Delft University of Technology
Master of Science Thesis in Embedded Systems

Development Of An Energy-Efficient Gaming Platform To Evaluate Novel User Interaction

Alejandro Cabrerizo Martinez de la Puente

TU Delft Delft University of Technology

Development Of An Energy-Efficient Gaming Platform To Evaluate Novel User Interaction

Master of Science Thesis in Embedded Systems

Embedded Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

Alejandro Cabrerizo Martinez de la Puente
A.CabrerizoMartinezDeLaPuente@student.tudelft.nl
contact@alejandrocabrerizo.com

28/08/2023

Author
Alejandro Cabrerizo Martinez de la Puente (contact@alejandrocabrerizo.com)
Title
Development Of An Energy-Efficient Gaming Platform To Evaluate Novel User Interaction
MSc Presentation Date
31/08/2023

Graduation Committee

Dr. Przemysław Pawełczak	Delft University of Technology
Dr. Maliheh Izadi	Delft University of Technology
Dr. Jasper de Winkel	Delft University of Technology

Abstract

In the modern age, the proliferation of electronic devices and their subsequent waste presents an increasing environmental challenge. With the continuing growth of technology and the urgency of sustainability becoming ever more important, there is an ever-increasing need to reimagine how we power and utilize our devices. This has been exacerbated in the last years with the proliferation of electric cars, which need a large amount of batteries to match the range of existing internal combustion cars.

The motivation behind this thesis stemmed from the desire to address these challenges head-on and create a new interactive research platform to help researchers improve user interactivity in a battery-free, energy-constrained system.

In this thesis, I cover the design, implementation, and evaluation of an innovative low-power gaming research platform, in the shape of a portable gaming console. The console's uniqueness is derived from its key characteristics: an interactive nature, a lack of traditional batteries, and a robust system for handling intermittent power.

The term "interactive" has been redefined in the context of a battery-free console, creating a dynamic user-console relationship that allows real-time responses to diverse user inputs and environmental factors. For example, one of the energy harvesting methods of the console, a crank, can also behave as an input to the system, allowing the mapping of in-game actions to crank movements.

The architecture of the gaming console, both in hardware and software aspects, is designed from the ground up to support key characteristics such as ease of use, future expandability, and app portability.

Due to the intermittent nature of the console, the software implementation has been designed around a seamless checkpointing system to handle power interruptions gracefully, where, from the point of view of the user, the game is resumed after a power failure without any apparent loss of progress.

Finally, in order to facilitate user experiments and future work, the console also features an efficient logging system to capture user interaction and system performance data during experiments, alongside an easy-to-use interface that facilitates porting of existing software to this platform.

Preface

In an age of rapid technological advancement, the line between fantasy and reality becomes increasingly blurred. The evolution of gaming consoles, in particular, has been nothing short of extraordinary. From their rudimentary beginnings to the immersive experiences they offer today, these devices have always been a testament to human ingenuity and the relentless pursuit of entertainment.
Moreover, it is our social responsibility to develop products that resonate with the realities of our changing world. This project, and the research that underpins it, is driven by a vision to further enhance the interaction with the gaming experience, while at the same time pushing boundaries to achieve a future where no batteries are needed.

First and foremost, I would like to express my sincere appreciation to my advisors, Dr. Jasper de Winkel and Dr. Przemysław Pawełczak for their invaluable guidance, patience, and unwavering belief in my capabilities. Your expertise and mentorship have been instrumental in shaping this work.
To my friends and family, your encouragement and faith in me have been my anchor during all these years. You've celebrated my victories and offered comfort during setbacks, reminding me of the bigger picture.

Alejandro Cabrerizo Martinez de la Puente

Delft, The Netherlands
8th September 2023

Contents

Preface **v**

1 Introduction **1**

2 Proposed System Architecture **5**
2.1 Gaming Console Architecture 5
2.1.1 System Modularity 5
2.1.2 Energy Storage 6
2.1.3 Energy Path 6
2.1.4 Computing 7
2.1.5 Screen 7
2.1.6 Software Abstraction 8
2.2 Logging Module 8

3 Hardware Implementation **11**
3.1 Main MCU 11
3.2 Video Output 11
3.2.1 Video Interface 12
3.3 Energy Path 13
3.4 Voltage Monitoring and Power Domains 14
3.4.1 Voltage Monitoring 14
3.4.2 Power Domains 15
3.5 Energy harvesting modules 15
3.6 Solar module 15
3.7 Crank Module 16
3.7.1 Custom MPPT System 16
3.7.2 Bidirectional Cranking 17
3.8 Logging Module 19

4 Software Implementation **21**
4.1 System Kernel 21
4.1.1 System Drivers 21
4.1.2 Applications 24
4.2 Checkpointing 25
4.2.1 Checkpoint Creation 25
4.2.2 Checkpoint Recovery 25
4.3 Internal File System and Download Mode 27
4.4 System-wide Settings 28

4.5 Other Built-in Apps . . . 28
4.5.1 Main Menu . . . 28
4.5.2 Settings . . . 28
4.5.3 Testing . . . 28
4.6 Software Porting . . . 28
4.6.1 Doom . . . 28
4.6.2 Game Boy Color Emulator . . . 29
4.6.3 In-Game Features Connected to Energy Harvesting Inputs 30

5 Evaluation 31
5.1 Evaluation Setup . . . 31
5.2 System Power Usage . . . 31
5.3 Energy Harvesting . . . 32
5.3.1 Solar Module . . . 32
5.3.2 Crank Module . . . 32
5.4 Power Path Efficiency . . . 33
5.5 Checkpointing . . . 34
5.5.1 Checkpoint Creation . . . 34
5.5.2 Checkpoint Recovery . . . 34

6 Preliminary Experiments 37
6.1 Experiment Description . . . 37
6.1.1 Gameplay . . . 37
6.1.2 Feedback Process . . . 38
6.2 Objective Data From Logs . . . 38
6.3 User Feedback . . . 39

7 Future Work 41
7.1 Hardware . . . 41
7.2 Software . . . 42

8 Conclusions 43

Chapter 1

Introduction

The portable device industry stands at a unique intersection, constantly battling the demands of cutting-edge performance and the ethical obligations of environmental sustainability. Users expect high-resolution graphics, great interactivity, and long use time, which often necessitates powerful hardware, a significant energy consumption, and shapes the portable device around the battery. In fact, 25% of the volume of a modern portable device is dedicated just to housing the battery [3], and battery degradation is also the most common cause of early obsolescence [5]. The environmental toll of e-waste is becoming increasingly hard to ignore, and it will become a bigger problem in the following years as many batteries in the current lithium battery revolution will reach their end-of-life [25].

Crafting a gaming experience that thrills without leaving a hefty environmental footprint is an intricate dance. Those two factors usually contradict each other, pushing in opposite directions.

With this in mind, this thesis aims to research whether it is possible to use incentives such as a crank in a battery-free, energy-constrained system to improve interactivity and provide a good user experience.

'Interactive' in our context refers to the dynamic nature of the gaming console that fosters a real-time connection between the user and the system. The console is designed to respond to user inputs - such as button pressing, crank rotating, or positional adjustments - creating a constant feedback loop. The user's actions directly influence the console's reactions, allowing for this two-way interaction. This design extends beyond the norm, integrating diverse mechanisms like cranks and incorporating environmental energy sources like solar charging.

To achieve this, a new research platform will be developed from the ground up, investigating novel methods to harvest energy, developing console hardware with minimal quiescent energy consumption, and creating a software stack that abstracts the researcher from the hardware, making porting new software painless.

This software stack will have a 'checkpointing' system, something pivotal to handling the intermittent power that comes with these energy sources. The term 'checkpointing' refers to the process of saving the state of a system at specific intervals. This process allows the system to resume from the last saved state if a disruption (usually a power loss) occurs. In our gaming platform, this

Figure 1.1: **Final version of the console.**

capability is essential for maintaining a seamless user experience despite the power supply's intermittent nature.

All of this will be done with the goal of providing an excellent starting point for future work, where researchers can test new interactive approaches and port new software with ease. All the hardware designs, software, firmware, and documentation created for this project will be released publicly on GitHub [8] with an open-source license. My hope is that this platform will become a stepping stone in future studies regarding low-power gaming systems and interactivity.

Finally, the system will be evaluated to check its real-world performance and a series of experiments will be conducted with real users to evaluate whether the presence of interactive elements and their respective incentives improve the interactivity and gaming experience as a whole.

Related Work

The idea to develop this platform was, in a significant way, inspired by the success of the original "Battery-Free Game Boy" [13] and, to a lesser degree, the "Motion-powered Gameboy" [28]. Therefore, some of the key future work points in those papers were directly addressed. For example, the new screen can now be used under low-light conditions, at the expense of using more energy to keep good visibility outdoors. Other considerations like adding sound or wireless connectivity were chosen to not be addressed due to their high energy consumption requirements.

In addition to that, instead of only creating an improved battery-free gaming console, I choose to pursue a more ambitious approach, developing an entire hardware and software platform that is closer to the current generation of consoles while providing a modular and easy-to-use platform for future research.

For example, as it can be seen in Figure 1.1, the screen has been upgraded immensely, expanding the display area to eight times more than the original Battery-Free Game Boy [13].

	Screen size	Energy harvesting	
		Solar (200 lux)	Eletromechanical
Battery Free GB	1.76”	0.165 mW	1.2 mW
Motion-Powered GB	1.76”	-	5.2 mW
Condenar (This one)	5”	0.174 mW	1400mW

Table 1.1: **Related work comparison.**

The input system has also been updated, it now uses a fully-featured joystick, triggers, and the typical ”ABXY” button configuration that is common in most state-of-the-art gaming controllers.

Lastly, the power harvesting capabilities have been enhanced significantly (as seen in Table 1.1), with support for a modular approach and crank-based interaction.

On average, the new approach can harvest 1000 times more power than the button and solar strategies used in the ”Battery-Free Game Boy” [13] and 250 times more than the bi-stable actuators in the ”Motion-powered Gameboy” [28].

During the development of this gaming platform, several innovative contributions took place:

- **User-focused checkpointing system:** This checkpointing system, apart from being more flexible and easier to use than other implementations [17] [28] [13], offers a user-focused checkpointing approach, where checkpoints only happen after a new frame has been delivered to the user. It removes the need for just-in-time checkpointing [13], making the software stack easier to port to different architectures.

- **Flexible power harvesting approach:** To allow for easier further iteration, the platform supports the use of energy harvesting modules, drastically reducing the time and money it takes to research new ways of powering the system.

- **An innovative crank harvesting approach:** This approach is able to get the best of all worlds, from a custom MPPT system to bidirectional cranking with feedback, all of that while maintaining high efficiency.

- **Easy app portability:** A simplified API allows easy porting and development of hardware. New apps have access to a lot of off-the-shelf tools like system-wide settings, internal block storage, heap management, and a file picker. All these tools are ready to be used and are extremely easy to implement in applications.

Chapter 2

Proposed System Architecture

The motivation for developing this platform was to create an easy-to-use platform that would allow researchers to evaluate the impact of different forms of user interaction in power-constrained systems.

Therefore, the platform architecture is designed with the goal of providing an excellent gaming experience, while at the same time, letting the researcher measure all aspects of the console (system performance, user interactivity, and gameplay experience) without disturbing the performance of the system.

With this in mind, the system is split into two parts, the console and the logging system. This separation is vital for the validity of the experiment results since the presence of the logging module must not change the behavior and performance of the console in any significant way.

