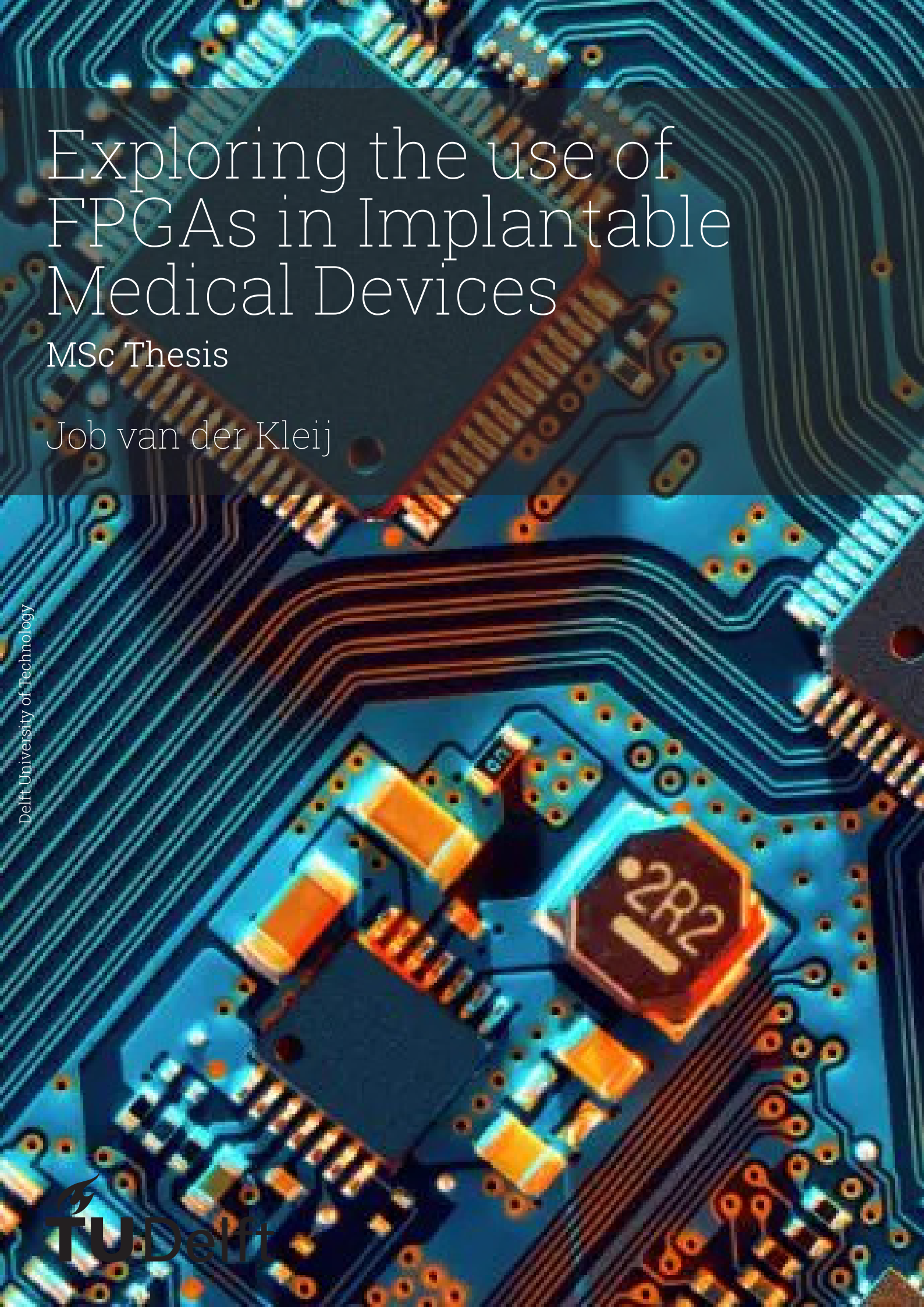
Exploring the use of FPGAs in Implantable Medical Devices

MSc Thesis

Job van der Kleij



Delft University of Technology

Exploring the use of FPGAs in Implantable Medical Devices

MSc Thesis

by

Job van der Kleij

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Thursday November 21, 2024 at 9:00 AM.

Student number:	4450906	
Project duration:	March 1, 2023 - November 21, 2024	
Thesis committee:	Dr. ir. C. Strydis	Erasmus Medical Center & TU Delft, thesis advisor
	Dr. ir. M.A. Siddiqi	Lahore University of Management Sciences , daily supervisor
	Prof. dr. ir. W.A. Serdijn	Erasmus Medical Center & TU Delft

TU Delft

List of Figures

List of Tables

List of abbreviations

ARX	add–rotate–XOR
ASIC	Application Specific Integrated Circuit
ASIP	Application-Specific Instruction set Processor
AES	Advanced Encryption Standard
CLK	Clock
CSD	Current-Source Density
CSV	Comma Separated Values
CPU	Central Processing Unit
ECG	electrocardiogram
EEG	electroencephalogram
EMC	Erasmus Medical Centre
FPGA	Field Programmable Gate Array
GPU	Graphic Processing Unit
HW	Hardware
I/O	Input/Output
IMD	Implantable Medical Device
IP	Intelectual Property
LFP	Local Field Potential
LUT	Look-Up Table
MAC	Multiply and Accumulate
MCU	MicroController Unit
MSB	Most Significant Bit
PCB	Printed Circuit Board
SRAM	Static Random Access Memory
SW	Software
TP	Testpoint
WMD	Wearable Medical Device

Abstract

This thesis aims to evaluate the viability of using a Field Programmable Gate Array (FPGA) in a neural Implantable Medical Device (IMD). The primary motivation for incorporating FPGAs is their potential to support future functionalities, such as running neural networks for medical condition analysis but also advanced cybersecurity algorithms. These algorithms are compute-intensive, and accelerators like FPGAs offer advantages in terms of speed and efficiency. To assess the effectiveness of such a device, state-of-the-art MicroController Units (MCUs) commonly used in similar applications are employed as a reference. Comparisons are made between MCU-only platforms and hybrid platforms integrating both an MCU and an FPGA. Feasibility analysis considers operational modes and use cases based on various realistic scenarios. The results show mixed outcomes across scenarios. Under a 100% duty cycle, the FPGA demonstrates higher efficiency, consuming less active power than the MCU. However, at lower duty cycles, MCUs are generally more effective on average. The use of an FPGA becomes practical when power-gating techniques are applied to minimize power consumption during inactive periods.

Contents

Acknowledgements

I would like to thank everyone who has supported me.

First of, from Erasmus Medical Centre and TU Delft, I would like to thank my thesis advisor, dr. ir. Christos Strydis, whose knowledge and vision made sure that the thesis went into the right direction.

I would also like to thank the daily supervisor dr. ir. M.A. Siddiqi, who was always available to help me through any problem that came up.

Finally, I would like to express my thanks towards my support system. My girlfriend, always ready to support and battle for me. My family, for proof reading the thesis, their helpful advice and welcome distractions. And finally, my friends, for always listening to my rambles. Without all of their unwavering support, I would not have finished my thesis.

Chapter 1

Introduction

This thesis delves into the realm of Implantable Medical Devices (IMDs). These platforms are implanted into the body of a patient to monitor and treat medical conditions. Once implanted, their primary function is to operate seamlessly and perform their intended tasks from within the human body. Due to being placed inside the human body, there are strict constraints imposed on the power consumption and heat generation .

IMDs have been in use for a considerable amount of time, evolving from basic solutions to sophisticated devices as technology advanced. Initially, these basic solutions were developed as immediate responses to problems, requiring minimal research. However, as knowledge expanded, so did the complexity of IMDs. For example, the development of the pacemaker involved extensive research over many years [1]. The pacemaker showcases a critical component of IMDs, they are safety-critical platforms. The IMD undergoes extensive research and development before reaching practical application to ensure their safety and effectiveness.

Today, advancements in implanted electronics, particularly, have expanded the capabilities of IMDs. These developments have enabled the use of electronics for purposes such as treating hearing impairments [2] and automating medicine distribution, such as in the case of diabetes [3].

1.1 Motivation

The future of IMDs holds vast potential. As technology advances, it is conceivable that people may choose to use implants to replace their biological components [4]. In the context of IMDs the goal is to stretch to new implementations. In the near future, the goal is to employ next-generation algorithms. These algorithms can range from new predictive analysis, to new (personalised) treatments, to machine learning [5]. Specifically neural networks within machine learning perform an ever-increasing amount of computations, made possible by ever increasing hardware capacity [6]. These networks, capable of performing increasingly complex tasks, would operate on the device, requiring more computing power, often run on Graphic Processing Unit (GPU) due to the parallel nature of the algorithm [6]. Therefore, a device or platform supported solely by a Central Processing Unit (CPU) may no longer be sufficient. In this project, we choose to try an alternative approach, by using an FPGA, given that it is a low-power alternative, which has benefits, befitting an IMD. Research signifies the extensive possibilities that come with integrating machine learning into healthcare [7] , which gives rise to an array of new possibilities, including use on an IMD.

1.2 Thesis goal

Previous works have attempted to find a definite result [8], but the research remains open-ended.

The goal, then, of this thesis work is to investigate whether if including a reconfigurable fabric in an IMD to enhance its computing capabilities is a viable option, given the stringent resource constraints of IMDs. Based on the thesis goal, the sub-goals are defined:

1. Find several algorithms which, combined, cover a wide range of active IMD operations.
2. Find several devices that correspond to processor types which might prove effective for an IMD.
3. Develop or alter the algorithms to work on each processor device effectively.
4. Measure performance and energy consumption across every algorithm and processor device.
5. Combine the measurements and compare the outcomes between processor devices.

1.3 Thesis structure

In Chapter 2, required background information is supplied. Chapter 3, introduces the experimental design, processor types, and algorithms. In Chapter 4, details of the implementations are provided. In Chapter 5, experimental results are presented. In Chapter 6, thesis contributions are summarised, while conclusions and potential future work are provided.

Chapter 2

Background

This chapter provides the necessary information to comprehend the key concepts of this thesis. It is divided into two sections: background information, which offers the essential knowledge to understand the core aspects of this research, and related works, where existing research relevant to this thesis is discussed.

2.1 Implants

A solid understanding of the structure and functionality of IMDs is essential for advancements in this field. Figure 2.1 illustrates the basic components commonly used in IMDs, starting with a MicroController Unit (MCU) as a base component and including sensors, stimulation electrodes (or actuators), and a general-purpose processing platform. An IMD may also feature additional processing hardware, such as accelerators or specialised chips, to enhance their capabilities. As IMDs evolve, integrating more advanced components continues to expand their potential applications in medical treatments.

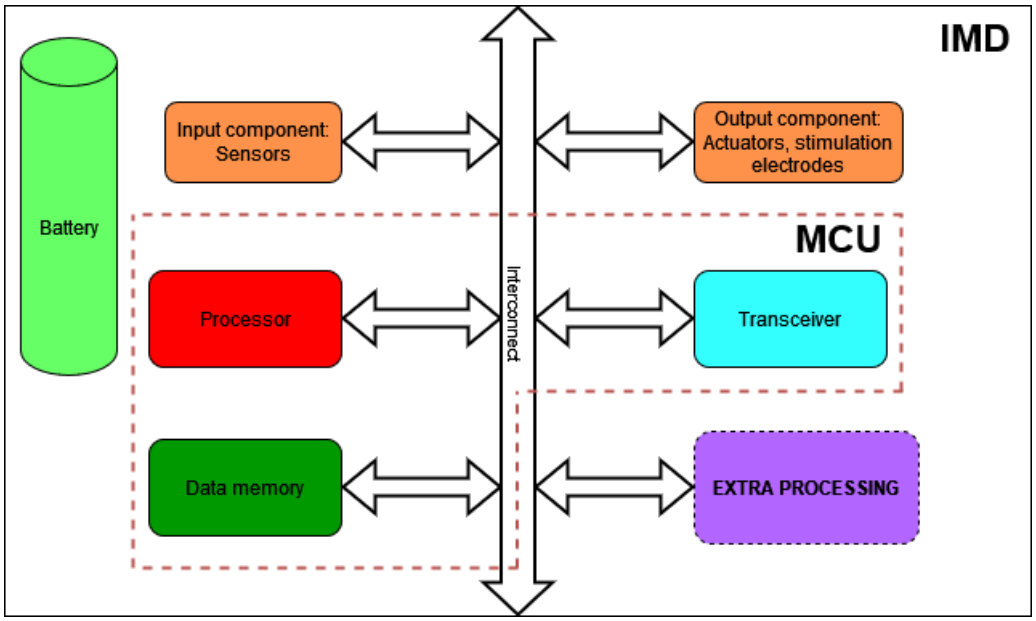


Figure 2.1: IMD layout, contains an MCU with attached sensors and actuators connected to Input/Output (I/O) ports of the MCU

In IMDs, modules are often employed for periodic activity. Sensors, actuators, and stimulation electrodes are the end modules in the system, with an MCU acting as the central unit that connects to data storage and transmission systems. These modules are crucial for the ability of the platform to interact with the body's physiological processes. The MCU is a compact device with a wide range of functionalities; further details about MCUs can be found in Section 2.2.1. The most important characteristics of an IMD are listed in Table 2.1 and are further elaborated upon in the following subsections.

Table 2.1: Characteristics from standard implementation of an IMD

Characteristics	Requirement
Real-time system	Tight deadlines
Battery-powered	Operate under a fixed energy budget
Biological environment	Upper-limited heat dissipation and device geometries
'Extra Processing'	Deliver desired new functionality with minimal energy, heat, and size increases

2.1.1 Real-time system

A crucial concept in modern systems is that of real-time systems. These are applications that must meet strict timing requirements to achieve their objectives. Real-time systems are commonly found in control systems, which are designed to regulate specific phenomena. For a control system to effectively manage a phenomenon, it must produce the correct output at precisely the right moment. If the output is delayed, it may fail to achieve the intended or necessary effect, or it may even be counter-productive. The effectiveness of a real-time system is measured by its ability to consistently meet deadlines. These deadlines, which repeat as long as the system is operational, will be defined in more detail in Section 2.1.3.

2.1.2 Battery-powered

A critical aspect of the platform is the battery. The choice of battery significantly impacts the overall lifespan, size, operating capabilities, and heat generation of the platform. Each of these factors plays a crucial role in the final design of the IMD. To ensure the IMD functions effectively, the battery must meet specific minimum requirements, as defined in [9]. Additionally, for use in an IMD, some options can enhance the performance of the platform i.e., rechargeable batteries combined with the ability to harvest energy while inside the body.

2.1.3 Biological environment

The environments in which an implant is deployed can vary significantly depending on its location within or near the body. A cochlear implant is typically situated near the brain and may be partially outside the body, while a pacemaker is often located near the heart. Each location imposes different requirements on the IMD. The terminology *environment* describes the function of the IMD. In this thesis, two types of environments are examined: cardiac and neural environments. These environments are used to define the deadlines discussed earlier in Section 2.1.1. The frequency for the cardiac deadline is 3 Hz [10], while the frequency for the neural deadline is 160 Hz [11]. The frequency $f[Hz]$ determines the time $t[s]$ allocated for each iteration of the algorithm, where $t = 1/f$.

2.1.4 Extra processing

Within an IMD platform, there is a processing space available for integrating additional functionality, typically separate from the MCU. This space can be used to house an accelerator, such as an Application Specific Integrated Circuit (ASIC) or a Field Programmable Gate Array (FPGA). An FPGA is particularly promising for this role because it combines the performance benefits of an ASIC with the flexibility of a CPU or GPU. An FPGA is flexible enough to be used for multiple implementations without having to alter the IMD, making it an excellent candidate for further research.

Size addition
The addition of an accelerator requires additional area to be used for the additional component. Space is a valuable resource on an IMD. Therefore an additional component requires a great deal of consideration, and subsequent research to determine if the pros outweigh the cons. The size expectation for the device is limited to comparable devices, as is referenced in Section 2.5.

Energy consumption
An additional component within an IMD will incur a penalty of energy consumption. Therefore, within the context of the research, considerations need to be made to accommodate the alternate operating way for the components in use.

Heat generation
With the previous consideration of energy consumption, additional energy usage will increase the heat generated by the device. Within the body heat dissipation is severely limited, making it a prime requirement to keep the patient healthy.

2.2 Processing elements

This section explores the different devices used as processor elements within an IMD for this thesis. The first consideration is the various possible implant versions, as outlined in Section 2.1. In the most relevant version of an IMD, the core component is a MCU, which incorporates a CPU. This processing unit is a highly flexible computer core capable of generating a wide range of outputs. The platform of the MCU includes standard computer components and offers the possibility of integrating additional components to that platform for dedicated and efficient processing—known as accelerators (see Section 2.1.4).

Table 2.2: Processing-element properties, defined according to 5 levels [very high, high, medium, low, very low]

Processor type	Flexibility	Speed	Power Consumption	Design speed	Parallelism
General Purpose Processors					
CPU	Very high	Very low	High	Medium/High	Low
GPU	High	Medium/High	High	Medium/High	Medium/High
Accelerators					
FPGA	High	Medium/High	Low/Medium	Low/Medium	Medium/High
ASIC	Very low	Very high	Very low	Very low	High

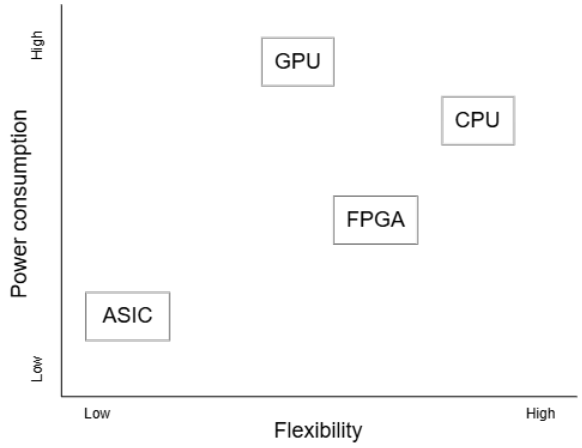


Figure 2.2: Comparison between the processor and accelerator options

In Table 2.2, some processor elements are denoted, used variants in the thesis and well-known types, the combination of these variants sketches where the options mentioned in this thesis lie. Figure 2.2 displays how the processor types, mentioned in the table, fall within two important metrics, power consumption and flexibility.

2.2.1 MCU

The MCU, is the platform on which all the hardware contained on a computer can be found. The crux of the MCU platform is that it is contained within a small area, thereby being a great choice for area-limited applications. As a result, an MCU is a great option for IMDs. Given their natural compatibility, an MCU is often the heart of an IMD, containing most of the management and functionality of the device.

2.2.2 CPU

The CPU is the processor available on an MCU, responsible for executing basic tasks and managing other components. It functions similarly to CPU in a computer, handling background processes and performing computations. For efficient MCU operation, the tasks performed on a CPU should be relatively simple and self-contained. The strength of a CPU lies in its versatility. While it may not be the fastest or most efficient option, it can perform a wide range of functions. This flexibility, and the fact that compilation of code is fast, is particularly valuable during the prototyping stage, where it allows for rapid development and testing. Once an algorithm is proven on the CPU, it can be optimized for specific hardware or software platforms to improve performance. However, this versatility comes at the cost of specialized performance. Compared to dedicated hardware, the CPU may not excel in certain tasks. Nevertheless, its general-purpose capabilities and reasonable cost make it a valuable component for many MCU and IMD applications.

2.2.3 FPGA

An FPGA is a chip that mimics an ASIC. An FPGA contains Look-Up Tables (LUTs) that are connected through programmable interconnects, as shown in Figure 2.3. The architecture of the FPGA is therefore able to perform different computations by using alternate configurations. Designing for an FPGA is also more time-consuming than for a CPU. This is partly due to the specialised hardware description language required and partly due to the nature of FPGAs. These factors make the iterative development process more challenging, as efficient changes often demand a deeper understanding of the underlying hardware. The resemblance of an FPGA to an ASIC offers significant benefits, particularly energy consumption, calculation speed, and, most importantly, the ability to perform concurrent operations.

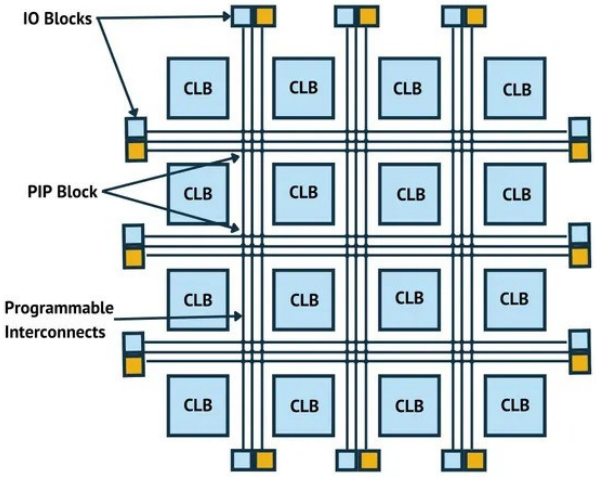


Figure 2.3: FPGA architecture, IO: Input/Output, PIP: Programmable Interconnect Point, CLB: Configurable Logic Blocks. [12]

Power consumption
Power consumption in an FPGA consists of static and dynamic consumption, which can be found according to the use of the components and the standard equations associated, as defined in [13] and [14].

$$P_{stat} = I_{stat} \times V_{DD} \tag{2.1}$$

where I_{stat} is the current that flows between the supply rails in the absence of switching activity and V_{DD} is the voltage on the drain.

$$P_{dyn} = \alpha \times C_L \times V_{DD}^2 \times f \tag{2.2}$$

where α is the switching activity, C_L is the capacitance of the load, V_{DD} is the voltage on the drain, and f is the operating frequency.

Flash-based FPGA
A specific subclass of FPGA types is the flash-based FPGA. A standard FPGA operates using Static Random Access Memory (SRAM) blocks that hold programmed values to make the hardware perform as intended. In contrast, flash-based FPGAs use flash memory to store these values instead of SRAM registers. This difference is significant because of the startup behaviour. SRAM blocks lose its values when the power is turned off, while flash memory retains the stored data even after power loss. This retention ability makes flash-based FPGAs advantageous in scenarios where quick startup is crucial, for instance, hardware is switched off to save power. However, the main drawbacks are that flash memory has a limited number of write cycles, and that FPGA technology has traditionally been centred around SRAM registers. As a result, utilising flash memory in FPGAs requires a different hardware structure than those designed for SRAM.

The difference between SRAM and flash memory also affects power dissipation. Flash memory cells retain data without power. In contrast, SRAM cells need continuous power to maintain data, resulting in higher

average power consumption during operation. Therefore, while flash cells consume more power during write operations, they offer lower average power consumption overall, especially in applications where data retention during power-off is critical [15] [16].

2.2.4 GPU

An GPU is a general-purpose processor, as is a CPU. The main difference is that a CPU is made to be able to perform a greater variety of instructions, which results in bigger size and additional overhead. An GPU instead is focused on performing a smaller amount of instructions, thereby retaining a reduced overhead but mainly having simple instructions. The reduced overhead gives the option to use multiple cores in parallel. The instructions are then able to be performed in parallel, thereby increasing the throughput. Using a greater amount of cores, however, can increases the power consumption and subsequent heat generation.

2.3 Algorithms

2.3.1 Medical algorithm

Medical therapy is continuously evolving, which in turn changes the requirements for workloads and the devices that handle them. As these requirements become increasingly demanding, current devices may eventually struggle to keep up. One of the most significant advancements in this field is the use of neural networks, which have high computational demands. To gain a comprehensive understanding, it is essential to test a combination of both newer high-requirement workloads and older use cases on various platforms.

Input data: ECG
The first data type is electrocardiogram (ECG) data, which refers to data generated through the use of Electrocardiography. Electrocardiography measures the electric field of a heart. Algorithms that use this data are often used in cardiac applications.

