Evaluation of the NESizer2 method as a means of wrapping embedded legacy systems

Contents

1 Abstract

Legacy computer systems are systems where several of the main hardware and software components date back several decades. Modernizing these systems is often considered a large monetary and temporal investment with high risk, and to keep maintaining them usually becomes more and more difficult over time, which is why these legacy systems are still being used to this day in many industry sectors. A solution is therefore to try and integrate the legacy system components into modern systems, and there are several ways of achieving this. This bachelor thesis project work aims to analyze one approach known as "wrapping". More specifically it analyzes *NESizer2 Method*, a method which utilizes relatively simple hardware and software interfaces to control the Ricoh RP2A03 processor found in the Nintendo Entertainment System, using an Atmega328 microprocessor. During the design and development phases of the project work a literature study was conducted, and experimental research method was utilized. The testing and experimental phases of the project work was focused on examining how identified key variables behaved when modifying certain parameters in the system. While we were able to produce some valid data, the results proved to be somewhat inconclusive, as certain operations such as memory operations did not work, leading to the conclusion that our circuit contained a faulty component.

1.1 Keywords

Legacy components, modernizing, microcontroller, data injection, Atmega328P, Ricoh 2A03

2 Sammanfattning

Legacydatorsystem är system där många av de huvudsakliga hårdvaru- och mjukvarukomponenterna är flera decennier gamla. Att modernisera dessa system ses ofta som en stor monetär och tidsmässig investering, och att fortsätta att underhålla dem blir vanligtvis svårare och svårare med tiden. En lösning är därför att försöka att integrera legacy-systemets komponenter i moderna system, och det finns ett flertal tillvägagångssätt att uppnå detta. Detta kandidatexamensarbete ämnar att analysera ett tillvägagångssätt känt som "wrapping". Mer specifikt analyseras *NESizer2-metoden*, en metod som utnyttjar relativt enkla hårdvaru- och mjukvarugränssnitt till att kontrollera Ricoh 2A03-processorn som finns i Nintendo Entertainment System, med hjälp av en Atmega328 mikroprocessor. Under design- och utvecklingsfaserna av projektarbetet utfördes en litteraturstudie, och experimentiell forskningsmetod användes. Test- och experimentfaserna av projektarbetet fokuserade på att undersöka hur identifierade nyckelvariabled betedde säg då man modifierade vissa parametrar i systemet. Även om vi lyckades producera en del riktig data visade sig resultaten vara ofullständiga, då vissa operationer såsom minnesoperationer inte fungerade, vilket ledde till slutsatsen att vår circuit innehöll en defekt komponent.

2.1 Nyckelord

Legacy components, modernizing, microcontroller, data injection, Atmega328P, Ricoh 2A03

3 Introduction

When integrating legacy information system components into a modern system, one of several usual approaches is to create an interface for the modern system to control or communicate with the legacy component. This approach is known as wrapping.[1], [2] Wrapping as a concept could be adapted for legacy hardware in embedded systems, to enable the original and proven functionality of the outdated system, by providing an interface to control the legacy components. There are few documented examples of migration of legacy systems that include the incorporation of the system including the hardware platform, and the few examples that exist are often designed with a specific functionality in mind. The NESizer2 project details a method wherein a modern microcontroller is used to wrap certain functionalities of the microprocessor used in a Nintendo Entertainment System, by dynamically injecting instructions to the microprocessor like an emulated ROM.[3] This thesis will evaluate how this method could be expanded upon to allow for a general use case of the legacy component, and to evaluate how well the method performs as a means of wrapping.

3.1 Background

Computer based information systems are an invaluable asset for modern enterprises and corporations. The use of information systems can range from data storage, analysis and organization, to communication systems such as mail servers. With continued updating and upgrading of these systems they become increasingly complex, and as technology evolves the existing technology of the systems are quickly rendered obsolete, and "*eventually, the existing information systems become too fragile to modify and too important to discard*" [4], and as such it can be considered a legacy system. [5] At this point the legacy systems must be modernized, or migrated into a more modern system to remain viable.

When incorporating legacy information systems into modern systems, there are usually three popular approaches: redevelopment, wrapping and migration.[1, pp. 2–4] While redeveloping an entire system is usually the best option in the long run, it is also the most expensive and risky. Therefore, migration is usually a more popular method, as it provides an interface to control the legacy components, while retaining its data and functionality. However, migrating systems to a modern platform can lead to unexpected behaviour, with a notable example being NASA's Ariane 5 flight 501.[6], [7]

When redevelopment and migration is too risky or expensive, wrapping offers a cost-effective option with less risk. It surrounds existing data, systems and interfaces with new interfaces, giving the legacy components a "new and improved" look [1, p. 3], and lets organizations reuse trusted components in a new, more up-to-date manner. While these methodologies and frameworks usually focus on legacy software systems, they are very much applicable to hardware systems and components as well.

One way of wrapping a legacy hardware component is detailed in the hobby project NESizer2 by *Johan Fjeldtvedt.*[3] The project is a MIDI controlled synthesizer, using the original *Audio Processing Unit* (APU) found embedded in a RP2A03 microprocessor - the microprocessor used in the Nintendo Entertainment System. In his method he uses a modern microcontroller to handle the normal functionality of a MIDI-controller aswell as controlling the RP2A03 by dynamically injecting instructions into the microprocessor when the APU is needed.

3.2 Problem

To address the issues of unexpected behaviour in an otherwise proven, well functioning system, the solution could be to keep only the crucial legacy components including their hardware platforms, and provide an interface for a modern system to control them - creating a wrapper for both hardware and software. While there exists implementations of similar approaches, they are often designed with a specific functionality of the legacy component in mind, and as such does not provide a method of controlling the component for a general use case. This poses the question;

Could these specific implementations be generalized into methods of controlling a legacy component, without any specific use case in mind? If so, how well do they perform as a means of modernization?

To try and answer these questions, we will investigate the method used in the NESizer2 project, hereafter referred to as *The NESizer2 method*, to see if it can be expanded upon to be used as a *wrapper* for the RP2A03 microprocessor. The method is considered a wrapper if it can allow a general use case of the microcontroller - specifically if it can successfully utilize the entire instruction set of the processor, thereby allowing any RP2A03 program to be run through the wrapper.

3.3 Purpose

The purpose of this report is to investigate how the NESizer2 method performs as a means of modernization, by repurposing the method to handle the entire instruction set of the RP2A03/6502 microprocessor, and measuring its performance in speed of execution aswell as investigating the complexity of implementing the method.

The purpose of the work is to provide some insight to how well a relatively simple method of wrapping an outdated microprocessor can be expanded upon to function as method of modernizing a legacy embedded system. Although our work is very basic and does not cover the entirety of how to wrap a whole legacy system, we hope our findings can be used as a future reference for others interested in modernizing embedded hardware.

3.4 Goal

The goal with the work is to provide insight into how an existing method of controlling legacy hardware can be extended to allow for general usage, and to give a performance evaluation of the method. This can hopefully give an indication of their usefulness as a method of modernizing an information system that uses legacy components.

3.4.1 Social benefits, ethics and sustainability

If it is possible to wrap entire embedded legacy systems with relatively easy means, it could provide an alternative for businesses that are dependent on legacy embedded components to upgrade their systems

without having to invest in, what most likely would be, expensive migrations, and with minimized risk - as wrapping would keep the legacy components intact. We also hope that our work can contribute to other research that aims towards a more sustainable solution than discarding still functioning computer systems, which is becoming an increasing threat to our environment.[8]

We acknowledge that our research could contribute to the continued use of legacy hardware. While the process of discarding obsolete hardware etc. for new parts can have a negative impact on the environment, it is also important to note that upgrading hardware could prove to be a better solution, as much research and development is aimed towards lower power consumption and with a more up-to-date view on sustainable engineering.

3.5 Methodology

In order to expand our knowledge and theoretical background in the field of research and define the research objectives, literature studies were conducted on several occasions during the research. A literature study is the systematic and methodical analysis of literature for a scientific purpose.[9]

A literature study was also conducted to decide on a suitable scientific method under which to conduct the research. The scientific method acts as a framework or guidance for the researcher to conduct their research in a well defined and systematic way, based on the works and experiences of researchers before them, and it is crucial to a research in order to ensure quality and correctness of gathered results and analysis.

We found that experimental research was most suited to the nature of our research. The experimental approach allows for observing how a system's behavior changes as one variable is manipulated while other variables are kept stable [10], and as such it is suitable for analyzing performance of a system.[11]

3.6 Stakeholders

As previously mentioned, wrapping could provide an alternative for businesses to keep their legacy systems alive, meaning it could save a company a substantial economic and temporal investment. As such, our main stakeholders are businesses and corporations where the need to maintain old systems exists.

Another stakeholder, or rather a target audience, are electronic enthusiasts who design and create embedded systems of their own in hobby projects. Using ourselves as an example, the idea for this project sprung from our interest in constructing a music device that incorporated the Ricoh RP2A03 microprocessor, used in the Nintendo Entertainment System (more details on this in the Trackers and the Chiptunes scene section).

3.7 Delimitations

The scope of this report is limited to the design and performance analysis of the NESizer2 method when it has been expanded to handle key parts of the 6502 instruction set, on a RP2A03 microprocessor. We chose to limit the instruction set by identifying groups of instructions that have similar characteristics, to ease implmentation and testing, and to be able to better draw conclusions on the overall performance of the system. Performance evaluation has been limited to speed of execution across the implemented instruction set, as well as the overhead as measured in time between consecutive instructions. The details of the implementation and evaluation criteria can be found in subsequent sections.

For a better indication of how well the communication method studied in our research can be adopted for other microchips/hardware and for a better picture of the behaviour of these communication methods on other systems, it would have been beneficial to implement them for two or more devices with different architectures. We have not tested intended use case performance of the RP2A03, i.e. having it read

program instructions from a regular ROM, and as such we were unable to make a tested comparison of execution speed.

3.8 Disposition

This report will firstly provide some background theory on legacy hardware and its use in a modern context, as well as an introduction to the hardware that has been used in this research. The research methodology used is presented in the following section together with other techniques and frameworks used to facilitate the research and development work. It is followed by a description of the project work, including the initial literature study and research phases, development phases and finally experiments and testing phases, followed by a section presenting the results from experiments. The last chapters will discuss the project work and the results from experiments, provide conclusions from the research work, and finally present ideas and thoughts about future work.

4 Background theory/Technical background

This chapter provides an introduction of what legacy hardware means, and also discusses how legacy hardware is used in modern systems and particularly how old hardware is still used to create and produce retro-sounding music. It also introduces a hobby project that sparked the idea for this research. The second part of this chapter discusses some earlier work related to the research problem, and work that was used as a basis for the communication methods designed for this research.

4.1 Legacy

Legacy is a term used in computing to mean "of, relating to, or being a previous or outdated computer system".[12] This could, for example, be computer systems or components that might have had a widespread usage or been considered a standard in the past, but are no longer manufactured or maintained by the producer.

4.2 Trackers and the Chiptunes scene

In the mid 1980's, a type of music sequencer commonly referred to as a "tracker" appeared on the personal computing markets. Today, a community colloquially named the "chiptune scene" consisting of musicians and retro enthusiasts fascinated with the characteristic sounds of the old video game consoles, create and perform their own music with these trackers. While many artists use software that can emulate the sounds of these machines on modern systems, it is often considered high status to create the music directly on the old hardware. An often recurring example representative of the scene is the tracker software LSDj [13], written for the Nintendo Game Boy. Its portable nature makes it an ideal option for artists, being able to carry their "instrument" anywhere with ease.

We wanted to, as a hobby project, develop a prototype for a portable music tracker, similar to the Game Boy and LSDj, using the characteristic sound from the popular Nintendo Entertainment Systems (NES) processor Ricoh RP2A03. In our research, we realized that it would be beneficial if we could write the tracker software for a modern microcontroller that would in turn control the Ricoh chip as a slave unit. This would give us all the expansive capabilities of a modern microcontroller, while also providing us with the actual audio output of the NES.