2.1 Gaming Console Architecture

The chosen console architecture is shown in Figure 2.1 and Figure 2.2.
The console architecture revolves around the use of a main MCU, that is in charge of both the real-time aspects of the system as well as the necessary computing/emulation to run the games. It is complemented by peripherals that add energy harvesting, button input, and video output among other things:

2.1.1 System Modularity

As one of the goals of this thesis is to create a system that can be easily iterated and improved upon, the hardware of the console is divided into three separate modules:

- **Motherboard:** Connects all the parts of the console together and contains most power-related components and user input (buttons and joystick).
- **CPU module:** The CPU module is an easily replaceable module that contains the main processor and FLASH along with the necessary hardware to output video to a screen.

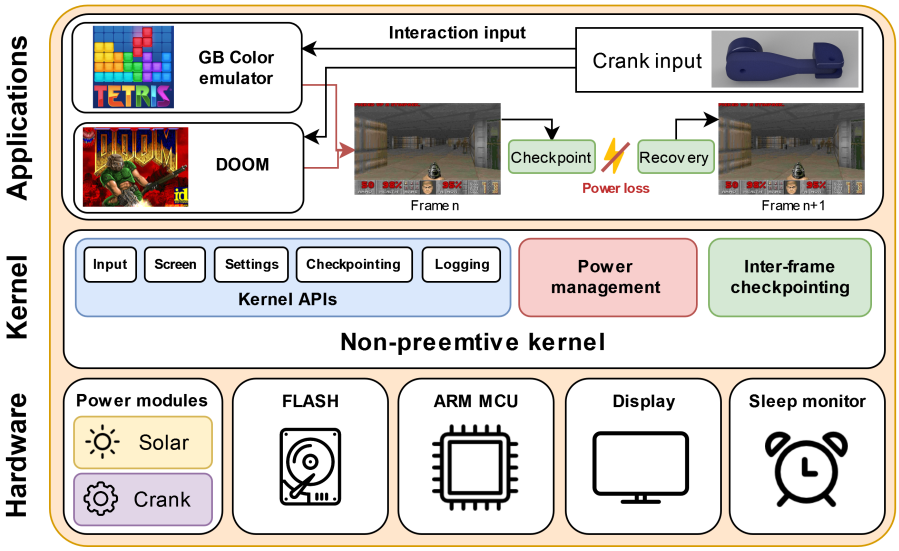


Figure 2.1: **General system architecture diagram.**

- **Power harvesting modules:** These modules are the key to the multimodal energy harvesting nature of the console. Each module is designed to provide power to the console in a different.
  This modular approach also makes it easier for the researcher to quickly try different approaches to energy generation. For this thesis, two example harvesting modules will be created, a solar module and a crank module.

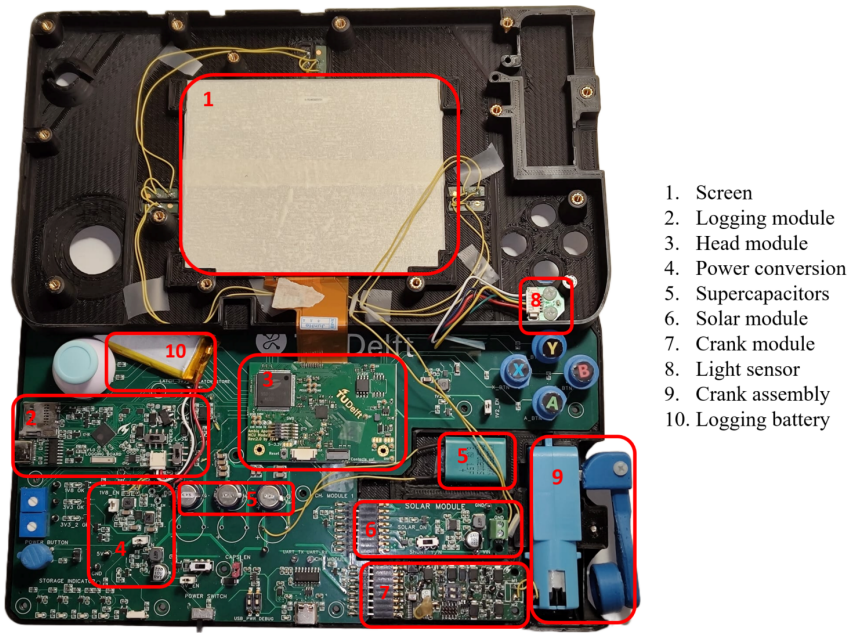
This separation of concerns also has the advantage of facilitating a faster initial development, since improvements in a certain part of the system don't require a whole remanufacturing of the whole system.

2.1.2 Energy Storage

To store this power, a set of supercapacitors will be used to replace the conventional batteries and give the console a good amount of storage. The capacitors used need to have an extremely low ESR (Equivalent Series Resistance), due to the intermittent nature of the system. A high ESR would make the capacitor voltage practically disappear when a lot of current is being drawn from it (when the backlight level is high), effectively reducing its usable size.

2.1.3 Energy Path

Due to the limited energy supply of the console, being as low power as possible is the main constraint of the system. To this goal, the power delivery path is optimized to provide granular control over which part of the system is powered or not.



1. Screen
2. Logging module
3. Head module
4. Power conversion
5. Supercapacitors
6. Solar module
7. Crank module
8. Light sensor
9. Crank assembly
10. Logging battery

Figure 2.2

2.1.4 Computing

The processing power of the console is centered around a state-of-the-art low-power MCU (Apollo 4) [16], which has a modest amount of computing capabilities while being extremely efficient (4uA/MHz). Another key point of this MCU is its extensive combination of peripherals, such as the Octal SPI drivers (where eight data bits are sent on each clock cycle in parallel) that will be used to emulate the screen interface during the implementation.

2.1.5 Screen

There are currently four different types of screens in the market that would work in a research platform like the one designed here:

- **E-INK display:** This type of screen is widely available, uses no energy while static but offers a very limited amount of colors and a very slow refresh rate.
- **MIP display:** MIP (Memory In Pixel) displays offer all the benefits of E-INK (low power consumption) while at the same time, having higher refresh rates [6]. They are unfortunately not as widespread so finding a commercially available MIP screen with the right size/resolution combination is not as straightforward. A LS044Q7DH01 [9] screen (4.4", 320x240) was tested, and even though its power consumption was appropriate for the needs of the project (3.5mW), the user experience was severely affected by its low resolution and it being black/white only.
- **OLED:** OLED screens typically feature high resolutions with the highest power consumption unless dark colors are being displayed. The main

drawback for our application was that OLED screens with low resolution and big sizes are not readely available in the market.

- **LCD:** This is the screen type we selected since it represents a nice equilibrium between usability (8 bit RGB, 60 FPS) and low enough power consumption. The LCD part of it uses 30mW, but it can be reduced down to 18mW if it is run at 30FPS instead. Running at 30FPS also creates some visual artifacts that can be bothersome for some people, but we made the decision that the large energy savings outweigh this issue.
  Nevertheless, LCD screens need a powerful backlight for the screen. Thankfully due to the non-linear way the human eye perceives light [1], it is possible to lower the backlight intensity indoors to negligible levels of power usage (4mW) while still displaying a visible image. Unfortunately, it also means that some kind of automatic brightness adjustment is necessary since it would be otherwise impossible to use it outdoors. Thankfully using the console outdoors usually also means that the increase in power input from the solar panels is roughly proportional to the energy requirements of the backlight to maintain good visibility, therefore partially negating this disadvantage.

2.1.6 Software Abstraction

From the software point of view, as seen in Figure 2.1, the software stack is made of a low-level Kernel, designed to be as lightweight as possible, with a set of Application Layer Interfaces (APIs) that provide a layer of abstraction over the hardware.

The Kernel itself behaves as a non-preemptive scheduler (where the apps hold the execution until given back to the kernel or forcefully terminated), that switches between apps and manages the system resources (storage, heap). Apart from that, it also handles the initialization and upkeep of the underlying hardware, the checkpointing system, and optimizes the power usage by powering down unused parts of the console.

The Kernel APIs are split into five different parts: Input, Screen, Settings, Checkpointing, and Logging. Each of these parts can be enabled or disabled at compilation time, effectively changing the memory footprint of the Kernel. Each one of those APIs allows hardware-agnostic access to the inputs and outputs of the system.

2.2 Logging Module

As it was mentioned before, the logging subsystem, which is in charge of doing real-time measurements of all aspects of the console, needs to behave as an independent system.

If the presence of the logging system were to alter the behavior of the console, the experiment measurements would not be valid.

This is why, a different, independent system is used to record log files for experiments. The independent nature of this subsystem includes the use of its own power supply and isolated inputs, which makes sure that the presence of logging does not affect the rest of the system.

To incentivize this separation further, no communication is allowed from the logging module to the main system. This way, the main system is completely agnostic to whether logging/recording is taking place and therefore, its behavior will be exactly the same no matter whether logging is taking place or not.

Once an experiment is finished, the stored logs can be retrieved and analyzed offline to gain insights into the system's operation. This is valuable for iterative system improvements, understanding user behavior, and diagnosing any system errors. The architecture supports an interface that allows easy retrieval and interpretation of these logs by researchers.

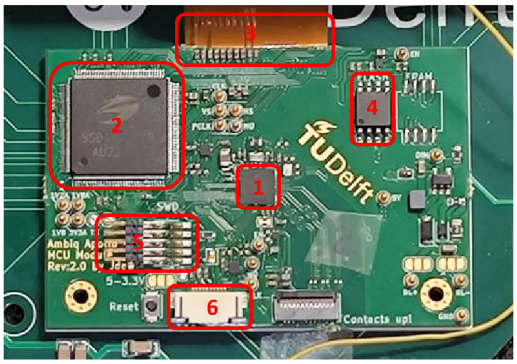
Chapter 3

Hardware Implementation

In a gaming system, the hardware serves as the bedrock upon which all gaming experiences are built, and it is therefore extremely important to choose a solid hardware foundation that would not be a limitation for further development. In this chapter, I will explore in detail each part of the hardware, comparing the different solutions for each one and the challenges that they present.

3.1 Main MCU

As the main MCU of the system, we are using an Apollo 4 from Ambiq [16]. This series of MCUs has extremely low power consumption while having a lot of processing power, memory, and peripherals. It can run up to 192MHz and has around 2MB of RAM and 2MB of non-volatile memory.



1. Apollo 4 MCU
2. SSD1963 Display controller
3. RGB cable to screen
4. Low-power 64Mb Flash
5. SWD programming connector
6. Low-power display connector

Figure 3.1: **Console Architecture**

3.2 Video Output

The video output of a portable console, as described in Section 2, is likely its most important component from the perspective of the user. Unfortunately, it is also potentially the largest single energy consumer in the system. Therefore,

choosing the right technology for the system is critical for achieving a good performance-to-usability relationship.

3.2.1 Video Interface

Choosing the right video interfaces to transmit the pixel information from the CPU to the display is also critical for achieving low power consumption. In today's era of high-resolution and high refresh rate displays, MIPI's DSI [2] protocol is used. The differential nature of DSI (two wires per signal) makes it nearly immune to external interference and can provide extremely high data rates with a small number of signals [2]. Unfortunately for us, high-speed differential signals, also known as Low Voltage Differential Signaling (LVDS), require the use of transceivers that use a lot of power. For example, a state-of-the-art single channel, low power LVDS driver (DSLVDS1001 from TI [21]), uses 23mW. Since a MIPI display interface needs at least two LVDS pairs (clock and signal), using a MIPI DSI interface would add at least 50mW to the overlay system power consumption.