Input data: EEG
The second data type is electroencephalogram (EEG) data, which refers to data generated through the use of Electroencephalography. Electroencephalography measures local variables on multiple input locations, in which the differences between two points gives information about phenomena. Algorithms that use this data are often algorithms that search for complex connections in the region of interest. Due to the dense data streams in the brain, such an algorithm is used in neural applications.

Classification algorithms
While preliminary phases may reduce noise or modify the output to emphasise important parts of the signal, classification algorithms are fundamental to the practical use of medical data, and thereby IMDs. These algorithms convert recorded data into actionable values that can guide subsequent tasks, such as mitigating an epileptic attack. Traditionally, classification algorithms are based on rigidly defined rules that categorise outputs, for example, determining whether an epileptic attack is occurring by producing a *True* or *False* response. Although there is extensive knowledge about classification algorithms, the challenge in medical applications lies in the vast amount of input data, where only small, critical pieces, easy to misinterpret, contribute to the overall picture. The introduction of neural networks, as shown in Figure 2.4, has begun to address this issue. Neural networks can be trained on large, variable datasets to classify outputs more effectively than traditional methods. Although this technology is still being refined, it offers tremendous possibilities when compared to classical classification algorithms. The aim is not to rely solely on machine learning algorithms to complete the task but to use them to identify meaningful correlations in the data, thereby advancing medical care.

2.3.2 Encryption algorithm

Due to the placement of an IMD, wireless transmission is a requirement when communicating. Wireless communication needs to be secured. Although security features can be resource-intensive, they provide a crucial benefit to the overall security of the system. As a result, implementing security measures will be a key focus in the use of wireless communication, with multiple algorithms explored to create a comprehensive solution.

Hash type
The hash function uses a predefined function to convert the input into a different one, which can not be reverse-engineered normally. The one-way function is the basis of the security for the hash type, it serves as a check-sum for the total. Due to the added value, the authenticity of the message can be checked.

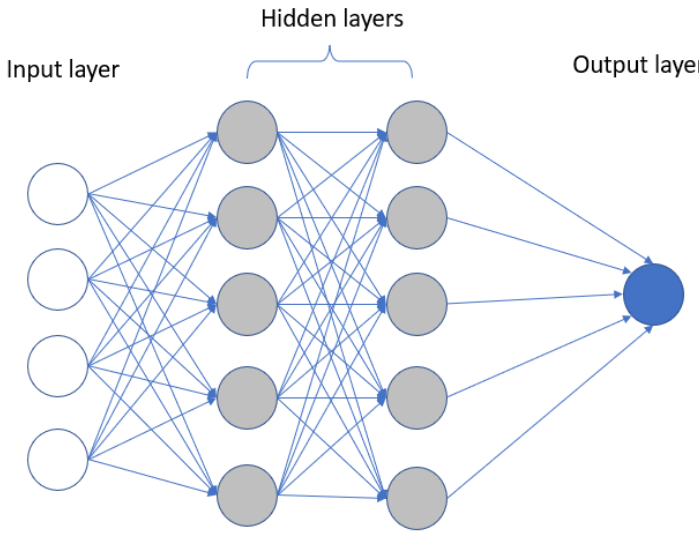


Figure 2.4: Neural network, this variant contains multiple layers. [17]

Cipher type
The cipher type alters the inputted plain text into the cipher text. The cipher text is obtained by encrypting the plain text with a key, whose general operation can be seen in Figure 2.5. The cipher shown uses a private key for encryption and decryption, making it a symmetric algorithm. The key used for the algorithm needs to be communicated or transferred before the start of the operation of the cipher algorithm.

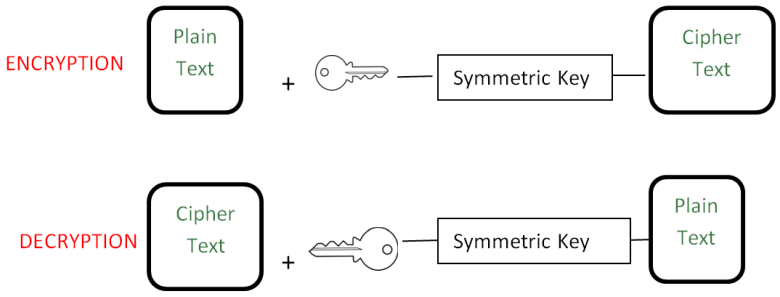


Figure 2.5: Symmetric encryption. [18]

2.4 Accelerator parameters

2.4.1 Reconfiguration

Reconfiguration is the term used to symbolise the alteration of the internal working of the logic level elements on an FPGA. The primary method of reconfiguration is by altering the internal connections and contents of LUTs which combine into a different functionality within the device. Reconfiguration is only used for FPGA.

2.4.2 Gating

Another option concerning saving power is gating, which includes clock gating and power gating. Gating involves placing a barrier between two parts of a device to reduce power consumption, as shown in Figure 2.6. These techniques are employed to minimise energy drain when components are not in use, in this case, processing devices. Clock gating generally works by removing the Clock (CLK) input to the subsequent part of the device, thereby eliminating dynamic power consumption in that section. This method does not require reprogramming or extended wake-up times. Oftentimes such a principle is implemented within the device itself, which is called sleep mode. A benefit of using sleep mode implemented on the device is that it requires no additional area and can benefit from additional power-saving options implemented on the device. Power

gating, on the other hand, involves cutting off the power supply to the subsequent part of the device. While power gating offers better static power efficiency, it comes at the cost of more intensive wake-up processes and the need for reprogramming for an FPGA. Additionally, it requires an additional device to shut off the power to the device. Therefore the area requirements and the wake-up time will increase. These devices also require additional power, but the amount of power is a fraction of the power used by the processing devices. Both types of gating require additional components to disconnect either a supply line or a signal line within the device, depending on the specific gating method employed. These gating components will be discussed in more detail in later chapters.

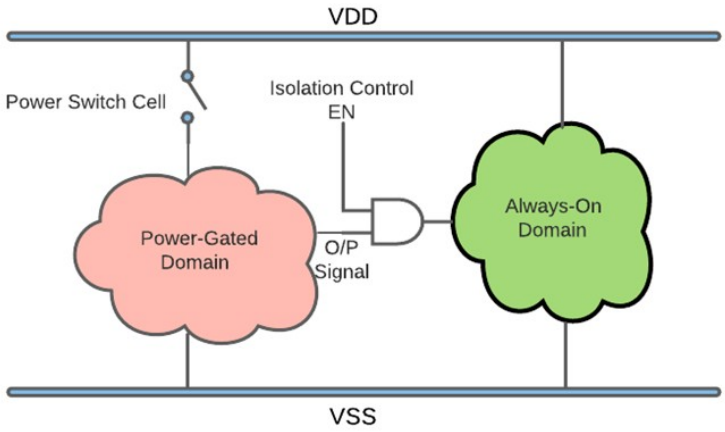


Figure 2.6: Figure showing the working of gating. [19]

2.4.3 Flash memory

Flash memory is non-volatile memory. Such a memory variant does not lose its contents when the power is turned off. The ability to retain data comes at the cost of speed or power used for such memory. The memory is required to save the required bitstreams to program the FPGA.

2.4.4 Cold wake-up

Cold wake-up refers to the time it takes a device to be used after powering it up. This type of wake-up, cold, is a primary concern due to the possibility of using power gating within this thesis. When using such a scenario the cold wake-up time needs to be checked. If it exceeds the point at which it can be ignored, it then needs to be accounted for in the assumptions made.

2.5 Related work

The related work in this section is focused on finding the answer to the question of the use of an FPGA in an IMD. The related works therefore focus on the use of FPGAs in an IMDs and ultra-low-power systems using FPGAs and CPUs.

2.5.1 Exploring FPGA use in medical section

Within the literature study, a previous thesis about the same subject was read and analysed. During finalisation of this work, new literature with similar topics was published. The publications were taken into account.

A separate thesis [8] has addressed the use of FPGAs in IMDs. At the time of that research, there was a lack of studies on next-generation workloads for use in an IMD, that were viable or accessible, which are the primary reason accelerators could be needed in IMDs. As a result, the previous thesis used alternative techniques to answer the question, particularly the use of embedded FPGAs (eFPGAs), which required a large amount of research. The results therefore were focused on a larger area.

Vaithianathan et al. (2024) [20], focuses on the identification of power-efficiency within wearable and implantable devices, thereby using FPGAs in a comparison to MCU and ASIC. The publication defines where it would be used, then how it would be used, the components and their benefits, and subsequently how the devices could be used in a more power efficient manner. Testing is mentioned, but the devices and values resulting from the measurements were not available in the publication.

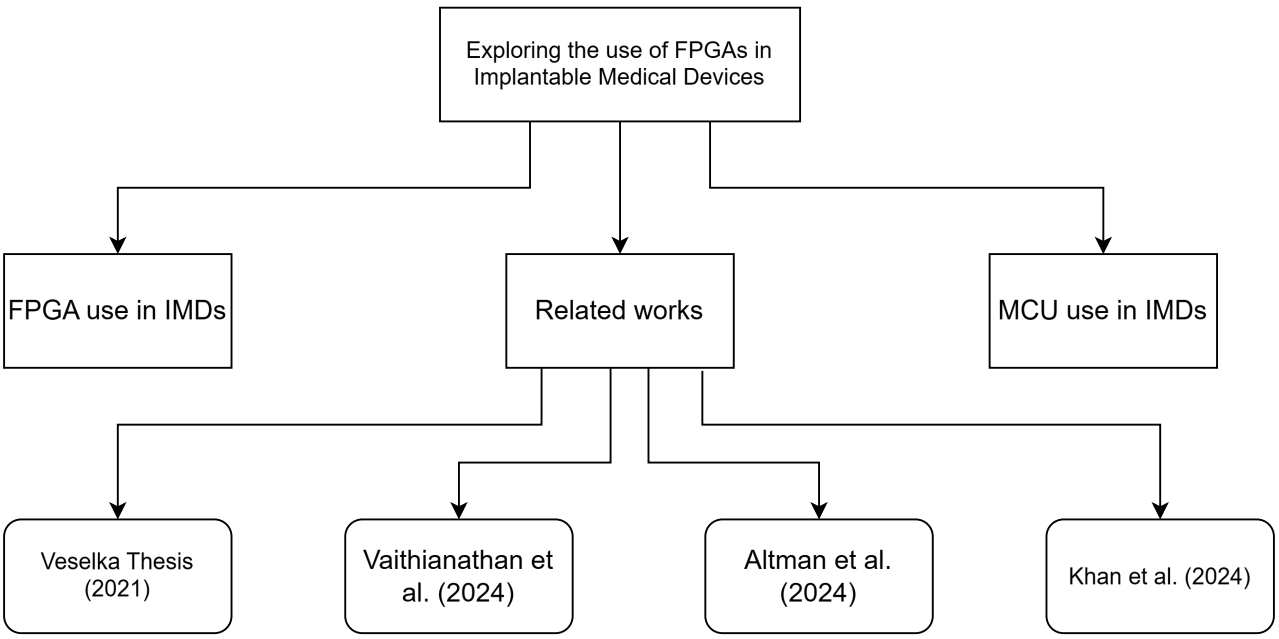


Figure 2.7: Taxonomy tree of related works for this thesis, Veselka Thesis: [8], Vaithianathan et al. (2024): [20], Altman et al. (2024): [21], Khan et al. (2024): [22]

Altman et al. (2024) [21], focuses on the use of FPGAs for machine learning in the field of biomedical engineering. It details an overview of the FPGAs used between 2001 and 2023 for the implementation of machine learning in biomedical applications. Given that this work is about next-generation workloads, this work combines a great deal of information with appropriate conclusions for efficient implementation. It is a review, not a practical application, therefore no new data is added in the form of device uses.

Khan et al. (2024) [22], focuses on the use of FPGAs for wearables. It details an overview of low-power FPGA design, specific families of FPGAs, focused on low-power applications, and future research venues, primarily, sleep modes, voltage scaling, partial reconfiguration, and flash/hybrid-based FPGA. It provides guidelines on the design of energy-efficient FPGA-based Wearable Medical Devices (WMDs). The target of this publication is closely-related to IMDs, therefore this is a valuable addition, it contains useful information for the design process on an IMD as well. It is a review, not a practical application, therefore no new data is added in the form of device uses.

2.5.2 MCUs used in IMDs

This section finds implementations of algorithm use on IMDs with the core component being an MCU. The goal is to find adequate devices that could be used in the exploration of next-generation algorithms and their effective use on IMD platforms. Table 2.3 summarises the devices found in literature, and their main factors, giving an overview to find devices from.

Table 2.3: MCUs used in literature, the entries are sorted by year and alphabetical afterward

Device	Vendor	Algorithm type	Total Power [mW]	Citation
EFM32 Leopard Gecko	Silicon Labs	Authentication system	0.002-20	[34]
MSP430FR5994	Texas Instruments	Early seizure detection	0.802	[35]
MSP430FR	Texas Instruments	Seizure detection	0.850	[36]
ATMEGA328PB	Microchip technology	Optical Wireless Communication	0.392	[37]
EFM32 Tiny Gecko	Silicon Labs	IMDfence	-	[38]
EFM32 Tiny Gecko	Silicon Labs	Cipher use	-	[39]
EFM32 Tiny Gecko	Silicon Labs	SecureEcho	-	[40]
FRDM-KL05Z	NXP	Spike detection for 128 channels	23.83	[32]

2.5.3 FPGAs used in IMDs

This section finds implementations of algorithm use on FPGAs with the core component being an MCU. The goal is to find adequate devices that could be used in the exploration of next-generation algorithms and their

effective use on IMD platforms. Table 2.4 summarises the devices found in literature, and their main factors, giving an overview to find devices from.

Table 2.4: FPGAs used in literature, the entries are sorted by year and alphabetical afterward

Device	Vendor	Algorithm type	Total Power [mW]	Citation
IGLOO AGL10	Microsemi	Wireless message management unit	0.101	[23]
IGLOO AGLN250	Microsemi	Neuroprocessor for 32 channels @ 25ksps	6.4	[24]
iCE40LP1K	Lattice	Sensor interface	0.12	[25]
XCS31400A	Xilinx	SHA-1 encryption	62.678	[26]
iCE40	Lattice	4-bitadder	0.11	[27]
IGLOO2	Microsemi	4-bitadder	0.5	[27]
IGLOO AGLN250V2	Microsemi	PID algorithm	0.168	[28]
IGLOO2 M2GL025	Microsemi	Neurostimulation	20.06	[29]
iCE40 HX-8K	Lattice	Neurostimulation	3.62	[29]
Cyclone V 5CEBA4	Intel	Neurostimulation	43.86	[29]
IGLOO AGL250V2	Microsemi	Seizure detection unit	0.0052	[30]
IGLOO AGLN250	Microsemi	ANNs for person identification	7.579	[31]
IGLOO M1AGL600	Microsemi	ANNs for person identification	15.261	[31]
IGLOO M1AGL1000	Microsemi	ANNs for person identification	30.213	[31]
iCE40UP5K	Lattice	ANNs for person identification	9.577	[31]
iCE40LP1K	Lattice	Spike detection for 128 channels	0.0374	[32]
Cyclone V	Intel	Tissue Stimulation	0.78	[33]

Chapter 3

Design methodology

This chapter defines the methodology used in the thesis. The chapter starts with an in-depth definition of the requirements for the experimental design. Then the definitions from the experimental design are used to find appropriate processing devices and algorithms. Next, the parameters and options pertaining to the alternative processing elements are discussed. Afterwards, the algorithms in use throughout the thesis are discussed. The next aspect to discuss is the measurement specifics for each device. To finish up the chapter, an analysis tool, made during and for the thesis, is discussed.

3.1 Experimental design

The research performed needs to be run systematically. Therefore, the following section contains the assumptions, the project flow and the general experiment to run to achieve the goal. An important aspect is the boundary conditions, otherwise named the assumptions and goals. The boundary conditions define where the research starts and ends, and this signifies when the research has been finished.

3.1.1 Vocabulary

The technical terminology used in the subsequent thesis is detailed in the following section.

Specialist
A *specialist* is to a doctor or other medical specialist. In the context of this thesis, such a person will primarily require data from an IMD to get the status of a patient's condition.

Patient
A *patient* refers to a person receiving treatment from the specialist. In the context of this thesis, such a person will carry the IMD.

Bedside reader or *doctor's programmer*
A *bedside reader* refers to a device that is situated close to the bed of a patient. The device then communicates with the implant on regular intervals to transfer the data collected by the IMD.

Workloads
Workloads will be used within this thesis to signify benchmark algorithms executed on the IMD.

Scenario (of work)
Scenario (of work) is used to signify the act of combining algorithms in a way to highlight a specific characteristic in the output. Another word that refers to this is a use-case.

'always on' algorithms
An *'always on' algorithm* is an algorithm that needs to be executed at all times. Such an algorithm is associated with an IMD since it contains algorithms that are often critical and always need to perform the required action.

'Off' time
'Off' time refers to power gating. To simulate power gating in the analysis tool, the workloads in use are alternated with 'off' time, thereby lowering the device's power usage to a level that would be equal to a

power-gated device.

'Sleep' time
'Sleep' time refers to clock gating. To simulate clock gating in the analysis tool, the workloads in use are alternated with 'sleep' time, thereby lowering the device's power usage to a level that would be equal to a clock-gated device.

3.1.2 Device choice

During the thesis, an additional component was to be selected in the form of an accelerator for next-generation workloads. Given that neural networks are among the new implementations, they add additional requirements. Neural networks are always updated with new techniques and training data [21], and therefore are updated often. Inflexible solutions therefore are at a disadvantage. Another aspect is the increase in computations, a neural network contains a great deal of Multiply and Accumulate (MAC) operations. Parallel execution is a great option, due to the implementation of neural networks, thereby greatly favouring a non singular device, capable of parallel execution.

There were a few options to check, an ASIC, an FPGA and a GPU. An ASIC device to use in this context is not a viable addition. An ASIC is a great option regarding power efficiency, but the rigidity makes it unsuited for this task. FPGAs are often used for prototyping ASIC devices. The test cycle is significantly faster and less costly. Given the use of ASIC implementations in IMD, this devices follows as a logic option for efficiency and parallel utilisation. An GPU implementation on the other hand can be a good addition. They have great parallel capacity, making a good option for an efficient and quick neural network.

The resulting choice was to compare a flexible, yet power efficient processing devices. Therefore the FPGA component was tested as an additional component on an IMD, apart from the standard of a MCU equipped with a low-power CPU. The reasoning was to be more efficient with contained compute power for new iterations of algorithms, while maintaining a good battery lifetime for the total device. For this purpose an FPGA was selected.

3.1.3 Assumptions

The following assumptions are made for the experimental design:

- *Data transfer*
  For the input conditions regarding data transfer, values are taken from a paper [38]. The values (and small variations) will be used as leading values for the subsequent results. These are an effective rate of 265kb/s and 400kb/s actual data rate. The size of the data transfer changes depending on the user accessing the device. The specialist has a transfer duration of 2 minutes, while the patient will have a transfer duration of 30 seconds.

- *Environment*
  The possible environments for the implant used within this thesis are cardiac and neural. The choice of environment impacts the type of workloads employed and therefore impacts the area used, the performance and the energy consumption. The cardiac environment uses less compute intensive algorithms.

- *Duty cycle*
  An IMD is not always executing the algorithms, which would entail that for short amounts of time, the device can go to a sleep state. Therefore a duty cycle needs to be defined to specify how long the device is in active and sleep mode.

- *Device usage*
  The memory functionality of the MCU is used at any time, due to the data that is being saved on the IMD. Therefore the static usage of an MCU platform is included for every implementation of a scenario.

- *External device*
  Data transfer is handled through a bedside reader or doctor's programmer. The device handles connections to the IMD and can save data in a different location. Such a setup removes the possibility of data being overwritten due to memory limitations.

- *Reconfiguration costs*
  The reconfiguration costs are costs for each separate startup of a workload, and should therefore be added to those workloads.

- *Measurements*
  The measurements are done across the dynamic parts of the workload execution on the devices to simplify obtaining results and subsequent comparison. Latent variables are taken into account in the analysis stage of the thesis.

- *Flash-based FPGA*
  To use a flash-based FPGA as a comparable device, a major assumption was made, considering potential future implementations. The assumption is that the used values for the flash-based FPGA correspond to the non-flash-based FPGA, the ICE40. Therefore the flash FPGA retains its programming during wake-up, and the active power usage values of the iCE40 are combined with the sleep mode values of the flash-based FPGA. This assumption implies that the FPGA does not lose its previous programming and can operate on a duty cycle over short intervals. For instance, it could be powered off for 50 ms and resume operation afterwards as if no interruption occurred.

3.1.4 Scenario definition

In the exploration process, a consistent format was required to denote combinations of algorithms. As defined in the vocabulary, this will be represented by scenarios. Figure 3.1 shows the connections with the different components in use, which are then used to sketch the operation on the IMD to make scenarios.

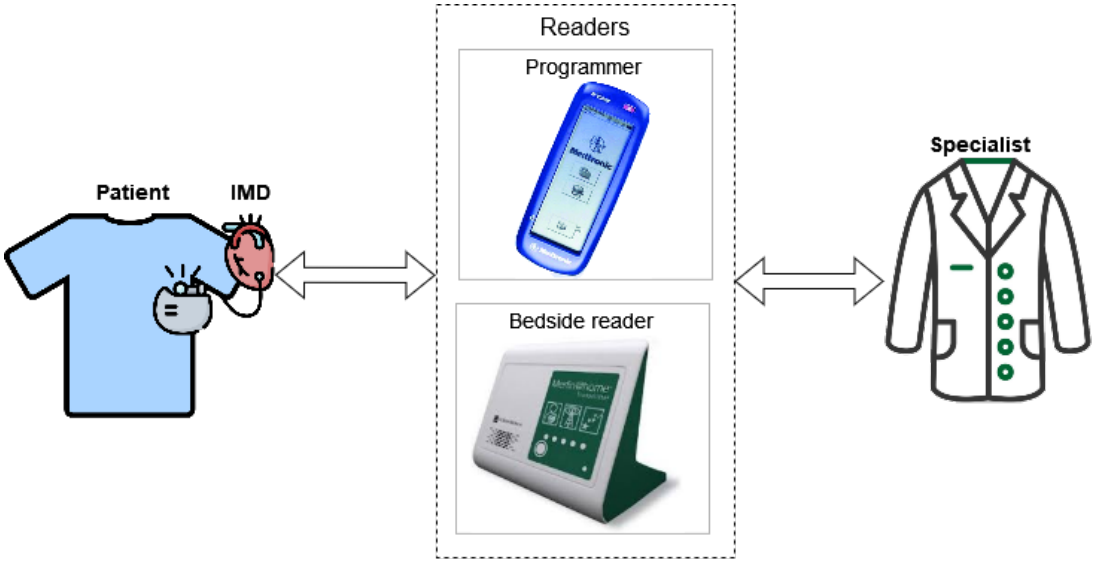


Figure 3.1: Definition of components within scenarios [38] [41]

An example of a scenario is provided below. The duty cycle is defined as a percentage of activity, which refers to the percentage of time the algorithm is used.

These scenarios are designed to showcase particular aspects, like device characteristics, or generic real-life situations. These aspects are utilised in benchmarks and operational scenarios.

- There is a patient with an IMD, a specialist, and a bedside reader.
- The IMD is accessed through a bedside reader, gathering data for the specialist and the patient.
- The IMD runs [ALGORITHM] with 5% active duty cycle.
- The bedside reader accesses the IMD [AMOUNT] times a week for [DURATION] at a time.
- [CLOCK/POWER] gating is used while the algorithm is not in use.