We realized that our need to control the Ricoh chip in this fashion could also be applicable to other legacy systems that are in need of upgrades, and where emulation is not a viable option.

4.3 MOS Technology 6502

The MOS Technology 6502 microprocessor and architecture was introduced on the market in 1975. It gained almost instant popularity due to its competative performance for a cheaper price than its competitors.[14]

The 6502 microprocessor contains instruction families and addressing modes to control every part of the architecture. Other than instructions which target basic CPU functionality (such as controlling program counter, reading status register etc.), there are groups of instructions to perform operations with the accumulator and memory. Included in these groups are immediate, zero page and absolute addressing.

Immediate addressing utilizes a 2-byte instruction format, where the first byte is the opcode specifying the operation and the address mode. The second byte takes a constant, hexadecimal value known to the programmer. Immediate instructions are commonly used when comparing variables to known values, as it does not require the programmer to first store the value in memory and then load it upon comparison; they would only have to specify the immediate value in the source code. It requires a minimum execution time of 2 cycles (`OPCODE` + `VALUE`).

Another 2-byte instruction address mode is zero page addressing. Along with the opcode, it takes a second byte which contains the effective address in page zero of the memory. Effectively, this means that zero page instructions can be used for quick accesses to memory locations in the range of `0x0000-0x00FF`. Zero page instructions are often used when working with intermediate values in larger calculations, to shorten the total execution time.

Absolute addressing mode takes 3 bytes in its instruction format: `OPCODE` + `ADDRESS_LOW_BITS` + `ADDRESS_HIGH_BITS`. Since this gives the programmer access to the full 16-bit range of memory (`0x0000-0xFFFF`), it's considered the most normal addressing mode..

A full, detailed explanation of all of the available addressing modes can be found in the MC6500 Microcomputer Family Programming Manual.[15] This research will utilize the three aforementioned addressing modes of the 6502 microprocessor. See Delimitations for further details on the choice of instruction families.

4.4 Ricoh 2A03

The microprocessor that was used in the Nintendo Entertainment System was a Ricoh RP2A03 chip. The RP2A03 is a proprietary chip based on the MOS Technology 6502 microprocessor architecture, with the difference that it has an added Audio Processing Unit (APU), and it does not support *decimal mode*[1] that would normally be available on a 6502 architecture. [14]

4.5 ATmega328P

The ATMega328P is an 8-bit, low-power CMOS microcontroller based on the AVR RISC architecture, with a throughput of up to 1 MIPS per MHz.[16] It is an easy-to-program, multi-purpose microcontroller that is included on the Arduino Uno and Nano microcontroller boards. It contains 32 KBytes ISP flash memory with true read-while-write operation, 1 KByte of EEPROM and 2 KByte of internal SRAM. It's 23 GPIO pins with programmable peripheral interfaces including SPI, I^2C and USART makes it an excellent light-weight microcontroller for relatively small scale projects.

[1] Decimal mode allows the processor to compute memory addresses written in decimal values, whereas it otherwise would use hexadecimal. REVIEW THIS FOOTNOTE AND ADD REFERENCES.

4.6 Related work

Since the topic of this research is to test the NESizer2 method as a way of implementing hardware wrapping, the NESizer2 project should be considered closely related to this project. [3] We have used and modified key parts of the communication protocol and expanded it to allow for general usage (i.e. enabled it to handle any instruction that targets the Accumulator, ALU or Memory, and that is within the Immediate, Zero Page and Absolute addressing modes), but other parts of the NESizer2 project may be of interest for enthusiasts and hobbyists that are interested in this kind of hardware wrapping.

During the literature study, we also encountered an article describing work on injecting instructions directly to the program memory of an older generation CPU. [17] Although our implementation does not inject instructions into a units program memory, the concept of injection is closely related. It is possible that the techniques described in that article could be used as an alternative method of hardware wrapping.

5 Development and methodology

This chapter gives an introduction to experimental research and how it can be used in system performance comparison and analysis, followed by a theoretical background to agile development.

5.1 Experimental research

During the research, a research method was applied to facilitate the process of analyzing and evaluating our implementation. A literature study was conducted in order to find an appropriate research method and strategy. The research methodology was chosen with the research question in mind; how to analyze and evaluate a system performance. The two main categories of research methodology are *quantitative* and *qualitative* research, which are separated by their founding philosophical assumptions. The qualitative research methodology assumes that observations, and importantly the conclusions drawn from them, are by their nature connected and dependent on prior knowledge and skill of the researcher and that the same observations might lead to different conclusions depending on the researcher. Qualitative research is mainly inductive in its nature, and the researcher will use their observations to infer *possible* hypotheses based on observations. Quantitative research, on the other hand, stems from positivism; the philosophical stand point that all things are governed by principles or laws (e.g. natural) and as such it is possible for researchers to observe these laws to draw conclusions in their research.[10, p. 23] Contrary to qualitative research, a quantitative approach is generally deductive, and is often aimed to confirm or deny a hypotheses that has been stated beforehand.[10], [11]

One example of quantitative research is *experimental research*. Experimental research is a strategy where the researchers try to control all variables that can affect the outcome of an observation. By methodically manipulate the state of one variable at a time, while keeping other variables stable, it is possible to understand how different variables affect the phenomenon that is to be researched.[10, p. 26], [11] As its main method of data collection, experimental research relies on experiments that are performed in this fashion. The gathered data can then be analyzed and used as a basis for conclusion to confirm or deny the stated hypotheses. In computer systems this method can be used to isolate the behaviour of the system for a certain input or event, and can be a useful method to analyze system performance.[11]

We chose to work according to the experimental research strategy, seeing as it is a suitable approach to analyze computer systems. To analyze the performance of our implementation we have chosen to observe how *per-instruction-overhead* and *completion time of a program* varies with respect to different sets of instructions and the program length. Because of limitations in our implementation in its current state, further described in subsequent chapters, we hypothesize worse performance than if

the chip could read instructions directly from a ROM, as intended. However, if the implementation is capable of executing the entire instruction set as expected, we believe that there are many areas of the implementation that can be optimized for better performance with relative ease.

5.1.1 Experimental techniques in computer system performance research

The development of computer systems has long been an area heavily driven by the marketplace. In order to be competitive on the market, a computer system has to either provide the highest performance, or the most cost effective computing engines. This means that as developers of computer systems, we need to succesfully "understand and then eliminate the system bottlenecks that prevent us from exploiting the available technologies". To gain a good understanding of how modern computer systems behave, and to localize the source of bottlenecks in a precise manner, experimental techniques are required.[18] In our research we have chosen one of these techniques when designing experiments to gather data and analyze the performance of our implementation - *hardware monitoring*.[18] The reason for chosing only one of these techniques is that the scope of interest for this research is mainly to see how well the embedded hardware functions as a means of wrapping older hardware, and to deduce this we can gain sufficient data by monitoring the timing of hardware signals.

5.2 Design and software development

This section provides a brief introduction to agile development and Scrum. These development frameworks was used during the research work to facilitate the design and development process of the research.

5.2.1 Agile development

The term agile, meaning "to have a quick resourceful and adaptable character"[19], was made popular in the software development circuit by the Manifesto for Agile Software Development.[20] The manifesto describes a model that, in contrast to traditional models for software development, embraces the fact that product description and requirements will most likely change during a development process, and adapts accordingly. It encourages building projects around motivated individuals, and promotes self-organization, continuous team meetings to reflect on the work that has been done, and regularly delivering work-in-process products to the product owner.

The agile software development model has spawned a number of frameworks to uphold the manifesto, including Extreme Programming (XP) and Scrum. These frameworks have helped set the standard of agile development, and has as such gained an immense foothold in the software development field. More recently, many universities are offering courses in the agile software development model, with research continously being done on how to effectively do so.[21]

5.2.2 Scrum

As previously mentioned, Scrum is one of many frameworks that implements the Manifesto for Agile Software Development. The creators Jeff Sutherland and Ken Schwaber define Scrum as the following[22, p. 3]:

> *"Scrum (n): A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value."*

Scrum utilizes an iterative, incremental approach to manage risks, and to dynamically develop a solution to a problem. The project is broken down into a set of time boxes known as "Sprints". The creators of Scrum recommend a sprint length of no longer than a month [22, p. 8], however the author of the popular Scrum introduction book "Scrum and XP from the Trenches" Henrik Kniberg recommends

new Scrum teams to experiment with sprint lengths until they find a time frame that the team feels comfortable with.[23, p. 22]

Each member of the Scrum team is assigned a role. These roles include product owner, developers and Scrum master. Each role have a specific set of tasks to fulfill.

- Scrum Master
  - Responsible for ensuring that Scrum is understood and enacted [22, p. 6] by making sure that the each member of the team follows the Scrum theory, practice, and rules.
- Product Owner
  - Responsible for maximizing the value of the product and the work of the development team[22, p. 5], and of managing the so called "Product Backlog", which contains items/tasks/requirements(?) that are to be completed in order for the product to meet the definition of done.
- Developer
  - Professionals who do the development work by delivering potentially releasable software at the end of each sprint.[22, p. 6]

The framework employs four formal events that help make sure that the team can deliver at the end of each sprint. These events are known as Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective.[22, p. 7]

- Sprint Planning
  - The Scrum team collaboratively decides on what can be delivered at the end of the sprint by moving tasks from the Product Backlog to a Sprint.[23, p. 24] A Sprint Goal is then created, which is a goal set by the team that can be reached by implementing the items in the Sprint backlog.
- Daily Scrum
  - A short meeting, usually around 15 minutes, where the team discusses the work that they will do on that day. This is done in order to synchronize team members, improve communication and improve the teams knowledge.
- Sprint Review
  - Helt at the end of each sprint, where the Scrum team and possible stakeholders collaborate and discuss what was done in the sprint. The attendees inspect the Product Backlog and any changes that was made, decide on what could be done in the next sprint in order to optimize value. The meeting is held with the intention of generating feedback.
- Sprint Retrospective
  - Held after the Sprint Review and before the next Sprint Planning meeting. It is held in order to inspect how the last Sprint went with regards to the team members, their relationships, the process and tools. The team tries to identify what went well, and what can be improved, with the aim to create a plan that improves performance in the next sprint.

5.3 Evaluation criteria

In order to gain an idea of how well the NESizer2 functions as a method of hardware wrapping, and to have some additional guidance in identifying the relevant variables that needed to be tested in experiments, we established a few points that we wanted to test and evaluate;

Performance in speed: We need to know if the wrapping method was significantly slower than the component(s) being wrapped. Since the wrapping would allow the component to be used only at crucial times, and allow a larger portion of programs to be run on a more modern machine, we were mainly focusing on short and simple programs that would for instance utilize memory mapped functionality of the legacy hardware.

Consistency in usage: It is important that the method does not introduce any quirks that would

have to be considered when implementing it as a wrapper. If certain programs or sequences of instructions behave differently when run via the wrapper compared to when run natively, the method would introduce to much friction to be viable as a wrapper. We decided to test this by grouping instructions by addressing mode into categories of instructions with similar properties, and to test groups of instructions both isolated and mixed together.

Correctness: Naturally, the wrapping method should not introduce or be responsible for any erroneous results during usage. We decided to use a digital logic analyzer in order to monitor the behavior of the system during the test programs.

Bottlenecks and overhead: For systems where timing is crucial, it is important to determine and understand any overhead or bottlenecks that come with the wrapper in order to determine its utility. Since the NESizer2 method was constructed to send one instruction at a time, we wanted to see how this affected a longer sequence of continuous instructions, and determine if there exists any significant delay between instructions.

6 Project Work and Development

This section details the project work, including literature studies and the hardware and software implementation/design process as governed by the Scrum framework, and the design and implementation of experiments according to the experimental research approach. It has been structured to closely follow the steps of the project work in chronological order.