On the other hand, one can use a parallel interface, where instead of pushing all the bits through the same signal, one can have as many signals as bits each pixel needs. In a typical 32-bit color system, this would mean having 32 times more signal wires going to the display, but each one of them only has to carry 1/32 of the data compared to a MIPI DSI interface. This drastic reduction in data rate makes it possible to use common single-ended (where only one wire is used for each signal) transceivers instead of LVDS signal pairs. This also has the advantage of reducing the parallel interface wire count by a factor of two, since all the signals are now single-ended.

Therefore, the final "cost" of using a parallel interface vs a MIPI DSI one is having sixteen times more wires, which is justified in our application due to the power savings.

Unfortunately, it is not practical to output video through a parallel interface from an MCU, since it would require nearly 35 pins (32 signals, a clock, and two synchronization signals that mark the end of a row and a frame (VSYNC, and HSYNC respectively). MCUs usually don't have any dedicated peripheral capable of streaming out this many bits at a time.

To accommodate for this limitation, an SSD1963 display controller from Solomon Tech [15] is used. This IC is located in between the MCU and the screen and accepts pixel data in a 6800 interface and outputs it to a 32-bit wide parallel interface. The 6800 interface only uses 8 data wires and two more slow signals to indicate the type of communication that is taking place (write/read and command/data). This allows us to use the MCU SPI peripheral in its 8-bit wide configuration (which outputs bytes 8 bits at a time) to automatically output a full frame of data with no CPU intervention. Nevertheless, the usage of the SSD1963 adds 1.8mW of power consumption, which is comparable to the power used by the MCU but an order of magnitude lower than the previous estimations for a MIPI DSI display.

The screen that was chosen was the COM50H5M81XLC from Sharp, a 32-bit 5" Parallel LCD screen that features good low light performance [7]. The low pixel density of this screen (resolution divided by area), further optimizes our design, making it possible to have a large screen while reducing the amount of

computing needed by lowering the resolution.

3.3 Energy Path

One of the advantages of this research platform is that due to its low power consumption, can be powered by supercapacitors instead of conventional batteries. In general, current lithium-based batteries are vastly superior to supercapacitors in all performance-related metrics [11]. Energy density and power density are significantly superior in conventional batteries whereas supercapacitors have the advantage of having a nearly unlimited number of recharge cycles, and therefore, not having a limited lifespan that generates e-waste.

A 6V 1.5F SCMR22H155PSBB0 [10] capacitor from Kyocera was chosen due to its low ESR (580mOhm). This is in contrast with the KR-5R5H474-R cell capacitors from Eaton, which have 3x more energy density but a significantly higher ESR (50Ohms) [4]. Even though the Eaton one are denser, its high ESR means that at our expected power draw at full backlight (30mA), they would be dropping by nearly 1.5V, also reducing the overall system efficiency down to 0%-30% (depending on the capacitor voltage). In contrast, with the SCMR22H155PSBB0 [23] capacitor, its worst-case efficiency is less than 2%.

A 6V 1.5F supercapacitor can store up to 20J of energy, which at our expected power consumption of 125mW would result in 160 seconds of system life. This number was chosen as a compromise between needing constant power input and not incentivizing enough interaction from the user.

Energy can be added to the system by three different methods:

- **USB port:** The main USB port of the console can be used to charge the capacitors. The rate of charge is limited to 500mA to comply with the USB specification. It is also useful for debugging purposes to disable this charging feature but keep logging enabled. This can be selected by the researcher through a set of hardware switches next to the USB port.

- **Logging module:** It is likely that in some variations of an experiment, the researcher might want to fully disincentivize the user from interacting with the energy harvesting inputs, so the logging module can also keep the console charged at all times. To allow for faster experiment turn-around and ease of use, this feature is software controlled by the main MCU, allowing for quick transitions between experiments.

- **Power modules:** As part of the modularity of this platform, you can plug in up to 2 different power modules to the console. These modules have a standardized size (60x22mm) and pinout and allow for quick iteration of new power systems. As part of this thesis, two power modules were developed, a crank-powered and a solar-powered one.

To facilitate maintaining a constant regulated voltage, no matter the state of charge of the capacitors, a low-power boost converter LTC3526 [19] from Analog Devices is used. It has 9uA of quiescent current, which is negligible compared to the self-discharge current of supercapacitors. This boost converter generates an "always-on" 5V rail, from which most power monitoring systems (auto wake-up, power latches) are fed. All the other system voltages are then created from this one following the hierarchy in Figure 3.2. This hierarchy is designed to provide

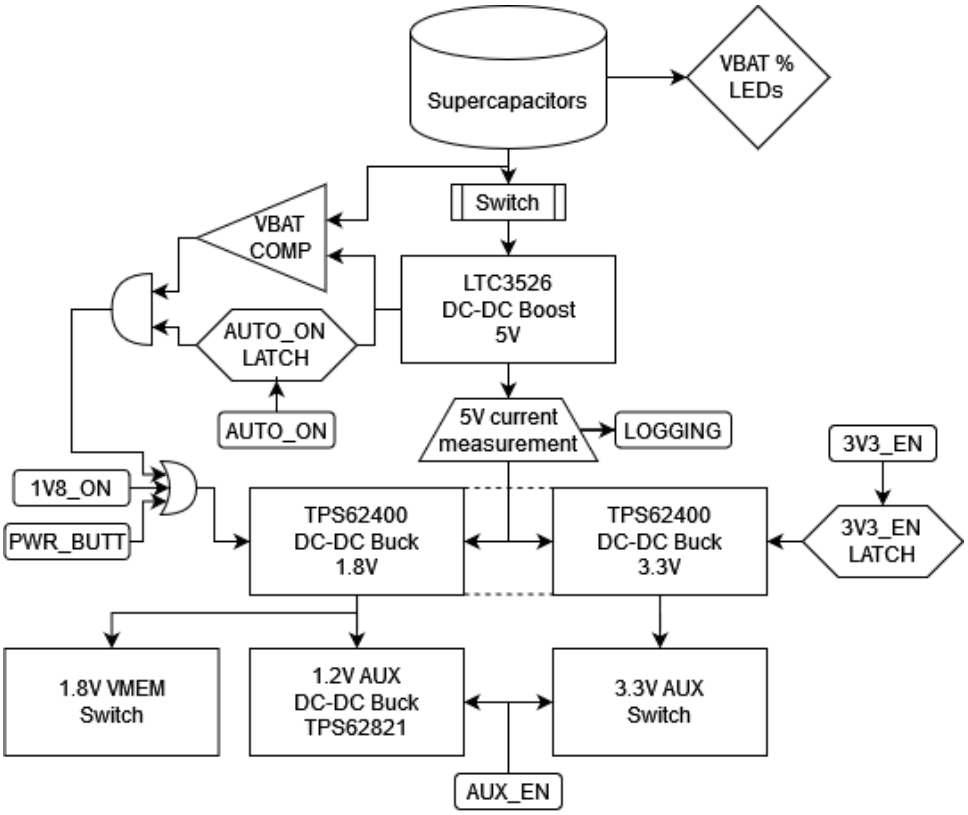


Figure 3.2: **Energy path structure.**

full control over which parts of the system are enabled, which allows to save as much power as possible if a component is not in use, while also providing a good compromise between granularity, efficiency, and system complexity. In Figure 3.2, arrows indicate the direction of the flow of energy and data, whereas dashed areas delimit parts of the design that are implemented with a single off-the-shelf component. Power converters and power switches are represented by rectangles and signal latches by diamonds. All other symbols are direct references to common electrical symbols, such as comparators and logic gates. All the inputs (rounded rectangles) and the output power rails are explained in more detail in Table 3.1.

3.4 Voltage Monitoring and Power Domains

3.4.1 Voltage Monitoring

As previously explained, the 5V rail of the console is always on. The power monitoring systems that are fed from it take care of powering back on the console once the energy storage has enough charge.

In a usual scenario, once the console detects that the amount of energy stored is running low, it can choose to finish any remaining work (such as displaying the video frame it is currently processing), enable the monitoring systems, and

Power rail	Parts powered	Control
5V	voltage monitoring, LCD, 3.3V, 1.8V	Always on
1.8V	main MCU, input, 1.2V, 1.8V-MEM	User/Auto/MCU
3.3V	Low power screen, 3.3V-AUX domain	MCU, latched
3.3V-AUX	LCD driver, LCD	MCU
1.2V-AUX	LCD driver	MCU
1.8V-MEM	external FLASH and SRAM	MCU

Table 3.1: **Power domains.**

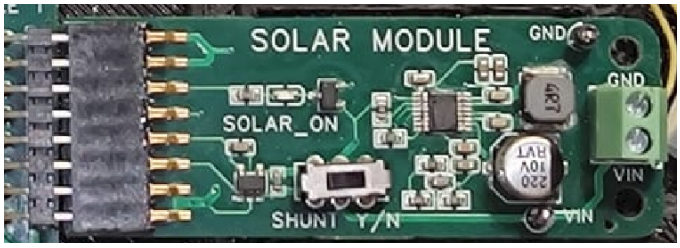


Figure 3.3: **Solar harvesting module.**

finally turn itself off. If the automatic power-on system was enabled before shutting itself off, once the super-capacitor voltage reaches a certain pre-programmed level, it will simulate a "power-on" button press, powering on the main CPU in the process. Once the CPU is enabled, it can choose to latch the power-on status and therefore start this cycle all over again, or just go to sleep again if, for example, the amount of energy stored is not high enough.

3.4.2 Power Domains

In order to save as much power as possible, the system is divided into 5 different power domains. Each of them can be individually controlled by either the CPU or the hardware-based voltage monitoring system:

3.5 Energy harvesting modules

The current implementation of the platform supports up to 2 power modules installed at the same time. Each module is responsible for charging the capacitors when power is available and if desired, sending data to the main MCU about the state of the module (for example, which direction and how fast the crank is rotating).

3.6 Solar module

The solar module (Figure 3.3) is a typical solar energy MPPT buck-boost harvesting configuration that can handle solar panels up to 15V and has current monitoring built-in. It also tracks the most efficient power point for the solar

Figure 3.4: **Crank harvesting module.**

panels (MPPT), which is usually at around 80% of the open-circuit voltage of the solar panels.

This module is built around the LTC3129 IC from Analog Devices and with the current solar panel configuration, it generates between 1.5-125mW depending on the amount of light. On a sunny day, it is therefore capable of powering the console by itself, or at least achieve a good duty cycle thanks to the intermittency capabilities of the platform.

The solar panels chosen for this platform are the EXL10-4V170, some state-of-the-art panels specifically designed for low-light conditions. As noted before, a parallel array of 8 of these panels (50x20mm each) can produce 1.5mW under 200lux, your typical indoor lighting conditions.

These panels, as seen in Figure 1.1 are placed in a way that avoids parts of the console where it is likely that the hands of the user would cover or shadow during normal gameplay.

The output current from the panels can then be optionally amplified by an INA180 amplifier from TI, so that the logging system can easily record it.