Measurements
The first stage in the result stage was to see how the algorithms compare based on the platform on which they are ran. The algorithms used in the thesis are defined in Section 3.3.

Benchmarks
The scenarios discussed in this section are designed to focus on individual aspects of device performance.

Each scenario is constructed to test a specific characteristic in isolation.

First benchmark This benchmark tests the impact of using medical workloads in their intended use, creating a more complex and diverse scenario. It is designed to highlight the flexibility of the device in handling multiple tasks simultaneously.

Second benchmark This benchmark examines the addition of a new type of algorithm, encryption. The encryption is added, with the medical workloads still operating in their regular context. The addition introduces additional complexity and helps assess the device's ability to handle more demanding tasks.

Third benchmark This benchmark introduces sleep time or off time into the scenario to evaluate how these factors influence performance. This addition simulates clock or power gating and measures the impact on device efficiency.

Fourth benchmark This benchmark follows the same structure as the third benchmark, but instead of using the regular FPGA, it uses the device parameters from the flash-based FPGA. This alteration shows how such a device would compare.

Table 3.1 provides a comprehensive list of the specifics for each benchmark.

Table 3.1: Definitions for the benchmarks (CSD: Current-Source Density), all workloads in use are based on integer values, for the use of encryption, there are two moments of communication, one weekly & one daily, spike sorting contains the spike detection and classification algorithms.

Benchmark index	Medical algorithms	Wireless algorithms	Duty cycle
Benchmark 1: Using medical workloads			
1.1	Spike sorting	-	-
1.2	CSD	-	-
Benchmark 2: Medical workloads with encryption			
2.1	Spike detection and classification	AES	-
2.2	CSD	AES	-
Benchmark 3: Medical workloads with gating			
3.1	Spike sorting & Sleep	-	5% active
3.2	CSD & Sleep	-	5% active
3.3	Spike sorting & Off	-	5% active
3.4	CSD & Off	-	5% active
Benchmark 4: Benchmark 3 with flash-FPGA used			
4.1	Spike sorting & Sleep	-	5% active
4.2	CSD & Sleep	-	5% active

Operational scenarios

Figure 3.2 presents the different variations in a tree pattern, branching into distinct scenarios. Each new leaf represents a different pathway, connected to a characteristic that defines the key attributes for that particular scenario. The various options exist for specific reasons. The environment defines a deadline, which, in turn, determines how much time is allocated to the algorithm. This deadline acts as the hard maximum for each iteration of the algorithm to complete. An important note is that gating is not used within the neural environment. The reason for this absence is that the duty cycle is defined as 100% active mode. Therefore, the assumption is made that the device will never be in sleep or off mode. Table 3.2 contains the specifics required to define the scenarios.

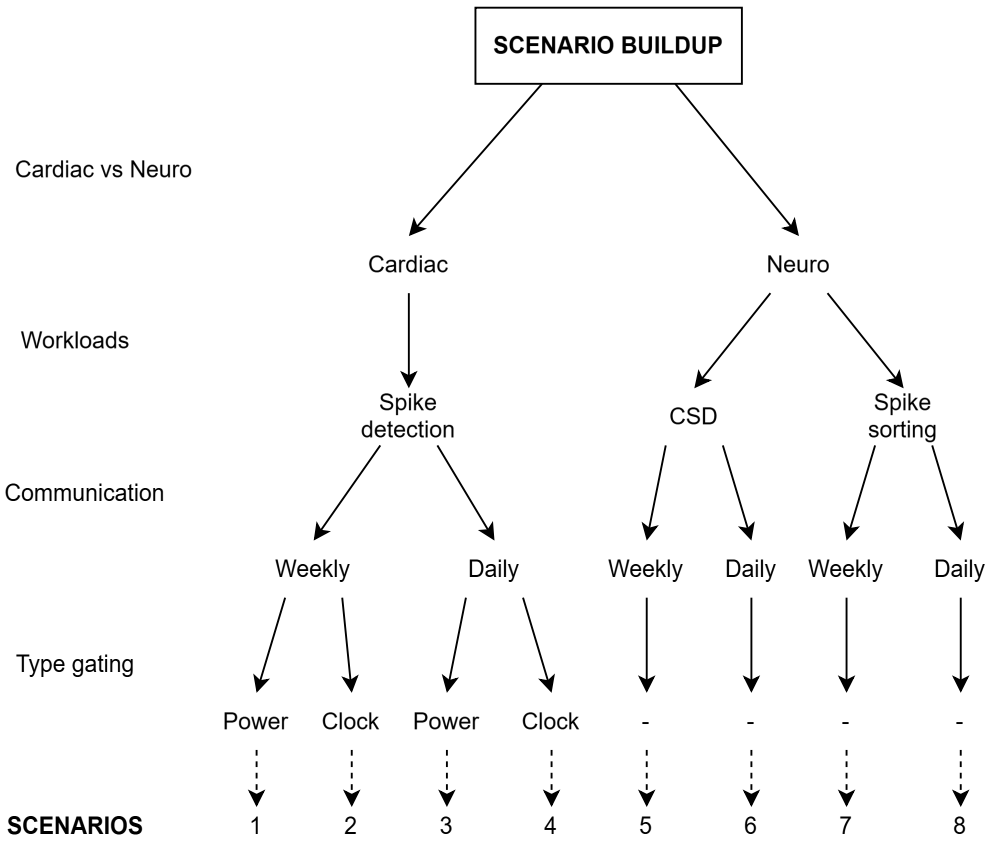


Figure 3.2: Tree displaying operational scenario structure. The duty cycle definition is 5% and 100% for *Cardiac* and *Neural* respectively, defined for the active operation of the processing device.

Table 3.2: Operational scenario definitions, the differences between scenarios are shown through the variables (CSD: Current-Source Density), all workloads in use are based on integer values, spike sorting contains the spike detection and classification algorithms.

Scenarios	*Environment*	*Duty cycle*	*Workloads*	*Communication*	*Gating*
1	Cardiac	5%	Spike detection	Weekly	Clock
2	Cardiac	5%	Spike detection	Weekly	Power
3	Cardiac	5%	Spike detection	Daily	Clock
4	Cardiac	5%	Spike detection	Daily	Power
5	Neural	100%	CSD	Weekly	-
6	Neural	100%	CSD	Daily	-
7	Neural	100%	Spike sorting	Weekly	-
8	Neural	100%	Spike sorting	Daily	-

3.1.5 Flow within the project

The following section will be concerning the flow of the thesis project. The goal is to give a visualization to understand how the results are achieved.
The start of the project is to define the required components, e.g. the building blocks, such as the algorithms. These components are then carefully analysed and updated according to their uses, e.g. the algorithms are implemented on different devices. Then they are used to generate results, e.g. measurements are done. Then an analysis is performed on the results, to fulfil the thesis goal.

During the project, a continuous rehash of previous results was required and increased in the duration of the project significantly. The loop that displays this phenomenon can be seen in Figure 3.3.

3.2 Processing elements

A crucial starting point of the method is deciding which processing devices to use to execute the algorithms, defined in Section 3.3. This decision is vital to addressing the research question effectively. To ensure valid

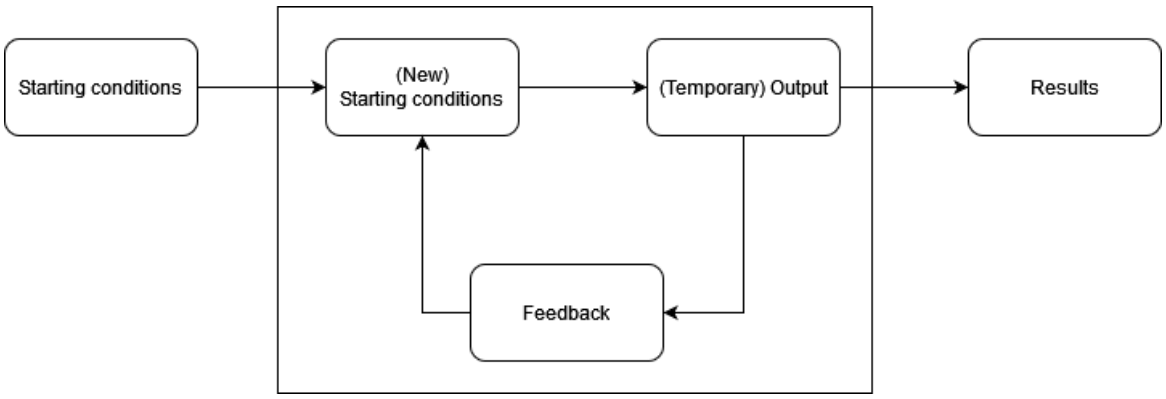


Figure 3.3: Flow within the project to find results

outputs, it is essential to select appropriate devices with great care. The selection should include the two basic cases: the currently used MCU platforms and the potential replacement, FPGA devices, as additions. Additionally, other devices may be required to address specific aspects, such as memory management. The chosen devices will form the foundation for conducting the research in this thesis. In Section 2.5, lists of devices used in IMDs were defined, with the vendors of which will be used to select appropriate devices.

This section will cover the devices used to acquire the measurements. The primary factors in selecting the specific devices for the thesis were their suitability and their availability. The suitability of these devices is focused on low-power applications with an appropriate compute throughput. The availability is focused on price and off-the-shelf devices.

3.2.1 MCUs

The first device type to examine is the MCU, which, as noted in Section 2.2, often serves as the processing unit of an IMD, in the form of a CPU. MCUs are intriguing devices because they encompass all the components necessary to be considered a microcomputer. They include a combination of static and dynamic memory, a CPU, interconnects, and peripherals, offering a versatile platform for experimentation. As such, MCUs play an integral role in this investigation. Within the MCU platform, certain models are specifically designed for low-power applications. These are of particular interest given the limited battery capacity and charging constraints in IMDs.

For the choice of devices in use for the thesis, publications were researched, which can be seen in Section 2.5.2. From this research the vendors for the devices were selected, being Silicon Labs [42] and [**noauthor_ti_nodate**], and within their assortment devices were selected based on the CPU and peripherals available on the platform. Within publications, the used CPU is often a ARM Cortex-M0, therefore a similar CPU was selected. The Gecko devices from Silicon Labs in particular contain a hard-Intelectual Property (IP) block for the Advanced Encryption Standard (AES) algorithm. The resulting devices were the Tiny Gecko [54] and the Giant Gecko [55]. The Tiny Gecko contains a ARM Cortex-M0+ CPU. The main difference with the Giant Gecko is that the Giant Gecko contains a stronger CPU, the ARM Cortex-M4. This difference in CPU will show how a stronger processing device will impact the IMD platform. Table 3.3 summarises important characteristics of the MCU devices used in the thesis.

Table 3.3: MCU characteristics

Device name	Frequency range [MHz]	Input voltage	Area characteristics
Tiny Gecko 11	1, 2, 4, 7, 13, 16, 19, 26, 32, 38, 48	3.3 V	CPU, RAM, Memory, UART, AES-HW ASIC
Giant Gecko 11	1, 2, 4, 7, 13, 16, 19, 26, 32, 38, 48, 56, 64, 72	3.3 V	CPU, RAM, Memory, UART, AES-HW ASIC

Gecko device
The Gecko devices have multiple different modes, ranging from an active mode to a deep-sleep mode. The EM4 mode is used for these devices, the deep-sleep mode. Deep-sleep mode will have certain parts of the device that need to wake up to move to active mode, which is considered instantaneous.

3.2.2 FPGAs

The accelerator option for running algorithms is the FPGA. One of the most intriguing aspects of an FPGA is its ability to get reconfigured. This capability is achieved through a combination of compiled software, from Verilog and VHDL sources, and a hardware implementation involving LUTs and programmable interconnects. With this setup, an FPGA can effectively simulate the behaviour of an ASIC, offering significant advantages in power consumption and throughput compared to general-purpose processing units, as discussed in Section 2.2.

For the choice of devices in use for the thesis, publications were researched, which can be seen in Section 2.5.3. From this research, vendors were selected to find appropriate devices for low-power applications. For SRAM-based FPGAs, the iCE40 series from Lattice [43] and the Cyclone V series from Intel [**noauthor_cyclone_nodate**] seemed promising. For flash-based FPGAs, the IGLOO series from Microchip Technology [44] seemed promising. Devices were subsequently selected based on the requirements for the size and low static-power drain. The resulting device which was selected was the *iCE40UP5K* [45]. The flash-based FPGA gives important low-power improvements, and therefore a suitable variant was selected based on equivalent flip-flops, IO banks, available RAM, and die size. The resulting device was the *Nano IGLOO AGL250* [56]. Table 3.4 summarises important characteristics of the FPGA devices used in the thesis.

Table 3.4: FPGA characteristics

Device name	Frequency range [MHz]	Input voltage	Area characteristics
iCE40UP5K	0.75, 1.5, 3, 6, 12, 24, 48	1.2 V	LUT, Flip-Flop, EBR, SPRAM, DSP, PLL, Oscillators
Nano IGLOO AGL250	0.75 - 250	1.2 V	System Gates, D-Flip-Flop, SPRAM, FlashROM, PLL, Oscillator

Reconfiguration on iCE40
The ability of an FPGA to be reconfigured is a major component of the strength it possesses. The reconfiguration alters the functionality it possesses, thereby becoming versatile. To alter this aspect, for a normal FPGA, a process is started, which first wipes the current value of the configuration, and afterwards loads a new one. This is a sequential process through a single data line, which puts in the values one by one to the required components.

For the device in use during the thesis, the iCE40, the reconfiguration parameters are used. These are a cycle count, additional bits and time and a operating frequency, which combine into a reconfiguration time [45].

$$t_{reconfig} = (n_{bits} + n_{additional\_bits}) * 1/f_{operating} + t_{additional} \tag{3.1}$$

With Equation 3.1, and values of 833288 bits, 157 additional bits, 0.001202 s additional time and a frequency of 19 MHz, the time for one reconfiguration cycle comes to 70.7 ms. All the values in question are found in the data sheet, under which the frequency which is the standard operating frequency in use on the device. The power consumption during reconfiguration for the iCE40 device is sourced from a well-documented page [46]. The plots for the booting show the expected procedure, delete current configuration, wait a small number of cycles, program the SRAM, and subsequently wait some cycles afterwards.

Flash-based FPGA
An additional aspect to consider is a flash-based FPGA, this gives a lower static power usage. The characteristics of such a device can be found in Section 2.2.3. To enable a proper comparison between normal and flash-based FPGAs, a device with characteristics similar to regular FPGA, the *iCE40UP5K* is identified. A suitable comparison is the *Nano IGLOO AGL250*. The Nano IGLOO consumes 20 μA or 34 μA in flash*freeze mode, depending on whether the voltage is 1.2 V or 1.5 V, respectively. Aside from static power usage, the same values measured for the iCE40 will be used, as the iCE40 is readily available, while the IGLOO Nano is not. An alternative option is to use the sleep mode of the iCE40, but in this mode, nearly all functionality and programming are disabled. Therefore, the sleep mode of the iCE40 was deemed unsuited for the task.

Power gating
When power gating is used in a later scenario, the appropriate idea is that a device is used which would be a combination of the Nano and the iCE40. The device would use flash memory for the ability to keep the configuration on the device when shut off, which would entail that the configuration would be active without requiring a reprogramming cycle after wake up.

3.2.3 Battery

For usage within this thesis, a battery is selected to power the IMD platform for an extended amount of time. The options for commensurate IMD batteries come from companies which have been making batteries for IMDs for years. The companies in question are EAGLEPICHER [38] and Resonetics [47]. The battery capacity used in the models is 1500 mAh, which is a typical capacity for modern IMDs [48] [49]. The batteries have a large battery capacity, and have a large area to match.

3.2.4 Flash memory

Flash memory is used to store FPGA reconfiguration bitstreams, which can be loaded directly into the FPGA from the memory. The specific flash memory device used in this thesis is the MX25V1006F [50], and it will be utilized in the computations going forward. The decision to use this flash device is based on its use of NOR flash technology. Unlike NAND flash, which employs a NAND logic gate, NOR flash uses a NOR logic gate to store data. NOR flash is better suited for read operations due to its faster read speed and lower standby current. Flash memory has a limited number of write operations. For the MX25V1006F, the minimum guaranteed is 100,000 erase/program cycles. Additionally, while read operations typically have fewer limitations, they are subject to a phenomenon known as "read disturb" [51], where frequent reads can alter surrounding data. This effect generally occurs after around 100,000 read cycles between erase/program cycles. However, it is more commonly associated with NAND flash, whereas the chosen device is a NOR flash, which is less susceptible to this issue.

3.2.5 Gating device

Due to the usage of power gating in this thesis, a way to cut off the power needs to be done smoothly. The variant in use for the thesis is the Ultra-Low Leakage Load Switch TPS22916B [52]. This device was chosen due to the low static power it uses, which was the primary requirement.

3.2.6 Multi-meter

For the measurement of analog voltage lines, Fluke Multi-meter 115 was used [53].

3.2.7 Resulting processing devices

Within the previous sections, several devices were selected, of which a few were processing elements. The actual devices and their serial numbers are important to cross reference, and therefore are defined in Table 3.5 .

Table 3.5: Basic device characteristics

Device name	Device type	Brand name	Serial Number of device	Citation
Tiny Gecko 11	MCU	Silicon Labs	EFM32TG11B520F128GM80	[54]
Giant Gecko 11	MCU	Silicon Labs	EFM32GG11B820F2048GL192	[55]
iCE40UP5K	FPGA	Lattice Semi	iCE40UP5K-B-EVN	[45]
Nano IGLOO AGL250	Flash-FPGA	MicroChip	AGL250-VQ100	[56]

3.3 Algorithms

A key aspect of ensuring the validity of this research is selecting algorithms that can be applied in appropriate contexts. Therefore, algorithms have been chosen based on their potential use in neural or cardiac IMDs. As mentioned before, there are two primary categories: wireless communication and medical therapy. These cases were selected based on future requirements for IMDs, including wireless accessibility and advancements in medical treatments. Each primary use case can be further divided into sub-cases, allowing for a more detailed evaluation of IMD platforms.

The algorithms are required to be representative of the environment in which to use an IMD, thereby establishing typical use cases for an IMD FPGA that aligns with the functions of IMDs within the cardiac or

neurophysiological domain. The resulting themes to look for are wireless transmissions, and medical algorithms, ranging from basic to next-generation workloads. The first wireless algorithm is the current standard for encryption, called AES. Then for the given options of the platform in question lightweight alternatives for were found [8]. These options are PHOTON as a lightweight hashing option and SIMON/SPECK as a lightweight cipher option. Subsequently, the medical algorithms were more difficult to find. The requirements from the thesis goal require future options for algorithms, which quickly makes them shift into neural network territory. Therefore a neural network with low computational intensity was required. Recently an algorithm was developed that was suitable for this thesis, the spike-sorting algorithm [11]. Another algorithm to use as reference was the Current-Source Density (CSD) algorithm, which is a conversion algorithm for Local Field Potential (LFP) input data into a CSD representation [57] [58]. This algorithm is a less compute-intensive algorithm when compared to the spike-sorting algorithm. The combination of these algorithms represents the algorithms in use for such devices. Table 3.6 summarises the algorithms used within the thesis. Within the table, the sampling frequency used for the measurements can be found.

Table 3.6: Algorithm characteristics with definitions for the measurements

Algorithm name	Algorithm type	Environment	Sampling frequency [Hz]
Medical algorithms			
Spike detector	Spike-sorting algorithm sub part	Neural & Cardiac	24414
Spike classifier	Spike-sorting algorithm sub part	Neural	1000
CSD	LFP to CSD algorithm	Neural & Cardiac	24000
Wireless algorithms			
AES	Block cipher	-	10, 100, 1000
SIMON/SPECK	Block cipher	-	10, 100, 1000
PHOTON	Hash function	-	10, 100, 1000

3.3.1 Medical algorithms

Medical workloads are much broader in scope, as it contains any algorithm performed on the device to facilitate data analysis, management, or simple data collection from sensors.

Examples of next-generation workloads often include increased hardware resource demands. These advancements provide greater functionality, thereby having a more substantial impact on healthcare. The following section will outline the specific workloads that will be utilized and tested in this thesis. To begin, a neural network based algorithm is selected to represent up-and-coming workloads. The algorithm chosen is spike sorting [11], which consists of two phases to select and analyse incoming data related to neural spikes. To balance this, a less compute-intensive workload is also included. For this, a simple algorithm with low implementation costs is chosen: CSD [57].

Spike sorting
Spike sorting is an algorithm used to identify behaviour within neurons. For the implementation used in this thesis, spike sorting is used to identify simple and complex spikes within Purkinje neural data.
The algorithms contained within spike sorting are the spike detection and spike classification algorithms. These two algorithms differ significantly in their functionality. The spike detection algorithm applies filters and basic operations to perform an initial screening of the incoming data, with some modifications to prepare the data for classification. It eventually identifies whether a spike has occurred. Once a spike is detected, the spike classification algorithm is triggered, which utilises a neural network to predict whether the detected spike is simple or complex.

One notable aspect of this combination is that the spike detection algorithm should run continuously in the background, even while the classification is being performed [11]. In the context of this thesis, the significance of this workload lies in its ability to be implemented on both an FPGA and an MCU, as well as its incorporation of a neural network.

Spike detection
The spike detection algorithm operates by receiving input from a sensor connected to a Purkinje cell. The input is then processed through multiple stages, including filtering, data alterations, and detection of spikes. The detected spike can either be simple or complex. The output consists of 40 subsequent samples, each comprising 8 Most Significant Bits (MSBs) of the original 10 bits received from the sensor.

Due to its reliance on filters and basic operations, the spike detection algorithm is well-suited for implementation on an FPGA. However, challenges may arise in achieving the desired accuracy, particularly when dealing with floating-point inputs, which are more difficult to emulate on an FPGA.

Spike classification
The spike classification algorithm operates by using the output from the spike detection algorithm and processing that data through a neural network to determine whether a simple or complex spike has occurred in the Purkinje cell.

The neural network in use for this algorithm defines a great deal about the operation. The currently used neural network contains 5 layers, 5 convolution layers, with a final activation function. Table 3.7 displays the structure of the neural network used for the spike classification.

Table 3.7: Neural network layer contents

Layer	Type	Neurons	Weight
1	Convolution	16	40
2	Convolution	7	16
3	Convolution	5	7
4	Convolution	4	5
5	Convolution	3	4

The neural network used in the spike classification algorithm relies heavily on MAC operations. MAC operations are well-suited for parallel implementation, making them highly compatible with FPGA-based concurrency. However, challenges may arise with the implementation of floating-point operations. While a fixed-point implementation of the neural network is possible, it is more difficult to achieve both accuracy and general functionality.

CSD
CSD has a long history in the medical field. Its principle involves converting LFP data into a more interpretable form, allowing for better analysis and deduction. LFP data is collected using multiple sensors placed at fixed distances from each other in different locations in the brain, often arranged on a single plane. To utilize CSD for uncovering deeper connections in the data, a specific method is required. LFP data points are obtained by providing a stimulus to the subject, which triggers brain activity. This data is then converted to CSD, a form better suited for analysing neural connections. The resulting activity is correlated with the stimulus to derive meaningful insights. The method of converting LFP to CSD has been instrumental in revealing important neuronal mechanisms [57].

The CSD algorithm used in this thesis is based on a variant available in an online MATLAB file [58]. The equation for the CSD algorithm is shown in Equation 3.2.

$$Y[i-1] = -A[i-1] + 2 \cdot A[i] - A[i+1],\ i \in \{1, n+1\} \tag{3.2}$$

The input data is denoted as A, and the output data as Y, A and Y are arrays with size n and $n-2$, respectively. The index i represents the location in the input/output data array. The n variable is the number of electrodes used as input for the algorithm The number of electrodes used in the testing is 12. The amount used was taken to be equal to the amount used during real data capture.

