Master's Thesis
"Master in Microelectronics:
Design and Applications of
Micro/Nanoscale Systems"

Design and implementation of a low-power low-cost smart embedded system for remote animal monitoring

Víctor Galvín Coronil

Advisors: Jorge Fernández Berni

Delia Velasco Montero

Date: November 6, 2023

Declaration of Authorship

I, Víctor Galvín Coronil, declare that this thesis titled, "Design and implementation of a low-power low-cost smart embedded system for remote animal monitoring" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UNIVERSIDAD DE SEVILLA

Abstract

Master in Microelectronics: Design and Applications of Micro/Nanoscale Systems

Design and implementation of a low-power low-cost smart embedded system for remote animal monitoring

by Víctor Galvín Coronil

This Master's thesis serves as the foundation for an innovative wildlife monitoring system, encompassing hardware design, firmware and software development, and offering insights into future directions. Leveraging the research group's extensive experience in research, development, and field deployment of wildlife technology solutions, the thesis has culminated in a device with versatile capabilities suitable for a wide range of applications. A central focus of this work is on energy efficiency, prioritizing low-power operation to facilitate extended field deployments and reduce maintenance requirements. The integration of AI capabilities is a core component, enabling real-time data analysis within the embedded system. The system's architecture is thoughtfully designed to seamlessly integrate data from diverse sources, including visual, acoustic, and environmental inputs, providing comprehensive insights into the natural world. Modularity in communication networks empowers the system to adapt to varying project requirements and network environments. The successful integration of hardware and software components enhances system performance, ensuring seamless data flow and efficient communication between different modules. The thesis underscores the importance of comprehensive testing, performance characterization, and real-world field testing for future research. In summary, this work represents a crucial step in the development of a versatile, energy-efficient, and AI-enhanced wildlife monitoring system with the potential to make substantial contributions to the field of conservation technology.

Acknowledgements

I would like to extend my sincere appreciation to Jorge and Delia for their invaluable guidance and support throughout the process of this Master's thesis. I also extend my gratitude to the dedicated members of the research group I have the privilege to work with, their camaraderie and support have transformed my work into a fulfilling and joyful experience in recent years.

To my biologist friends at the EBD, I am deeply grateful for your warm welcome and for granting me insight into the intersection of engineering and your field.

To my family, for their unwavering support and unconditional encouragement in all my pursuits.

To Ana Belén, for being a constant companion and a source of assistance during another significant chapter of my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The lost of biodiversity is one of the most critical problems nowadays. Ecosystems and their inhabitants face increasing challenges due to human activity, environmental changes and habitat loss. In order to reverse this trajectory and stop biodiversity loss, data acquisition and analysis is crucial to document the status of species and ecosystems, discern the root causes of extinction and environmental decline, evaluate the efficacy of mitigation efforts, and monitor the evolution of our natural surroundings and species to design functional strategies. In this context, technology can play a fundamental role in achieving all these tasks.

In the last few years, several devices available in the market like conventional data-loggers, camera traps or audio-loggers, included within the term *conservation technologies*, have been used by conservationists to collect data and then interpret information mostly using manual data analysis or conventional statistics techniques aided by software. As sensor technology has progressed and reduced its cost, these devices are now extensively employed and the amount and types of data collected have increased significantly. Consequently, manual data processing has become a principal obstacle to realizing the full potential of these technological advancements. Even leveraging on recent developments in cloud-based services, which can simplify the processes on data collection and off-site analysis, the bottleneck of transmitting extensive data flows to the cloud persists, specially for devices operating at remote locations with few network connection options.

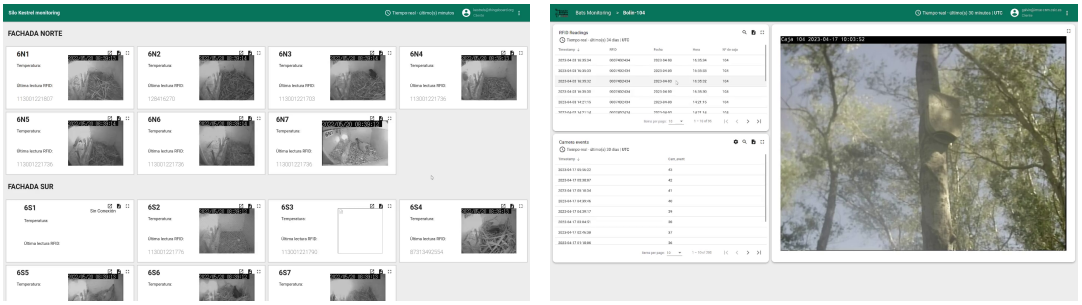
Developing "smarter" devices by using artificial intelligence (AI) and specifically deep neural networks (DNNs) can be the solution to this bottleneck. In this way, a device deployed in the field could collect all the information from the natural environment that surrounds it, perform automatically on-site analysis of these data collected and send just the information of interest that is going to be useful to researchers, managers and conservationists. Achieving these characteristics in a robust way in such changing environments, with reduced access to power and communication networks, and for solutions as variable as those we can find in the study of biodiversity, is a great challenge. The vast majority of embedded platforms that exist on the market for these tasks are designed for specific applications and focused on the industry, so their use in this area is not always favorable.

In this Master's thesis, a custom embedded device design is presented, designed to fulfill the features described above, that is, data collection and on-site analysis, and subsequent transmission of information through the available communication infrastructure. The system is configurable enough to be adapted to different applications and environments with the ultimate goal of implementing smarter monitoring of biodiversity.

1.1 Background

The advisors of this thesis are part of a research group of the *Institute of Microelectronics of Sevilla (IMSE)* that has been working for over 15 years on ultra-low-power embedded devices and artificial intelligence applied to environmental monitoring. An example of their contributions in this area is the V-MOTE project [1], where a power-efficient hardware chip was designed to implement vision in the nodes of a wireless sensor network (WSN). This chip was applied into a system for early forest fire detection. More recently, the research was focused on the integration of artificial intelligence at the edge for applications such as smart camera trapping, as reported in [2].

To develop this kind of projects is crucial a strong collaboration with the experts in this area who are potential adopters of this technology. In this regard, several collaborations have been carried out with the *Doñana Biological Station (EBD)* in the last few years. The most recent one, in which the author of this Master's thesis was involved as an engineer, is the project *SUMHAL (Sustainability for Mediterraneam Hotspots in Andalusia integrating LifeWatch ERIC)* [3], specifically in WP4, dedicated to *Combining field data, citizen science and IoT to monitor anthropogenic impacts on Andalusian biodiversity and society*. In this project, a complete solution for remote monitoring of species was designed including developments in hardware, software, network architecture and algorithms, everything integrated in an IoT cloud platform deployed in the EBD servers. The specific aim of these solutions was the advance monitoring of two species: The great noctule bat (*Nyctalus lasiopterus*) and the lesser kestrel (*Falco naumanni*). Both are listed as endangered species, vulnerable and "least concern", respectively, by the International Union for Conservation of Nature (IUCN) [4].



(A) Kestrels dashboard. (B) Bats dashboard.

FIGURE 1.1: Cloud-based platform deployed in SUMHAL to visualize information in dashboards.

As a key component of this solution, smart nests and boxes equipped with sensors were developed. These sensors allow monitoring temperature, humidity, presence (utilizing PIR technology), image capture, animal weight via load cells, and RFID (Radio Frequency Identification) readers to identify individual animals. The collected data are processed on-site by a microcontroller, thereby detecting the presence of individuals and selectively transmitting only the pertinent information. Furthermore, these data are seamlessly integrated into a cloud-based service alongside other input sources, such as live video streams from CCTV cameras. Off-site processing is then applied to summarize information over time and display only the most relevant events. Under this approach, information collected by different sources –

including the sensors – is aggregated and easily accessible through user-friendly dashboards (Figure 1.1) featuring graphical elements, mobile notifications configuration, periodic email reports, etc. This provides biologists with valuable information about the species they are studying.

Concerning the hardware, which is the principal focus of this thesis, a first prototype of custom embedded system for animal monitoring was designed; it is shown in Figure 1.2. Due to timeline restrictions of the project, which required a functional system to be deployed in the field in the short-term, commercial off-the-shell embedded hardware was employed to design this first prototype. The processing core is an embedded development board called *Adafruit Feather M0 Adalogger* [5]. This device comprises a ATSAMD21G18 ARM Cortex M0 microcontroller [6] and additional features that met the conditions needed for this system, such as a micro SD card socket, built-in lithium battery charger circuit and enough GPIO pins and communication protocol interfaces.

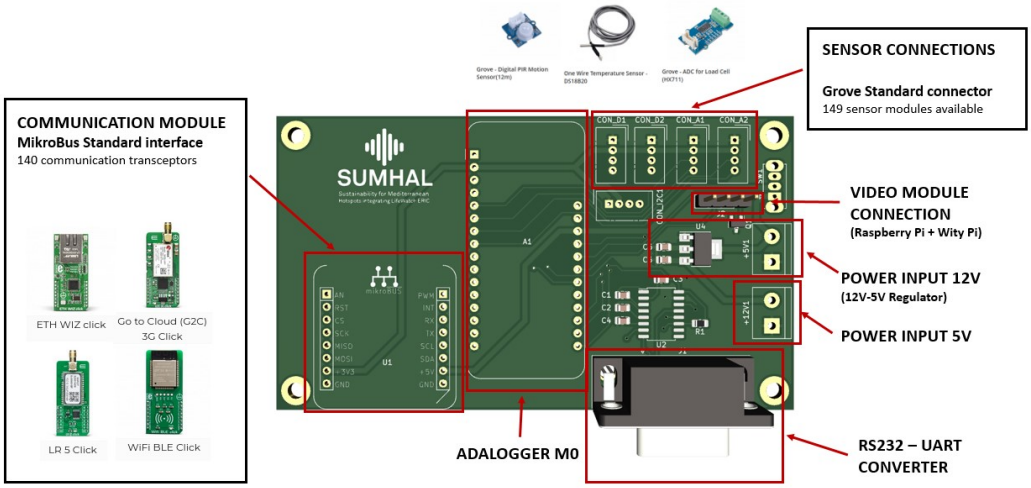


FIGURE 1.2: Modular custom embedded hardware system designed in project SUMHAL [3].

The designed board contains a LDO regulator to accept power inputs up to 12V and can alternatively be powered from a 5V input. Regarding communication, an standard connector called *mikroBUS* [7] was used. This standard has more than 140 communication transceivers available at market, which gives a huge range of possibilities. Following the same strategy for the sensor interface, five *Grove* [8] connectors were added to the board. Around 150 sensor modules complying with this standard are available in the market. The RFID reader modules being used at the moment for registering ISO animal transponders implement the RS232 communication protocol, so a RS232 to UART TTL 3.3V circuit was added to the design of the system to interface with these readers. Finally, given that image capture is an important source of information for biologists, and taking into account that for a low-level microcontroller as the one used in the system is difficult to deal with image acquisition and processing while performing other tasks, an interface to a Raspberry Pi

board was provided in the system. Raspberry Pi boards feature powerful processing capabilities. As an additional advantage, the research group has experience in employing them to build smart monitoring systems. A drawback of these platforms is their high energy consumption even when idle. This interface in the modular system designed was intended to be used with an add-on board to the Raspberry Pi called Witty Pi [9]. This board includes a power management circuit that can be externally controlled. This will give the possibility to completely turn on or off the powerful capabilities of the Raspberry Pi board when needed to capture and process images.

Several units of this system were deployed at different locations adapted to smart nests for kestrels and boxes for bats, as seen in Figure 1.3. One of the deployment locations was the ICTS-RBD (Singular Scientific-Technical Infraestructure of Doñana), where a large infrastructure of communications and power facilities is available in certain areas of the Doñana National Park. For instance, Ethernet wire connection or WiFi are available for the deployment of the devices, as well as 12V power supplies. On the other hand, the study location for kestrels is a remote area close to the town of la Palma del Condado, where there is only weak mobile coverage and no power options. These scenarios exemplify the great configurability required in the system, and the variety of sensors for each specie under study.

(A) Preparation of devices. (B) Device deployed in smart nest.

FIGURE 1.3: Deployment preparation of devices for bat boxes (a) and smart nest for kestrel (b)

The operation of the system shown in Figure 1.2 turned out to be very fruitful, providing a large set of valuable data during the project. Nevertheless, several improvement opportunities were found, such as the capability of processing additional sources of sensor information, increasing the power efficiency of the system, and extra processing capabilities to perform on-device DNN-based algorithms. With these objectives among others, the group started the project *ULTIMATE (smart mULTIsensor eMbedded platform for advanced nATurE monitoring)*, under which this thesis has been developed. Passive acoustic monitoring using low-power DNN-based audio analysis for identification of bird calls, an automatic system for weighing and identifying the Scottish wildcats and a full remote and monitored trap system for feral cats are some of the application scenarios covered in this project.

1.2 State of the art

This section provides an overall view of the state-of-the-art of existing hardware technologies for wildlife monitoring. First, the focus is put on commercial devices available in the market associated to the term *conservation technologies*. Subsequently, some of the embedded platforms that could potentially be used in the development of these types of solutions are analyzed.

The term *conservation technologies* includes a huge variety of hardware and software solutions, ranging from camera traps, data-loggers, and radio location tracking to UAVs, desktop and smartphone applications, geospatial data tools, and environmental DNA analysis, among others [10]. The survey conducted in [11] analyzes the most frequently used conservation technologies. Among these, those that involve the deployment of devices in the field, that is, camera trapping, passive acoustic monitoring and networked sensors, can be considered flagship conservation technologies.

(A) Conventional camera trap. (B) Conventional Audiologger.

FIGURE 1.4: Conventional conservation technology devices.

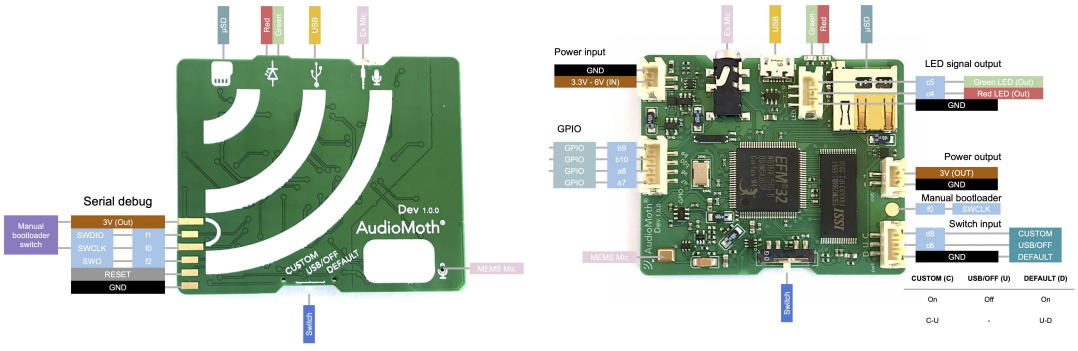
Camera traps represent the dominant segment in the market of available wildlife monitoring devices. These devices typically feature high-resolution image sensors and leverage passive infrared motion sensors (PIR) to initiate recording upon motion detection. Additionally, they employ infrared LED arrays for nocturnal recording, ensuring minimal disturbance to the environment and collecting a great deal of information on the behaviour of species under study. The vast majority of camera traps are designed primarily for data storage, capturing bursts of images or videos sequences once the PIR is triggered, but without on-site data processing. Recent advancements have seen the integration of wireless communication capabilities into specific models. This innovation allows users to remotely access data and reconfigure devices, reducing the need for physical intervention.

Audio-loggers represent another prominent technology within passive acoustic monitoring (PAM). These devices are designed ad-hoc for collecting acoustic data in natural habitats, mirroring the non-invasive approach of camera traps. Audio-loggers typically feature sensitive microphones positioned to record ambient sounds, including vocalizations and acoustic signals emitted by wildlife, which gives a lot of

information, especially for species that communicate through acoustic signals. Examples of common conservation devices available in the market, including a conventional camera trap and an audiologger, are illustrated in Figure 1.4.

A significant challenge associated with conventional conservation technologies is the huge volume of data they generate, which represent a substantial administrative and processing burden on users. While these technologies have proven valuable for wildlife monitoring, the manual handling and analysis of vast datasets can be labor-intensive and time-consuming. To address this challenge, a new trend of "smart" conservation technologies is emerging, offering innovative solutions that streamline data management and analysis, thereby revolutionizing the field.

One widely-spread solution of this new approach is the Audiomoth project [12], an academic work published and commercialized under an open-source philosophy. Audiomoth introduces a cost-effective and efficient device for passive acoustic monitoring, with the key feature of being open-source and well-documented to be programmed, which emphasizes the implementation of solutions for automated data processing and analysis. This device has been widely-used in the aforementioned IMSE's group for several projects, such as the work presented in [13], where a neural network was implemented in the Audiomoth itself for the identification of the most characteristic call of the lesser kestrel, discerning other sounds and calls of other species. This device features a EFM32 Gecko 32-bit Microcontroller based on the ARM Cortex-M3 core with features oriented to battery-operated applications, such as very low sleep current with RTC (0.9µA) and short wake-up time from energy-saving modes. Figure 1.5 provides additional details about the components comprising the Audimoth system.



(A) Top view. (B) Front view.

FIGURE 1.5: Parts of the Audiomoth smart datalogger PCB.

In the domain of camera traps, while devices as prevalent as Audiomoth are yet to emerge, the landscape is witnessing a surge in emerging projects, stemming from both academic research and industry. An illustrative case of a device poised for commercialization is *Sentinel* [14], a product developed by *Conservation X Labs*, currently in its pre-release phase. The *Sentinel* system empowers standard commercial camera traps with AI capabilities, automating data analysis and facilitating real-time alert generation. Another project, emerging from the academic sphere, is detailed in [15]. This project introduces a "smart bridge" hardware system, specifically designed to interface with a commercial camera trap, thereby integrating artificial intelligence

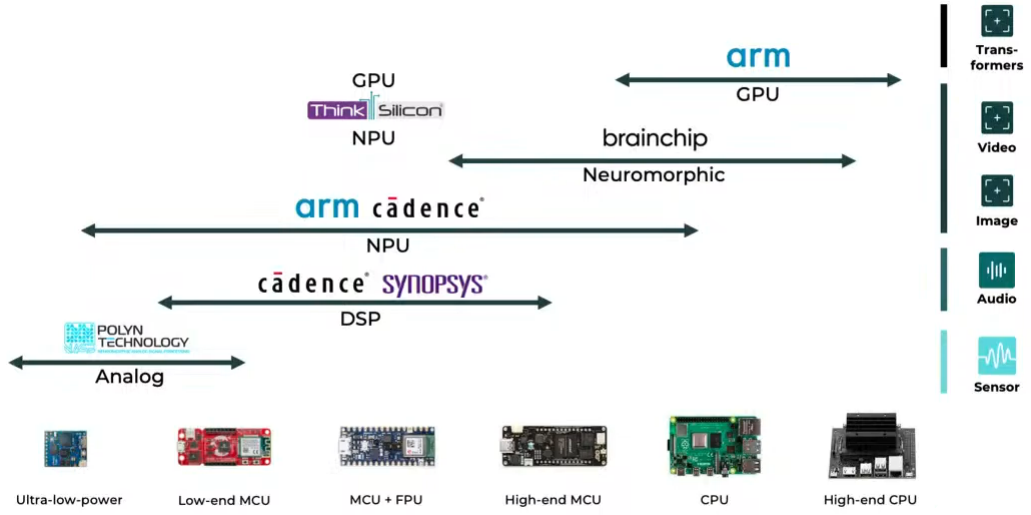


FIGURE 1.6: Overview of the latest embedded platforms on the market and their applications (*Imagine 2022, Conference for Edge AI by Edge Impulse*).

processing. The resultant system excels in the generation of real-time alerts transmitted via satellite networks. Numerous analogous projects in this niche domain harness the potential of AI to autonomously classify camera trap images directly at the data collection point. These developments are well-documented in the literature, including studies by the research group in which this thesis has been developed [16–18].

