM) are widely used in MCU architecture for storage. EEPROMs memories are organised as arrays of floating-gate transistors. These memories can be programmed and erased in-circuit, thanks to dedicated programming signals. NAND-type Flash memories are a form of EEPROM that write data at a multiple-byte page level and erase data at a multiple-page block level thanks to high electrical voltage. NAND-type Flash memories allow very high density but very slow read and write speeds. Besides, they require relatively high voltages making them not suitable for MCU. In NOR-type Flash memories, the connections of the individual memory cells and interface provided for reading and writing the memory are different. Indeed, it is possible to write data at a byte level and erase data at a block level. NOR type Flash memories allow faster writing and require lower voltage at the cost of lower density and slower read speed than NAND Flash. Furthermore, this technology allows random access for reading, while NAND allows only page access. For this reason, NOR-type Flash memory is widely used in commercial MCU for storage. However, Flash-based technology offers low endurance (up to 10^5 cycles) and very low write and read speeds (mainly due to complex management caused by page erase and word programming); this is why these technologies cannot replace SRAM as MCU main memory.

3.1.3 Emerging non-volatile memory technologies

Emerging NVM (ENVM) technologies present very interesting characteristics. These technologies are non-volatile and exhibit more competitive write currents and latencies than Flash. They also offer much higher densities than SRAM and good integration compatibility with CMOS technology ([74, 75, 76]). These technologies can be used to build NV-RAM. The resulting memories offer good opportunities in the context of sensor nodes operating in the normally-off computing paradigm.

Ferroelectric RAM (FRAM) uses a ferroelectric crystal layer between capacitor plates

instead of conventional dielectric material to store data and achieve non-volatility. Phase-change memory (PCRAM) stores data in a material through conversion between an amorphous and a crystalline state. Resistive RAM (RRAM) store data in a dielectric material where a conductive filament is created or broken depending on the applied current, changing its resistance. Finally, Magnetoresistive RAM (MRAM) store data in the magnetic domains thanks to a Magnetic Tunnel Junction (MTJ). The MTJ is formed from two ferromagnetic layers (reference layer and free layer), separated by a thin insulating layer. The magnetic orientation of the reference layer is fixed, while the free layer's magnetic orientation can be changed. Pulsing current in conductive lines above and below a MTJ allows switching the magnetic orientation of the free layer. Thus, the cell can be in two states: parallel or antiparallel. The parallel state results in a low resistance of the MTJ, while the antiparallel state results in high resistance of the MTJ. It is therefore sufficient to apply a current to the MTJ to read the stored information. There are several more or less mature technologies based on magnetoresistivity, such as Thermally Assisted MRAM (TAS), Spin-Transfer Torque MRAM (STT) or Spin-Orbit Torque MRAM (SOT).

Table 3.1 compares the characteristics of the previously presented memory technologies. On the left side are grouped mainstream memories, considered as mature technologies available on the market in the form of external memories but also integrated into most microcontrollers. On the right side of the table are gathered emerging NVM technologies, less mature but offering promising properties.

Table 3.1: *Emerging memory technologies characteristics*

	MAINSTREAM MEMORIES				EMERGING MEMORIES			
			FLASH					
	SRAM [74, 75, 76]	**DRAM** [75, 76]	**NOR** [76]	**NAND** [74, 75, 76]	**FRAM** [74, 75, 76]	**PCRAM** [74, 75, 76]	**RRAM** [74, 75, 76]	**STT-MRAM** [74, 75, 76]
Cell size (SLC)*	120 - 200 F^2	6 - 100 F^2	10 - 100 F^2	4 - 6 F^2	6 - 40 F^2	4 - 30 F^2	4 - 12 F^2	6 - 50 F^2
Voltage	0.6 ~1.1 V	<1 V	>10 V	>10 V	~1.8 V	1.5 ~1.8 V	3.3 ~6.5 V	0.8 ~1.8 V
Write energy	~1 fJ/b	~10 fJ/b	~100 pJ/b	~10 fJ/b	~0.1 pJ/b**	~10 pJ/b	~0.1 pJ/b	~0.1 pJ/b
Write time	~0.2 - 2 ns	~10 ns	10 µs - 1 ms	100 µs - 1 ms	50 - 75 ns	20 - 150 ns	~50 ns	3 - 50 ns
Read time	~0.2 - 2 ns	~10 ns	~50 ns	~10 µs	20 - 80 ns	20 - 60 ns	~10 ns	2 - 35 ns
Retention	(volatile)	~64 ms	>10 years	>10 years	>10 years	>10 years	>10 years	>10 years
Endurance	10^{16}	$>10^{15}$	10^4 - 10^5	10^4 - 10^5	10^{14} - 10^{15}	10^8 - 10^9	10^7 - 10^{11}	10^{12} - 10^{15}
Byte adressability	yes	yes	no	no	yes	yes	yes	yes

* SLC: Single-Level Cell; F denotes the smallest lithographic dimension in a given technology node.
** Capacitive switching (no static current required).

FRAM is the most mature technology (available in commercial products): it provides low write energy (0.1 pJ/b) and high endurance (up to 10^{15} cycles). PCRAM and RRAM allow higher density but lower endurance than FRAM. Finally, MRAM offers a relatively high endurance at access times similar to those of large SRAM memories. ENVMs allow for good dynamic performance while providing retention without needing a power supply. In applications where the system spends most of its time waiting, these memories may enable significant energy savings.

3.2 Energy saving strategies based on emerging NVM

In this section, different energy saving-strategies based on saving the system state in non-volatile memories are studied. We investigate how emerging memory technologies can be used in different parts of the MCU to build energy-efficient devices for intermittent applications such as those embedded in sensor nodes.

3.2.1 Energy saving strategies

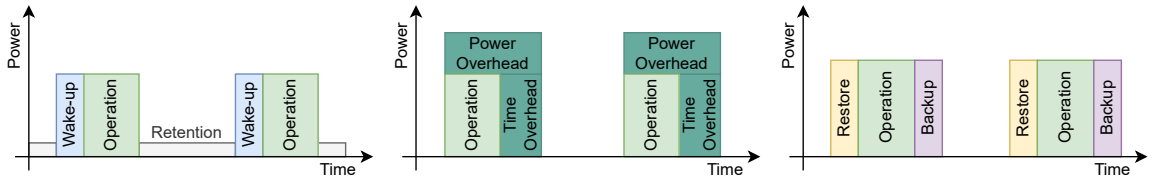
As mentioned in Chapter 2, MCUs often operate intermittently in sensor nodes, alternating active and inactive phases. In the case of low duty-cycled applications, the MCU spends most of its time on standby, waiting for the next event. Due to limited energy availability, power management is a critical aspect in the design of MCU operating in the normally-off computing paradigm. For these standby power critical applications, it is very important to minimise the energy consumed during the standby phases. However, energy-saving techniques targeting standby phases can negatively impact the active phases. Thus, for a technique to be successful, the energy overhead imposed on active phases E_{ovhd} must be sensibly lower than the energy saved during the standby phases E_{saved}. As the standby energy depends on the standby duration, there is a minimum standby duration from which this rule becomes true:

$$\exists T_{inact} \in \mathbb{R}_{+} \text{ such as } E_{ovhd} < E_{saved}. \tag{3.2}$$

For example, we can consider an MCU that wakes up to collect data from a sensor and transmits the data by radio before going back to sleep. During the standby phase, the system is powered down by a tiny external controller that can connect and disconnect the power supply of the MCU (wake-up controller). In this case, the activity overhead is due to the boot process of the microcontroller caused by the power down during the standby phases ($E_{ovhd} = E_{boot}$). If the sleep period becomes too short, the energy cost added by the boot phase will be higher than the energy saved during the standby phase.

There are several options to reduce the energy consumption of the system during the sleep phase. Most commercial MCUs include deep sleep modes to ensure data retention while reducing standby energy consumption [77, 78, 79] (see Figure 3.2a).

When using a retention mode, the activity phase can be divided into a wake-up phase and an operation phase. The transition from a deep sleep state to an active state is not instantaneous and is referred to as the wake-up phase. The operation phase is the



(a) *Volatile strategy* **(b)** *Non-Volatile strategy* **(c)** *Hybrid strategy*

Figure 3.2: *Memory energy saving strategies*

one during which the application code is executed before returning to a sleep state as standby again. In this case, the standby energy E_{stdb} depends on the standby duration T_{inact} and the consumption of the circuit to ensure data retention P_{ret}.

$$\begin{aligned} T_{act} &= T_{wu} + T_{op} \\ E_{ovhd} &= E_{wu} \\ E_{act} &= E_{ovhd} + E_{op} \\ E_{stdb} &= P_{ret} \times T_{inact} \\ E_{MCU} &= E_{act} + E_{stdb}. \end{aligned} \tag{3.3}$$

Thus, these mechanisms still have an impact on the energy consumption of active phases. The duration of the wake-up phase depends on the aggressivity of the retention mode. Indeed, voltage regulators need a certain amount of time before reaching the desired voltage, and some oscillators also require time to stabilise their output signals: the more aggressive a solution is to save energy, the higher the energy and latency penalty during the wake-up phase is. However, in commercial MCUs, the duration and, therefore, the energy of these wake-up phases is often very short and, therefore, negligible compared to the operation phases.

Even lower energy consumption can be achieved in the standby phase by completely disconnecting MCU's components from the main power grid. At the extreme, every component can be turned off, except for a tiny wake-up controller, which is used to turn on the other components if necessary. However, powering off a typical MCU causes the loss of all data stored in registers and volatile memories, which is not acceptable for the application's consistency and progress. A way to avoid the loss of data during the power-down is to replace volatile memories and registers with a non-volatile counterpart. This second category is illustrated in Figure 3.2b and includes all the solutions that make it possible to obtain a purely non-volatile architecture. Such systems combined with a wake-up controller allow almost zero power consumption during standby phases.

$$
\begin{aligned}
R_T &= \frac{(T_{CLK})_{NV}}{(T_{CLK})_V} = \frac{(F_{CLK})_V}{(F_{CLK})_{NV}} \\
R_P &= \frac{(P_{OP})_{NV}}{(P_{OP})_V} \\
T_{act} &= R_T \times T_{op} \\
E_{ovhd} &= (E_{op} \times (R_T \times R_P - 1) \\
E_{act} &= E_{ovhd} + E_{op} \\
E_{stdb} &\simeq 0 \\
E_{MCU} &= E_{act} + E_{stdb}.
\end{aligned}
\tag{3.4}
$$

The time required to execute the operations is multiplied by R_T, which corresponds to the slowdown of the system clock induced by the use of non-volatile memory elements. In the same way, the system power consumption is affected. It can be multiplied by R_P, which is the power increase factor between a standard circuit and its NV counterpart.

A third alternative, illustrated in Figure 3.2c, includes all the hybrid solutions that target context saving, a limited impact on the active phase and near-zero energy consumption during the sleep phase. These solutions combine volatile and non-volatile technologies to build efficient backup and restore mechanisms in order to prevent losing the important data of a system when using aggressive power-saving mechanisms, or when the power source is no longer sufficient enough (*e.g.*, battery-less wireless sensor nodes with a discontinuous source of energy). In these systems, traditional volatile technologies are used during the operation phases. Furthermore, backup and restore procedures are used to transfer data between volatile and non-volatile elements during the transition to standby and active phases. For these hybrid solutions, the energy overhead during active phases E_{ovhd} only depends on the backup and restore energy (E_{bu} and E_{re}).

$$
\begin{aligned}
T_{act} &= T_{re} + T_{op} + T_{bu} \\
E_{ovhd} &= E_{re} + E_{bu} \\
E_{act} &= E_{ovhd} + E_{op} \\
E_{stdb} &\simeq 0 \\
E_{MCU} &= E_{act} + E_{stdb}.
\end{aligned}
\tag{3.5}
$$

3.2.2 Emerging NVM for NV-MCU

In order to save a maximum of energy during long standby phases, it is possible to design a non-volatile MCU capable of near-zero power consumption while inactive. For this purpose, it is possible to replace all memory elements with non-volatile elements or to combine non-volatile technologies with volatile technologies through backup and restore mechanisms.

The purpose of backup and restore mechanisms is to save and reload the execution context to sustain computation progress throughout the standby cycles. Accordingly, a backup should store in a persistent memory all the context data required to ensure the continuity of the application execution upon wake-up. Throughout the execution of an application, this context grows and shrinks. For instance, the volatile context can be reduced to a program counter when the application stores all the relevant memory data and architecture registers onto the off-chip non-volatile memory and waits for all the running instructions to commit (*i.e.*, no relevant state left in microarchitecture registers). However, in most of the execution cycles of an application, the state is distributed over all the volatile memory elements shown in Figure 3.1b.

Regarding memory, S_{inst} is generally already stored in non-volatile memory, while S_{data} is usually stored in volatile memory. It is then necessary before powering down the system to save the contents of the volatile memory to the non-volatile memory and vice versa before restarting the application after powering up the system. This transfer can be a very slow and energy-demanding process, especially because the only non-volatile, erasable and programmable memories available in most of today's low-power microcontrollers are implemented in Flash technology. These memories have a low write speed because they cannot be directly written (page erase, word programming, ~5 µs in the fastest cases [80]) and a limited endurance ($< 10^6$ write/erase cycles) resulting in complicated memory management.

Alternatively, emerging non-volatile memory technologies, such as FRAM and MRAM, can be used to persistently store the relevant data before entering into the standby phase [75]. These memories are faster (around three orders of magnitude), support random access for both read and write operations and have better endurance (up to 10^{15} for FRAM) than Flash as shown in Table 3.1; they can help to reduce the energy cost of backup operations. For microcontrollers that support external memories, commercial off-chip solutions are available (MR4AxxBUYS45 by Everspin, 16 Mb MRAM). However, this solution is less efficient in terms of writing and reading speed because of the limited bandwidth of off-chip communication. Moreover, using several chips duplicates the energy regulation circuits, resulting in a less energy-efficient system. However, these emerging technologies offer good CMOS compatibility and

can easily be integrated on-chip directly as a memory array. There are already several microcontrollers on the market with emerging NV memory arrays, such as the Texas Instrument MSP430, which offers up to 128 kB of on-chip FRAM. There are two possible combinations for NVM utilisation at the memory level:

- Hybrid solutions: On-chip memories are both volatile and non-volatile. Because of their read-only property, S_{inst} are located in non-volatile memories that require much less energy for reading than for writing. Concerning S_{data}, it is located in a volatile memory which has very good dynamic performances. During the backup and restore phases, S_{data} is transferred between volatile and non-volatile memories. Data copy can be either done by software (each word is manually copied by the core) or by hardware (here, the core is not involved, the data transfer is ensured by a dedicated controller such as DMA). The time required for this sequential transfer depends on the amount of data, the word size WS and the time required to read from one memory and write to another.

$$\begin{aligned} T_{bu} &= \frac{S_{data}}{WS} \times ((T_{rd})_{VM} + (T_{wr})_{NVM}) \\ T_{re} &= \frac{S_{data}}{WS} \times ((T_{rd})_{NVM} + (T_{wr})_{VM}). \end{aligned} \tag{3.6}$$

- Non-volatile solutions: All on-chip memory is made non-volatile. Both S_{inst} and S_{data} are stored in non-volatile memory. In this case, no backup nor restore phases are needed, S_{data} is implicitly saved after each memory write.

$$T_{bu} = (T_{wr})_{NVM}. \tag{3.7}$$

In their work, Patrigeon et al. [81] study the impact of different on-chip STT-MRAM memory integration strategies on dynamic consumption:

- S_{inst} implemented with flash and S_{data} implemented with SRAM.
- S_{inst} implemented with STT-MRAM and S_{data} implemented with SRAM.
- Both S_{inst} and S_{data} implemented with STT-MRAM.
- Both S_{inst} and S_{data} implemented with SRAM.

They conclude that the solution offering the best energy efficiency is to use STT-MRAM for S_{inst} and SRAM for S_{data} (energy reduction of 23% compared to a traditional Flash-based approach). However, normally-off applications undergo two phases (activity and sleep phases). As said previously, there is a trade-off between the energy saved by technology in some phases and the overhead associated in other phases. It is therefore important to study the impact of NVM integration on the autonomy of the system.

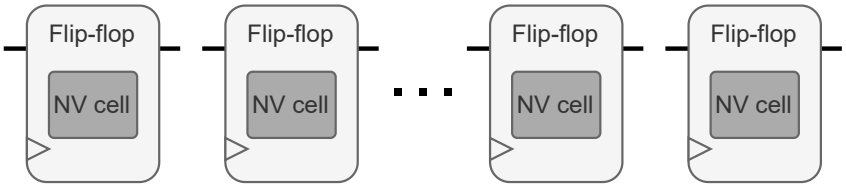
Vater et al. [82] study the autonomy of a system operating in the normally-off com-

puting paradigm and integrating emerging NVM. The authors compare a Flash-based standard solution to different strategies for integrating FRAM, MRAM and RRAM by exploring a hybrid and non-volatile approach for each technology. Three different applications are tested, from a very small application with a very small amount of data to a larger data-intensive application. For the smallest application, on average, a hybrid approach allows a $43\times$ increase in autonomy while a non-volatile approach allows, on average, a $46\times$ increase. As the size of the application (S_{inst}) and the amount of data (S_{data}) increases, the size of the memories also increases, MRAM and FRAM allow up to $1000\times$ increase in the autonomy. They also conclude that RRAM technology cannot be used for S_{data} because of its relatively low endurance (10^7 cycles).

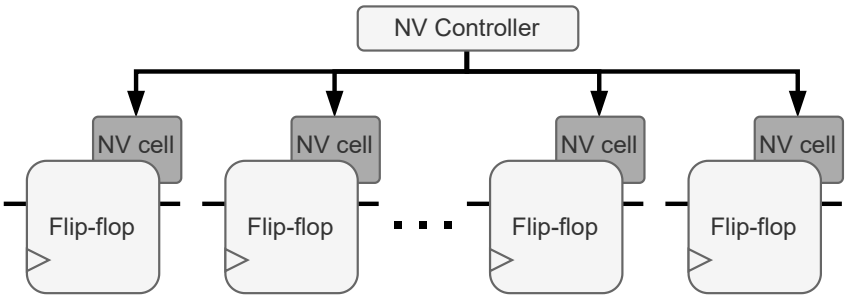
In some cases, copying the entire data memory is not possible because it would require too much time and energy. It is necessary to either use a non-volatile approach or to set up mechanisms to accelerate the data transfer between volatile and non-volatile memories. It is possible to reduce the time needed for data saving by avoiding unnecessary transfers. Pala et al. [83] introduce a backup and restore controller that can reduce the backup size by monitoring the memory accesses. This component keeps track of the memory locations that are modified (written) during active phase. Then, before a power outage, only the words that were modified (write operations) are saved to the NVM. This technique makes it possible to achieve up to an 87.7% reduction in the size of backups depending on the application.

However, saving S_{inst} and S_{data} is not sufficient to build a NV-MCU with instant on/off capability to sustain computation progress throughout the standby cycles and thus ensure the continuity of the application execution upon wake-up. To avoid the restart of the application each time the system wakes up, it is necessary to save the useful registers for its operation to be able to resume the application after standby. To achieve instant on/off capability, most of the register must be saved ($S_{regImp} \cup S_{regFile} \cup S_{regPeriph}$). At core level, the entire content of the register file $S_{regFile}$ must be saved but most of the time concerning S_{regImp}, it is not required to save the content of each implicit register used by the CPU core and the peripherals in order to fully recover their state after a sleep phase. Indeed, depending on the application, some of them have their content either outdated or replaced at wake-up. For example, the shift registers of a serial communication peripheral that were powered down will have either no data or corrupted data (if a transfer was interrupted by the sleep phase), so keeping their content is useless. Another example is the program counter registers of a CPU core: sometimes the CPU starts an interrupt handler just after waking-up because it may have to deal with an event. In this case, these registers will be flushed anyway. Moreover, the energy and time cost of fetching again the few instructions loaded into the pipeline stages have to be compared with the backup and restore operations performed on these registers. This is a trade-off between re-configuring a register or

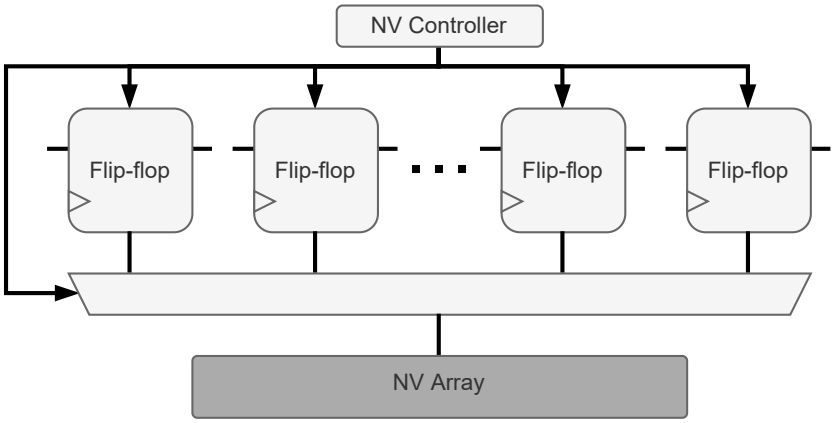
implementing a backup and restore mechanism, and also between the different backup solutions depending on how each register is used. Since implicit registers are by definition not addressable, they are not accessible by software, it is therefore necessary to integrate hardware mechanisms to save and restore the content of these registers.



(a) *Non-volatile flip-flop*



(b) *Flip-flop shadow non-volatile element*



(c) *Flip-flop connected to a non-volatile array*

Figure 3.3: *Different solutions to integrate non-volatility at flip-flop level*

NV-MCU have recently received a lot of attention from academia and industry for their instant on-off capability that allows them to operate continuously from an intermittent power supply. They are mostly used for very low duty-cycled applications or in the case of energy harvesting based applications running on battery-less systems. In such systems, the entry into the standby phase can occur at any time (*i.e.*, when the energy source weakens), so the application's duty cycle depends on the energy available in the environment. In these applications, the backup and restore phases are used to save the whole state of the microcontroller when power is not available. A mechanism capable of detecting a power loss (voltage drop detector) initiates the backup procedure. Then, a capacitor connected to the power line provides enough energy to finish

the backup operations. Once the backup procedure is completed, the system can be safely powered off. When the power source is recovered, and the capacitor is charged, the restore phase is launched and the application can restart. Despite the use of capacitors, in such application, the energy budget for backup and restore operation remains very limited. These mechanisms must be very fast and consume as little energy as possible.

To create such a NV-MCU allowing a forward progression of the application through standby phases, it is necessary to integrate non-volatility at the register level. Of course, the register file can be implemented directly using non-volatile memory arrays, but for implicit registers, it is necessary to replace all the registers to be backed up by their non-volatile equivalent. As shown in Figure 3.3a these registers use flip flops (FF) that store their contents into non-volatile elements each time they are written. As for using a non-volatile memory as data memory, non-volatile flip-flops (NVFF) do not require to be saved nor restored before and after powering down the circuit, which is a very efficient solution for backup and restore since they are implicitly done at each cycle. However, the performances and energy efficiency of an NVFF are not the same as for a classic CMOS FF, leading to potential performance limitations and energy consumption and area overhead. As explained in [84], there is a trade-off between the energy overhead (Figure 3.2b; and the run-time overhead in some cases) in active phase due to the poor dynamic performance of such NVFF and the cost of backup and restore operations of the previous and following solutions (Figure 3.2c).

To suppress the energy overhead during run-time, it is possible to use a hybrid FF (Figure 3.3b): a synchronous memory cell composed of a classic CMOS FF and a local non-volatile memory element. The CMOS FF, which is volatile, is used during normal operations and thanks to dedicated signals from a specific hardware device called a non-volatile controller, its content is stored to or restored from the NV elements during the backup and restore phases. With this configuration, hybrid FF matches the performances of classic CMOS FF while being able to backup and restore information thanks to non-volatile memory elements. These non-volatile elements can either be located above the FF [85, 86, 87, 88, 89, 90] or in a dedicated memory array (Figure 3.3c), outside the logic circuits [91, 92]. In the case of a local approach, the content saving of all FF could be done simultaneously in one cycle (more precisely one "step", as it may require more than one clock cycle to write data into the NV part), but this is not always possible because, depending on the amount of data to be backed up, it may generate a peak current that is too high for the power grid. To avoid damaging the power grid, the placement of hybrid FF is optimised to equalise as much as possible the current consumption along the grid during a backup operation, which can also be performed sequentially. The FF are divided into subgroups with a GS size and the backups of each group are done successively taking multiple steps instead of one. This sequential

operation is used for example in [93]. The number of steps can be optimised and be different between backup and restore operations.

When the NV elements are located in a memory array outside the logic circuits and no longer directly behind the FF, the backup and restore operations consist of copying the FF content into a NV array. They take multiple write operations to the persistent array but there can be several arrays that operate in parallel. For each array, the completion time is limited by its maximum write speed and the size of the array [94]. On the other hand, compared to hybrid NVFF, memory arrays are more area-efficient (*i.e.*, periphery shared amongst multiple memory cells) and they facilitate the integration of memory technologies with lower CMOS compatibility.

$$
\begin{aligned}
(E_{bu})_{FF} &= card(S_{regImp} \cup S_{regFile} \cup S_{regPeriph}) \times E_{store} \\
(E_{re})_{FF} &= card(S_{regImp} \cup S_{regFile} \cup S_{regPeriph}) \times E_{restore} \\
T_{FFparaBu} &= T_{store} \\
T_{FFparaRe} &= T_{restore} \\
T_{FFseqBu} &= \frac{card(S_{regImp} \cup S_{regFile} \cup S_{regPeriph})}{GS} \times T_{store} \\
T_{FFseqRe} &= \frac{card(S_{regImp} \cup S_{regFile} \cup S_{regPeriph})}{GS} \times T_{restore}.
\end{aligned}
\tag{3.8}
$$

NV-MCUs are generally designed for energy harvesting based applications, but their zero leakage capability makes them very good candidates for any type of application based on the normally-off computing paradigm. Table 3.2, Table 3.3 and Table 3.4 compare the state of the art reported NV-MCU that use FRAM, RRAM and MRAM. We mainly focus on the classic metrics of a microcontroller (maximum operating frequency, active power, standby power, and circuit area). To these metrics will be added those specific to the backup procedure (backup time and backup energy) and to restore procedure (restore time and restore energy).

FRAM-based hybrid FF ([87, 94, 90]) are generally slow for backup and restore because of the time required to charge and discharge the FeCap, which needs to be relatively large to avoid endurance problems [99, 100, 101]. Indeed, increasing the capacity of the FeCap increases the reliability of the FF but has the effect of slowing down the writing speed of the non-volatile element and increasing FF area overhead so there is a fundamental trade-off between reliability and access speed. For instance, Su et al. [94] show that high reliability is guaranteed with a little less than 50% area penalty. FRAM technology is therefore limited in its speed by this physical limit. A slight reduction in the surface area of the FeCap increases the writing speed but exponentially raises

Table 3.2: *FRAM based NV-MCUs*

Work	[95]	[92, 91]	[87, 94]	[90]
Technology	130 nm	130 nm HVT	130 nm	130 nm
Processor	MSP430	ARM Cortex-M0	MCS51	ATMEL 8051
Voltage/Frequency	24 MHz	1.5 V 8 MHz	Core: 0.9V-1.5V 25 MHz	Logic: 1.3 V FRAM, analog, I/O: 3.3 V 16 MHz
Memory Architecture	16 kB FRAM (with ECC)	2537 NVFF (10x 256b array) 64 kB FRAM	1607 FeNVFF 8 kB SRAM	1911 NVFF 2x 32 kB FeRAM
Area	4.4 mm^2	Core: 4.4 mm^2 CPU: 1.1 mm^2	11.86 mm^2	22.09 mm^2
Active Power	82 µA/MHz	75 µW/MHz	160 µW/MHz	371 µW/MHz
Standby Power	-	0	0	0
Backup energy/time	-	2.2 pJ/bit, 320 ns	14.4 pJ/bit, 7 µs	9 µs
Restore energy/time	-	0.66 pJ/bit, 384 ns	5.04 pJ/bit, 3 µs	3 µs
Operating context	Energy harvesting	Low duty-cycle	Energy harvesting	Energy harvesting
Memory arrays	Hybrid	Hybrid	Volatile only	Non-volatile only
Flip-flops	Volatile	Hybrid (using array) core only	Hybrid core only	Hybrid core and peripherals

the error rate. However, the use of memory arrays ([92, 91]) allow greatly reducing the time needed for backup and restore at the price of an increase of the FF area by 19% and the total area of the SoC by 3.63%.

Concerning RRAM, hybrid FFs typically allow faster restores (up to one hundred times) than-FRAM based ones [102, 103, 104]. To limit the restore time, it is possible to use an intelligent adaptive non-volatile controller ([88, 96]). This module allows to adapt backup and restore content by analysing the power source and preventing inefficient NVM store/restore operations. Finally, hybrid FF based on MRAM technology are generally the most efficient (reduction of store time by an order of magnitude compared to RRAM technology) at the cost of a larger area overhead (312% of overhead compared to conventional CMOS FF) [105, 106, 107].