The CSD algorithm performs a simple operation involving several additions and multiplications, closely resembling a MAC operation. Due to its simplicity, the algorithm is easy to implement on any type of hardware. Moreover, as the number of inputs increases, the workload becomes more suitable for accelerators, thanks to the potential for parallelism.

3.3.2 Wireless algorithms

In today's world, privacy is a critical concern, highlighting the need to protect personal data. The algorithms discussed in this section are prime examples of addressing this need. By integrating security primitives into the data-transmission pipeline, privacy can be better protected without requiring extensive computational resources. Combining multiple security primitives can yield even stronger results.

The first workload type involves security primitives, with the AES serving as a prime example. The purpose of these workloads is to secure wireless transmissions between the device and external parties, such as medical professionals. The importance of this workload lies in safeguarding the private data of the patient. The security algorithms used in this workload type include AES, SIMON/SPECK, and PHOTON.

The most critical aspect of this workload type for use in IMDs is speed, as it involves communication and requires real-time data processing. This requirement can present challenges for slower or less efficient devices. However, due to the nature of the algorithm, the workload may benefit from concurrent processing.

Block ciphers
The principle of a block cipher is to transform input text into cipher text, which can then be decoded using a specific method that requires a dedicated key for both encoding and decoding. This transformation ensures that only the party with the correct key can decipher and read the content. The block cipher workloads chosen for implementation in this thesis are SIMON and AES [59] [60]. SIMON and SPECK were selected for its lightweight implementation, while AES has been a widely accepted standard for many years.

AES
The AES algorithm is an algorithm that was selected as the 'best' algorithm during a contest, thereby gaining the privilege of becoming the defacto standard for secure communications and thereby being named Advanced Encryption Standard. The original name was the Rijndael cipher, named after the two creators, Joan Daemen and Vincent Rijmen. AES is a symmetric-key algorithm, meaning that the key used for encryption and decryption is the same key. AES performs 10, 12 or 14 rounds based on the size of the cipher key, 128, 196 or 256 respectively. For each round AES uses a 128-bit round key, made through key expansion according to the AES key schedule. The key is subsequently used to transform the input text. The algorithm first uses a substitution box (implemented as a look-up table) as a first step. The substitution box is used to obscure the relation between the plaintext and the ciphertext. Then in subsequent steps the text is transformed further before starting the next round.

The AES algorithm is a heavy algorithm for the devices used in this thesis. It contains quite a few steps of complex computations and therefore will take some time to actively run. Due to the status of the algorithm, there are hardware additions to standard platforms for efficiency purposes, such as can be found on the *Gecko* devices. This already existing hardware addition within the original MCU devices underlines the high suitability for relocation of this algorithm onto more efficient devices.

SIMON and SPECK
SIMON and SPECK is a family of lightweight block ciphers. SIMON/SPECK is a add–rotate–XOR (ARX) cipher, which means that their round function includes: (A) modular addition, (R) rotation with fixed rotation amounts, and (X) XOR. The ARX cipher setup is popular because they are fast and can run on cheap hardware. Apart from being fast, the algorithm also runs symmetrically amongst all internal functions, protecting against timing attacks. It is important to note that SIMON is an implementation that is optimised for hardware, while SPECK is an implementation that is optimised for software [59]. The SPECK algorithm is a relatively cheap algorithm to run, therefore being a good choice for running on IMDs.

Hash functions
A cryptographic hash function processes input text, in the form of plaintext, and generates an output of fixed length, serving as a unique identifier for the input text. The workload chosen for the hash implementations is PHOTON [61], selected for its lightweight implementation, which is well-suited for use in hardware within an IMD.

3.4 Measurements

For a proper comparison, workloads must be measured using the same metrics and under identical input conditions. While actual measurements can be conducted on different setups, any changes to the setups introduce variability. To maintain consistency, throughput will be the primary requirement for comparison. The workloads will be executed at an operating frequency, but for comparison purposes, the goal is to standardise the throughput. Therefore, the operating frequency is matched to the required throughput for each algorithm, as defined in Section 3.4.1. A combination of metrics will be used, with particular emphasis on accurately measuring power consumption. Consistent power measurements require careful analysis of the usage of the device, which presents a challenge due to differences in device types and execution methods. These differences also affect the accuracy of the measurements.

3.4.1 Throughput frequency

The frequency used in the measurements is critically important. The standard approach for comparing devices and their results is by referencing their operating frequency. However, the frequency directly influences the overall operation of the device, including its power consumption. Instead of standardising the frequency across all devices, the frequency is based on the rate at which output is transmitted. This rate, in turn, determines the amount of output being processed. By using this as a fixed factor, execution time is excluded from consideration. While execution time is typically a key metric, the choice was made to focus on average power consumption as the primary contributing factor in this analysis.

The calculation of the appropriate operating frequency uses the sampling frequency and the cycle count for one iteration of the algorithm. Multiplying the sampling frequency by the cycle count yields the operating frequency required for that specific algorithm on the given device. The formula for this calculation is shown in Equation 3.3.

$$f_{operating} = f_{sampling} * n_{cycles} \tag{3.3}$$

An important point to consider is that energy consumption is not always linearly dependent on frequency, as can be seen in Equation 2.2. In some cases, instead of reducing the operating frequency, it may be more efficient to increase the processing speed, complete the computations more quickly, and then shut down the device. This approach could potentially optimise energy efficiency.

3.4.2 Device understanding for measurements

A thorough understanding of the devices and their inner workings is essential for obtaining accurate measurements. This knowledge ensures that measurements are not unduly influenced by unknown characteristics in the setup of the devices. If any setup-related errors do occur, they can be identified and accounted for in the analysis.

3.4.3 Setting up the measurements

The approach used for the measurement is to define the operating conditions based on the sampling rate rather than standardising the operating frequency across all devices, as outlined in Section 3.4.1. For each device, the cycle count for one iteration was considered. Since devices differ in their characteristics, computations had to be made for each algorithm on each device and then converted for the appropriate frequencies. Afterwards, the most important consideration is the distinction between static and dynamic-power consumption. For these measurements, dynamic-power usage was prioritised, as focusing on dynamic power simplifies the process and the conditions that need to be accounted for. Static-power consumption is determined through the device's data sheets.

3.5 Analysis tool

The measurements need to be thoroughly analysed, and for this purpose, a specialised software tool was developed during the thesis. This analysis tool is particularly useful due to the processing of large amounts of data. The nature of the question revolves around different types of processing elements and algorithms, which need slightly differing follow up analysis. The tool helps to keep a consistent analysis throughout the whole process. It was developed in Python, due to the ease of development, large amount of libraries and the fact that it is becoming a standard tool within the TU Delft. A visualisation of the components of the tool can be seen in Figure 3.4.

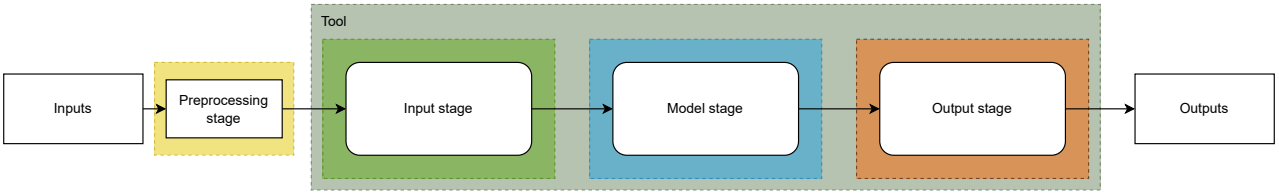


Figure 3.4: Overview of the analysis tool, which shows the separate stages used

3.5.1 Target of tool

The goal of this tool is to generate output that addresses the original objectives outlined in Section 1.2. To ensure consistent development, each component of the tool was rigorously tested for proper functionality, and new features were added as needed. The main challenges that the tool helps combat are:

- Ease of producing results for new scenarios
- Produce plots showing the efficiency of devices in defined scenarios
- Keep consistency between device types

Therefore the design of this tool was focused on flexibility. The tool has the ability to adapt to scenarios that are either not yet fully defined or that evolve as new data and results are accumulated.

3.5.2 Input

The starting conditions for the development cycle were based on the completion of the previous part of the thesis, which involved gathering measurements as outlined in Section 3.4. The inputs to the tool can be classified as:

- Dynamic variables
  The dynamic variables are the variables that come from the execution of the algorithms on the device, such as time and power.
- Static variables
  The static variables for each device originate from sleep mode and additional devices. An example of an additional device for this context is the power-gating device. The standard operating costs and additional devices are added to dynamic costs in the tool.
- Overhead variables
  Overhead variables for the device are additional considerations. An example here is the reconfiguration; it is a device-specific consideration and does not fall under static or dynamic.
- Scenario variability
  For each scenario, a certain amount of options can be used. The amount of variation needs to be large enough to provide valuable insights while being contained enough to be able to be used in practice in both development and usage afterwards.

A significant challenge was implementing variability between platforms, including considerations like sleep mode power consumption and reconfiguration on an FPGA. The tool details will be provided in Section 4.3.

Chapter 4

Implementation

The goal of this chapter is to describe the fully realised versions of the algorithms on each device, the measurement process, and the implementation of the analysis tool. Once these aspects produce the desired outputs (i.e. finishing sub-goals to the point of producing results), the implementation phase of this thesis can be considered complete.

4.1 Measurements

The following section will cover the implementation factors for the measurements. The setup details and input conditions can be found in Section 3.4. It is important to note that each platform and brand has a distinct design philosophy and implementation, making direct comparisons between platforms a challenging task. An approach for handling these differences was outlined in Section 3.4.3. Table 4.1 defines the specific type of measurements used for each device.

Table 4.1: Measurement methods for devices

Device name	Device type	Measurement
Tiny Gecko 11	MCU	Simplicity Studio - measuring through internal sensors
Giant Gecko 11	MCU	Simplicity Studio - measuring through internal sensors
iCE40UP5K	FPGA	Multimeter - using test points on the device

4.1.1 Simplicity Studios measurement

The Gecko device types are operated and programmed using a software platform known as Simplicity Studio [62]. This program, in combination with the device, provides built-in current and power measurement capabilities. Notably, Simplicity Studio also offers features for monitoring device outputs during operation. These built-in current and power measurement options were utilised during the measurement process for the Gecko devices. Figure 4.1 displays the results of using Simplicity Studios to perform measurements on the Gecko devices.

4.1.2 Multimeter

For measurements on devices without built-in capabilities, a multimeter is used, which is connected to Testpoint (TP)s on the device. The test points are connected to specific voltage lines, which either transmit data or supply power to the device.

The device in use without build-in capabilities is the iCE40 device. Measurements on the iCE40 can therefore be taken from the development platform using a multimeter. For this study, the primary focus is on measuring the voltage across the core. I_{CC} can be measured across the series resistor R76 1 Ω at TPs 11 and 12 [45], as can be seen in Figure 4.2.

4.1.3 Comparable measurements

To effectively compare measurements from different devices, it is essential to maintain consistency in the assumptions used to obtain those measurements. The prior assumptions were outlined in Section 3.4.3, and

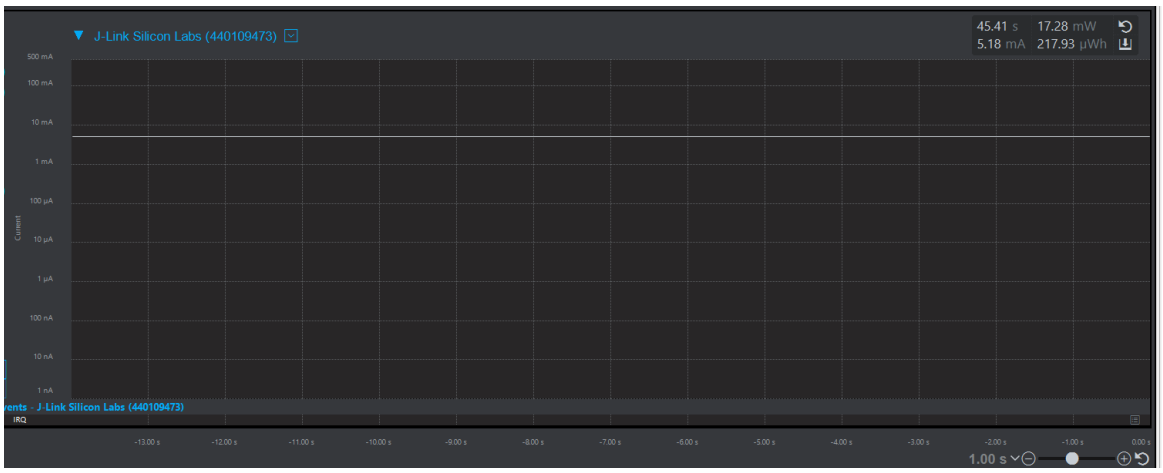


Figure 4.1: The tool used for the measurements on the *Gecko* devices, Simplicity Studios [62]

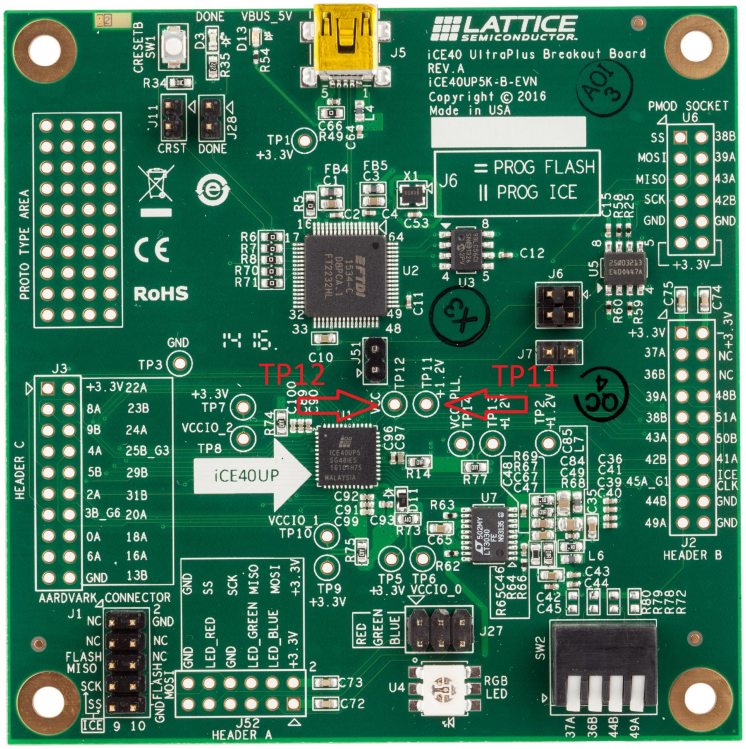


Figure 4.2: Printed Circuit Board (PCB) with the test points for the measurement on the *iCE40* device

now that the measurements have been collected, the comparison phase begins. For the comparison, a plethora of metrics are available. Within these metric, the primary one to consider is power, which can be derived by converting both current and voltage measurements into a common form. Since power is comparable across platforms, this conversion satisfies the requirements for cross-device comparison. An equally important metric is the energy, which will be used after the measurement stage. The execution time is normally taken as a big component, but due to the implementation factors considered, the execution time should be about equal between devices, and therefore be of limited importance. It will be displayed for the measurements, but further on be discarded.

4.2 Algorithm implementation on devices

This section introduces the initial stages of algorithm implementation on the devices. Based on the setup information and assumptions outlined in Section 3.2.

4.2.1 Algorithm design for devices

The algorithm design went through three iterations. It began with basic versions of the algorithms in their primary forms, essentially just running the code with minimal modifications. This initial version consisted of either found code, modified to be executable, or a basic algorithm implemented quickly. Due to the multiple platforms involved, this phase included implementations in both C and Verilog/VHDL. Once these initial versions were deemed functional, the next step was to optimise the algorithms for the selected devices. This optimisation introduced several challenges, primarily due to the varying design philosophies across different manufacturers, platforms, and device types. Once the algorithms were made sufficiently efficient, the final challenge emerged. The final challenge was ensuring consistency in the comparison between algorithms on different processing elements. Given the aforementioned differences between the processing elements, selecting a base to compare them on was difficult. A compromise was eventually reached by using the sampling frequency as a common guideline (Table 4.2), as outlined in Section 3.4.1.

4.2.2 Data transfer

For data transfer between a device and an end user, the primary concerns are the amount of data, transfer speed, and security requirements. The values for data amount and speed were sourced from [38], which provided data for weekly and daily transfers in various sizes and durations. An alternative approach would be to define transfer values for general use and specific algorithms, which would require input from a domain expert to obtain standard usage information. For the purposes of this thesis, the decision was made to use one daily and one weekly transfer definition. The daily transfer occurs once per day, while the weekly transfer occurs once per week. The daily transfer is defined as a half-minute process during which the device transmits basic information, such as battery status or a time diagram of the patient's medical data. The weekly transfer is a two-minute process used by a specialist to download all available data from the device, which is then analysed to provide the patient with detailed health insights. The resulting values used can be seen in Table 4.2, together with the citation where the algorithm originated.

Table 4.2: Sampling frequencies for algorithms

Algorithm	Sampling frequency [Hz]	Source
CSD	24000	[58] [57]
Spike detector	24414	[11]
Spike classifier	1000	[11]
Security	Variable	[38]

4.2.3 Implementation on Geckos

In this section, we will focus on algorithms designed for the Gecko devices, specifically the Tiny Gecko and the Giant Gecko. For every algorithm on display, a value was required to calculate an appropriate sampling frequency. To find the cycle count an implementation was required to transfer the values to the user. Therefore additional implementations were made.

CSD
The algorithm receives input data from multiple electrodes, processes the data using the CSD equation, and outputs the resulting value.

The original CSD algorithm was written in MATLAB [58]. The primary change involved converting the MATLAB code into C code for the MCU. The functionality of the original code was retained, with a small alteration to the computation used for the array. The alteration makes use of more basic operations.

Spike detection
The algorithm processes input data, detects whether a spike has occurred, filters the data, and passes relevant portions to the spike classification algorithm.

The spike detection algorithm was originally implemented on the Gecko devices in [11]. The primary adjustment made was altering the operating frequency to match the calculation's sampling frequency. This variant functioned as expected after the adjustment.

Spike classification
The spike classification algorithm receives filtered input from the spike detection algorithm and uses a neural network to determine whether the detected spike is simple or complex.

The spike classification algorithm was originally implemented in [11]. As with the spike detection algorithm, the key adjustment was modifying the operating frequency based on the sampling frequency. However, a challenge arose with the Tiny Gecko device due to compatibility issues with the TensorFlow Lite library [63]. While the library could be compiled and run on the device, it encountered operational issues that affected performance. Therefore the performance of the Tiny Gecko device was worse than would fall in line with the expectations based on the Giant Gecko.

AES
AES is classified as a cipher-based encryption algorithm.

Due to time constraints, the implementation could not be made on the Gecko device. The values used in the analysis were sourced from values from data sheets for the Gecko devices [54] [55]. The operating frequency of the Gecko devices was 19MHz.

SIMON/SPECK
SIMON/SPECK is a lightweight encryption algorithm in the cipher category.

Due to time constraints, the implementation could not be made on the Gecko device. The values used in the analysis were sourced from values from data sheets for the Gecko devices [54] [55]. The operating frequency of the Gecko devices was 19MHz.

PHOTON
PHOTON is a lightweight encryption algorithm in the hash category.

Due to time constraints, the implementation could not be made on the Gecko device. The values used in the analysis were sourced from values from data sheets for the Gecko devices [54] [55]. The operating frequency of the Gecko devices was 19MHz.

4.2.4 Implementation on iCE40UP5K

This section will discuss the implementation of algorithms for the iCE40UP5K device.

CSD
The major modification was converting the MATLAB code into VHDL for implementation on the FPGA. This conversion process was time-intensive due to the specific requirements for an efficient hardware implementation of the algorithm. A continuous problem regarding the FPGA implementations remains the amount of I/O ports. The amount was insufficient to implement a proper parallelise implementation.

Spike detection
During testing, it was discovered that certain aspects of the code were inefficient and prevented successful synthesis on the device. These inefficient components were partially removed, allowing the full code to be synthesised. Additional insights into the algorithm's functionality were also gained, and minor adjustments were made to improve efficiency. A continuous problem regarding the FPGA implementations remains the amount of I/O ports. The amount was insufficient to implement a proper parallelise implementation.

Spike classification
Early testing revealed that the original form of the code could not be accommodated on the limited size of the FPGA. To address this, pipelining was introduced. Pipelining restructured the execution of the neural network by allowing multiple layers of neurons to be processed sequentially, sharing components and reducing area usage, while increasing time linearly. A continuous problem regarding the FPGA implementations remains the amount of I/O ports. The amount was insufficient to implement a proper parallelised implementation.

Additionally, within each neuron's execution, the weights of the neural network were optimised to improve the computational efficiency. The alteration stemmed from altering values, internal values from the neural network, making them solvable with a bit shift instead of a multiplication. Ultimately, for the original code, a conversion program was made to allow for an automatic conversion of alternate versions of the neural network into VHDL code for this purpose. This adaption makes it suitable for quicker prototyping and implementing for FPGA or ASIC.

AES
The AES algorithm used was sourced from [8]. The primary modification involved addressing issues with the communication to the MCU, which consisted of UART or AHB-Lite. The communication was replaced with a simpler input and save mechanism. Other minor edits were made for consistency between algorithms.

SIMON/SPECK
The SIMON/SPECK algorithm also originated from [8]. Similar to AES, the primary modification involved addressing issues with the communication to the MCU. The communication was replaced with a simpler input and save mechanism. Other minor edits were made for consistency between algorithms.

PHOTON
The PHOTON algorithm, like AES and SIMON/SPECK, came from [8]. The same modification was made to the communication to the MCU, replacing it with a simple input and save function. Additional changes were made for consistency, these were minor edits.

4.3 Analysis tool

This section will cover the development of the tool, including its design specifications, internal workflow, and the accuracy of its outputs. The setup information and input conditions can be found in Section 3.5.

4.3.1 Options

Based on the previously defined goals of the tool, a subset of specific options can be identified. These options will address potential corner cases and provide additional features for the tool to utilize. A significant challenge to consider is that each new option adds complexity to the tool's operation.

4.3.2 Supplying input to tool

The input primarily consists of Comma Separated Values (CSV) files, many of which are automatically generated from previous data, although new data can easily be added. The input files are device-specific and are categorised into three types: measurements, cycles, and devices. The measurement files contain data related to performance measurements, such as power consumption, the cycle files provide information on the number of cycles required for one iteration of an algorithm, and the device files store information about the hardware on which the algorithms are executed. These files are consolidated into an Excel file, which is then used to perform calculations within the tool.

4.3.3 Overview of internal function

The global tool architecture is shown in the Figure 4.3. This overview displays how the stages are connected. Figures 4.4 till 4.7 contain the architecture of the separate components within the tool. Every stage shows what data is required, and what steps are taken to give an output.