Collectively, these initiatives highlight the growing traction in the direction of intelligent, data-efficient, and automated conservation technologies, marking an exciting frontier in the realm of wildlife monitoring.

Following the above overview on both commercial and academic initiatives in wildlife monitoring solutions, embedded platforms, vital for the development of such solutions, are examined from a technical point of view. Figure 1.6, presented at [19] provides a clear picture of the intricate landscape of embedded platforms within the realm of edge AI. It effectively categorizes various types of embedded platforms along the x-axis based in its power consumption, from ultra-low-power devices to high-end CPU platforms going through the different levels of microcontroller-based systems (MCU). Simultaneously, it maps a diverse spectrum of applications along the y-axis, including sensors, audio processing, image analysis, video processing, and transformer-based tasks. This graphical representation, which highlights the latest solutions from leading silicon manufacturers, offers valuable insights into the dynamic interplay between platform types and their corresponding applications in the context of edge AI. By leveraging this graphical baseline, a detailed analysis of these embedded platforms is done. In this analysis, the pivotal role these platforms play in the development of cutting-edge wildlife monitoring solutions is underlined, and their their diverse capabilities and suitability for distinct project requirements are clarified.

Ultra-low-power devices that specialize in analog signal processing represent

a significant stride in the pursuit of energy-efficient solutions. One notable example is Polyn Technology's Neuromorphic Analog Signal Processing (NASP) [20], a groundbreaking approach that mimics the human brain's neural network to process analog signals. These devices stand out due to their unparalleled ability to minimize power consumption while performing real-time signal processing tasks. These ultra-low-power devices are particularly well-suited for specialized sensing tasks, such as monitoring the physiological states of animals through biologgers. When attached to wildlife, these devices can efficiently collect and process analog signals from various sensors, offering insights into the animals' behaviors, health, and environmental interactions.

The embedded system exemplified in SUMHAL project, presented in the background section, serves as a prime example of a low-end MCU- (Microcontroller Unit) based embedded platform. These MCUs, predominantly part of the ARM Cortex-M processor family, are renowned for their energy efficiency and adaptability, making them particularly well-suited for specific applications in wildlife monitoring. This platform excels at collecting and processing low-level sensor data, including basic audio processing, while simultaneously minimizing power consumption. The proper use of timers and sleep modes inherent to these microcontrollers allows for efficient energy management, thereby extending the battery life and prolonging field operation. However, it is worth noting that low-end MCUs may not be ideal for handling resource-intensive tasks such as deep neural network (DNN)-based algorithms or image processing, which require more robust computational capabilities. Yet, within the Cortex-M processor family, we encounter variants like the Cortex-M4, which introduce more powerful cores with added capabilities, including a Floating-Point Unit (FPU) and Digital Signal Processing (DSP) Extension. This enhancement constitutes a significant leap, enabling the group of MCU+FPU devices to implement advanced artificial intelligence techniques. Moreover, in terms of software compatibility, these devices are often compatible with frameworks like TensorFlow Lite for microcontrollers, which significantly streamlines the development process. It is within this category that devices like Audiomoth find their niche, aligning with more advanced AI capabilities and offering a streamlined development pathway for wildlife monitoring applications.

High-end MCUs represent a significant advancement in computational power, memory capacity, and additional capabilities, making them well-suited for wildlife monitoring. These devices excel in complex tasks, including AI and image processing, thanks to integrated graphical hardware accelerators. They offer higher levels of development abstraction, enabling programming in languages like Python (with MicroPython) and the use of machine learning frameworks such as TensorFlow Lite. The research group behind this thesis has dedicated efforts to harness the potential of high-end MCUs. These devices balance computational strength with low power consumption, crucial for extended wildlife observation. The group has tested devices like openMV H7 Plus [21], Sony Spresense [22], and Arduino Portenta H7 [23], which often feature multi-core architectures for enhanced processing capabilities, such as the dual Cortex-M7+M4 configuration in the STM32H747XI (Arduino Portenta H7) and the six Cortex-M4F cores in the Sony Spresense.

In the realm of CPU-based devices, Raspberry Pi boards have emerged as the most prominent and widely-adopted choice for prototyping AI algorithms at the edge. Complementing the Raspberry Pi, a set of similar single-board computers

populate the landscape of edge computing. These devices consistently feature microprocessors from the Cortex-A series, offering a versatile spectrum of solutions for handling complex computational tasks. Their capabilities extend to hosting rich operating system platforms and seamlessly supporting a multitude of software applications. This versatility empowers developers to harness the full potential of high-level AI development and implementation, significantly broadening the horizons of wildlife monitoring applications. Notably, these systems offer impressive processing power and capabilities without imposing excessively high power consumption. Nevertheless, it is vital to acknowledge that they lack low-power characteristics, meaning that even in idle states, a system like the Raspberry Pi may still consume power on the order of hundreds of milliamperes (mA), a consideration that becomes significant in field deployments where reduced energy consumption is a fundamental specification to be met. Finally, high-end CPU devices like the NVIDIA Jetson family [24] leverage powerful GPU capabilities to tackle the most advanced AI tasks. However, in the context of wildlife applications, the significantly higher power consumption of these devices often makes them less practical for field deployments.

1.3 Motivation and Objectives

The development process for embedded systems used in the field of wildlife monitoring is both extensive and resource-intensive, requiring various domains of expertise and substantial time investment. Adapting and redeveloping systems for the diverse needs of wildlife observation projects can pose a significant challenge. To address this challenge and facilitate the expansion of conservation efforts, the research group recognizes the imperative need for a highly configurable device that can be easily tailored for different specifications, environments, and network requirements. This flexibility is pivotal for the group to efficiently engage in more substantial and diverse conservation initiatives, amplifying their contribution to biodiversity preservation.

Building upon the experience garnered from the development of the original embedded system in the SUMHAL project and subsequent wildlife monitoring projects detailed in the background sections, the research group is primed for a comprehensive redesign of their embedded system. This evolution aims to produce a more capable, low-power, and cost-effective smart embedded system designed specifically for remote animal monitoring.

This Master's thesis serves as the initial phase of the embedded system development process, focusing on requirements analysis, architectural design, hardware development, hardware-software integration. The research objectives for this thesis encompass five critical dimensions:

1. **Energy Efficiency:** Prioritizing low-power operation to extend field deployment capabilities and minimize maintenance needs.

2. **AI-Enhanced Architecture:** Develop a hardware framework that enables advanced AI capabilities, allowing real-time data analysis and decision-making within the embedded system.

3. **Multi-Modal Data Integration:** Designing the system to be able to collect, process, and fuse data from various sensors, including visual, acoustic, and environmental inputs, to provide comprehensive insights into wildlife and habitat conditions.

4. **Communication Networks Modularity:** Enabling modularity in communication networks to be easily adapted to diverse project requirements and network environments.

5. **HW-SW Integration and Compatibility with AI Frameworks:** Ensuring seamless integration between hardware and software components and providing compatibility with AI frameworks, such as TensorFlow.

Chapter 2

Hardware Development

2.1 Requirements Analysis and Specifications

The initial phase of embedded system design involves requirements analysis, a critical step in which a comprehensive understanding of the system's functional needs is acquired. The goal is extracting essential technical specifications from these requirements, thereby setting the foundational basis for the system's design. In this subsection, various requirements, derived from the research group's experience working with conservation organizations and past development endeavors, are meticulously examined and translated into precise technical specifications that the system must adhere to.

2.1.1 Processing capabilities

The requirement for advanced image and audio processing, as described in Section 1.2, is associated to the need for devices with robust AI capabilities. While high-end MCUs demonstrated their suitability for specific applications, it becomes evident that achieving powerful AI capabilities within an embedded device – including, for instance, seamless integration with the TensorFlow Lite framework – requires a CPU-based system running a Linux embedded operating system.

To optimize the fusion of extensive processing capabilities and low-power operation, the proposed solution involves a hybrid architecture comprising both an MCU and a CPU. In this architectural model, the MCU takes charge of data acquisition, preprocessing, and network data management, excelling in efficiency. Meanwhile, the CPU serves as a dedicated processing acceleration module, stepping in once environmental data are acquired to execute in-depth processing tasks employing AI-based methodologies. This collaborative division of labor ensures high-performance levels while maintaining energy efficiency.

Interestingly, some deployment scenarios, contingent on their specific data and processing needs, may not demand the full-scale processing power offered by the CPU. In such instances, the MCU's capabilities may suffice. Therefore, incorporating a modular approach that facilitates the attachment or detachment of the CPU module emerges as a versatile feature for the device. This modularity offers adaptability, enabling the system to cater to a spectrum of deployment scenarios, from those necessitating extensive AI processing to those where MCU capabilities align with the requirements. This design choice harmonizes with the device's adaptability to changing field conditions and conservation project needs.

2.1.2 Multi-modal data integration

One of the critical requirements of the system is to establish a robust framework for accommodating a diverse range of input data sources encountered in the field, encompassing visual, acoustic, and environmental data. To attain this level of versatility, the system will leverage the Grove interface utilized in the prior developments, thereby offering support for a broad spectrum of digital interfaces, including I2C, UART, and SPI. The Grove interface is compatible with over 150 sensor modules available in the market, addressing various physical parameters. This foundational choice facilitates seamless expansion and customization according to specific field requirements.

In addition to the Grove interface, supplementary interfaces tailored to enhance compatibility with visual and acoustic data sources will be explored. Notably, the Camera Serial Interface (CSI), a standard defined by the Mobile Industry Processor Interface (MIPI) Alliance, will be considered as a pivotal addition, potentially serving as the bridge between the system and camera-based deployments. This interface allows the connection of a camera to the host processor, enabling the acquisition of high-quality visual data.

For acoustic data integration, analog interfaces must be considered, potentially featuring pre-amplifier opamp-based circuitry followed by analog-to-digital converters (ADCs) to accommodate both stereo and mono standard analog microphones. Furthermore, support for digital-ready microphones with direct digital interfaces like I2S will be explored. The inclusion of these options underscores the commitment to flexibility and adaptability in capturing and processing acoustic data, catering to various field scenarios and project requirements.

2.1.3 Data storage

Considering the prolonged deployments of the system in remote areas, where robust network connections are not guaranteed for transmitting the collected data, it is imperative to incorporate an efficient data storage solution. A preferred choice within the realm of conservation technology devices is the utilization of SD cards. It offers notable advantages, allowing users to extract and replace them for data retrieval when network communication is unavailable. Moreover, they can serve as a dependable back-up solution in the event of potential data loss due to environmental circumstances prevalent in these challenging field environments. Therefore, one potential data storage option to be included in the device is a micro SD card slot.

Communication modularity

Continuing from previous successful developments, the MikroBus standard will be employed to endow the system with communication modularity. This approach ensures compatibility with a wide range of sensor modules and accessories, offering flexibility in adapting the system to varying field conditions and research objectives.

2.1.4 Power supply

Power supply requirements are fundamental to the functionality and performance of the field-deployed device, as they determine its autonomy, reliability, and adaptability to the diverse environmental conditions. In this context, several power supply considerations are relevant:

Battery power: Given that this device will likely operate in remote field locations without access to an electrical grid, the ability to rely on battery power is crucial. Among the available battery technologies, lithium polymer (LiPo) batteries stand out due to their high energy density, making them the preferred choice. These batteries offer a dependable and readily-accessible source of power, ensuring uninterrupted operation.

Variable DC input: Field deployments often necessitate flexibility in power sources. The device should be compatible with variable DC inputs, allowing operation in deployments where there is an existing power infrastructure. This adaptability is essential for seamless integration into diverse deployment scenarios.

Solar panel integration: To further enhance the device's autonomy and reduce its reliance on traditional power sources, the incorporation of solar panels is a practical solution.

USB power: The inclusion of a USB port is essential for facilitating system programming and debugging. This port not only serves as a communication interface but also acts as a power source for these activities. Therefore, ensuring that the device can receive direct power through this port is a critical aspect of its design.

Efficient charging circuit: Given the reliance on LiPo batteries and the potential for solar panel usage, an efficient charging circuit is a vital component. This circuit is instrumental in achieving self-sufficiency from solar panels and maintaining battery health by ensuring optimal charging. It offers a practical solution for extended field deployments and recharging between missions.

Stable Voltage Levels: The device's internal and external components, including the MCU, CPU, sensors, and communication modules, require stable voltage levels to operate effectively. Both 3.3V and 5V power supplies are essential for these components. Incorporating high-efficiency circuits into the power supply unit is necessary to maintain consistent and reliable voltage levels, which, in turn, contribute to the device's overall performance.

Power control: To maximize energy efficiency and achieve ultra-low power consumption during idle periods, the power supply unit should feature control mechanisms. These mechanisms enable the selective activation and deactivation of components, minimizing power consumption when specific elements of the system are not in use.

2.1.5 Environmental considerations

Wildlife monitoring often involves the deployment of devices in challenging environmental conditions. It is therefore mandatory to carefully consider the operational temperature ranges and humidity tolerance of the components used in the device's design. To ensure the device's robustness, internal sensors, such as temperature and humidity sensors, or accelerometers, can be integrated into the device's board. These sensors serve to monitor the device's condition, detect extreme environmental factors, and alert about potential harm.

Furthermore, it is essential to account for the need to enclose the device in a waterproof box for most deployments. Therefore, the design of the PCB should include specific considerations regarding the dimensions and placement of connectors. These aspects play a crucial role in simplifying the attachment and manipulation of the device within the waterproof enclosure, making it easier to adapt it to varying field conditions and ensuring the device's long-term functionality and reliability.

In summary, the requirements analysis and specifications phase has provided a comprehensive understanding of the technical needs and environmental considerations for the development of the embedded system. Table 2.1 outlines the specifications derived from this analysis, which will guide the subsequent phases of the design process.

Category	**Specification**
Power Supply	USB, Battery, Solar Panel, Variable DC Input LiPo Charging Circuit 3.3V and 5V Power Supply Controlled load switch
Processing	AI Capabilities Low-Power Hybrid Architecture (MCU+CPU)
Multi-Modal Data Integration	Grove Interface Camera Serial Interface (CSI) Audio Interfaces (Analog and Digital)
Data Storage	Micro SD Card Slot
Communication Modularity	MikroBus Standard
Environmental Considerations	Internal temperature-humidity sensor to measure condition Internal accelerometer for free-fall detection PCB Design for Easy attachment and manipulation within the enclosure

TABLE 2.1: Specifications based on requirements analysis.

Keeping these specifications in mind, the hardware development and software design phases will be undertaken to transform these requirements into a fully functional, efficient, and versatile embedded system for wildlife monitoring.

2.2 Hardware architecture

This section describes the hardware architecture of the proposed embedded system, highlighting the components and their interconnections. Figure 2.1) provides a functional diagram of the full system, showcasing all the elements and their interconnections, as well as specifying the interfaces required for each component.

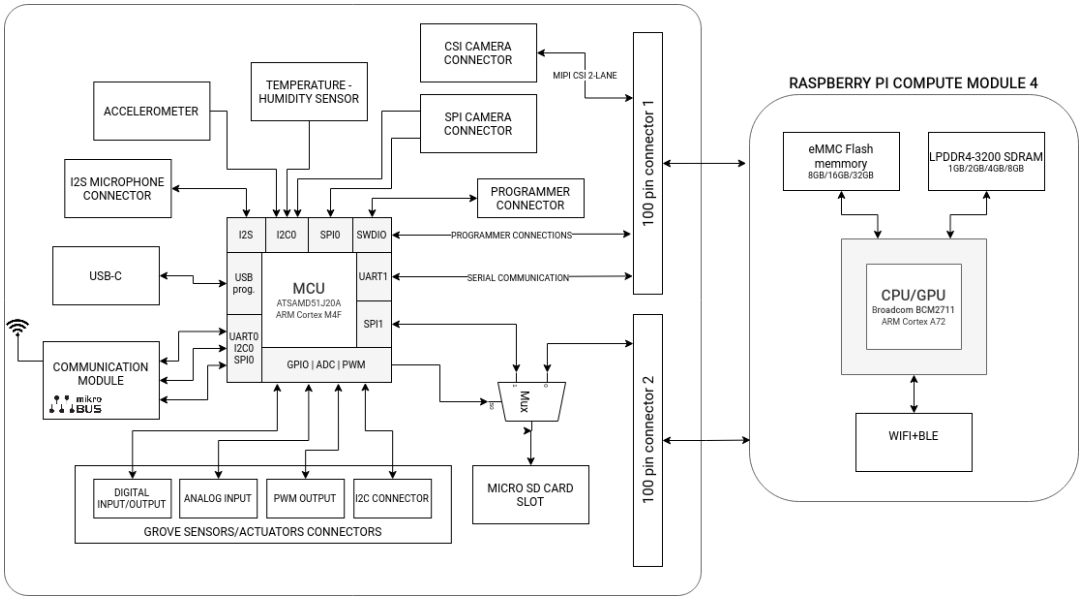


FIGURE 2.1: Functional diagram of the full system.

Within this diagram, several key elements and connections are highlighted. The system follows a hybrid architecture, combining a microcontroller unit (MCU) and a central processing unit (CPU) – being the CPU module optional as a plug-in. The primary MCU selected for the system is the SAMD51 microcontroller, featuring an ARM Cortex-M4 core with a floating-point unit. This MCU can manage the various interfaces and elements connected to it and even perform lightweight machine-learning (ML) tasks with input data. Consequently, it is well-suited for handling scalar inputs from sensors and for realizing basic audio and image acquisition and filtering.

For applications demanding more robust AI-based processing, the CPU module can be attached to enhance the system's capabilities. The chosen CPU module is the Compute Module 4 (CM4), which is a System-on-Module (SoM) providing the computational power of the Raspberry Pi 4 within a compact form factor designed for deeply embedded applications. To facilitate communication between the two modules, the system includes programmer connections, a serial communication interface, and an SD card interface shared through a multiplexer/demultiplexer (mux/demux) circuit. These interfaces enable seamless data sharing and bilateral communication, making the entire system capable of acting as a whole.

The complete device – with the CM4 module connected in particular – offers ample resources to handle a wide range of data sources, processing tasks, and AI capabilities. The design's modularity enables easy adaptation to less demanding

projects, allowing on-the-fly adjustments in terms of features and costs. This adaptability makes it an ideal hardware system for various applications within the field of wildlife monitoring.

The subsequent sections provide details about each element of the system, giving insights into the electronic design process and component selection. This offers a comprehensive understanding of the hardware development and how the system's architecture aligns with the technical requirements specified before.

2.3 Data acquisition and low-power processing unit

2.3.1 Microcontroller

The selection of the microcontroller for an embedded system is critical to achieve the right balance between power efficiency and processing capacity. In this context, the ARM Cortex cores emerge as the default choice for MCU-based devices, particularly the Cortex-M family, which is well-optimized for low-cost and energy-efficient integrated circuits.

In the previous version of the embedded system developed for the SUMHAL project [3], a Cortex M0+ core – featured in the SAMD21 microcontroller – was employed. While this microcontroller offers a suite of useful interfaces, it lacks the processing capabilities required for implementing ML tasks. To accommodate ML processing in the selected MCU, an upgrade within the Cortex-M family became essential.

The subsequent models within the Cortex-M family, such as the Cortex M4, introduce features like the Floating-Point Unit (FPU) and Digital Signal Processing (DSP) Extension. The inclusion of an FPU is particularly significant as it eliminates the need for emulating floating-point instructions in software, substantially reducing processing overhead. These features are instrumental in facilitating more advanced processing capabilities in the microcontroller.

Upon selecting the Cortex M4 as the core model, the next step involves identifying specific microcontrollers that incorporate this core. Multiple models are available from various manufacturers, with many of them sharing similar characteristics. The differentiating factor often lies in the development tools and ecosystem provided by each manufacturer. Given our prior experience with the development process using the SAMD21, the ATSAMD51J20A [25], which leverages the Cortex M4 core, was chosen as the primary microcontroller for the system. This selection aligns with the goal of enhancing processing capabilities while maintaining a power-efficient profile. When considering ML capabilities, several devices featuring the ATSAMD51J20A MCU are supported by TensorFlow Lite for Microcontrollers [26], emphasizing its compatibility with advanced AI and ML frameworks. In terms of power consumption, this MCU offers a robust array of low-power modes, including idle, standby, hibernate, and back-up. The device can operate with minimal power draw, even reaching values as low as 3µA in hibernate mode, where the internal real-time clock (RTC) continues to run.