6.1 Literature study

The project work started with a literature study, to gain knowledge on related work and theoretical background knowledge in the field of modernizing legacy hardware and legacy hardware used in modern applications. The search was performed using mainly the following databases of scientific publications: (i) IEEE Xplore, (ii) ACM Digital Library and (iii) Google Scholar. Additional searching tools used was simple internet searching tools such as *Google*, which could often provide ideas for additional keywords used when further searching in the databases. The results from this literature study was searched for using keywords: modernization/modernizing, legacy, hardware, microprocessor, computer. Based on the results of this search we further defined our keywords to target specific methods that seemed relevant, in order to find references on related previous work. The keywords used in this search was: legacy, microcontroller, microprocessor, master, slave, injection, wrapping, shared memory. The results from both searches was selected with title, abstract and publication year taken into consideration. Most of the related work was found to be older than 10 years, but considering that the articles mentioned methods of controlling legacy hardware, and that the problem of upgrading/modernizing hardware is generally a problem for machines older than 10-20 years, we found them relevant to the research. Figure 1 shows a simple illustration of the iterative process of the literature study.

Another literature study was conducted in order to gain further knowledge on experimental research, how it is used in performance comparison and evaluation, and general information on scientific methodology and how it is used in research. The keywords used in this search was: experimental, research, methodology, computer, system, performance. The search was conducted over the same databases mentioned above, and evaluated and selected using the same process. Anne Håkansson's article *Portal of Research Methods and Methodologies for Research Projects and Degree Projects*[11] mentions the book *Introduction to Research in Education*[10] as a source, and it has proved to be of great help when trying to understand what experimental research means, and how it can be used as a research strategy/methodology.

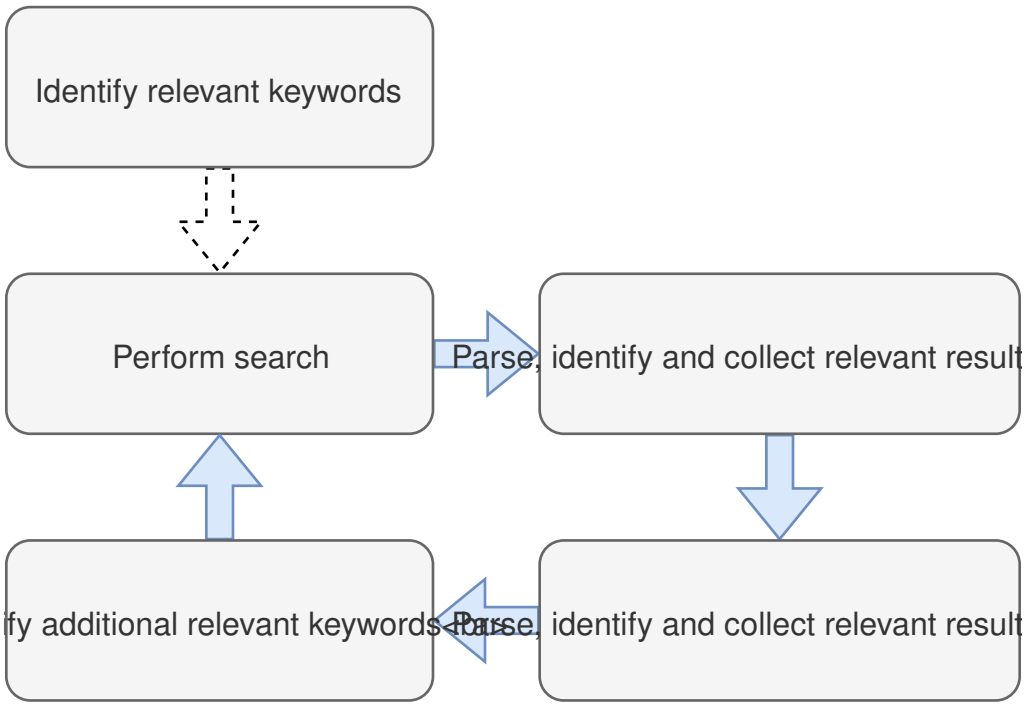

Figure 1: Illustrating the iterative process of the literature study.

6.2 Designing experiments

The experiments had to be designed not only to reflect our evaluation criteria, but in accordance with the experimental research approach they had to be designed around understanding how *the single independent variable* and *the single significant variable* affect the system performance.

When designing the experiments to evaluate performance in time, we used the categories of programs discussed in section Evaluation criteria and identified a set of all controllable variables that could have a significant impact on the results from each category. The variable chosen were:

i. Length of program, as measured in number of instructions
ii. Selection of timing for reset function
iii. Using emulated ROM functions or "pure" assembly

For each category we then designed an experiment that would test the performance when changing each of these variables while keeping the others stable.

To address the evaluation criteria *behaviour*, we included a data validation test to each experiment. The data validation is simply to verify that all test programs produced the expected output at each RP2A03 write cycle.

The outline of the experiments are further detailed in section Experiments phase.

6.3 Hardware and Software implementation process

When planning the project work we decided to use agile development and the framework *Scrum* to govern the design and implementation process. We chose divide design and implementation work into categories *hardware* and *software*, with one person responsible for each category. Before our first sprint we set up milestones and goals, and created a backlog of stories, or tasks, that would work towards

reaching set goals. This backlog is what we used to define tasks to include in each sprints.

The sprint goals were then set to reflect stages of iteratively increasing implemented functionality in the method, and stories and tasks for that specific sprint were then chosen to reflect the sprint goal. For each sprint we also defined a set of one or more deliverables that should represent the result of that sprint. A detailed description of the work concluded in each sprint during the implementation process will be given below, and a summary of the sprint goals and deliverables can be seen in Table 1.

Table 1: Overview of sprints and deliverables

Sprint #	Goal	Deliverables
1	Research and Design phase done	Research and design documents for hardware and software
2-3	Basic communication working	Components, circuitry, and basic software needed for communication
4	Sending instructions	Hardware for debugging, Software for sending instructions
5	Entire instruction set working	Software supporting entire instruction set

6.3.1 Sprint 1, Research and design phase

The research and design phase included research the NESizer2 software and hardware implementation. This research was made in order to pinpoint the parts from NESizer2 that we would need, and what modifications had to be done to it. We found that we could use the assembly routines (which was at the heart of the communication with the RP2A03) together with a simple 8-bit databus through an 8-bit latch to control flow of instructions, as a foundation for our implementation. This is detailed in the circuit diagram found in Appendix A.

We also made research on the RP2A03 microprocessor, but since the RP2A03 is a proprietary chip, owned by Nintendo, there was not much official information available. However, a community of NES enthusiasts has, through reverse engineering, gathered much information on the processor on their forums and wiki-site *Nesdev Wiki*.[24] Through these channels we were able to learn that the chip differs little from the MOS Technology 6502 architecture, and as such we could learn much about the chip through official 6502 hardware and software development guides.

Hardware design work then included research and decision of what hardware components were needed for our implementation, and finally to create a circuit diagram that we could use when building. Software design work resulted in a program flowchart and diagram of software components.

6.3.2 Sprint 2 & 3, Basic communication

The first part of the sprint was aimed at basic testing of components. Testing the RP2A03 was hard without any specialized tools, but we found a online document detailing a simple method of testing (i) power on and (ii) data bus of 6502.[25] The method mentions how to test this by using an array of LEDs on the address pins, but we decided to expand the test to observe the chip with an oscilloscope; we monitored the Read/Write and Clock output pin of the chip to confirm that different inputs produced expected sequence of RW signals. This gave us the confidence that the data bus and instruction parsing functioned, at least on some level.

After component testing was concluded, the next step was building a simple circuit of components that, together with basic software and a simple test program, could confirm that basic communication between the microcontroller and RP2A03 was working as intended. The results were inconsistent and

erroneous, and the build had to be debugged, which resulted in the sprint "overflowing" to the next sprint.

We finally realized that the problem was due to a misunderstanding in how the latch was used in NESizer2, and a simple edit to the circuitry resulted in consistent expected results. This confirmed that communication between the chips worked at a basic level. Figure 9 and 9 illustrates the misbehaving circuit and the corrected circuit, respectively.

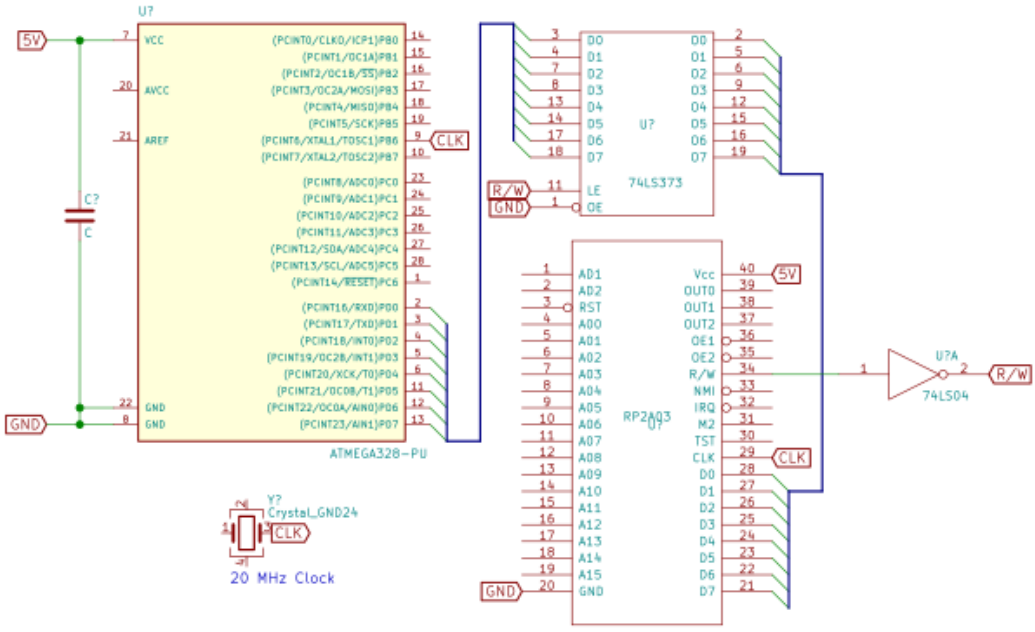

Figure 2: The circuit with the misplaced LE pin on the 74LS373 latch .

6.3.3 Sprint 4, Sending instructions

Hardware work included implementing simple debugging and diagnostics that could be used to determine if something was not working as intended in a quick manner, and proved to be very helpful during the sprint. After concluding the hardware work of the sprint, the software work was distributed to both parties and implemented together.

The NESizer2 uses high(er) level functions for instructing the RP2A03 to play a note, or to modify the sound, etc., with the help of hardcoded assembly instructions that performed set memory operations. We wanted to extend these assembly routines to allow for any instruction to be sent, and to build our own higher level C functions that could be used in a C program for the microcontroller used in the implementation. This was achieved by categorizing the 6502 instruction set (which is the instruction set used by the RP2A03) into instruction families that use the same number of operands. At this stage we chose to focus on three main families; (i) Immediate operations, (ii) Absolute addressing memory operations, and (iii) Zero Page addressing memory operations. A summary of the characteristics of these families can be found in Table 2.