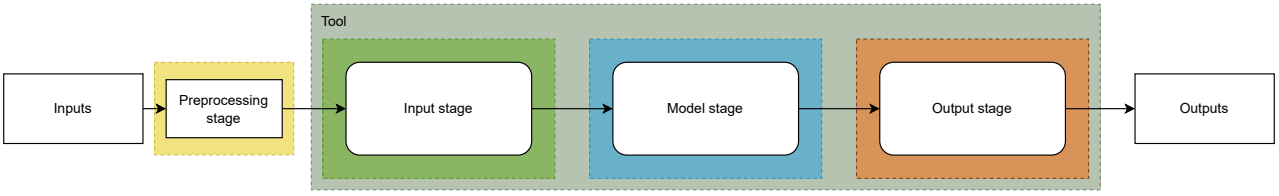


Figure 4.3: Overview of the analysis tool, which shows the separate stages used

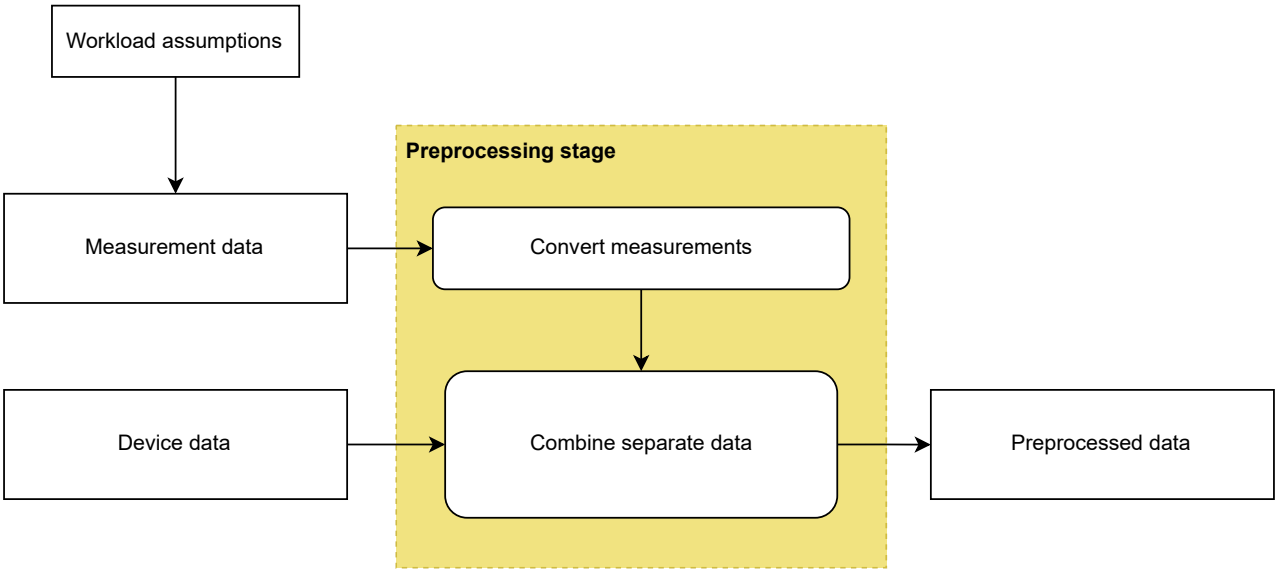


Figure 4.4: The preprocessing stage of the tool

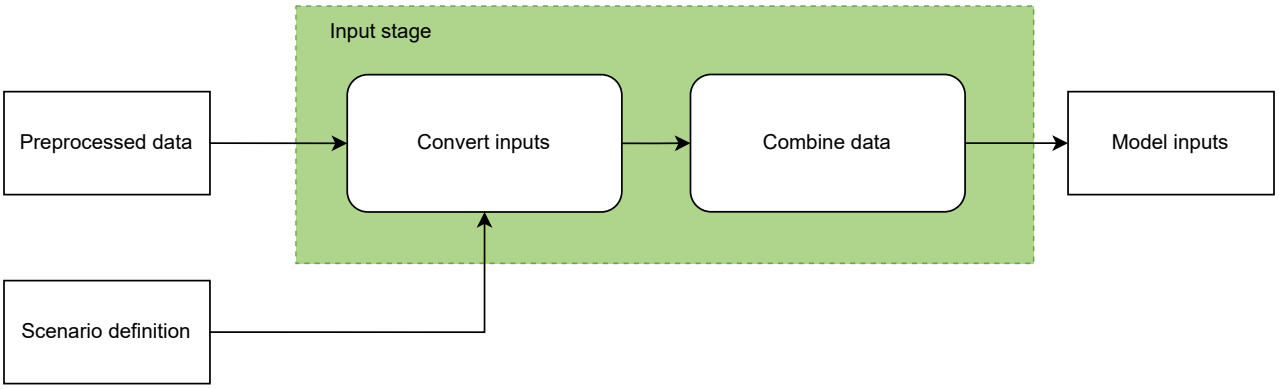


Figure 4.5: The input stage of the tool

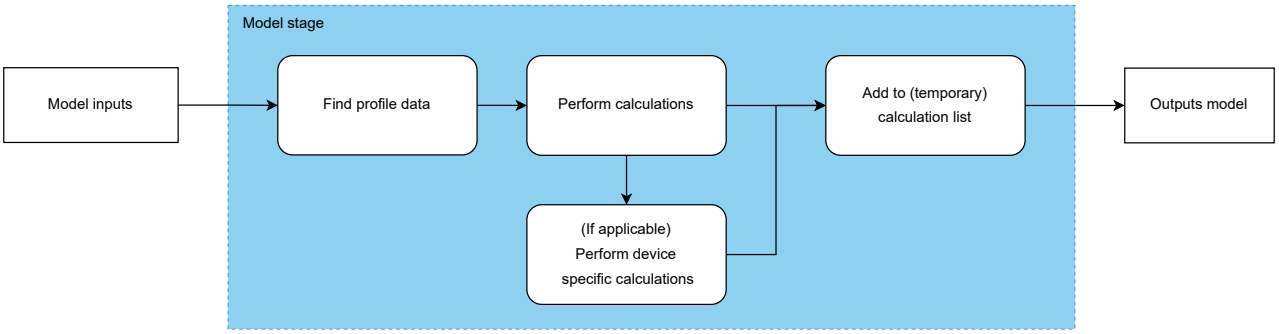


Figure 4.6: The model stage of the tool

4.3.4 Functionality

The primary functionality of the tool is to combine measurements and data in a way that reflects realistic scenarios. To achieve this, it is essential to understand device operations across all possible and required operating conditions. These conditions include the maximum operating capacity of the algorithm on the device, the minimum operating capacity under similar contexts, and characteristics concerning wake-up times.

The primary input to the analysis tool is the measurements, which will be referred to as a *workload* variable within the analysis tool. These variables contain the results of the direct execution of the algorithms. Within the analysis tool, additional variables are required for a realistic scenario. These additional variables will be referred to as *overhead* variables within the analysis tool. These will contain characteristics concerning reconfiguration, sleep mode and waking up from power down.

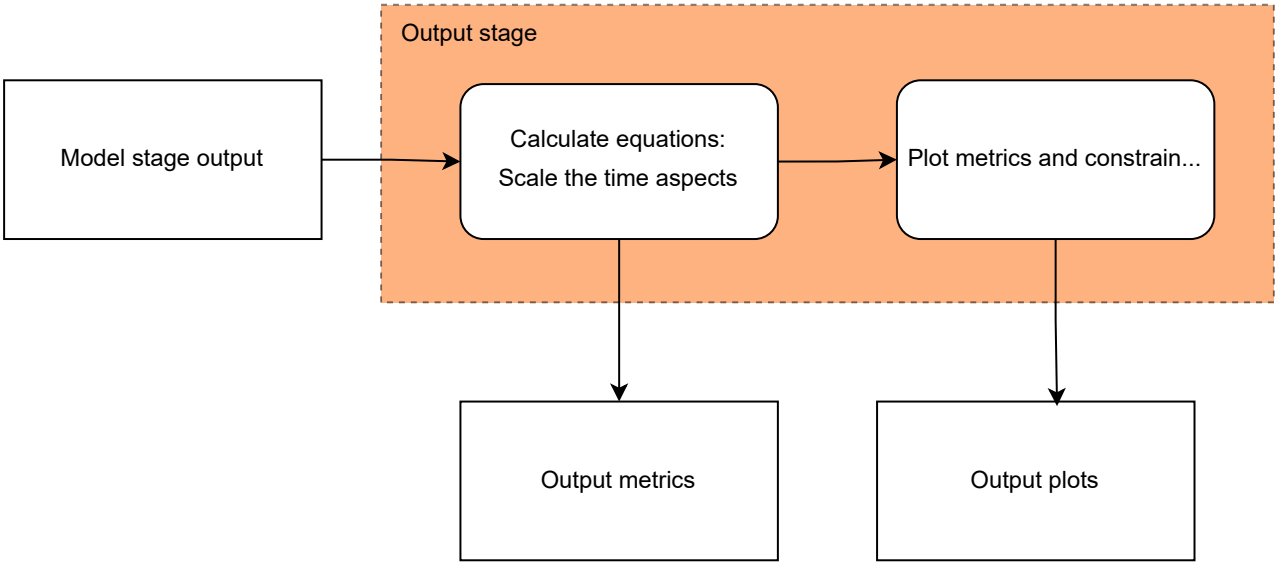


Figure 4.7: The output stage of the tool

4.3.5 Equations

The tool utilises various types of equations, each corresponding to different metrics. These include time-dependent, power-dependent, and resource-dependent equations. In each calculation, the metric used is referred to as a *unit*. Once the unit is defined, the output value is calculated for the plethora of options within the scenario in use. Every calculation is done according to the same equations.

The first equation is the summation type; this equation defines a basic addition to combine separate components to find the total value.

$$unit_{total_n} = unit_{overhead_n} + unit_{workload_n} \tag{4.1}$$

with n being the combination of devices used for this calculation.

The second equation is the averaging type; a multiplication and afterwards a summation is used to combine separate values to find the total value.

$$unit_{total_n} = unit_{overhead_n} * time_{overhead_n} + unit_{workload_n} * time_{workload_n} \tag{4.2}$$

with n being the combination of devices used for this calculation.

4.3.6 Outcome

Equations 4.1 and 4.2 are used to find the outcome of the metrics in use for the supplied inputs to the tool. The supplied inputs to the tool define a single scenario, with multiple devices that are used internally in the scenario. The output of the equation is then the result for a single scenario, a single device type, and a single metric. This can then be further used in an analysis, where the results are combined within a scenario, or between scenarios.

Chapter 5

Experimental results

In this chapter, the evaluation results from our experiments are collected and presented, focusing on metrics such as power consumption and execution time.

5.1 Analysis tool validation

The primary validation involved performing manual calculations for specific scenarios and comparing them with the tool's outputs. This straightforward process was applied to all required aspects and values to ensure consistency.

5.1.1 Validation steps

The validation process follows a set of simple steps:

1. Use the values directly from the scenario definition in the tool.
2. Input the selected values in the *tool calculations* section.
3. Perform the same equations manually and input them in the *manual calculations* section.
4. Compare the results from the manual calculations with those from the tool. Both should match; if not, any discrepancies must be investigated and corrected.

The values in use should adhere to the same metrics.

5.1.2 Scenario validation

The validation steps were carried out on a selection of scenarios, one benchmarking and one realistic scenario. The validation results are presented for various metrics, such as execution time, average power, and battery life. The benchmarking scenario will be shown to show how the validation was performed.

However, during the initial validation process, several issues within the tool were identified and subsequently resolved. The more complex cases of the tool relied on the validated scenarios as a foundation. At the end, the values from the tool and the validation matched, and the validation demonstrated that the tool worked as intended. The process of finding errors in the results of the tool also showed the significance of using the validation process.

5.2 Measurement results

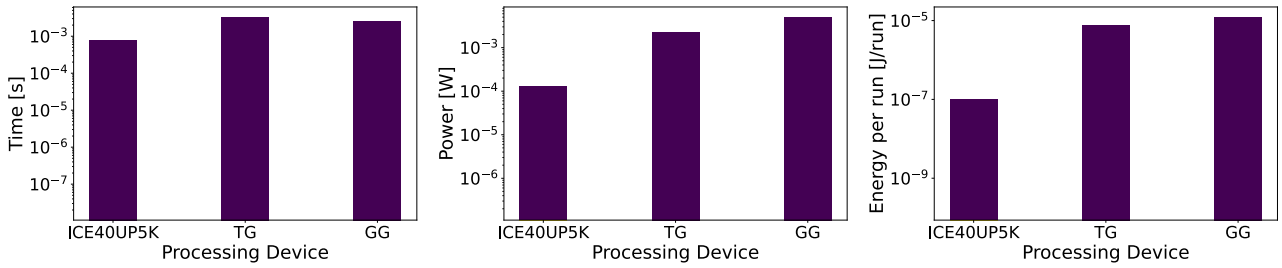
This section presents individual measurements of algorithms executed across different devices, those being MCU only or FPGA+MCU. While this section is not directly critical to answering the problem statement, it offers valuable insight into the performance factors of different algorithms. Later, these individual algorithm results can be used to understand scenario results, providing a more comprehensive argument for the final conclusion.

These algorithms are shown in this section: AES, PHOTON, SIMON, CSD, spike detector and spike classifier.

Figures 5.1 through 5.7 display the differences between various hardware setups for the same workload. Each figure illustrates different metrics, such as execution time, average power consumption, and total energy. Execution time is used to verify whether the original throughput assumptions hold, average power consumption is used to make a comparison between workloads, and the energy consumption signifies the impact that the workload will have on the fixed IMD battery capacity.

5.2.1 AES

The results of the AES workload are shown in Figure 5.1. These are Software (SW)-only results for the Geckos.

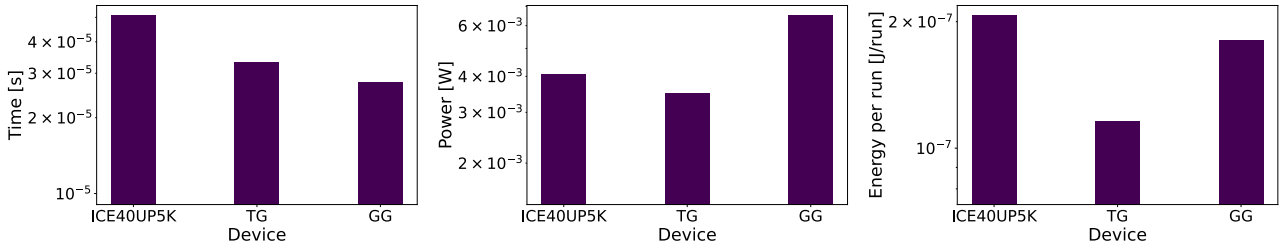


(a) Execution time (b) Average power (c) Energy per run

Figure 5.1: Bar plots containing metrics from the AES algorithm

The energy consumption shows that the FPGA impacts the battery life significantly less than the other platforms. The energy metric aligns with the power results, confirming that the FPGA consumes less energy overall.

In addition to the regular AES setup, there is an option to run the AES algorithm on dedicated hardware (hard-IP block) on the Gecko devices, designed specifically for efficient encryption. This option is available on both the Tiny Gecko and the Giant Gecko. The results are displayed in Figure 5.2. For this measurement it will be referred to as the AES HW setup.



(a) Execution time (b) Average power (c) Energy per run

Figure 5.2: Bar plots containing metrics from the AES algorithm with AES ASIC enabled on the MCU

The setup for the AES which includes the Gecko's hard-IP block demonstrates a substantial increase in speed and a significant reduction in power consumption compared to the standard AES configuration on the Gecko devices. The Gecko devices run hotter with the dedicated IP block, which makes them able to run faster than the FPGA. However, the Gecko devices also run hotter, the combination of increased energy use in operation and quicker execution of the algorithm results in an altogether great improvement for the Gecko devices.

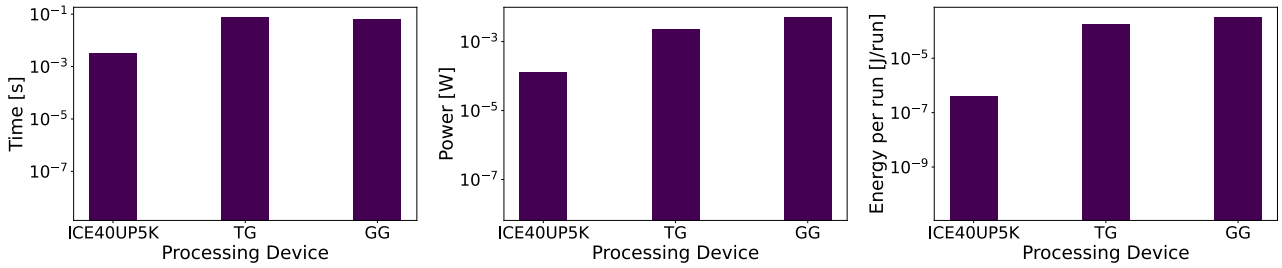
5.2.2 PHOTON

The results of the PHOTON workload are shown in Figure 5.3.

Similar to the AES algorithm, the power consumption shows that the FPGA operates more efficiently when compared to the other platforms. The energy metric follows the power results, confirming that the FPGA consumes less energy overall.

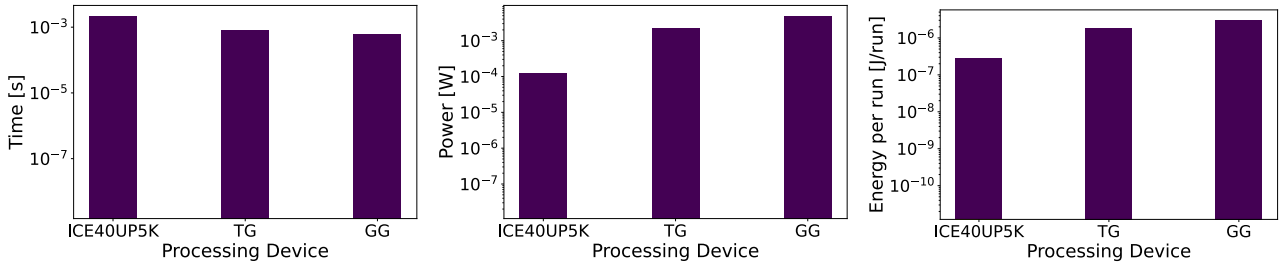
5.2.3 SIMON/SPECK

The results of the SIMON/SPECK workload are shown in Figure 5.4.



(a) Execution time (b) Average power (c) Energy per run

Figure 5.3: Bar plots containing metrics from the PHOTON algorithm



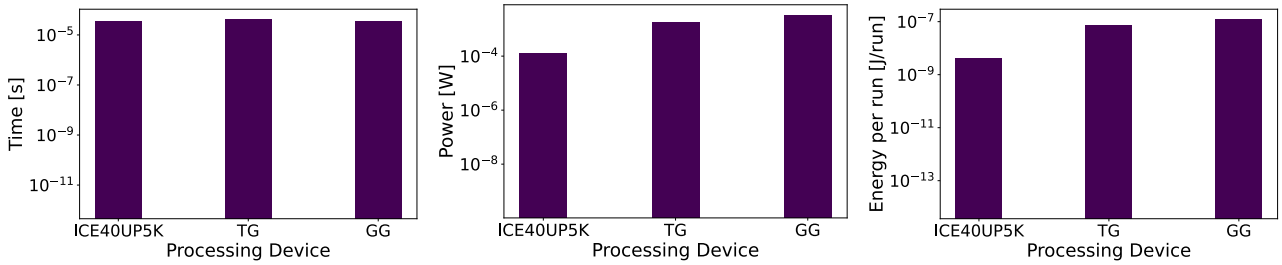
(a) Execution time (b) Average power (c) Energy per run

Figure 5.4: Bar plots containing metrics from the SIMON/SPECK algorithm

Similar to the AES and SIMON algorithms, the power consumption shows that the FPGA requires less power than the other platforms. The energy metric aligns with the power results, confirming that the FPGA consumes less energy overall.

5.2.4 CSD

The bar plots for the CSD algorithm are shown in Figure 5.5.



(a) Execution time (b) Average power (c) Energy per run

Figure 5.5: Bar plots containing metrics for the CSD workload

When it comes to power consumption, the results are more interesting than the previous algorithms. The FPGA is still more efficient, but the MCU is closer to the FPGA for this algorithm. This is due to the simple algorithm. Therefore the MCU can easily provide an efficient outcome. The FPGA could be more efficient if it had enough I/O ports to accommodate enough sensors to use a parallelized approach.

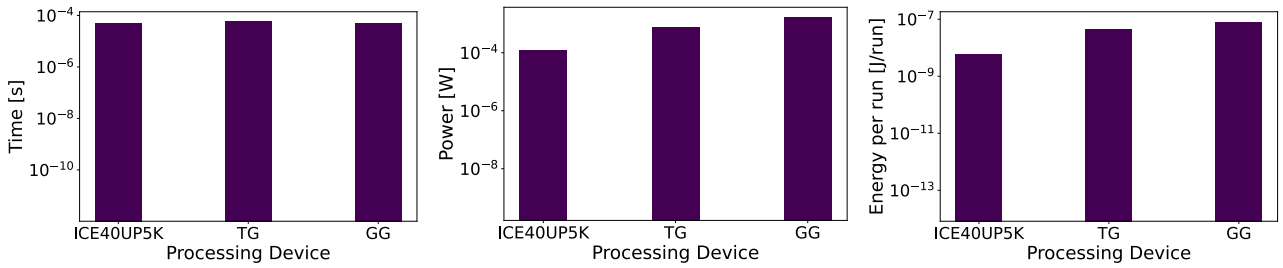
5.2.5 Spike detector

The bar plots for the spike-detector algorithm are shown in Figure 5.6.

The power consumption for this algorithm lies closer than the one used in the CSD algorithm. The expected reason is due to the necessary alterations required for the FPGA implementation.

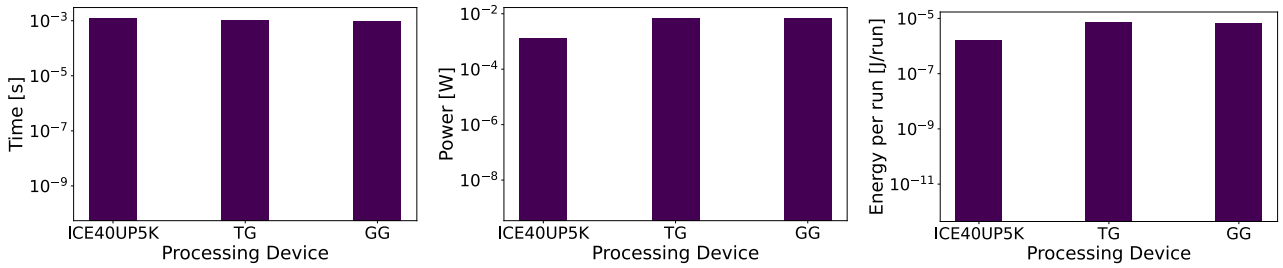
5.2.6 Spike classifier

The bar plots for the spike-classifier algorithm are shown in Figure 5.7.



(a) Execution time (b) Average power (c) Energy per run

Figure 5.6: Bar plots containing metrics for the spike-detector workload



(a) Execution time (b) Average power (c) Energy per run

Figure 5.7: Bar plots containing metrics for the spike-classifier workload

The power consumption for this algorithm lies closer than the spike-detector algorithm. We suspect that this is due to the necessary alterations required for the FPGA implementation.

5.2.7 Conclusions

A general trend emerges from the measurements: throughput-selected operating frequencies greatly favour implementations with FPGA components. These results assume that the MCU is in sleep mode while the FPGA handles the execution of the algorithm.

In conclusion, the FPGA+MCU used in these tests demonstrates lower average power consumption compared to the MCU-only options. The primary contributor to this reduction is the difference in active components between the processing elements, the FPGA runs lighter in comparison to the MCU. Additionally the cycle count favours the FPGA execution of the algorithm. Typically, the FPGA algorithms require fewer cycles, being able to execute the algorithm faster, spending less time and energy on active operation.

While obtaining measurements, the various encryption algorithms produced very similar results in terms of time and power consumption. Therefore, the decision was made to focus primarily on AES, due to the availability of a hard-IP block on the Gecko devices, which allows for testing multiple versions of the algorithm. While using different encryption algorithms could add variety, it would not provide significant new insights.