3.7 Crank Module

During the development of this thesis, the cranking module (Figure 3.4) went through a lot of iterations. At first, the approach was similar to the one used in the solar module, a simple boost converter capable of taking the 2-4V from the motor input and charging the supercapacitors at 5V. Nevertheless, this approach is hugely inefficient. When a motor is used as a generator, it has a characteristic current vs voltage curve that determines at which % load the generator operated more efficiently. Since efficiently harvesting energy from a motor is not as common as doing it from a solar panel, there are no off-the-shelf solutions that offer a good Maximum Power Point Tracking (MPPT) system for motor inputs. With that in mind, a custom MPPT tracking system was developed.

3.7.1 Custom MPPT System

The custom MPPT tracking system revolves around a small, very efficient MCU (NHS3152 from NXP), that controls when a buck-boost converter needs to be enabled for the crank to operate at its most efficient speed and load. The MCU constantly monitors the input voltage, periodically disconnects the converter for a small amount of time to measure the open circuit voltage, and runs a custom

algorithm that makes a decision on whether the converter should be enabled or not in order to stay in the most efficient range. Since this MCU is powered by the converter output itself, it also needs to make sure that it periodically enables the converter so it can recharge a series of input capacitors that keep the MCU alive while the converter is not running.

3.7.2 Bidirectional Cranking

The power input of this module is a conventional DC brushed motor, so the polarity of its output voltage changes depending on the direction of rotation. This makes it necessary to either block the inverse voltage (not allowing cranking in one direction) or rectify the input power before delivering it to the buck-boost converter. Therefore, there are different approaches that can be taken to address this situation:

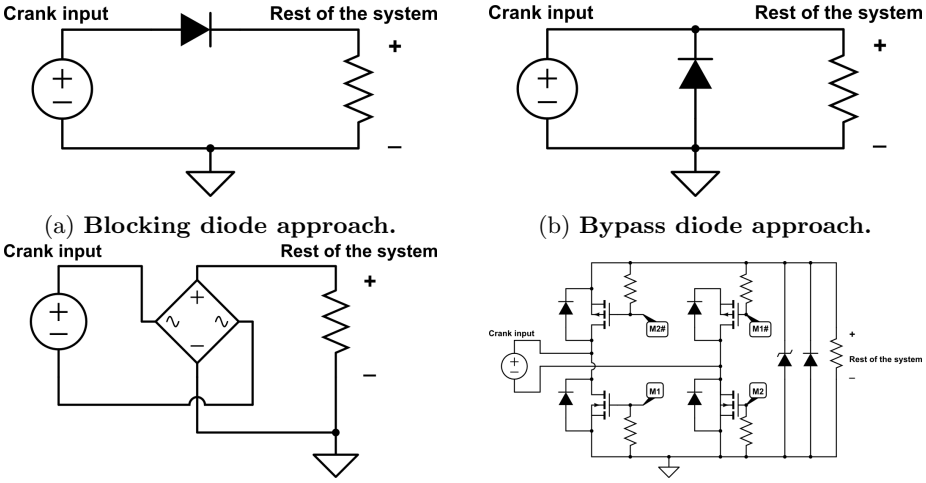
Blocking diode

This is the simplest solution, where a diode is placed in series with either one of the motor terminals (See Figure 3.5a). Due to the reverse current-blocking nature of a diode, it will block current flow in one direction, creating an open circuit from the point of view of the motor. With this approach, if the user tries to crank if the wrong direction, the crank will offer no resistance, letting the user turn it as quickly as possible. After testing, this was deemed the correct approach, since it feels like no work is being done and the user will quickly start cranking in the other direction. Unfortunately, due to having a diode in series with the circuit, part of the energy will be lost in the forward voltage of the diode. Since the motors used output anywhere from 1-4V under load, the 0.7V drop from the diode will result in only 30-82.5% efficiency, depending on the cranking speed. A Schottky diode can also be used here, since it has a lower voltage drop (0.35V), resulting in higher efficiencies (65-91.25%). Nevertheless, Schottky diodes have the downside of having significantly increased reverse current and lower blocking voltage, which under certain conditions (such as fast reverse cranking, where there is no load, so easy to achieve) can damage the electronics.

Inverse nypass diode

This approach (Figure 3.5b) is similar to the blocking one, but the placement of the diode is slightly different. In order to avoid the voltage drop across the diode, the diode is put in parallel instead of in series to short the motor when it is being cranked in the wrong direction. This makes this solution nearly 100% efficient but has two downsides:

- **Crank feeling:** Due to the diode behaving as a short-circuit when the crank is being rotated in the wrong direction, the user will feel like they are doing a lot of work, furthermore incentivizing the cranking in this (wrong) direction. If this continues for a certain period of time, it is also possible for the diode to overheat and burn, since all the cranked power is dissipated in the diode itself.



(a) **Blocking diode approach.** (b) **Bypass diode approach.**

(c) **Bridge rectifier approach.** (d) **Ideal full-bridge approach**

Figure 3.5: **Crank harvesting approaches.**

- **Residual negative voltage:** Due to the nature of a diode, even if a diode is in its conductive region, it will still have a small voltage drop across it. This means that when the system is cranking in the wrong direction, and the diode, therefore, behaves as a theoretical short, there will still be a small voltage across it. This means that it won't be able to clamp the negative voltage to 0V, but to its forward voltage. If a Schottky diode is used, then there will be around -0.3V, which might be big enough to damage certain sensitive components or affect the overall long-time reliability. In testing performed with this approach, this small negative voltage was not large enough to break any hardware, but long-term effects are unknown.

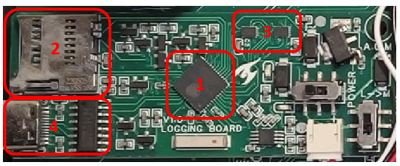
Signal rectification

The goal of this solution is to provide a way to allow bidirectional cranking, which might be useful to add certain interactions during gameplay. For example, we used this to control how the pieces in the game "Tetris" were rotated, adding even more incentive to use the energy-harvesting features of the platform. This approach uses a full bridge rectifier, as seen in Figure 3.5b, to provide two symmetrical rectification paths for the input current. Unfortunately, with this approach you always have two diodes in series with your circuits, resulting in twice the voltage drop as with the original one. Even with Schottky diodes, the efficiency of the approach is quite bad, in the 30-82.5% range, depending on cranking speed.

The chosen solution - An ideal bridge rectifier with MPPT support

The chosen solution shown in Figure 3.5c is based on the "Signal rectification" approach, where a full bridge rectifier with bypassing MOSFETs is used.

In this approach, each diode of the bridge rectifier has a complementary MOSFET which is off by default and can be controlled from the MCU of the mod-



1. ESP32 MCU
2. Micro SD card holder
3. Digital isolators
4. Charging/programming USB

Figure 3.6: **Logging module.**

ule. By enabling a MOSFET, its respective diode is bypassed, removing the voltage drop and only leaving a small resistance in place (The RDSon, resistance of the MOSFET when conducting, 35 mOhm). This makes this approach highly efficient, 98-99% (not taking into account the energy used by the MCU, which is already there to provide MPPT). This approach requires a companion MCU, which is responsible for detecting the current cranking direction, and then switching the correct set of two MOSFETs on to enter the "ideal" mode. This can be risky, if, for some reason, a bug in the code or external interference, the wrong combination of MOSFETs is turned on, and a large negative voltage might be created in the input, damaging the whole system. It is especially important for the MCU to be constantly monitoring the motor inputs since the user might change the direction of cranking at any time.

Another necessary protection mechanism is that, if the motor voltage is too low (slow cranking), it is safer to disable all MOSFETs, since it is more likely that a change in direction will happen and it could cause damage if te system can not detect it fast enough.
Finally, it is possible to add a diode in reverse (as explained in the "Inverse bypass diode" approach) after the ideal bridge rectification. This joint approach makes sure that, if for some reason, the ideal bridge implementation is faulty and accidentally generates a negative voltage, the reverse diode will short it and clamp the power rail to its forward voltage. This is especially useful during development to avoid costly mistakes, but it can also be left there after development since it is not a detriment to the efficiency, it just wastes a small amount of area.

3.8 Logging Module

The logging module (Figure 3.6) is a compact module that can be used to measure and log different system parameters without affecting the behavior of the system being measured. Its main use case is to provide logging capabilities to study the behavior of the user and the system during controlled experiments. It is built around the wireless ESP32 MCU, which is a powerful yet cost-effective solution to sample all the necessary logging data channels while supporting WiFi and Bluetooth connectivity for future expandability. It is powered by its own battery to avoid interfering with the game console itself. This along with having isolated inputs coming from the rest of the system, ensures that the presence of this module has no effect on the rest of the platform. All the data that the module samples are stored in a microSD card for easy extraction.

A communication link exists between the main system and the logging module. In order to prevent the module from having an effect on the main system,

all communication happening between the both (to log extra things) is only one way, from the system to the logging module. This way, there is no way the logging module could affect the behavior of the main MCU. As part of this communication channel, the system can tell the logging module to start/stop an experiment and can log up to 8 extra channels of data, to be used at the researcher's discretion

Chapter 4

Software Implementation

In this chapter, the layers of software that abstract the hardware in our console are unpacked, detailing the mechanisms and design philosophies that govern its operation and provide an easy-to-use interface to the researcher and end-user.

4.1 System Kernel

The system kernel is built as a non-checkpointed (the kernel state is not saved by the checkpointing system), non-preemptive (it is up to the application to return control back to the kernel) lightweight abstraction layer on top of the hardware to make porting applications easy.
The system kernel is not checkpointed since the amount of time it takes to initialize the kernel each time (2.25 ms, as can be seen in Figure 5.6) is insignificant compared to the time it takes to get the hardware ready during startup. This decision also simplified the checkpointing process, reducing the amount of data stored in each checkpoint.
Since this kernel is not checkpointed, during startup it is in charge of initializing the hardware and kernel component, restoring the last checkpoint, and finally giving the execution back to the application itself. Due to the non-preemptive nature of the kernel, it uses the API calls that the application calls at the end of each frame to do the necessary housekeeping tasks such as checkpointing and drawing charge level indicators and other interface objects on top of each application. The kernel also includes other things like a power failure-safe flash file system, file picker UI, settings menu, home menu, and a download mode to upload files to the console, which will be explained later in this chapter.

4.1.1 System Drivers

To abstract and simplify the porting process for new applications, each system component has its own driver, with a user-friendly API. This also allows supporting different hardware and platform revisions with no changes from the researcher's point of view.

Graphics

The graphics driver is implemented as a hardware-agnostic abstraction layer, and it handles queuing draw commands, double buffering, and vertical synchronization of the screen. To display a new frame on the screen, an application can queue draw commands to update the information being shown, and then let the kernel know that the current frame is done drawing. The graphics driver handles the delivery of new frames to the LCD controller, making sure that it is only done when the screen is not being refreshed, to avoid screen tearing issues. The LCD controller uses the 6800 interface, which is not usually directly supported by any peripherals in modern MCUs. A typical 6800 interface uses one clock, 8 data, a read/write, and a data/command signals, where before any packets are transferred, the read/write and data/command signals are set to the correct value to indicate the type of operation being carried out and then left in that state while the 8 data lanes and clock transfer the data one bit per clock cycle. This simplicity made it possible to repurpose one of the MSPI peripherals in the MCU to handle most of the work. If one manually sets the read/write and data/command pins manually and then starts a normal MSPI transaction, you can use the Direct Memory Access (DMA) features of the MSPI peripheral to do most of the transaction in the background. This frees up the CPU to start processing the next frame while the current one is being sent to the display controller buffer.
A double-buffered approach is used to let the CPU work while the last frame is being sent. Unfortunately, double-buffering has the drawback of using a lot of memory. Each frame (320x240 pixels, 32bit color) uses 2.5Mbit of memory, and since we have two of them, nearly 5Mbit of fast RAM is used just to allocate the framebuffers.
If the researcher porting a game needs this extra RAM, this double buffering system can be easily disabled, switching back to using only 1 buffer, or even no buffers at all (directly drawing to the display). Each of these approaches is slower than the one before, so unless this extra RAM is absolutely needed, it might be a better idea to try optimizing the RAM usage of the app in particular instead.