The NV-MCU developed in [84, 98] demonstrated both the highest operating frequency and the smallest active power (in µW/MHz) thanks to very fast hybrid FF and to a memory access controller (MEM4X) that accelerates the data transfer between the STT-MRAM and other modules by up to four times by incorporating both memory-access multiplexing and code-length-aware prefetching techniques. These FF can save their state in 2.39 ns and restore it in 0.12 ns. As the writing energy of such technologies is much higher than the reading energy, it is also possible to minimize the total energy needed for backup by using the read before write method. Indeed, in [86], they integrated an active-high dirty bit (DT) to designate that the register's data differs from the non-volatile element's one and thus avoid the unnecessary backup. This mechanism is particularly effective for the peripheral registers because they toggle very rarely (it

Table 3.3: *RRAM based NV-MCUs*

Work	[89]	[88, 96]	[97]
Technology	150 nm	65 nm SVT	130 nm
Processor	Intel 8051	8-bit processor	MSP430
Voltage/Frequency	0.8 V 20 MHz	Core: 0.8 V (0.4 V - 1 V) HV: 3 V 100 MHz	0.71 - 1.2 V 10 MHz
Memory Architecture	2189 ReNVFF 4 kb NVSRAM	1422 NVFF 4 kB NVSRAM 8 kB ReRAM	18 kB RRAM 8 kb SRAM
Area	3.6864 mm²	4.4616 mm²	11.25 mm²
Active Power	1.1 mW/MHz	33 µW/MHz	47 µW/MHz
Standby Power	0	0	0
Backup energy/time	57.96 nJ, 100 ns	0.40 µJ, 4 µs	1.6 nJ, 4.7 µs
Restore energy/time	0.51 nJ, 50 ns	0.45 nJ, 20 ns	152 pJ, 200 ns
Operating context	Energy harvesting	Low duty cycled and Energy harvesting	Low duty cycled
Memory arrays	Non-volatile only	Non-volatile only	Hybrid
Flip-flops	Hybrid	Hybrid	Volatile

is possible to get more than a 50% reduction in backup energy when 10% of peripheral registers toggle).

To summarise, the common goal of these NV-MCUs is zero power consumption during standby. It is possible to effectively combine emerging non-volatile memories with CMOS technologies to design NV-MCUs well suited for normally-off applications. All these works focused on optimising the content of these backups (adaptive non-volatile controller, read before write) and reducing the backup and restore time (using very high-speed hybrid FF) to reduce the percentage that these backups represent on the energy available. Indeed, most of these works target energy harvesting based systems embedding simple applications. In our case, we are targeting edge computing applications. Thus, we plan to explore the integration of these energy-saving strategies in edge-computing enabled MCUs. To do this, we need to model the power consumption of an MCU embedding such mechanisms. For this reason, in the next section, we propose an activity-based energy model for NV-MCUs.

3.3 NV-MCU energy modelling

In the previous sections, we described emerging NVM and how their integration in MCU architecture allows the building of energy-efficient systems for normally-off applications. In this section, we build a parametric energy model of NV-MCU and its different components. This model takes into account the energy related to the integration of backup mechanisms for energy saving. To do so, we split the active phase into three

Table 3.4: *MRAM based NV-MCUs*

Work	[86]	[84, 98]	[85]
Technology	90 nm SVT	40 nm	CMOS 180 nm, STT 200 nm
Processor	MSP430	ARM Cortex-M0	SecretBlaze
Voltage/Frequency	Core: 1 V HV: 1.8 V - 3.3 V 20 MHz	Core: 1.1 V / 1.3 V Per.: 1.8 V / 3.3 V 200 MHz	Core: 1.8 V IO: 3.3 V 20 MHz
Memory Architecture	64 kB 3T-SpinRAM	476 NVFF 64 kB 2T-STT-MRAM	2126 NVFF 512 B ROM 2x 16 kB SRAM 16 kB STT-MRAM
Area	22.94 mm²	22.09 mm²	23 mm²
Active Power	145 µW/MHz	26.7 µW/MHz	-
Standby Power	<0.1 µW	0.7 µW	-
Backup energy/time	6 pJ/bit, 4 ns	866.38 fJ/bit, 2.39 ns	437 nJ, 4.15 µs
Restore energy/time	0.3 pJ/bit, 120 ns	8.19 fJ/bit, 0.12 ns	98.7 nJ, 4.15 µs
Operating context	Low duty-cycle	Low duty-cycle	Low duty-cycle
Memory arrays	Non-volatile only	Non-volatile only	Hybrid
Flip-flops	Hybrid	Hybrid	Hybrid

sub-phases: (1) restore, (2) operation and (3), backup as depicted in Figure 3.4. Then, we can build a simplified power model for each logic block (*e.g.*, CPU core or peripherals) of the architecture by knowing their consumption during each phase. The challenging part is memory energy evaluation, as the consumption of a memory depends on both its operating modes and its accesses. In addition, the architecture can integrate multiple memories. Thus, we model the memory energy as $E_{mem} = E_{dyn} + E_{stat}$, with E_{dyn} being the sum of the dynamic energy of each memory (e_{dyn}) and E_{stat} the sum of their static energy (e_{stat}). Figure 3.5 illustrates how we model the energy of a single memory.

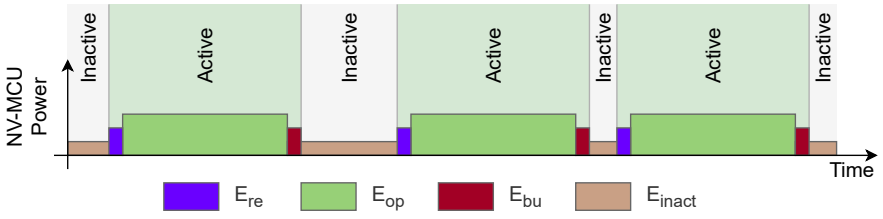


Figure 3.4: *NV-MCU energy model*

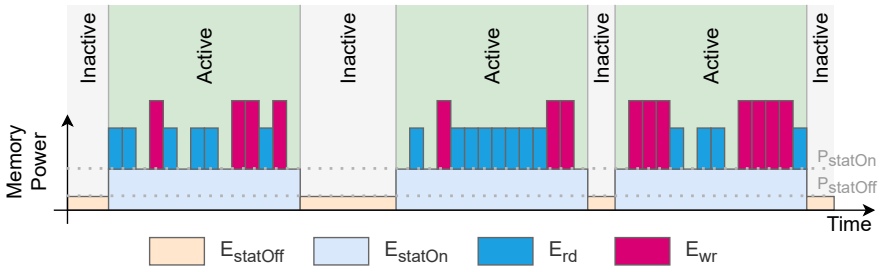


Figure 3.5: *Memory energy model*

For static energy, we assume a typical memory level power management, *i.e.*, power modes can be controlled independently for each memory. We also assume `On` and `Off` modes. Memories can only be accessed in `On` mode, which corresponds to a static leakage power of P_{statOn}. The `Off` mode allows a lower static leakage power $P_{statOff}$ but leads to the loss of the volatile memory content. The static energy of an execution phase of duration T_{phase} is $e_{stat} = P_{stat} \times T_{phase}$. For P_{stat} during a any active phase, we consider all memories that contain data or instructions as `On` while other memories are `Off`. Finally, all memories are `Off` during inactive phases. In this way, it is sufficient to use the temporal parameters defined earlier, such as T_{act} or T_{inact}.

We define the dynamic energy of a memory as $e_{dyn} = (n_{rd} \times E_{rd}) + (n_{wr} \times E_{wr})$, n_{rd} and n_{wr} being the number of read and write accesses, E_{rd} and E_{wr} the read and write energy, respectively. Finally, we consider no activity in the memories during inactive phase. Thus, to calculate the energy for different memories, we simply model each technology with the following parameters: E_{rd}, E_{wr}, P_{statOn}, $P_{statOff}$. We will see in Section 4 how to collect and use these parameters to calculate the energy consumption of a memory.

3.4 Summary

In this chapter, we dived into the memory architecture of MCUs, first describing their hierarchy and the different technologies that can be used. Then we explained how NV-MCUs are designed to build energy-efficient devices for intermittent applications before defining an energy model for their logic blocks and memories. The detailed model of the MCU energy will allow the exploration of the MCU architecture, especially the integration of non-volatility. The next challenge is to evaluate how edge computing can affect the energy consumed by end-nodes and their MCU to identify bottlenecks. For this, we need a solution to evaluate the set of parameters defined for different IoT applications.

IV

Platform for end-node energy evaluation

Contents

In the previous chapters, we proposed models for sensor node and MCU energy evaluation based on activity. These models are based on the typical execution of a sensor node application software and allow to translate for each sensor node and MCU component a sequence of events into an energy consumption.

To evaluate different applications, we need a platform to run sensor node application software. This platform must support a wide range of applications, from simple temperature-like sensor nodes to more complex edge computing capable sensor nodes. Due to the event-based nature of embedded applications, this solution must also take into account the interactions with the environment and with the whole IoT network. As it may be necessary to evaluate these activities over long periods of time, it is preferable to run the application in real-time (*e.g.*, we need one second to evaluate the application for a one-second evaluation period).

Regarding activity evaluation, the solution must allow assessing the activity of the different components at node level but also at MCU level during the execution of the application (see Figure 4.1). Furthermore, these evaluations should be cycle accurate and not impact the execution of the application and, thus, interactions with the environment and the network. Thus, based on our predefined models and the activity of the different components, we can evaluate the energy consumption of the different components of the sensor nodes. These evaluations must be precise enough to identify the energy bottleneck and to guide the hardware and software exploration to reduce the energy consumption of the node.

To improve energy efficiency, the platform should allow the exploration of different software and hardware alternatives at both sensor node and MCU levels and consider different environments and network interactions at the application level.

In this chapter, we propose an FPGA-based platform that meets all these requirements for the evaluation and exploration of sensor node applications. We then propose two typical applications that represent the ones that can be found in these nodes. Finally, we use our platform to evaluate the energy of the nodes for each of the two representative applications.

4.1 State of the art of evaluation methods

Performance and energy evaluation of electronic devices while executing applications is a challenging task. The challenging part is that sensor nodes are intended to interact with their environment and a network, thus, their energy consumption is directly related to the application and their environment. To summarise, we need a platform

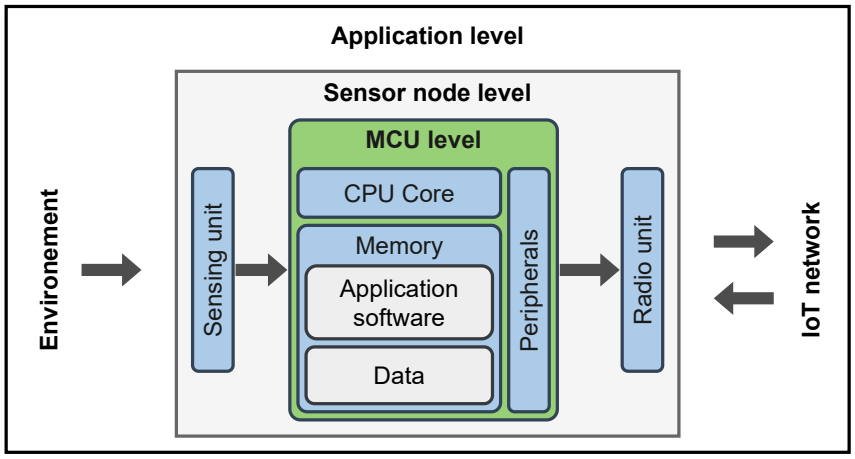


Figure 4.1: *Different levels for evaluations a explorations*

for real-time execution of a wide range of sensor node applications under real-life conditions. This solution should allow non-intrusive cycle-accurate evaluation of the activity of different components and application-architecture co-optimisation at both the sensor node and MCU level.

We can classify existing methods into two main categories: (1) simulator and (2) hardware prototyping. In this section, we review the methods that can be used to retrieve component activity while running different IoT applications. Finally, we describe the chosen methodology that allows for real-time evaluation, taking into account the application context.

4.1.1 Simulators

The first approach is to use a simulator. Simulators can simulate the operation of an electrical circuit that runs a software application and thus evaluate the activity of its different components. These simulators are a flexible and inexpensive solution for system evaluation and design space exploration. Any architectural modification can be made to the model as long as it is editable, thus representing any behaviour of a system. It can be rerun and multiple simulations can be launched in parallel. The simulator does not require specific hardware, which makes it a cheaper solution than prototyping.

Table 4.1: *Different simulator tools for activity evaluation*

	Simulator			
Tool	Activity eval.	Real-time	Expl. flexibility	Eval. in real-life conditions
RTL/Microarchitecture simulator	Cycle/Bit	No	High	Difficult
Architecture Simulator	Cycle	No	High	Difficult
Instruction Set Simulator	Instruction	Yes	Low	Difficult
WSN Simulator	Network	Yes	Low	Possible

As depicted in Table 4.1, there are several types of simulators depending on their abstraction level and their accuracy:

- RTL/Microarchitecture simulators such as ModelSim or QuestaSim are bit-accurate, but time-consuming. Even if these simulators allow for all kinds of design space exploration and to evaluate the performance and activity of a system in a very precise way, they are too slow to be considered in our case.
- Architecture Simulators such as gem5 are faster than RTL simulators. They allow building, configuring, and simulating an architecture using behavioural models for each component, making them a suitable solution for architecture evaluation and exploration. Their main limitation in our context is that they are computer architecture orientated and do not integrate models for commonly used MCUs. Moreover, even if they are faster, they do not allow for real-time evaluation.
- Instruction Set Simulators (ISS) are even faster, allowing real-time. However, these simulators focus on instruction execution and thus are really detached from the hardware, which leads to the loss of information and accuracy [108].
- Sensor Network Simulators [109] allows sensor network performance evaluation. These simulators have a higher level of abstraction and allow to emulate the data exchange in a sensor node network in order to analyse its performances (*e.g.*, energy, node to server latency). They are often used for algorithms and communication protocol evaluation in wireless sensor networks. Some of these simulators even integrate modules to emulate the interactions with the environment [110]. However, in our case, the level of abstraction of these simulators is too high to explore the MCU architecture.

Cycle-accurate simulators for MCU architecture explorations do not easily model the interactions between the different components of the sensor node and the interactions with the environment. Indeed, they need to be combined with complex test benches to allow a close-to real-life evaluation. To sum up, these simulators are either accurate and slow (it takes a long time to simulate the system's functioning during a short period) or faster, but their level of abstraction is too high to allow the exploration of the architecture in detail. In our case, we need a solution with a low-level abstraction for design exploration and capable of evaluating activity in real-time to enable evaluation over long periods.

4.1.2 Prototyping

Hardware prototyping allows for real-time evaluation in real-life conditions. As depicted in Table 4.2, there are two possible solutions: (1) silicon prototyping and (2) FPGA prototyping.

Table 4.2: *Different prototyping techniques for activity evaluation*

Prototyping				
Tool	Activity eval.	Real-time	Expl. flexibility	Eval. in real-life conditions
Silicon prototype	Cycle	yes	Low	Native
FPGA prototype	Cycle/Bit	yes	High	Native

The first solution consists in building a silicon prototype based on existing MCUs equipped with performance and activity monitors to evaluate the activity of different blocks in the circuit. Commercial ultra-low power (ULP) MCUs integrate trace modules or Performance Monitoring Units (PMU). These performance monitors consist of programmable counters, configurable from software, used to capture events indicative of processor performance. These advanced tools are typically used for programme analysis and optimisation. They are capable of collecting information about the execution of an application and can be used to analyse CPU activity [111] [112]. The monitor output can be used to evaluate architecture bottlenecks during application execution (*e.g.*, miss rates). It can also be used for real-time and accurate performance and power evaluations [113]. The activity and the events monitored can be associated with energy consumption (*e.g.*, activation-based estimation), so it is possible to evaluate the consumption of the system in addition to its performance with the output of these monitoring units. Even if these tools allow to evaluate the consumption of several blocks of the MCU architecture, they are not available on all MCU, particularly on small ones, their presence being dependent on the manufacturer's choices. Furthermore, these modules are focused on certain blocks of the logic circuit and cannot monitor the activity or events of other blocks. The main limitation of this method is the lack of flexibility in using a non-modular circuit, which severely limits design space exploration.

The second solution is to prototype the system on an FPGA target. In this way, it is possible to combine real-time evaluation with a high degree of flexibility. For example, programmable logic flexibility 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. [114] propose an FPGA implementation that integrates a monitor to evaluate and optimise 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. [115] present an implementation of a PMU in an FPGA-based LEON3 platform, but the presented solution focuses on processor performance. Their architecture is a complex multi-core architecture that does not correspond to the ULP SoC archi-

tecture. Moreover, their PMUs are located in each core and their counters only focus on the core performance (including L1 miss rates). In [116] and [117], the authors use an FPGA prototyping method to evaluate the performance of hybrid memory architecture. However, these works target high-performance computing architectures and applications, which differ significantly in the intermittent edge computing domain targeted in this work.

Since the microcontrollers used in IoT nodes usually operate at low frequencies, FPGA prototyping 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. There are a large number of microcontroller architectures described in a Hardware Description Language (HDL). They target the development of prototypes and applications for evaluation and design exploration. For instance, the PULP platform [118] is an open-source low-power RISC-V architecture. PULPissimo [119] 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 building this complexity gradually. As a result, it also requires many FPGA resources for prototyping. Alternatively, in recent works of the ADAC team, Patrigeon et al. presented Flexnode, a configurable prototyping platform used for MCU architecture exploration and real-time application evaluation [81]. A memory technology exploration methodology that targets MRAM integration in ultra-low power MCU architecture is proposed. This platform is based on an MCU implementation on an FPGA connected to real life sensor and radio unit. Thus the platform can run IoT applications in real-time and in real-life conditions. The MCU architecture is instrumented with a monitor that collects the activity of the different components of the MCU which when combined with energy models, allows to evaluate their consumption. The use of programmable logic allows a great degree of freedom in the explorations, which corresponds perfectly to the above-mentioned requirements. However, the Flexnode evaluation methodology suffers from the following drawbacks and limitations:

- Even if the consumption of the radio and sensing units can be measured on the prototype, their activity is not measured during the execution of the application, making activity-based estimation impossible. Thus it is necessary to measure their consumption for each modification of the application or the architecture.
- The MCU architecture is too basic for edge computing applications. The proposed architecture is based on a Cortex-M0 CPU core, which is connected to a single master bus with memories and peripherals. This architecture is not optimal to consider processing and storage intensive applications.

- The interfacing and management of the activity monitor is handled by the CPU and thus in the application, so it is necessary to stop the execution of the application to collect the activity data. This solution is simple but intrusive: the collection of data from the monitor has an impact on the execution of the application.
- The core used is a Cortex-M0, a proprietary core developed by ARM. It is used as a hard macro, so it is not editable and it is very difficult to monitor its internal activity for micro-architecture or software/hardware interface explorations.

4.2 Proposed evaluation methodology

We follow a similar methodology as the one proposed by Patrigeon et al. but with an enhanced platform aiming at assessing energy with activity monitoring. The proposed platform, which is presented in detail in Section 4.3, can be used to evaluate the node performance and energy consumption and enable further application-architecture co-exploration. Our enhanced platform enables:

- Activity-based sensing and radio unit energy evaluation.
- The possibility to execute more complex applications thanks to an adapted MCU architecture.
- A non-intrusive monitoring of the activity of the different components thanks to the integration of a dedicated monitoring unit isolated from the other components.
- Further application-architecture co-exploration thanks to a hard macro free design based on an open-source Instruction Set Architecture (ISA).

Figure 4.2 presents the proposed evaluation flow. The methodology can be divided into three main steps:

- ① Design of the platform according to the application's specifications.
- ② Measurement of the consumption of the components according to their activity or mode of operation to build their power models.
- ③ Evaluation of the energy of the different components based on the execution of the application during a T_{eval} duration.

①: From the application specifications that describe the device's mission to provide a service, we can choose the optimal IoT network infrastructure and the sensing and radio unit that nodes must embed. Then, we build an adapted MCU architecture and the application sources to go with it. The MCU architecture is then implemented in an FPGA target, which is connected to the sensing and radio to form the node prototype. Application sources are built with a toolchain to generate a binary file that can

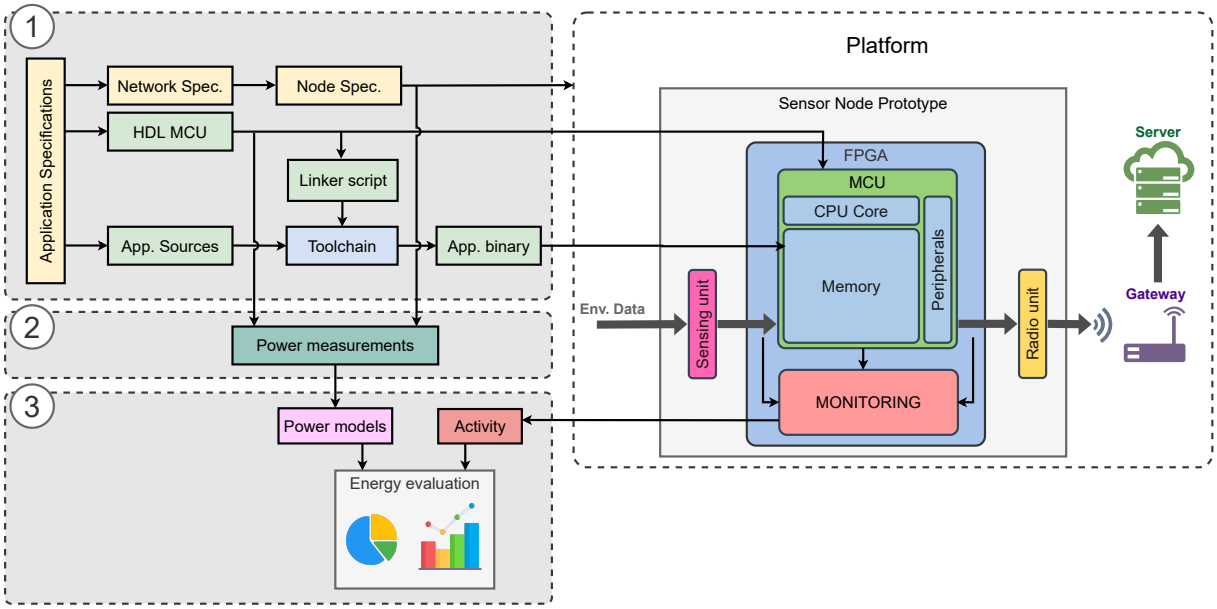


Figure 4.2: *Evaluation methodology and platform*

be loaded into the prototype. Finally, the developed nodes are deployed in their target environment and connected to an IoT network to operate in real-life conditions to constitute our evaluation platform. The so-called platform is flexible, allowing it to be used for a wide range of applications. Indeed, it is possible to implement various designs, the only limits being the operating frequency and the hardware resource limits of the target FPGA. It also allows a high degree of freedom for exploration. It makes it possible to test different environments and IoT network setups at the application level. The node prototype can embed different sensing and radio unit for node-level explorations. Finally, thanks to the programmable logic, it is possible to explore the hardware architecture, the embedded software and the interfacing between the two at the MCU level.

②: Once the node is designed, the second step consists in measuring the consumption of the different components according to their activity (*e.g.*. energy consumption in their different operating modes or energy related to different operations) to build their power models. The goal is to make energy evaluations at both the node and MCU levels by separating the contribution of the different components at both levels. At the node level, it is therefore sufficient to characterise the power consumption of the sensing and radio units as a function of their activity. As real components are used on a prototype, we can get their power consumption using measurement tools directly on the prototype (as done in Section 4.5.1). Concerning MCU, power model building is more complex, as the power consumption of the microcontroller and its different components cannot be measured in the same way as for sensing and radio units. Indeed, the power consumption of a circuit implemented on FPGA substrate is not representative of the power consumption of a circuit directly implemented with stan-

dard cell ASIC technology. The objective is to evaluate the power consumption of the various logic circuits and memory banks contained in the MCU. To estimate the power consumption of the MCU's logic blocks, we measure their power consumption by performing Register Transfer Level (RTL) simulation for particular functional primitives (*e.g.*, execution of an add instruction). Then we can build a simplified power model for each block of the architecture as described in Section 3.3. Concerning memories, based on the energy model proposed in Section 3.3, we simply need for each memory to collect a set of four parameters (E_{rd}, E_{wr}, P_{statOn}, $P_{statOff}$) depending on their technology and size.

③: Once the platform and power models of each component are ready, the third step consists in evaluating the energy consumed by the different components at both levels during application execution. To do this, the FPGA target embeds a monitoring unit that is able to record the activity of the different components at both levels while the application is running for a T_{eval} duration. This monitor counts for each component certain events that describe their activity and thus combined with their power models allowing to evaluate their energy. The monitor counts the number of cycles spent in each operating mode for the sensing and radio unit and MCU's logic block. Thus by knowing the system clock speed, we can get the time spent by each component in each of their operating modes. For example, to evaluate the consumption of the radio in our application, we can use the monitor to know the duration of each application phase. Knowing that the radio module consumes P_{act} in the communication phases and P_{stdb} which is equal to zero thanks to dedicated power gating in all other phases. The total energy consumed by the radio module is $P_{act} \times T_{com}$. Concerning memories, we use the same approach for static energy, but concerning dynamic energy, the monitor must also count each memory access in memory.

The ① and ② steps are only done at the beginning for each application, and must be repeated only in case of addition or modification of a component. The ③ step, on the other hand, can be executed quickly and even in real-time without affecting the normal operation of the application. Thus this method allows to combine evaluation of the explorations at all levels from software to hardware through the toolchain and a fast evaluation. Next, we will describe the design of this platform in more detail.

4.3 Platform development

The following section describes in detail the proposed sensor node prototype before describing the proposed state-of-the-art representative MCU architecture. We then describe how the prototype is instrumented with a monitoring unit to retrieve all pa-

rameters needed to evaluate the energy of the different components of the node and the MCU to evaluate their energy. Finally, we use the methodology described in Section 4.2 with the proposed prototype to build a platform to evaluate node energy for two different applications.

4.3.1 Node prototype

Our methodology is based on a platform to evaluate the consumption of the node and its components during the execution of a real application in real-time. Our platform embeds one or more FPGA-based node prototypes. A node prototype corresponds to an FPGA target on which an MCU is prototyped and connected to sensing and radio units. As we target edge computing applications with high storage requirements, we need an FPGA target with enough hardware resources to implement large memories (>512kB). For this reason, we chose to use the Nexys video board proposed by Digilent. This board integrates a Xilinx XC7A200T FPGA that embeds 1,625kB of BRAM. This FPGA allows us to implement more than 1MB of storage in the MCU architecture. In addition, Digilent's development boards, such as the Nexys Video, have PMOD connectors to interface with other boards. Digilent offers a wide range of PMOD modules, with different sensor and radio units available to support a wide range of applications.

We have also developed a sensing and communication board that can also be connected to the Nexys video. This board embeds several sensing and radio units in the same way as the PMOD modules. However, this board is designed to facilitate the power model building (step ②) of each unit of the sensor node. Each unit has its own power domain allowing for easy isolation of power consumption. In addition, each domain has an integrated connector that allows the power consumption of the unit to be easily measured according to its operating mode.

Figure 4.3 is a picture of the node prototype. The Nexys video FPGA board is connected to the sensor and radio board via its PMOD connectors. This board integrates an energy management unit for battery voltage regulation. The board embeds several sensors, temperature and humidity that represent those commonly used in simple IoT applications but also an inertial measurement unit, a microphone and a camera which are commonly used in edge computing applications. For communication, the board embeds two radio modules:

- A LoRa module because it is a technology that allows conciliating long range with low power. The disadvantage of this technology is its low throughput but this is not a problem in the case of edge computing because the nodes communicate small amounts of data (see Section 2.3).