Regarding the circuit design around the microcontroller, Figure 2.2 shows several key components and connections. Notably, an inductor is strategically placed

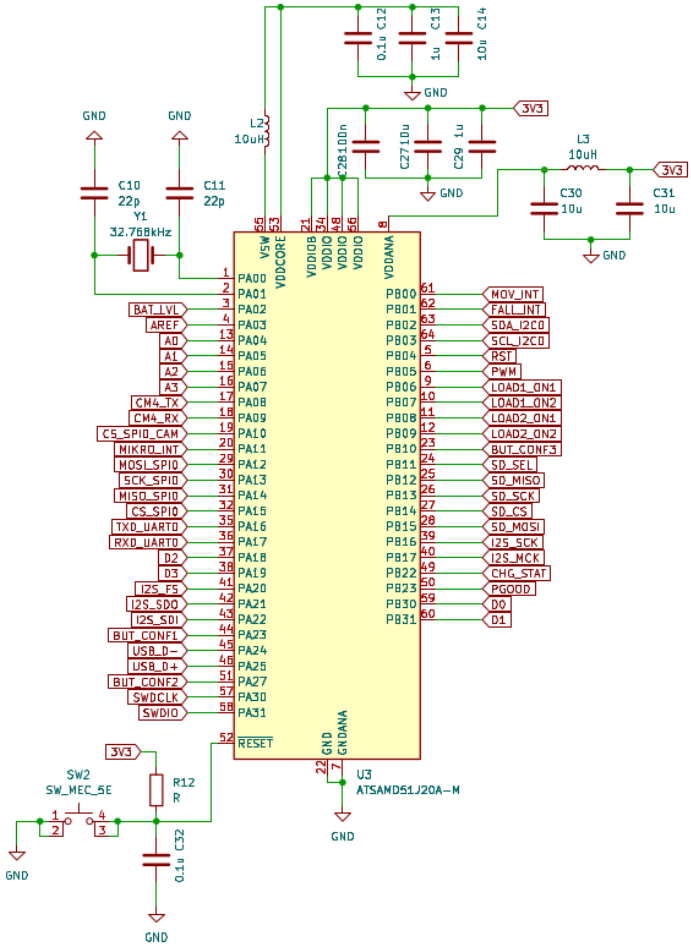


FIGURE 2.2: Schematics of the microcontroller unit

between VSW and VDDCORE, serving the purpose of using the internal regulator in Switching mode. This particular mode offers the highest level of power efficiency when both the CPU and peripherals are operational. Decoupling capacitors are integrated into the design, primarily serving the role of noise filtering. These capacitors help maintain a stable and clean power supply for the microcontroller and associated components, ensuring the reliability of data processing and functionality. Moreover, an LC (inductor-capacitor) filter is employed for VDDANA, which represents the analog power input. This filter serves the essential function of reducing high-frequency noise and interference in the analog components of the system. Maintaining a clean and noise-free analog power source is crucial for accurate data acquisition. Additionally, a reset button in pull-up configuration is incorporated into the design. This allows for the device to be reset and placed in boot mode, facilitating ease of debugging, programming, and reconfiguration when necessary.

When designing the microcontroller's pin allocation for the SAMD51J20, careful consideration is given to the 64 available pins, each serving various functionalities. A notable feature of this microcontroller is its internal multiplexing capability. By default, each pin can be controlled by the PORT as a general-purpose input/output (I/O). Additionally, these pins can be alternatively assigned to one of the peripheral functions, offering a high degree of configurability.

The peripheral functions encompass a wide range of options, including common

TABLE 2.2: Distribution and functionallity of each SERCOM port used in the MCU.

SERCOM	Interface	Function
SERCOM 0	UART0	Communication between MCU and CPU (CM4)
SERCOM 1	UART1	MikroBus network communication module
SERCOM 2	SPI0	MikroBus network communication module SPI camera connector
SERCOM 4	SPI1	SD Card
SERCOM 5	I2C	MikroBus network communication module Internal/external I2C sensors
SERCOM 7	UART	Externals sensor or other modules in Grove connector

digital interfaces like UART, SPI, and I2C, as well as more specialized functions such as timer signals, RTC, USB, CAN, I2S, and more. These functions can be allocated to specific pins as per the pinout directions specified in the microcontroller's datasheet.

For clarity and reference, Table 2.2 highlights the interface selected for each Serial Communication (SERCOM) module and the intended function for each SERCOM. A comprehensive allocation table detailing the distribution of interfaces and the function of each group of pins is provided in Appendix A, providing a thorough resource for understanding the complete pin allocation strategy.

This pin allocation process ensures that the microcontroller can effectively and efficiently manage its various functions and interfaces, meeting the specific needs of a wildlife monitoring system.

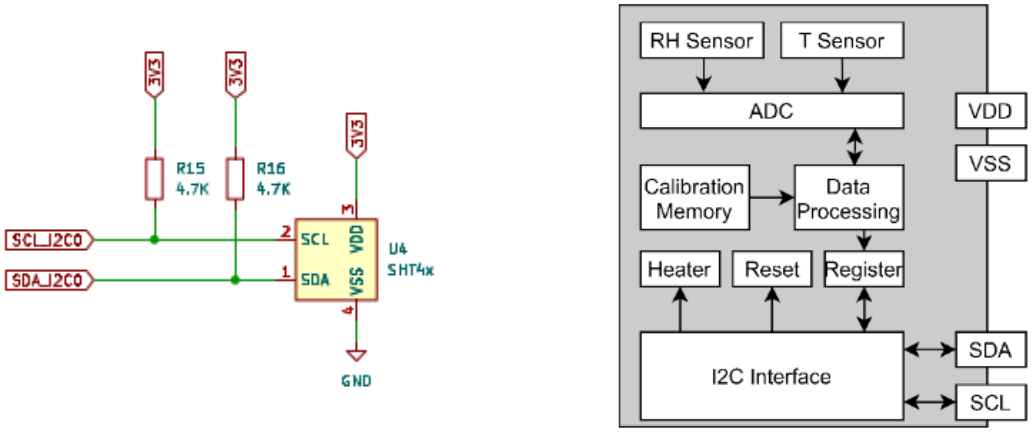
2.3.2 Internal sensors

In order to monitor the system's state in response to environmental disturbances that might affect its operation, internal sensors are incorporated. These sensors are responsible for tracking temperature, humidity, and movement, with the primary purpose of identifying critical environmental conditions within the device's enclosure. Examples of such conditions include water leaks, extreme weather events, or temperature fluctuations that could impact the performance and integrity of the embedded system. Furthermore, the ability to detect motion is essential for identifying instances of free falls, accidental disturbances, or potential tampering, which is a significant concern for devices deployed in the wild.

For these monitoring functions, high-precise sensor data are not required, as the sensors are primarily utilized to trigger alerts when specific thresholds are surpassed. The selection of sensors, therefore, prioritizes factors such as cost effectiveness and ease of integration into the electronic design to maintain simplicity without unnecessary complexity in the circuitry. Simplifying integration is facilitated by sensors with a digital I2C interface, which enables the control of multiple devices with only two communication signals.

Regarding temperature and humidity monitoring, the Sensirion SHT40I [27] has been chosen. This sensor offers both temperature and humidity sensing capabilities

in a single integrated circuit (IC). It's important to note that the SHT40I is specifically designed for challenging industrial applications, making it an excellent fit for wildlife monitoring. Notable features include a wide operating voltage range, small form factor, and robust housing. Its low-power consumption ($22\mu A$) ensures that it minimally impacts the device's overall energy consumption. Figure 2.3a shows the integration of the sensor into the electronic design, requiring only two signals for the I2C interface that are appropriately pulled up using 4.7Ω resistors.



(A) Schematics. (B) Functional block diagram.

FIGURE 2.3: Schematics of the temperature and humidity internal sensor.

The LIS3DHTR [28] 3-axis MEMS accelerometer was selected for movement detection. It offers an array of features that perfectly aligns with the requirements of wildlife monitoring. The "nano" family of accelerometers excels in ultra-low-power operational modes, an essential characteristic for a device intended for extended deployments in the field. One of the most noteworthy features is its ability to generate programmable interrupt signals based on inertial wake-up/free-fall events and device orientation. This capability is crucial for the system as it offloads the microcontroller from the task of continuous event monitoring, allowing it to efficiently operate by reacting only to interrupt signals transmitted by the sensor. This not only conserves energy but also streamlines the system's response to critical events in wildlife monitoring environments. As depicted in Figure 2.4, the integration of this sensor into the system is straightforward from an electronic design perspective. The implementation merely involves incorporating pull-up resistors for the I2C communication and connecting the interrupt pins of the sensor (*INT1* and *INT2*) to their respective microcontroller digital inputs designated as *FALL-INT* and *MOV-INT* for free-fall and movement detection, respectively.

2.3.3 Grove connectors

Five Grove standard [8] connectors have been integrated. Each Grove standard connector has four pins: two for power (VCC and GND) and two for signals. These connectors serve different purposes: two are designated for digital communications, two for analog inputs, and one is reserved for the I2C interface. Note that the microcontroller's signal multiplexing functionality allows for further configuration of

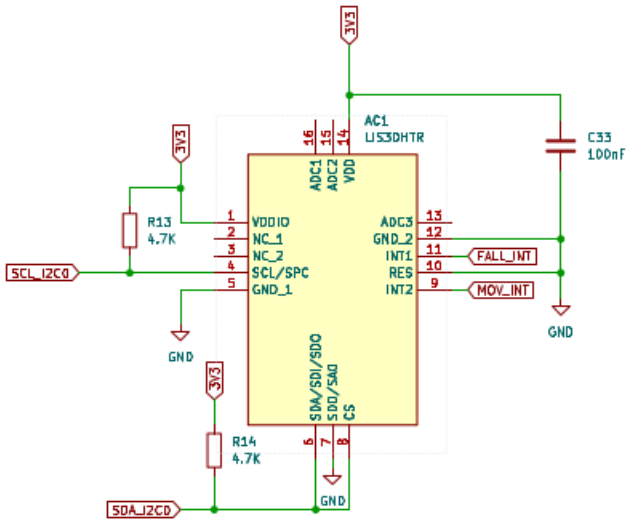


FIGURE 2.4: Schematic of the internal accelerometer.

these interfaces, making it possible to adapt them for UART or SPI if needed. Furthermore, all these connectors operate at the standard 3.3V voltage level, which is well-suited for low-power sensors. However, there is one exception, as one of the connectors is specifically reserved for 5V sensors or more power-hungry actuators that may be employed in the system for certain applications. These Grove connectors enhance the versatility of the embedded system, ensuring its seamless connection to a variety of sensors modules and devices available at market.

2.3.4 SPI camera connector

While the AI acceleration unit, which will be discussed in the subsequent subsection, primarily handles advanced image processing tasks, some projects may demand straightforward image acquisition without the need for intricate computations. In such cases, utilizing the MCU in isolation, without the AI accelerator module, can provide cost-effective and energy-efficient image solutions.

There are SPI cameras [29] compatible with microcontrollers like the SAMD51. These cameras employ an FPGA-based camera controller responsible for managing the intricate and high-speed timing of camera video signals. This controller directly interfaces with the image sensor, handling the entire image capture process. By offloading the processing tasks from the MCU to the FPGA, the MCU only needs to issue a capture command and retrieve images byte by byte as time permits. The captured images are stored in an off-chip frame buffer, ensuring they are preserved until explicitly flushed. This architecture addresses the challenge of lacking a dedicated camera interface. This architectural configuration is depicted in Figure 2.5.

The SPI camera modules are designed to utilize SPI for image data acquisition and I2C for configuring image sensor register settings. To accommodate these SPI camera modules and other devices potentially requiring a full SPI interface or employing non-standard digital interfaces with more than the two signal pins provided by the Grove standard, a specialized connector is included in the system.

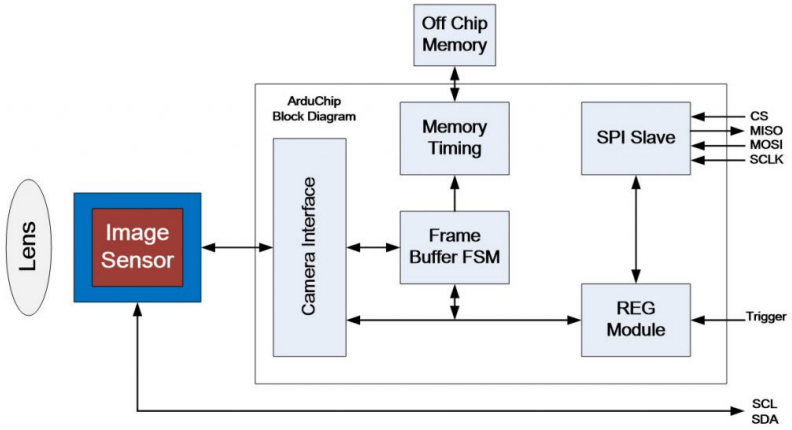


FIGURE 2.5: Functional block diagram from the SPI camera module of [29].

2.3.5 I2S microphone connector

Audio is a vital source of data in conservation technology, particularly for passive acoustic monitoring, which provides essential information about biodiversity, including birds and bats, especially ultrasounds. To accommodate these requirements, the SAMD51 microcontroller features an I2S (Inter-IC Sound) interface, a widely used protocol for transmitting high-quality digital audio data. I2S MEMS microphones have become prevalent in the market due to their ability to deliver high-quality, noise-free digital audio input. The I2S protocol simplifies the process of capturing audio data, minimizing the need for external circuitry compared to analog microphones, which require amplifiers and filtering circuits, in addition to the MCU's analog-to-digital converter (ADC).

To ensure flexibility in the selection from the variety of microphone models available in the market, the system includes an I2S interface. It is important to note that microphones themselves are not provided as part of the system; instead, the interface is made available to accommodate different models. This approach allows users to select the most suitable microphone for their specific needs and environmental conditions. Moreover, external connection of the microphone is preferred over in-board options, as it avoids issues related to audio signal attenuation when the device is enclosed in a waterproof housing, even with dedicated vent holes. The external microphone interface opens up possibilities for customization and adaptability, enhancing the versatility of the system.

2.4 AI processing unit

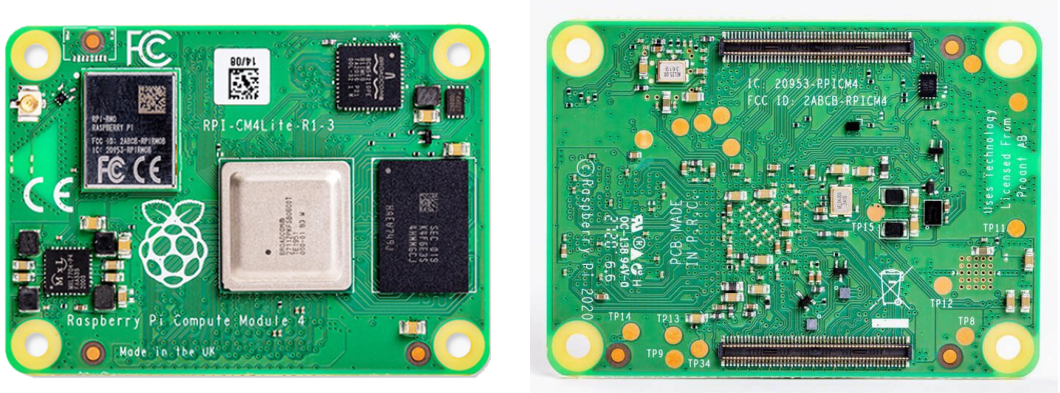
To fulfill the critical role of advanced AI processing capabilities, particularly for handling high-level audio and image data, the system incorporates the Compute Module 4 (CM4) [30]. This versatile System-on-Module (SoM) harnesses the substantial processing power of the Raspberry Pi 4, making it an ideal choice for this application. The Raspberry Pi 4 provides a suitable balance between high processing capabilities and a fully developed AI environment, including extensive software support and widespread popularity. The CM4's compact form factor, tailored for deeply

embedded applications, serves as a highly suitable solution for the system's requirements. While the Raspberry Pi 4 offers a lot of interfaces and features, including multiple HDMI ports, USB connections, and Ethernet capabilities, many of them are not needed for the system's specific needs. By leveraging the CM4 with the minimal number of components required from the Raspberry Pi 4 in a SoM format, the system benefits from the Raspberry Pi's computational power without the overhead of unnecessary interfaces, resulting in a streamlined and efficient solution tailored to the project's unique demands. This strategic selection of the CM4 endows the system with the processing capabilities necessary for advanced AI tasks, while also optimizing efficiency and conserving hardware resources.

Furthermore, the CM4's pinout and physical connector constitute a standard interface for various SoM designs offered by different manufacturers, largely due to the widespread adoption and versatility of the Raspberry Pi ecosystem. This standardization ensures that the system's carrier board, which features CM4 connections, can seamlessly interface with alternative SoMs that incorporate different CPU solutions. This adaptability and compatibility open up a wide range of possibilities, enabling the system to evolve in response to shifting requirements, explore diverse CPU architectures, and address any potential supply chain constraints that may arise in the future. Some examples of this compatible SoM available at this moment are:

- SOQUARTZ Compute Module. Featuring a QUAD 64-bit ARM Cortex-A55 CPU and 0.8 TOPS Neural Network Acceleration Engine.
- Radxa CM3. Based on the Rockchip RK3566 SoC, a 64bit Quad Cortex A55.
- BPI-CM4. with Amlogic A311D Quad core ARM Cortex-A73 and dual core ARM Cortex-A53 CPU, ARM G52 MP4(6EE) GPU, NPU for AI at 5.0 TOPS.
- ARVSOM. Based on RISC-V StarFive 71x0 SoC.

The CM4 (Figure 2.6) comprises a quad-core ARM Cortex-A72 processor, LPDDR4-3200 SDRAM options ranging from 2GB to 8GB, and a VideoCore VI graphics core. This combination provides substantial computational power and memory capacity to handle demanding AI tasks effectively.



(A) Top view. (B) Bottom view.

FIGURE 2.6: Raspberry Pi Compute Module 4 [30].

The system interfaces with the CM4 through two 100-pin connectors, providing an extensive range of connections for seamless integration. The first connector (pins 1 to 100) encompasses a variety of interfaces, including Ethernet for networking capabilities, GPIO (General-Purpose Input/Output) pins for flexible digital connectivity, and SD card signals for storage and data transfer. This connector includes interfaces such as HDMI for high-definition video output, CSI (Camera Serial Interface) for connecting camera modules, DSI (Display Serial Interface) for interfacing with displays, and PCIe (Peripheral Component Interconnect Express) for high-speed expansion options.

Critical for the operation of the AI accelerator module is establishing effective communication between the MCU system and the CPU within this module. This communication is facilitated through a Remote Procedure Call (RPC) mechanism, enabling both cores to invoke functions on the other processor without interruptions or further complications. Details on how this integration is achieved at software and firmware level will be provided in the following section. Concerning hardware, a serial UART communication channel has been included to serve as the interface between the MCU and the CM4. This UART channel ensures reliable and efficient data exchange, making it possible for the two processing units to collaborate effectively and execute their respective tasks in a coordinated manner.

Another interface employed in the system is the direct connection of the CM4 to an CSI (Camera Serial Interface) port. The CSI port is specifically designed for camera modules and is well-suited for capturing high-speed image signals. This connection enables the system to be fully compatible with high-speed image acquisition, a fundamental requirement for tasks such as real-time image processing.