Due to the impossibility of saving and restoring these framebuffers (due to their size) on checkpoints, a checkpointing system that happens at the end of each frame is utilized. This lets us avoid storing the contents of these buffers, freeing 10Mbit of FLASH (since we also double-buffer our checkpoints) that can be used in porting a larger number of apps or more complex ones.

Screen

Each screen driver is responsible for translating the low-level commands coming from the graphics driver so that they can be applied to the physical screen. For example, the screen driver is responsible for abstracting things like setting the screen brightness, contrast, and image rotation and sending either full frames or parts of a frame. Many of these features are designed to be optional, and if any of them is impossible with a certain hardware configuration, the system will automatically (during compilation) best adapt to what is available.

Crank Module

As explained in the previous chapter, the crank module has a MCU on it that is responsible for tracking the most efficient point of the crank motor and making the bridge rectifier ideal. The firmware that runs on the MCU, as explained in Algorithm 1 evaluates these parameters at 10K̇Hz and takes the appropriate decisions to maintain maximum efficiency.

Algorithm 1 Sense Motor Direction and Control Bridge.

```
while true do
  Measure sense motor direction sense inputs
  if both sense inputs are high OR both are low then
    Turn off ideal bridge {Disable the bridge if both sense inputs are similar
    because this is an unknown state}
    Return {Exit the loop}
  end if
  if only one is high then
    if Input1 is high then
      Enable bridge in direction 1
    else
      Enable bridge in direction 2
    end if
  end if
  Write to DAC to disable the DCDC converter {Disable the DC-DC con-
  verter to take an unloaded sample}
  Wait 100u̇S {Wait for the input voltage to stabilize}
  Take voltage measurement from the MCU supply capacitor
  if MCU voltage < 3V then
    Wait 10ṁS {Avoid shutting down by waiting additional time}
    Continue {Proceed to the next loop iteration}
  end if
  Read input voltage from correct sense input {Use MPPT algorithm to
  maintain the input voltage in its most efficient point}
  if loaded measurement is less than 50% of the unloaded measurement
  then
    Enable the DCDC converter by writing to the DAC
  else
    Continue {Do nothing and proceed to next loop iteration}
  end if
end while
```

Flash

The Flash subsystem is split into two parts:

- **Internal Flash driver:** This driver is in charge of providing a hardware-independent abstraction of fast, persistent storage. Its goal is to also provide seamless byte-level access to this memory, removing size and alignment requirements.

- **External Flash driver:** This driver has a scope similar to the internal flash one, but it also has to deal with mapping the external flash memory so that it can be accessed by the typical read/write CPU instructions. In our MCU, this is implemented by using the Execute in Place (XIP) capabilities of an MSPI peripheral. Unfortunately, due to the limitations of the MSPI implementation in the Apollo 4 (in contrast with the Apollo 3), XIP data read operations are not cached by the internal CPU cache. This lack of caching reduces its performance significantly, so a configurable 4-way associative cache is also implemented in software that allows speed-ups of up to ten times in common memory operations:

4.1.2 Applications

This Kernel uses the concept of "Apps" to provide modularity and limit dependencies between parts of the system. Each app in the system is independent from each other and they have no way of interacting with each other other than using the Storage subsystem. On registration, an app must provide the following details:

- **Name:** A name that will be used for logging and in the main menu.
- **Entry point:** Entry point where execution of the app will start from.
- **Checkpoint callback (Optional):** If the app wants to perform some action specific for recovering from a checkpoint (such as reloading/caching data that is not checkpointed), it can register a callback here.
- **App icon:** A path to an icon contained in the internal file system.
- **Storage size:** Amount of storage that the app is requesting to have available. Depending on availability, the system might not be able to allocate it and all storage-related calls from this app will be ignored.

Even though each app is completely independent (it can freely use the heap and storage assigned to it as much as it wants without interfering with other apps in the system), only one app can be loaded into memory at the same time.

Storage

As explained before, each app can request to have available a certain chunk of non-volatile memory where it can store arbitrary data. The maximum amount of storage space in the system is designated at compile time, and the Kernel will try to (at runtime) fulfill as many allocation requests from apps as possible. If a chunk of storage space is successfully allocated, it will persist in the non-volatile memory until the app that uses it is removed from the system. Due to the fragmentation that this removal process creates, the system might choose to move storage chunks around to compact them and make space for new ones.

Input

The input system is also handled by the Kernel to abstract changes in the button layout amount different versions of the hardware. It currently supports up to 8 buttons and two joystick channels. It also additionally also supports up to 3 interaction inputs.

4.2 Checkpointing

The checkpointing system is designed to provide a seamless power-down recovery experience for the user of the system. The checkpointing system is also double-buffered to ensure that, if a power-off happens during checkpointing, there will always be at least one valid checkpoint to recover from.
The kernel has intentionally been architected in a way where it is totally state-free, and it, therefore, doesn't need to be stored in the checkpoint. This, in exchange for extra initialization time, provides faster checkpointing since the amount of memory that gets checkpointed is reduced significantly.
When a firmware update is performed, it is generally a good idea to fully erase the checkpoints to avoid invalid checkpoints. This invalidation is currently done at the tooling level when new firmware is updated by a programming probe.

4.2.1 Checkpoint Creation

During the checkpointing process, interrupts are disabled to ensure the consistency of the checkpoint. First, as seen in Algorithm 2, the application heap is stored. Modifications in the application heap are tracked through the use of the memory protection peripheral found in the ARM core of the MCU. It is based on the MPatch implementation from the original Battery-Free Game Boy [13]. Once the heap is stored, all the designated linker sections (.data and .bss), which is where global C variables used by the kernel and apps are in memory are also copied to the checkpoint. Finally, the stack and current register values are also stored in the checkpoint. Since any C code running here could potentially modify it, this part is written in assembly, taking special care to avoid overwriting any of the registers. Finally, since the non-volatile memory we are using has limited erase cycles, checkpoint creation is limited to only happenning while the console is active (when there has been user input in the last 5 seconds) and could be configured to also skip checkpointing frames if for example, the energy storage % is high (so less risk of a turn-off)

4.2.2 Checkpoint Recovery

When the MCU is booted, it initializes the hardware present in the console and then checks if there is a valid checkpoint available to recover from it. If there isn't, it will just launch the "Main Menu" application. On the other hand, if a checkpoint exists, it will disable interrupts, and fully restore the heap and the linker sections. Then, in order to avoid modifying any registers or the stack, some assembly code carefully restores both the registers and the stack and performs a jump to the code location right after creating a checkpoint. This way, the system has no idea that it has recovered from a checkpoint, from its point of view, it just kept running after saving a checkpoint. This process is detailed in Algorithm 3

In order to add more extensibility to the system, applications can have a special callback that is called after recovering from a checkpoint (but before returning the control) so that it can perform application-specific tasks. For example, it could be used to preload/cache data into the faster, non-checkpointed SRAM.

Algorithm 2 Algorithm used to create a new checkpoint.

1: am_hal_interrupt_master_disable() {Disable interrupts}
2: k_MemWatcher_GetModifiedRegions(&modifiedRegions) {Get modified regions}
3: k_MemWatcher_ResetRegions() {Reset memory regions}
4: newCheckpointID = (lastCheckpointID == 0 ? 1 : 0) {Get new checkpoint ID}
5: k_markCheckpointAsInvalid(newCheckpointID) {Mark checkpoint as invalid}
6: k_ppr_checkpointHeap(newCheckpointID) {Store heap state}
7: k_ppr_checkpointLinkerSections(newCheckpointID) {Store linker sections}
8: *ASM Commands for Register Storing* {Store CPU registers}
9: *ASM Commands for Stack Copy* {Copy the stack state}
10: k_markCheckpointAsValid(newCheckpointID) {Mark checkpoint as valid}
11: *ASM label "restorePoint:"* {Label for restoring the state} {The checkpoint recovery process will jump here when done to continue execution}
12: am_hal_interrupt_master_enable() {Enable interrupts}
13: *ASM Command to flush memory*

Algorithm 3 Algorithm to recover from a checkpoint.

1: {Check if checkpoint is valid}
2: **if** !k_isCheckpointValid(lastCheckpointID) **then**
3: **return**
4: **end if**
5: k_WaitForPendingScreenFrames() {Wait for pending screen frames}
6: am_hal_interrupt_master_disable() {Disable interrupts}
7: MemWatcher_SetEnabled(false) {Disable memory watcher}
8: k_ppr_restoreHeap(lastCheckpointID) {Restore heap}
9: MemWatcher_SetEnabled(true) {Enable memory watcher}
10: k_ppr_restoreLinkerSections(lastCheckpointID) {Restore linker sections}
11: *ASM Command to flush memory* {Memory flush}
12: *ASM Commands for Stack Restoration* {Restore stack}
13: *ASM Commands for Register Restoration* {Restore registers}
14: *ASM Command to jump to restorePoint* {Jump to restore point in 2}

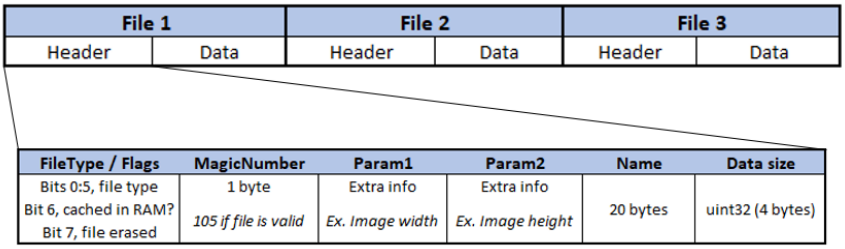


Figure 4.1: **Block format used in the internal file system.**

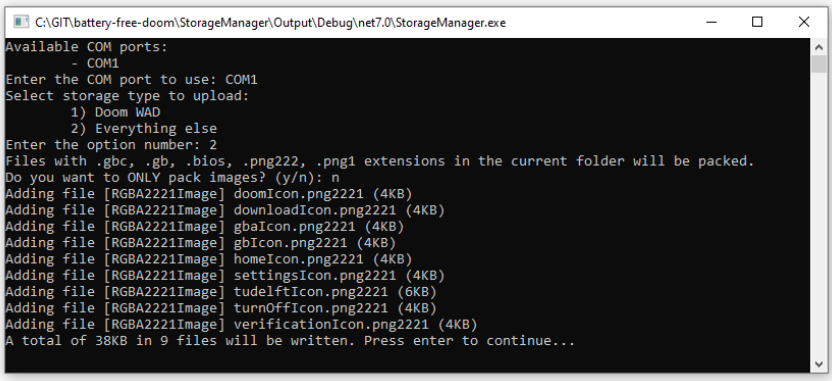


Figure 4.2: **Desktop app used to upload new files to the file system.**

4.3 Internal File System and Download Mode

Since the internal non-volatile memory of the MCU is not large enough to accommodate most game ROMs, an external Flash chip is used.