5.3 Benchmarks

The next stage in the analysis involves examining how the devices perform when different algorithms are paired, highlighting key characteristics of each setup. These highlighted characteristics reveal which devices excel in specific areas, providing valuable insights for interpreting the final results. This process, focused on determining the characteristics of specific scenarios, will be referred to as benchmarking. In Figure 3.1.4 the setup and details for the operational scenarios are found, the details are repeated in Table 5.1.

Points of interest to be explored in the subsequent section include:

- The impact of using medical workloads.
- The addition of encryption algorithms.
- The use of gating and duty cycles to reduce average power consumption.

The measurement section focused on the performance of the base algorithms, for this purpose the energy consumption was a good metric. In this section the focus is the processing elements in use, and for this purpose the battery lifetime is a better metric to use. The battery lifetime presents a good idea how the combination of devices will perform in a realistic scenario. Therefore a viable battery capacity was selected in Section 3.2.3. The battery capacity is then used here with the energy consumption to show the expected time the IMD can be run with the used algorithms. The overhead variables, which contain the reconfiguration costs are plotted, but are often too small to display.

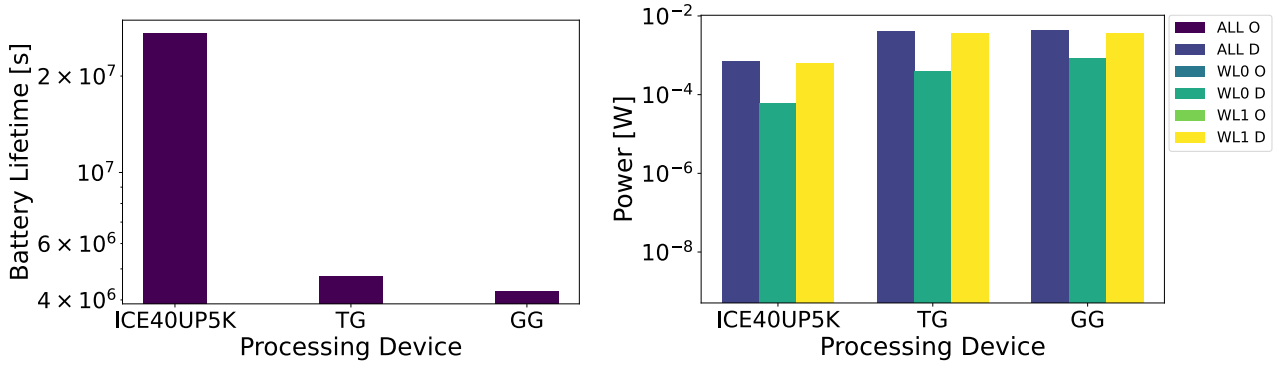
Table 5.1: Definitions for the benchmarks (CSD: Current-Source Density), all workloads in use are based on integer values, for the use of encryption, there are two moments of communication, one weekly & one daily, spike sorting contains the spike detection and classification algorithms.

Benchmark index	Medical algorithms	Wireless algorithms	Duty cycle
Benchmark 1: Using medical workloads			
1.1	Spike sorting	-	-
1.2	CSD	-	-
Benchmark 2: Medical workloads with encryption			
2.1	Spike sorting	AES	-
2.2	CSD	AES	-
Benchmark 3: Medical workloads with gating			
3.1	Spike sorting & Sleep	-	5% active
3.2	CSD & Sleep	-	5% active
3.3	Spike sorting & Off	-	5% active
3.4	CSD & Off	-	5% active
Benchmark 4: Benchmark 3 with flash-FPGA used			
4.1	Spike sorting & Sleep	-	5% active
4.2	CSD & Sleep	-	5% active

5.3.1 Benchmark 1 - Using medical workloads

The goal of the first benchmark is to evaluate the effect of using medical workloads in their designed use and assess the impact on performance when running these medical algorithms concurrently.

Benchmark 1.1 includes the spike detection and spike classification algorithms. In this scenario, the algorithms are executed sequentially, without communicating the resulting data further. The purpose is to evaluate the combined performance of the two algorithms. The resulting metrics can be seen in Figure 5.8, the base workload concerning the full spike sorting algorithm can be seen in effect. The addition of multiple workloads can be seen to affect the results, the resource usage of the FPGA increases.



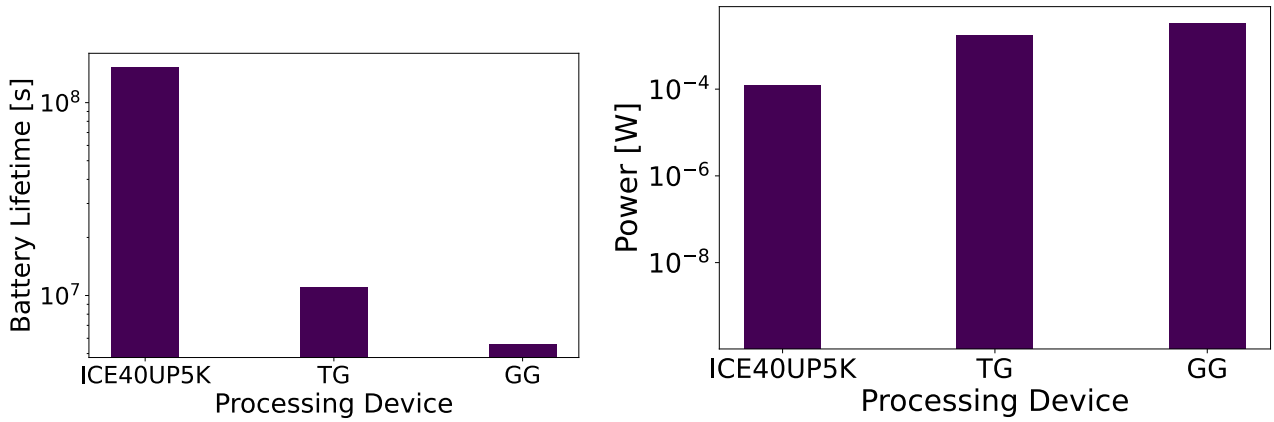
(a) Battery lifetime

(b) Average power

Figure 5.8: Bar plots for benchmark 1.1 (WL: Workload, D: dynamic, O: overhead), *ALL* is the combined value for all workloads, *WL0* is the spike detector, *WL1* is the spike classifier

Benchmark 1.2 uses the CSD workload. In this scenario, the algorithm is executed without communicating the resulting data further. The purpose is to evaluate the algorithm in the intended work setup. The resulting metrics can be seen in Figure 5.9, the base workload concerning the CSD algorithm can be seen in effect.

In the cases within benchmark 1, it can be seen that running multiple workloads increases the frequency of switching FPGA functionality, leading to higher power consumption for the FPGA due to the reconfiguration



(a) Battery lifetime

(b) Average power

Figure 5.9: Bar plots for benchmark 1.2 (WL: Workload, D: dynamic, O: overhead), *ALL* is the combined value for all workloads, *WL0* is CSD

costs. This impact becomes more pronounced as the frequency of reconfiguration increases. The power used by the FPGA increases more rapidly compared to the MCU when being in sleep mode. Therefore, when there is additional downtime between tasks, which happens when the FPGA reconfigures, the FPGA incurs significant penalties. The performance of the FPGA+MCU in handling the medical workloads demonstrates an advantage over the MCU-only in terms of average power consumption . The FPGA continues to maintain a lead in power efficiency despite the increased switching.

5.3.2 Benchmark 2 - Medical workloads with encryption

The second benchmark introduces encryption into the mix of workloads. This additional workload affects both time requirements and power usage.

Benchmark 2.1 includes the spike detection and classification algorithms, with the resulting data being communicated using AES encryption. As in the previous benchmark, the spike sorting algorithms are run sequentially. The scenario is visualised in Figure 5.10. The resulting metrics can be seen in Figure 5.11, the addition of the cipher to the spike sorting algorithm can be seen. This addition shows as an additional component in the plot, but is a small component of the total usage. The implementation for the cipher is more efficient for the FPGA, thereby keeping the devices in the same position as with the first benchmark.

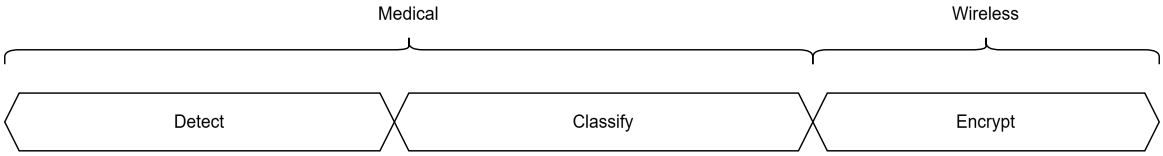
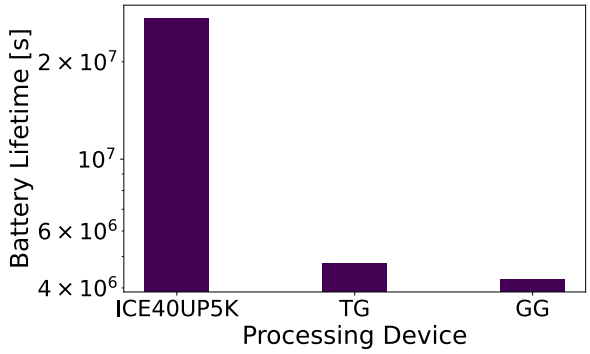


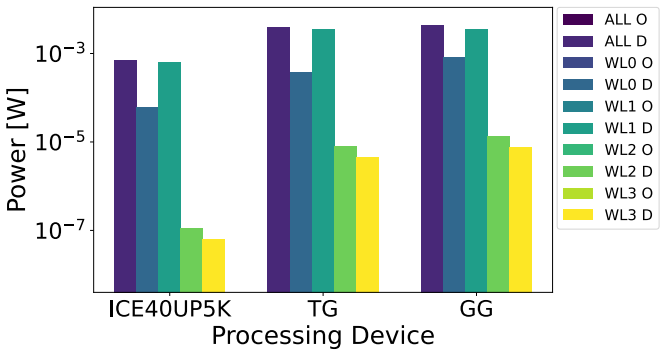
Figure 5.10: Benchmark 2.1 in practice.

The next set of figures addresses benchmark 2.2, which differs from benchmark 2.1 in using the CSD algorithm instead of the spike-sorting algorithms. The scenario is visualised in Figure 5.12. The resulting metrics can be seen in Figure 5.13, the addition of the cipher to the CSD algorithm can be seen. As with benchmark 2.1, the addition is a relatively small component of the total usage, thereby not altering the state of the devices much.

Benchmark 2 looks visually similar to those in the previous benchmark, from figures 5.8 and 5.9. The addition of encryption and other workloads requires the FPGA to reconfigure more frequently, increasing its downtime when no tasks can be processed. However, the downtime is short compared to the active operation, and the power consumption during reconfiguration remains minimal. The downtime is 70.7 ms per reconfiguration cycle, see Table 3.2.2 , as compared to active operation of 12 hours in a row. Even with the additional workloads and reconfiguration, the FPGA continues to maintain an advantage over the MCU in terms of average power usage.



(a) Battery lifetime



(b) Average power

Figure 5.11: Bar plots for benchmark 2.1 (WL: workload, D: dynamic, O: overhead (defined in Section 3.5.2)), *ALL* is the combined value for all workloads, *WL0* is spike detection, *WL1* is spike classification, *WL2* is AES daily, *WL3* is AES weekly

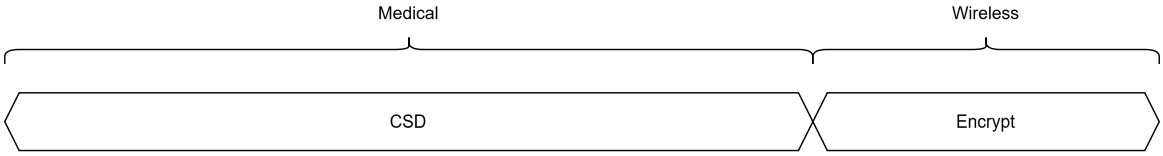
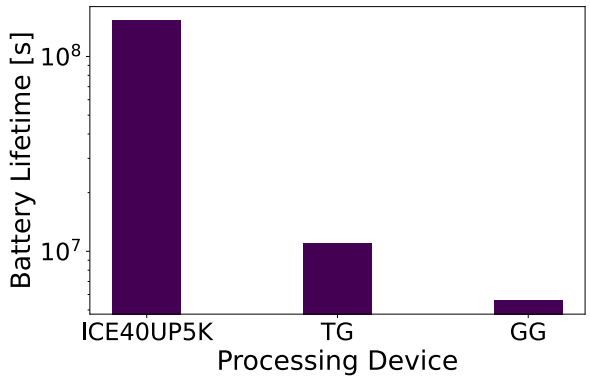
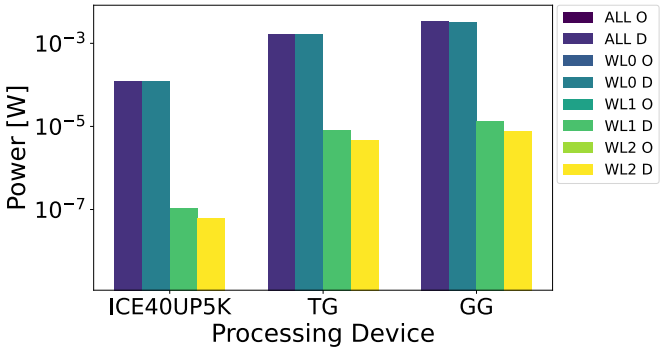


Figure 5.12: Benchmark 2.2 in practice.



(a) Battery lifetime



(b) Average power

Figure 5.13: Bar plots for benchmark 2.2 (WL: workload, D: dynamic, O: overhead (defined in Section 3.5.2)), *ALL* is the combined value for all workloads, *WL0* is CSD, *WL1* is AES daily, *WL2* is AES weekly

5.3.3 Benchmark 3 - Medical workloads with gating

The third benchmark evaluates the impact of using gating techniques. In this context, the analysis tool simulates power and clock gating. This is executed with a duty cycle over a 24-hour period to gather data. The duty cycle defines what percentage of the time the device is actively executing an algorithm. The goal is to reduce the power requirements of the implant over the same time period, thus extending the device's operational independence. The implementation of this option involves the devices executing the required algorithms for part of the day while remaining in a low-power mode for the remainder of the day.

Benchmark 3.1 features spike detection and classification algorithms The data resulting from these algorithms is not communicated in this scenario; only the algorithms are run and executed sequentially. The duty cycle of the algorithm is set to 5% active operation, with the device being set to clock-gate mode for the remainder. The scenario is visualised in Figure 5.14. The resulting metrics can be seen in Figure 5.15, the large amount of time that the devices spend in sleep mode is a major component of the consumption of energy. The MCU devices have a significantly more efficient consumption in sleep mode.

Similarly, benchmark 3.2 differs from benchmark 3.1 by using the CSD algorithm instead of the spike sorting algorithms. The data resulting from these algorithms is not communicated in this scenario; only the algorithms

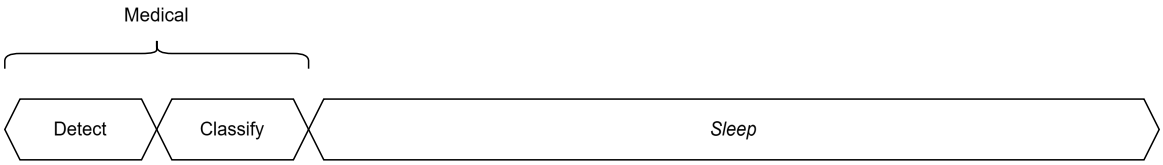
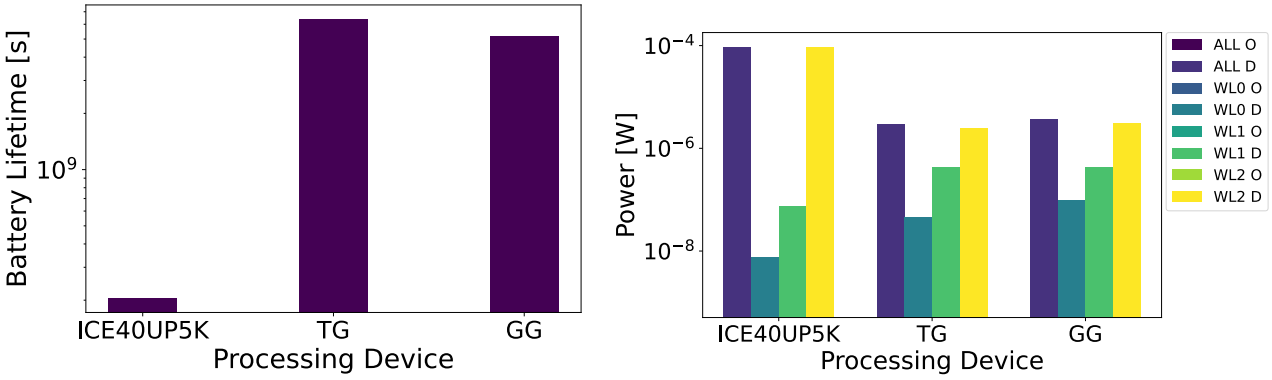


Figure 5.14: Benchmark 3.1 in practice.



(a) Battery lifetime

(b) Battery lifetime

Figure 5.15: Bar plots for benchmark 3.1 (WL: workload, D: dynamic, O: overhead (defined in Section 3.5.2)), *ALL* is the combined value for all workloads, *WL0* is spike detection, *WL1* is spike classification, *WL2* is Off

are run and executed sequentially. The duty cycle of the algorithm is set to 5% active operation, with the device being set to clock-gate mode for the remainder. The scenario is visualised in Figure 5.16. The resulting metrics can be seen in Figure 5.17, as with the previous benchmark, the large amount of time that the devices spend in sleep mode is a major component of the consumption of energy. The difference in the consumption is smaller compared to benchmark 3.1, but the FPGA still consumes more energy.

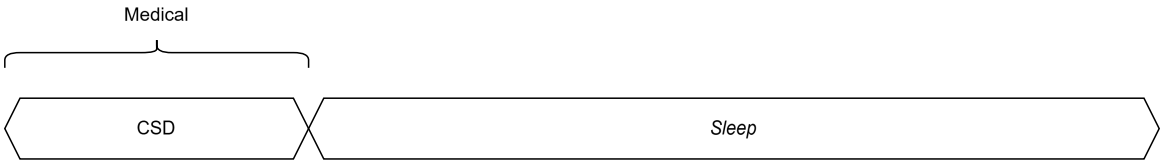
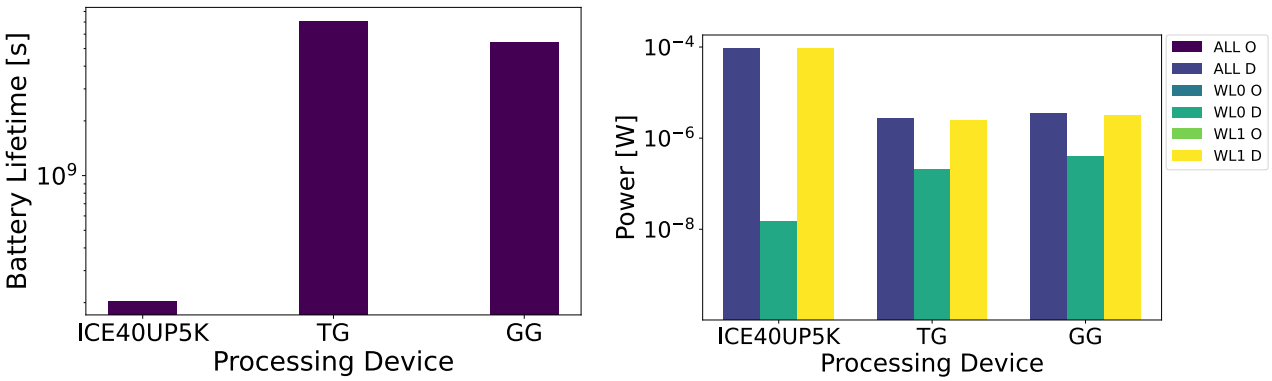


Figure 5.16: Benchmark 3.2 in practice.



(a) Battery lifetime

(b) Average power

Figure 5.17: Bar plots for benchmark 3.2 (WL: workload, D: dynamic, O: overhead (defined in Section 3.5.2)), *ALL* is the combined value for all workloads, *WL0* is CSD, *WL1* is Off

Benchmark 3.3 includes spike detection and classification algorithms. The data resulting from these algorithms is not communicated in this scenario; only the algorithms are run and executed sequentially. The duty cycle of the algorithm is set to 5% active operation, with the device being set to power-gate mode for

the remainder. The scenario is visualised in Figure 5.18. The resulting metrics can be seen in Figure 5.19, the addition of power gating instead of clock gating lowers the drain of the devices when not in use to the point that the FPGA and the MCU devices operate comparable.

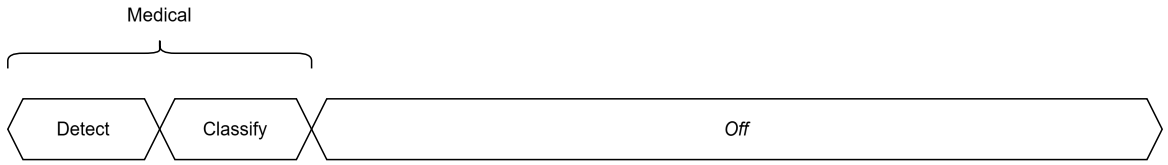
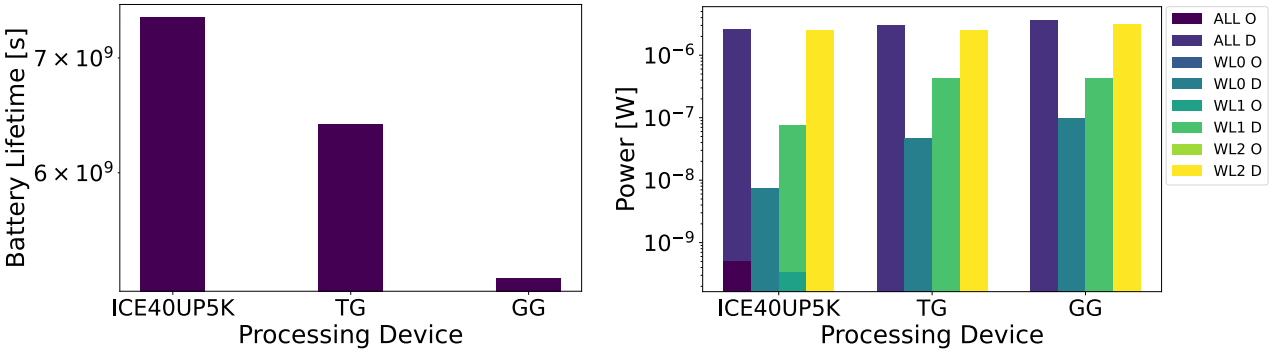


Figure 5.18: Benchmark 3.3 in practice.



(a) Battery lifetime

(b) Average power

Figure 5.19: Bar plots for benchmark 3.3 (WL: workload, D: dynamic, O: overhead (defined in Section 3.5.2)), *ALL* is the combined value for all workloads, *WL0* is spike detection, *WL1* is spike classification, *WL2* is Off

The next set of figures focuses on benchmark 3.4, which differs from benchmark 3.3 by using the CSD algorithm instead of the spike sorting algorithms. The data resulting from these algorithms is not communicated in this scenario; only the algorithms are run and executed sequentially. The duty cycle of the algorithm is set to 5% active operation, with the device being set to power-gate mode for the remainder. The scenario is visualised in Figure 5.20. The resulting metrics can be seen in Figure 5.21, as with benchmark 3.3, the devices here have a similar efficiency. The usage of power gating from clock gating bridges the gap in efficiency between their inactive modes.