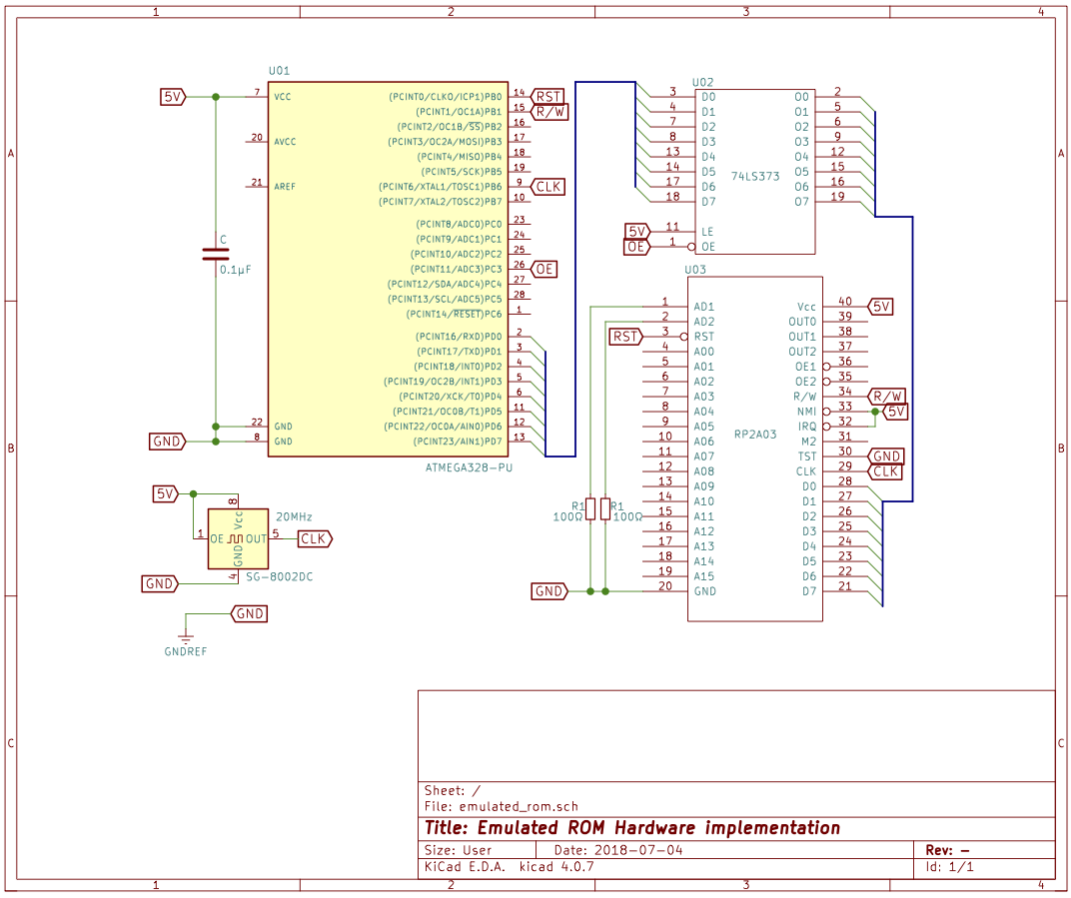

Figure 3: The circuit with the corrected LE pin on the 74LS373 latch.

Table 2: Instruction families and their characteristics

Instruction family	Size of instruction [B]	# Cycles / instruction	R/W Cycle sequence
Immediate	2	2	R, R
Absolute	3	4	R, R, R, R/W
Zero page	2	3	R, R, R/W

These characteristics was then used to redesign the assembly routines and to build the functions mentioned above. When implemented it was possible to send any opcode[2] together with any operand, handled properly by the assembly routines.

6.3.4 Sprint 5, Entire instruction set

The first part of this sprint focused on further extending the functionality of the previously written instruction handling, allowing a 6502 program to be stored and parsed as an array of byte instructions, functioning as a simple emulated ROM. This simplified the setup routine that had to be performed on boot of the RP2A03, as it could simply be written into the ROM, followed by the rest of the RP2A03 instructions we wanted to run.

When analyzing the 6502 instruction set further, we concluded that we were not interested in implementing support for branching instructions that conditionally jumps the program counter. This is because the RP2A03 reads instructions directly from the microcontroller, and as such branching has no real effect.

The second part of the sprint was aimed at thoroughly testing the entire instruction set, making sure that all instructions produced expected results. While all operations that used the accumulator register (immediate loads, ALU[3] operations, etc.) worked consistently and as expected, we noticed that seemingly *none of the memory operations worked*. This is further detailed and discussed in sections Results and [Conclusions & discussion].

The sprint, and some continuous work during the experiments phase, was concluded without managing to resolve this issue.

6.4 Experiments phase

In order to measure time according to our criterias, we attempted to set up the SPI peripheral on the Atmega328P. Unfortunately the programmer we used did not support two-say SPI communication, which forced us to further extend the implementation with a second microcontroller unit.

The implementation was extended in hardware and software to include simple communication with an Arduino M0 Pro board, which was used to measure time. The choice of the Arduino M0 Pro as a hardware timer was the increased resolution of time, and the fact that it ran on a clock with more than double the frequency of the wrapper system, further increasing accuracy of the measurements.

To measure communication timing and cycles between instructions, we used a digital logic analyzer to monitor digital output. The data from the analyzer could then be collected both numerically (in the form of CSV) and as diagrams. On the RP2A03 we chose to monitor all bits of the data bus, as well as the R/W and output clock pins.

The experiments phase was conducted according to design, with the exception that any test cases involving memory operations could not be confirmed to produce expected results. The experiments were performed according to the following steps, and further illustrated in figure 4:

[2]Operation Code, the portion of a machine code instruction that specifies what operation to perform.
[3]Arithmetic and Logic Unit. In a CPU it is the component responsible for arithmetic and logic operations.

1. Validate data output and record per-instruction-overhead when all variables are at default values
2. Measure time of completion as program increases in length
3. Measure time of completion as time interval of reset increases
4. Measure time of completion when program is called with emulated ROM functions
5. Measure time of completion when program is called without emulated ROM functions
6. Switch to next category and repeat process until all categories have been tested

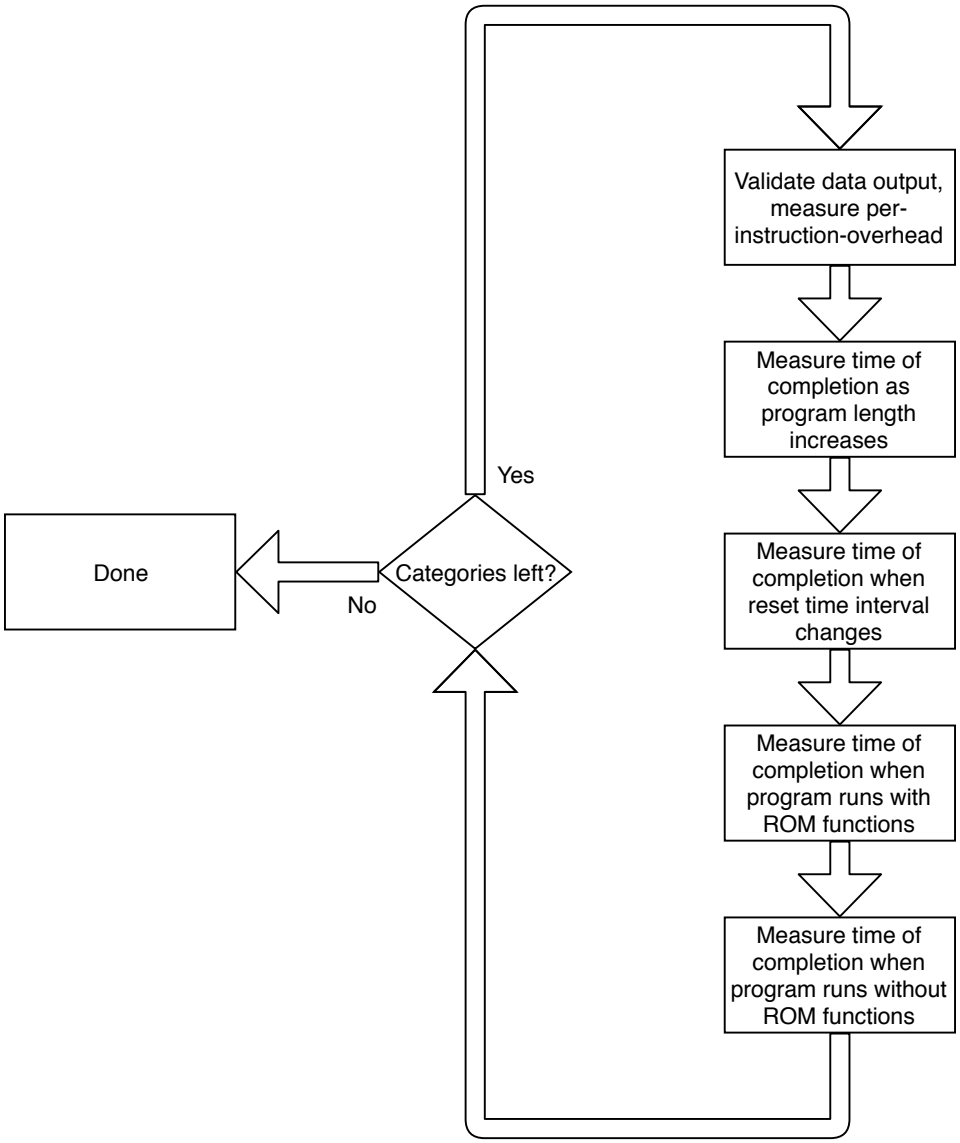

Figure 4: Block diagram illustrating the iterative testing process.

For step 1 we used a digital logic analyzer to measure the bit value on the output of the RP2A03, in order to ensure that an expected value was output. We also recorded the time and number of cycles between instruction for each instruction type at the same time.

For steps 2 through 5 we utilized the Arduino M0 Pro to act as a master controller unit, which we programmed to tell the Atmega328P to start executing programs on the RP2A03 on our command, and at the same time measure the time it took for the Atmega328P to execute. Three different test

programs were written for each controller device (Arduino M0 Pro and Atmega328P), each following the same pattern:

The Arduino waits for the Atmega328P to signal that it has completed its setup routine. The Atmega328P then waits for a start signal from the Arduino, which is sent upon the press of a hardware button. When the button is pressed, the Arduino sends a *go* signal to the Atmega328P and starts a timer. It then waits for the Atmega328P to signal that it has finished it execution, where upon the Arduino will stop it's timer and save the results. When the test program has finished, the Arduino outputs all the measured times.

7 Results

This chapter details the results of all tests that were performed on the system. The tests were performed as outlined in the subchapter Experiments phase, and the test results have been labeled

1. Data validation and per-instruction-overhead
2. Increasing number of instructions
3. Increasing frequency of resets
4. Using ROM-Emulation functions
5. Not using ROM-Emulation functions

, respectively.

Firstly all tests are presented and compared across categories, followed by a comparison of tests for each separate category. The *Mixed* category was not applicable in test case 5, since the task of reading mixed instructions and placing them correctly in the data structures used for communication would effectively result in the exact steps taken by the ROM-emulation functions.

7.1 Comparison: Test cases

Test cases 1 through 5 are shown in figures 5, 6, 7 and 8 respectively. Categories are compared to each other in each figure where applicable. Table 3 details the number of cycles needed per instruction in each category, together with a comparison against the number of cycles needed in an unwrapped system. See Appendix B for tables of raw test result data.

Table 3: Performance measured in Cycles per Instruction

Category	Wrapped	Unwrapped	Difference
Immediate	8	2	400%
Zero Page	8	2	400%
Absolute	9	3	300%

As shown in figure 5, data could not be validated for categories *Zero Page* and *Absolute*, and subsequently not *Mixed*. The instructions sent from the master unit and the execution time on the slave unit behaves as expected, but the data output after execution is not correct. As shown in table 4, the `LDA (0xA5)` instruction loads the value `0x04` from Zero Page address `0x24`, however trying to store the accumulator at another memory address strangely enough outputs `0x24` from the accumulator instead.

Table 4: Illustrating error in memory operations

Data bus	Comment
`0xA5`	Zero Page `LDA` instruction Opcode

Data bus	Comment
0x24	Zero Page LDA Operand, Zero Page Address 0x24
0x04	Returned value 0x04 from memory
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	Zero Page STA instruction Opcode
0x06	Zero Page STA Operand, Zero Page Address 0x06
0x24	Value on accumulator to be written to memory (Expected 0x04)

When not performing memory operations, i.e. accumulator writes and ALU operations, all data was validated as expected, even when performing sequences of connected operations. Table 5 shows a short sequence of instructions storing a value in the accumulator and performing an *Exclusive OR* (EOR) operation on it.