The CM4 is also equipped with a substantial capacity of RAM and eMMC flash memory, making it well-suited for hosting the system's operating system file system and temporary data spaces required during processing. This built-in memory provides ample storage and processing capabilities for a wide range of applications. However, to enhance data sharing and minimize the need for continuous data transfers between the CM4 and the MCU system, a dedicated solution has been implemented to share SD storage; this solution is described in a subsequent subsection. An SDIO interface available on the CM4 module is employed to access data stored in the shared storage system, offering a practical and optimized way to access, process, and manage data without the need for constant data transmission.

2.5 Network communication unit

To ensure adaptability and versatility in terms of network communication, the system leverages the MikroBus standard [7], a widely used interface employed by an extensive set of commercially available communication transceiver modules. The MikroBus standard defines a unified socket, shown in Figure 2.7, that consists of a pair of 1×8 female headers with a proprietary pin configuration. This standardized pinout, always presented in the same order, comprises three main groups of communication pins (SPI, UART, and I2C), along with six additional pins: PWM, Interrupt, Analog input, Reset, and Chip select. Furthermore, the MikroBus socket includes two power groups, providing both +3.3V and 5V, to accommodate a wide range of

communication modules.

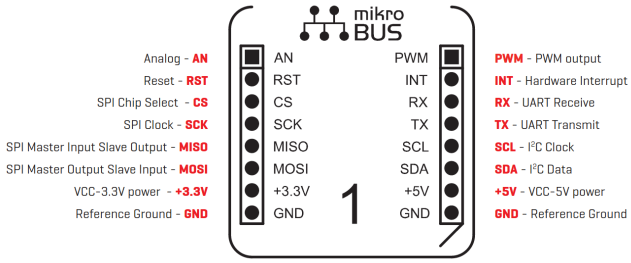


FIGURE 2.7: MikroBus pinout specification [7]

The utilization of the MikroBus standard simplifies the process of incorporating commercial communication transceiver modules, making it easy for the developer to select and integrate the most suitable modules to meet particular requirements of wildlife monitoring projects. These modules may encompass Wi-Fi, GSM, LoRa, and Zigbee, among others.

2.6 Power management unit

The power management unit of the proposed wildlife monitoring system is a critical component responsible for efficiently managing and distributing power to various elements within the system. The power requirements for the system were carefully analyzed in previous sections. Figure 2.8 provides a detailed power diagram showcasing the diverse power input sources, including solar, DC, USB, and LiPo batteries, along with all the components required to control the power.

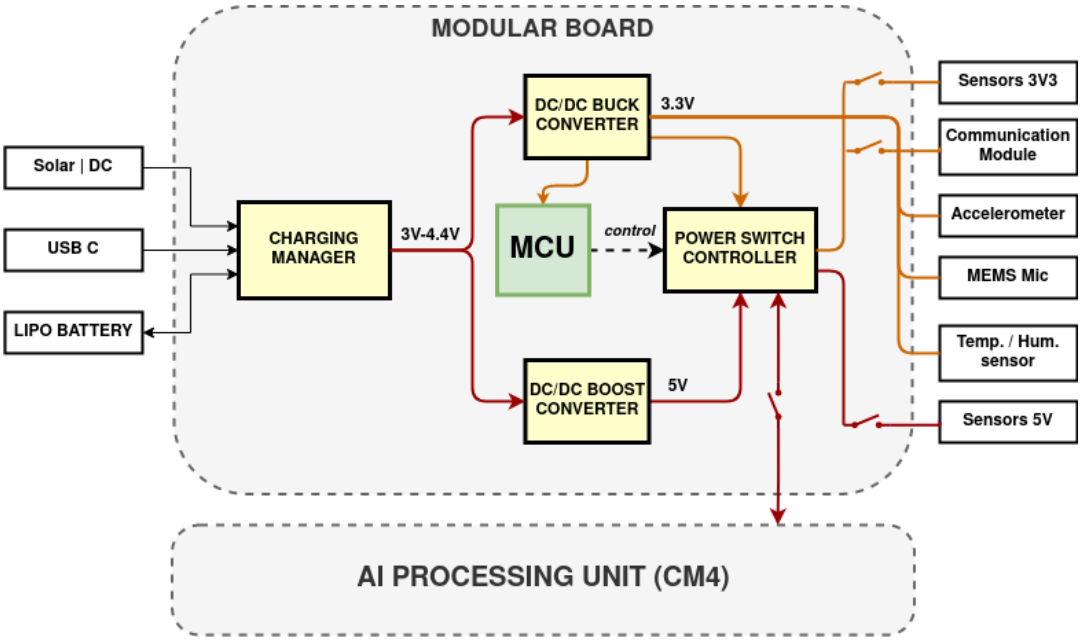


FIGURE 2.8: Diagram of the power management unit of the proposed wildlife monitoring system.

To ensure the reliable and efficient operation of the system, the power management unit comprises several key elements, each designed to fulfill specific functions. These essential components include a charging manager, LDO 3.3V regulator, DC/DC 5V boost converter, and power switch controller.

2.6.1 Charging manager

Given that the system relies on LiPo batteries for power, the choice of an appropriate charging controller ensures optimal battery performance and long lifetime. The two primary categories of charging controllers, namely linear and switch-mode chargers, were considered in the selection process. Furthermore, given the utilization of solar panels for recharging, Maximum Power Point Tracking (MPPT) controllers can be integrated to extract maximum energy from these sources.

MPPT controllers are instrumental in optimizing solar panel-based charging systems. They operate by continuously tracking the voltage and current characteristics of solar panels to maximize power output. This approach necessitates an efficient DC/DC converter to deliver optimal power conversion. However, adopting a genuine MPPT solution in this system introduces a series of challenges and trade-offs. Integrating a DC/DC converter into a LiPo charger circuit increases both the cost and the complexity of the circuit. For smaller solar panels and low-current circuits drawing under 1A, the potential efficiency gains from MPPT may not be substantial. In such cases, the increased cost associated with the implementation of a DC/DC converter often outweighs the efficiency improvements that MPPT offers. At low voltages and currents, where the panel voltage remains marginally higher than the battery charging voltage, DC/DC converters do not consistently outperform linear converters. The wildlife monitoring system has been designed to work with small 6V solar panels and 1-cell LiPo battery packs with nominal voltage of 3.7V. According to the specific operational parameters commented, the choice of a linear charger for the LiPo batteries offers a cost-efficient and proficient charging solution.

The chosen charger is the BQ24074 [31], which is a 1-cell 1.5A linear battery charger, providing an ample charging current suitable for the battery capacities typically used in wildlife monitoring projects. One of its notable features is its wide input operating voltage range, spanning from 4.5V to 10V. This broad range ensures compatibility with various power sources, such as small 6V solar panels, USB adapters operating at 5V, and DC wall chargers delivering up to 9V.

The BQ24074 incorporates dynamic power path management (DPPM), which is particularly valuable for the system's operation. This function intelligently allocates the source current between the system and battery charging processes, automatically reducing the charging current if the system load increases. When charging from a DC port, the input dynamic power management (VIN-DPM) circuit reduces the input current when the input voltage falls below a specific threshold, effectively preventing the DC port from crashing. This power-path architecture is versatile, allowing the battery to supplement the system's current requirements when the input source cannot provide the peak system currents. These capabilities make the BQ24074 an excellent choice for the system, effectively addressing the various operating conditions and ensuring efficient solar charging, even when effectively functioning close to an MPPT solar charger.

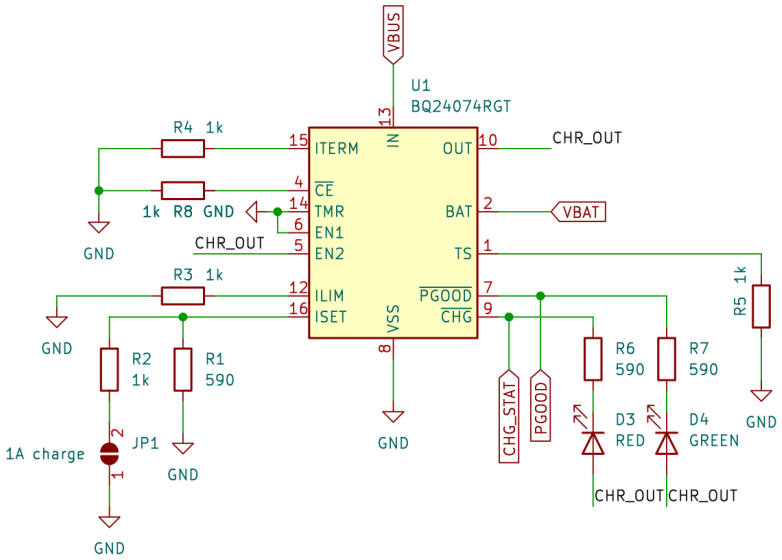


FIGURE 2.9: Schematic of the charging manager circuit.

The schematic shown in Figure 2.9 contains the charging manager circuit. The EN1 and EN2 pins have been configured with 0 and 1, respectively, to set the maximum input current through the use of an external resistor connecting ILIM to VSS. A 1k resistor was selected following the provided formula $R_{ILIM} = K_{ILIM}/I_{IMAX}$ to establish an input current limit of up to 1A. Likewise, the maximum charging current has been set to 1.5A using a 590-ohm resistor, using the formula $R_{ISET} = K_{ISET}/I_{CHG}$ to guide the selection. To provide clear visual indications of system status, LEDs have been positioned at the PGOOD and CHG pins. These LEDs serve as visual indicators, with one signaling when a valid input source is connected and the other conveying the charging status.

2.6.2 3.3V power supply

The 3.3V power supply, as illustrated in the power diagram shown in Figure 2.8, provide power to the MCU, internal and external sensors, and communication module within the system, which constitute the low-power system components. The system can operate in various modes, each with its own power consumption profile. In sleep mode, power consumption should ideally be in the range of microamperes (μA), while power consumption of the sensor acquisition mode depends on the specific sensors used; generally, for low-power sensors, it should not exceed 100mA. The highest power consumption is encountered in network communication mode, where, depending on the type of communication, peak currents of up to 350-400mA can be reached.

The power supply source for this unit is derived from the output of the charging manager circuit, which can vary from a regulated 4.4V when powered from an external USB or DC source to levels below that when solely reliant on the 1-cell LiPo battery. The voltage of this battery ranges from 4.2V at full capacity to a cutoff level of 2.5V. However, this voltage range is non-linear, with the battery typically operating at a nominal voltage of 3.7V for 80% of the discharge curve. Manufacturers

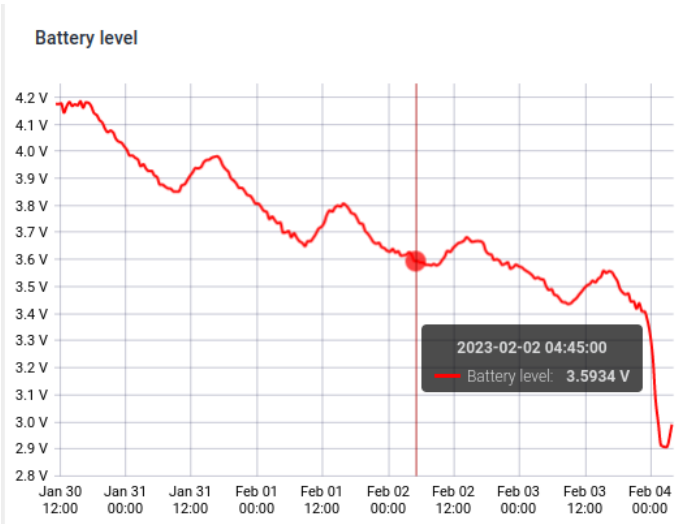
recommend not discharging batteries below 3V to extend charge/discharge lifecycles and prevent permanent damage.

Considering these design parameters, the power supply should offer a maximum output current of over 500mA, support input voltage in the range of 3V to 4.5V, and maintain a fixed output voltage of 3.3V. The quiescent current, which represents the current consumed by the internal control circuitry of the power supply, is another important parameter, especially since the system consumes only microamperes in sleep mode. High quiescent current could significantly impact the efficiency of this mode of operation.

In terms of efficiency, the choice between a low-dropout linear regulator (LDO) and a switch-mode DC/DC converter does not significantly impact this application due to the very low typical power consumption of the system and the minimal voltage drop between the input and output of the power supply. However, the key consideration is the minimum dropout voltage, a parameter often associated with LDOs in this application, which tends to be around 300mA. This implies that the power supply would require a minimum input voltage of 3.6V, resulting in a significant loss of the battery's total discharge curve.

To determine the appropriate minimum input voltage for the power supply, an analysis of experimental data obtained during a battery-powered device test was conducted. During this test, the device was deployed with high load charging and transmitted its battery level every minute. The system was also powered by a small solar panel. As depicted in the plot illustrating the collected data (Figure 2.10b), the solar panel recharged the battery during the day but could not fully recover it to 100%. Consequently, after a period of time, the battery reached its cutoff voltage. The total discharge period of the battery was observed to be 110 hours and 30 minutes.

(A) Device deployed.



(B) Discharge plot of the battery during the conducted test.

FIGURE 2.10: Battery-powered device solar charging test.

If the cutoff voltage were set at 3.6V, as would be the case when using an LDO, the total discharge period would be reduced to 65 hours, resulting in a significant capacity loss of 41.18%. On the other hand, by setting the cutoff voltage at 3.3V,

the discharge period extended to 109 hours, incurring only a minor capacity loss of 1.36%.

Based on these results, the use of an LDO was discarded due to the inefficiency it would introduce into the system. Consequently, considering a switch-mode DC/DC converter, it can be concluded that a buck converter can be the most suitable choice for maintaining the ideal cutoff voltage at 3.3V. This decision ensures that the system can make the most of the battery's capacity without compromising battery degradation.

The LM3671 [32] step-down converter is an ideal choice for the power supply in the wildlife monitoring system. It boasts a maximum output current of 600mA and operates within the desired input and output voltage ranges. What makes it particularly suitable for this application is its unique mode-switching capability, optimizing power consumption during system sleep mode, a crucial feature for energy-efficient operation.

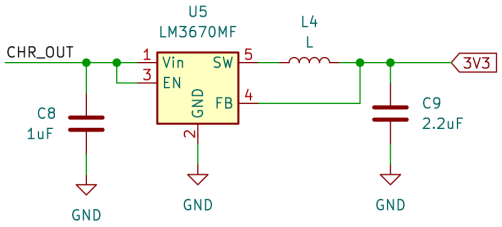


FIGURE 2.11: Schematic for the 3.3V buck converter circuit based on LM3671.

The LM3671 offers automatic intelligent switching between PWM low-noise and PFM low-current modes, providing enhanced system control. In PWM mode, the device operates at a fixed frequency of 2MHz, ensuring stable power delivery. During periods of light load and standby operation, the hysteretic PFM mode comes into play, reducing the quiescent current to just 16µA. This feature significantly extends the battery life during low-power operation. Moreover, the internal synchronous rectification enhances efficiency during PWM mode operation, making the LM3671 an excellent choice for wildlife monitoring, where power efficiency is critical, especially in sleep mode. The schematic in Figure 2.11 illustrates that the design for the 3.3V power supply using this controller is remarkably straightforward.

2.6.3 5V power supply

The 5V power supply provides power to the AI processing unit, based on the Compute Module 4 (CM4), and external interfaces for sensors or additional modules operating at 5V. The power consumption of the CM4 varies depending on the processing tasks it performs but can reach approximately 1.4A. Considering possible external modules connected as well, the power supply must be capable of delivering a maximum output current of at least 2A. Even though these elements are intended for use during short periods, it is imperative that the power supply operates with high efficiency to minimize additional battery capacity losses. Therefore, maximum output current and efficiency are the primary considerations in the design of this

power supply.

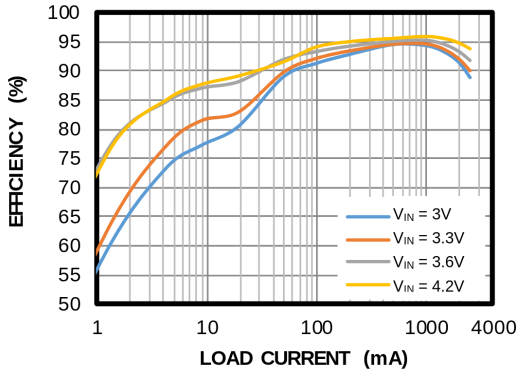


FIGURE 2.12: Typical performance of MP3437 in terms of efficiency vs. load current extracted from the datasheet.

The MP3437 [33] was selected as the foundation for designing the 5V boost converter. This boost converter features an input voltage range starting as low as 2.7V and supports up to 20W of load power from a 1-cell battery, making it highly suitable for this application. Referring to the efficiency vs. load current performance plot shown in Figure 2.12, which was extracted from the datasheet, we can observe that for the anticipated range of input voltages in this circuit, the efficiency remains around 95% for loads of 1A and above, with over 90% efficiency for loads up to 2A. Note that these results are for an output voltage of 8V, meaning that the efficiency will be slightly better for a 5V output. These efficiency characteristics align perfectly with the system requirements, ensuring optimal performance and minimal energy waste.

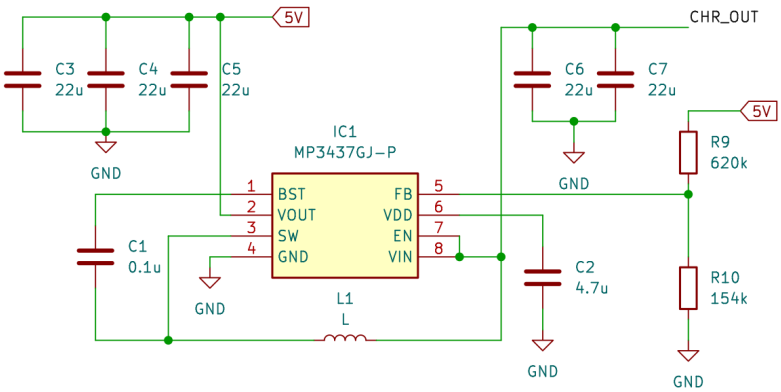


FIGURE 2.13: Schematic of the 5V DC/DC boost converter based on MP3437.

When delving into the design details of this power supply (Figure 2.13), it becomes evident that special attention is warranted, especially due to its capacity to handle moderately high output currents up to 2A. The selection of input and output capacitors and inductors is of paramount importance in ensuring the system's proper functionality. The boost converter, due to its discontinuous output current, necessitates the presence of an output capacitor to supply a consistent DC current to

the load. It is imperative to maintain the output voltage within a tight range, ideally within $\pm$0.1V despite allowed variations of $\pm$0.25V in the input voltage of the CM4 module [30].

To assess the output voltage ripple and ensure it falls within acceptable limits, the following formula is provided in the datasheet, taking various factors into account:

$$\Delta V_{OUT} = \frac{V_{OUT}}{f_{SW} \times R_L \times C2} \times \left(1 - \frac{V_{IN}}{V_{OUT}}\right) \tag{2.1}$$

Considering the use of three 22 µF parallel capacitors, as indicated in the schematics, a load of 2A, and an input voltage from a typical 1-cell LiPo battery (3.7V), the calculated output voltage ripple remains within acceptable bounds at 0.013V. The datasheet recommends the use of ceramic capacitors with X5R or X7R dielectrics due to their low equivalent series resistance (ESR) and minimal temperature coefficients.

Turning to the inductor, its primary role is to transfer energy between the input source and the output capacitors. Selecting an appropriate inductor value impacts the ripple current and peak inductor current, thus influencing the stress on the power MOSFET. The datasheet provides the following equation for inductor value calculation:

$$L = \frac{V_{IN} \times (V_{OUT} - V_{IN})}{f_{SW} \times V_{OUT} \times \Delta I_L} \tag{2.2}$$

Assuming a 50% ripple current of the maximum 2A output, in line with the voltage considerations mentioned earlier, a 1.6 µH inductor value is derived. For added prudence, a 2.2 µH inductor can be considered. It is essential to choose an inductor with low series resistance (DCR) to minimize resistive power losses.