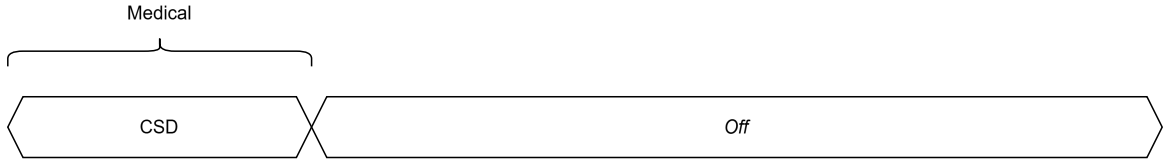
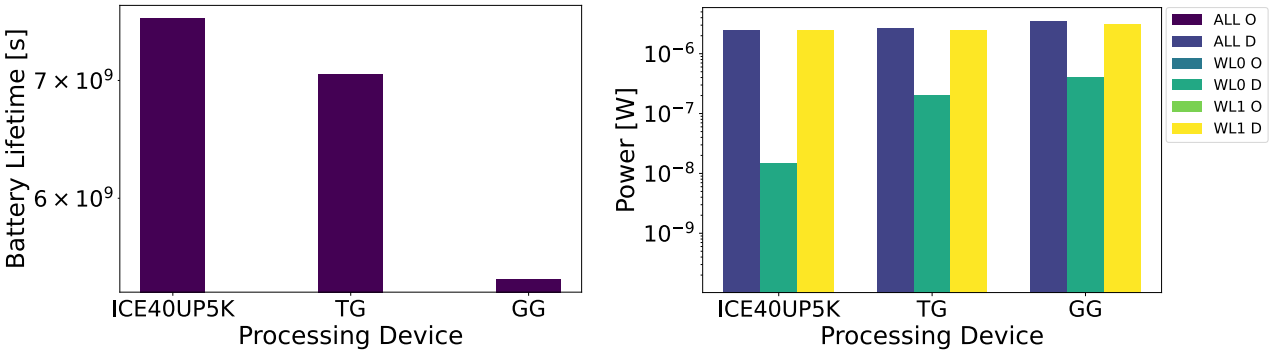


Figure 5.20: Benchmark 3.4 in practice.



(a) Battery lifetime

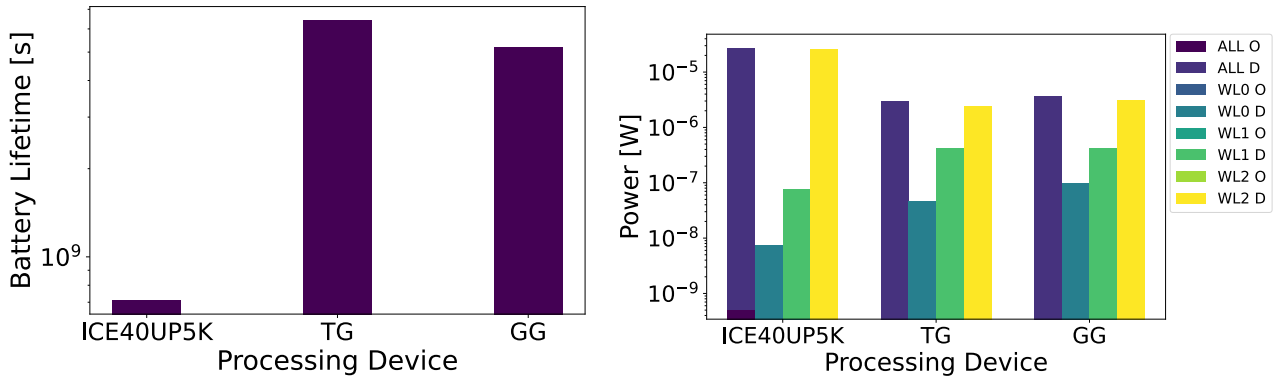
(b) Average power

Figure 5.21: Bar plots for benchmark 3.4 (WL: workload, D: dynamic, O: overhead (defined in Section 3.5.2)), *ALL* is the combined value for all workloads, *WL0* is CSD, *WL1* is Sleep

Benchmark 3 contains 2 sets of bar plots, containing different types of gating. indicate that the clock gate negatively impacts the FPGA's performance. This is due to the need to keep more devices in a static mode, leading to higher power consumption. The performance disadvantage is considerable when compared to the fully active mode scenarios previously discussed. In contrast, the bar plots presented in figures 5.15 and 5.17 , the FPGA shows significantly better performance compared to the MCU, even surpassing previous benchmark scenarios. The inclusion of power gating benefits the FPGA the most. This was expected, because the FPGA uses the most energy in active or sleep mode, and thereby can gain the most from being shut off.

5.3.4 Benchmark 4 - Benchmark 3 with flash-FPGA used

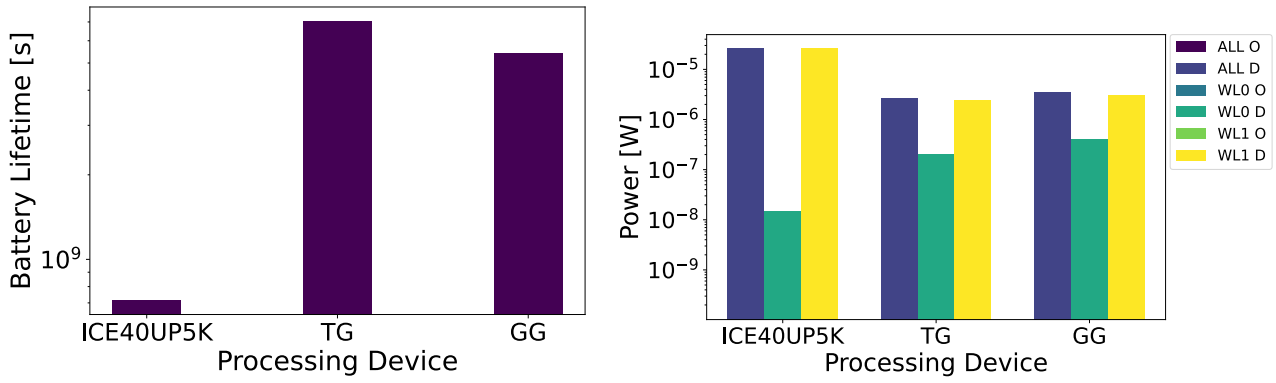
In addition to the previous benchmark 3 part, the FPGA shows a bad performance for sleep mode. Using a flash-based FPGA improves the sleep mode utilisation, due to lower energy usage while in sleep mode. This benchmark will use the flash-based FPGA to show the difference. These will be run for two algorithms, the spike sorting algorithm and CSD algorithm, with the same setup as benchmark 3.1 and 3.2 respectively. The names for these two variants are benchmark 4.1 and 4.2 respectively. Figure 5.15 shows the operation for benchmark 4.1 and Figure 5.17 shows the operation for benchmark 4.2. These correspond to the visualisation of Figure 5.14 and Figure 5.16.



(a) Battery lifetime

(b) Average power

Figure 5.22: Bar plots for benchmark 4.1 (WL: workload, D: dynamic, O: overhead (defined in Section 3.5.2)), *ALL* is the combined value for all workloads, *WL0* is spike detection, *WL1* is spike classification, *WL2* is Sleep



(a) Battery lifetime

(b) Average power

Figure 5.23: Bar plots for benchmark 4.2 (WL: workload, D: dynamic, O: overhead (defined in Section 3.5.2)), *ALL* is the combined value for all workloads, *WL0* is CSD, *WL1* is Sleep

The resulting metrics can be seen in figures 5.22 and 5.23. It can be seen that the usage of the flash-based FPGA in these computations shows improved result for the FPGA, but the results still favour the MCU implementation.

Overall, the results of most benchmarks are favourable for the FPGA. In these use cases, the MCU operates in the background in a static state. These positive results from complex benchmark scenarios suggest that FPGA technology could be highly beneficial for use in IMD platforms. This provides a strong indication that an FPGA-based solution is viable and potentially superior for such applications.

5.4 Operational scenarios

In the previous chapter, the benchmarks were performed to have a basic analysis to work off of. To continue the analysis, focus on comparing how the device variants perform when all benchmark aspects are combined, creating more complex situations. These results are then used to draw conclusions for the problem statement. In Table 3.1.4 the setup and details for the operational scenarios are found, the table and details will be repeated here. In Table 5.2 the details of the scenarios are displayed. The environment defines the location and subsequent duty cycle, the algorithms in use determine how much energy is required for the execution of said algorithms, the communication frequency determines how much energy is expended on encryption and finally the gating component defines how the processing elements are set to outside of the active execution of the algorithms.

Table 5.2: Operational scenario definitions, the differences between scenarios are shown through the variables (CSD: Current-Source Density), all workloads in use are based on integer values, spike sorting contains the spike detection and classification algorithms.

Scenarios	*Environment*	*Duty cycle*	*Workloads*	*Communication*	*Gating*
1	Cardiac	5%	Spike detection	Weekly	Clock
2	Cardiac	5%	Spike detection	Weekly	Power
3	Cardiac	5%	Spike detection	Daily	Clock
4	Cardiac	5%	Spike detection	Daily	Power
5	Neural	100%	CSD	Weekly	-
6	Neural	100%	CSD	Daily	-
7	Neural	100%	Spike sorting	Weekly	-
8	Neural	100%	Spike sorting	Daily	-

Then subsequently, Table 5.3 contains the resulting values for each scenario. In the section below, these values are used in bar plots to visualise how the results compare.

Table 5.3: The results from the operational scenarios, used to produce plots. The definitions for the scenarios can be found in Table 3.2 (iCE40: iCE40UP5K, TG: Tiny Gecko, GG: Giant Gecko)

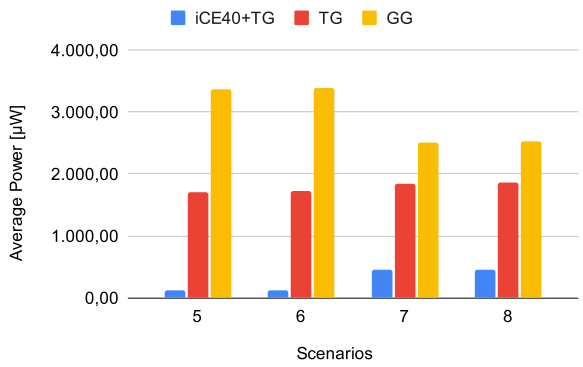
	Average Power [μW]			*Battery lifetime [years]*		
Scenarios	*iCE40+TG*	*TG*	*GG*	*iCE40+TG*	*TG*	GG
1	94,06	45,65	92,89	6,37	13,13	6,45
2	8,60	45,65	92,89	69,73	13,13	6,45
3	94,17	72,55	137,11	6,37	8,26	4,37
4	8,96	72,55	137,11	66,98	8,26	4,37
5	123,68	1.711,15	3.362,50	4,85	0,35	0,18
6	123,70	1.717,98	3.377,35	4,85	0,35	0,18
7	455,14	1.847,11	2.497,09	1,32	0,32	0,24
8	454,18	1.852,29	2.519,71	1,32	0,32	0,24

5.4.1 Comparison of scenarios

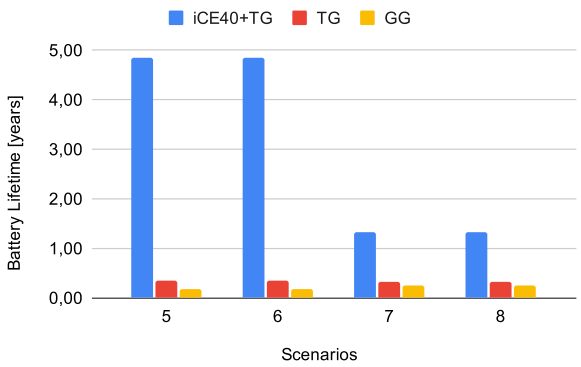
Each scenario provides output values for various devices, which can be compared to assess the effectiveness of those devices in the given scenario. By combining different scenarios, more insights can be gained about the performance of the combined devices.

Case 1: Neural environment
In this case, all defined scenarios for neural-IMD applications are compared. These comparisons primarily focus on basic differences between devices. Notably, the difference between power gating and clock gating does not significantly impact these scenarios, so they have been excluded as separate options. In the comparison shown in Figure 5.24, the FPGA outperforms the other devices in terms of power consumption. The primary reason for this is that the neural environment involves an 'always-on' algorithm, where the FPGA excels due to its lower power drain while continuously running algorithms compared to other device options.

Case 2: Cardiac environment
The second case examines the operation of devices in cardiac applications. In this environment, the active algorithm, the spike detection algorithm, operates under a lower frequency due to the duty cycle, allowing the use of gating techniques. The type of gating used makes a significant difference, there is clock gating or power



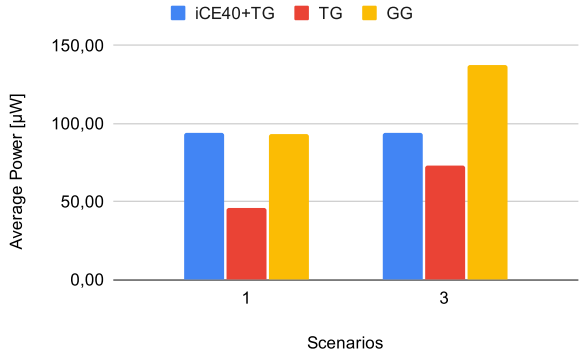
(a) Comparison for the average power



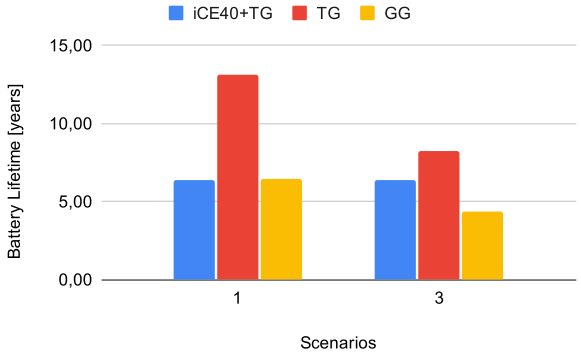
(b) Comparison for the battery lifetime

Figure 5.24: Comparison of the scenarios within the neural environment (iCE40: iCE40UP5K, TG: Tiny Gecko, GG: Giant Gecko)

gating. Clock gating removes the *CLK* input from the device, thereby lowering the power use on the device. Power gating on the other hand cuts the device off from power, saving a great deal of power.



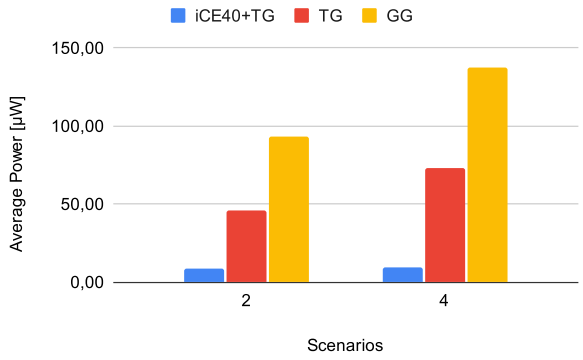
(a) Comparison for the average power



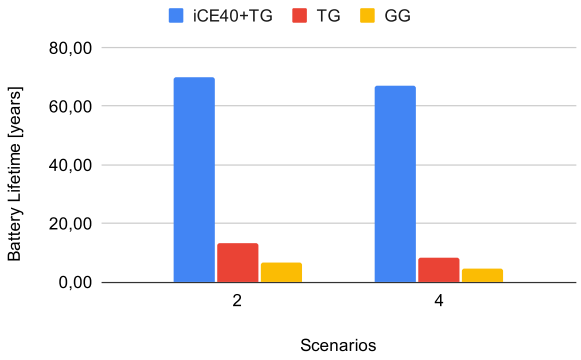
(b) Comparison for the battery lifetime

Figure 5.25: Comparison of the clock-gated scenarios within the cardiac environment (iCE40: iCE40UP5K, TG: Tiny Gecko, GG: Giant Gecko)

In the comparison shown in Figure 5.25, the performance of the FPGA and MCU variants is quite similar. The reason for this is that in FPGA configurations, the MCU is running in the background, contributing to additional static power consumption. In this case, the power savings from algorithm execution on the FPGA are not enough to offset the overall static-power usage, leading to comparable performance between the two.



(a) Comparison for the average power



(b) Comparison for the battery lifetime

Figure 5.26: Comparison of the power-gated scenarios within the cardiac environment (iCE40: iCE40UP5K, TG: Tiny Gecko, GG: Giant Gecko)

When power gating, as seen in Figure 5.26, the FPGA shows a significant advantage over the MCU in power consumption. The primary reason for this is that the power savings achieved for the FPGA through power gating are large enough to offset the static-power consumption incurred when having both the FPGA and MCU active in the scenario.

To sum up, the clock-gated scenario shows that using the FPGA processing element does not improve the battery lifetime, thereby not resulting in a positive outcome for the thesis goal. However, when power gating, the FPGA processing element has a positive outcome on the battery lifetime, thereby granting a positive outcome for the thesis goal.

5.5 Summary

Chapter 5 began with the tool validation, which confirmed that the tool's results were consistent with manual calculations. After validation, the tool was able to produce a significant amount of reliable data. Given the complexity of the question and the many variables involved, this validation was crucial for achieving the final results.

Next, the measurement results were presented, showing individual device performance. These results demonstrated a clear trend: the FPGA consistently outperformed other options, with the exception of the MCU Hardware (HW) variant for AES, which uses dedicated hardware for the algorithm.

Following the measurement results, the benchmark results were introduced, focusing on specific aspects of device performance. The benchmarks revealed that the FPGA performed more efficiently in the first and second benchmark scenarios. However, in the third benchmark scenario, involving clock gating, the FPGA did not show an advantage. When power gating was applied, though, the FPGA once again outperformed the MCU.

The operational results focused on the practical use of algorithms and comparing the outcomes. These results showed that the FPGA used significantly less power than the MCU across most scenarios. The exception was the clock-gated cardiac scenario, which highlighted a key limitation of the FPGA—leaving it in a static mode requires additional power to maintain its configuration. Therefore, power gating, which lowers that drain, makes an FPGA a viable addition to the IMD platform.

Chapter 6

Conclusions

6.1 Contributions

In this thesis, we have made the following contributions:

- We have successfully modified the algorithms to be fully functional across all devices used in this thesis.
- We have performed all device measurements, except for the security algorithm measurements (AES, PHOTON, and SIMON) on the MCU.
- We have analysed the results and provided the final conclusion of the thesis, which can be found in Chapter 5 and Chapter 6 respectively.

6.2 Summary

In Chapter 2, the background information was provided for this thesis. This includes concepts which are focused on implementation factors from a later chapter, as well as basic knowledge of components. Knowledge regarding IMDs was provided, which continued into processing devices, algorithms and finally parameters for the accelerators within the processing devices. Together it formed a basis on which to base the subsequent research.

In Chapter 3, the theoretical start of the research was defined, starting with the experimental design. Through use of the experimental design, devices within their respective categories were selected, under which processing devices, which were required for the test setup, and devices for gating. Subsequently algorithms were selected and researched, providing a start for the implementation of the algorithms on the processing devices. Then the characteristics of the devices were converted into the requirements surrounding the measurements. The theoretical design for the analysis tool is given in the finish of the chapter.

In Chapter 4, the definitions made in Chapter 3 were implemented. For this to happen, the details for measurements on each processing device were researched. Subsequently the code for the algorithms was altered to be able to be used on each processing device in use. After checking the functionality, the measurements were then taken and saved, to be used in the analysis tool. The analysis tool was designed and subsequently used to combine the obtained values in a consistent way, making us able to draw conclusions.

In Chapter 5, finally, we have determined that all the separate workloads produced similar trends (Section 5.2), which largely held in the subsequent sections (Section 5.3, Section 5.4). One of the key indicators of the FPGA's power efficiency, even before the measurements were conducted, was the lower cycle counts of the algorithms. To ensure a fair comparison between devices, a consistent throughput (bits per unit of time) was used as the primary factor for equalizing performance across platforms. A higher cycle count requires a higher operating frequency, and an increase in operating frequency leads to increased dynamic power consumption. The lower cycle counts observed for the FPGA meant it can operate at a lower frequency, resulting in lower dynamic power consumption. This outcome was anticipated early in the thesis.

The validation of the analysis tool confirmed that the results of the tool were consistent with manual calculations. After validation, the tool was able to produce a significant amount of reliable data. Given the

complexity of the question and the many variables involved, this validation was crucial for achieving the final results.

Next, the measurement results were presented, showing individual device performance. These results demonstrated a clear trend: the FPGA consistently outperformed other options, with the exception of the MCU HW variant for AES, which uses dedicated hardware for the algorithm.

There were notable differences in the benchmark scenarios that highlighted various aspects of the devices. The first case, as described in Section 5.3.1, involved the combination of multiple medical algorithms. This use case simulates detecting a defect and subsequently processing the defect. The results for this benchmark closely mirrored the measurement results. The second case, described in Section 5.3.2, introduced a security algorithm alongside a medical algorithm. Similar to the first benchmark, the results remained consistent with the initial measurement outcomes. The third case, described in Section 5.3.3, added gating to the medical algorithms, showing great promise for the FPGA. However, clock gating resulted in unfavourable outcomes for the FPGA. The sleep mode for the FPGA is inefficient due to its reliance on SRAM, which requires constant power to maintain the configuration. Over time, it would be more power-efficient to shut down the device and reconfigure it rather than maintain it in sleep mode. However, when using a duty cycle, the device is expected to execute the algorithm once in a predefined timespan, thereby limiting the use of power-gating for a conventional SRAM based FPGA. To use power gating without reconfiguration, a flash-based FPGA should be used. As described in Section 5.3.4, using a flash-FPGA provides a solution, as it saves the configuration during power-down, enabling the use of power gating. Power gating eliminates the static power costs associated with SRAM, significantly boosting the FPGA's performance and making it a strong addition to the IMD platform.

The results presented in Section 5.4 combine scenarios that reflect real-world usage of the algorithms on the platform. These results show that the FPGA performs well enough to be considered a viable addition to an IMD platform, given that the implementation is done in an appropriate way to avoid the pitfalls of an FPGA.

The main reason for the pronounced difference between the MCU, and indirectly the CPU, and FPGA in the results lies in the comparison method used. The metric in this thesis focused primarily on average power and battery lifetime. Typically, power consumption and execution time are combined to assess device performance. However, in this thesis, the metric of throughput frequency—which alters the operating frequency—was used, excluding execution time from the equation. This alteration proved to make an easy way to compare devices, using a constant operating frequency can prove effective for this purpose as well. Optimising for the device used across multiple types of devices, algorithms and settings would require a large amount of additional work.

The conclusion is that, for the given inputs, the FPGA performs exceptionally well. The measurements demonstrate the natural efficiency of the FPGA, without showing any signs of underperformance. Even in complex workloads, which would typically favour the MCU due to its flexibility, the FPGA exhibits highly favourable results in terms of average power consumption. The benchmark for which the FPGA did not perform well was clock gating, the static power consumption of the FPGA outweighs the gains from using it on regular algorithms. Based on the duty cycle used, the gains an FPGA achieves during active operation will start to outweigh the static costs while clock gated. The inclusion of an FPGA in an IMD platform is not only suitable for prototyping but also highly practical for regular use.

6.3 Future work

Any research is never truly complete. Each discovery or resolved question often leads to new inquiries. The most prominent new avenues arising from this thesis will be highlighted here.