Table 5: Illustrating a sequence of two accumulator operations

Data bus	Comment
0xA9	Immediate LDA instruction Opcode
0x01	Immediate LDA Operand, Immediate value 0x01
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x49	Immediate EOR instruction Opcode
0xFF	Immediate EOR Operand, Immediate value 0xFF
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	"Idle" Zero Page STA instruction
0x85	Zero Page STA instruction Opcode
0x85	Zero Page STA Operand, Zero Page Address 0x85
0xFE	Result of EOR in accumulator to be stored in memory

8 Discussion & conclusions

This chapter will present a summary of our thoughts on the project work and experiments. Firstly we present thoughts on our choice of methods, how effectively we perceived them to be in practice for this particular study, and thoughts on our choice of tools and technology, and the project work in general. Secondly we will discuss the experiments and measured results, and finally present selected conclusions.

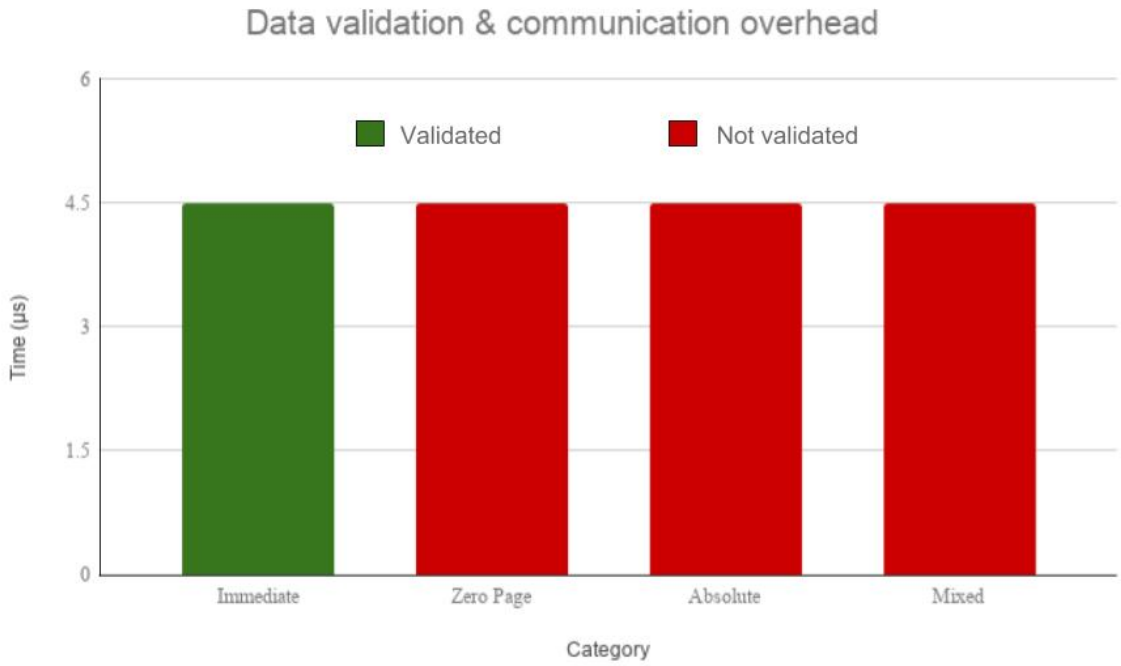

Figure 5: Validated and not validated instruction types, as well as their respective communication overhead.

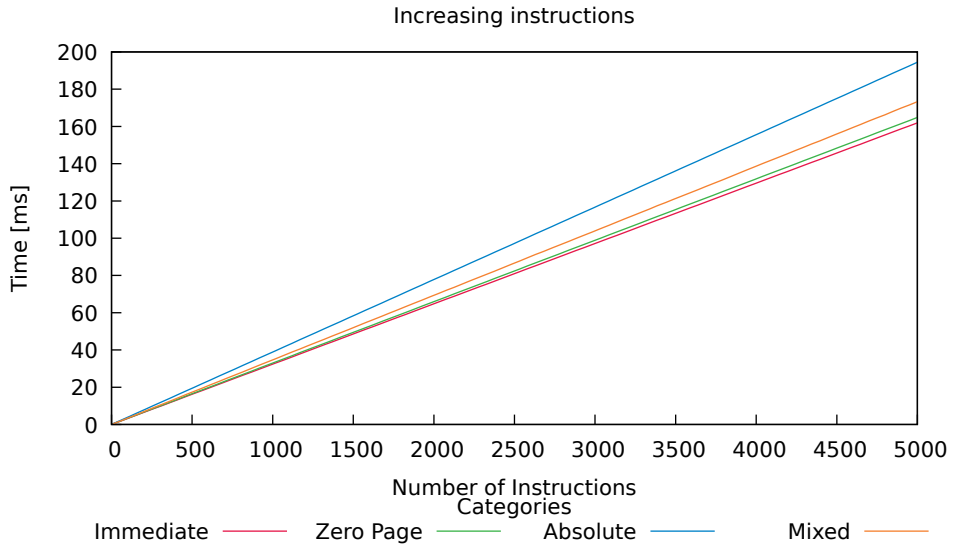

Figure 6: Execution time for each instruction type as a function of number of instructions.

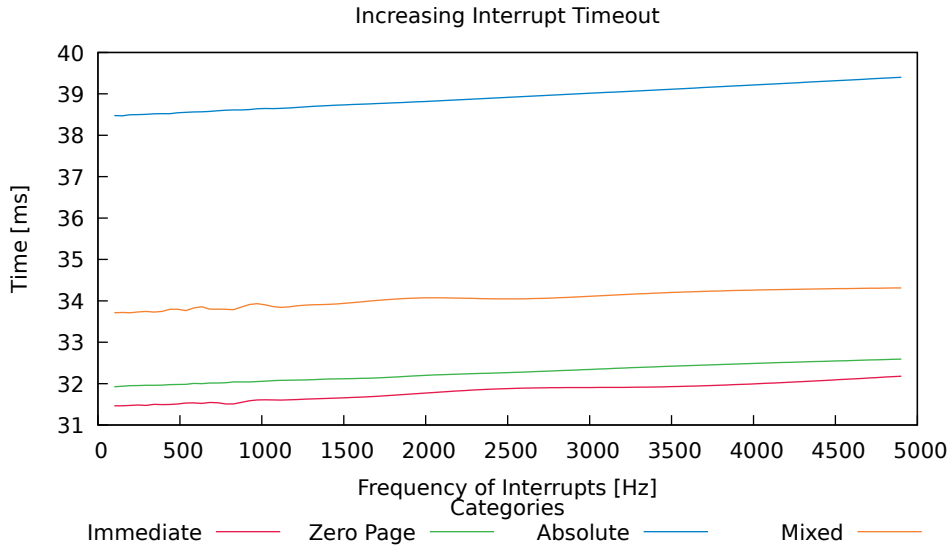

Figure 7: Execution time of 1000 instructions for each instruction type when increasing the frequency of interrupts.

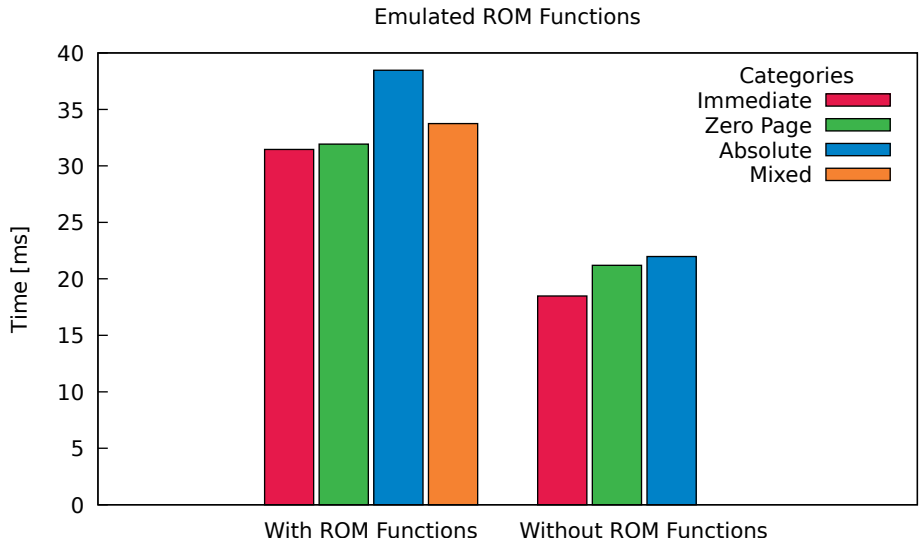

Figure 8: Comparison of each instruction types execution time of 1000 instructions with ROM functions, as well as without ROM functions.

8.1 Project work

8.1.1 Research Methodology

Since experimental research is based on identifying and testing the underlying variables that make up the entire system, much of the project was designed with these variables and how to structure experiments in mind. We found that experimental research was a good fit for this type of problem, however we believe that the process of identifying the variables requires a better theoretical understanding of the target system and the experimental approach than we had at the time of design and implementation. We would suggest a more thorough study of both experimental research philosophy and approach, as well as the system that should be analyzed, before designing an experimental research study of this kind.

When analyzing our results, we evaluated them against our evaluation criteria to draw conclusions. We feel that our conclusions are true, however in hindsight we realize that we should have applied a proven, more rigid framework or methodology in order to avoid potential bias.

8.1.2 Scrum in this project

We had initially planned to implement additional wrapping methods, in order to compare the performance between the different methods of wrapping, but due to technical issues and time limitation we were unable to implement them fully. However, this affected our choice of working method; we believed initially that *Scrum* could give us a good overview of the implementation work across the different wrapping methods, each with their own requirements and tasks. When we realized that it was no longer feasible to continue development of the additional wrapping methods, Scrum was not as effective anymore, since we started to focus more on either the same tasks or closely related tasks. This also meant since that we could continuously plan and discuss the iterative process, we no longer had much use for sprint and daily meetings. For projects of this limited scope, we believe that there may not be a need for more involved frameworks such as *Scrum*.

8.1.3 Arduino M0 Pro

As mentioned previously in the text, the idea for this project was sparked by a hobby project which was to implement a portable music tracker utilizing the NES processor Ricoh 2A03, together with a microcontroller. Initially, we set out to use an Arduino unit in this hobby project, but after experimenting with an Arduino M0 Pro we realized that, due to the lack of documentation and its limited flexibility, we'd much prefer a microcontroller with a more detailed documentation where we could, in a more familiar manner, create our own complementary library code and use a compiler of our choosing. We settled for the Atmega328P as the master controller in the circuit. The Arduino M0 Pro saw a comeback though, as it was used in the test phase of the project and proved to be quite useful for programming simple tasks, and its high clock frequency allowed us to produce more accurate measurements.

8.1.4 Atmega328P

The Atmega328P was useful in our project, as it is fairly easy to program (AVR architecture), and is very well documented. It is also the same processor that the NESizer2 method implements (with minor differences which do not affect the implementation), meaning that we only had to convert some of the source code, and that the assembly routines could be used as is. We felt that the choice was appropriate for two reasons: (i) it made the software easier to implement, and (ii) we could stay closer to the source code that we wish to evaluate.

8.1.5 Programming toolchain

During the research phase, we discovered that that a popular compiler for AVR microcontrollers was AVRDUDE [26], a cross-platform programming software compatible with a wide range of programmers. It needs a compiled hex file, meaning a separate compiler is required. We used avr-gcc [27] for compiling our source code, and we used AVR Libc [28] which provides library functions for AVR microprocessors. As avr-gcc is a C compiler, a language both of us are comfortable with, and as all of the above mentioned tools are documented and widely used, we felt that this entire toolchain was the obvious choice.

We found, also during the research stage, that a popular programmer for AVR micro controllers was the USBasp [29], which we obtained and used for the project. One negative aspect of this was that there was no serial interface to use for debugging, something we did not notice until later in the project. Had we done more thorough research earlier in the project work, we could found and used a different programmer that included this functionality so that we did not have to write Arduino programs for the testing phase, which could have saved us some time.