2.6.4 Power switch controller

The power switch controller is responsible for regulating the activation and deactivation of loads such as external sensors, the AI processing unit, and the communication module. Every integrated circuit has a quiescent current consumption, even when powered off, typically measured in microamperes. While this consumption may initially appear insignificant, in a system with sleep mode functions, where it's crucial for the system to operate over extended periods while waiting for specific events, these residual currents can become critical in terms of preserving autonomy. To address this concern, a switch-controlled system has been implemented to physically disconnect these loads from the power supply when they are not in use.

Load switches represent a simple and cost-effective solution for toggling power rails on and off. The design incorporates two TPS22976 [34] dual-channel load switches, each equipped with controlled turn-on capabilities. One switch manages the 3.3V loads, including sensors and the communication module, while the other handles the 5V loads, such as the CM4 and 5V sensor modules. Each switch can be independently controlled through dedicated on and off inputs (ON1 and ON2), which can interface directly with control signals from the MCU. When these switches power off a load, the internal consumption of each switch is only 0.002 µA, almost negligible in practical terms. Additionally, when the switches are in the powered-on state, the typical consumption of both channels reduces to 37µA, further minimizing the energy loss. This efficient load switching design helps extend

the system's autonomy during sleep mode and inactive periods. Figure 2.14 displays the circuit for one of the two load switches responsible for controlling the 3.3V loads. The other load switch employs a similar design but with a 5V input and is responsible for managing the load of the AI processing unit and the external 5V Grove connector.

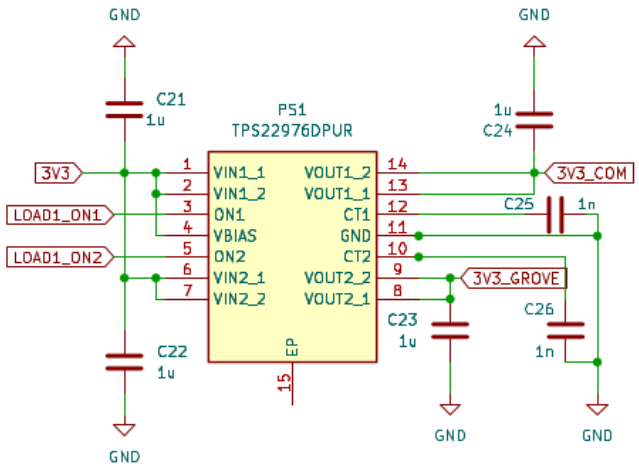


FIGURE 2.14: Schematic of the two-channel load switch to control 3.3V loads in the circuit. It is based on TPS22976.

2.7 Shared SD storage

The shared SD storage component is essential for efficient data handling, particularly when managing moderate amounts of data, such as audio or image inputs. In the context of the system's architecture, transferring all this collected information from the acquisition and low-power processing unit based on the MCU to the AI processing unit via the UART serial interface can be highly inefficient. To address this issue, a shared storage unit has been integrated. As mentioned in the requirements analysis, SD cards are the preferred storage choice for conservation technology devices. Consequently, a multiplex/demultiplex circuit has been integrated into the system to create a shared storage unit that can store data collected and processed by both parts of the system. Achieving this functionality is not only a matter of hardware but also a firmware-level implementation, involving a dedicated *Remote Procedure Call* mechanism. This mechanism serves as the coordination link for writing and reading operations between both processing cores, ensuring efficient data sharing and synchronization between the MCU and AI processing unit.

The TS3A27518E [35] is a component that serves as a bidirectional, 6-channel, 1:2 multiplexer-demultiplexer. Its primary function is to enable the expansion of any SD, SDIO, and multimedia card host controllers to accommodate multiple cards or peripherals. This functionality is particularly valuable in the context of the system because the SDIO interface relies on 6-bits, which include CMD, CLK, and Data[0:3] signals. Given this characteristic, the TS3A27518E is an ideal choice for this application, where efficient management and switching of these signals are essential for proper system operation. Figure 2.15 illustrates the circuit design of the shared SD storage system employing this component.

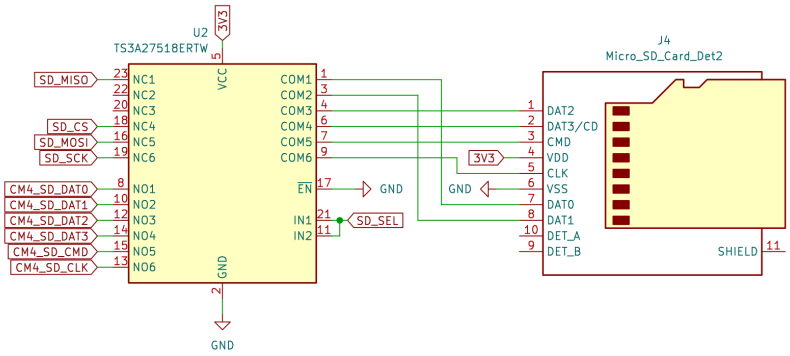


FIGURE 2.15: Schematic of the SD shared storage circuit.

The schematics detailing the hardware designs presented and discussed in the preceding sections are available in Appendix B. It's noteworthy that the schematic design process was accomplished using the open-source software KiCad. Hierarchical sheets were thoughtfully employed to structure the schematics into different modules, enhancing accessibility and simplifying the process of making any necessary adjustments or modifications.

Chapter 3

Software Development

The proposed wildlife monitoring system exhibits a heterogeneous embedded system architecture. This architecture encompasses multiple cores that differ in terms of both architecture and processing capabilities. It combines a microprocessor core with a microcontroller-class core to provide a balanced solution. Additionally, the system features a mix of peripherals, some exclusively interfacing with the MCU core, while others are shared between both cores through the shared SD card storage system. This heterogeneous design offers several advantages in terms of configurability, processing optimization, and power efficiency. However, it introduces complexity to the software development process. Consequently, it is essential to design a robust software architecture to manage this intricacy effectively.

This section begins with an overview of the designed software architecture for the system. Subsequently, it delves into the various software components that comprise this architecture, encompassing firmware, the Linux operating system (OS), and embedded software, among others. Note that developing a software solution for a system of this nature necessitates a multidisciplinary approach, combining expertise in firmware design, Linux kernel knowledge, high-level programming languages like Python, AI workflow development using frameworks such as TensorFlow, and UI design. While this Master's thesis focuses primarily on the low-level firmware aspect, the foundation is laid for other elements of the software system in terms of architectural design, without delving into the specifics of software development.

3.1 Software architecture

The software architecture has been designed to accommodate various levels of interaction and system configuration. This design is visually represented in the architecture block diagram shown in Figure 3.1, which illustrates the interconnection of different software layers: firmware, embedded software, and user interface (UI) application.

At the primary level of software development, the firmware resides within the MCU of the system. This firmware plays a pivotal role in managing low-level hardware interfaces. Its primary functions encompass real-time data acquisition from connected sensors and modules, preprocessing of collected data, data structuring and storage on the SD card, configuration of low-power modes using internal timers, RTC (Real-Time Clock), external interrupts, communication with transceivers for data transmission to the cloud, and setting of communication with the AI processing unit through the remote procedure call (RPC) mechanism. To ensure flexibility

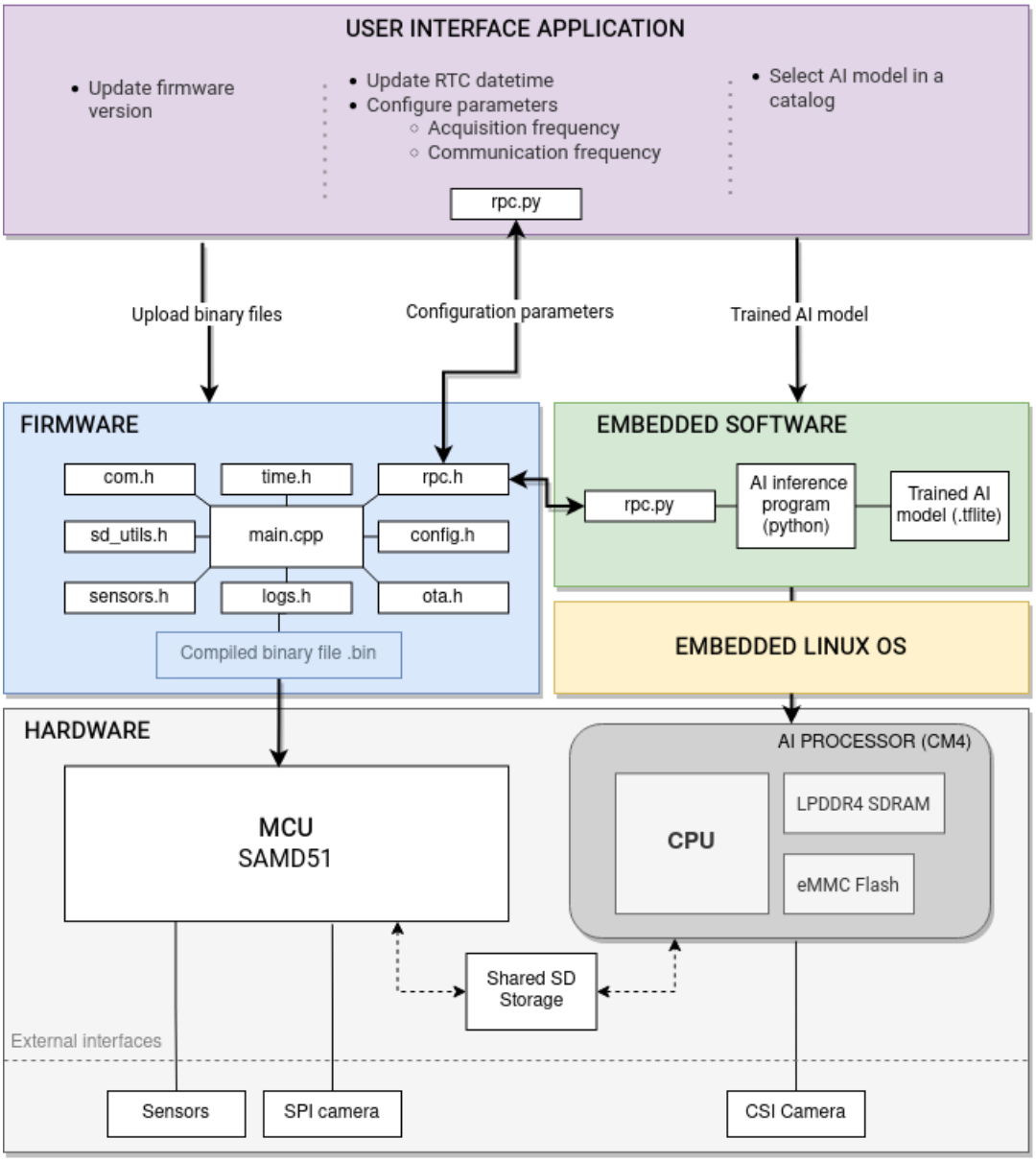


FIGURE 3.1: Block diagram of the software architecture of the system.

and ease of updates and configuration, the firmware binaries can be uploaded to the microcontroller's program memory through various methods, including USB, SD card, or network-based "Over the Air Programming" (OTA) mechanisms.

At the core of the AI processing unit lies the embedded Linux operating system, which comprises a bootloader, the Linux kernel, and the root filesystem. This Linux-based system is responsible for resource management, task scheduling, hardware access, and various operations that shield the user from low-level aspects of the hardware. The Linux kernel plays a crucial role in managing hardware resources, such as the CPU, memory, and I/O operations. It provides a range of Application Programming Interfaces (APIs) that abstract these resources, simplifying the deployment of user applications and libraries. In the context of the Raspberry Pi CM 4, the embedded Linux OS utilized is a customized version of Raspbian, which is derived from Debian and tailored to suit the specific requirements of this board.

Within the root filesystem of the embedded Linux OS, which is stored in the internal eMMC Flash memory of the CM4, various software programs run. This includes system programs, utilities, and configurations, among other components. A primary program within this block is responsible for retrieving the data, collected by the firmware and stored in the SD card, and executing inference tasks using a pre-trained AI model. The preferred programming language for developing software at this level, particularly for AI applications, is Python. An essential aspect within this block of embedded software is the interface between the firmware and the AI processing unit through Remote Procedure Call (RPC). In the architecture diagram, this interface is represented as a Python library called "rpc.py." This library should encompass all the necessary functions for seamless collaboration with the firmware. It serves as a critical bridge in the development process, ensuring that high-level AI developers working with languages like Python do not need to possess specific knowledge about the underlying hardware or the intricacies of firmware operations. Both the "rpc.py" library and the corresponding "rpc.h" library at the firmware level should provide an abstraction layer that simplifies communication and data exchange between the low-power MCU and the AI processing unit. This abstraction enhances development efficiency and allows for transparent information flow between these system components.

The aforementioned software elements set the basis for the development of software tailored to the proposed wildlife monitoring system. These tools and components provide the necessary infrastructure to implement various functionalities within the system. However, given that this is a versatile device designed for deployment across different field applications, the inclusion of a user interface (UI) for configuration purposes is critical. This UI can serve a dual purpose, benefiting not only end-users but also system engineers, because it facilitates the configuration of specific operational parameters, allowing the system to be easily adapted for different applications and deployment scenarios.

3.2 Firmware design

The firmware is developed in C++, a common choice for modern microcontroller architectures. Although the Arduino framework is employed, it is not the primary development environment; rather, it is utilized to leverage the bootloader and core libraries. This approach allows the system to take advantage of the extensive compatibility offered by the Arduino framework with a wide array of available libraries for sensors, network communication modules, and other components. However, it does not restrict the system to the more basic features often associated with the Arduino IDE. Instead, the PlatformIO IDE [36] serves as the primary development platform. PlatformIO provides advanced features such as debugging, unit testing, and static code analysis, enhancing the firmware development process. Internally, PlatformIO utilizes the GNU Arm Embedded Toolchain as the code compiler for the SAMD51 microcontroller and other tools like the Basic Open Source SAM-BA Application (BOSSA) for uploading binary files to the MCU.

The firmware code comprises a central `main.cpp` file that defines the fundamental structure and operation of the device. In addition to this core file, the code includes multiple specific modules, each responsible for managing different interfaces

and functions within the system. This primary file manages a finite state machine that transitions through various operation modes. These modes encompass data acquisition, processing, network communication, and sleep mode. The flow between these modes is determined by a set of flags and parameters, ensuring the device functions efficiently. Figure 3.2 illustrates this state machine.

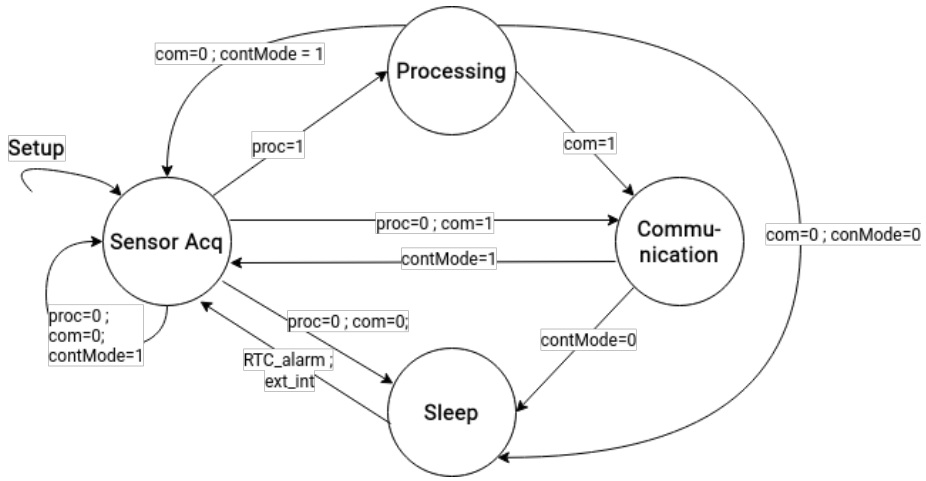


FIGURE 3.2: State machine performed in the main firmware process.

Setup The code in this section is executed only once at the start of device operation. It initializes all the microcontroller interfaces required for the proper functioning of connected hardware components. These interfaces include those for the SD card, sensor interfaces, internal RTC, timers, and network communication interfaces. Configuration is based on predefined parameters specific to the device's operation.

Sensor Acquisition During this mode, each sensor interface or data input source connected to the system is individually initialized and instructed to acquire raw data through their respective interfaces. The MCU then decodes and pre-processes these data before storing them in the micro SD card in a structured format.

Processing In the processing mode, raw sensor data are transformed into valuable information. The processing can be executed solely by the MCU, leveraging its low-power capabilities, or in collaboration with the AI processing unit. In some cases, initial filtering and preprocessing are carried out by the MCU, followed by more sophisticated processing by the dedicated AI unit. Further details on how this coordination between both units occurs will be explained later.

Communication In the communication mode, the information available for reporting is transmitted via the selected network communication module to designated cloud servers responsible for data storage and visualization. It includes configuration routines of the modules and the use of network communication protocols such as HTTP or MQTT.

The firmware's operation varies depending on the state of different flags, allowing for a wide range of device behaviors to suit various applications. The key flags and their corresponding functions are:

- *proc* (Processing): This flag determines whether data processing is performed. When set, the firmware will engage in processing activities.
- *com* (Communication): The communication flag controls the execution of data communication. If enabled, the firmware will initiate communication tasks.
- contMode (Continuous Mode): When this flag is set, the device operates continuously without entering the sleep mode. There is no wait time between the different operational modes.

These flags provide flexibility and configurability to adapt the device's behavior based on specific application requirements. For instance, a conventional data logger may loop between acquisition and sleep states, while a connected data logger includes the communication state between them. In contrast, a device with full capabilities will complete the entire loop of acquisition, processing, communication, and sleep modes as determined by the flag settings.

The settings of these flags are controlled by independent RTC alarms, each of which is configured with the desired time frequency for acquisition, processing, and communication. When configured, these RTC alarms trigger an interruption in the MCU, causing it to exit the sleep mode and set the corresponding flag to 1. This approach allows for complete configurability in terms of timing within the different operational states. For example, a device could be collecting data every 10 minutes, processing data on an hourly basis, and transmitting results twice a day.

The RTC alarms can be changed at runtime, providing the system with dynamic adaptability. This capability allows a self-powered device to monitor its battery level and switch between different modes of operation based on the available power capacity. When the battery level is low, the system can reduce the frequency of processing and communication to lower the average power consumption, thus extending its runtime. Conversely, when the battery level is high, the device can increase the frequency of processing and communication to take advantage of the available power resources. This dynamic adjustment of operational modes based on real-time battery status is a smart and efficient approach to power management, particularly for self-powered devices equipped with solar panels. It optimizes power utilization while ensuring continuous functionality under varying conditions.

In certain applications, data acquisition may be initiated by an external event, such as the detection of animal movement. In such cases, an additional interrupt can be configured to trigger the system's exit from sleep mode and initiate an operation. Unlike the RTC-based interruptions, which are time-driven, these external sensor-triggered interruptions are event-driven and are generated in response to specific conditions detected by external sensors. This allows the system to respond to real-time events or stimuli, making it suitable for a wide range of monitoring and data collection scenarios.

The software code is organized into different modules, each responsible for a specific aspect of the device's operation. These modules provide functions to perform various tasks within their respective areas. Here is an overview of the code modules and their functionalities:

***sensors.cpp*:** This module handles the initialization of sensors connected to the system and collects raw data from these sensors. It serves as the interface between the microcontroller and the various sensors, facilitating data acquisition.

***time.cpp*:** Functions related to real-time clock (RTC) configuration and alarm setup are found in this module. It enables the device to manage time and schedule operations based on the configured alarm triggers.

***sd-utils.cpp*:** The SD card utility module offers functions to simplify the initialization of the SD card and to write and read information in a structured manner.

***communication.cpp*:** This module deals with the configuration of transceiver modules and provides functions for establishing connections with cloud servers to transmit data collected by the device. It manages the communication aspect of the system.

***rpc.cpp*:** This module is used to establish communication between the MCU and the AI processing module through Remote Procedure Call (RPC).