6.3.1 Option 1: Partial reconfiguration

During the course of the project, promising approaches emerged regarding the implementation details of smaller algorithms. In an FPGA, the reconfiguration process is often challenging, especially for large devices or concurrent execution of algorithms. However, if only a portion of the device can be reconfigured as needed, both the power during active and static operation could be reduced and the time required for reprogramming could be reduced. This addition would enhance the device's viability by mitigating some of the downsides associated with FPGAs.

6.3.2 Option 2: Thermal characteristics

Another important concern that was identified but not explored in detail is the thermal characteristics of the IMD use case. IMDs, being implanted in the body, generate heat. The exact amount of heat produced by the device, along with the body's capacity to absorb and dissipate heat, is currently too broad a scope to research in the required capacity in this thesis. Understanding these thermal variables is crucial for determining the safety and feasibility of the device in a real-world setting.

6.3.3 Option 3: Additional device type

The introduction of another device type could provide valuable data for more comprehensive comparisons. Ideally, this additional device would feature dedicated hardware optimized for algorithm acceleration. Hardware capable of parallel execution, increased speed or reduced power consumption would offer significant advantages. Given the chosen trajectory for IMDs, this device must maintain a degree of flexibility. Examples of potential devices include GPUs, Application-Specific Instruction set Processor (ASIP)s, or alternative FPGAs that focus on flash memory in their development. It is recommended to consider low-power FPGAs with flash memory or a low-power ASIP for further exploration. An alternate component which could prove interesting is an FPGA which has a small amount of flash cells near the SRAM component used to save the configuration. Such a device could hold multiple configurations that can switch quickly due to the lack of memory bandwidth required during the reconfiguration process. Such an implementation would increase the viability of a small FPGA which could be used as an accelerator for multiple algorithms.

6.3.4 Option 4: Additional algorithms

Additional algorithms could be used in the analysis. There exist promising avenues for additions, specifically compression algorithms for data transfer, a basic algorithm in use for pacemakers or an additional medical algorithm that uses a neural network. Using a combination of the original and the additions would make a well-rounded set to draw conclusions from.

6.3.5 Option 5: Cohesive comparison for hybrid combinations of processing devices in IMDs

In this thesis, comparisons were made between various algorithms running on a single device. Each algorithm was executed on the same device. However, combining the strengths of different devices to run specific algorithms was not explored, though it presents a promising opportunity. For instance, a lightweight algorithm could run on the MCU while the FPGA handles a more computationally intensive algorithm. Another possible implementation is concurrent utilisation, using two devices to run two algorithms at once.

The primary reason for not implementing this hybrid approach in the thesis is the significant increase in complexity when using multiple devices. The number of potential results expands exponentially, from 4 to 4^n, where n represents the number of algorithms used at the same iteration. This exponential increase necessitates filtering or classifying the outputs to determine which configurations are viable for the intended application.

Bibliography

[1] O Aquilina. "A brief history of cardiac pacing". In: *Images in Paediatric Cardiology* 8.2 (2006), pp. 17–81. ISSN: 1729-441X. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3232561/ (visited on 09/14/2023).

[2] Robert P. Carlyon and Tobias Goehring. "Cochlear Implant Research and Development in the Twenty-first Century: A Critical Update". en. In: *Journal of the Association for Research in Otolaryngology* 22.5 (Oct. 2021), pp. 481–508. ISSN: 1525-3961, 1438-7573. DOI: 10.1007/s10162-021-00811-5. URL: https://link.springer.com/10.1007/s10162-021-00811-5 (visited on 11/11/2024).

[3] Anna Trafton. *An implantable device could enable injection-free control of diabetes*. Sept. 2023. URL: https://news.mit.edu/2023/implantable-device-enable-injection-free-control-diabetes-0918.

[4] Ansheed A. Raheem et al. "A Review on Development of Bio-Inspired Implants Using 3D Printing". en. In: *Biomimetics* 6.4 (Nov. 2021), p. 65. ISSN: 2313-7673. DOI: 10.3390/biomimetics6040065. URL: https://www.mdpi.com/2313-7673/6/4/65 (visited on 11/11/2024).

[5] I. C. Mceachern. *The Future of Implantable Medical Devices: What to Expect in 2024*. Feb. 2024. URL: https://www.iancollmceachern.com/single-post/the-future-of-implantable-medical-devices-what-to-expect-in-2024.

[6] Alejandro Baldominos, Yago Saez, and Pedro Isasi. "A Survey of Handwritten Character Recognition with MNIST and EMNIST". en. In: *Applied Sciences* 9.15 (Aug. 2019), p. 3169. ISSN: 2076-3417. DOI: 10.3390/app9153169. URL: https://www.mdpi.com/2076-3417/9/15/3169 (visited on 11/11/2024).

[7] Ali Olyanasab and Mohsen Annabestani. "Leveraging Machine Learning for Personalized Wearable Biomedical Devices: A Review". en. In: *Journal of Personalized Medicine* 14.2 (Feb. 2024), p. 203. ISSN: 2075-4426. DOI: 10.3390/jpm14020203. URL: https://www.mdpi.com/2075-4426/14/2/203 (visited on 11/11/2024).

[8] David Veselka. *Exploring Feasibility of FPGAs in Implantable Medical Devices | TU Delft Repositories*. URL: https://repository.tudelft.nl/islandora/object/uuid%3Ad27d9003-9c68-4383-b835-0335d5a68ba7 (visited on 09/14/2023).

[9] David C. Bock et al. "Batteries used to power implantable biomedical devices". en. In: *Electrochimica Acta* 84 (Dec. 2012), pp. 155–164. ISSN: 00134686. DOI: 10.1016/j.electacta.2012.03.057. URL: https://linkinghub.elsevier.com/retrieve/pii/S0013468612003969 (visited on 07/15/2024).

[10] Deb Hipp. *Normal resting heart Rate by Age (Chart)*. en. Mar. 2024. URL: https://www.forbes.com/health/wellness/normal-heart-rate-by-age/.

[11] Muhammad Ali Siddiqi et al. "A Lightweight Architecture for Real-Time Neuronal-Spike Classification". en. In: *Proceedings of the 21st ACM International Conference on Computing Frontiers*. Ischia Italy: ACM, May 2024, pp. 32–40. DOI: 10.1145/3649153.3649186. URL: https://dl.acm.org/doi/10.1145/3649153.3649186 (visited on 11/04/2024).

[12] Ravi Rao. *FPGA Design: A Comprehensive Guide to Mastering Field-Programmable Gate Arrays*. en. FPGA structure. Apr. 2023. URL: https://www.wevolver.com/article/fpga (visited on 10/08/2024).

[13] Jan M. Rabaey, Anantha P. Chandrakasan, and Borivoje Nikolić. *Digital integrated circuits: a design perspective*. eng. 2. ed. Prentice Hall electronics and VLSI series. Upper Saddle River, NJ: Prentice Hall, 2003. ISBN: 978-0-13-090996-1.

[14] Microsemi. *Dynamic Power Reduction in Flash FPGAs*. en. Tech. rep. Microsemi, 2012, p. 25. URL: https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ApplicationNotes/ApplicationNotes/dynamic_power_reduction_an.pdf.

[15] PG Scholar. "A SRAM Memory Cell Design in FPGA". en. In: *International Journal of Computer Applications* 71.1 (June 2013), pp. 23–26. ISSN: 0975-8887. DOI: 10.5120/12323-8541.

[16] P. Pavan et al. "Flash memory cells-an overview". en. In: *Proceedings of the IEEE* 85.8 (Aug. 1997), pp. 1248–1271. ISSN: 00189219. DOI: 10.1109/5.622505. URL: http://ieeexplore.ieee.org/document/622505/ (visited on 09/14/2023).

[17] Devashree. *Visualize Deep Learning Models using Visualkeras*. en. Apr. 2023. URL: https://www.analyticsvidhya.com/blog/2022/03/visualize-deep-learning-models-using-visualkeras/ (visited on 10/09/2024).

[18] GeeksforGeeks. *Visual Cryptography: Introduction*. en. Sept. 2023. URL: https://www.geeksforgeeks.org/visual-cryptography-introduction/ (visited on 10/09/2024).
[19] AnySilicon. *The Ultimate Guide to Power Gating*. en. Jan. 2022. URL: https://anysilicon.com/power-gating/.
[20] Muthukumaran Vaithianathan et al. "Energy-Efficient FPGA Design for Wearable and Implantable Devices". en. In: (2024).
[21] Morteza Babaee Altman et al. "Machine learning algorithms for FPGA Implementation in biomedical engineering applications: A review". en. In: *Heliyon* 10.4 (Feb. 2024), e26652. ISSN: 24058440. DOI: 10.1016/j.heliyon.2024.e26652. URL: https://linkinghub.elsevier.com/retrieve/pii/S2405844024026835 (visited on 11/12/2024).
[22] Muhammad Iqbal Khan and Bruno Da Silva. "Harnessing FPGA Technology for Energy-Efficient Wearable Medical Devices". en. In: *Electronics* 13.20 (Oct. 2024), p. 4094. ISSN: 2079-9292. DOI: 10.3390/electronics13204094. URL: https://www.mdpi.com/2079-9292/13/20/4094 (visited on 11/12/2024).
[23] V. Rosello, J. Portilla, and T. Riesgo. "Ultra low power FPGA-based architecture for Wake-up Radio in Wireless Sensor Networks". en. In: *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*. Melbourne, Vic, Australia: IEEE, Nov. 2011, pp. 3826–3831. ISBN: 978-1-61284-969-0. DOI: 10.1109/IECON.2011.6119933. URL: http://ieeexplore.ieee.org/document/6119933/ (visited on 09/14/2023).
[24] Fei Zhang, Mehdi Aghagolzadeh, and Karim Oweiss. "A Fully Implantable, Programmable and Multimodal Neuroprocessor for Wireless, Cortically Controlled Brain-Machine Interface Applications". en. In: *Journal of Signal Processing Systems* 69.3 (Dec. 2012), pp. 351–361. ISSN: 1939-8018, 1939-8115. DOI: 10.1007/s11265-012-0670-x. URL: http://link.springer.com/10.1007/s11265-012-0670-x (visited on 09/14/2023).
[25] Shoichi Yamaguchi et al. "Programmable wireless sensor node featuring low-power FPGA and microcontroller". en. In: *2013 International Joint Conference on Awareness Science and Technology & Ubi-Media Computing (iCAST 2013 & UMEDIA 2013)*. Aizuwakamatsu, Japan: IEEE, Nov. 2013, pp. 596–601. ISBN: 978-1-4799-2364-9. DOI: 10.1109/ICAwST.2013.6765509. URL: http://ieeexplore.ieee.org/document/6765509/ (visited on 09/14/2023).
[26] Qing Yang et al. "An on-chip security guard based on zero-power authentication for implantable medical devices". en. In: *2014 IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS)*. College Station, TX, USA: IEEE, Aug. 2014, pp. 531–534. ISBN: 978-1-4799-4134-6. DOI: 10.1109/MWSCAS.2014.6908469. URL: https://ieeexplore.ieee.org/document/6908469 (visited on 09/14/2023).
[27] He Qi, Oluseyi Ayorinde, and Benton H. Calhoun. "An ultra-low-power FPGA for IoT applications". en. In: *2017 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. Burlingame, CA: IEEE, Oct. 2017, pp. 1–3. ISBN: 978-1-5386-3766-1. DOI: 10.1109/S3S.2017.8308753. URL: http://ieeexplore.ieee.org/document/8308753/ (visited on 09/14/2023).
[28] Lijuan Xia et al. "A low power flash-FPGA based brain implant micro-system of PID control". en. In: *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. Seogwipo: IEEE, July 2017, pp. 173–176. ISBN: 978-1-5090-2809-2. DOI: 10.1109/EMBC.2017.8036790. URL: https://ieeexplore.ieee.org/document/8036790/ (visited on 11/13/2024).
[29] Santiago Martinez and Juan P. Oliver. "A low power FPGA based control unit for an implantable neuromodulation circuit". en. In: *2019 X Southern Conference on Programmable Logic (SPL)*. Buenos Aires, Argentina: IEEE, Apr. 2019, pp. 63–68. ISBN: 978-1-72811-363-0. DOI: 10.1109/SPL.2019.8714506. URL: https://ieeexplore.ieee.org/document/8714506/ (visited on 09/14/2023).
[30] Tianyu Zhan et al. "A Resource-Optimized VLSI Implementation of a Patient-Specific Seizure Detection Algorithm on a Custom-Made 2.2 cm^2 Wireless Device for Ambulatory Epilepsy Diagnostics". en. In: *IEEE Transactions on Biomedical Circuits and Systems* 13.6 (Dec. 2019), pp. 1175–1185. ISSN: 1932-4545, 1940-9990. DOI: 10.1109/TBCAS.2019.2948301. URL: https://ieeexplore.ieee.org/document/8876674/ (visited on 09/14/2023).
[31] Marwen Roukhami et al. "Very Low Power Neural Network FPGA Accelerators for Tag-Less Remote Person Identification Using Capacitive Sensors". en. In: *IEEE Access* 7 (2019), pp. 102217–102231. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2931392. URL: https://ieeexplore.ieee.org/document/8777074/ (visited on 09/14/2023).
[32] Zheng Zhang, Oscar W Savolainen, and Timothy G Constandinou. "Algorithm and hardware considerations for real-time neural signal on-implant processing". en. In: *Journal of Neural Engineering* 19.1 (Feb. 2022), p. 016029. ISSN: 1741-2560, 1741-2552. DOI: 10.1088/1741-2552/ac5268. URL: https://iopscience.iop.org/article/10.1088/1741-2552/ac5268 (visited on 05/16/2024).
[33] Keyvan Farhang Razi and Alexandre Schmid. "Programmable Seizure Detector Using a 32-bit RISC Processor for Implantable Medical Devices". en. In: *2023 IEEE 14th Latin America Symposium on Circuits and Systems (LASCAS)*. Quito, Ecuador: IEEE, Feb. 2023, pp. 1–4. ISBN: 978-1-66545-705-7. DOI: 10.1109/LASCAS56464.2023.10108303. URL: https://ieeexplore.ieee.org/document/10108303/ (visited on 11/14/2024).

[34] Masoud Rostami, Ari Juels, and Farinaz Koushanfar. "Heart-to-heart (H2H): authentication for implanted medical devices". en. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*. Berlin, Germany: ACM Press, 2013, pp. 1099–1112. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516658. URL: http://dl.acm.org/citation.cfm?doid=2508859.2516658 (visited on 11/13/2024).

[35] Maria Hugle et al. "Early Seizure Detection with an Energy-Efficient Convolutional Neural Network on an Implantable Microcontroller". en. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. Rio de Janeiro: IEEE, July 2018, pp. 1–7. ISBN: 978-1-5090-6014-6. DOI: 10.1109/IJCNN.2018.8489493. URL: https://ieeexplore.ieee.org/document/8489493/ (visited on 09/14/2023).

[36] Simon Heller et al. "Hardware Implementation of a Performance and Energy-optimized Convolutional Neural Network for Seizure Detection". en. In: *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. Honolulu, HI: IEEE, July 2018, pp. 2268–2271. ISBN: 978-1-5386-3646-6. DOI: 10.1109/EMBC.2018.8512735. URL: https://ieeexplore.ieee.org/document/8512735/ (visited on 09/14/2023).

[37] Illsoo Sohn, Yong Hun Jang, and Sang Hyun Lee. "Ultra-Low-Power Implantable Medical Devices: Optical Wireless Communication Approach". en. In: *IEEE Communications Magazine* 58.5 (May 2020), pp. 77–83. ISSN: 0163-6804, 1558-1896. DOI: 10.1109/MCOM.001.1900609. URL: https://ieeexplore.ieee.org/document/9112747/ (visited on 11/14/2024).

[38] Muhammad Ali Siddiqi, Christian Doerr, and Christos Strydis. "IMDfence: Architecting a Secure Protocol for Implantable Medical Devices". en. In: *IEEE Access* 8 (2020), pp. 147948–147964. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3015686. URL: https://ieeexplore.ieee.org/document/9165063/ (visited on 09/14/2023).

[39] Muhammad Ali Siddiqi, Angeliki-Agathi Tsintzira, and Georgios Digkas. "Adding Security to Implantable Medical Devices: Can We Afford It?" en. In: *Proceedings of the 2021 International Conference on Embedded Wireless Systems and Networks*. EWSN '21. Delft: Junction Publishing, Feb. 2021, p. 12. ISBN: 978-0-9949886-5-2. DOI: 10.5555/3451271.3451278.

[40] Muhammad Ali Siddiqi et al. "Securing Implantable Medical Devices Using Ultrasound Waves". en. In: *IEEE Access* 9 (2021), pp. 80170–80182. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3083576. URL: https://ieeexplore.ieee.org/document/9440455/ (visited on 10/15/2024).

[41] M.A. Liker et al. "Deep Brain Stimulation: An Evolving Technology". en. In: *Proceedings of the IEEE* 96.7 (July 2008), pp. 1129–1141. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2008.922559. URL: http://ieeexplore.ieee.org/document/4534851/ (visited on 10/15/2024).

[42] *EFM32 32-bit Low Power Microcontroller (MCU) - Silicon Labs*. en. URL: https://www.silabs.com/mcu/32-bit-microcontrollers/efm32-gecko.

[43] *iCE40 UltraPlus | ML/AI Low Power FPGA | Lattice Semiconductor*. en. URL: https://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40UltraPlus.

[44] *IGLOO® FPGAs | Microchip Technology*. en. URL: https://www.microchip.com/en-us/products/fpgas-and-plds/fpgas/igloo-fpgas.

[45] *iCE40 UltraPlus Family Data Sheet*. en. Datasheet. Lattice Semiconductors, 2020, p. 52. URL: https://www.latticesemi.com/view_document?document_id=51968.

[46] Tinyvision-Ai-Inc. *GitHub - tinyvision-ai-inc/ice40_power: Power analysis of the ICE40UP5K-SG48 devices*. en. 2020. URL: https://github.com/tinyvision-ai-inc/ice40_power.

[47] Justin Miller. *Resonetics Announces Acquisition of EaglePicher Medical Power - Resonetics*. Apr. 2022. URL: https://resonetics.com/news/resonetics-announces-acquisition-of-eaglepicher-medical-power/.

[48] *Commercial Power Solutions | Carefree Battery*. URL: https://www.eaglepicher.com/markets/commercial-power-solutions/.

[49] *Medical Batteries | Resonetics*. June 2024. URL: https://resonetics.com/sensor-technology-medical-power/medical-batteries/.

[50] *MX25V1006F*. en. Datasheet. Macronix, 2017, p. 55. URL: https://www.mxic.com.tw/Lists/Datasheet/Attachments/8673/MX25V1006F,%202.5V,%201Mb,%20v1.0.pdf.

[51] Yu Cai et al. "Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery". en. In: *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. Rio de Janeiro, Brazil: IEEE, June 2015, pp. 438–449. ISBN: 978-1-4799-8629-3. DOI: 10.1109/DSN.2015.49. URL: https://ieeexplore.ieee.org/document/7266871 (visited on 08/24/2024).

[52] *TPS22916xx*. en. Datasheet. Texas Instruments, 2021, p. 29. URL: https://www.ti.com/lit/ds/symlink/tps22916.pdf?ts=1721747294778&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTPS22916%253Fbm-verify%253DAAQAAAAJ_____577mfzTgyS2w6PqVJpduwVNcnVbyI1-E7fzNxhKe_xF-TVMt7K5vwZ5OnDuo-VFokGwDCgdA1LVCcxC0uuWY1nwudjF_ckR3T45vCijDZ4T7guC8DSYQxTwck4QleAmmfmdM5the5XCSrD7-WKB0lbrCiQRrMNUUACFZGC72358lNMOgmklOrT5slYzsx7OyduAlAvyk3sezPWztfMkAbiaVcfWGzBqZ3c1uQJXfuqGS3vUuaTjnf16BDs6CokXLxDzWlGjFksxZgfCVXRkgpjTqxKKLjxYvtyL09nlsRgGNy-9-lmcMc1hyds0gWLq2_vDEZp6Fr9Id1csUHXYlazIckCig-T1yYjQWQQX0W9ubm7arHY-FKhLeLWCSj9KBo1sEXyNnqeETQdD8sbIHw.

[53] *Fluke 114, 115, 116 and 117 Digital Multimeters*. en. Datasheet. Fluke Coorporation, 2020, p. 2.
[54] *EFM32TG11 Family Data Sheet*. en. Datasheet. Silicon Labs, 2018, p. 153. URL: https://www.silabs.com/documents/public/data-sheets/efm32tg11-datasheet.pdf.
[55] *EFM32GG11 Family Data Sheet*. en. Datasheet. Silicon Labs, 2022, p. 262.
[56] *IGLOO Low Power Flash FPGAs with Flash*Freeze Technology*. en. Datasheet AGLxxx. Microsemi, 2022, p. 220. URL: https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/DataSheets/IGLOO+Low+Power+Flash+FPGAs+with+FlashFreeze_Datasheet.pdf.
[57] C. Nicholson and J. A. Freeman. "Theory of current source-density analysis and determination of conductivity tensor for anuran cerebellum". en. In: *Journal of Neurophysiology* 38.2 (Mar. 1975), pp. 356–368. ISSN: 0022-3077, 1522-1598. DOI: 10.1152/jn.1975.38.2.356. URL: https://www.physiology.org/doi/10.1152/jn.1975.38.2.356 (visited on 07/17/2024).
[58] Timothy Olsen. *Current source density (CSD)*. Nov. 2024. URL: https://nl.mathworks.com/matlabcentral/fileexchange/69399-current-source-density-csd.
[59] Ray Beaulieu et al. "The SIMON and SPECK lightweight block ciphers". In: *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2015, pp. 1–6. DOI: 10.1145/2744769.2747946.
[60] Joan Daemen. "The Rijndael Block Cipher". en. In: 1820 (1998), pp. 277–284. ISSN: 978-3-540-67923-3. DOI: 10.1007/10721064_26.
[61] Jian Guo, Thomas Peyrin, and Axel Poschmann. "The PHOTON Family of Lightweight Hash Functions". en. In: *Advances in Cryptology – CRYPTO 2011*. Ed. by David Hutchison et al. Vol. 6841. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 222–239. ISBN: 978-3-642-22791-2. DOI: 10.1007/978-3-642-22792-9_13. URL: http://link.springer.com/10.1007/978-3-642-22792-9_13 (visited on 09/14/2023).
[62] *AN0822: Simplicity Studio™ User's Guide*. en. Manual. Silicon Labs, 2018, p. 40. URL: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf.
[63] *TensorFlow Lite for Microcontrollers*. May 2023. URL: https://www.tensorflow.org/lite/microcontrollers (visited on 08/23/2024).
[64] Mario Vestias and Horacio Neto. "Trends of CPU, GPU and FPGA for high-performance computing". en. In: *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. Munich, Germany: IEEE, Sept. 2014, pp. 1–6. ISBN: 978-3-00-044645-0. DOI: 10.1109/FPL.2014.6927483. URL: http://ieeexplore.ieee.org/document/6927483/ (visited on 05/16/2024).
[65] Arian Maghazeh et al. "General purpose computing on low-power embedded GPUs: Has it come of age?" en. In: *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. Agios konstantinos, Samos Island, Greece: IEEE, July 2013, pp. 1–10. ISBN: 978-1-4799-0103-6. DOI: 10.1109/SAMOS.2013.6621099. URL: http://ieeexplore.ieee.org/document/6621099/ (visited on 11/12/2024).
[66] Matthias Birk et al. "A comprehensive comparison of GPU- and FPGA-based acceleration of reflection image reconstruction for 3D ultrasound computer tomography". en. In: *Journal of Real-Time Image Processing* 9.1 (Mar. 2014), pp. 159–170. ISSN: 1861-8200, 1861-8219. DOI: 10.1007/s11554-012-0267-4. URL: http://link.springer.com/10.1007/s11554-012-0267-4 (visited on 11/12/2024).