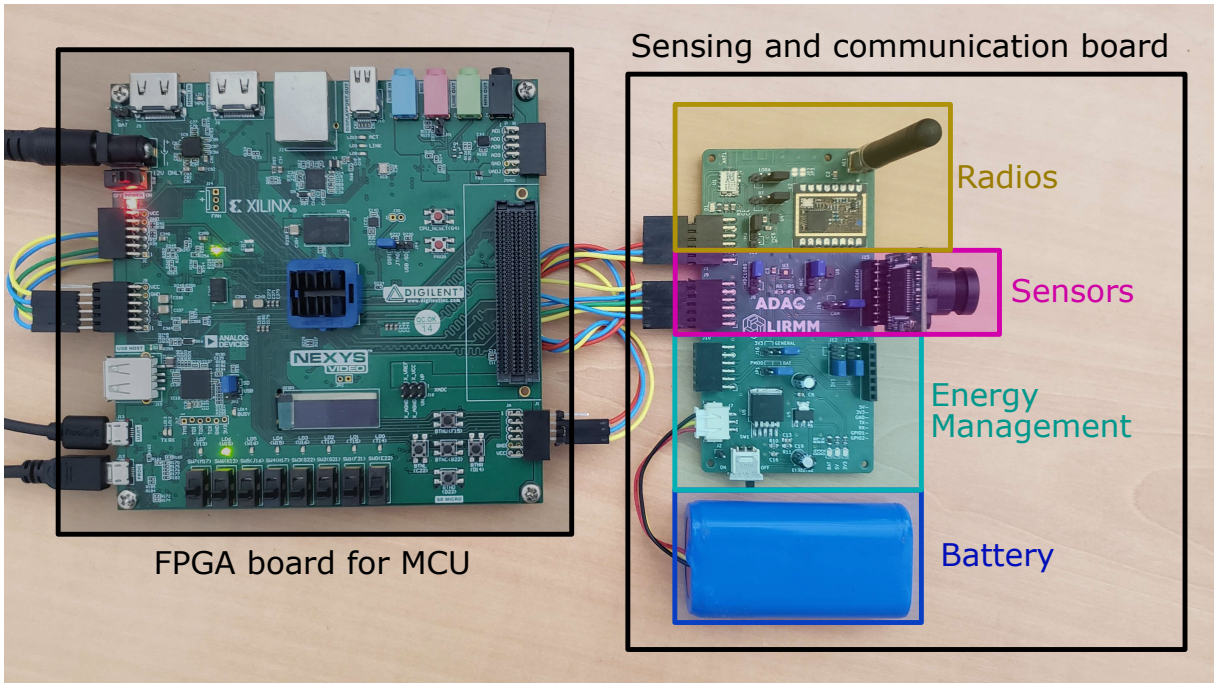


Figure 4.3: *Proposed node prototype*

- A Bluetooth Low Energy module because this technology allows combining high throughput and low power consumption. The disadvantage of this technology is the small range of the signal (see Section 2.3).

The control and data transfers between the FPGA target and the radio and sensor units are done with General Purpose I/O (GPIO) and embedded communication buses such as UART, SPI or I2C (described in the next section). Thus, the MCU must be equipped with these communication interfaces to design a platform that can integrate a wide range of sensing and radio units. In summary, the FPGA target must include an MCU with processing capability, memories, GPIO and communication peripherals.

4.3.2 MCU architecture

There are various types of MCU suited for different applications. The application specifications determine which microcontroller to use. Manufacturers offer a very wide range of microcontrollers implementing different functionality and having different properties. For each application, there is a matching MCU offering the optimal compromise between performance and minimum power consumption. As shown in Figure 4.4, most MCUs include a Reduced Instruction Set Computer (RISC) CPU core, a set of task-specific logic circuits such as Direct Memory Access (DMA) or Wake-Up Controller, peripherals to communicate with other components, Volatile Memory (VM) and Non-Volatile Memory (NVM). NVM is usually based on a ROM for the boot code and a storage memory (Flash or EEPROM) for the program and read-only data. The

VM (SRAM) is used to store data during program execution. There are microcontrollers with different package types and sizes, operating frequencies, communication interfaces, memory technologies, memory capacities, and power consumption modes.

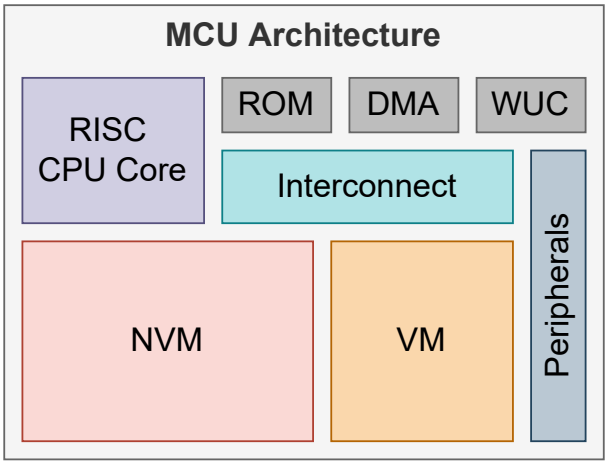


Figure 4.4: *General MCU architecture*

Wireless sensor node dedicated MCU are designed to have low power consumption, both in operation and in inactive phases, and high energy efficiency. They typically have a small CPU core, smaller memory capacities than most microcontrollers designed for higher power applications, and some basic communication interfaces (UART, I2C, SPI). Some microcontrollers incorporate a radio engine to eliminate the need for an external radio unit, thus reducing the size of the node, its power consumption and potentially its cost.

4.3.2.1 State-of-the-art commercial MCUs

Figure 4.5 is a schematic of the STMicroelectronics STM32L072 MCU architecture from its datasheet. This microcontroller is sold to be used in low-power embedded systems such as sensor nodes. We can identify in this architecture the following different components:

- Cortex M0+ 32b CPU Core (purple).
- 192 kB Flash and 6kB EEPROM as non-volatile memory (red).
- 20 kB SRAM as volatile memory (orange).
- Interconnect matrix (light blue).
- Various peripherals (dark blue).

This architecture is representative of the state of the art of commercial MCUs for low-power connected objects.

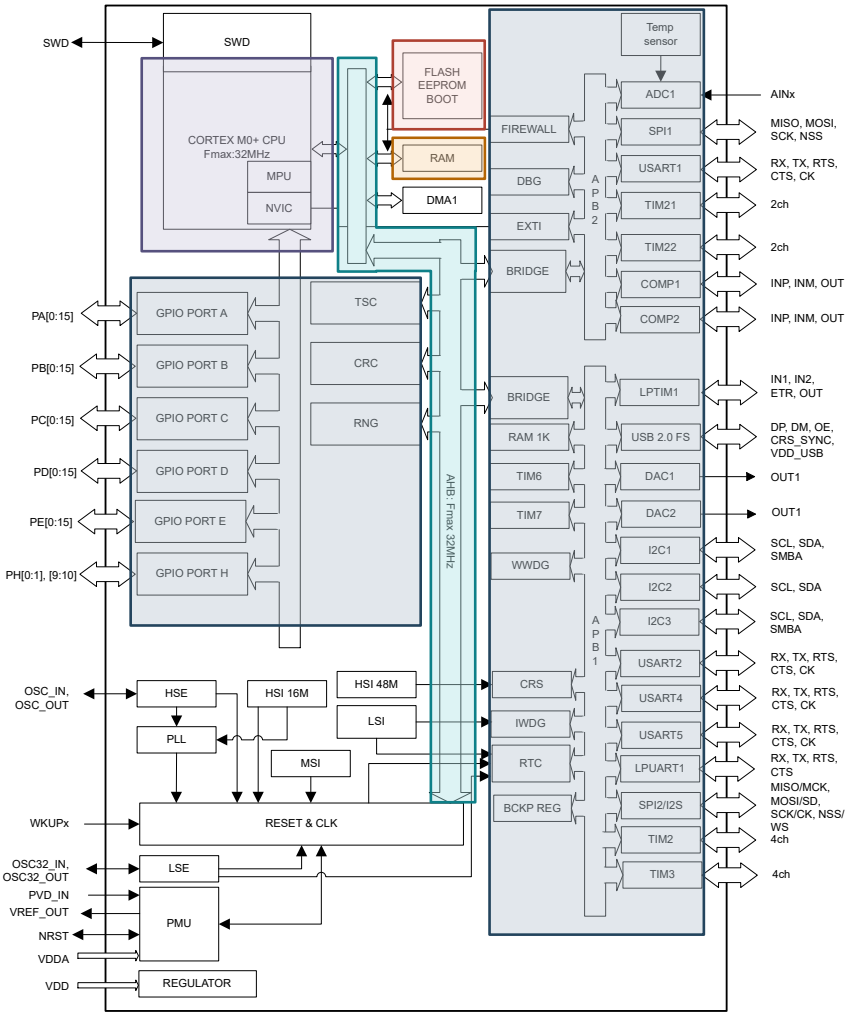


Figure 4.5: *STM32L072 architecture [120]*

RISC CPU cores are based on different Instruction Set Architectures (ISA) and can use different data bus widths. Data bus width affects the core size and its performance. It also affects the rest of the MCU architecture, as the other components must be compatible. Consequently, the performance and power consumption of the MCU architecture is highly related to the data bus width. Under equal conditions, a 32-bit processor has a higher power consumption than an 8 or 16-bit processor, but tends to perform better when handling large data types and performing complex calculations; thus, its energy consumption to perform a given task is not systematically higher than that of an 8 or 16-bit processor. In the design of MCUs for IoT end-nodes, manufacturers mainly use 32-bit RISC CPU cores. Among these cores, there are several ISAs available such as ARM or MIPS based cores. ARM-based cores dominate the ultra low power MCU market for end-nodes. This ISA is owned by the ARM company, which offers a wide range of small ultra-low power cores such as the Cortex-M0, the Cortex-M0+ and, most recently, the Cortex-M23. These cores have simple architecture and are designed to operate at low frequencies (<50MHz) for optimum energy efficiency.

These proprietary cores are sold as hard 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. This is not suitable for explorations, in fact, for development and exploration, the use of softcore is usually prefered. 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) [121]. RISC-V is an open ISA, unlike ARM or MIPS. It is provided under open source licenses that do not require fees to use. RISC-V is a modular ISA consisting of different base parts completed with optional extensions. The ISA, including its extensions, is supported by a dedicated GCC compiler. There is a lot of interest in the development of RISC-V, both in academia and in industry, including the development of the microarchitecture and the ISA (*e.g.*, adding new extensions).

Concerning memories, low-power MCUs usually have at least one non-volatile and one volatile memory, as shown in Table 4.3.

Table 4.3: *Example of commercial MCUs for IoT [26]*

			VM	NVM	
Product	Frequency	CPU Core	SRAM	Flash	EEPROM
STM32L0 series [122] STMicroelectronics	32MHz	ARM Cortex-M0+	2 to 20 kB	8 to l92 kB	512 to 6144 B
STM32F0 series [123] STMicroelectronics	48MHz	ARM Cortex-M0+	4 to 32 kB	16 to 256 kB	-
LPC800 series [124] NXP Semiconductors	30MHz	ARM Cortex-M0+	1 to 16 kB	4 to 64 kB	-
LPC1100 series [125] NXP Semiconductors	50 MHz	ARM Cortex-M0 ARM Cortex-M0+	2 to 36 kB	4 to 256 kB	512 to 4096 B
PSoC 4 family [126] Cypress Semiconductor	48 MHz	ARM Cortex-M0 ARM Cortex-M0+	2 to 32 kB	8 to 256 kB	-
RXl00 series [127] Renesas	32MHz	Renesas RXv1	8 to 64 kB	8 to 512 kB	-
PIC32MM Family [128] Microchip	25MHz	32-bit MIPS	4 to 32 kB	16 to 256 kB	-
SAM L MCUs [129] Microchip	48MHz	ARM Cortex-M0+ ARM Cortex-M23	4 to 40 kB	16 to 256 kB	1024 to 8192 B

For non-volatile memory, Flash technology is mainly used, while for volatile memory, SRAM technology is used. Some MCUs can integrate other types of memory:

- ROM (Read Only Memory) often contains a boot code that allows the MCU to boot directly and/or program the Flash content with a new application code by retrieving it from a communication peripheral. The content of this memory is fixed at production and cannot be modified.
- EEPROM (Electrically Erasable Programmable Read-Only Memory) are non-volatile memories similar to Flash. They have the advantage of being easier to program but have a very low density. However, low-capacity EEPROMs are used to save data or parameters in case the microcontroller is switched off, the power supply is interrupted or an aggressive power-saving mode is entered, resulting in the loss of the volatile memory content.

Some MCUs also have an interface to add external memory to increase their capacity. The size of the memory in the MCU varies greatly, as it depends heavily on the application's needs. As unnecessarily large memory leads to unnecessary power consumption, there are products with all memory sizes so that there is a suitable product for every application. It should be noted, however, that non-volatile memory based on Flash technology rarely exceeds a few MB and that the amount of volatile memory is often much smaller (512 KB maximum).

An MCU integrates several peripherals in order to communicate with other components, other processing units or external storage. The peripherals are based on registers connected through dedicated logic to other components to perform specific functions. These functions range from simple input/output controllers to state machines for communication with different communication protocols, or circuits for analog/digital conversion:

- UART for Universal Asynchronous Receiver Transmitter is an asynchronous serial communication protocol.
- SPI for Serial Peripheral Interface is a master/slave based synchronous serial communication protocol.
- I2C for Inter-Integrated Circuit is a master/slave based packet switched synchronous serial communication protocol.
- CAN for Controller Area Network is an industrial communication bus with a protocol layer for error detection.
- USB for Universal Serial Bus is a host-scheduled, token-based serial bus protocol.
- ADC for Analog to Digital Converter allow the conversion of analog signals into digital information that can be used by the MCU.
- DAC for Digital to Analog Converter allow generating analog signals from the MCU.

MCUs also embed a set of components that are controlled in the same way as the peripherals but which are not dedicated to interaction with the outside world. In fact, these components provide multiple functions ranging from simple counters to complex calculation acceleration. Among these components are timers, Real Time Clock (RTC), Direct Memory Access (DMA), Random Number Generator (RNG), Phase-Locked Loop (PLL) or accelerators (*e.g.*, cryptography, digital signal processing, machine learning).

There are multiple solutions for the interconnection of all these components. From simple buses to complex crossbars, there are a large number of communication protocols used in MCU architecture. The most commonly used protocols are derived from the open-standard Advanced Microcontroller Bus Architecture (AMBA) from ARM. The Advanced eXtensible Interface (AXI) is used in a large number of com-

mercial SoCs. This interface targets high-performance and high clock frequency system designs and is therefore not suitable for low-power architecture. Advanced High-performance Bus (AHB) is simpler than AXI providing less energy and consuming less energy. AHB-Lite is a subset of AHB. This subset simplifies the design for a bus with a single master, thus reducing its energy consumption. Finally, Advanced Peripheral Bus (APB) is designed for low bandwidth control accesses such as peripheral register access. These protocols are very well defined, offer good performance and good energy efficiency. However, they are designed to be used with ARM components that have the same interfaces. If we look at other protocols available, some RISC-V cores use the Open Bus Interface (OBI) protocol. The starting point for the OBI definition is the custom bus interface as used on the RI5CY and Ibex RISC-V cores. This protocol is a request-grant-based protocol similar to the AMBA AXI protocol, but it is simpler, based on few signals and therefore allows for compact and energy-efficient designs.

4.3.2.2 ICOBS: A flexible MCU architecture

We propose ICOBS, a state-of-the-art representative MCU architecture for architecture exploration. This architecture is highly configurable allowing design exploration at all architecture levels. The proposed MCU can handle various applications with high computational and memory requirements. Figure 4.6 describes the proposed architecture.

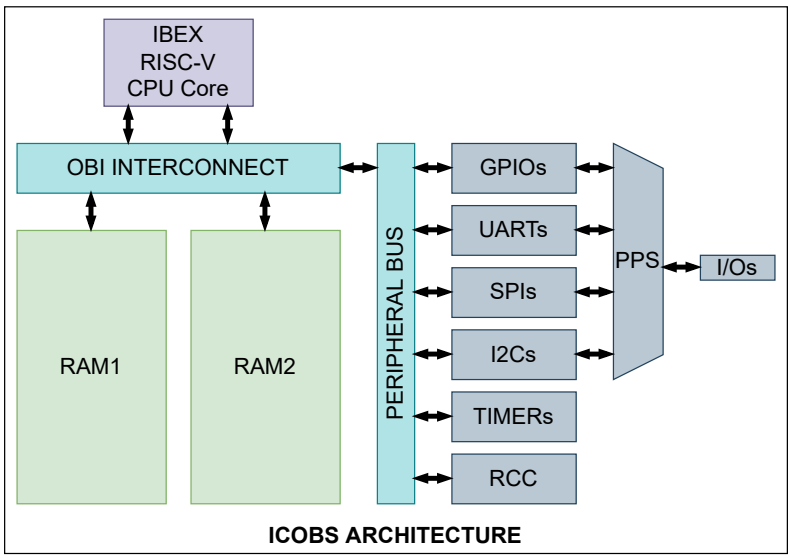


Figure 4.6: *ICOBS microcontroller architecture instance*

As mentioned earlier, there are a large number of RISC-V cores. 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 configurable and well suited for embedded IoT applications. Originally developed as part

of the PULP platform [118] under the name "Zero-riscy", Ibex is now maintained and developed by lowRISC [130]. It is currently under active development.

The Ibex core and memories are interconnected thanks to an OBI crossbar. The Ibex core integrates two different OBI 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 to the OBI 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. Furthermore, this solution has the advantage of being very flexible and allows easy integration of memories and peripherals.

Regarding memory sizing, the size and number of memories are configurable parameters. On the FPGA, the memories will be implemented with block RAM (BRAM). The Nexys video board integrates a Xilinx XC7A200T FPGA, which contains 1,625kB of BRAM. Thus, to allow MCU implementation, we limit the size of the memories to a total of 1,024kB. Nevertheless, our platform could easily be ported to a larger FPGA if higher capacity memories are required. In order to be able to run the most complex applications such as those integrating machine learning algorithms (needing several hundred kB), we chose to use the largest possible memories. We chose to use two 512kB memories that will incorporate the application code and data (RAM1 and RAM2). Note, however, that the memory distribution can easily be changed depending on the target application. The 1024kB are sufficient to store both instructions and data of a wide range of applications.

To interact with sensing and radio units to build a fully functional IoT node, we connect the OBI 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 dedicated MCUs. The peripheral bus embeds four UARTs, four TIMERs, two SPIs and two I2Cs. These peripherals will allow the proposed MCU to communicate with various I/O components. We also integrate a reset and clock control module (RCC). This peripheral controls the clock and reset signals of other peripherals. For GPIO management, we integrate a Peripheral Pin Select (PPS) module. This module is directly connected to the peripheral bus and allows mapping the output and input signals of the peripherals to any GPIO. The architecture can use up to four different GPIO ports. GPIO reconfiguration is necessary and a powerful tool when designing applications.

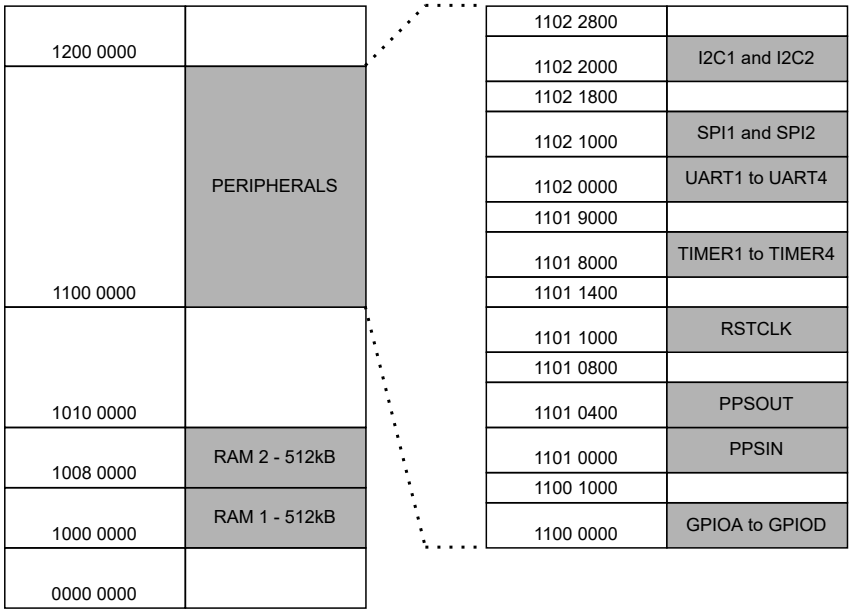


Figure 4.7: *ICOBS microcontroller memory mapping*

Figure 4.7 shows an example of MCU memory mapping. The interconnect decoders are setup as follows:

- RAM1 is mapped at the address 1000 0000_h with a 0007 $FFFF_h$ mask.
- RAM2 is mapped at the address 1008 0000_h with a 0007 $FFFF_h$ mask.
- The peripheral bus is mapped at the address 1100 0000_h with a 00FF $FFFF_h$ mask.

Peripherals' address mask is 00FF $FA00_h$ and their register address mask is 0000 $03FF_h$. At the ibex core level, the boot address is 1000 0000_h, so the core automatically boots into RAM1, where the application code is located. The whole system is clocked by a 50MHz system clock.

Next to the architecture, we also provide a set of libraries allowing to build applications targeting the developed MCU. The core library embeds various assembler macros to easily access ibex Control and Status Registers (CSR) and manage its interruptions vector. This library provides macro to mask/unmask ibex interruption or any interrupt line individually. The peripheral libraries describe the structure of each peripheral and allow to easily address the registers of each peripheral. We also provide a set of drivers. These drivers use the peripheral libraries in functions to properly use each peripheral. For example, for communication peripherals, these drivers include functions to initiate communication, send and receive data. A custom linker script is used to describe memory organisation and instruction and data mapping in RAM1 and RAM2. Finally, we provide a custom makefile allowing to generate application binary from source codes in C or C++, the libraries and the linker script. This makefile uses the RISC-V toolchain configured with the same extension as the ibex core for cross-compilation.

In summary, we propose a flexible MCU architecture based on a RISC-V core which is similar to the STM32L072 from STMicroelectronics. The proposed architecture does not contain any hard macros allowing exploration at any level. With its large memory and its OBI crossbar for optimal memory access management, the ICOBS architecture is designed to support a wide range of applications, including edge computing ones. In addition, we propose an associated application development environment allowing easy application design. The ICOBS architecture is designed to integrate a wide range of applications thanks to high-capacity memories. However, this architecture is not instrumented yet. In fact, it does not allow the collection of all the parameters described above to estimate the consumption of the MCU and the node.

4.3.3 Non-intrusive activity monitoring

Our goal is to measure the energy of the node and MCU components during the execution of an application. To do this, we need to evaluate their activity. To do so, we monitor T_{sens}, T_{proc}, T_{com}, T_{inact} the total duration in each phase. Concerning memory energy, we also monitor the number of byte read and write n_{rd} and n_{wr} in RAM1 and RAM2. To achieve this goal, we integer in the architecture a truly non-intrusive monitoring unit for real-time evaluation. This monitoring unit is responsible for application boot loading, MCU control and real-time MCU monitoring. Figure 4.8 describes the proposed instrumented architecture.

The monitor unit architecture is composed of a simplified version the ICOBS MCU with only the components needed to perform the control and monitoring tasks. We use another Ibex Core connected to two memories and a peripheral bus through an OBI crossbar. The peripheral bus contains one RCC, one timer, one UART and the monitor that contains the counters. Concerning the memories, ROM0 is a 16kB read-only memory which contains the monitoring unit code. This memory cannot be written during execution. Its content is initialised when generating the bitstream. The RAM0 is a 8kB ram simply used for monitoring application data storage. The monitoring unit is connected as a master in the MCU crossbar allowing it to access MCU memories and peripherals. Figure 4.7 shows the resulting monitoring unit memory mapping arround the MCU:

- ROM1 is mapped at the address 0800 0000_h with a 0000 $3FFF_h$ mask.
- RAM0 is mapped at the address 0800 4000_h with a 0000 $1FFF_h$ mask.
- The peripheral bus is mapped at the address 0900 0000_h with a 00FF $FFFF_h$ mask.

After reset, the monitoring unit enters its first operating mode: the bootloading. This process allows the MCU's RAM1 content to be modified at startup. This allows

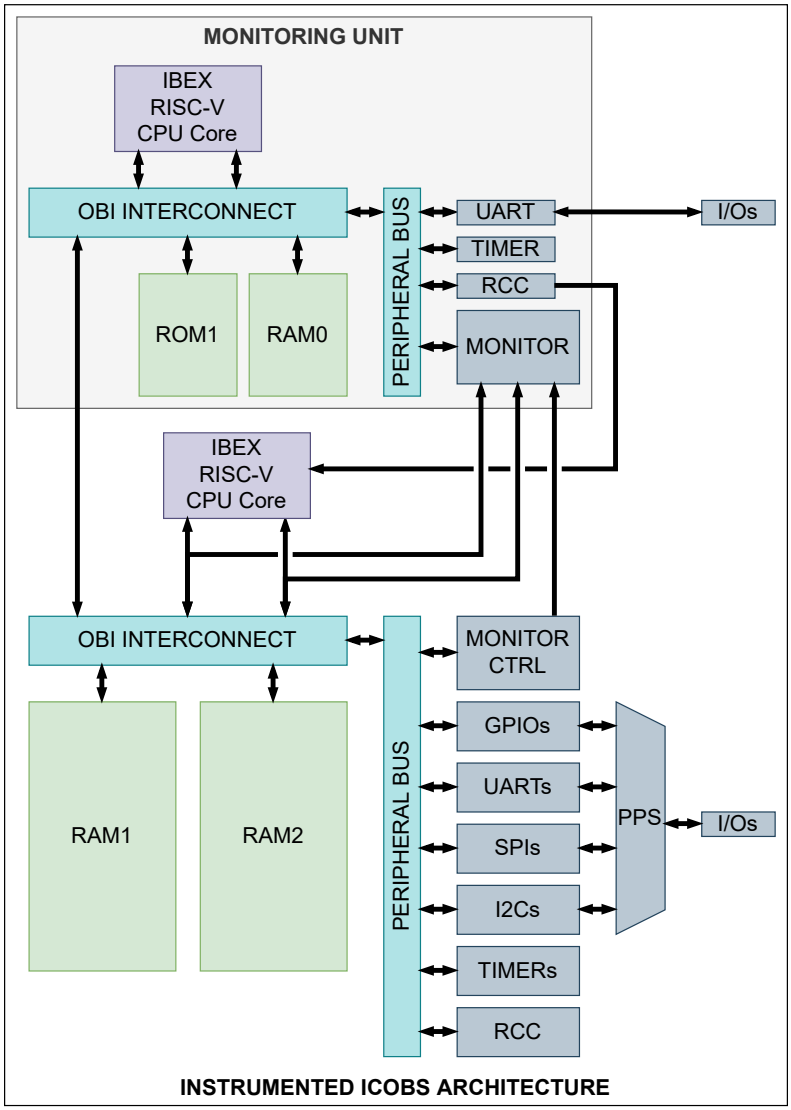


Figure 4.8: *Instrumented microcontroller architecture*

the application to be modified easily without having to redo the synthesis and implementation of the FPGA target each time. The monitoring unit waits for the reception of the application binary via its UART To do this, we propose a code loader that allows the binary to be sent via the COM port of the user's computer. The monitoring unit receives the instructions from the application and stores them in RAM1. Once all the instructions have been transferred to RAM1, we simply start the monitor, reset its counters to make sure they are empty and enable them. Then, the monitoring unit will start the MCU by setting the fetch enable signal of the MCU core. Before application execution, the MCU confirms its startup by setting the ready signal between the monitor control and the monitor. This allows the monitor counters to start.

To monitor the total duration in each phase: the monitor integrates eight user counters. These counters are controlled in the application by writing a register in the monitor control using simple assembler macros to be as non-intrusive as possible. In the

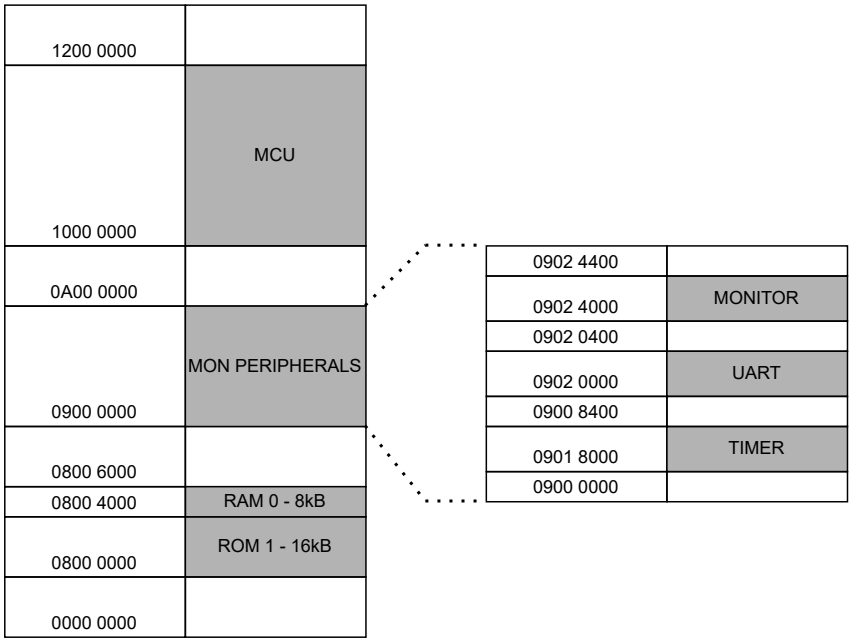


Figure 4.9: *Monitoring unit memory mapping*

application, we can easily start a counter and stop it once a task is completed. Then we can use these counters to track the activity of sensors, CPU core and communication module (T_{sens}, T_{proc} and T_{com}). In addition, the IBEX core includes a signal allowing the monitor to track the time spent in run and sleep mode, allowing to get T_{inact}. To monitor the number of read and write accesses n_{rd} and n_{wr} in RAM1 and RAM2, we position a probe at each MCU crossbar master interface. As all transactions are initiated by the masters, we can track all of them 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. In this way, we can easily monitor the overall activity of each memory and peripheral. The activity of a memory is defined by the number of reads and writes on 8, 16 and 32 bits. Of course, the monitor can easily be extended to monitor other relevant run-time activity.