***ota.cpp*:** Over-the-air (OTA) updates are handled in this module. It allows for firmware updates without the need for physical access to the device, ensuring that the device can be kept up to date.

***logs.cpp*:** The logging module is responsible for recording log messages related to the device's operation. These logs can be valuable for debugging and monitoring the device's performance during deployments.

***config.h*:** This header file contains configuration parameters, such as those that determine the frequencies of different operational states.

By structuring the code into modules, the software architecture becomes modular and maintainable, allowing for efficient development, testing, and debugging of the system. Each module focuses on a specific aspect of the device's functionality, making it easier to manage and extend the software as needed. For reference, the code for most of the modules described herein is available in Appendix C, which provides a comprehensive overview of the codebase.

3.3 Remote Procedure Call (RPC) Service

Within the embedded software layer, the crucial component responsible for enabling seamless interactions between the firmware and the AI processing unit is the Remote Procedure Call Service.

This service is intended to be initiated automatically by the Linux OS at the system's startup and consistently run in the background, regardless of other processes or configurations. It acts as a mediator for communication between the MCU unit and the AI processing unit, with the MCU typically functioning as the master and the AI processing unit as the slave. In this typical operation, RPC commands are dispatched from the MCU to the AI processing unit, which then responds accordingly.

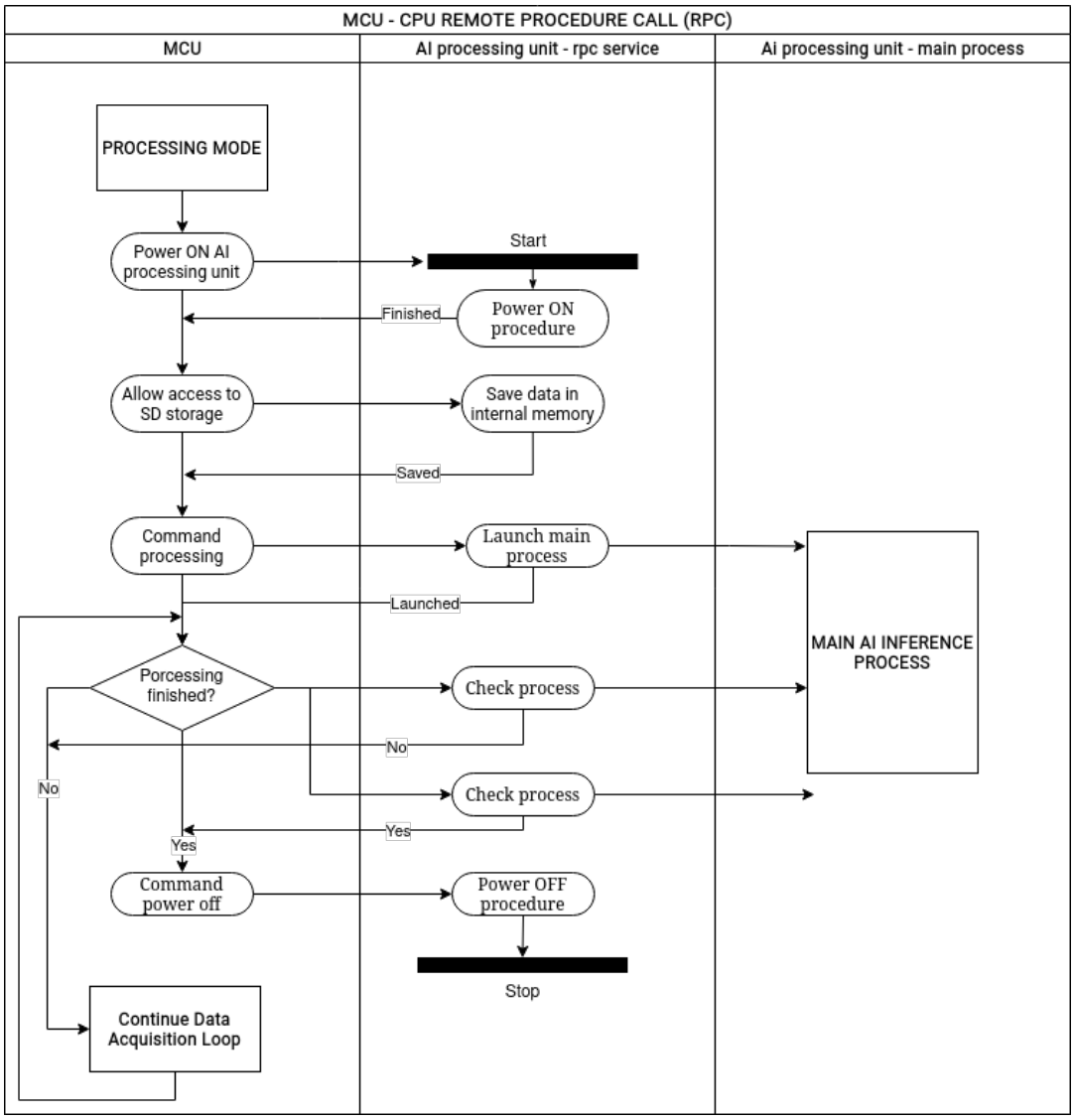


FIGURE 3.3: Block diagram representing the RPC interactions between the MCU and AI processing unit.

Figure 3.3 provides a breakdown of the interactions between these units through the RPC service, specifying the necessary commands and responses when the device undergoes an AI inference process in the AI processing unit. These interactions include:

- Power On Command: Initiates the AI processing unit.
- Allow SD Storage Access Command: Authorizes access to the shared SD storage.
- Processing Command: Triggers the AI inference process.
- Check Processing Status Command: Monitors the status of the inference process.
- Power Off Command: Shuts down the AI processing unit.

Note that when the AI processing unit is engaged in the inference process, the MCU unit remains unblocked and can continue its normal data acquisition loop in

parallel. This is achieved by storing data from the shared SD card in the internal memory of the AI processing unit. The only time the MCU unit is momentarily blocked is during the transfer of substantial data to the internal memory of the AI unit. To mitigate this, data can be divided into manageable blocks, ensuring that the MCU unit is not blocked for an extended period.

Chapter 4

Critical Analysis and Future Directions

4.1 Critical Analysis

The development process of a complex embedded system, such as the proposed wildlife monitoring device, is inherently multifaceted and spans numerous domains. While this Master's thesis mainly lays the groundwork for the architecture, hardware design, and basic firmware, it is important to acknowledge that the remaining steps required to realize a fully functional device of this nature will demand significant effort and time. These steps encompass a spectrum of distinct areas of development needed to deploy a device in the field, including hardware engineering, firmware development, software design, and product development. The work presented in this Master's thesis can be viewed as the initial foundational steps in this comprehensive development process.

Taking the aforementioned challenges into consideration, we are aware that this Master's thesis does not include experimental testing or results of the system's operational performance. The primary reason for this omission is the time constraints associated with the extensive process of manufacturing the physical device, which, regrettably, fell beyond the scope of this thesis. However, it is crucial to highlight that prior experiences in the SUMHAL project did involve the creation and field testing of a system with comparable, though more modest, functionalities. The insights and expertise gained from these earlier endeavors serve as a robust base for the present development efforts.

Owing to the absence of field deployment, a comprehensive evaluation of the performance of the wildlife monitoring system in real-world scenarios remains unexplored. This highlights the need for several future tasks.

4.2 Future Directions

Recognizing the extensive and varied nature of embedded system development, future directions are proposed for the evolution and realization of the proposed wildlife monitoring system.

4.2.1 Test Plan

A thorough test plan should encompass various aspects, including:

- **Electrical Testing:** In particular, testing of the power management unit will be conducted to validate its efficiency and reliability. This will involve assessing the unit's ability to regulate power sources, charge the battery, and supply power to different components of the system. Efficiency and consistency will be key evaluation criteria.
- **Functional Testing:** Functional testing will focus on the different interfaces of the system. The primary objective is to ensure seamless integration with the firmware and hardware-software interface. This testing phase will validate the proper operation of sensors, data acquisition, storage, and communication interfaces. It will also include evaluations of sensor accuracy, data integrity, and data transmission reliability.
- **Hardware-Software Integration:** Rigorous hardware-software integration tests will be conducted to assess compatibility and data flow between the hardware components and the AI processing unit. This phase aims to confirm that the AI unit can effectively process the data collected by the firmware and that data are transmitted accurately. This testing will address compatibility, data transfer rates, and data preprocessing.

4.2.2 Characterization

The process of characterization is essential for gaining a comprehensive understanding of the wildlife monitoring system and creating a detailed specifications datasheet.

- **Power Consumption Characterization:** Measurements of its power consumption in various operating modes will be carried out. Specialized laboratory instruments will be employed to provide accurate and reliable data.
- **Performance Characterization:** An important aspect of characterization involves assessing performance metrics, particularly in terms of processing efficiency. This entails evaluating the system's computational capabilities and its ability to process data in a timely and efficient manner. The results of performance characterization will inform the specifications related to processing capacity, speed, and performance under different workloads.
- **Comparative Analysis:** A comparative study will be conducted to assess the power consumption and performance metrics of the wildlife monitoring system when using AI models. This comparison will include reference to AI models previously tested at [17], providing valuable insights into potential power efficiency improvements.

4.2.3 Field testing and validation

Field testing is a critical phase in the development process. It involves deploying the wildlife monitoring system in real-world environments to evaluate its performance under actual operating conditions. Field testing serves multiple purposes, including:

- **Performance Validation:** This is particularly important to ensure that the system functions as intended and can withstand the challenges and variations encountered in natural environments.

- **Environmental Adaptability:** Real-world deployment allows for assessing the system's adaptability to diverse environmental conditions. It is essential to confirm that the device can operate reliably across a range of temperatures, humidity levels, and other environmental factors commonly found in wildlife habitats. The device's internal temperature and humidity sensor will facilitate this analysis.
- **Durability and Reliability:** The wildlife monitoring system must demonstrate durability and reliability during field testing. It should withstand exposure to environmental elements, potential physical impacts, and extended operation periods. This phase helps identify any weaknesses and areas for improvement in the system's design and materials.
- **Data Validation:** Field testing also serves to validate the quality and integrity of the data collected by the system. It allows for comparison of the data generated by the device with real-world observations thanks to the validation of biologists, helping to ensure the accuracy and reliability of the collected information.

Chapter 5

Conclusions

Building on the insights gained during the development of a basic system within the SUMHAL project, which included extensive field deployment and collaborative data validation alongside biologists, this Master's thesis undertook a mission to address a series of objectives in the creation of an innovative wildlife monitoring system. These objectives have been methodically explored, setting the basis for a versatile, energy-efficient, and AI-enhanced embedded system.

Energy efficiency has been at the forefront of the design process. The thesis recognized the critical role of low-power operation in extending the field deployment capabilities of wildlife monitoring systems. By carefully selecting components and implementing advanced power management techniques, the system has been engineered to minimize energy consumption and reduce maintenance requirements. The architecture, power management unit, and choice of components, such as LiPo battery charger and efficient switch-mode power supplies controllers, collectively work to ensure efficient energy utilization, making the system capable of prolonged autonomous operation.

The development process placed a strong emphasis on designing a hardware framework that supports advanced AI capabilities. This framework facilitates real-time data analysis and decision-making directly within the embedded system. Through the seamless integration of AI processing units, the system is capable of executing complex video and audio analytics, recognizing patterns, and providing valuable insights into wildlife and habitat conditions. The versatility of the device is further enhanced by the ability to enable or disable the AI processing unit as needed, coupled with the low-power MCU's capacity for basic AI implementations. This adaptability makes the device cost-effective and suitable for a wide range of applications.

The system has been designed to seamlessly integrate data from a diverse array of sources, encompassing visual, acoustic, and environmental inputs. This multimodal data integration capability empowers the system to efficiently collect, process, and combine various data streams, leveraging a variety of sensors that comply with the Grove standard and other provided interfaces, including I2S microphones, SPI cameras, and CSI advanced image sensors. With its versatile architecture, the system can be effectively employed across a wide spectrum of applications in conservation technology, including smart camera trapping, passive acoustic monitoring, and environmental or individual activity detection.

Given the varying network environments in which the proposed system can operate, communication network modularity has been another major design aspect. The system has been engineered to offer adaptability in communication networks

through the MikroBus standard, making it easily configurable to meet the requirements of different projects. This modularity enables a seamless connection to a variety of network environments, ensuring that the system can operate efficiently in diverse contexts.

The successful integration of hardware and software components is a key element in achieving a highly functional device and streamlining the development process, facilitating configuration for future projects. The proposed software architecture establishes a symbiotic relationship between hardware and software, guaranteeing smooth data flow and enabling efficient communication and processing between various system components, from the low-level firmware to the embedded software running on the Linux OS, through the RPC method. This integration not only enhances system performance but also paves the way for future advancements and adaptability.

Finally, recognizing the several steps that remain to be completed outside of this Master's thesis due to time restrictions, a roadmap has been devised for the next phases of development. This roadmap outlines the necessary tasks and milestones to progress toward the ultimate goal of a fully operational wildlife monitoring system.

In summary, considering the research group's focused efforts on the development of smart conservation technologies, encompassing novel AI algorithm design and implementation, new sensor technology designs, and efficient IoT communication flows, the proposed wildlife monitoring system is anticipated to play a fundamental role in future studies. It will become the centerpiece for conducting cutting-edge research in the field based on advanced technology. This will enhance the group's capacity to make more significant contributions to the critical issue of massive biodiversity loss.

Appendix A

Microcontroller Pin Allocation

Pin Allocation

GPIO	SERCOM	PAD	NAME	PURPOSE	ALT NAME	ALT PURPOSE
PA00			Oscillator	Oscillator		
PA01			Oscillator	Oscillator		
PA02			BAT_LVL	BATTERY LEVEL		
PA03			Aref			
PA04			A0	CON_A1		
PA05			A1			
PA06			A2	CON_A2		
PA07			A3			
PA08	SERCOM0	PAD[0]	TXD_UART1	COM_CM4		
PA09	SERCOM0	PAD[1]	RXD_UART1			
PA10			CS_SPI0_CAM			SPI CAMERA
PA11			MIKRO_INT	MIKROBUS		
PA12	SERCOM2	PAD[0]	MOSI_SPI0		MOSI_SPI0	SPI CAMERA
PA13	SERCOM2	PAD[1]	SCK_SPI0		SCK_SPI0	
PA14	SERCOM2	PAD[2]	MISO_SPI0		MISO_SPI0	
PA15	SERCOM2	PAD[3]	CS_SPI0			
PA16	SERCOM1	PAD[0]	TXD_UART0			
PA17	SERCOM1	PAD[1]	RXD_UART0			
PA18			D2	CON_D2		
PA19			D3			
PA20			I2S_FS	I2S MICROPHONE		
PA21			I2S_SDO			
PA22			I2S_SDI			
PA23			BUT_CONF1			
PA24			USB_D-	USB		
PA25			USB_D+			
PA27			BUT_CONF2			
PA30			SWDCLK	PROGRAMMER		
PA31			SWDIO			
PB00			FALL_INT	ACCELEROMETER INTERRUPT.		
PB01			MOV_INT			
PB02	SERCOM_ALT	PAD[0]	SDA_I2C0	MIKROBUS	SDA_I2C0	INTERNAL SENSORS
PB03	SERCOM_ALT	PAD[1]	SCL_I2C0		SCL_I2C0	
PB04			RST			
PB05			PWM			
PB06			LOAD1_ON1	LOAD SWITCHED CONTROLL		
PB07			LOAD1_ON2			
PB08			LOAD2_ON1			
PB09			LOAD2_ON2			
PB10			BUT_CONF3			
PB11			SD_MUX_SEL	SD CARD	FLASH_CS	
PB12	SERCOM4	PAD[0]	SPI1_MISO			
PB13	SERCOM4	PAD[1]	SP1_SCK			
PB14	SERCOM4	PAD[2]	SP1_CS			
PB15	SERCOM4	PAD[3]	SP1_MOSI			
PB16			I2S_SCK	I2S MICROPHONE		
PB17			I2S_MCK			
PB22			CHG_STAT	CHARGER STATUS		
PB23			PGOOD			
PB30	SERCOM7	PAD[0]	D0	CON_D2	SWO	
PB31	SERCOM7	PAD[1]	D1			

Appendix B

Schematics

LOW-POWER MCU MODULAR BOARD

Power Circuit

Archivo: power.kicad_sch

internal-sensors

Archivo: internal-sensors.kicad_sch

Microcontroller

Archivo: microcontroller.kicad_sch

Communication Module

Archivo: mikrobus-module.kicad_sch

AI PROCESSING UNIT (CM4)