8.1.6 Oscilloscope, Digital logic analyzer

In attempts to verify that the RP2A03 would execute a basic program, we initially used an oscilloscope to monitor the R/W pin to see if it behaved according to the input instructions, which we were successful in doing. However we could not find any good way of validating our input and output data. After consulting with our supervisor, we were provided with a digital logic analyzer. This helped us immensely in moving forward with the project, as we could validate that our inputs produced expected outputs, and we would recommend any reproducing persons to use one as well.

8.2 Experiments & Results

8.2.1 Data validation & Memory operations

As shown in the Results section, a major part of the intended functionality did not produce correct results. Any program or sequence of instructions involving memory operations led to faulty data. Although we were unable to determine the cause of this error, we have established two possible explanations; (i) a bug in the software implementation of memory operations, or (ii) a defect memory unit of the RP2A03. Since this error did not occur for operations involving the ALU, accumulator or system level operations (i.e. disable interrupts etc.), and only for memory operations independent of instruction size, case (i) seems unlikely. We could also confirm that the data on the data bus, and the behavior of the R/W pin, was indicative of the correct instructions being sent at correct timing, which further makes (i) a less likely cause. However, since we did not have access to a replacement RP2A03 unit, or diagnostic tools to investigate the hardware of the unit we used, we were unable to rule out the RP2A03 as the cause either. Fortunately, as this error only affected the actual stored and loaded data and not the machines behavior in terms of cycles per instruction etc., we could still use memory operations in our test programs in order to measure timing.

8.2.2 Minimum communication overhead

We could see a minimum communication overhead, measured as the time between each consecutive instruction in a program, of 4.5 μs. This time was independent of instruction type and size, and seems to affect each instruction equally. We found that this minimum communication overhead (or possibly main bottleneck of the communication method) was made up of two sets of the idle instruction, running over a total of six RP2A03 CPU cycles. An example of this can be seen in the Results section, in Table 5. Since each instruction sending starts with a synchronization step and ends by placing the idle instruction on the data bus before returning to the calling function, the lowest theoretical overhead is one full idle instruction. The resulting performance hit can be seen in table 3 which shows as much as 400% difference at least.

However, upon returning to the main program, the master unit will have to load a new instruction and turn of all interrupts before initiating communication again, and we believe that the time it takes for the master unit to perform these steps causes it to initiate communications somewhere during the second idle instruction. Between instructions, the master unit also has to handle timer interrupts and perform periodic resets of the RP2A03 program counter. This will cause longer times between instructions at periodic intervals. We decided not to include this in our results since it occurs relatively infrequently, and is highly dependent on the settings for the timer interrupts.

If it is possible to increase the frequency gap between the master and slave unit, it may be possible to further decrease the minimum communication overhead. This could perhaps be done by using a master unit that runs on a higher frequency in combination with a greater clock divider. Since our implementation requires six whole cycles between each contiguous instruction, it is clearly not suited for systems where timing is critical as of its current state. It is possible that this can be remedied by modifying the communication protocol; given a reset of the slave unit, the system should theoretically be able to continuously send instructions by simply fetching and putting one byte at a time on the data bus, as long as the sequence of instruction fits within a full period without the need for a reset. We can however not confirm this hypothesis, and it is possible that the communication protocol can only guarantee synchronization for shorter bursts of data.

8.2.3 Running times of test programs

The behavior of the system when increasing the program length was as expected, and we could observe a linear increase with increasing the number of instructions. The Absolute and Mixed category had a longer execution time than the other categories which is to be expected, considering that absolute addressing mode instructions require 4 Bytes. Interestingly, we can observe a difference between Zero Page and Immediate categories, even though the instructions are of the same length. It would have been beneficial to compare against a regular ROM to see if this is normal behavior or not.

As for increasing the frequency of the timer interrupts, that are needed to reset the program counter of the slave unit, it produced hardly any noticeable difference. This means that the settings for the timer could be set to trigger more often, in order to further ensure a safety margin of not accidentally interpreting internal memory data as instruction data, without a significant performance hit.

9 Future work

First and foremost, we acknowledge that the proposed method of hardware wrapping presented in this work shows inconclusive results. In order to tell whether this method is at all viable, it is required to (i) investigate whether the "broken" memory operations encountered during the tests was caused by a faulty RP2A03 unit, or if it was because of an error in the proposed implementation, and (ii) compare it against other methods of hardware wrapping on roughly equal level of complexity in implementation. As for case (ii), we had originally intended to compare this implementation against a different method of wrapping, which used a *shared memory* to communicate between the master and slave unit, however we were unable to complete it because of the aforementioned problematic memory operations. The schematics for the shared memory approach can be found in Appendix A.

A natural step from our work would be to expand the functionality further by allowing for a complete instruction set. As discussed in section Sprint 5, Entire instruction set, there is no need for branching or jumping instructions, but there are still a large number of instructions left that was outside the scope of our study. When all relevant parts of the instruction set has been implemented, the final step(s) would be to test a real application of the method, and test how well this method works in practice.

10 References

11 Appendix

TODO The following contains rövsex, figures, source code etc. Appendix A contains a, B b, C c osv ne?

11.1 Appendix A

In this section, the flowcharts and circuit diagrams that were created during the research design and design phase are compiled, as well as revised versions from later in the project work.

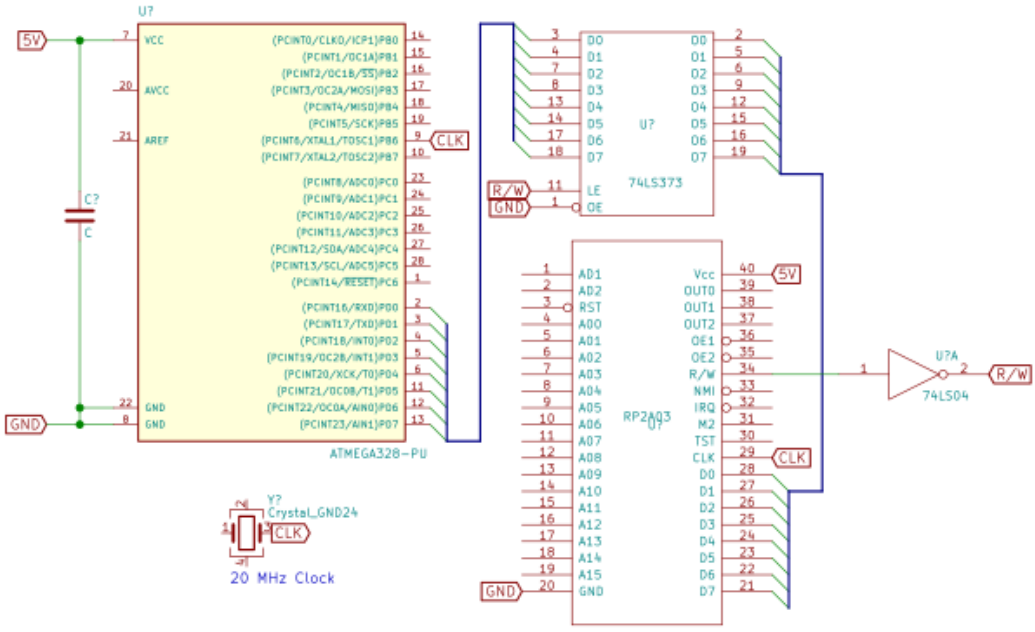

Figure 9: The first iteration of the circuit, with the misplaced LE pin on the 74LS373 latch .

11.2 Appendix B

This section contains the raw data collected during the experiment phase, each presented in a table.

Table 6: Time to execute 1000 instructions on default settings with ROM functions, measured in ms

No. of instructi	ons Abso	lute Imme	diate Zero	Page Mixed
1000	38.47	31.45	31.93	33.74

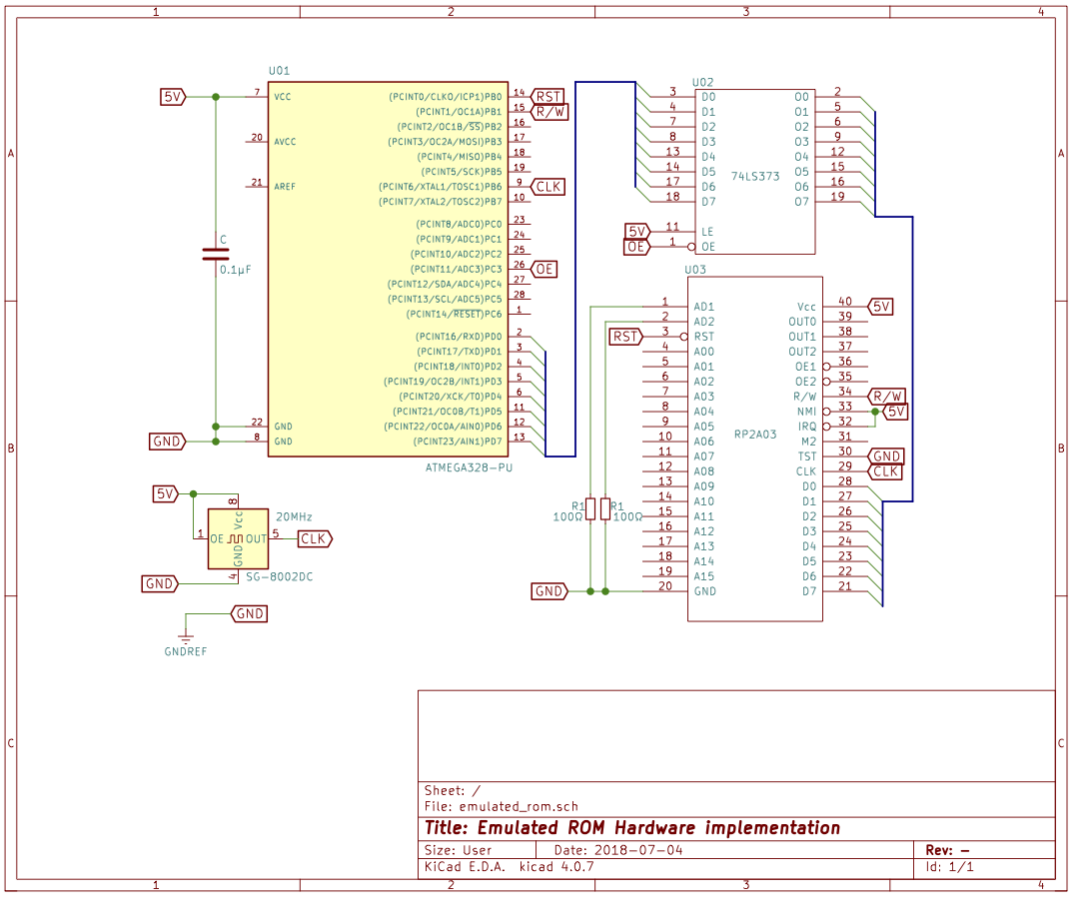

Figure 10: The second iteration of the circuit, with the corrected LE pin on the 74LS373 latch.

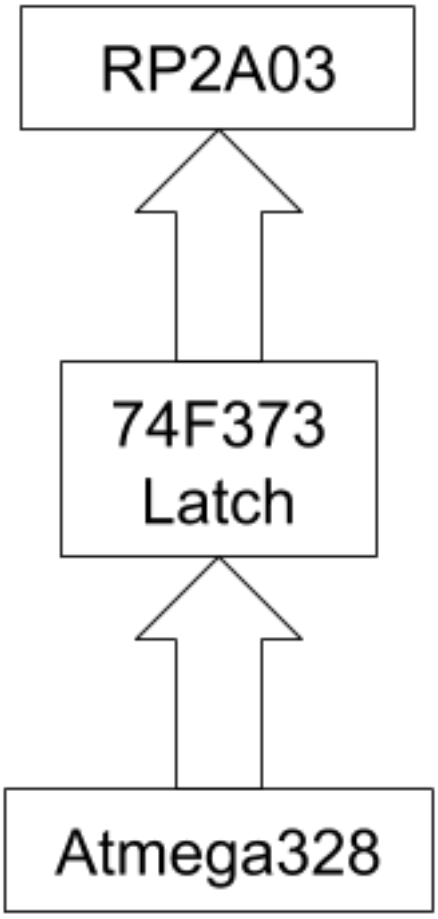

Figure 11: Block diagram of the hardware components and the communication channels of the analyzed implementation.