Once the MCU is started, the monitoring unit enters its second operating mode: counters tracking. In this mode, the monitoring unit periodically sends the monitor counter values via its UART peripheral. For this purpose, a timer is configured to generate a burst in a monitor register every 250ms. This burst is used to synchronise the copy of each counter into dedicated registers. Once copied, the contents of these registers are transmitted via the UART. Thus the counters are not interrupted. The snapshots have no influence on the execution of the application in the MCU and on the value of the counters, making the monitoring truly non-intrusive.

4.4 IoT applications

We now propose two IoT applications to demonstrate our evaluation platform. These applications are designed to represent a wide range of IoT applications, which are often encountered in modern day-to-day life. The objective is to bound the application spectrum, ranging from applications that require low computing and storage to applications that require high computing and storage but are still suitable for the studied architecture. First, we propose a light application that handles small bits of data in a periodic and low-frequency interval, requiring no or low computational workload followed by a minor use of communication effort. Then, we propose a heavy application dealing with a large amount of data at high frequency, involving a substantial computational workload and may require heavy communication needs. We propose two different scenarios for each application. Scenario 1 follows a traditional cloud computing approach, while scenario 2 is based on edge computing to explore communication/processing tradeoffs. We then evaluate these applications with the proposed methodology to identify energy bottlenecks.

4.4.1 Light application

The first application represents classic IoT applications based on the deployment of simple low-end nodes. We propose an application that can control the lights according to the amount of natural light available in the place. A light sensor is responsible for acquiring the amount of light in the room, this value is then used to calibrate the brightness level of the light bulbs. The luminosity acquiring process occurs each second resulting in $f_s = 1Hz$, whereas the calibration routine depends on the scenario. For node communication, we chose LoRa for its range and because this technology is widely used in state-of-the-art wireless sensor nodes. As illustrated in Figure 4.10, in both scenarios, a value is sent by the end-node and received through a LoRa gateway by the server, which is responsible for adapting the brightness of the light bulbs sending the value over a Zigbee gateway. The end node platform embeds a light sensor based on a silicon NPN epitaxial planar phototransistor (TEMT6000X01) with reading ensured by an ADC via SPI protocol (ADC081S021). Concerning communication, we use the SX1272 LoRa module, which also has an SPI interface.

The first scenario is depicted in figure Figure 4.11a. The program first executes the pin configuration and sets a 1 second period timer. Then, the MCU remains inactive, waiting for a timer interrupt to launch the sensing task. During this phase, the end node reads the 12-bit luminosity value from the TEMT6000X01 through the ADC081S021 via SPI giving $S_{sample} = 2B$. Once the sensing task is done, the LoRa

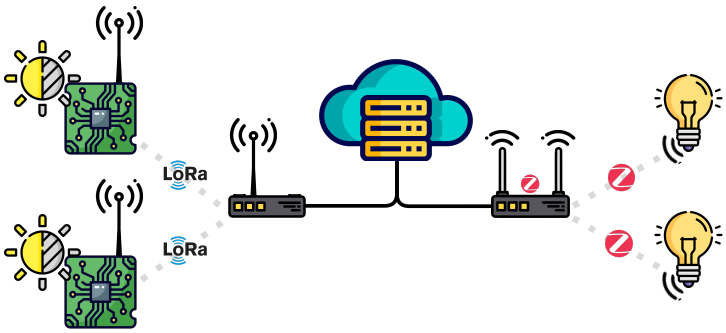
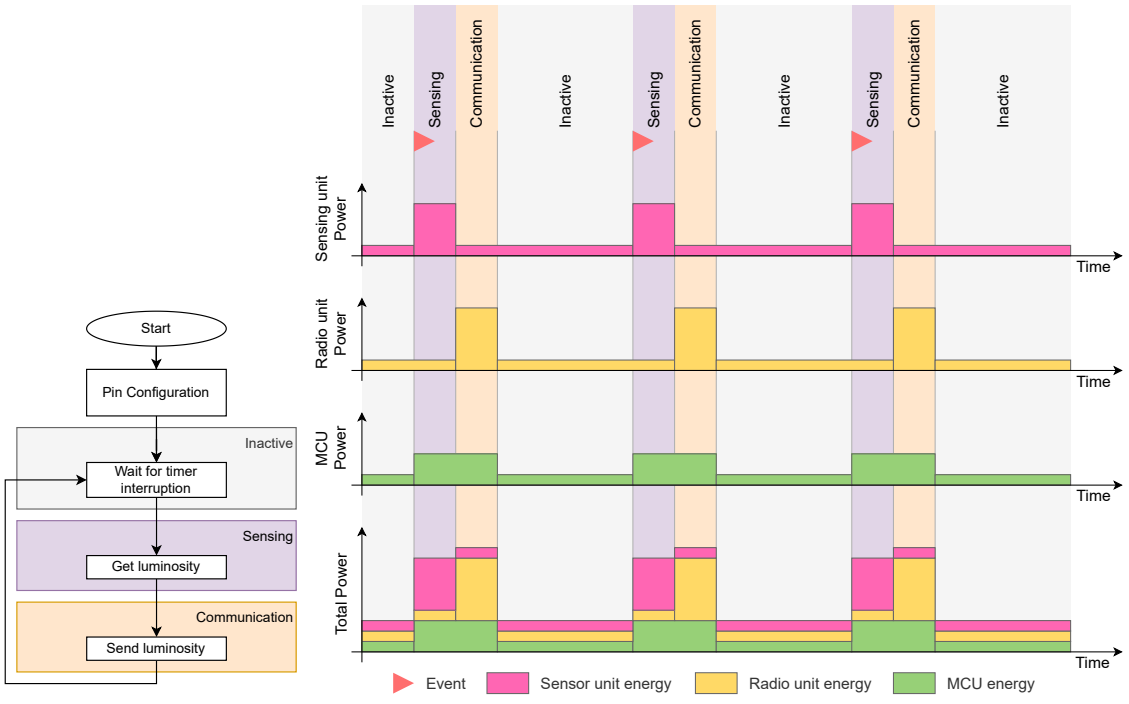


Figure 4.10: *Light application network setup*

module is initialised and used to transmit the luminosity value in two 8-bit packages to the gateway. Finally, the MCU enters inactive mode. The luminosity value is received by the server, which passes it through a first-order filter and calculates the equivalent amount of light brightness needed. The new brightness value is then sent to the light bulbs via the Zigbee gateway. In this scenario, the data treatment is done by the server, hence, all the data collected must be transmitted to it. This results in a periodic use of both the sensing unit and LoRa module, as shown in figure Figure 4.11b.

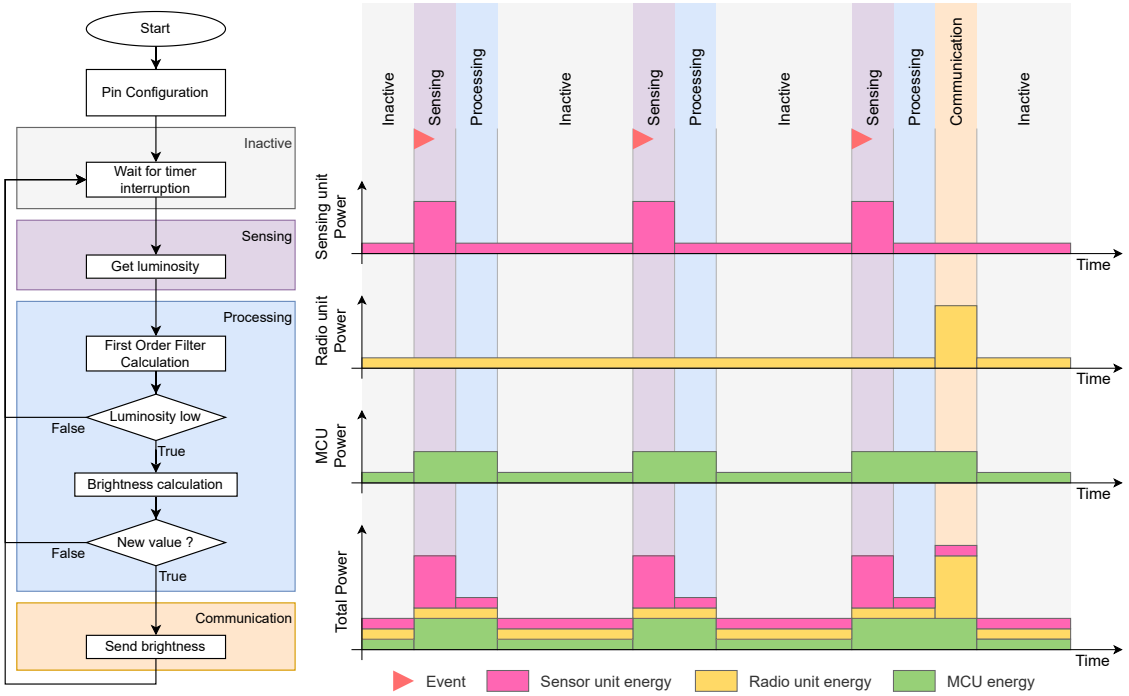


(a) *Flowchart* **(b)** *Energy model*

Figure 4.11: *Light application execution flowchart and energy model for cloud computing scenario*

The second scenario is depicted in Figure 4.12a. After pin and timer configuration, the MCU remains inactive, waiting for an interrupt to launch the sensing task. Once done, the end node processes the 12-bit luminosity value through a first-order filter. The program then calculates the new brightness command: if this value is different

from the last value sent, the communication task starts. LoRa is used to send an 8-bit brightness value instead of the 16-bit luminosity value in the first scenario. On the server side, the brightness value is simply forwarded to the light bulbs via the Zigbee gateway. Since the data processing is done by the end node itself, the communication does not have to be done after each timer event. This results in a much less frequent awakening of the LoRa module, as shown in figure Figure 4.12b.



(a) *Flowchart* **(b)** *Energy model*

Figure 4.12: *Light application execution flowchart and energy model for edge computing scenario*

4.4.2 Heavy application

The second application represents an edge computing application based on the deployment of sensor nodes for event detection and identification. These nodes collect continuous data such as motion or sound and are able to detect and identify particular events in the data stream generated by the sensing unit. These applications are, for example, used for anomaly detection. The proposed application can handle the lights over voice control, allowing the user to turn them on or off using two different keywords. A microphone is responsible for acquiring ambient sound. If a word is detected, then it is used as input for a machine learning inference engine that returns if one of the two keywords was detected. Depending on the keyword detected, the application turns the light(s) on or off. The buffering of the audio samples is 1 second long.

As illustrated in Figure 4.13, in both scenarios, the server is responsible for changing the state of the light bulbs sending the value over a gateway using Zigbee protocol. A Low Noise Zero-Height SiSonic Microphone is used on the node prototype. The sensor consists of a MEMS microphone (SPA2410LR5H-B) with reading ensured by an A/D converter via SPI protocol (ADCS7476). LoRa is used for communications from the nodes to the network.

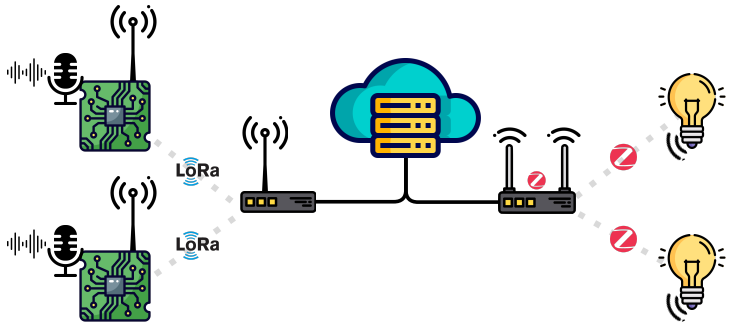
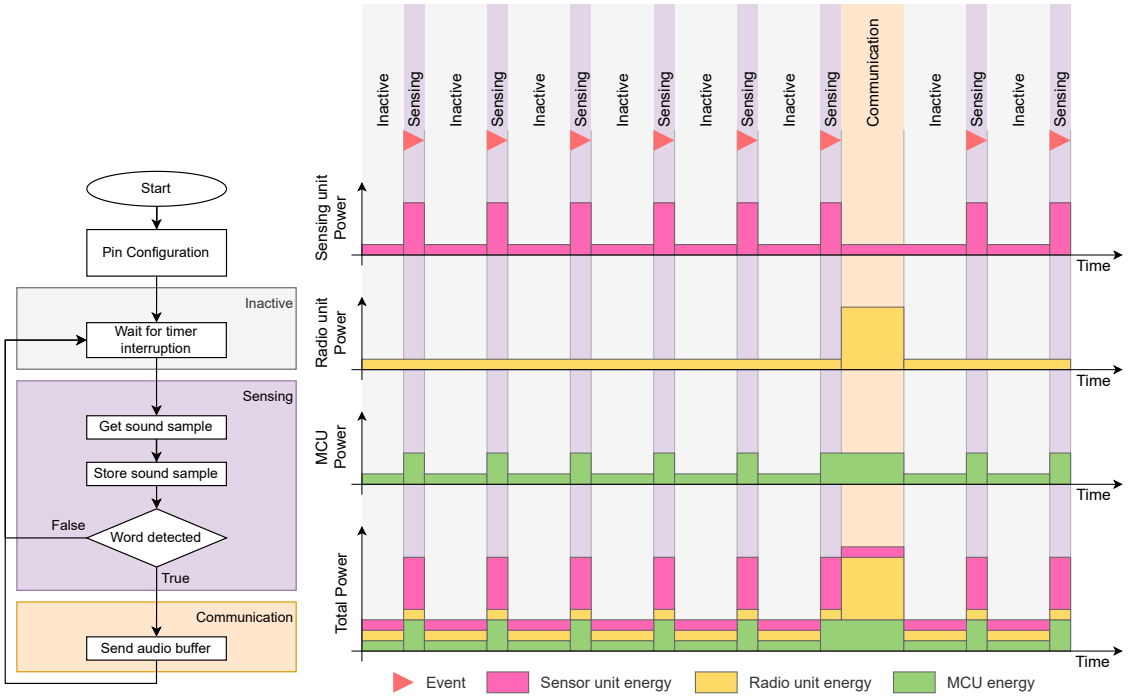


Figure 4.13: *Heavy application network setup*

The first scenario is depicted in Figure 4.14a. First, the program executes the pin configuration and configures a 62.5 microseconds period timer. Then the MCU remains inactive, waiting for an interrupt that launches the sensing task. During this task, the end node reads the 12-bit audio sample from the SPA2410LR5H-B microphone through the ADCS7476 via SPI generating a $S_{sample} = 2B$ with $f_s = 16kHz$. The sample is then normalised and stored in a one-second audio circular buffer. If there is a word detected in the buffer, the communication task starts. The 32 kB audio buffer is finally sent to the server over LoRa. On the server side, a Fast Fourier Transform (FFT) of the received audio buffer is executed to build a 49x50 spectrogram. Then, this matrix is used as an input of a convolutional neural network (CNN) model whose inference is performed with the TensorFlow framework: it returns the detection probability of each keyword. Depending on the keyword detected, the server determines the command value (ON or OFF) before transmitting it to the light bulbs via the Zigbee gateway. In this scenario the data treatment is done by the server, hence the whole audio buffer has to be sent through LoRa. This results in a very long period of use of the radio module, as shown in figure Figure 4.14b, each time a word is detected.

The second scenario is depicted in figure Figure 4.15a. After word detection, the whole processing is handled by the MCU, including the FFT (with the KissFFT library), and the inference (with TensorFlow Lite for microcontrollers and a quantified CNN model). After this last step, the score of each keyword is stored in a table. The LoRa module is then used to transmit the command relative to the keyword. On the server side, the command value (ON or OFF) is simply forwarded from the LoRa gateway to the light bulbs via the Zigbee gateway. Since the data processing is done by the end node, this results in a very long period of active processing as shown in Figure 4.15b.



(a) *Flowchart* **(b)** *Energy model*

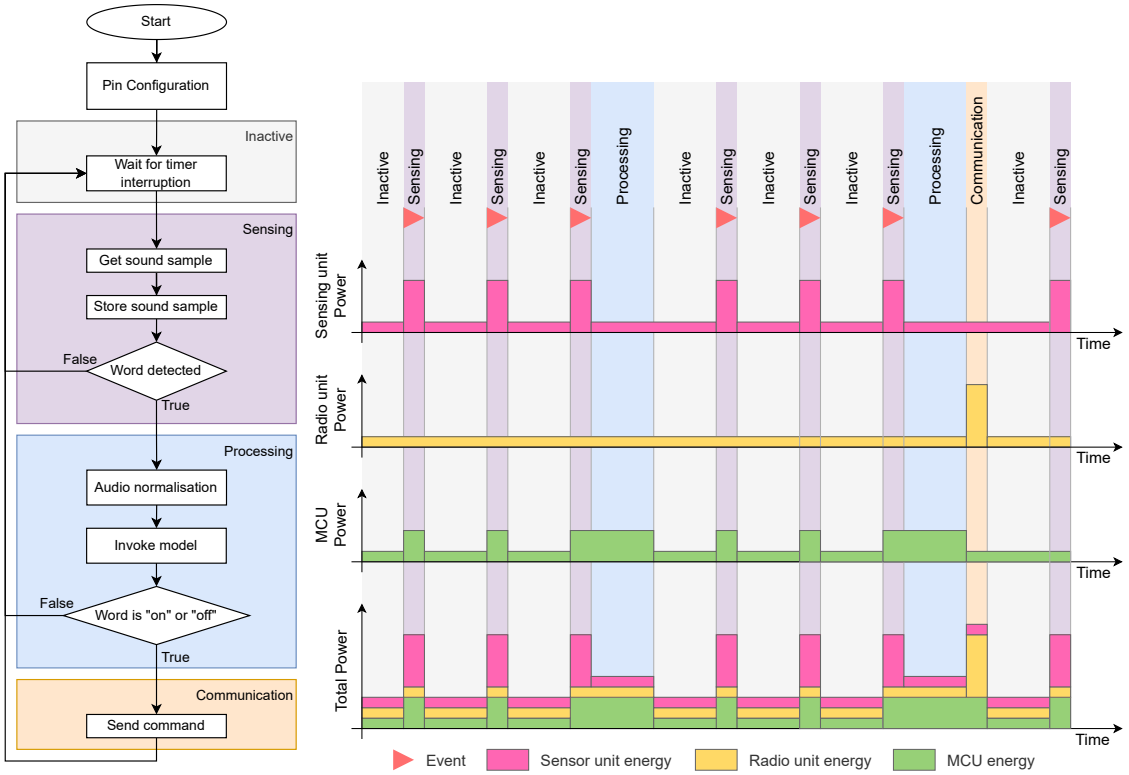
Figure 4.14: *Heavy application execution flowchart and energy model for cloud computing scenario*

4.5 Energy evaluation

The objective is to explore the computation/communication tradeoffs for the two proposed applications by comparing the cloud and edge computing approaches, as well as the resulting component's consumption. To do this, we follow the steps ② and ③ of the methodology proposed in Section 4.2. First, we build the power model of each component at both node and MCU levels (step ②). Then, we evaluate and study the energy consumed by the different components of the node for both applications (step ③).

4.5.1 Power models

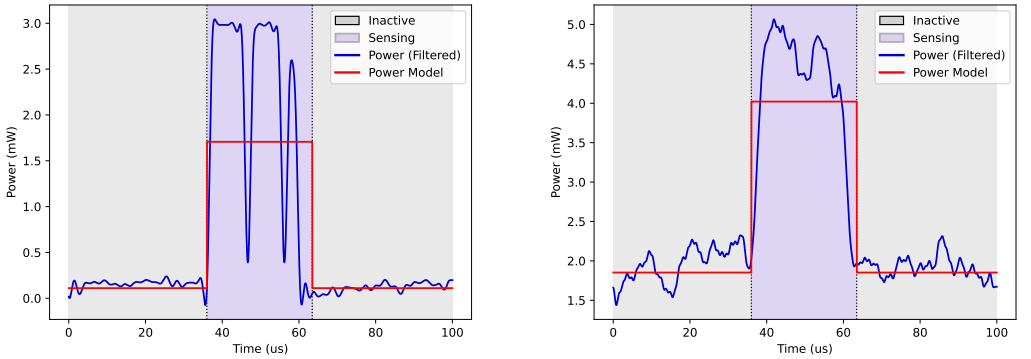
For each application, we now proceed series of tests to build the power models of each component. To build sensing and radio unit power models, several test runs were performed using a Nexys Video board based on the Artix-7 FPGA, in which the Instrumented ICOBS MCU is implemented. We ran several test-benches on the platform. These test-benches are intended to replicate the usage of each input and output component, with different triggers to track their different operating modes. During active mode, the application either makes a request or ships information to the compo-



(a) *Flowchart* **(b)** *Energy model*

Figure 4.15: *Heavy application execution flowchart and energy model for edge computing scenario*

nent. Meanwhile, during inactive mode, no requests or shipments are made, leaving the component in an idle state considered inactive. The signal measurements were acquired using an oscilloscope equipped with a current probe. The measures are then analysed with the help of a dedicated Python script. We finally estimate the average current drawn by each unit in each one of its operating modes (P_{act} and P_{stdb}).



(a) *Luminosity sensor power characterisation* **(b)** *Microphone power characterisation*

Figure 4.16: *Sensor power characterisation*

Figure 4.16a and Figure 4.16b illustrate the power model construction of the light sensor and the microphone respectively. Each activation of the light sensor involves

awakening the ADC, making a sample request and then shutting down the component through SPI. The converter is shut off and awakened at each iteration to improve power consumption during the inactive state. With the ADC and the phototransistor we obtain an average P_{act} of 1.707mW during the sensing phase and an average P_{stdb} of 0.110mW. Because the microphone is connected to an ADC of the same kind, the active state follows the same steps as the light sensor. The average P_{act} and P_{stdb} of the ADC and MEMS microphone set are 4.021mW and 1.852mW, respectively. These components are used intermittently, so with dedicated circuitry, it is possible to control their power supply via an MCU I/O pin so that P_{stdb} can be considered as zero.

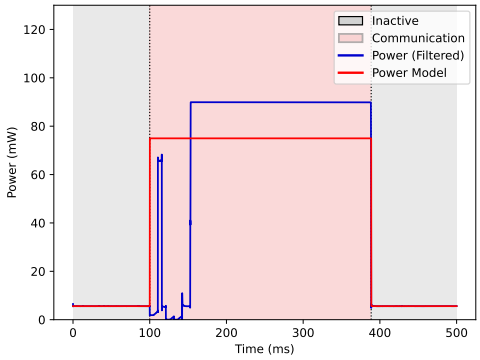


Figure 4.17: *LoRa module power characterisation*

Figure 4.17 illustrates the power model construction of the LoRa module. An activation in this module comprises waking it up, making a radio transmission of a package and shutting it down, everything done via SPI. The duration of the active state is determined by the size of the package. For this test, we send a two-byte long package, taking about 288.661ms to be send. We obtain an average P_{act} of 74.972mW during the communication phase and an average P_{stdb} of 5.601mW. In the same way as for the sensors, it is possible to control the LoRa module power supply via an MCU I/O pin so that P_{stdb} can be considered zero.

For MCU energy modelling, the energy of the CPU core and interconnect circuits is modelled as a function of their operating mode in the same way as the sensing and radio unit with an average of P_{act} and an average of P_{stdb}. Concerning the memories, we model a memory bank energy with its static power consumption depending on their operating mode P_{statOn} and $P_{statOff}$ for on and off mode, respectively. We also take into account memory dynamic energy for each read and write operation (E_{rd} and E_{wr}).

To evaluate P_{act} and P_{stdb} of the CPU core and interconnect, we use a test bench and validate the correct operation of the final architecture by simulation using ModelSim tool. Once the complete MCU architecture has been validated at HDL description level and gate level after the digital synthesis, the power consumption flow can be set

up. First, it is necessary to generate the .sdc (Synopsys Design Constraint) file and the .sdf (Standard Delay Format) file, both from the Synopsys Design Compiler Synthesis tool. Then, the power consumption estimation has to be defined on a specific time frame, from the beginning to the end of the test bench execution for instance. Therefore, during the gate-level simulation of the Verilog netlist, it is essential to generate an activity file, the .vcd (Value Change Dump) file, that provides for each signal its transitions. Finally, using all these generated files, the power consumption evaluation can be performed with the Synopsys Prime Power tool. It enables to extract data such as Net Switching Power, Cell Internal Power, Cell Leakage Power and Total Power. Finally, Prime Power enables to generate a .fsdb (Fast Signal Data Base) file that provides graphical visualisation capabilities. We obtain for both CPU core and interconnect an average P_{act} of 0.59mW and an average P_{stdb} of 0.24mW. It is important to note that in this evaluation, we do not consider any clock gating mechanism during inactive phases. Thus the value of P_{stdb} used in the rest of the manuscript is over-evaluated.

Table 4.4: *SRAM memories high-level energy models*

Size [kB]	8	16	64	512
E_{rd} [J/B]	5.60e-12	5.90e-12	7.70e-12	2.46e-11
E_{wr} [J/B]	5.08e-12	5.22e-12	6.04e-12	1.37e-11
P_{statOn} [W]	1.13e-05	1.32e-05	2.46e-05	1.30e-04
$P_{statOff}$ [W]	1.01e-05	1.09e-05	1.59e-05	6.17e-05

Table 4.4 presents the SRAM memory models for different memory sizes. We first extract the model for two 28nm FD-SOI SRAM (16kB and 64kB) built with the single-port high-density compiler supporting forward body bias (SPHD-BODYBIAS) from STMicroelectronics. As SRAM is a volatile memory, it cannot be completely shut down, however there is a standby mode that ensures the retention of data while reducing static consumption. This retention consumption is used as $P_{statOff}$. Finally, we use linear extrapolation to build models of memory with sizes adapted to our applications (8kB and 512kB). For the light application we suppose a 2x8kB SRAM architecture while for the heavy application, we suppose a 2x512kB SRAM architecture. Once the power models of all the components are complete, we move to step ③ for both applications.

4.5.2 Light application energy analysis

To do an energy analysis on the light application, the platform was set up in order to gather both monitor and application data. Following the Figure 4.18, the whole light application network is set up with our FPGA-based prototype connected to a computer as a sensor node. In addition to its normal tasks during application execution, we add to the server the storage of all commands received and transmitted in a log file.

- ① : First, we load the application binary in MCU memory.
- ② : Then, we start the server.
- ③ : The application runs for 1 day
- ④ : Finally, the monitor data is sent to the computer and stored in a CSV file.

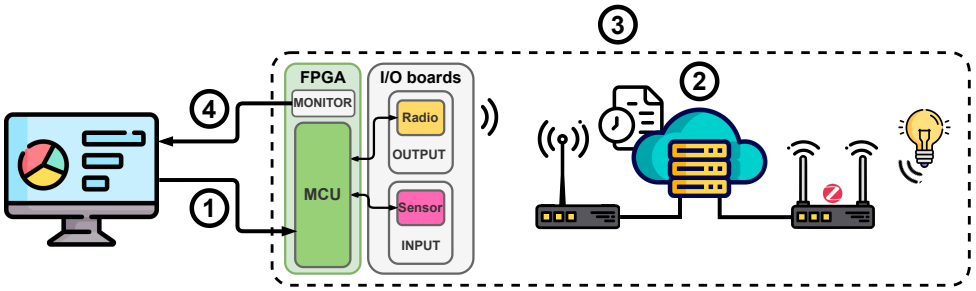
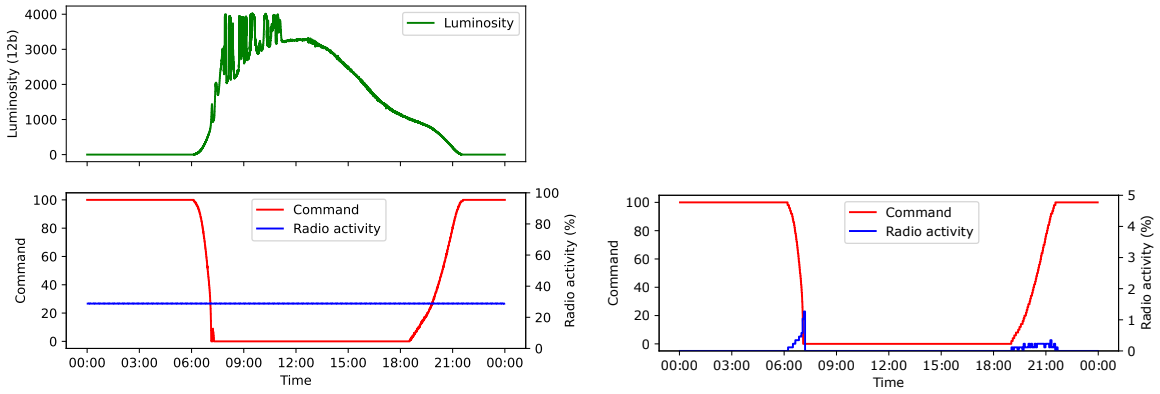


Figure 4.18: *Experiment setup*

Figure 4.19 contains the server log during two different days of the experiment, each with a different scenario. For the cloud scenario, we can see in the Figure 4.19a top graph the evolution of the ambient luminosity during a day. In the bottom graph, we can see the associated brightness command calculated by the server and the radio activity during the day. As in this scenario luminosity value is transmitted every second by the node, we notice that the radio activity is constant and high (around 30% for 250s windows). For the edge scenario, the ambient luminosity is not transmitted to the server. We only have as information the brightness command that has been forwarded and the radio activity, as we can see in the Figure 4.19b bottom graph. We observe that in this scenario, the brightness command is transmitted only when required, reducing the radio activity a lot. If we compare the command values of the two scenarios, we can see that they are very similar. This verifies that both scenarios have been programmed to calculate the command value in exactly the same way and therefore provide the same service.