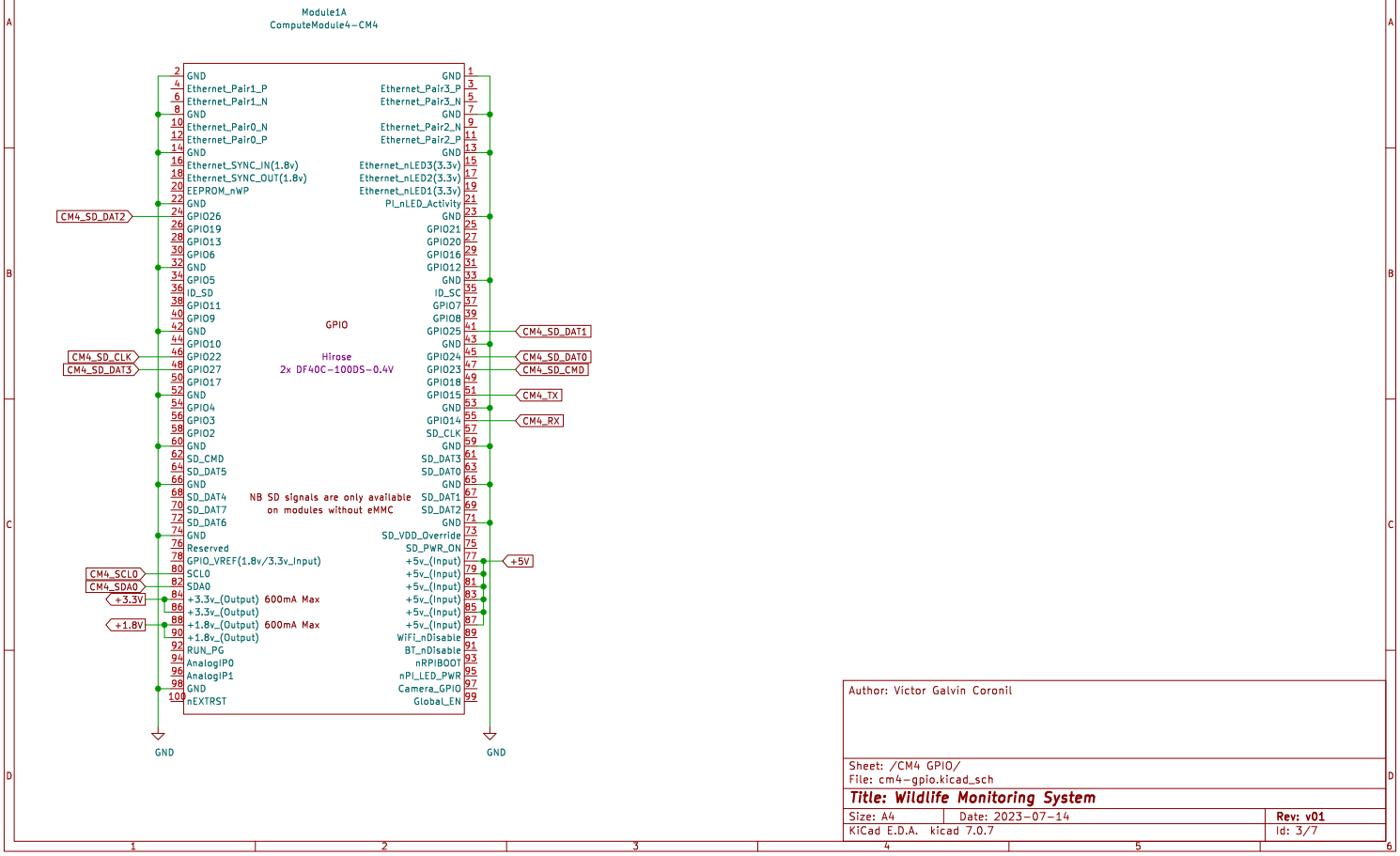
CM4 GPIO

Archivo: cm4-gpio.kicad_sch

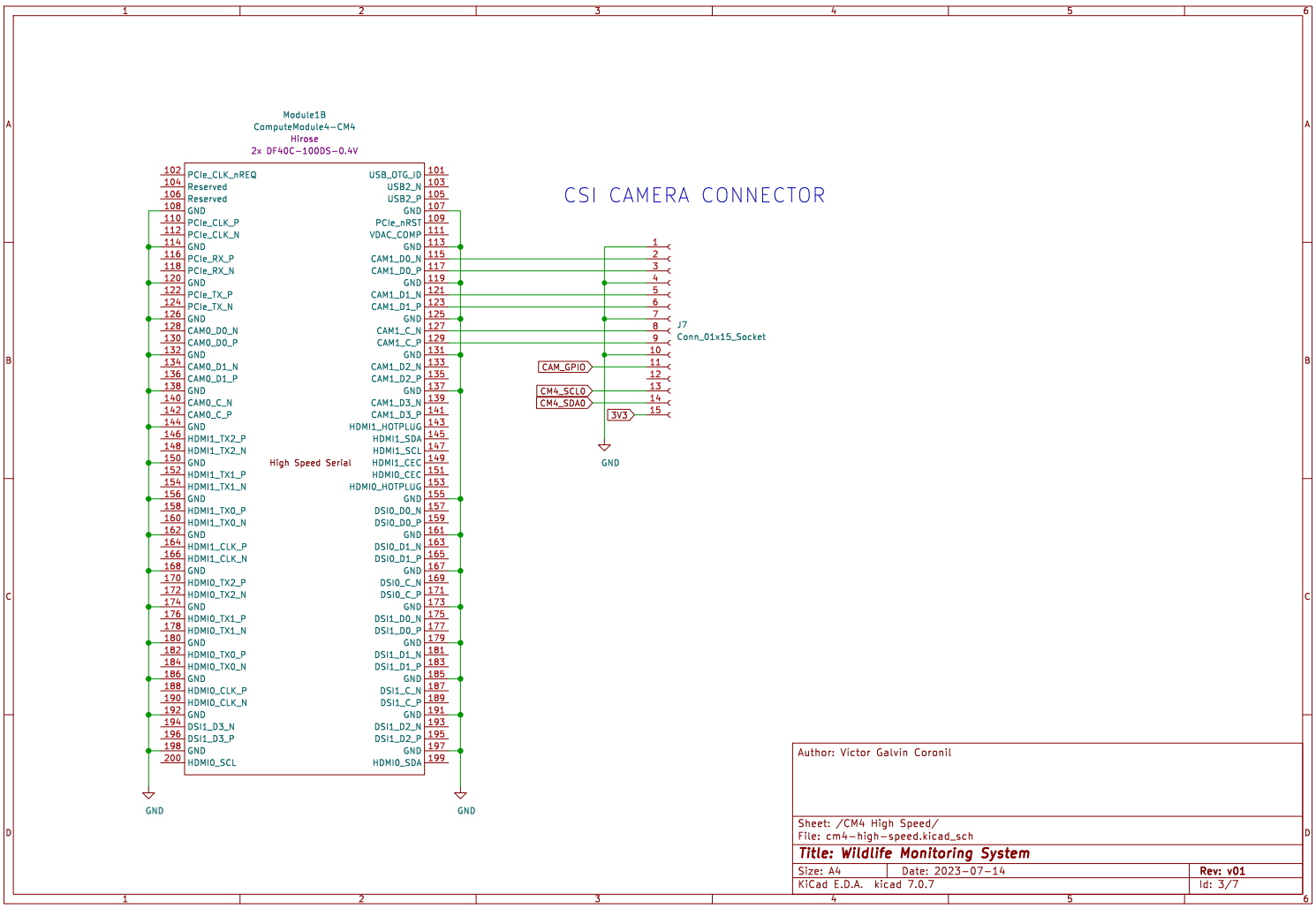
CM4 High Speed

Archivo: cm4-high-speed.kicad_sch

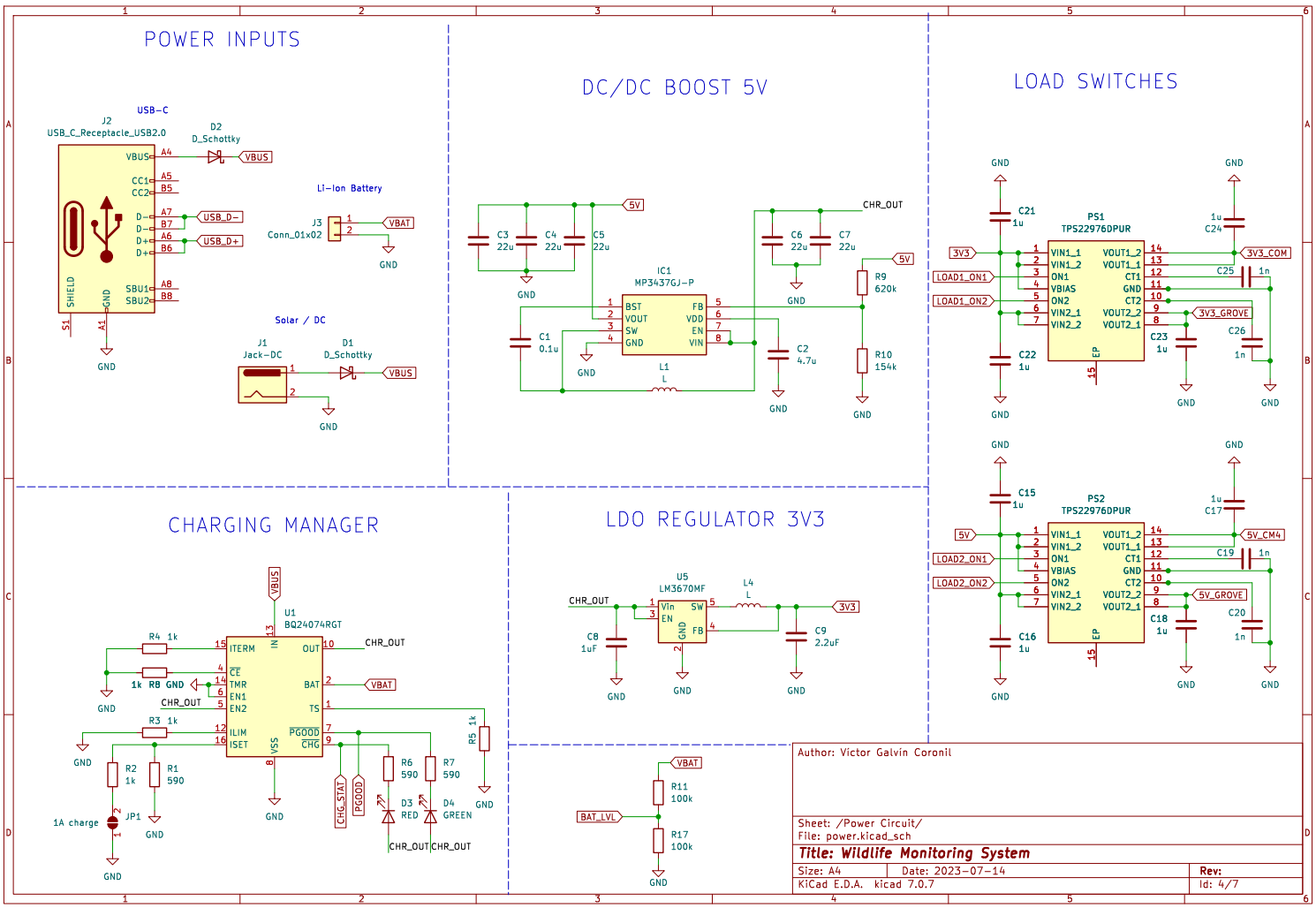
Author: Victor Galvin Coronil		
Sheet: / File: biodaimod.kicad_sch		
Title: Wildlife Monitoring System		
Size: A4	Date: 2023-07-14	**Rev: v01**
KiCad E.D.A. kicad 7.0.7		Id: 1/7

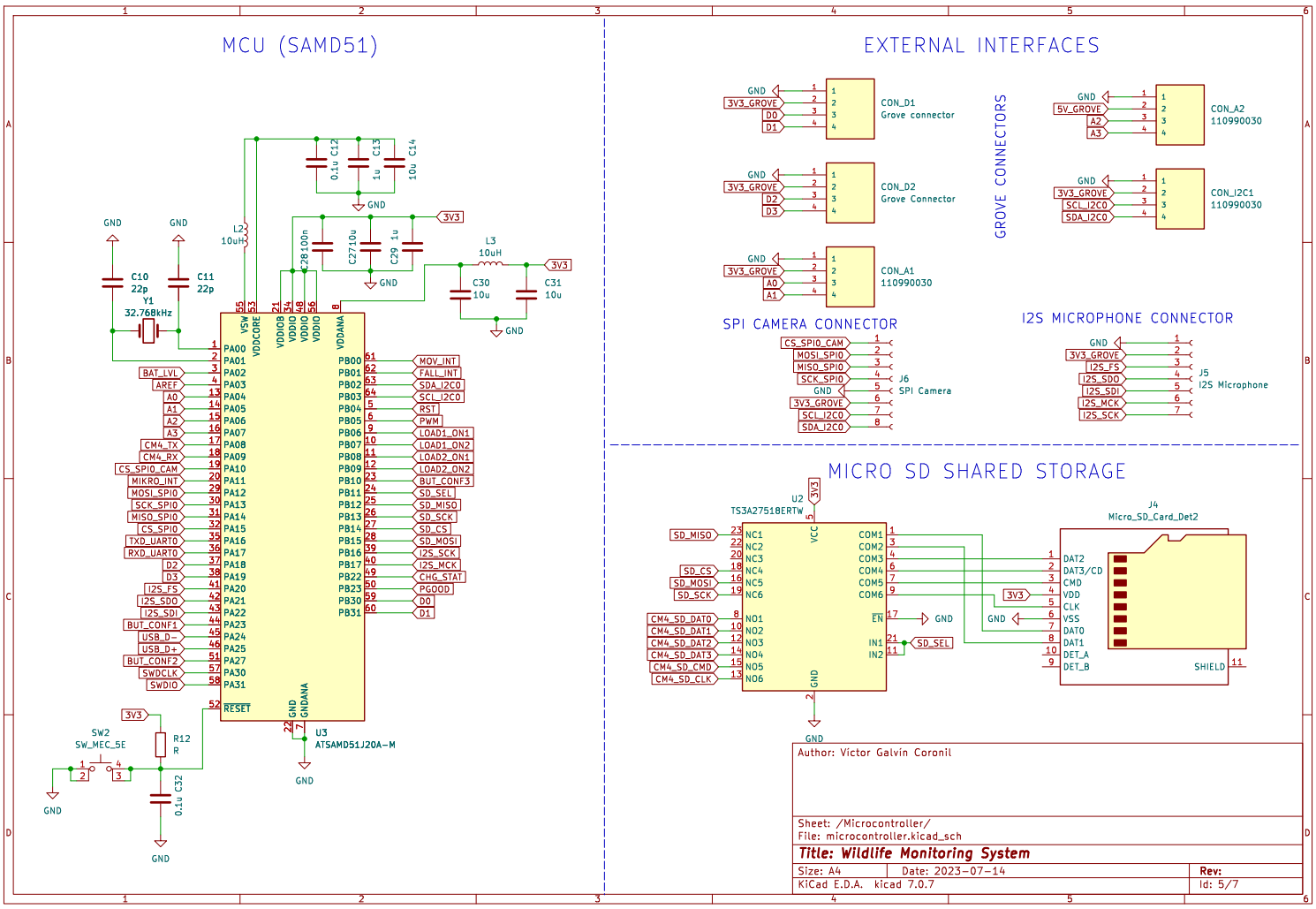


Author: Victor Galvin Coronil		
Sheet: /CM4 GPIO/ File: cm4-gpio.kicad_sch		
Title: Wildlife Monitoring System		
Size: A4	Date: 2023-07-14	**Rev: v01**
KiCad E.D.A. kicad 7.0.7		Id: 3/7

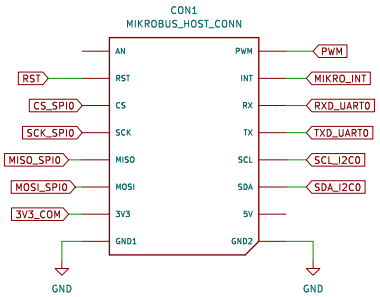


Author: Victor Galvin Coronil		
Sheet: /CM4 High Speed/ File: cm4-high-speed.kicad_sch		
Title: Wildlife Monitoring System		
Size: A4	Date: 2023-07-14	**Rev: v01**
KiCad E.D.A. kicad 7.0.7		Id: 3/7


POWER INPUTS
USB-C
J2
USB_C_Receptacle_USB2.0
D2
D_Schottky
VBUS
USB_D-
USB_D+
SHIELD
GND
S1
A1
Li-Ion Battery
J3
Conn_01x02
VBAT
Solar / DC
J1
Jack-DC
D1
D_Schottky
DC/DC BOOST 5V
IC1
MP3437GJ-P
BST
VOUT
SW
GND
FB
VDD
EN
VIN
5V
CHR_OUT
C1
0.1u
C2
4.7u
L1
L
R9
620k
R10
154k
LOAD SWITCHES
PS1
TPS22976DPUR
PS2
TPS22976DPUR
3V3
LOAD1_ON1
LOAD1_ON2
LOAD2_ON1
LOAD2_ON2
3V3_COM
3V3_GROVE
5V_CM4
5V_GROVE
CHARGING MANAGER
U1
BQ24074RGT
ITERM
CE
TMR
EN1
EN2
ILIM
ISET
IN
OUT
BAT
TS
PGOOD
CHG
VSS
R1
590
R2
1k
R3
1k
R4
1k
R5
1k
R6
590
R7
590
1k R8 GND
1A charge
JP1
CHG_STAT
PGOOD
D3
RED
D4
GREEN
LDO REGULATOR 3V3
U5
LM3670MF
Vin
SW
EN
GND
FB
L4
C8
1uF
C9
2.2uF
R11
100k
R17
100k
BAT_LVL
Author: Victor Galvin Coronil
Sheet: /Power Circuit/
File: power.kicad_sch
Title: Wildlife Monitoring System
Size: A4
Date: 2023-07-14
Rev:
KiCad E.D.A. kicad 7.0.7
Id: 4/7

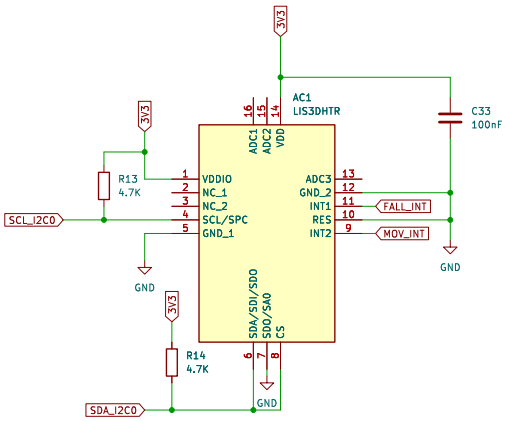

MCU (SAMD51)
EXTERNAL INTERFACES
GROVE CONNECTORS
CON_D1
Grove connector
CON_D2
Grove Connector
CON_A1
110990030
CON_A2
110990030
CON_I2C1
110990030
SPI CAMERA CONNECTOR
J6
SPI Camera
I2S MICROPHONE CONNECTOR
J5
I2S Microphone
MICRO SD SHARED STORAGE
U2
TS3A27518ERTW
J4
Micro_SD_Card_Det2
U3
ATSAMD51J20A-M
SW2
SW_MEC_5E
Author: Victor Galvin Coronil
Sheet: /Microcontroller/
File: microcontroller.kicad_sch
Title: Wildlife Monitoring System
Size: A4
Date: 2023-07-14
KiCad E.D.A. kicad 7.0.7
Rev:
Id: 5/7

MIKROBUS CONNECTOR

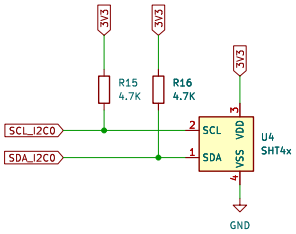


Author: Victor Galvin Coronil		
Sheet: /Communication Module/ File: mikrobus-module.kicad_sch		
Title: Wildlife Monitoring System		
Size: A4	Date: 2023-07-14	**Rev:**
KiCad E.D.A. kicad 7.0.7		Id: 6/7

ACCELEROMETER



TEMPERATURE/HUMIDITY



Author: Victor Galvin Coronil	
Sheet: /internal-sensors/ File: internal-sensors.kicad_sch	
Title: Wildlife Monitoring System	
Size: A4 | Date: 2023-07-14	**Rev:**
KiCad E.D.A. kicad 7.0.7	Id: 7/7

Appendix C

Firmware code

C.1 Main process - *main.cpp*

```
#include "config.h"
#include "logs.h"
#include "communication.h"
#include "ota.h"
#include "time.h"
#include "rfid_acq.h"
#include "sd_utils.h"
#include "sensors.h"

uint8_t mode = 0;

extern char fechaChar[20];

const char* title = "fw";
const char* version = "1";

bool continuous_mode = true;
bool connection_status;
float battery_level;
float temperature;

void connectionStatus();

// SD
String lineBackup;
boolean initSD = false;

void setup() {
  delay(5000);

  //================== Serial Port setup ==================
  Serial.begin(115200);
  Serial.println("Start debug serial");

  pinMode(A5, OUTPUT);

  pinMode(A1, OUTPUT);
  digitalWrite(A1, HIGH);

  //================== SD setup ==================
  initSD = initializeSD();

  //================ Communication setup =============
  Serial.println("Initialize Connection");
  beginConnection();
  delay(1000); // give the Ethernet shield a second to initialize:
```

```
47
48    //================== RTC setup ================
49    RTCsetUp();
50
51    //================== Logs setup ================
52    // SD initialization logs
53    SDinitLogs();
54
55    // Ethernet initialization logs
56    EthernetInitLogs();
57
58    //================== Cloud Connection =============
59    TBconnection();
60
61    //setAlarmRTC();
62    initServo();
63  }
64
65  void loop() {
66
67    switch (mode)
68    {
69    case SENSOR_ACQ:
70      Serial.println("____SENSOR ACQUISITION MODE____");
71      battery_level = getBatteryVoltage();
72
73      initTempSensor();
74      temperature = getTemperature();
75      mode = RFID_ACQ;
76      break;
77
78    case PROCESSING:
79      Serial.println("____PROCESSING MODE____");
80      break;
81
82    case COMMUNICATION:
83      Serial.println("____COMMUNICATION MODE____");
84
85      if (!continuous_mode)
86      {
87        beginConnection();
88        TBconnection();
89      }
90      sendBakcupDataToTB();
91      sendSensorDataTB("vbat", battery_level);
92      sendSensorDataTB("temp", temperature);
93      if (!continuous_mode) {
94        powerOffModule();
95      }
96      mode = SLEEP;
97      break;
98
99    case SLEEP:
100     Serial.println("____SLEEP MODE____");
101
102     if (!continuous_mode) {
103       setAlarmRTC(0);
104       enableSleepMode();
105     }
106     else {
107       delay(10000);
108       mode = SENSOR_ACQ;
109     }
```

```
110     break;
111
112   default:
113     break;
114   }
115 }
```