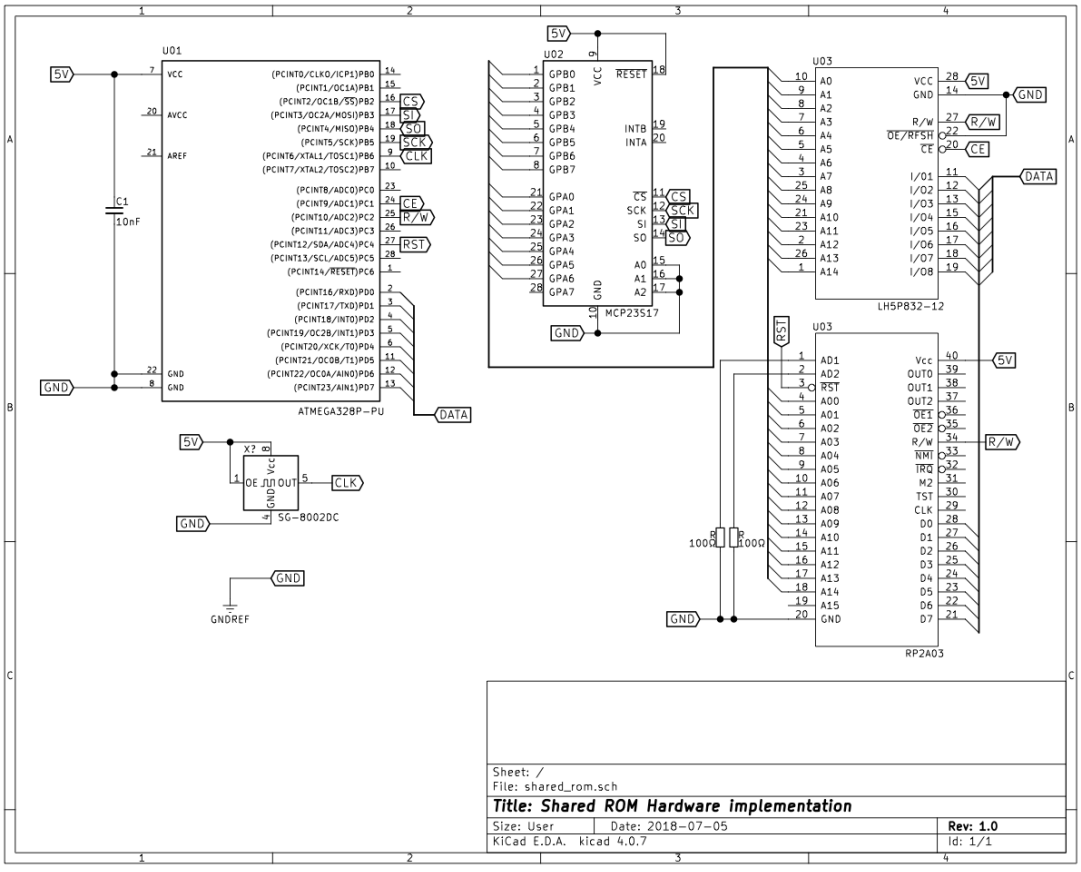

Figure 12: Circuit diagram of the cancelled shared memory implementation.

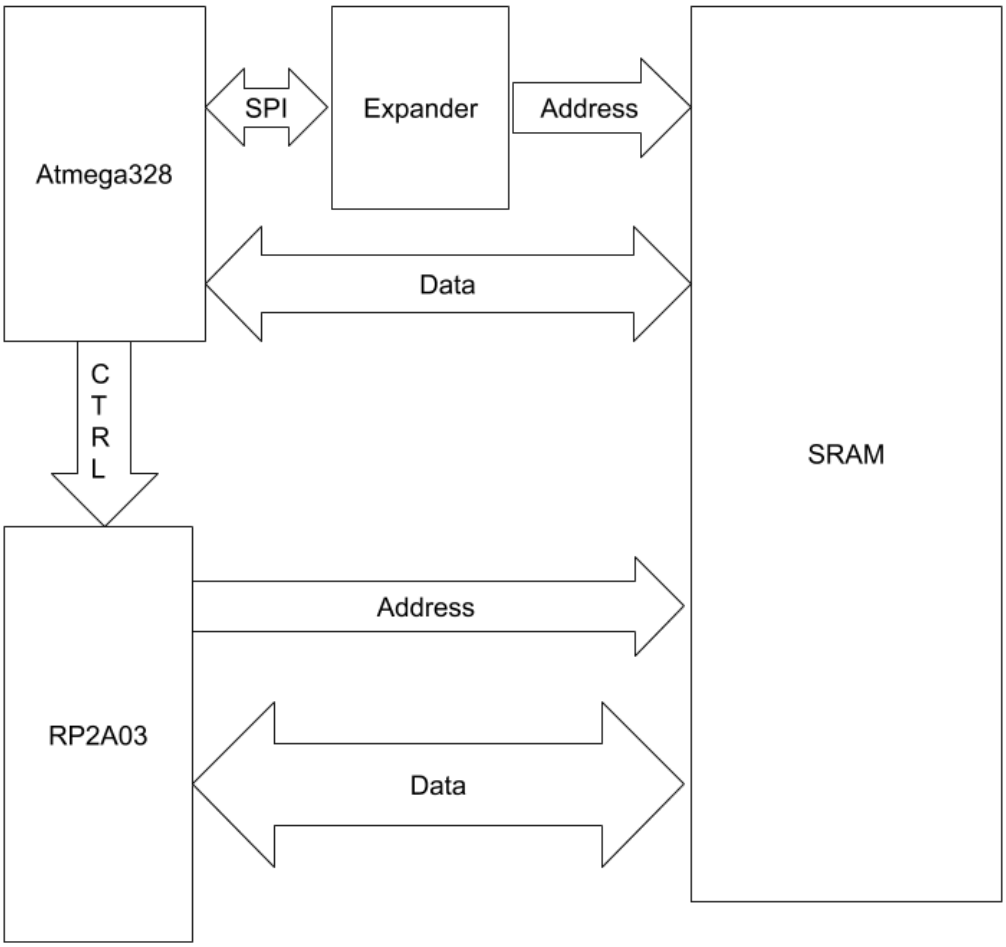

Figure 13: Block diagram of the hardware components and the communication channels of the cancelled shared memory implementation.

Table 7: Time to execute 1000 instructions on default settings without ROM functions, measured in ms

No. of instructi	ons Abso	lute Imme	diate Zero Page
1000	21.98	18.48	21.20

Table 8: Time to execute programs of varying instruction size, ranging from 10 to 5000, measured in ms

No. of instructions	Absolute	Immediat	e Zero Pag	e Mixed
10 0.396	0.337	0.333	0.363	
20 0.784	0.656	0.667	0.712	
30 1.173	0.980	0.996	1.030	
40 1.563	1.297	1.326	1.352	
50 1.952	1.627	1.653	1.791	
60 2.336	1.948	1.983	2.032	
70 2.724	2.276	2.316	2.484	
80 3.116	2.594	2.643	2.732	
90 3.506	2.921	2.976	3.149	
100 3.892	3.240	3.306	3.508	
110 4.286	3.568	3.641	3.803	
120 4.679	3.889	3.958	4.161	
130 5.067	4.215	4.293	4.518	
140 5.454	4.544	4.618	4.870	
150 5.837	4.865	4.959	5.214	
160 6.228	5.182	5.282	5.489	
170 6.620	5.517	5.603	5.922	
180 7.020	5.836	5.939	6.290	
190 7.405	6.158	6.272	6.532	
200 7.788	6.483	6.599	6.983	
210 8.180	6.807	6.933	7.265	
220 8.560	7.127	7.261	7.595	
230 8.953	7.462	7.583	7.991	
240 9.348	7.773	7.924	8.359	
250 9.737	8.112	8.241	8.680	
260 10.124	8.424	8.575	9.012	
270 10.511	8.749	8.908	9.330	
280 10.910	9.075	9.236	9.677	
290 11.293	9.396	9.569	10.092	
300 11.678	9.720	9.894	10.437	
310 12.075	10.041	10.227	10.687	
320 12.464	10.364	10.552	11.157	
330 12.845	10.698	10.880	11.393	
340 13.239	11.020	11.212	11.820	
350 13.624	11.340	11.545	12.157	
360 14.017	11.664	11.875	12.467	
370 14.397	11.984	12.203	12.810	
380 14.797	12.302	12.530	13.175	
390 15.181	12.642	12.861	13.527	
400 15.572	12.957	13.193	13.886	

No. of instructions	Absolute	Immediat	e Zero Pag	e Mixed
410 15.961	13.281	13.521	14.233	
420 16.355	13.608	13.850	14.519	
430 16.733	13.937	14.183	14.902	
440 17.122	14.250	14.509	15.303	
450 17.518	14.573	14.840	15.548	
460 17.903	14.892	15.168	15.981	
470 18.296	15.228	15.503	16.295	
480 18.683	15.551	15.830	16.605	
490 19.075	15.878	16.166	16.961	
500 19.460	16.198	16.492	17.363	
510 19.842	16.519	16.817	17.677	
520 20.238	16.855	17.145	18.013	
530 20.622	17.174	17.479	18.380	
540 21.016	17.495	17.813	18.751	
550 21.411	17.824	18.141	19.087	
560 21.792	18.142	18.462	19.331	
570 22.189	18.464	18.803	19.819	
580 22.568	18.791	19.132	20.051	
590 22.965	19.118	19.451	20.531	
600 23.352	19.435	19.782	20.766	
610 23.740	19.768	20.111	21.122	
620 24.127	20.076	20.446	21.507	
630 24.518	20.412	20.773	21.853	
640 24.903	20.732	21.107	22.190	
650 25.294	21.056	21.432	22.516	
660 25.689	21.382	21.768	22.870	
670 26.072	21.709	22.093	23.249	
680 26.455	22.030	22.422	23.607	
690 26.857	22.355	22.753	23.843	
700 27.242	22.679	23.088	24.323	
710 27.629	22.991	23.415	24.542	
720 28.017	23.324	23.740	25.037	
730 28.405	23.650	24.073	25.281	
740 28.795	23.963	24.398	25.622	
750 29.182	24.290	24.726	25.989	
760 29.572	24.624	25.059	26.363	
770 29.963	24.937	25.385	26.693	
780 30.343	25.266	25.725	27.021	
790 30.743	25.602	26.044	27.415	
800 31.128	25.910	26.379	27.767	
810 31.520	26.234	26.705	28.074	
820 31.904	26.568	27.034	28.388	
830 32.296	26.888	27.371	28.823	
840 32.675	27.214	27.696	29.069	
850 33.075	27.532	28.024	29.465	
860 33.459	27.859	28.355	29.855	
870 33.853	28.179	28.684	30.136	
880 34.237	28.503	29.016	30.483	
890 34.632	28.829	29.350	30.848	
900 35.017	29.146	29.671	31.201	