(a) *Cloud computing scenario* **(b)** *Edge computing scenario*

Figure 4.19: *Light application running one day*

The monitor log over the period of 1 day allows comparing the cloud and edge computing scenarios at the node level. Figure 4.20 shows the time distribution between tasks during the day. Over the experiment, the end-node stays inactive most of the time in both scenarios, where also the time in processing and sensing tasks is negligible. Nevertheless, the time spent by the end-node in the communication task is 28.8% higher in the cloud computing scenario, reaching almost a third of the experiment time. On the other hand, the edge computing scenario leaves the end-node unused for more than 99.9% of the time.

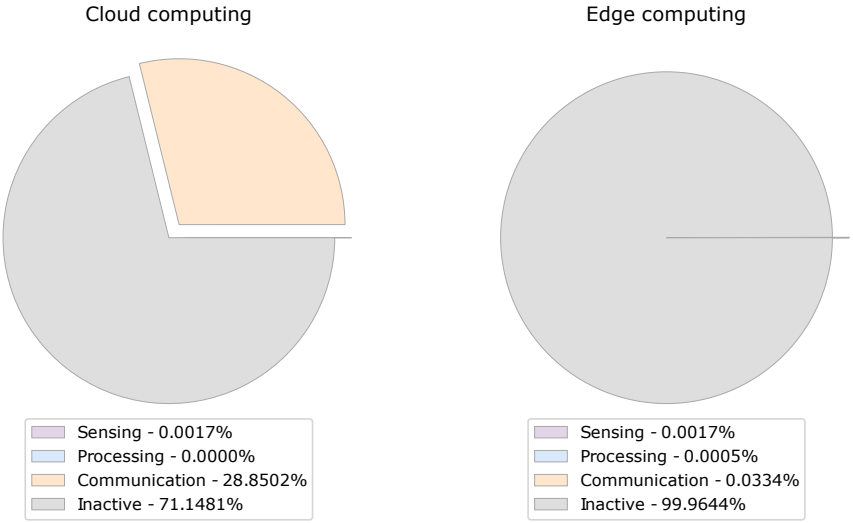


Figure 4.20: *Time sharing between different tasks for both scenarios*

First, using the energy consumption model of a sensor node and the power models defined previously, we evaluate the consumption of the nodes during the experiment for both cloud computing and edge computing scenarios. This leads to the following graph in Figure 4.21, where in the cloud computing scenario, the total energy used by the node is about 1921J. However, in the edge computing scenario, the total energy used by the node is around 24J. In comparison, in the cloud computing scenario, the node uses 80 times more energy than in the edge computing scenario. If we consider a Lithium Polymer battery with a 3.7V voltage and a 1,800mAh capacity, we can compare the estimated autonomy of the node for each scenario. We estimate an autonomy of 12 days for the cloud scenario, while for the edge scenario, we estimate an autonomy of 980 days.

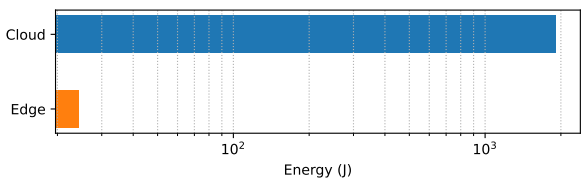


Figure 4.21: *Total node energy for both scenarios*

Figure 4.22 shows the energy breakdown in the node between the sensor, the radio

and the MCU. In accordance with the great communication task time, the cloud computing scenario spends more than 97% of its total energy on the radio module. This accounts for 88.5% more than the edge computing scenario, which spends less than 9% of its energy in this module. The energy consumption of the MCU dominates in the edge computing scenario, representing more than 90% of the total energy consumed by the node.

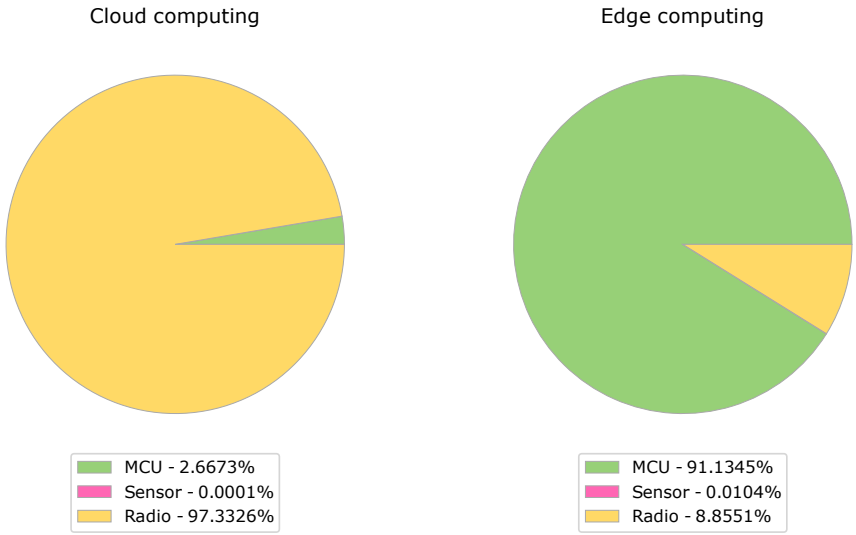


Figure 4.22: *Energy sharing between different components in the node*

Figure 4.23 illustrates the power distribution in the MCU for both scenarios. We can differentiate the consumption of the CPU core and the memories in both active and inactive phases. In the cloud scenario, the active energy is dominant (more than 69% of the overall MCU energy). This is due to the high activity rate of this scenario. However, in the edge scenario, due to the very low activity rate, we observe that the energy in the inactive phase widely dominates the MCU energy (more than 99.8%).

To summarise, for simple applications such as the light application, it is possible to add very little computing to communicate when necessary. The edge scenario minimises the energy consumed by the radio, reducing the power consumption of the node (98.7% reduction for the light application) and therefore increasing its autonomy. In addition, this approach greatly reduces the activity on the IoT network and therefore the energy used to transport the data to the server. In this case, we observe that the majority of the energy consumed by the node is directed around the MCU during the inactive phases.

4.5.3 Heavy application energy analysis

To do an energy analysis on the heavy application, we analyse the output of the monitor, ensuring that for both scenarios, the application is working properly and the ser-

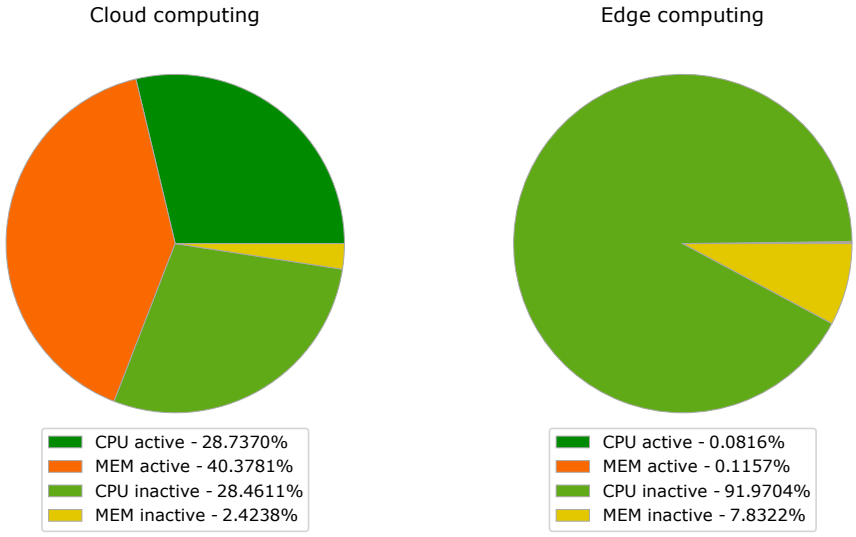


Figure 4.23: *Energy sharing in MCU for both scenarios*

vice is provided. In this application, sensing is very frequent and continuous in both scenarios. It is only when a word is detected that there is a difference between the two scenarios. Sensing can thus be seen as a background task. In the cloud scenario, after each detection, the whole audio buffer is transmitted directly via radio. In the edge scenario, after detection, the whole audio buffer is passed through the TensorFlow-lite keyword detection model, then, two possibilities:

- A keyword is detected, and then a simple command is sent via radio.
- No keyword detected, then the MCU return to the inactive state.

Firstly, we characterise the activity of the system after detection in the cloud scenario and for the two cases of the edge scenario. Both applications are run in the instrumented platform and different types of detection are generated. For the cloud scenario, a word is spoken to generate a detection with communication. For the edge scenario, an unknown word is spoken to launch a detection with only an inference from the model, and a keyword is spoken to generate a detection with inference and communication. For each of them, the output of the monitor is analysed.

Table 4.5: *Duration of one sensing and one processing task depending on the scenario*

	Cloud	Edge	
	(C)	(P)	(P)(C)
Processing time (s)	0	1.69	1.69
Communication time (s)	566.1	0	0.29

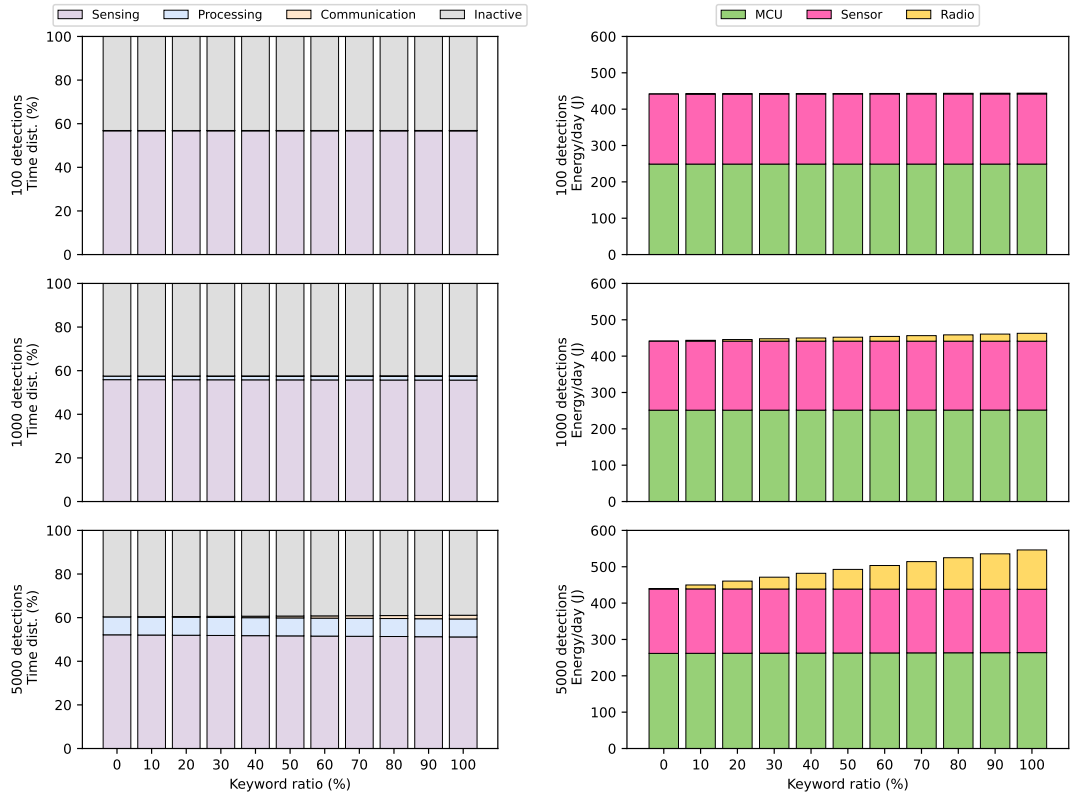
Table 4.5 shows the processing and communication task duration for both scenarios. For the cloud scenario, we can see that the communication task takes more than 9 minutes. This is because LoRa communication is not adapted to transmit so much data. Implementing such a solution would result in a high RF utilisation and a high collision probability in networks with numerous nodes. Furthermore, in the case of

our service, this has the effect of generating too much latency making LoRa not suitable for our application. It is possible to use more suitable radios such as Bluetooth or BLE to overcome this. However, these technologies offer a much lower range making them more complicated to deploy in a given environment. In summary, transmitting an audio buffer via an LPWAN radio technology such as LoRa is inappropriate. Therefore the cloud computing approach is not suitable for this application. Therefore, we will not further evaluate the cloud computing approach for the heavy application.

Concerning the edge scenario, the FFT and inference time is around 1.7s and constant in both cases. If a keyword is spotted, the communication task takes less than 300ms. In order to study the sensor node consumption for this application, we study the monitor output after several runs. During these runs, we generate a set of detection or none. The collected traces allow us to know the consumption of the system without any detection and to identify the contribution of each type of detection on the monitor counters. Thus by extrapolation, we can build up monitor traces for variable run times with different detection types. The energy of the node therefore depends on its environment and the probability of detection. Moreover, in the case of the edge scenario, it also depends on the keyword ratio. When a word is detected, this can result in the detection of a keyword or not. The keyword ratio is therefore defined as the proportion of keywords among the detected words.

Figure 4.24a and Figure 4.24b show the time distribution between tasks and the node energy with the contribution of different components for a different number of detection during the day and for different keyword ratios. We see that the node spends a lot of time in sensing task. Indeed each reading is done by the MCU and the sensor, a reading lasts 35.2µS and the readings are done every 62.5µs to ensure a sampling frequency of 16kHz. The activity rate of the node is therefore quite high, above 50%. The sensing phase is ensured by the MCU and the sensor, that's why on the right side graphs, we can see that the energy distribution between the different components is mainly oriented around the MCU and the sensor. We can also see that as the number of detections increases and the keyword ratio increases, the radio can have some impact on the node's power consumption. In the worst case (5000 detections per day with a 100% keyword ratio), the node autonomy is estimated to drop to 43 days.

In the same way as for the light application, Figure 4.25 shows the corresponding energy distribution in the MCU between CPU and memory and between energy during active and inactive phases. We can see that the energy of the MCU is basically due to its power consumption during the active phases and more precisely to the power consumption of the memories during these phases. This is due to the high cyclic ratio because sensing with CPU is not efficient. Indeed, it is not necessary to use a component as complex as an MCU for a task as simple as sensor data collection, data buffering



(a) *Time distribution of tasks* **(b)** *Node energy*

Figure 4.24: *Edge scenario with sensing task assured by CPU*

and word detection. Indeed it is possible to design a small peripheral dedicated to this task. It is possible to modify our model to include the use of such a peripheral by considering the MCU inactive during the sensing phases. Since in the monitor traces the sensing is done by the MCU, we have to remove the corresponding memory activity.

We can also see that in Figure 4.24b, the energy consumed by the sensing unit is also very high. Indeed for these experiments, the sensing unit used is a PMOD MIC3 which embeds an analog MEMS microphone (SPA2410LR5H-B) with reading ensured by an A/D converter via SPI protocol (ADCS7476). The resulting sensing unit consumes 4mW while active, which is very high. It is possible to use more energy-efficient microphones, such as the ICS-41351, that have a PDM (Pulse Density Modulation) output. Connecting this type of microphone directly to the MCU is possible by integrating a small device capable of deserialising the PDM signal. We use this type of microphone on the sensor and radio board presented in Section 4.3.1. This microphone consumes 90% less power compared to our initial solution.

The same studies have been carried out but now with the power model of the ICS-41351 PDM microphone and the sensing performed by a dedicated peripheral. It is important to note that we do not model the consumption of this peripheral, so we slightly underestimate the energy of the MCU.

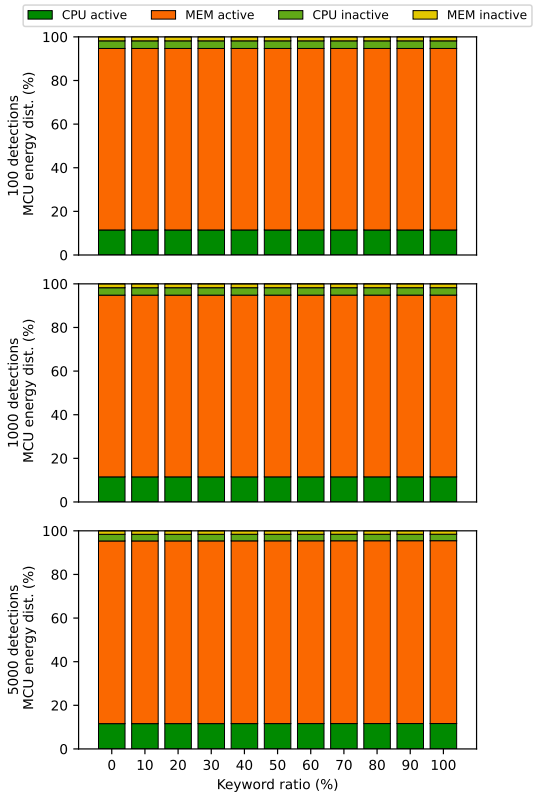
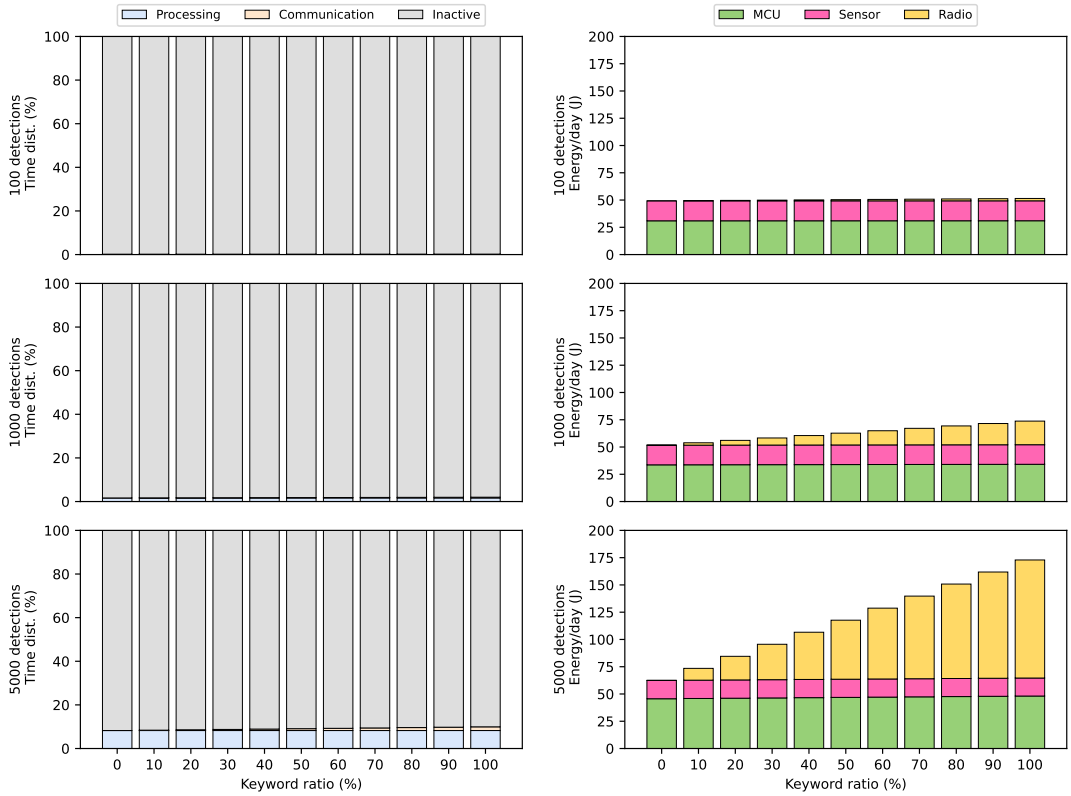


Figure 4.25: *Energy distribution in the MCU with sensing task assured by CPU*

In terms of time distribution, we can see that the system is much less active. Indeed, in this case, the activity rate is directly linked to the number of detections and the keyword ratio, as shown in Figure 4.26a. In Figure 4.26b, we can see that we reduced a lot the sensor energy. The MCU energy has been highly reduced as it is unused most of the time in this case. Finally, the impact of radio depending on the number of detection and the keyword ratio is now higher. The use of this peripheral, combined with a better microphone, reduces the consumption of the MCU and, therefore, of the sensor node. In the worst case (5000 detections per day with a 100% keyword ratio), the node autonomy rises to 138 days.

Figure 4.27 shows the corresponding energy distribution in the MCU between CPU and memory and between energy during active and inactive phases. We can see that the energy of the MCU is basically due to the power consumption of its CPU core and memories during inactive phases now. In cases where the system is most active (5000 detections/day), the energy consumed by the MCU during the inactive phases still represents around 60% of its total energy.

To summarise, for the heavy application, the edge computing approach not only reduces system energy consumption but also makes the application suitable. In this case, edge computing makes it possible to combine high-speed sensors and long-range radio. These applications make frequent use of the sensing unit, which is why much



(a) *Time distribution of tasks* **(b)** *Node energy*

Figure 4.26: *Edge scenario with sensing task assured by dedicated peripheral*

effort must be put into reducing the power consumption of this unit, as we did. Once we replaced the microphone with a more energy-efficient solution and considered the use of a dedicated peripheral for sensing, we managed to reduce the duty cycle of the MCU and therefore its power consumption. Radio consumption can become dominant if the number of detection is high and they all generate communication. However, in all other cases, we observe that the majority of the energy consumed by the node is directed around the MCU during the inactive phases. During the inactive phases, the CPU core consumption is slightly higher than the memory consumption. This is due to our over-evaluation when we built the CPU core power model. Indeed, we do not consider any clock gating mechanism during its power evaluation in inactive phases. If we consider clock gating on the Core CPU, then the memory consumption during inactive phases dominates the MCU consumption.

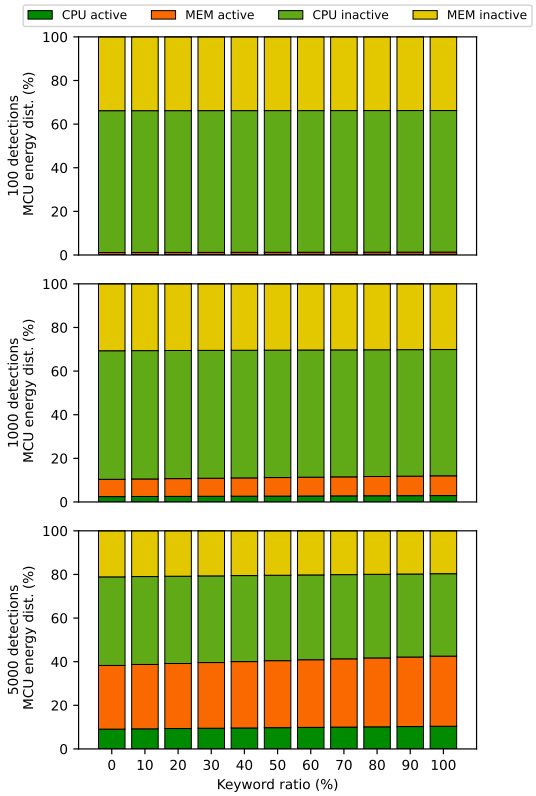


Figure 4.27: *Energy distribution in the MCU with sensing task assured by dedicated peripheral*

4.6 Summary

We first studied methods to evaluate the consumption of a sensor node following our application model. It was concluded that the fabrication of a prototype allows a real-time and real-condition evaluation. It can be combined with FPGAs to allow high flexibility and ease of exploration. Moreover, the use of FPGAs allows a lot of freedom in monitoring to retrieve the parameters of our model. We use the same methodology as Patrigeon et al. [26] but with an enhanced platform. The proposed FPGA-based node prototype embeds a state-of-the-art representative MCU architecture and several sensing and radio units. This architecture, combined with a set of libraries, allows to design a wide range of applications based on our model. We also designed a monitoring unit that can easily modify the contents of the MCU's memory to make the application easier to use. However, the main mission of this monitor is to retrieve all the parameters of the model in real-time, being non-intrusive during the execution of the applications. To test our platform, we developed two applications ranging from a very simple application generating few data to a much more complex application based on the use of sensors that generate a lot of data. For each application we explore the tradeoff between communication and processing at the edge. We developed, for each of them, two scenarios, one based on cloud computing and one based on edge computing. Then we deployed the whole IoT network for both applications with our

FPGA-based platform as a sensor node to analyse its consumption and identify energy bottlenecks.

For each application, the energy of the node was evaluated in both scenarios before showing the energy distribution at the node level and at the MCU level. It was found that an edge approach reduces the energy consumed by the node as it reduces the utilisation of the radio, which is generally the component with the highest power consumption. For the heavy application, the edge computing approach is not only a plus, but it also makes the application suitable. Regarding the power distribution in the node, not surprisingly, the majority of the power is consumed by the radio in cloud scenarios. However, in the case of edge computing, we notice that in this scenario, the system becomes low duty-cycled. This is due to the fact that even very intensive processing is often much faster than sending data via radio, especially when the radio technologies offer low data rates and there is a lot of data generated by the sensor(s), as it is the case in the heavy application. So we notice that in the case of edge computing, the energy consumed by the MCU is not negligible and can sometimes represent a large part of the total energy consumed (up to 73%). Moreover, if we look at the energy distribution in the MCU in the case of edge computing, we notice that most of the energy is consumed during the inactive phases because of the very low activity rate (still around 60% in most active cases). For this reason, the integration of emerging NVMs for energy-saving strategies will be explored in the next chapter.

V

Memory exploration

Contents

In the previous chapter, we used an FPGA-based platform to evaluate several sensor node applications by comparing for each of them a cloud computing and an edge computing scenario. We have shown that in the edge computing context, the MCU is often the most energy-consuming device in the node. Then, we have shown that the majority of the energy consumed by the MCU is consumed during inactive phases by its memories. However, in Chapter 3, we demonstrate how emerging NVMs can be integrated into the architecture of MCUs to build NV-MCUs to save energy during inactive phases. In this chapter, we first motivate the need to explore the memory architecture of NV-MCUs in an edge computing context. Then, we present MemCork, a tool for NV-MCU memory architecture exploration in intermittent computing devices at the edge. Based on an instrumented execution of the application on the technology-agnostic FPGA prototype, our tool exhaustively explores the possible data mapping and memory architecture combinations to find the most energy-efficient solution.

5.1 Motivation

As seen in Section 4.3, MCUs are composed of a set of memories used to store instructions and data. As the amount of instructions and data depends on the application, it results in different memory capacity requirements for different software source codes. Instructions and data have different purposes, which means that they are not used in the same way during the execution of the application. In addition, the memory architecture may combine different technologies with different properties. Thus, for a given application, there are many possible solutions combining different instruction/data mappings in different memory technologies, resulting in many options for memory sizing.

In a system where several instruction and data sets can be placed in several memory technologies, finding the optimal solution is not straightforward. Moreover, in the case of event-based applications, access to instructions and data depends strongly on the event probability, which depends itself on the environment. Thus, event probability and the nature of the event may affect the optimal memory solution and data mapping and, therefore, the resulting memory sizing.

Memory architectures in the state-of-the-art NV-MCU are most of the time based on a single non-volatile memory except for some works that also embed an additional SRAM volatile memory to build a hybrid memory architecture (see Chapter 3). However, most of these works target energy harvesting based systems and thus backup must be very fast and energy effective. The use of hybrid memory architecture, in this case, is sometimes not suitable because the backup time of the memory content is too

long to be considered in such applications. This is why most solutions are based on a non-volatile memory in which all instructions and data are stored allowing an implicit backup of the memory after each access. We also notice that in most cases, these works use small memories (<64kB). This is due to the fact that in most cases, these works target relatively simple applications.

In summary, for a given application, there are many possible solutions that are difficult to evaluate because the node environment can affect their energy efficiency. State-of-the-art solutions target simple applications with tight constraints on backup and restore. It is difficult to generalise these solutions to more complex applications (*e.g.*, edge computing) where there are relatively low constraints on the backup and restore phases. For memory and CPU-intensive event-driven applications, there is a real trade-off between backup and NV overhead which also depends on the environment motivating further explorations.