A small and efficient block-based file system, as described in Figure 4.1, has been developed to allow for files to be stored there. Currently, this file storage contains all icons/pictures the UI has, resource files for applications, and any game ROM files that might want to be used by any of the ported emulators. Another advantage of this system is that it automatically makes use of the custom flash caching system explained before in Section 4.1.1 for certain abstractions such as drawing pictures to the screen since the same memory is read on each frame.

Files in this file system can be changed by connecting the console through a normal USB cable to a computer and running a small client app (Figure 4.2 that organizes, prepares, and uploads files. The main reason for this is to allow a quick and easy way to load new ROMs and icons for faster iteration cycles. Finally, any app can request the kernel to show a file-picking dialog to let the user choose which file to open, providing useful features such as pagination, scrolling, and filtering by file type.

4.4 System-wide Settings

The kernel implements a set of system-wide settings, which can be read and written at any time by any app. The user can also change them by using the "Settings" application in the console. These settings are especially useful for selecting different hardware configurations and allowing the researcher to have different experiment setups/variations. These settings are persistent, statically typed, and are designed to have extremely fast read accesses while providing a user-friendly "Folder and File" naming structure.

4.5 Other Built-in Apps

4.5.1 Main Menu

The main menu is an application that shows a list of the other applications in the system and lets the user enter any of them. It is also the default application the kernel will go to if an application crashes in a non-fatal way or exits.

4.5.2 Settings

The settings application is used as a front-end for the Settings subsystem of the driver. It is able to parse each setting and create a folder/file structure from the setting names. This makes it way easier for the end-user to quickly find and modify a setting. For development purposes, it can also call contain settings of the type "Action", which can call arbitrary functions and that allows the creation of quick scripts/tasks without the need to create a whole new app.

4.5.3 Testing

This simple application is designed to test the different systems of the console. You can interact with all the inputs of the system and see them change in real-time on the screen.

4.6 Software Porting

For our experiments, two commonly known pieces of software were ported to the console, both to demonstrate how easily existing titles can be ported, but also as a baseline for our experiments on user interaction. The first one is Doom, a First Person Shooter that was released in 1993, and which has become a cornerstone of gaming as we know it today. Apart from that, it was also chosen to represent a piece of software where one has complete access to its source code, and can therefore easily modify and add new interactions. The second application is a Game Boy Color emulator, for which you have no access to the source code of its games. This fact makes it harder to both know the game state and to make changes to it, which makes adding advanced interactions quite more challenging.

4.6.1 Doom

This Doom port is based on the work of the previous nrf53840Doom port [14].

Figure 4.3: **Doom running in the console.**

Several features such as sound and networking were removed due to the limited energy available in our system. Many optimizations that were customized for the NRF52 hardware had to be removed and reverted back to the original DOOM source. The game files are stored as a resource file in the external Flash, which amounts to 3.1MB of data after some optimizations (down from 4MB). All the game logic is contained in the code itself, which is stored in the MRAM memory of the MCU.

To add checkpoint support for the game, the RAM used to store the actual game status had to be physically separated from the one being used as a cache (for example to store the current level geometry) so that the amount of memory needed to save the game state could be reduced to a minimum. All the cached assets that need to be present at runtime but do not need to be stored are loaded from the external Flash when the system recovers from a checkpoint. This differentiation makes it possible to achieve high framerates with a significant reduction in the amount of checkpointable RAM (1.3MB down to 100KB) in exchange for a slightly longer recovery time from a checkpoint.

4.6.2 Game Boy Color Emulator

The Game Boy Color emulator port is based on Gaembuoy [26] project. It is a dual Game Boy and Game Boy Color emulator that implements most features of the original hardware. Our port removes the sound and networking features and adds new optimizations to be able to achieve 60FPS in most games. These optimizations include changing the memory layout of many internal structures of the emulated CPU to make it easier for our MCU to have them in the cache, detecting "hot-paths" in the cartridge code, and storing these tight loops in RAM to make it emulation faster and frame skipping. The frame-skipping system tries to skip the rendering of the screen during some frames when it is likely that 60FPS is not going to be reached. This enables the game to keep running at its designed speed (60FPS) while saving power (we only output

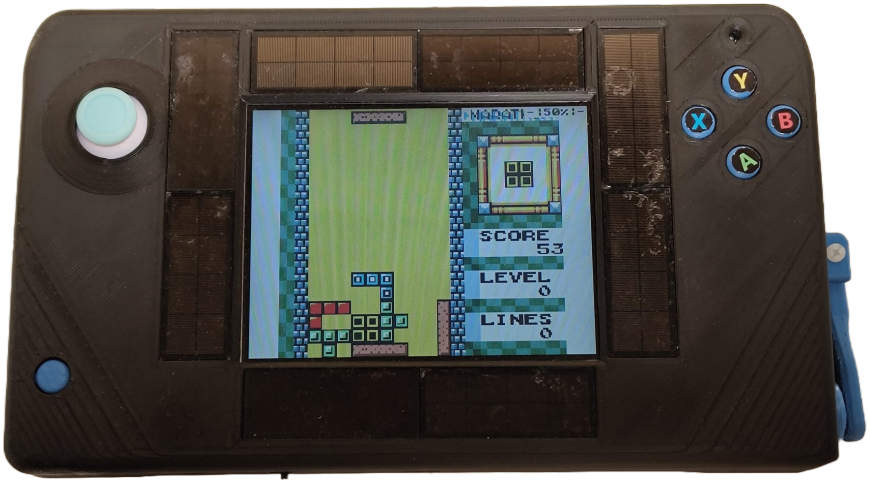


Figure 4.4: **Doom running in the console.**

20FPS to our screen) and making the game playable without slowdowns.

The whole system memory of the emulated Game Boy is checkpointed, providing easy and seamless integration with the checkpointing system.

4.6.3 In-Game Features Connected to Energy Harvesting Inputs

As part of the interactive aspect of the system, an application can know at any time the status of the crank, including both the cranking direction and speed. This makes it possible to have complex interactions, which not only take into account whether the crank is being turned, but also its speed, real energy transfer, and direction.

- **Doom:** Doom is a fast-paced First-Person Shooter (FPS) that requires constant input (movement and firing) to avoid losing. This makes it really hard to switch one's hand from the buttons to the crank without being detrimental to the gameplay experience. Here is where adding a gameplay element to the crank is really useful. By connecting the crank movement to the fire animation of the Doom minigun, one can provide an intuitive and satisfactory way to entice the user to crank the system. Since the character is also harder to move around to dodge attacks while cranking, an invincibility status is also given to the character while cranking.
- **Game Boy Color (Tetris):** Contrary to the Doom integration, when emulating a GBC game it is rare to have access to the source code of the game, so the range of available ways to integrate the crank is limited to the actual button inputs. For Tetris, we chose to link the crank movement to the rotation of the pieces while falling. This is done by simulating A/B button pushes while the crank is turning.

Chapter 5

Evaluation

This research platform was designed to demonstrate whether incentives in a power-constrained environment increase the interaction and usability of the system.

It is therefore essential to make sure the system is capable of generating enough energy and is efficient in using that limited resource. For that, thanks to the integrated measurement capabilities of the logging module, the performance of the solar panels and the crank module will be measured, alongside the power consumption in different scenarios.

After that, I will go into detail analyzing the performance of the software implementation of the kernel, and especially the checkpointing system.

5.1 Evaluation Setup

The measurements taken for the efficiency measurements were done by putting a SDM3055 multimeter in series with each power rail, thanks to the jumpers that were designed into the PCB that allow easy access to this type of measurement.

Any other electrical measurement that appears in this section has been recorded through the integrated logging module of the hardware platform. This module uses the Analog to Digital Converters (ADC) of the ESP32 microcontroller to perform the measurements. The measurements taken by the ADCs of each ESP32 module are then refined in software based on the individual in-factory calibration values recorded in the internal memory of each MCU. Based on the ESP32 specification, the maximum error after this calibration is 1.1% [18].

Unless stated otherwise, all timing measurements in this section are deterministic and have been carried out by the console itself (through the use of hardware timers) or by using the timestamps in the files produced by the logging module.

5.2 System Power Usage

The overall system power usage and capacitor voltage over a play session can be observed in 5.1. There is a noticeable difference in power usage depending on the game the user is playing.

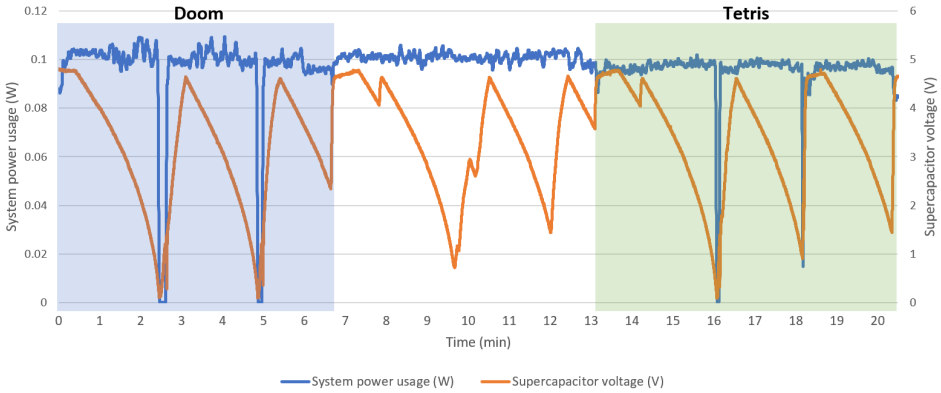


Figure 5.1: **System power consumption and capacitor level.**

During that play session, the average power usage while playing DOOM was 103 mW, and it slightly decreased over time because enemies were being killed and therefore, the overall complexity of running the game went down too.

The power usage while playing Tetris (emulated in the Game Boy emulator) was constant over time, resulting in an average of 97.5 mW. These measurements were taken by the logging module by measuring the voltage dropped across a resistor in the output of the 5V regulator (After being amplified by an INA180 current-sense amplifier from TI [22].

5.3 Energy Harvesting

The measurements for the harvesting modules use an INA180 [22] amplifier from TI, and they are carried out by the logging module as explained in the introduction of this chapter.

5.3.1 Solar Module

The energy harvested through the solar module can be seen in Figure 5.2. This example represents an indoor environment, where the solar panels are able to produce 1-1.2 mW. The relative light level during the experiment can also be seen, as measured by an OPT3004 sensor from TI [20]. Even though this sensor is calibrated to provide accurate light intensity in lux, due to the way it was placed in the console enclosure, only the light incident within 20° of the surface normal could hit the sensor.

Nevertheless, the light measurement should still be valid for relative comparison, and in fact, it is directly proportional to the amount of solar energy being harvested. Due to dynamic range limitations of the measurement ADC, the measurement for solar power clips at 1.9 mW.