C.2 Time module - *time.cpp*

```
1  #include "time.h"
2  #include "communication.h"
3
4  //RTC
5  RTCZero rtc;
6  String fechaString;
7  String fechaBackup;
8  String ts_millis;
9
10 extern HttpClient httpclient;
11 int timeFromServer = 0;
12 char fechaChar[20]= "1/1/1";
13 extern uint8_t mode;
14
15 String getFormatedDate(){
16
17   fechaString = rtc.getDay();
18   fechaString += "/";
19   fechaString += rtc.getMonth();
20   fechaString += "/";
21   fechaString += rtc.getYear();
22   fechaString += " ";
23   fechaString += rtc.getHours();  //UTC hour
24   fechaString += ":";
25   fechaString += rtc.getMinutes();
26   fechaString += ":";
27   fechaString += rtc.getSeconds();
28   return fechaString;
29 }
30
31 void RTCsetUp(){
32
33   rtc.begin();
34   timeFromServer = getServerTime();
35
36   rtc.setEpoch(timeFromServer);
37 }
38
39  void setAlarmRTC(uint8_t seconds) {
40    // Configure RTC alarm to upload connection status
41    //rtc.setAlarmMinutes(minutes);
42    rtc.setAlarmSeconds(seconds);
43    rtc.enableAlarm(rtc.MATCH_SS);
44    rtc.attachInterrupt(alarmMatch);
45    delay(50);
46    Serial.println("Alarm set");
47  }
48
49  void alarmMatch() {
50   Serial.println("Alarm match produced");
51   mode = SENSOR_ACQ;
```

```
}

void enableSleepMode() {
 Serial.println("Entering sleep mode");
 rtc.standbyMode();
}
```

C.3 Communication module (GSM transceptor) - *communication.cpp*

```
#include "communication.h"
#include "time.h"
#include "logs.h"
#include "config.h"
#include "rfid_acq.h"
#include "sd_utils.h"

TinyGsm        modem(Serial2);
TinyGsmClient client(modem);
PubSubClient  mqtt(client);

ThingsBoard tb(client);
HttpClient httpclient = HttpClient(client, "10.15.1.217", 8080);

// GPRS credentials, if any
// const char apn[]      = "orangeworld";
const char apn[]      = "airtelnet.es";
const char gprsUser[] = "";
const char gprsPass[] = "";

// MQTT parameters
const char* broker = "161.111.232.198"; //MQTT broker EBD
const char* tb_topic = "/sensor/data"; // Topic telemetries Thingsboard
const char* mqtt_client = "GSMdevice";
const char* mqtt_user = "user";
const char* mqtt_pass = "hS4H4JmjzAW@9x";

extern String fechaBackup;
extern String fechaString;
extern String ts_millis;
extern char fechaChar[20];
extern char mensajeLogTB_KO [35];
extern char mensajeLogTB_OK [35];
extern char mensajeLogBackupToTB [35];
extern char rfid_code[12];
extern String lineBackup;
extern bool connection_status;
char rfid_json_char[120];
bool subscribed = false;
String rfid_json_string;

bool tbDataSendCheck = true;
bool dataSentCheck = true;

extern RTCZero rtc;


void beginConnection() {
  digitalWrite(A5, HIGH);
```

```
50    delay(1000);
51    digitalWrite(A5, LOW);
52    delay(1500);
53    SerialAT.begin(9600);
54
55
56    Serial.println("Initializing GMS modem...");
57
58    modem.init();
59
60    String modemInfo = modem.getModemInfo();
61    Serial.print("Modem Info: ");
62    Serial.println(modemInfo);
63
64    if (GSM_PIN && modem.getSimStatus() != 3) {
65    Serial.println("Unlocking SIM with PIN...");
66    modem.simUnlock(GSM_PIN);
67    }
68
69    Serial.print("Waiting for network...");
70    if (!modem.waitForNetwork()) {
71      Serial.println(" fail");
72      delay(10000);
73      return;
74    }
75    Serial.println(" success");
76
77    if (modem.isNetworkConnected()) { Serial.println("Network connected")
        ; }
78
79      // GPRS connection parameters are usually set after network
        registration
80    Serial.print(F("Connecting to "));
81    Serial.print(apn);
82    if (!modem.gprsConnect(apn, gprsUser, gprsPass)) {
83      Serial.println(" fail");
84      delay(10000);
85      return;
86    }
87    Serial.println(" success");
88
89    if (modem.isGprsConnected()) { Serial.println("GPRS connected"); }
90
91    // MQTT Broker setup
92    mqtt.setServer(broker, 1883);
93    // mqtt.setCallback(mqttCallback); TODO: include for biderectional
        communication
94  }
95
96  bool checkServerConnection() {
97
98    // Make sure we're still registered on the network
99    if (!modem.isNetworkConnected()) {
100     Serial.println("Network disconnected");
101     if (!modem.waitForNetwork(180000L, true)) {
102       Serial.println(" fail");
103       delay(10000);
104       return false;
105     }
106     if (modem.isNetworkConnected()) {
107       Serial.println("Network re-connected");
108       return true;
109     }
```

```
110
111     // and make sure GPRS/EPS is still connected
112     if (!modem.isGprsConnected()) {
113       Serial.println("GPRS disconnected!");
114       Serial.print(F("Connecting to "));
115       Serial.print(apn);
116       if (!modem.gprsConnect(apn, gprsUser, gprsPass)) {
117         Serial.println(" fail");
118         delay(10000);
119         return false;
120       }
121       if (modem.isGprsConnected()) {
122         Serial.println("GPRS reconnected");
123         return true;
124         }
125     }
126   }
127 }
128
129 void sendBakcupDataToTB(){
130
131   //bool connection_status = checkServerConnection();
132   bool connection_status = true;
133   Serial.print("Estado de la conexion: ");
134   Serial.println(connection_status);
135
136   if (connection_status == true ) {
137     lineBackup = readLine("backup.csv", false, true);
138
139     while(lineBackup != "" && tbDataSendCheck == true){   // TODO:
    tbDataSendCheck
140
141       Serial.println("Backup data: ");
142       Serial.println(lineBackup);
143
144       getValuesFromSDLine();
145
146       ts_millis = rtc.getEpoch();
147       Serial.println("TS GET EPOCH");
148       Serial.println(ts_millis);
149
150       rfidJsonFormat();
151
152       Serial.println("Sending backup data: ");
153       writeFile("log.txt",mensajeLogBackupToTB, fechaChar);
154
155       mqtt.publish(tb_topic, rfid_json_char);
156
157       lineBackup = readLine("backup.csv", false, false);
158
159       //Once everything is sent, delete de backup file
160       if(lineBackup == ""){
161         SD.remove("backup.csv");
162       }
163
164       //if(tbDataSendCheck == false){      TO DO: Check from TB the
    reception of the packages
165       //
166       //}
167       delay(3000);
168     }
169   }
170 }
```

```
171
172 void TBconnection(){ // TODO: return bool with connection status
173
174   Serial.print("Connecting to ");
175   Serial.print(broker);
176
177   // Connect to MQTT Broker
178   boolean status = mqtt.connect(mqtt_client, mqtt_user, mqtt_pass);
179
180   if (status == false) {
181     Serial.println(" fail");
182     //return false;
183   }
184   Serial.println(" success");
185
186   mqtt.connected();
187 }
188
189 void powerOffModule() {
190   modem.gprsDisconnect();
191   delay(5000L);
192   if (!modem.isGprsConnected()) {
193     Serial.println("GPRS disconnected");
194   } else {
195     Serial.println("GPRS disconnect: Failed.");
196   }
197     modem.poweroff();
198   Serial.println("Poweroff.");
199   digitalWrite(A5, HIGH);
200   delay(1000);
201   digitalWrite(A5, LOW);
202   delay(1500);
203 }
204
205 void tbLoop() {
206     //tb.loop();
207 }
208
209 int getServerTime() {
210
211   DBG("Retrieving time again as a string");
212   String time = modem.getGSMDateTime(DATE_FULL);
213   DBG("Current Network Time:", time);
214
215 }
```

C.4 Sensors module (temperature) - *sensors.cpp*

```
1  #include "sensors.h"
2
3  OneWire oneWire(ONE_WIRE_BUS);
4  DallasTemperature sensors(&oneWire);
5
6  Servo myservo;
7
8  float getBatteryVoltage() {
9      float measuredvbat = analogRead(VBATPIN);
10     measuredvbat *= 2;    // we divided by 2, so multiply back
11     measuredvbat *= 3.3;  // Multiply by 3.3V, our reference voltage
12     measuredvbat /= 1024; // convert to voltage
```

```
    Serial.print("VBat: " ); Serial.println(measuredvbat);

    return measuredvbat;
}

void initTempSensor() {
  sensors.begin();
  delay(50);
}

float getTemperature() {
    sensors.requestTemperatures();
    float temp = sensors.getTempCByIndex(0);
    Serial.print("Temperature: " ); Serial.println(temp);
    return temp;
}
```

C.5 SD utils module - *sd-utils.cpp*

```
#include "sd_utils.h"

File myFile;

boolean initializeSD() {

  int CHIPSELECT = 4; //Pin for the SD card
  pinMode(CHIPSELECT,OUTPUT);//Pin to initialize SD card
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
   if (!SD.begin(CHIPSELECT)) {
    Serial.println("SD initialization failed!");
    return false;
  } else {
    Serial.println("SD initialization done.");
    return true;
  }
}

String readLine(char *name, bool close_file, bool open_file) {
  String lineRead;

  if (open_file){
    myFile = SD.open(name, FILE_READ);
    Serial.print("Open file ");
    Serial.println(name);
    Serial.println(myFile.available());
  }
  if (myFile) {
    // read one line of the file
    Serial.println(myFile.available());
    lineRead = myFile.readStringUntil('\n');

    // close the file:
    if (close_file) {
      myFile.close();
      Serial.print("Close file ");
    Serial.println(name);
    }
  }
  return lineRead;
```

```
}


String stringSplitRFID(String line, int index1, int index2){
    String dato = line.substring(index1,index2);
    return dato;
}

boolean writeFile(char *name, char *lectura, char *fecha){
  myFile = SD.open(name, FILE_WRITE);
  // if the file opened okay, write to it:
  if (myFile) {
    Serial.print("Writing to: ");
    Serial.print(name);
    myFile.print(fecha);
    myFile.print(",");
    myFile.println(lectura);
    // close the file:
    myFile.close();
    Serial.println(" done.");
    return true;
  } else {
    // if the file didn't open, print an error:
    Serial.print("error opening ");
    Serial.println(name);
    return false;
  }
}
```

C.6 OTA module - *ota.cpp*

```
#include "ota.h"
#include "config.h"
#include "communication.h"

extern HttpClient httpclient;
extern EthernetClient client;

//OTA
const char* BIN_FILENAME = "update.bin"; // expected by the SDU library
const short VERSION = 1;
bool enviado = false;
bool actualizacionFirmware = false;
String tbFirmwareVersion = "";
const char* charToken = "C1_TEST_TOKEN";

void handleSketchDownload(const char* title, const char* version) {
  Serial.println("UPDATING FIRMWARE CRACK");
  //const char* SERVER = "10.15.1.217"; // must be string for
    HttpClient
  //const unsigned short SERVER_PORT = 8080;
  SD.remove(BIN_FILENAME);
  String path = "/api/v1/";
  path.concat(charToken);
  path.concat("/firmware?title=");
  path.concat(title);
  path.concat("&version=");
  path.concat(version);
  char buff[64];
  char* pathChar;
```

```
29    path.toCharArray(pathChar,64,0);
30    snprintf(buff, sizeof(buff), pathChar, VERSION + 1);
31    Serial.print("Check for update file ");
32    Serial.println(buff);
33    httpclient.get(path);
34    int statusCode = httpclient.responseStatusCode();
35    Serial.print("Update status code: ");
36    Serial.println(statusCode);
37    if (statusCode != 200) {
38      client.stop();
39      return;
40    }
41    long length = httpclient.contentLength();
42    if (length == HttpClient::kNoContentLengthHeader) {
43      client.stop();
44      Serial.println("Server didn't provide Content-length header. Can't
        continue with update.");
45      return;
46    }
47    Serial.print("Server returned update file of size ");
48    Serial.print(length);
49    Serial.println(" bytes");
50    File file = SD.open(BIN_FILENAME, O_CREAT | O_WRITE);
51    if (!file) {
52      client.stop();
53      Serial.println("Could not create bin file. Can't continue with
        update.");
54      return;
55    }
56    byte b;
57    while (length > 0) {
58      if (!client.readBytes(&b, 1)) // reading a byte with timeout
59        break;
60      file.write(b);
61      length--;
62    }
63    file.close();
64    client.stop();
65    if (length > 0) {
66      SD.remove(BIN_FILENAME);
67      Serial.print("Timeout downloading update file at ");
68      Serial.print(length);
69      Serial.println(" bytes. Can't continue with update.");
70      return;
71    }
72    //    sd_utils.writeFile("log.txt",mensajeLogSD_newFirmware, fechaChar
        );
73    Serial.println("Update file saved. Reset.");
74    Serial.flush();
75    NVIC_SystemReset();
76  }
```

Bibliography

[1] R. Carmona-Galán. *V-MOTE project*. 2010. URL: http://www2.imse-cnm.csic.es/vmote/english_version/ (visited on 2023).

[2] Delia Velasco-Montero, Jorge Fernández-Berni, and Angel Rodríguez-Vázquez. *Visual Inference for IoT Systems: A Practical Approach*. Springer International Publishing, 2022. DOI: 10.1007/978-3-030-90903-1.

[3] *SUMHAL*. 2022. URL: https://lifewatcheric-sumhal.csic.es/en/.

[4] International Union for Conservation of Nature. *International Union for Conservation of Nature Web*. 2023. URL: https://www.iucn.org/ (visited on 2023).

[5] *Adafruit Feather M0 Adalogger*. SAMD21. Adafruit. Dec. 2015. URL: https://learn.adafruit.com/adafruit-feather-m0-adalogger/overview.

[6] *Low-Power, 32-bit Cortex-M0+ MCU*. ATSAMD21G18. Microchip Technology Inc. 2021. URL: https://ww1.microchip.com/downloads/aemDocuments/documents/MCU32/ProductDocuments/DataSheets/SAM-D21-DA1-Family-Data-Sheet-DS40001882H.pdf.

[7] *Standard specifications*. MikroBus. MikroElektronika. 2015. URL: https://download.mikroe.com/documents/standards/mikrobus/mikrobus-standard-specification-v200.pdf.

[8] *Grove Ecosystem*. Seeed Technology Co. URL: https://wiki.seeedstudio.com/Grove_System/.

[9] *Realtime Clock and Power Management for Raspberry Pi*. Witty Pi 4. UUGear. URL: https://www.uugear.com/doc/WittyPi4_UserManual.pdf.

[10] Andrew K. Schulz et al. "Conservation tools: the next generation of engineering–biology collaborations". In: *Journal of the Royal Society Interface* 20.205 (Aug. 2023). DOI: 10.1098/rsif.2023.0232.

[11] Talia Speaker et al. "A global community-sourced assessment of the state of conservation technology". In: *Conservation Biology* 36.3 (Feb. 2022). DOI: 10.1111/cobi.13871.

[12] Andrew P. Hill et al. "AudioMoth: A low-cost acoustic device for monitoring biodiversity and the environment". In: *HardwareX* 6 (Oct. 2019), e00073. DOI: 10.1016/j.ohx.2019.e00073.

[13] Carmen Lozano Pons. "Cyber-Physycal Systems for Remote Animal Monitoring". MA thesis. University of Sevilla, 2020. URL: https://idus.us.es/handle/11441/127955.

[14] Conservation X Labs. *Sentinel*. URL: https://conservationxlabs.com/sentinel.

[15] Robin C. Whytock et al. "Real-time alerts from AI-enabled camera traps using the Iridium satellite network: A case-study in Gabon, Central Africa". In: *Methods in Ecology and Evolution* 14.3 (Jan. 2023), pp. 867–874. DOI: 10.1111/2041-210x.14036.

[16] Delia Velasco-Montero, Jorge Fernández-Berni, and Angel Rodríguez-Vázquez. "A Case Study: Remote Animal Monitoring". In: *Visual Inference for IoT Systems: A Practical Approach*. Springer International Publishing, 2022, pp. 125–159. ISBN: 978-3-030-90903-1. DOI: 10.1007/978-3-030-90903-1.

[17] Delia Velasco-Montero et al. "Performance analysis of real-time DNN inference on Raspberry Pi". In: *Real-Time Image and Video Processing 2018*. Ed. by Nasser Kehtarnavaz and Matthias F. Carlsohn. SPIE, May 2018. DOI: 10.1117/12.2309763.

[18] Delia Velasco-Montero et al. "Towards an efficient smart camera trap for wildlife monitoring". In: *3rd International Workshop Camera Traps, AI and Ecology*. Jena, Germany, Sept. 2023.

[19] Edge Impulse. *Imagine 2022: Edge AI Conference*.

[20] Polyn Technology. *Neuromorphic Analog Implementation for Tiny AI Applications*. URL: https://www.polyn.ai/wp-content/uploads/2022/05/neuromorphic-analog-signal-processing-white-paper.pdf.

[21] *OpenMV Cam H7 Plus*. OpenMV. URL: https://openmv.io/products/openmv-cam-h7-plus.

[22] *Spresense 6-core microcontroller board with ultra-low power consumption*. CXD5602PWBMAIN1. Sony. URL: https://developer.sony.com/spresense/product-specifications#secondary-menu-desktop.

[23] *Portenta H7*. Arduino. URL: https://docs.arduino.cc/resources/datasheets/ABX00042-ABX00045-ABX00046-datasheet.pdf.

[24] *Jetson Modules*. NVIDIA. URL: https://developer.nvidia.com/embedded/jetson-modules.

[25] *32-bit ARM® Cortex®-M4F MCU*. ATSAMD51J20A. Microchip Technology Inc. URL: https://ww1.microchip.com/downloads/aemDocuments/documents/MCU32/ProductDocuments/DataSheets/SAM-D5x-E5x-Family-Data-Sheet-DS60001507.pdf.

[26] TensorFlow. *TensorFlow Lite for Microcontrollers*. URL: https://www.tensorflow.org/lite/microcontrollers.

[27] *Digital humidity and temperature sensor*. SHT40I-AD1B. Sensirion. URL: https://sensirion.com/media/documents/1D662E57/64D3B086/Sensirion_Datasheet_SHT4xI-Digital.pdf.

[28] *3-axis MEMS accelerometer*. LIS3DH. STMicroelectronics. URL: https://www.st.com/resource/en/datasheet/lis3dh.pdf.

[29] *5MP Plus OV5642 Mini Module Camera Shield SPI*. ArduCam. URL: https://www.arducam.com/product/arducam-5mp-plus-spi-cam-arduino-ov5642/.

[30] *Compute Module 4: A Raspberry Pi for deeply embedded applications*. Raspberry Pi Ltd. URL: https://datasheets.raspberrypi.com/cm4/cm4-datasheet.pdf.

[31] *Standalone 1-cell 1.5-A linear battery charger*. BQ24074. Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/bq24074.pdf?ts=1699114561355.

[32] *2MHz, 600mA Step-Down DC-DC Converter*. LM3671. Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/lm3671.pdf?ts=1699132926550&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fes-mx%252FLM3671.

[33] *16VOUT, 9.5A, High-Efficiency, Fully Integrated, Synchronous Boost Converter*. MP3437. Monolithic Power Systems, Inc. URL: https://www.monolithicpower.com/en/documentview/productdocument/index/version/2/document_type/Datasheet/lang/en/sku/MP3437/document_id/6889/.

[34] *2-ch, 5.7-V, 6-A, 14-mΩ, load switch*. TPS22976. Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/tps22976.pdf?ts=1699125999042&ref_url=https%253A%252F%252Fwww.ti.com%252Ftool%252FTIDA-01451.

[35] *3.3-V, 2:1 (SPDT), 6-channel analog multiplexer*. TS3A27518E. Texas Instruments. URL: https://www.ti.com/lit/gpn/ts3a27518e.

[36] *PlatformIO IDE*. URL: https://platformio.org/.