5.2 MemCork

As the design of the memory architecture for edge computing-enabled NV-MCUs is not straightforward, we propose here a method to automate the exploration of the memory design space. For this, we first study how for a given application, instructions and data are organised in the memories of a typical MCU architecture. Then, as we target memory architecture and thus sizing explorations, we describe how we build emerging memory models for different memory sizes. We then adapt our platform and more precisely our monitoring unit to improve the granularity of the monitoring of memory content activity. Finally, we present MemCork, our exploration tool for memory architecture design space exploration of NV-MCU memory architecture. This tool proposes a systematic and automated exploration framework of MCU memory architecture with custom data placement based on the output of our platform.

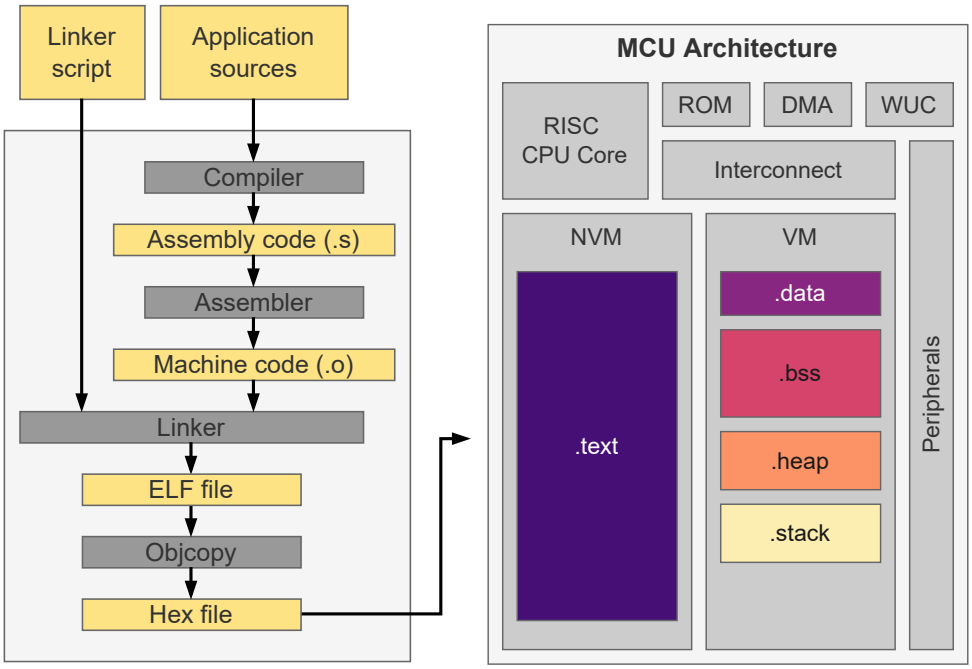
5.2.1 Data mapping

Figure 5.1 describes how application sources are built with a toolchain (see Figure 5.1a) and how instructions and data are mapped in a typical MCU memory architecture (see Figure 5.1b). As shown in Table 4.3, most MCUs include several memories combining Volatile Memories (VM) and NVM. NVM is usually based on a ROM for the boot code and a storage memory (Flash or EEPROM) for the program and read-only data. The VM (SRAM) is used to store data during program execution. At the software level, the source code is compiled and then linked to generate the executable, using a dedicated

toolchain. We focus on the linker script, which declares and defines the memories and their corresponding address ranges. At compile time, this script maps instructions and data to the memories by grouping them into typically five sections:

- The *.text* section contains the application code as a sequence of instructions, the read-only data and the default values of the application global variables initialised to specific values other than zero.
- The *.data* section contains all global variables, their default values are initialised from the *.text* section at startup and can be modified during execution.
- The *.bss* section contains all uninitialised variables and is initialised to zero at startup.
- The *.heap* is used for dynamic memory allocation, its size is defined by the programmer in the linker script.
- The *.stack* for temporary data such as function parameters, return address and local variables, its size is defined by the programmer in the linker script.

As depicted in Figure 5.1b, the *.text* section is stored in NVM, while *.data*, *.bss*, *.heap* and *.stack* are located in VM.



(a) *Compilation flow* **(b)** *MCU architecture*

Figure 5.1: *Typical microcontroller compilation flow and section mapping*

5.2.2 Memory model

In order to explore the memory architecture, we need to be able to evaluate the power consumption of each memory. In Section 3.3, we defined an activity-based energy model to calculate the energy consumed by a memory. This model takes into account the static consumption of a memory but also its dynamic consumption based on the number and type of accesses made to the memory during a given evaluation duration denoted T_{eval}. To calculate the energy of a memory of a given technology and size, we use four parameters: E_{rd}, E_{wr}, P_{statOn}, $P_{statOff}$. As we are exploring both technology mapping and memory sizing, we choose to collect these parameters for different technologies and express them as a function of the memory size. We collect data for different memory sizes of each technology and approximate each parameter as an affine function $ax + b$ with x being the memory size, and a and b technology-dependent constants.

For SRAM technology, we use the same model that was used in Chapter 4. This model was obtained from two 28nm FD-SOI SRAM designs (16kB and 64kB) built with the single-port high-density compiler supporting forward body bias (SPHD-BODYBIAS) from STMicroelectronics.

Emerging NVMs, such as MRAM, are less mature than mainstream technologies. As a result, we have no access to datasheet information or memory compilers. However, in the context of the NV-APROC project, we are working with Spintec, which develops STT and SOT MRAMs. As part of this collaboration, they design full memories at the transistor level and characterise them to extract the mentioned parameters. We choose to design a high-density 32-bit word single bank with an SRAM-like interface. The chosen CMOS process is 28nm FDSOI from STMicroelectronics and the considered MRAM processes are STT and SOT, with MTJ diameters of 28nm.

The first step performed by the Spintec team is to calibrate the compact MTJ models of the two MRAM technologies for electrical simulations. For each of them, they calibrate the model to fit the state of the art for a MTJ diameter of 35nm, and then extrapolate to 28nm. Based on these compact models, they can design and characterise 128kB STT and SOT memories. Both memories include 1024 rows and columns. Since the word size is 32 bits, they are 32 words per row, resulting in a 32-to-one multiplexing to select the word. The address is thus composed of 15 bits: 10 to select the row and 5 to select the word in the row. They characterise the bit cell by evaluating the parasitic resistance and capacitance of the access lines depending on the memory size. This is evaluated by drawing a very simplified layout of the memory array to approximate the size of the access rows and parasitics. These parasitics are injected in the critical path of the memory, which contains the bit cell, made of the MTJ and access transistor, the

writing drivers, reading amplifiers and access transistors such as activation transistors and multiplexing. Thus, the critical path is representative of the worst-case scenario for any bit cell and allows an exhaustive set of electrical simulations using standard electrical simulators such as Spectre from Cadence. They use these simulations to ensure the operation of the memory and extract the main parameters in terms of writing and reading currents, as well as timing data. Full memory operation is then performed and validated using a "fast-spice" simulator, such as UltraSim from Cadence, to reduce the simulation runtime and cover a large number of cases. From these other simulations, they can extract write and read energies, the `On` and `Off` static power due to leakage of a 128kB memory for both STT and SOT as shown in Table 5.1.

Table 5.1: *128 kB MRAM high-level energy models provided by Spintec*

	STT 28nm	SOT 28nm
Size [kB]	128	128
Byte read energy [J/B]	7.50e-12	7.50e-12
Byte write energy [J/B]	7.50e-12	3.75e-12
Static power in mode `On` [W]	1.50e-04	1.50e-04
Static power in mode `Off` [W]	2.40e-09	2.40e-09

To extend these estimations to other memory sizes, we finally build linear functions of the MRAM parameters. We assume that P_{statOn} and $P_{statOff}$ increase linearly with the memory size as all powered transistors contribute to leakage. For E_{rd} and E_{wr} we assume constant values because the bit-cells selected in a memory access remain constant for any memory size.

5.2.3 Exploration tool

As we want to explore the mapping of data in memory, monitoring the memory activity as we did in Chapter 4 is not sufficient. Instead, we need to evaluate the activity of the different data and instructions stored in the memories. Monitoring memory addresses individually is not realistic as it would generate excessively large traces. Thus, we need to extend the activity monitoring described in Chapter 4 to record at the section granularity.

5.2.3.1 Monitor extension

We extend the FPGA-based platform with an enhanced monitor, capable of retrieving run-time information and to evaluate the activity of memories with section granularity. For memory activity evaluation, the monitor described in Section 4.3.3 was recording the OBI crossbar signals to count the access in the two RAMs. For each memory, the monitor counts the number of reads and writes on 8, 16 and 32 bits allowing to track

n_{rd} and n_{wr}, the number of reads and writes in both memories. Figure 5.2 describe the architecture of the extended monitor.

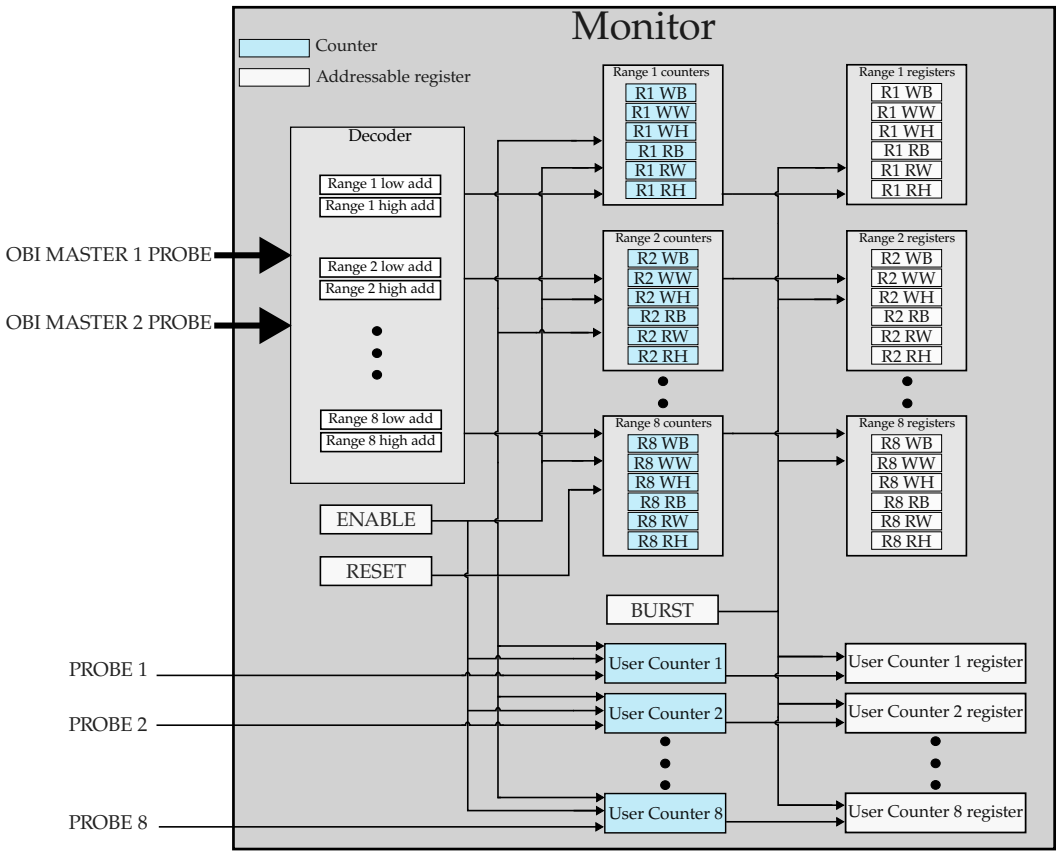


Figure 5.2: *Enhanced monitor for section activity monitoring*

The new monitor can now track n_{rd} and n_{wr}, the number of reads and writes in up to eight sections. The monitor now embeds eight sets of memory activity counters. Each set counts the memory access between two configurable addresses (from a lower address to a higher address). To do this, these counters are controlled by a decoder that detects the different types of memory accesses on the OBI bus and increments the counter for the concerned ranges. As explained in Section 4.3.3, our monitoring unit is responsible for loading the binary application into the MCU memory, then it starts the MCU while tracking different events. The new version of the monitoring unit also collects the compilation time parameters and more specifically the lower address and higher address of each section in order to configure the registers of the monitor. Once the monitor is configured, then the monitoring unit can start the MCU and start tracking different events. To report the value of the counters, the monitoring unit generates a burst in a monitor register. This burst is used to synchronise the copy of each counter into dedicated registers. Once copied, the content of these registers is transmitted via the UART so that it can be retrieved by a computer. In this way, the counter values snapshot represents the sate of the counters during the same cycle.

This monitor allows to track the activity in each section during the application execution. It is therefore possible to design an application incorporating the backup and restore mechanisms necessary to emulate the use of volatile and non-volatile memory. However, we want to explore a wide range of possible section mapping in different memories (volatile and non-volatile). Therefore, we choose not to include backup and restore phases in the application execution, as they depend on the memory technology and the section mapping. Instead, their execution is considered in the subsequent memory exploration step described in Section 5.2.3.2.

5.2.3.2 Exploration tool

To explore the many possible solutions combining different instruction/data mappings in different memory technologies, resulting in many options for memory sizing, we propose MemCork. Based on the size and the activity tool for NV-MCU memory architecture exploration in intermittent computing devices at the edge.

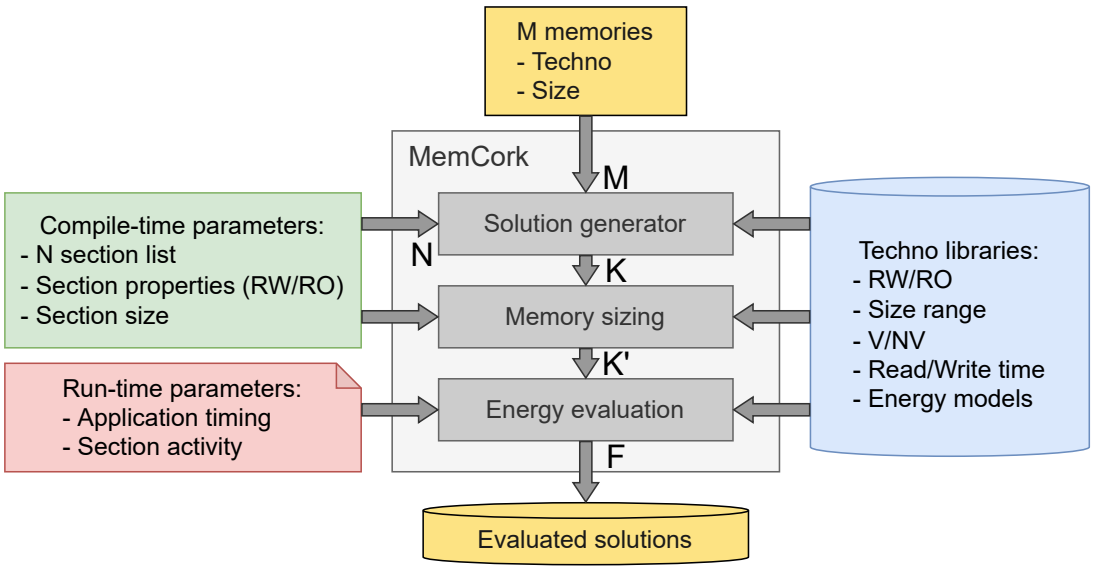


Figure 5.3: *MemCork exploration flow*

As illustrated in Figure 5.3, this exploration tool uses five inputs;

- A library containing the properties and energy models of each available memory technology.
- M, the set of memories, with their corresponding technology and size defined by the user.
- N, the set of sections and their properties.
- The compile-time parameters that contain information about sections such as their name and size.

- The output of the monitor with the recorded run-time parameters during the T_{eval} evaluation period.

We propose an algorithm that explores all possible memory architecture and section mapping combinations for the evaluated application, with $m = |M|$ the number of memories, and $n = |N|$ the number of sections. A memory M_j can store $|M_j|$ bytes, and can be read-only (R), or read/write (R+W). $|N_i|$ is the size in bytes of a section N_i. A feasible mapping is an application $f : N \mapsto M$ where all sections can be mapped to the available memories. One or more N_i sections can be mapped into a memory M_j provided that R+W N_i sections are only mapped to a R+W M_j memory. Based on its inputs, the tool generates K, the set of solutions (feasible mappings) with cardinal $|K| \leq m^n$, depending on the constraints of the different sections and memory, *i.e.*, type (R or R+W) and size.

If the solution contains one or more sections in volatile memory, then the tool will create a backup section with a size equal to the sum of the sizes of the sections mapped in volatile memory. Then the tool will create derived solutions where it will position this backup section in each available non-volatile memory. Based on the section sizes and the required backup space, the tool computes the dimension of each memory for each solution of the K set. A user-defined upper and lower limit is assumed for each memory technology. According to the generated section mapping, it determines the minimal memory depth that minimises the total consumption. Memory size is valid only if it is within the range allowed in the corresponding memory technology library. Once calculated for each solution, the tool extracts K', the subset of K containing all solutions for which the memory size is valid.

Finally, based on the run-time parameters obtained with the monitor described above, the solutions of the set K' are evaluated following the energy modelling methodology proposed in the previous chapter. Each solution of the K' set can possibly require different backup and restore operations. The memory activity and duration related to backup and restore are introduced at this stage. The energy of each memory for each application phase during the evaluation period is calculated. The output of the tool is the solution set F, which is a subset of the set K'. It only includes the solutions where execution is feasible, *i.e.*, complies with the minimum inactive time required for backup and restore.

Figure 5.4 illustrates the organisation of the MemCork framework. To use the tool, an architecture explorer must be instantiated. Then, it is possible to add memories and sections in the instantiated explorer. To evaluate an application, the output of the monitor in .csv format is loaded into the explorer. Then, the tool will automatically generate and evaluate all the solutions of the F set.

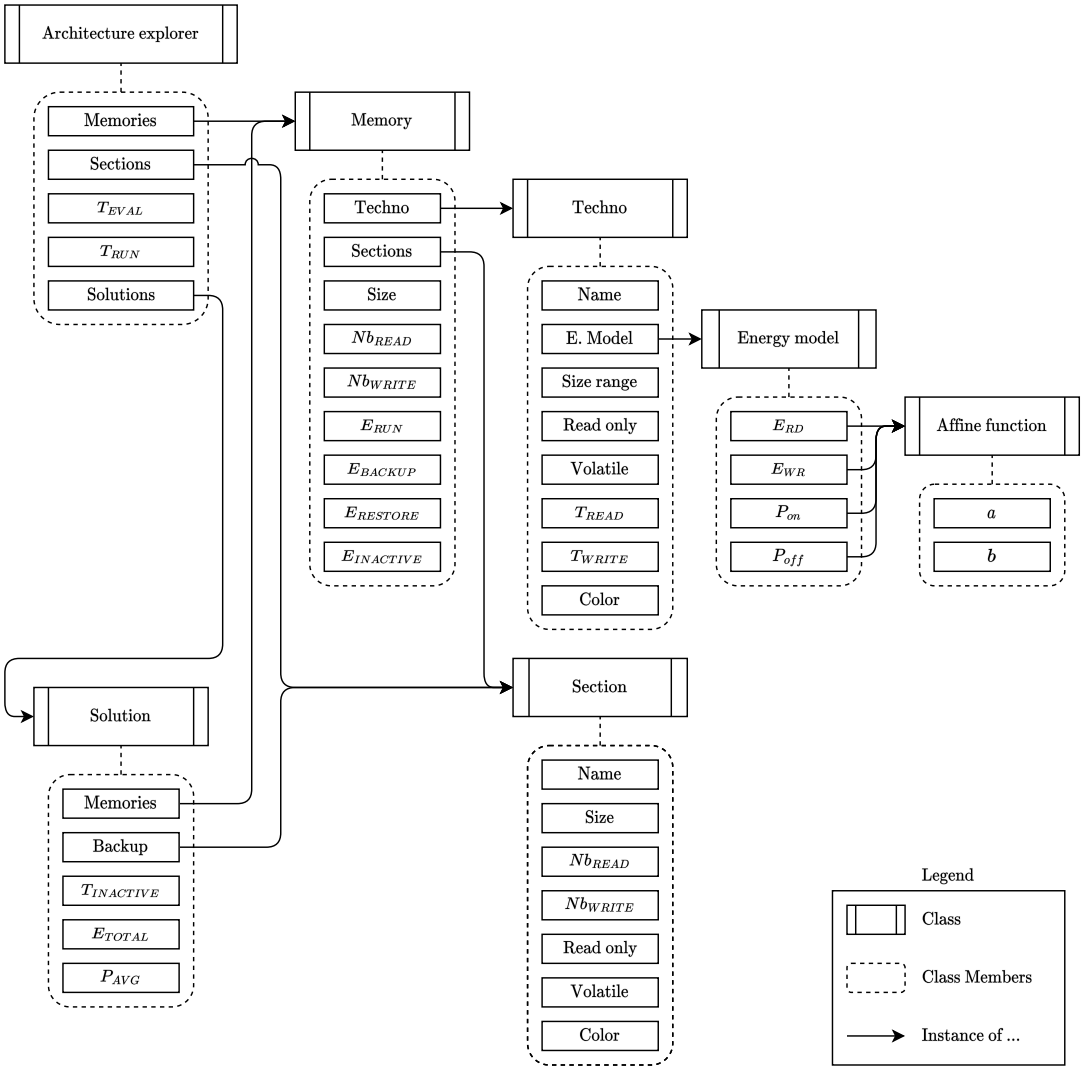


Figure 5.4: *MemCork framework organisation*

5.3 MCU architecture explorations

In previous sections, we motivated the need to explore the memory architecture of MCUs in an edge computing context. Then, we extended our evaluation platform to track memory activity with a section granularity and proposed MemCork, an MCU memory exploration tool. In this section, we use our exploration tool with our FPGA-based platform on the applications described in Chapter 4 to find memory architecture that allows to reduce MCU and thus node energy consumption.

5.3.1 Demo example and experimental setup

To illustrate the functioning and the presentation of the results of our tool, we start with a simple example of two sections in two memories. First, we define the N set with a

list of sections specifying for each of them if they are read-only sections or not. For this example, we use an application with only two sections that we name as Section 1 and Section 2. Then, we define the M set with a list of memories with for each memory a name, a technology and a maximum size. Here we define two memories, an SRAM memory of 16kB maximum size and an STT-MRAM memory of 64kB maximum size. Then we instantiate an architecture explorer which will generate the solutions of the set K and then K' from the compile time parameter of the application which contains in particular, the sizes of the sections after compilation. To evaluate the solutions, we load in the explorer the run-time parameters which are in the output of the monitors during the evaluation period as a .csv file. These parameters include information such as the activity of all sections, the total duration of the MCU in active and inactive phases, as well as the number of transitions between active and inactive phases. The sum of the total duration in active and inactive phases is used to calculate the evaluation period, which in this case is 24 hours. Finally, we just need to retrieve the set of four feasible and evaluated solutions that constitute the set F as shown in Figure 5.5.

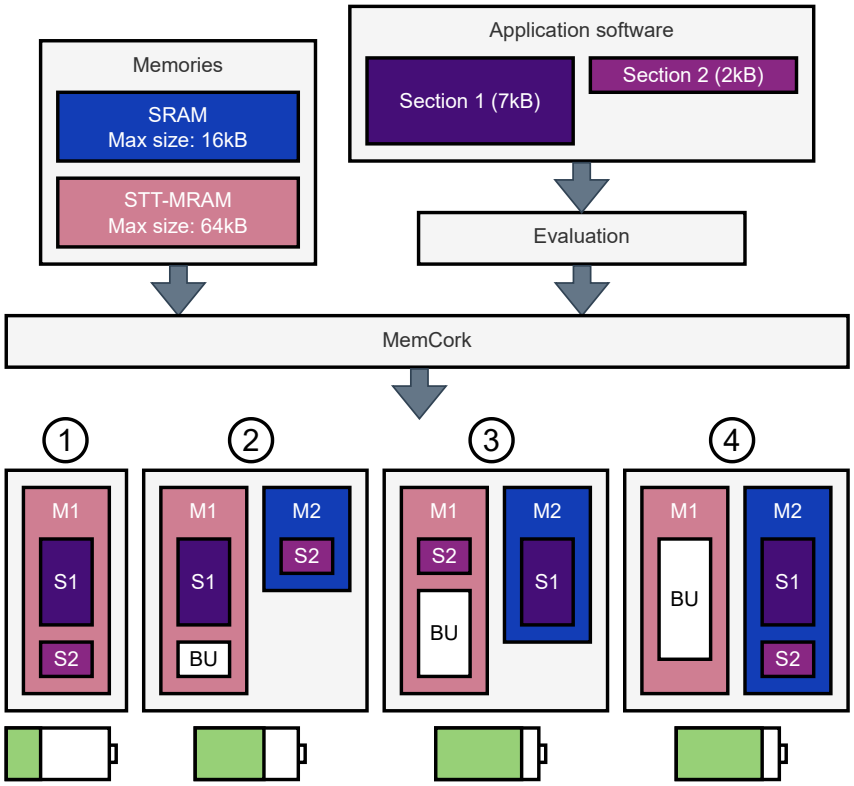


Figure 5.5: *MemCork example for two section in two memories*

Figure 5.6 illustrates the way we represent the result of the tool for the previously defined setup. Each line of the left side graph represents a $N \mapsto M$ mapping solution of the F set and the associated memory energy consumption on the right. Here we display all four solutions of the F set classified according to the energy consumed by the memory architecture during the evaluation period. For example, the first line of the left side graph represents a mapping solution where both section 1 (dark purple)

and section 2 (purple) are located in a 16kB STT-MRAM memory (pink). On the right graph, we can see the corresponding energy during the one-day evaluation period, which is around 34mJ. In the same way, the second line of the left side graph represents a mapping solution where section 2 is located in a 2kB SRAM memory (dark blue). Section 1 is located in a 16kB STT-MRAM as well as a memory zone reserved for backup (white). This section is used here to store section 2 during the inactive phase through backup and restore phases. On the right graph we can see that this solution consume around 74mJ per day.

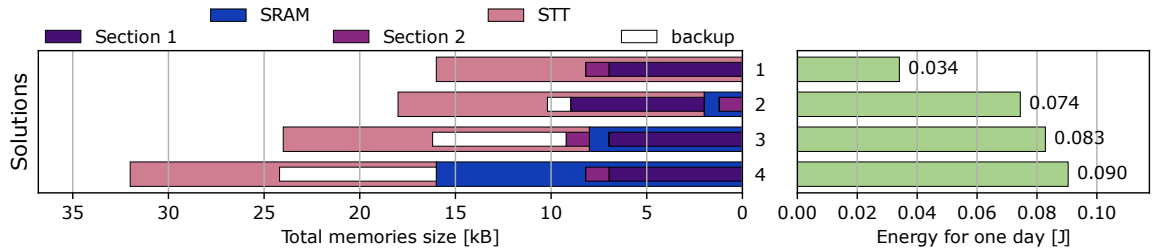


Figure 5.6: *MemCork (mode 1) output example for two section in two memories*

In the example shown in Figure 5.6, the tool calculates the optimal size of each memory according to the section mapping. If a memory contains at least one section, the tool calculates the minimum power of two size of the memory that can fit its contents. If a memory is empty, the tool removes it from the solution. Thus the tool generates the optimal memory architecture for each solution. However, it is possible to use the evaluation tool in another way. There is another possible mode where the tool does not calculate the optimal memory size and simply uses the maximum size defined for each memory as shown in Figure 5.7. This mode is also interesting because it allows to find the optimal mapping for an application in a given architecture.

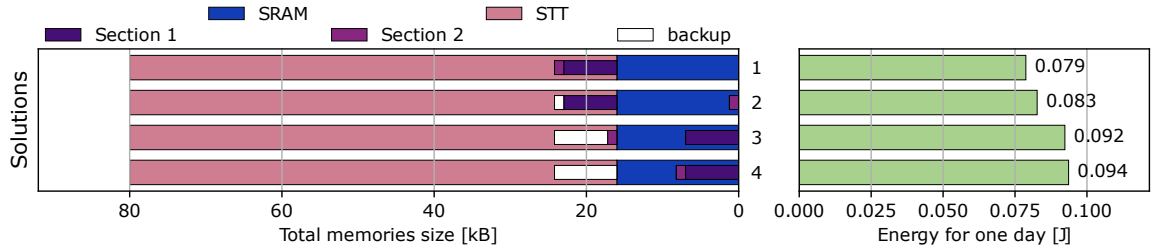


Figure 5.7: *MemCork (mode 2) output example for two section in two memories*

Now that we have illustrated how our tool works and how the results will be represented, we set up the tool to explore memory architecture of an MRAM-based NV-MCU running our two previously defined applications. The tool is configured this way:

- The M set contains six available memories: two SRAM, two STT-MRAM and two SOT-MRAM with a maximum size of 512kB for each.
- The N set contains six sections: *.text*, *.data*, *.bss*, *.heap* and *.stack*.