5.3.2 Crank Module

As it can be seen in Figure 5.3, the crank is the main energy contributor to the console. The amount of energy harvested through the crank varies depending

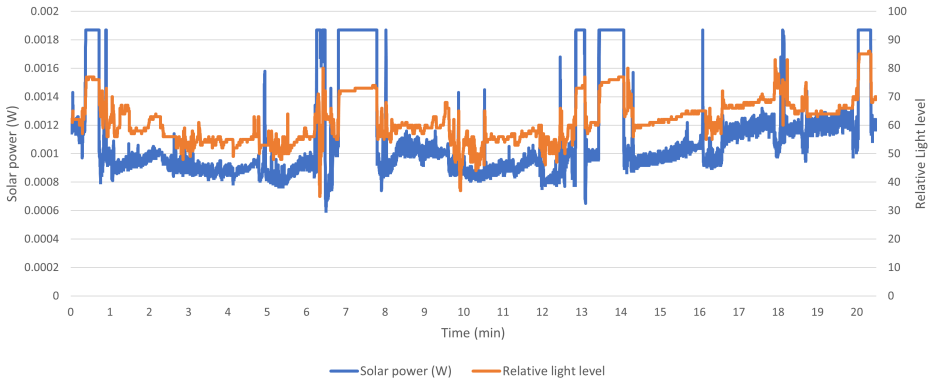


Figure 5.2: **Solar module generation and relative light level.**

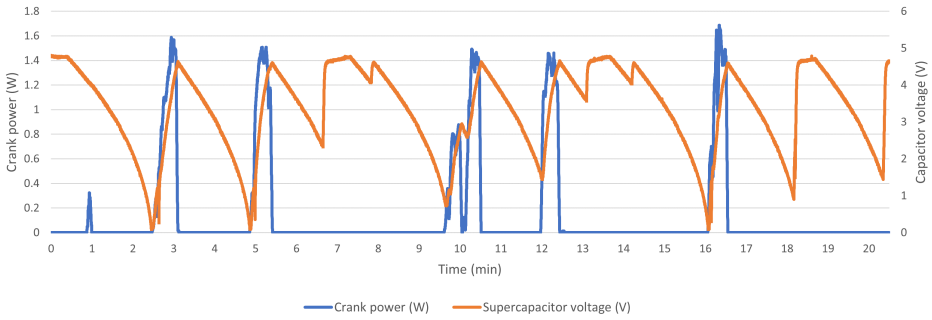


Figure 5.3: **Crank module generation and capacitor voltage.**

on the cranking speed, but it is easy to reach 1.4 W under normal conditions.

To evaluate the performance of the custom Maximum Power Point Tracking (MPPT) system developed for the crank module, a separate measurement is done, where the crank speed is kept constant and the power harvested by the system is measured at different output capacitor voltage. As can be seen in figure 5.4, if the MPPT system is disabled (and therefore the DC-DC converter is always on), the performance of the system is significantly worse than with MPPT enabled. This difference is even more visible when the output capacitor voltage is close to 0, since in this case the converter will try to load the crank motor as much as possible, moving it further from its maximum power point.

5.4 Power Path Efficiency

Table 5.1 shows the average amount of current drawn by the system from each power rail while idling in the main menu. Efficiency figures for the power rails were obtained by either disabling the ones not being measured (for the ones that do not affect the rest of the system, like MEM or 1.2V), or analytically measuring the input and output power of each converter in its path.

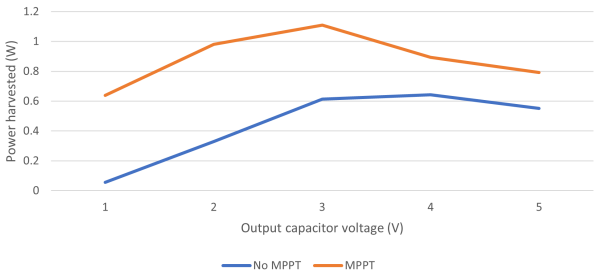


Figure 5.4: **Crank module generation vs output capacitor voltage. The MPPT approach increases the performance significantly, especially when the main supercapacitor level is low.**

Rail	5V	3V3	AUX	1V8	MEM	1V2	TOT
Current (mA)	4	0.15	10.3	4.6	0.4	1.2	-
Efficiency	91%	81%	81%	79%	79%	73%	84%
Power used (mW)	27	0.5	44	9.3	0.7	1.64	83.1
Power loss (mW)	2.7	0.1	10	2	0.2	0.6	15.6

Table 5.1: **Power usage and efficiency while idle.**

5.5 Checkpointing

5.5.1 Checkpoint Creation

This checkpointing saving process takes 1.5ms with the checkpoint configuration seen in Table 5.2 This number is a worst-case checkpointing scenario, where the whole heap needs to be stored.

That configuration, as reported by the linker after compilation, represents a release build optimized for performance. It is also the firmware used to conduct the rest of the experiments, and contains all the software necessary to run DOOM and the Game Boy emulator.

Since our targeted framerate is 30FPS (33ms per frame), checkpointing only represents a worst-case overhead of 5% (1.25 ms out of 33 ms). This isn't even noticeable most of the time since the console is usually faster than this goal, and the CPU would just stay in low power mode waiting for the remaining of the frame-time. In the case where the load is large enough to make the console run at, for example, only 20FPS, this overhead would only reduce overall FPS by 0.6FPS. Anything lower than that, where this overhead would be more noticeable, would also be deemed unplayable by most users. As it can be seen in Figure 5.5, once your FPS increases above the FPS limit, the checkpointing overhead that is noticeable by the user drops to zero.

5.5.2 Checkpoint Recovery

The process of restoring a checkpoint takes 0.5ms when restoring the whole heap, which is insignificant when compared with the time it takes to initialize all the hardware in the console (129.75ms) 5.6 during the kernel start-up. The screen interface initialization takes the longest (80 ms, most of that waiting for

Heap	Stack	Register + header	Linker sections
100KB	16KB	20B	55KB

Table 5.2: **Checkpoint benchmark configuration.**

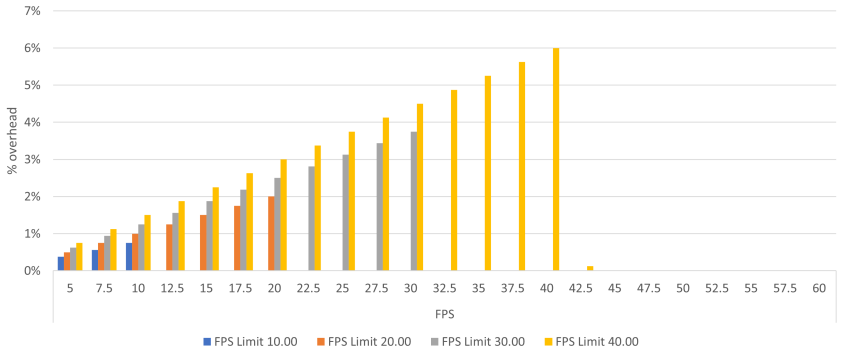


Figure 5.5: **Perceived checkpointing overhead vs FPS. The amount of overhead grows linearly with the desired FPS. If the game FPS is larger than the FPS limit of the system, the overhead becomes invisible to the user.**

the internal clocks to stabilize), followed by the Random Number Generator (RNG) with 20 ms and Flash memory (15 ms).

Due to the deterministic nature of the hardware, these measurements are stable on each initialization, unless the system encounters a random error and has to retry after a certain period of time.

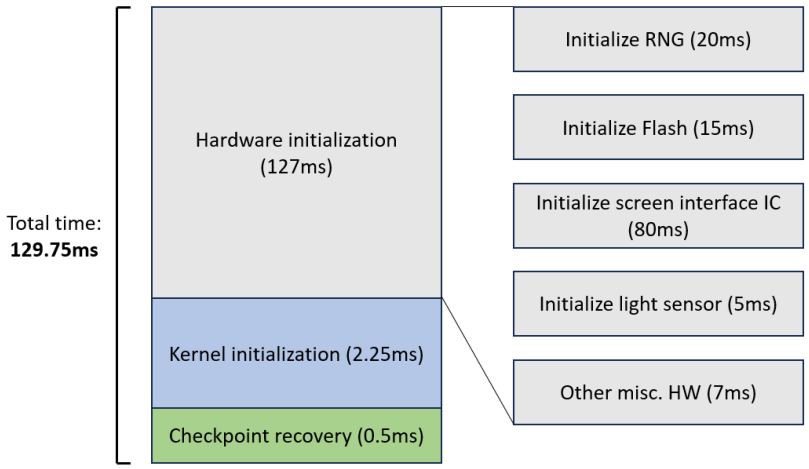


Figure 5.6: **Initialization time breakdown.**

Chapter 6

Preliminary Experiments

This chapter details the methodology, setup, and observations taken from the preliminary experiments to try to answer the question: *Can incentives such as a crank in an energy-constrained environment improve interactivity?*. The goal is to provide a transparent account of the experimental process, the insights gained from it, and the implications of these findings on future work.

Due to time constraints for the presentation of this thesis caused by several hardware iterations and complications creating a reproducible environment for the experiments, this analysis will be based on a limited sample size. Therefore, these results will be used as a pre-study evaluation to gain insights into how to run the actual experiment. These preliminary results are based on a sample size of 8, all of which are students or faculty at TUDelft. All the study participants knew beforehand that they would receive a 5 euro gift card at the end of the experiment. The data used for this analysis can be found in [27].

6.1 Experiment Description

Upon arrival to the experiment room, participants are introduced to the context of the experiment and asked to sign an informed consent form, ensuring their understanding and agreement to participate in the experiment. Participants are then introduced to the game console. They are given an initial system tutorial, during which they are familiarized with the console's interface, controls, and main features.

6.1.1 Gameplay

Participants are given either Tetris or DOOM to play first. Clear instructions are provided to start the game. In all cases, the game console was fully charged and set to the appropriate console settings for the experimental procedure. Participants are given control of the console and they are instructed to play the game for a period of about five minutes.

After the initial game, participants were then given the second game. Just like before, they receive instructions and are given control of the console for another five-minute gameplay session.

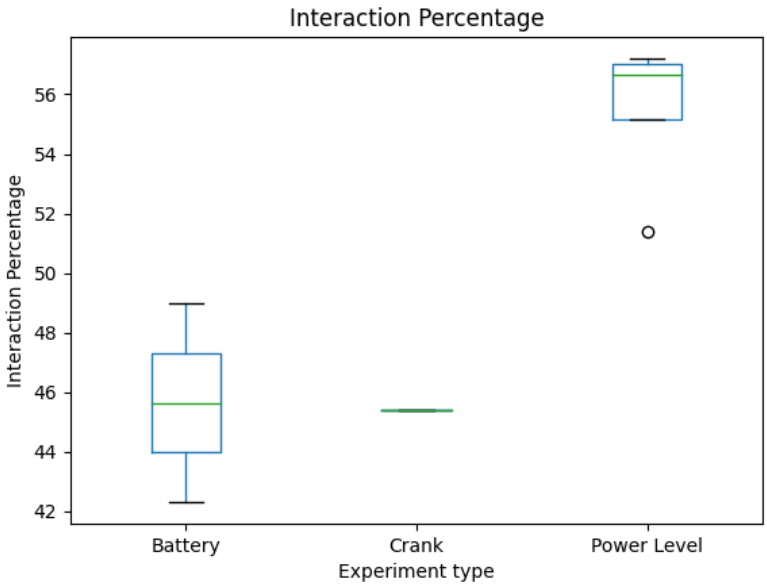


Figure 6.1: **User interaction by experiment type.**

6.1.2 Feedback Process

Following the gameplay, participants are asked to fill out a survey and engage in a semi-structured interview. They are asked about their overall impression of the system, their feeling of connectedness to the games console and its energy-harvesting mechanics, the best and worst parts of their experience, their use of the pause feature during gameplay, and suggestions for other potential applications for the console's energy harvesting mechanics.