We repeat the same experiments done in the Chapter 4 with the platform equipped with the new monitor. For all experiments, the evaluation period is one day, as we did in Chapter 4. As we want to explore the architecture and the memory mapping, the tool is configured in its first operating mode.

5.3.2 Light application

First, we study the sizes and activities of the sections for the two scenarios of the light application as illustrated in Table 5.2. For the cloud scenario, the code is very simple (see Figure 4.11a) resulting in a *.text* section smaller than 6kB. The data section is empty while the *.bss* section is a bit more than 1kB because it contains all the user-level input and output buffers of communication peripherals. Finally, the sizes of the *.heap* and the *.stack* are defined by the application designer according to the needs of the application. In this case, they both have a size of 1kB which is largely sufficient for this type of application. Indeed, the source code of this application is very simple, it integrates few functions that use few variables, so the stack utilisation during execution is very low. Concerning the edge computing scenario, we observe that the only difference compared to the cloud approach is the size of the *.text* section which is now more than 7kB due to the integration of little processing stage (see Figure 4.12a).

Table 5.2: *Light application section size in bytes for both scenarios*

Scenario	Cloud	Edge
.text	5.67e+03	7.14e+03
.data	0	0
.bss	1.24e+03	1.24e+03
.heap	1.02e+03	1.02e+03
.stack	1.02e+03	1.02e+03

Table 5.3: *Light application section activity analysis for both scenarios*

	Cloud scenario		Edge scenario	
Section	Rd [B]	Wr [B]	Rd [B]	Wr [B]
.text	3.59e+12	0	4.46e+09	0
.data	0	0	0	0
.bss	1.04e+06	6.91e+05	1.04e+06	6.91e+05
.heap	0	0	0	0
.stack	4.68e+07	5.20e+07	8.90e+06	9.08e+06

Concerning the sections activity, as we can see in Table 5.3, in both cases, reads in *.text* section represent a large part of the overall activity. For both scenarios, there is no activity in the *.data* section as it is empty. This is because there are no initialised global variables in the source code of this application. Furthermore, the application does not use dynamic allocation resulting in no activity in the *.heap* section defined in the linker script. We observe that contrary to what we might have thought, the activity in the other sections is globally lower for the edge computing scenario. Indeed, the

processing is so simple that it represents less activity and time than communication via the radio. The *.stack* sections is the most active after the *.text* section. For the cloud scenario, during one day, the total active time T_{act} is less than 7h against only 31s for the edge scenario. This means that the activity rate of the cloud scenario is about 29% whereas the edge scenario is 0.03%.

After launching the exploration tool, we extract for analysis, among thousands of generated mappings, three relevant solutions: HYB (hybrid), NV (fully non-volatile) and BEST (optimal solution). The hybrid solution uses a SOT memory for *.text* section and backup content plus a single SRAM memory for all other sections. The fully non-volatile solution uses a single SOT memory for all sections. We use these two solutions as a baseline to represent typical microcontroller architecture with SOT replacing Flash and both Flash and SRAM. Finally, the optimal solution is the one that results in the lowest energy consumption during the evaluation period.

5.3.2.1 Cloud scenario

The three relevant solutions for the cloud scenario are illustrated in Figure 5.8.

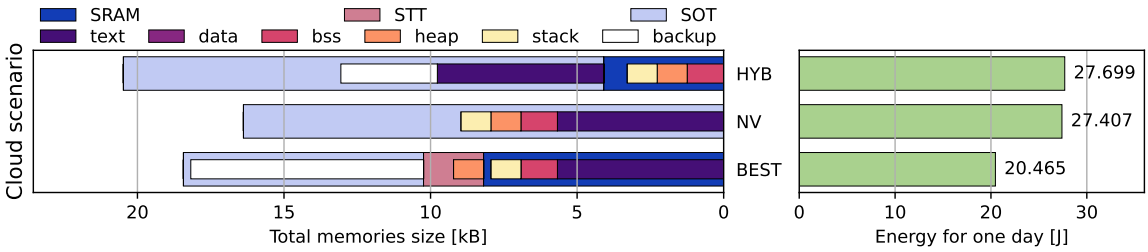


Figure 5.8: *MemCork output for light application in the cloud computing scenario*

For the cloud scenario, the BEST solution selected by MemCork is a hybrid solution with the *.text* (dark purple), the *.bss* (dark pink) and the *.stack* (yellow) sections located in a 8kB SRAM memory while the *.heap* (orange) section is located in a 2kB STT-MRAM. Even if the heap is not used, the tool will allocate it a space corresponding to the minimum size of the heap defined in the linker script because it is not possible to predict whether the application will use it or not before compilation. The use of the 2kB STT-MRAM for the *.heap* allows the use of an 8kB SRAM instead of a 16kB one and has no impact on dynamic performance as this section is unused by this application. This solution uses STT-MRAM but there is another solution that uses SOT-MRAM with the same performance. This is because the SOT-MRAM and STT-MRAM models only differ in terms of dynamic performance, and there is no access to this memory while the application is running. A third memory is used to store the backup (white) of the sections *.text*, *.bss* and *.stack* during inactive phases. This solution allows up to a 26% reduction in total energy consumption compared to the NV and HYB solutions. In-

deed, this solution takes advantage of the good dynamic performance of the SRAM during the active phases which represent a significant part of the total time (29%).

5.3.2.2 Edge scenario

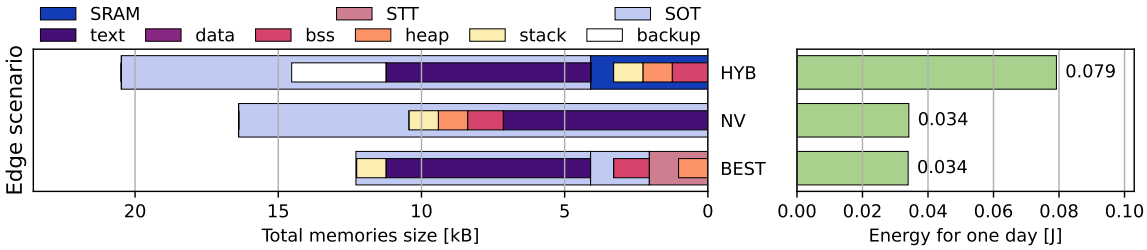


Figure 5.9: *MemCork output for light application in the edge computing scenario*

Considering the edge scenario, the best solution is a fully non-volatile solution where all sections are distributed in three non-volatile memories, including the two available SOT-MRAM and one STT-MRAM. In this case, the tool is able to slightly minimise the energy compared to the classic fully non-volatile solution by using multiple memories to limit the amount of unused memory. We also see that the sections are distributed in the memories in order to limit the unused memory. Furthermore, the heap has been mapped into the STT-MRAM because it offers the lowest dynamic performance and there is no activity in this section. It can be noticed that for all three solutions, the overall energy consumption of the memories is much lower than for the cloud scenario, which is consistent with the results in Chapter 4. The BEST and NV solutions allow up to a 57% reduction in total energy consumption compared to the HYB solution (*.text* section and backup content in SOT-MRAM and all other sections in SRAM). This means that a hybrid solution with volatile memory combined with backup and restore phases is not adapted in this case.

We observe for both scenarios that the best solutions are different. From this study, we may conclude that for low duty-cycled applications, it is preferable to use a fully non-volatile approach because during the inactive phase, the leakage current of the power gating mechanisms of the SRAM memories consume more than the non-volatile ones. However, there is an activity rate at which it is preferable to use volatile memories and backup and restore mechanisms due to their good energy performance during the active phases. We also notice that the best solutions generated by the memory mapper are optimised to avoid large unused memory segments, which reduces the aggregated size of the memories and thus their energy.

5.3.3 Heavy application

As we did for the light application, we start by studying the sizes and activities of the sections for both scenarios as illustrated in Table 5.4. The activity was measured over a day where the node was continuously sensing and detected one hundred words. We chose one hundred words because of the strong limitation of the cloud scenario due to its long communication period. For the cloud scenario, the code is quite simple (see Figure 4.14a), the only complex part is the audio buffer management. Thus, the resulting *.text* section is around 11.6kB. The data section is empty while the *.bss* section is around 33kB because it contains the 32kB audio buffer. Finally, the sizes of the *.heap* and the *.stack* are the same as for the light application (1kB).

Concerning the edge computing scenario, the size of the resulting sections is very different. Indeed, in this case, the application is much more complex (see Figure 4.15a), it integrates in fact the audio buffer management functions, the Kiss FFT library to build the audio spectrogram and of course the Tensorflow lite library for microcontroller which includes a 20kB pre-trained model the keyword detection resulting in a 200kB *.text* section. The *.data* is 144B long as the application code uses several global variables. The *.bss* section is around 47kB. It contains, for example, the audio buffer of 32kB, a memory area dedicated to the storage of the intermediate results of the Tensorflow computation called Tensorflow arena with a size of 10kB or the spectrogram with a size of 2kB. Finally, the sizes of the *.heap* and the *.stack* must be bigger to support such a complex application. We chose 32kB for each to ensure a good safety margin.

Table 5.4: *Heavy application section size in bytes for both scenarios*

Scenario	Cloud	Edge
.text	1.16e+04	2.01e+05
.data	0	1.44e+02
.bss	3.30e+04	4.72e+04
.heap	1.02e+03	3.28e+04
.stack	1.02e+03	3.28e+04

Table 5.5: *Heavy application section activity analysis for both scenarios*

	Cloud scenario		Edge scenario	
Section	Rd [B]	Wr [B]	Rd [B]	Wr [B]
.text	7.38e+12	0	7.44e+12	0
.data	0	0	5.38e+03	2.00e+00
.bss	1.64e+10	1.09e+10	1.65e+10	1.10e+10
.heap	0	0	1.17e+08	7.94e+07
.stack	2.19e+10	1.65e+10	2.91e+11	2.83e+11

Regarding sections activity, as we can see in Table 5.5, for the cloud scenario, we can see that reads in *.text* section represent a large part of the overall activity. There is no activity in the *.data* section as it is empty and the application does not use dynamic allocation resulting in no activity in the *.heap* section. Finally, we observe a very

high activity in the *.stack* section and the *.bss* section. This high activity is due to the continuous recording of the audio signal performed by the MCU. Indeed, to ensure a sampling frequency of 16kHz, the MCU reads the value of the audio signal from the microphone every 62.5µs and this task lasts about 35.2µS. This means that even if there is no detection during the day, a minimum activity rate of 56% can be expected.

For the edge scenario, we observe that the activity of the sections is globally higher. We observe a slight activity in the *.data* section. We also observe a strong activity of the *.heap* section, indeed, this section is notably used by the kiss FTT library to build the spectrograms. The *.stack* sections is the most active after the *.text* section. The memory-intensive nature of the edge computing approach can therefore be easily observed. In terms of activity rate, as the audio recording is done in the same way as for the cloud scenario, a minimum activity rate of 56% can also be expected.

As this application is highly dependent on the environment, we study the monitor output after several runs as we did in Chapter 4. During these runs, we generate a set of detection or none. The collected traces allow us to identify the contribution of each type of detection on the monitor counters. Thus by extrapolation, we can build up monitor traces for variable run times with different detection types. As we demonstrated in Section 4.5.3, for this application, the cloud computing approach is not suitable. Thus, our next experiments will only focus on the edge computing approach.

We start by extrapolating the monitor's output for a one-day evaluation period with 1, 10 and 100 detection with communication. We then use the MemCork tool on each of these to extract the three relevant solutions: HYB (hybrid), NV (totally non-volatile) and BEST (optimal solution). We notice that the HYB solution is not generated by the exploration tool. This is due to the fact that it is not possible to use a non-volatile memory and to execute a backup and restore phase during the 27.3µs available between two microphone sensings. During this short time, it is possible to backup and restore only a little more than 1kB. Figure 5.10 illustrates the results for the edge scenario.

In Figure 5.10, we observe that whatever the number of detection, MemCork proposes a non-volatile solution where the *.heap* and *.stack* sections are located in a 64kB SOT-MRAM and the *.text*, *.bss* and *.data* (purple) sections are mapped in a 256kB SOT-MRAM. As mentioned in the Chapter 4 the use of the MCU to collect data from the sensor is not efficient. It is preferable to use a dedicated peripheral that will consume much less than the MCU and that allows it to wake up only when there is a detection to do processing and/or communication. Thus, we will now look at the edge computing approach with sensing performed by a dedicated peripheral. We extrapolate the monitor output for a one-day evaluation period for a number of detections ranging from 500 to 5000. Then we run our exploration tool on these outputs and collect the BEST

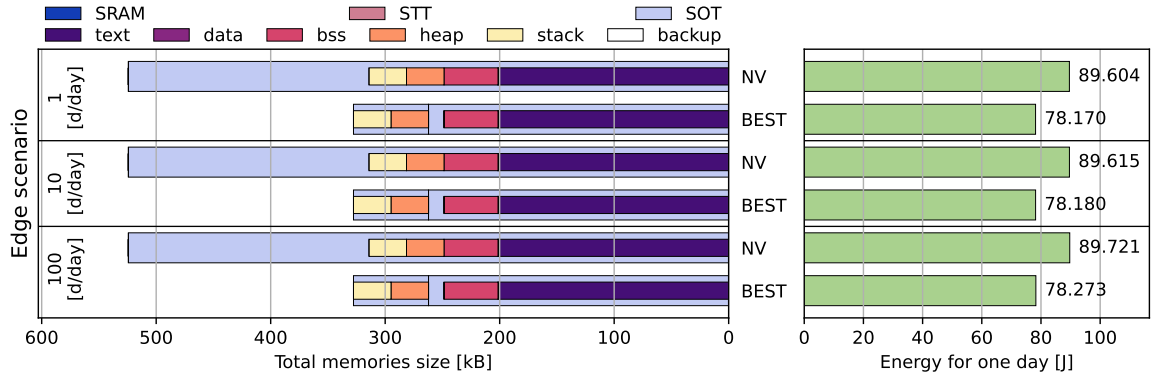


Figure 5.10: *MemCork output for heavy application in the edge computing scenario depending on the number of words detected*

solution for each of them (see Figure 5.11).

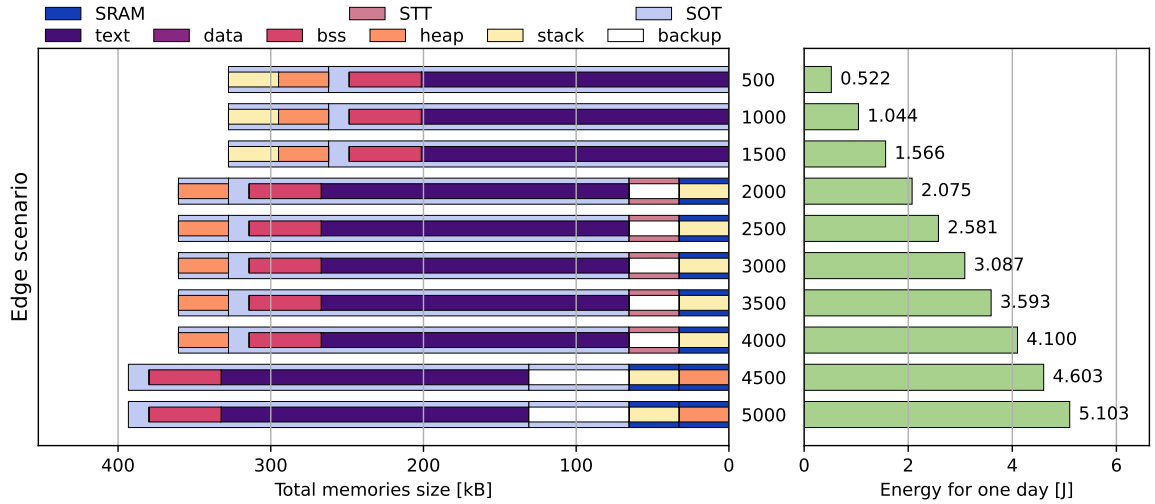


Figure 5.11: *MemCork output for heavy application in the edge computing scenario with sensing performed by a dedicated peripheral depending on the number of words detected*

We observe that for a number of detection lower than 2000, MemCork proposes a non-volatile solution where the *.heap* and *.stack* sections are located in a 64kB SOT-MRAM and the *.text*, *.bss* and *.data* sections are mapped in a 256kB SOT-MRAM. For a number of detection ranging from 2000 to 4000, MemCork proposes a hybrid solution where the *.stack* section is now mapped in a 32kB SRAM and its backup is performed in an additional 32kB STT-MRAM. Finally, from 4500 detections, MemCork proposes a hybrid solution where the two sections *.heap* and *.stack* are located in a 64kB SRAM and their backup is performed in the 64kB SOT-MRAM. It can be observed that if the system is low duty-cycled it is preferable to use non-volatile technologies. However, if the activity phases are more recurrent, then it is possible to take advantage of a hybrid architecture based on the use of small volatile memory to store the sections where the activity is important during the numerous active phases.

In Figure 4.26b, we note that, for 1000 and 5000 detections per day, depending on the number of words detected, the share of energy consumed by the MCU can represent up to 65% and 73%, respectively. In this energy, we find, for 1000 detection, 59% dedicated to the CPU core during the inactive phases and 38% to the memories. For 5000 detections, 41% is dedicated to the CPU core during the inactive phases and 50% to the memories. It should be noted that the power consumption of the CPU core during inactive phases used to obtain these results is over-estimated, indeed, we did not consider clock gating in its evaluation. By using a non-volatile CPU core based on hybrid flip-flops as presented in Section 3.2.2, we can reduce to almost zero the consumption of the CPU core during the inactive phases. Regarding memories, as a reminder, the memory architecture used in Chapter 4 was based on volatile memories (SRAM) in retention mode during inactive phases. MemCork has found for 1000 detections a memory architecture solution that allows to reduce by 92% the energy consumed by this initial architecture. In the same way, for 5000 detections, the tool found a memory architecture solution that allows to reduce by 78% the energy consumed by the memories. This means that with a non-volatile CPU core and an optimised memory architecture, we can reduce the MCU power consumption by 94% and 80% for 1000 and 5000 detections/day, respectively. Finally, if we go back to the node level, for 1000 detections, the MCU energy consumption represents 65% which makes a decrease of the total node energy consumption of 61%. For 5000 detections, the MCU energy consumption represents 73%, which makes a decrease of the total consumption of the node of 58%.

5.3.4 Summary

In this chapter, we studied the organisation of data and instructions in MCU memories and defined the different sections. Then we improved the platform presented in Chapter 4 to have a fine grain section activity monitoring. We also extended our memory models with size-dependent MRAM models. Finally, we introduce MemCork, a systematic and automated framework for hybrid memory architecture and data mapping exploration. This tool allows for identifying the most energy-efficient solution for a given application based on a fine-grained tracking of the memory activity performed on our technology-agnostic FPGA-based node prototype running applications in real-time and in a real-life environment. We used MemCork on our two applications to study the influence of the application type and its environment on the most energy-efficient solution.

VI

Conclusion

Contents

This thesis provided an exploration method and an FPGA-based platform to perform application-architecture co-optimisations. Thanks to the proposed platform and methodology, this thesis explored sensor nodes' communication/processing tradeoffs and identified energy bottlenecks. It studied and analysed how emerging NVM can be integrated into MCU architecture to increase sensor node energy efficiency.

We summarise here our key contributions and propose some recommendations for future works.

6.1 Contributions

The ever-increasing amount of data generated by sensor nodes in IoT networks led to the distribution of data processing across the network, especially for performance, energy and security requirements (*i.e.*, edge computing). This distribution increases the processing and storage requirements of the sensor nodes, which is not suitable considering their high energy constraints. This is why, our final objective is to improve the energy efficiency of sensor nodes to enable further computing and storage at the edge by using emerging non-volatile memories. We have identified three key challenges to achieve this goal. Next, we summarise our three main contributions to address these challenges.

6.1.1 Sensor node energy modelling

The first challenge was to evaluate for a given application, the consumption of sensor nodes and their different components to guide our future optimisations. To do so, based on an analysis of sensor nodes' architecture and the applications they embed, we defined an energy model for each component of the node while running a typical application. Then we dived into microcontroller architecture as this component is the one responsible for computing and storage in such nodes. First, we studied in detail how emerging non-volatile memories are integrated into MCUs to build NV-MCUs from their registers to their memories. Then, in the same way as for the node level, we use this analysis to build an energy model for a NV-MCU and it's different components. All these models allow to evaluate the consumption of the different components at the sensor node level as well as at the MCU level from their activity by counting specific events for each of them.

6.1.2 FPGA-Based node prototype

The second challenge was to find a solution to evaluate the activity of different components at the sensor node and MCU level for different types of applications in a realistic environment that allows application-architecture co-optimisation at both the sensor node and MCU level. For that, we propose an FPGA-based node prototype that combines real-time and real-condition evaluation of hardware prototyping with the high flexibility and ease of exploration of programmable logic. We designed a custom MCU architecture based on a RISC-V CPU core that can support a wide range of applications from low-end nodes to smart nodes. This architecture is prototyped on an FPGA target connected to real-life sensors and radios, creating a real-time operating sensor node. The FPGA target also integrates a monitoring unit that allows to count in a non-intrusive way (without affecting the normal execution of the application) the set of events that allow to estimate the energy consumed by the different components at both node and MCU levels. We then used this platform on two applications, one based on sensors that generate a little data and one based on sensors that generate a lot of data. For each of these applications, we compared a cloud computing approach and an edge computing approach. We have shown that edge computing allows a minimal use of the radio and thus reduces the power consumption of the node. Furthermore, in some case, the cloud scenario is not even feasible as it results in too much communication on the network. However, we realise that in this context, the part of the energy consumed by the MCU and especially during these long inactive phases becomes dominant in the total consumption of the sensor node. This is why the use of emerging non-volatile technologies in the MCU architecture to design MCUs with instant on-off capacity (continuous application execution even in case of an intermittent power source) and near-zero power consumption while inactive seems to be a good solution.

6.1.3 Memory architecture design space explorations

Once the platform was ready, the final challenge was to explore how the integration of emerging non-volatile memories can enable the design of energy-efficient NV-MCUs for computing at the edge. To do so, we propose MemCork, a design exploration framework to optimise the memory system of NV-MCU for normally-off applications. We used our FPGA-based platform to trace the activity in the different instruction and data sections that constitute the memory space of an application. MemCork uses as input the size of the different sections and their activities during the execution of the application as well as the energy models of different memory technologies (SRAM, Spin-Transfer Torque (STT) and Spin-Orbit Torque (SOT) MRAM supplied by Spintec).

Then, our tool exhaustively evaluates all possible mappings between memory sections and physical memory options. It explores different memory sizes and technologies and selects the most energy-efficient solutions. When MemCork selects an SRAM memory to store a section, it automatically includes the cost of the backup and restore processes needed to power down the MCU without data loss. We used MemCork on our two applications to study the influence of the application type and its environment on the most energy-efficient solution. Our tool allowed us to find for each application a memory architecture and a section mapping that allows to reduce the memory consumption compared to standard solutions and this according to different application and environment parameters.

6.2 Perspectives

The model used for sensor node energy evaluation is still very simple, especially for the sensing and radio units. Indeed, we only consider two operating modes for these components. In the future, we would like to consider making this model more complex and accurate by dividing our sensing and communication tasks into different subtasks and defining and characterising more operating modes for the corresponding components. Concerning NV-MCU energy evaluation, we do not consider peripherals for the moment. For some applications, we can have an intensive use of peripherals; this is why we have to model their energy too.

Regarding the proposed MCU architecture, the choice of RISC-V allows for having no black boxes. Everything is customisable from software to hardware and even the toolchain. We have control over the entire hardware software interface. The proposed architecture is flexible, we can easily add or remove peripherals, and thanks to our OBI crossbar, we can easily add masters to the architecture (other cores, DMA or accelerators). We would like to integrate two essential components into this architecture to make it as close as possible to the MCUs that can be found in the industry:

- A Wake-up Controller (WuC) that would allow to support external interrupts instead of having to go through peripheral interrupts.
- A Direct Memory Access (DMA) to avoid using the CPU for simple data transfer.

Concerning the monitoring unit, thanks to its own CPU core, it is able to collect the activity of all different components at both node and MCU levels in real-time, being totally non-intrusive. For the moment, it has been designed without regard to the hardware resources required for its implementation. In the near future, we would also like to try to improve this monitor by making it more flexible and optimising its different components to reduce the required hardware resources for its implementation. For

the moment, this monitor is used for event-based energy evaluation, but we could also use it for some other purposes such as security evaluation to detect events that could result in security issues in embedded systems.

For the two proposed applications, we compare the edge and cloud scenarios, but there are many different solutions between the two we do not implement that might be very interesting. We can use this platform to explore the computation/communication tradeoff in sensor nodes in more detail. For example, it is also possible for an application to implement some of the data processing in edge devices and some in the cloud. Or it is possible, for example, to distribute the inference throughout the network.

Concerning memory design space explorations, we plan to improve our memory models by doing reverse engineering on mainstream memories in commercial MCU. We would also like to collect energy models for other emerging memory technologies such as FRAM or RRAM. Regarding MemCork, we plan to make it more flexible and add features such as memory area evaluation. We would also like to have a finer mapping granularity than sections, especially for instructions. Currently, the exploration of the memory design space only considers a flat memory space. In the future, we would also like to consider a more complex memory architecture, *e.g.* integrating multiple memory levels with caches.

Our evaluation method and how we have used it for exploration with MemCork is only the beginning. This platform can be used to perform all kinds of hardware and software explorations. In the future, we would like to consider integrating non-volatile logic into our designs. We would also like to explore the hardware architecture to accelerate the inference of machine learning algorithms to build ML-enhanced MCUs. In-Memory Computing explorations are a possible way for future works.

6.3 Publications and communications

- International Conference: Theo Soriano, David Novo, and Pascal Benoit. "An FPGA-based Emulation Platform for Edge Computing Node Design Exploration," 32nd International Workshop on Rapid System Prototyping (RSP). 2021.
- National Symposium: Theo Soriano, David Novo, and Pascal Benoit. "Exploration of Ultra Low Power Architectures for Machine Learning at the Edge," Poster in GDR SoC2 Colloque, 2021.
- Communication: Theo Soriano, David Novo, and Pascal Benoit. "An FPGA-based Evaluation Platform for Machine Learning at the Edge," LIRMM's 30th anniversary/Science Festival, September 29 and 30, and October 8, 2022.
- Code Release: Theo Soriano. "ICOBS," https://gite.lirmm.fr/adac/icobs
- International Conference: Theo Soriano, David Novo, Guillaume Prenat, Gregory Di Pendina, and Pascal Benoit. "MemCork: Exploration of Hybrid Memory Architectures for Intermittent Computing at the Edge," in Proceedings of VLSI-SOC, 2022.

6.4 Concluding remarks

In this thesis, we explore how to increase sensor node energy efficiency in an edge computing context. We study sensor node architecture and application to build a model for energy evaluation. Then, we propose an FPGA-based platform allowing application-architecture co-optimisation for exploration. Finally, we use this platform to explore how the integration of emerging non-volatile memory can help to improve sensor node energy efficiency. We plan to make the latest version of our MCU architecture and the MemCork source code available as open source. We hope that MemCork will encourage the integration of emerging non-volatile memory technology into such embedded systems. We also hope that this thesis will encourage the use of the proposed FPGA-based platform with its RISC-V based MCU architecture for further explorations.

Bibliography

[1] Jordan Teicher. https://www.ibm.com/blogs/industries/little-known-story-first-iot-device/. [Accessed September-2022]. 2

[2] John Romkey. Toast of the iot: the 1990 interop internet toaster. *IEEE Consumer Electronics Magazine*, 6(1):116–119, 2016. 2

[3] ITU. https://www.itu.int/en/ITU-T/gsi/iot/. [Accessed September-2022]. 2

[4] Statista. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/. [Accessed September-2022]. 3

[5] Wikipedia contributors. Wireless sensor network — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Wireless_sensor_network&oldid=1102534484, 2022. [Online; accessed 12-September-2022]. 2

[6] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer SJ Pister. Smart dust: Communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, 2001. 2

[7] Shayma Wail Nourildean, Mustafa Dhia Hassib, and Yousra Abd Mohammed. Internet of things based wireless sensor network: a review. *Indonesian Journal of Electrical Engineering and Computer Science*, 27(1):246–261, 2022. 2

[8] Wei Ying Yi, Kin Ming Lo, Terrence Mak, Kwong Sak Leung, Yee Leung, and Mei Ling Meng. A survey of wireless sensor network based air pollution monitoring systems. *Sensors*, 15(12):31392–31427, 2015. 2

[9] Dan Chen, Zhixin Liu, Lizhe Wang, Minggang Dou, Jingying Chen, and Hui Li. Natural disaster monitoring with wireless sensor networks: A case study of data-intensive applications upon low-cost scalable systems. *Mobile Networks and Applications*, 18(5):651–663, 2013. 2

[10] Yang Lu. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of industrial information integration*, 6:1–10, 2017. 3

[11] Milica Pejanović Đurišić, Zhilbert Tafa, Goran Dimić, and Veljko Milutinović. A survey of military applications of wireless sensor networks. In *2012 Mediterranean conference on embedded computing (MECO)*, pages 196–199. IEEE, 2012. 3

[12] Bahar Farahani, Farshad Firouzi, Victor Chang, Mustafa Badaroglu, Nicholas Constant, and Kunal Mankodiya. Towards fog-driven iot ehealth: Promises and challenges of iot in medicine and healthcare. *Future Generation Computer Systems*, 78:659–676, 2018. 3

[13] D Baskar, Mary Arunsi, Vinod Kumar, et al. Energy-efficient and secure iot architecture based on a wireless sensor network using machine learning to predict mortality risk of patients with covid-19. In *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, pages 1853–1861. IEEE, 2021. 3

[14] Alaa Azmi Allahham and Md Arafatur Rahman. A smart monitoring system for campus using zigbee wireless sensor networks. *Int. J. Softw. Eng. Comput. Syst*, 4(1):1–14, 2018. 3

[15] Jin-Lian Lee, Yaw-Yauan Tyan, Ming-Hui Wen, and Yun-Wu Wu. Applying zigbee wireless sensor and control network for bridge safety monitoring. *Advances in Mechanical Engineering*, 10(7):1687814018787398, 2018. 3

[16] Yan Liu and Chunhua Bi. The design of greenhouse monitoring system based on zigbee wsns. In *2017 IEEE international conference on computational science and engineering (CSE) and IEEE international conference on embedded and ubiquitous computing (EUC)*, volume 2, pages 430–433. IEEE, 2017. 3

[17] Khongdet Phasinam, Thanwamas Kassanuk, Priyanka P Shinde, Chetan M Thakar, Dilip Kumar Sharma, Md Mohiddin, Abdul Wahab Rahmani, et al. Application of iot and cloud computing in automation of agriculture irrigation. *Journal of Food Quality*, 2022, 2022. 3

[18] Rahat Ali Khan and Al-Sakib Khan Pathan. The state-of-the-art wireless body area sensor networks: A survey. *International Journal of Distributed Sensor Networks*, 14(4):1550147718768994, 2018. 4

[19] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016. 5, 13

[20] Amina El Attaoui, Salma Largo, Soufiane Kaissari, Achraf Benba, Abdelilah Jilbab, and Abdennaser Bourouhou. Machine learning-based edge-computing on

a multi-level architecture of wsn and iot for real-time fall detection. *IET Wireless Sensor Systems*, 10(6):320–332, 2020. 5

[21] Hiroshi Nakamura, Takashi Nakada, and Shinobu Miwa. Normally-off computing project: Challenges and opportunities. In *Proceedings of the 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 1–5, Singapore, Singapore, January 2014. IEEE. ISSN: 2153-697X. 6

[22] Frank Vater and Mario Schoelzel. Investigation of new nv memory architectures for low duty-cycle embedded systems. In *2017 IEEE East-West Design & Test Symposium (EWDTS)*, pages 1–6. IEEE, 2017. 6

[23] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. Emerging nvm: A survey on architectural integration and research challenges. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(2):1–32, 2017. 6

[24] Yongpan Liu, Zewei Li, Hehe Li, Yiqun Wang, Xueqing Li, Kaisheng Ma, Shuangchen Li, Meng-Fan Chang, Sampson John, Yuan Xie, et al. Ambient energy harvesting nonvolatile processors: From circuit to system. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015. 6

[25] Fang Su, Kaisheng Ma, Xueqing Li, Tongda Wu, Yongpan Liu, and Vijaykrishnan Narayanan. Nonvolatile processors: Why is it trending? In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 966–971. IEEE, 2017. 6

[26] Guillaume Patrigeon. *Ultra Low-Power Integrated Systems for the Internet-of-Things*. PhD thesis, Montpellier, 2020. 7, 58, 82

[27] NV-APROC. http://nv-aproc.cea.fr/Controler/home.ctrl.php. [Accessed September-2022]. 8

[28] Spintec. https://www.spintec.fr/. [Accessed September-2022]. 8

[29] CEA-Leti. https://www.leti-cea.fr/cea-tech/leti. [Accessed September-2022]. 8

[30] Chee-Yee Chong and Srikanta P Kumar. Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, 2003. 10

[31] Sandro Carrara. Body dust: Well beyond wearable and implantable sensors. *IEEE Sensors Journal*, 21(11):12398–12406, 2020. 10

[32] Sarah A Al-Qaseemi, Hajer A Almulhim, Maria F Almulhim, and Saqib Rasool Chaudhry. Iot architecture challenges and issues: Lack of standardization. In *2016 Future technologies conference (FTC)*, pages 731–738. IEEE, 2016. 11

[33] Libelium Smart Parking node. https://development.libelium.com/smart-parking-technical-guide/smartparkingnode. [Accessed September-2022]. 11

[34] Ineo-Sense ESG-Logger-Hygro. https://www.ineo-sense.com/portfolio/esg-logger-hygro/. [Accessed September-2022]. 12, 19

[35] Wikipedia contributors. Microcontroller — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Microcontroller&oldid=1099288835, 2022. [Online; accessed 25-September-2022]. 12

[36] MultiTech IP67 LoRaWAN gateway. https://lora-alliance.org/lora_products/multitech-conduit-ip67-base-station/. [Accessed September-2022]. 12

[37] Libelium Cloud. https://www.libelium.com/libelium-cloud/. [Accessed September-2022]. 12

[38] Google cloud speech to text API. https://cloud.google.com/speech-to-text. [Accessed September-2022]. 13

[39] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, 2012. 13

[40] Mugen Peng, Shi Yan, Kecheng Zhang, and Chonggang Wang. Fog-computing-based radio access networks: Issues and challenges. *Ieee Network*, 30(4):46–53, 2016. 13

[41] Massimo Merenda, Carlo Porcaro, and Demetrio Iero. Edge machine learning for ai-enabled iot devices: A review. *Sensors*, 20(9):2533, 2020. 14

[42] MG Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. Machine learning at the network edge: A survey. *ACM Computing Surveys (CSUR)*, 54(8):1–37, 2021. 14

[43] Tiny ML. https://www.tinyml.org/. [Accessed September-2022]. 14

[44] George Plastiras, Maria Terzi, Christos Kyrkou, and Theocharis Theocharidcs. Edge intelligence: Challenges and opportunities of near-sensor machine learning applications. In *2018 ieee 29th international conference on application-specific systems, architectures and processors (asap)*, pages 1–7. IEEE, 2018. 15

[45] Wikipedia contributors. Instructions per second — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Instructions_per_second&oldid=1106156303, 2022. [Online; accessed 15-September-2022]. 15

[46] Wikipedia contributors. Flops — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=FLOPS&oldid=1105186915, 2022. [Online; accessed 15-September-2022]. 15

[47] AWS IoT Over the Air (OTA). https://www.freertos.org/ota/. [Accessed September-2022]. 18

[48] Zhanserik Nurlan, Tamara Zhukabayeva, Mohamed Othman, Aigul Adamova, and Nurkhat Zhakiyev. Wireless sensor network as a mesh: Vision and challenges. *IEEE Access*, 10:46–67, 2021. 18

[49] Emanuel Popovici, Michele Magno, and Stevan Marinkovic. Power management techniques for wireless sensor networks: a review. In *5th IEEE International Workshop on Advances in Sensors and Interfaces IWASI*, pages 194–198. IEEE, 2013. 18

[50] ST Microelectronics. LSM9DS1 IMU. https://www.st.com/en/mems-and-sensors/lsm9ds1.html. 18

[51] Imran Ali, Muhammad Asif, Muhammad Riaz Ur Rehman, Danial Khan, Huo Yingge, Sung Jin Kim, YoungGun Pu, Sang-Sun Yoo, and Kang-Yoon Lee. A highly reliable, 5.8 ghz dsrc wake-up receiver with an intelligent digital controller for an etc system. *Sensors*, 20(14):4012, 2020. 18

[52] Ineo-Sense TRK-Slim. https://www.ineo-sense.com/portfolio/trk-slim-2/. [Accessed September-2022]. 19

[53] Florent Berthier. *Conception d'un processeur ultra basse consommation pour les noeuds de capteurs sans fil*. PhD thesis, Université Rennes 1, 2016. 20

[54] Jacob Fraden. Interface electronic circuits. In *Handbook of Modern Sensors*, pages 191–270. Springer, 2016. 21

[55] Haider AH Alobaidy, JS Mandeep, Rosdiadee Nordin, and Nor Fadzilah Abdullah. A review on zigbee based wsns: concepts, infrastructure, applications, and challenges. *Int. J. Electr. Electron. Eng. Telecommun*, 9(3):189–198, 2020. 22

[56] Haider Mahmood Jawad, Rosdiadee Nordin, Sadik Kamel Gharghan, Aqeel Mahmood Jawad, and Mahamod Ismail. Energy-efficient wireless sensor networks for precision agriculture: A review. *Sensors*, 17(8):1781, 2017. 22

[57] Louis Frenzel. *Handbook of serial communications interfaces: a comprehensive compendium of serial digital input/output (I/O) standards*. Newnes, 2015. 22

[58] Louis E Frenzel. *Electronics explained: the new systems approach to learning electronics*. Newnes, 2010. 22

[59] Anzar Mahmood, Nadeem Javaid, and Sohail Razzaq. A review of wireless communications for smart grid. *Renewable and sustainable energy reviews*, 41:248–260, 2015. 22

[60] Holger Karl and Andreas Willig. *Protocols and architectures for wireless sensor network*. John Wiley & Sons Ltd, Chichester, England, 2005. 22

[61] Zayan El Khaled, Hamid Mcheick, and Fabio Petrillo. Wifi coverage range characterization for smart space applications. In *2019 IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT)*, pages 61–68. IEEE, 2019. 23

[62] Yifan Zhang and Qun Li. Exploiting zigbee in reducing wifi power consumption for mobile devices. *IEEE Transactions on Mobile Computing*, 13(12):2806–2819, 2014. 23

[63] Bharat S Chaudhari, Marco Zennaro, and Suresh Borkar. Lpwan technologies: Emerging application characteristics, requirements, and design considerations. *Future Internet*, 12(3):46, 2020. 23

[64] Benny Vejlgaard, Mads Lauridsen, Huan Nguyen, Istvan Z. Kovacs, Preben Mogensen, and Mads Sorensen. Coverage and capacity analysis of sigfox, lora, gprs, and nb-iot. In *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, pages 1–5, 2017. 23

[65] Juha Petäjäjärvi, Konstantin Mikhaylov, Marko Pettissalo, Janne Janhunen, and Jari Iinatti. Performance of a low-power wide-area network based on lora technology: Doppler robustness, scalability, and coverage. *International Journal of Distributed Sensor Networks*, 13(3):1550147717699412, 2017. 23

[66] Drew Gislason. *Zigbee wireless networking*. Newnes, 2008. 23

[67] C Muthu Ramya, M Shanmugaraj, and R Prabakaran. Study on zigbee technology. In *2011 3rd international conference on electronics computer technology*, volume 6, pages 297–301. IEEE, 2011. 23

[68] Ievgeniia Kuzminykh, Arkadii Snihurov, and Anders Carlsson. Testing of communication range in zigbee technology. In *2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, pages 133–136. IEEE, 2017. 23

[69] Jin-Shyan Lee, Chun-Chieh Chuang, and Chung-Chou Shen. Applications of short-range wireless technologies to industrial automation: A zigbee approach. In *2009 Fifth Advanced International Conference on Telecommunications*, pages 15–20. IEEE, 2009. 23

[70] Bluetooth SIG. *Bluetooth Core Specification*, 12 2019. Rev. v5.2. 23

[71] Jacopo Tosi, Fabrizio Taffoni, Marco Santacatterina, Roberto Sannino, and Domenico Formica. Performance evaluation of bluetooth low energy: A systematic review. *Sensors*, 17(12):2898, 2017. 23

[72] Borja Martinez, Marius Monton, Ignasi Vilajosana, and Joan Daniel Prades. The power of models: Modeling power consumption for iot devices. *IEEE Sensors Journal*, 15(10):5777–5789, 2015. 25

[73] Joseph Finnegan and Stephen Brown. An analysis of the energy consumption of lpwa-based iot devices. In *2018 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2018. 25

[74] Tetsuo Endoh, Hiroki Koike, Shoji Ikeda, Takahiro Hanyu, and Hideo Ohno. An Overview of Nonvolatile Emerging Memories— Spintronics for Working Memories. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 6(2):109–119, June 2016. 30, 31

[75] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. Emerging NVM: A Survey on Architectural Integration and Research Challenges. *ACM Transactions on Design Automation of Electronic Systems*, 23(2):1–32, January 2018. 30, 31, 35

[76] Shimeng Yu and Pai-Yu Chen. Emerging Memory Technologies: Recent Trends and Prospects. *IEEE Solid-State Circuits Magazine*, 8(2):43–56, 2016. Conference Name: IEEE Solid-State Circuits Magazine. 30, 31

[77] STMicroelectronics. Optimizing power and performance with STM32L4 Series microcontrollers, December 2019. 32

[78] NXP Semiconductors. Power Management for S32K1xx, May 2018. 32

[79] Atmel. SAM AT03782: Using Low Power Mode in SAM4N Microcontroller, July 2013. 32

[80] Tatsuo Shiozawa, Hirotsugu Kajihara, Tatsuro Endo, and Kazuhiro Hiwada. Emerging Usage and Evaluation of Low Latency FLASH. In *Proceedings of the 2020 IEEE International Memory Workshop (IMW)*, pages 1–4, Dresden, Germany, May 2020. IEEE. ISSN: 2573-7503. 35

[81] Guillaume Patrigeon, Paul Leloup, Pascal Benoit, and Lionel Torres. Flexnode: a reconfigurable internet of things node for design evaluation. In *Proceedings of the IEEE Sensors Applications Symposium (SAS)*, 2019. 36, 50

[82] F. Vater and M. Schoelzel. Investigation of new nv memory architectures for low duty-cycle embedded systems. In *2017 IEEE East-West Design Test Symposium (EWDTS)*, pages 1–6, 2017. 36

[83] Davide Pala, Ivan Miro-Panades, and Olivier Sentieys. Freezer: A Specialized NVM Backup Controller for Intermittently-Powered Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–15, 2020. 37

[84] Masanori Natsui, Yasuo Noguchi, Mitsuo Yasuhira, Hideo Sato, Shoji Ikeda, Hideo Ohno, Tetsuo Endoh, Takahiro Hanyu, Daisuke Suzuki, Akira Tamakoshi, Toshinari Watanabe, Hiroaki Honjo, Hiroki Koike, Takashi Nasuno, Yitao Ma, and Takaho Tanigawa. A 47.14-µW 200-MHz MOS/MTJ-Hybrid Nonvolatile Microcontroller Unit Embedding STT-MRAM and FPGA for IoT Applications. *IEEE Journal of Solid-State Circuits*, 54(11):2991–3004, November 2019. 39, 41, 43

[85] Mehdi Baradaran Tahoori, Sarath Mohanachandran Nair, Rajendra Bishnoi, Lionel Torres, Sophiane Senni, Guillaume Patrigeon, Pascal Benoit, Gregory Di Pendina, and Guillaume Prenat. A Universal Spintronic Technology based on Multifunctional Standardized Stack. In *Proceedings of the 2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 394–399, Grenoble, France, March 2020. IEEE. ISSN: 1558-1101. 39, 43

[86] Noboru Sakimura, Yukihide Tsuji, Ryusuke Nebashi, Hiroaki Honjo, Ayuka Morioka, Kunihiko Ishihara, Keizo Kinoshita, Shunsuke Fukami, Sadahiko Miura, Naoki Kasai, Tetsuo Endoh, Hideo Ohno, Takahiro Hanyu, and Tadahiko Sugibayashi. A 90nm 20MHz fully nonvolatile microcontroller for standby-power-critical applications. In *Proceedings of the 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 184–185, San Francisco, CA, USA, February 2014. IEEE. ISSN: 2376-8606. 39, 41, 43

[87] Yiqun Wang, Yongpan Liu, Shuangchen Li, Daming Zhang, Bo Zhao, Mei-Fang Chiang, Yanxin Yan, Baiko Sai, and Huazhong Yang. A 3µs wake-up time nonvolatile processor based on ferroelectric flip-flops. In *2012 Proceedings of the ESSCIRC (ESSCIRC)*, pages 149–152, Bordeaux, France, September 2012. IEEE. 39, 40, 41

[88] Zhibo Wang, Yongpan Liu, Albert Lee, Fang Su, Chieh-Pu Lo, Zhe Yuan, Jinyang Li, Chien-Chen Lin, Wei-Hao Chen, Hsiao-Yun Chiu, Wei-En Lin, Ya-Chin King, Chrong-Jung Lin, Pedram Khalili Amiri, Kang-Lung Wang, Meng-Fan Chang,

and Huazhong Yang. A 65-nm ReRAM-Enabled Nonvolatile Processor With Time-Space Domain Adaption and Self-Write-Termination Achieving > 4x Faster Clock Frequency and > 6x Higher Restore Speed. *IEEE Journal of Solid-State Circuits*, 52(10):2769–2785, October 2017. 39, 41, 42

[89] Fang Su, Wei-Hao Chen, Lixue Xia, Chieh-Pu Lo, Tianqi Tang, Zhibo Wang, Kuo-Hsiang Hsu, Ming Cheng, Jun-Yi Li, Yuan Xie, Yu Wang, Meng-Fan Chang, Huazhong Yang, and Yongpan Liu. A 462GOPs/J RRAM-based nonvolatile intelligent processor for energy harvesting IoE system featuring nonvolatile logics and processing-in-memory. In *Proceedings of the 2017 Symposium on VLSI Circuits*, pages C260–C261, Kyoto, Japan, June 2017. IEEE. 39, 42

[90] Yongpan Liu, Fang Su, Yixiong Yang, Zhibo Wang, Yiqun Wang, Zewei Li, Xueqing Li, Ryuji Yoshimura, Takashi Naiki, Takashi Tsuwa, Takahiko Saito, Zhongjun Wang, Koji Taniuchi, and Huazhong Yang. A 130-nm Ferroelectric Nonvolatile System-on-Chip With Direct Peripheral Restore Architecture for Transient Computing System. *IEEE Journal of Solid-State Circuits*, 54(3):885–895, March 2019. 39, 40, 41

[91] Sudhanshu Khanna, Steven C. Bartling, Michael Clinton, Scott Summerfelt, John A. Rodriguez, and Hugh P. McAdams. An FRAM-Based Nonvolatile Logic MCU SoC Exhibiting 100% Digital State Retention at VDD = 0 V Achieving Zero Leakage With < 400-ns Wakeup Time for ULP Applications. *IEEE Journal of Solid-State Circuits*, 49(1):95–106, January 2014. Conference Name: IEEE Journal of Solid-State Circuits. 39, 41

[92] Steven C. Bartling, Sudhanshu Khanna, Michael P. Clinton, Scott R. Summerfelt, John A. Rodriguez, and Hugh P. McAdams. An 8MHz 75µA/MHz zero-leakage non-volatile logic-based Cortex-M0 MCU SoC exhibiting 100% digital state retention at VDD=0V with <400ns wakeup and sleep transitions. In *Proceedings of the 2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 432–433, San Francisco, CA, USA, February 2013. IEEE. ISSN: 2376-8606. 39, 41

[93] Sophiane Senni, Lionel Torres, Gilles Sassatelli, and Abdoulaye Gamatie. Non-Volatile Processor Based on MRAM for Ultra-Low-Power IoT Devices. *J. Emerg. Technol. Comput. Syst.*, 13(2):17:1–17:23, December 2016. 40

[94] Fang Su, Yongpan Liu, Yiqun Wang, and Huazhong Yang. A Ferroelectric Nonvolatile Processor with 46 µs System-Level Wake-up Time and 14 µs Sleep Time for Energy Harvesting Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(3):596–607, March 2017. Conference Name: IEEE Transactions on Circuits and Systems I: Regular Papers. 40, 41

[95] Michael Zwerg, Adolf Baumann, Rüdiger Kuhn, Matthias Arnold, Ronald Nerlich, Marcus Herzog, Ralph Ledwa, Christian Sichert, Volker Rzehak, Priya Thanigai, and Bjoern Oliver Eversmann. An 82µA/MHz microcontroller with embedded FeRAM for energy-harvesting applications. In *Proceedings of the 2011 IEEE International Solid-State Circuits Conference*, pages 334–336, San Francisco, CA, USA, February 2011. IEEE. ISSN: 2376-8606. 41

[96] Yongpan Liu, Zhibo Wang, Albert Lee, Fang Su, Chieh-Pu Lo, Zhe Yuan, Chien-Chen Lin, Qi Wei, Yu Wang, Ya-Chin King, Chrong-Jung Lin, Pedram Khalili, Kang-Lung Wang, Meng-Fan Chang, and Huazhong Yang. A 65nm ReRAM-enabled nonvolatile processor with 6× reduction in restore time and 4× higher clock frequency using adaptive data retention and self-write-termination nonvolatile logic. In *Proceedings of the 2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 84–86, San Francisco, CA, USA, January 2016. IEEE. ISSN: 2376-8606. 41, 42

[97] Tony F. Wu, Binh Q. Le, Robert Radway, Andrew Bartolo, William Hwang, Seungbin Jeong, Haitong Li, Pulkit Tandon, Elisa Vianello, Pascal Vivet, Etienne Nowak, Mary K. Wootters, H. S. Philip Wong, Mohamed M. Sabry Aly, Edith Beigne, and Subhasish Mitra. A 43pJ/Cycle Non-Volatile Microcontroller with 4.7µs Shutdown/Wake-up Integrating 2.3-bit/Cell Resistive RAM and Resilience Techniques. In *Proceedings of the 2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 226–228, San Francisco, CA, USA, February 2019. IEEE. ISSN: 2376-8606. 42

[98] Masanori Natsui, Daisuke Suzuki, Akira Tamakoshi, Toshinari Watanabe, Hiroaki Honjo, Hiroki Koike, Takashi Nasuno, Yitao Ma, Takaho Tanigawa, Yasuo Noguchi, Mitsuo Yasuhira, Hideo Sato, Shoji Ikeda, Hideo Ohno, Tetsuo Endoh, and Takahiro Hanyu. An FPGA-Accelerated Fully Nonvolatile Microcontroller Unit for Sensor-Node Applications in 40nm CMOS/MTJ-Hybrid Technology Achieving 47.14µW Operation at 200MHz. In *Proceedings of the 2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 202–204, San Francisco, CA, USA, February 2019. IEEE. ISSN: 2376-8606. 41, 43

[99] J. Wang, Y. Liu, H. Yang, and H. Wang. A compare-and-write ferroelectric nonvolatile flip-flop for energy-harvesting applications. In *The 2010 International Conference on Green Circuits and Systems*, pages 646–650, 2010. 40

[100] H. Kimura, T. Fuchikami, K. Maramoto, Y. Fujimori, S. Izumi, H. Kawaguchi, and M. Yoshimoto. A 2.4 pj ferroelectric-based non-volatile flip-flop with 10-year data retention capability. In *2014 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pages 21–24, 2014. 40

[101] M. Qazi, A. Amerasekera, and A. P. Chandrakasan. A 3.4pj feram-enabled d flip-flop in 0.13µm cmos for nonvolatile processing in digital systems. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 192–193, 2013. 40

[102] I. Kazi, P. Meinerzhagen, P. Gaillardon, D. Sacchetto, Y. Leblebici, A. Burg, and G. De Micheli. Energy/reliability trade-offs in low-voltage reram-based nonvolatile flip-flop design. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(11):3155–3164, 2014. 41

[103] A. Lee, C. Lo, C. Lin, W. Chen, K. Hsu, Z. Wang, F. Su, Z. Yuan, Q. Wei, Y. King, C. Lin, H. Lee, P. Khalili Amiri, K. Wang, Y. Wang, H. Yang, Y. Liu, and M. Chang. A reram-based nonvolatile flip-flop with self-write-termination scheme for frequent-off fast-wake-up nonvolatile processors. *IEEE Journal of Solid-State Circuits*, 52(8):2194–2207, 2017. 41

[104] M. Biglari, T. Lieske, and D. Fey. High-endurance bipolar reram-based nonvolatile flip-flops with run-time tunable resistive states. In *2018 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 1–6, 2018. 41

[105] T. Na, K. Ryu, J. Kim, S. H. Kang, and S. Jung. A comparative study of stt-mtj based non-volatile flip-flops. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 109–112, 2013. 41

[106] Weisheng Zhao, E. Belhaire, and C. Chappert. Spin-mtj based non-volatile flip-flop. In *2007 7th IEEE Conference on Nanotechnology (IEEE NANO)*, pages 399–402, 2007. 41

[107] D. Chabi, W. Zhao, E. Deng, Y. Zhang, N. B. Romdhane, J. Klein, and C. Chappert. Ultra low power magnetic flip-flop based on checkpointing/power gating and self-enable mechanisms. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(6):1755–1765, 2014. 41

[108] Sebastien Fontaine, Luc Filion, and Guy Bois. Exploring iss abstractions for embedded software design. In *2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, pages 651–655. IEEE, 2008. 48

[109] Muhammad Imran, Abas Md Said, and Halabi Hasbullah. A survey of simulators, emulators and testbeds for wireless sensor networks. In *2010 International Symposium on Information Technology*, volume 2, pages 897–902. IEEE, 2010. 48

[110] Sameer Sundresh, Wooyoung Kim, and Gul Agha. Sens: A sensor, environment and network simulator. In *37th Annual Simulation Symposium, 2004. Proceedings.*, pages 221–228. IEEE, 2004. 48

[111] Johannes Bauer and Felix Freiling. Towards cycle-accurate emulation of cortex-m code to detect timing side channels. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 49–58. IEEE, 2016. 49

[112] Brinkley Sprunt. The basics of performance-monitoring hardware. *IEEE Micro*, 22(4):64–71, 2002. 49

[113] Mohamad El Ahmad, Mohamad Najem, Pascal Benoit, Gilles Sassatelli, and Lionel Torres. Adaptive power monitoring for self-aware embedded systems. In *2015 Nordic Circuits and Systems Conference (NORCAS): NORCHIP & International Symposium on System-on-Chip (SoC)*, pages 1–4. IEEE, 2015. 49

[114] Erwan Lenormand, Thierry Goubier, Loïc Cudennec, and Henri-Pierre 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. 49

[115] Nam Ho, Paul Kaufmann, and Marco 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. 49

[116] Krishna T. Malladi, Mu-Tien Chang, John Ping, and Hongzhong Zheng. FAME: A fast and accurate memory emulator for new memory system architecture exploration. In *Proceedings of MASCOTS*, 2015. 50

[117] Mu-Tien Chang, I. Stephen Choi, Dimin Niu, and Hongzhong Zheng. Performance impact of emerging memory technologies on big data applications: A latency-programmable system emulation approach. In *Proceedings of GLSVLSI*, 2018. 50

[118] Davide Rossi, Francesco Conti, Andrea Marongiu, Antonio Pullini, Igor Loi, Michael Gautschi, Giuseppe Tagliavini, Alessandro Capotondi, Philippe Flatresse, and Luca Benini. PULP: A parallel ultra low power platform for next generation IoT applications. In *Proceedings of the IEEE Hot Chips Symposium (HCS)*, 2015. 50, 61

[119] PULPissimo. https://github.com/pulp-platform/pulpissimo. [Accessed July-2021]. 50

[120] STMicroelectronics. STM32L072 datasheet. https://www.st.com/resource/en/datasheet/stm32l072cz.pdf. [Accessed October-2022]. 57

[121] RISC-V International. RISC-V Cores and SoC Overview. https://github.com/riscvarchive/riscv-cores-list. [Accessed July-2022]. 58

[122] STMicroelectronics. STM32L0 series. https://www.st.com/en/microcontrollers-microprocessors/stm32l0-series.html. [Accessed July-2022]. 58

[123] STMicroelectronics. STM32F0 series. https://www.st.com/en/microcontrollers-microprocessors/stm32f0-series.html. [Accessed July-2022]. 58

[124] NXP Semiconductors. LPC800 series. https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc800-cortex-m0-plus-:MC_71785. [Accessed October-2022]. 58

[125] NXP Semiconductors. LPC1100 series. https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc1100-cortex-m0-plus-m0:MC_1392389687150. [Accessed October-2022]. 58

[126] Cypress Semiconductor. PSoC 4 family. https://www.infineon.com/cms/en/product/microcontroller/32-bit-psoc-arm-cortex-microcontroller/psoc-4-32-bit-arm-cortex-m0-mcu/. [Accessed October-2022]. 58

[127] Renesas. RX100 series. https://www.renesas.com/eu/en/products/microcontrollers-microprocessors/rx-32-bit-performance-efficiency-mcus. [Accessed October-2022]. 58

[128] Microchip. PIC32MM family. https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus/pic32-32-bit-mcus/pic32mm. [Accessed October-2022]. 58

[129] Microchip. SAM L family. https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus/sam-32-bit-mcus/sam-l. [Accessed October-2022]. 58

[130] LowRISC. https://github.com/lowRISC/ibex. [Accessed July-2021]. 61