6.2 Objective Data From Logs

Even though the sample count from this preliminary study is not large enough to make any substantial conclusions, a significant trend can already be seen in the data so far. In these experiments, the presence of interaction is defined as the existence of interaction between the user and the console within a 100ms interval. Joystick movement, button presses, and crank usage are taken into account. When the percentage of time a user is interacting with the console is plotted against the experiment type in Figure 6.1, a significant difference can be seen between the baseline (battery-powered) and the self-powered one with the remaining energy indicator (power level). This result confirms the question behind the thesis since it establishes a relationship between the presence of incentives and the interaction with the system.

Nevertheless, Due to the low sample number, no conclusions can be taken at this point for the one with crank incentives.

6.3 User Feedback

In general, user feedback was focused on a series of key criticisms:

- **User interaction**: From our direct observations during gameplay, we found that the learning curve of operating the crank system varied among users, with some understanding the need for faster cranking only after the second DOOM session. Others noted that the crank system was hard to operate during gameplay, especially in fast-paced games like DOOM. Some participants were annoyed with the noise produced by the crank system and its imposition on the gameplay experience.

- **Feedback on the performance of the console**: Regarding the console's performance, user feedback highlighted several areas for improvement. The rapid battery discharge and the need to continually operate the crank system for charging were cited as notable concerns. Users found the cranking system to be disruptive to gameplay and expressed a preference for a longer-lasting charge that would allow for extended gameplay without the need for cranking. Moreover, users voiced dissatisfaction with the console's screen visibility, particularly under dim light conditions. The use of a dimmed screen was described as disturbing, and even "hated" by a participant.

- **Feedback on the console design**: Participants found the console similar to regular game consoles, with the crank seen as a "nice extra feature" but not a significant distinction. Some participants found the console too big to hold in one hand, suggesting a need for a more ergonomic design.

- **Suggested improvements**: When asked to provide suggestions for games where the cranking interaction could be integrated into gameplay, participants had difficulty coming up with ideas, with one suggesting a football game where cranking could be used to pass the ball. For power storage, one user suggested that a 55% charge should allow for 5 minutes of gameplay and that a 10% charge could serve as an indication to start cranking.

- **Considerations for further experiments**: A key observation from the experiments was the varying light conditions during gameplay, which appeared to influence users' gameplay experience. Future experiments will need to control this factor.

Chapter 7

Future Work

The evaluation and experiments have proven that it is possible to have a battery-free gaming platform capable of playing relatively new games while also being completely self-powered.

The increased power available to the system thanks to the improvements to the energy harvesting system, as detailed in Section 1, opens the door to a lot of additional features that were unthinkable before.

Moreover, the development of a system as complex as this one requires a lot of iterations and further refinement. With that in mind, these improvements can be separated into two topics, hardware and software:

7.1 Hardware

- **Adding support for wireless communication:** Nowadays, online gaming represents over 53% of the gaming industry[12]. Adding support for wireless communication would be a significant improvement to this platform and would further increase interactivity by allowing multiplayer experiences. Using one of the upcoming NRF54 series of 2.4 GHz MCUs from Nordic Semiconductor might be acceptable energy-wise, since their RX power consumption has been significantly reduced, down to 5mW according to their announcement [24].

- **Audio support:** Even though having speakers is out of the question due to their huge power consumption, adding headphone support for sound effects might be helpful in increasing user immersion in exchange for increased power usage.

- **Improve enclosure ergonomics:** The current physical format should be redesigned to make the console thinner, using a custom motor gear assembly and using multiple smaller capacitors instead of a big one. This would also make the console thinner, and therefore lighter and more ergonomic.

7.2 Software

- **Simplify the app build system:** One of the most limiting issues of the platform right now is the need for all the code that runs on the system to be compiled and linked all together. This severely limits the developer-friendliness of the system, since it is not possible to add/update/remove applications at runtime. Nevertheless, the system architecture is designed with this in mind, providing extremely light coupling between applications and the kernel, so making this change should be quite straightforward.
- **Add support for other screen technologies:** Support for black/white screens should be added back into the Kernel, with the appropriate dithering techniques to make the image quality better.

Chapter 8

Conclusions

This thesis presented an in-depth study into the design, implementation, and evaluation of an interactive, low-power, intermittent gaming research platform. The integration of a robust checkpointing system effectively navigated the unique challenges of intermittent power, enabling a seamless gaming experience that was resilient to power interruptions. The innovative console has successfully navigated the balance between user interaction, system performance, and energy efficiency.

The development process showcased that the adoption of common programming frameworks and APIs could simplify the process of porting existing games to the new platform. All the hardware-dependent aspects of porting games like Doom and the GameBoy Color emulator were removed, effectively focusing the porting efforts in maintainability and optimization.

The conducted experiments have also proven to be an invaluable source of insight. Not only did they provide valuable feedback about understanding how the end user interacts and values each part of the console, but they also illuminated potential areas of improvement and refinement. These findings will serve as a foundation for future developments and iterations of the platform.

The preliminary results obtained seem to indicate that the answer to the question that originated this thesis (whether incentives can increase interactivity in power-constrained systems) is "Yes", but more data is needed before a definitive answer can be given. The preliminary study conducted has also been invaluable in refining and simplifying the experiment procedure for the final study.

Moreover, the platform's design was able to maintain a strong focus on sustainability, minimizing electronic waste and emphasizing eco-friendly gaming solutions. This perspective is instrumental as we move towards more sustainable practices in technological development.

Bibliography

[1] Eric J Denton and Maurice Henri Pirenne. "The absolute sensitivity and functional stability of the human eye". In: *The Journal of physiology* 123.3 (1954), p. 417.

[2] Richard Lawrence. "MIPI high speed serial interface standard for mobile displays". In: *Mobile Displays: Technology and Applications* (2008), pp. 315–328.

[3] Eric Eason. *Smartphone battery inadequacy*. Tech. rep. Stanford University, 2010, pp. 3–6.

[4] Shuhong Zhao et al. "A measurement method for determination of dc internal resistance of batteries and supercapacitors". In: *Electrochemistry Communications* 12.2 (2010), pp. 242–245. ISSN: 1388-2481. DOI: https://doi.org/10.1016/j.elecom.2009.12.004. URL: https://www.sciencedirect.com/science/article/pii/S1388248109005980.

[5] Wendy Wilhelm, Alice Yankov and Patrick Magee. "Mobile phone consumption behavior and the need for sustainability innovations". In: *Journal of Strategic Innovation and Sustainability* 7.2 (2011), pp. 20–40.

[6] Masaya Tamaki et al. "A memory-in-pixel reflective-type LCD using newly designed system and pixel structure". In: *Journal of the Society for Information Display* 22.5 (2014), pp. 251–259.

[7] Sharp Displays. *COM50H5M81XLC Datasheet*. 2017. URL: https://www.distec.de/fileadmin/pdf/produkte/TFT-Displays/Ortustech/COM50H5M81XLC_Datasheet.pdf.

[8] Mona Lisa and Hew Bot. *My Research Software*. Version 2.0.4. Dec. 2017. DOI: 10.5281/zenodo.1234. URL: https://github.com/github-linguist/linguist.

[9] *LS044Q7DH01LD28605 Display Datasheet*. 9th Jan. 2017. URL: https://www.sharpsde.com/fileadmin/products/Displays/Specs/LS044Q7DH01_28Jun16_Spec_LD-28605A.pdf (visited on 26/08/2023).

[10] *AVX-SCM Supercapacitor Series Datasheet*. 11th Oct. 2018. URL: https://www.farnell.com/datasheets/2675070.pdf (visited on 14/07/2023).

[11] Changfu Zou et al. "A review of fractional-order techniques applied to lithium-ion batteries, lead-acid batteries, and supercapacitors". In: *Journal of Power Sources* 390 (2018), pp. 286–296. ISSN: 0378-7753. DOI: https://doi.org/10.1016/j.jpowsour.2018.04.033. URL: https://www.sciencedirect.com/science/article/pii/S0378775318303768.

[12] Kenneth S Horowitz. "Video games and English as a second language: The effect of massive multiplayer online video games on the willingness to communicate and communicative anxiety of college students in Puerto Rico." In: *American journal of play* 11.3 (2019), pp. 379–410.

[13] Jasper De Winkel et al. "Battery-free game boy". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4.3 (2020), pp. 1–34.

[14] *Porting Doom to an nRF52840-based USB Bluetooth-LE Dongle – next-hack.com*. 16th Nov. 2021. URL: https://next-hack.com/index.php/2021/11/13/porting-doom-to-an-nrf52840-based-usb-bluetooth-le-dongle/ (visited on 04/08/2023).

[15] *SSD1963 Video Interface IC Datasheet*. 1st June 2021. URL: https://www.seacomp.com/sites/default/files/references/Solomon-Systech-SSD1963.pdf (visited on 14/07/2023).

[16] Ambiq. *Apollo 4 MCU Series Datasheet*. 24th Apr. 2022. URL: https://ambiq.com/apollo4/Apollo4-Document103.pdf.

[17] Yen-Ting Chen et al. "SACS: A Self-Adaptive Checkpointing Strategy for Microkernel-Based Intermittent Systems". In: *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 2022, pp. 1–6.

[18] *ESP32 MCU Series datasheet*. 3rd July 2022. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (visited on 28/08/2023).

[19] *LTC3526/LTC3526B (Rev. E) DCDC Converter Datasheet*. 28th Nov. 2022. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/ltc3526.pdf (visited on 14/07/2023).

[20] *OPT3004 Ambient Light Sensor (ALS) With Excellent Angular IR Rejection datasheet (Rev. A) - opt3004.pdf*. 5th June 2022. URL: https://www.ti.com/lit/ds/symlink/opt3004.pdf?ts=1670764640397&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FOPT3004 (visited on 27/08/2023).

[21] *DSLVDS1001 400-Mbps, Single-Channel LVDS Driver datasheet (Rev. A)*. 5th May 2023. URL: https://www.ti.com/lit/ds/symlink/dslvds1001.pdf?ts=1689272593316 (visited on 14/07/2023).

[22] *INA180 data sheet, product information and support — TI.com*. 27th Aug. 2023. URL: https://www.ti.com/product/INA180 (visited on 27/08/2023).

[23] *KR Supercapacitor data sheet - eaton-kr-supercapacitors-coin-cells-data-sheet.pdf*. 12th July 2023. URL: https://www.eaton.com/content/dam/eaton/products/electronic-components/resources/data-sheet/eaton-kr-supercapacitors-coin-cells-data-sheet.pdf (visited on 14/07/2023).

[24] *Nordic Semiconductor redefines its leadership in Bluetooth Low Energy with the announcement of the nRF54 Series - nordicsemi.com*. 28th Aug. 2023. URL: https://www.nordicsemi.com/News/2023/04/Nordic-Semiconductor-redefines-its-leadership-in-Bluetooth-Low-Energy-with-the-nRF54-Series (visited on 28/08/2023).

[25] Federico Rossi et al. “Environmental optimization model for the European batteries industry based on prospective life cycle assessment and material flow analysis”. In: *Renewable and Sustainable Energy Reviews* 183 (2023), p. 113485.

[26] *simias/gaembuoy: Game Boy Color emulator in C*. 4th Aug. 2023. URL: https://github.com/simias/gaembuoy (visited on 04/08/2023).

[27] TUDSSL. *Condenar data repository*. Aug. 2023. URL: https://github.com/tudssl/condenar-data.

[28] Yue Zhu, Xin Li and Junrui Liang. “Motion-Powered Gameboy”. In: *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*. SenSys ’22. Boston, Massachusetts: Association for Computing Machinery, 2023, 778–779. ISBN: 9781450398862. DOI: 10.1145/3560905.3568070. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/3560905.3568070.