No. of instructions	Absolute	Immediat	e Zero Pag	e Mixed
910 35.408	29.483	30.002	31.560	
920 35.792	29.798	30.334	31.913	
930 36.179	30.119	30.661	32.219	
940 36.579	30.452	30.993	32.555	
950 36.953	30.773	31.320	32.979	
960 37.347	31.098	31.664	33.213	
970 37.737	31.417	31.979	33.688	
980 38.130	31.736	32.313	33.932	
990 38.516	32.062	32.641	34.283	
1000 38.9	10 32.3	84 32.9	75 34.6	98
1010 39.2	97 32.7	06 33.3	01 35.0	29
1020 39.6	81 33.0	37 33.6	29 35.3	63
1030 40.0	74 33.3	62 33.9	57 35.6	93
1040 40.4	59 33.6	91 34.2	88 36.0	18
1050 40.8	51 34.0	10 34.6	15 36.3	76
1060 41.2	43 34.3	39 34.9	55 36.7	84
1070 41.6	27 34.6	55 35.2	79 37.1	02
1080 42.0	17 34.9	78 35.6	03 37.3	87
1090 42.4	00 35.3	02 35.9	38 37.8	36
1100 42.7	93 35.6	25 36.2	66 38.0	85
1110 43.1	78 35.9	60 36.5	98 38.4	54
1120 43.5	77 36.2	77 36.9	24 38.8	82
1130 43.9	59 36.5	97 37.2	58 39.1	49
1140 44.3	54 36.9	25 37.5	86 39.5	05
1150 44.7	41 37.2	63 37.9	11 39.8	62
1160 45.1	30 37.5	68 38.2	47 40.2	17
1170 45.5	18 37.8	92 38.5	71 40.5	64
1180 45.9	10 38.2	23 38.9	07 40.8	69
1190 46.2	94 38.5	41 39.2	33 41.2	27
1200 46.6	86 38.8	59 39.5	59 41.6	42
1210 47.0	69 39.1	79 39.8	89 41.8	81
1220 47.4	65 39.5	07 40.2	21 42.3	46
1230 47.8	50 39.8	27 40.5	48 42.5	82
1240 48.2	36 40.1	54 40.8	77 43.0	14
1250 48.6	26 40.4	80 41.2	05 43.3	40
1260 49.0	17 40.8	12 41.5	38 43.6	59
1270 49.4	10 41.1	38 41.8	71 44.0	12
1280 49.7	98 41.4	49 42.1	96 44.3	62
1290 50.1	82 41.7	82 42.5	28 44.7	23
1300 50.5	71 42.1	10 42.8	64 45.0	70
1310 50.9	65 42.4	35 43.1	89 45.3	89
1320 51.3	46 42.7	55 43.5	16 45.7	13
1330 51.7	41 43.0	87 43.8	46 46.1	52
1340 52.1	27 43.4	08 44.1	79 46.3	86
1350 52.5	17 43.7	28 44.5	08 46.8	50
1360 52.9	09 44.0	58 44.8	39 47.0	94
1370 53.2	96 44.3	66 45.1	65 47.5	09
1380 53.6	85 44.6	91 45.5	02 47.8	60
1390 54.0	76 45.0	24 45.8	26 48.1	68
1400 54.4	59 45.3	45 46.1	56 48.5	18

No. of instructions	Absolute	Immediat	e Zero Pag	e Mixed
1410 54.8	49 45.6	68 46.4	78 48.8	72
1420 55.2	36 45.9	87 46.8	15 49.2	29
1430 55.6	25 46.3	14 47.1	43 49.5	74
1440 56.0	27 46.6	46 47.4	74 49.8	47
1450 56.4	05 46.9	65 47.8	03 50.2	83
1460 56.7	99 47.2	99 48.1	33 50.6	41
1470 57.1	80 47.6	15 48.4	63 50.8	90
1480 57.5	74 47.9	30 48.7	96 51.3	41
1490 57.9	62 48.2	56 49.1	25 51.6	27
1500 58.3	57 48.5	83 49.4	52 51.9	39
1510 58.7	35 48.8	92 49.7	79 52.3	49
1520 59.1	31 49.2	29 50.1	15 52.7	06
1530 59.5	12 49.5	51 50.4	34 53.0	30
1540 59.9	04 49.8	86 50.7	64 53.3	72
1550 60.3	04 50.2	06 51.0	92 53.6	95
1560 60.6	80 50.5	29 51.4	31 54.0	34
1570 61.0	70 50.8	59 51.7	61 54.4	49
1580 61.4	62 51.1	77 52.0	83 54.7	97
1590 61.8	45 51.5	00 52.4	14 55.0	46
1600 62.2	39 51.8	12 52.7	48 55.5	17
1610 62.6	26 52.1	40 53.0	73 55.7	52
1620 63.0	14 52.4	58 53.4	05 56.1	73
1630 63.4	03 52.7	87 53.7	41 56.5	15
1640 63.7	98 53.1	13 54.0	73 56.8	26
1650 64.1	82 53.4	42 54.4	07 57.1	63
1660 64.5	72 53.7	70 54.7	19 57.5	35
1670 64.9	60 54.0	88 55.0	58 57.8	94
1680 65.3	53 54.4	10 55.3	92 58.2	46
1690 65.7	37 54.7	31 55.7	18 58.5	91
1700 66.1	32 55.0	70 56.0	41 58.8	81
1710 66.5	19 55.3	79 56.3	82 59.2	68
1720 66.9	13 55.7	15 56.7	08 59.6	64
1730 67.2	94 56.0	35 57.0	31 59.9	06
1740 67.6	83 56.3	47 57.3	60 60.3	41
1750 68.0	78 56.6	74 57.6	95 60.6	58
1760 68.4	64 56.9	99 58.0	20 60.9	70
1770 68.8	51 57.3	28 58.3	45 61.3	21
1780 69.2	44 57.6	55 58.6	85 61.7	10
1790 69.6	29 57.9	67 59.0	08 62.0	34
1800 70.0	24 58.2	89 59.3	49 62.3	70
1810 70.4	07 58.6	17 59.6	75 62.7	43
1820 70.7	96 58.9	34 60.0	01 63.1	16
1830 71.1	87 59.2	57 60.3	34 63.4	49
1840 71.5	72 59.5	93 60.6	64 63.6	92
1850 71.9	60 59.9	09 60.9	90 64.1	85
1860 72.3	52 60.2	39 61.3	24 64.4	09
1870 72.7	40 60.5	56 61.6	50 64.8	89
1880 73.1	30 60.8	88 61.9	76 65.1	15
1890 73.5	20 61.2	01 62.3	11 65.4	82
1900 73.9	07 61.5	37 62.6	41 65.8	61

No. of instructions	Absolute	Immediat	e Zero Pag	e Mixed
1910 74.3	03 61.8	54 62.9	68 66.2	21
1920 74.6	83 62.1	88 63.2	98 66.5	48
1930 75.0	70 62.5	07 63.6	29 66.8	67
1940 75.4	60 62.8	42 63.9	51 67.2	28
1950 75.8	44 63.1	51 64.2	81 67.6	02
1960 76.2	38 63.4	68 64.6	09 67.9	69
1970 76.6	28 63.8	02 64.9	42 68.2	02
1980 77.0	18 64.1	17 65.2	74 68.6	84
1990 77.4	05 64.4	48 65.6	00 68.9	13
2000 77.7	97 64.7	72 65.9	28 69.3	99
2010 78.1	84 65.0	90 66.2	72 69.6	43
2020 78.5	72 65.4	14 66.5	96 69.9	76
2030 78.9	66 65.7	44 66.9	24 70.3	39
2040 79.3	56 66.0	84 67.2	53 70.7	21
2050 79.7	41 66.4	00 67.5	81 71.0	44
2060 80.1	31 66.7	17 67.9	13 71.3	77
2070 80.5	20 67.0	45 68.2	37 71.7	66
2080 80.9	08 67.3	76 68.5	72 72.1	25
2090 81.2	87 67.6	91 68.9	01 72.4	30
2100 81.6	84 68.0	06 69.2	28 72.7	44
2110 82.0	71 68.3	28 69.5	61 73.1	83
2120 82.4	60 68.6	52 69.8	91 73.4	22
2130 82.8	55 68.9	74 70.2	18 73.8	27
2140 83.2	39 69.3	15 70.5	56 74.2	02
2150 83.6	33 69.6	28 70.8	84 74.5	00
2160 84.0	20 69.9	52 71.2	12 74.8	43
2170 84.4	08 70.2	94 71.5	45 75.2	08
2180 84.7	99 70.5	90 71.8	75 75.5	60
2190 85.1	83 70.9	35 72.1	98 75.9	22
2200 85.5	73 71.2	49 72.5	31 76.2	59
2210 85.9	65 71.5	71 72.8	53 76.5	83
2220 86.3	48 71.9	00 73.1	87 76.9	09
2230 86.7	45 72.2	28 73.5	13 77.3	44
2240 87.1	30 72.5	48 73.8	42 77.5	78
2250 87.5	22 72.8	58 74.1	69 78.0	51
2260 87.9	04 73.1	83 74.5	03 78.2	90
2270 88.2	99 73.5	07 74.8	36 78.6	34
2280 88.6	89 73.8	31 75.1	66 79.0	53
2290 89.0	72 74.1	62 75.4	93 79.3	93
2300 89.4	66 74.4	87 75.8	26 79.7	27
2310 89.8	44 74.8	12 76.1	57 80.0	45
2320 90.2	44 75.1	57 76.4	84 80.3	75
2330 90.6	28 75.4	51 76.8	10 80.7	42
2340 91.0	17 75.7	84 77.1	49 81.1	43
2350 91.4	08 76.1	04 77.4	73 81.4	59
2360 91.7	91 76.4	25 77.8	00 81.7	47
2370 92.1	91 76.7	54 78.1	28 82.1	99
2380 92.5	68 77.0	77 78.4	68 82.4	41
2390 92.9	66 77.4	07 78.7	89 82.8	03
2400 93.3	59 77.7	32 79.1	19 83.2	38

No. of instructions	Absolute	Immediat	e Zero Pag	e Mixed
2410 93.7	39 78.0	56 79.4	44 83.5	17
2420 94.1	25 78.3	86 79.7	78 83.8	64
2430 94.5	25 78.7	08 80.1	19 84.2	18
2440 94.9	07 79.0	28 80.4	40 84.5	73
2450 95.2	99 79.3	50 80.7	64 84.9	23
2460 95.6	87 79.6	65 81.0	99 85.2	34
2470 96.0	81 80.0	00 81.4	30 85.5	76
2480 96.4	64 80.3	22 81.7	52 85.9	90
2490 96.8	57 80.6	55 82.0	87 86.2	38
2500 97.2	38 80.9	62 82.4	16 86.7	02
2510 97.6	27 81.2	88 82.7	51 86.9	38
2520 98.0	24 81.6	15 83.0	74 87.3	75
2530 98.4	07 81.9	32 83.4	03 87.7	08
2540 98.7	97 82.2	59 83.7	27 88.0	11
2550 99.1	84 82.5	91 84.0	71 88.3	65
2560 99.5	76 82.9	11 84.3	90 88.7	19
2570 99.9	61 83.2	38 84.7	24 89.0	79
2580 100.	356 83.5	70 85.0	62 89.4	34
2590 100.	751 83.8	94 85.3	80 89.7	41
2600 101.	131 84.2	14 85.7	20 90.0	73
2610 101.	517 84.5	31 86.0	47 90.5	09
2620 101.	904 84.8	54 86.3	69 90.7	51
2630 102.	294 85.1	81 86.7	04 91.2	17
2640 102.	690 85.4	79 87.0	34 91.4	47
2650 103.	076 85.8	25 87.3	56 91.8	70
2660 103.	466 86.1	44 87.6	89 92.2	20
2670 103.	852 86.4	63 88.0	22 92.5	22
2680 104.	238 86.7	93 88.3	48 92.8	79
2690 104.	621 87.1	16 88.6	82 93.2	28
2700 105.	024 87.4	32 89.0	18 93.5	81
2710 105.	405 87.7	64 89.3	43 93.9	27
2720 105.	795 88.1	01 89.6	68 94.2	10
2730 106.	192 88.4	08 89.9	93 94.6	41
2740 106.	571 88.7	49 90.3	34 95.0	11
2750 106.	963 89.0	71 90.6	56 95.2	52
2760 107.	353 89.3	84 90.9	87 95.6	99
2770 107.	741 89.7	17 91.3	17 95.9	89
2780 108.	133 90.0	55 91.6	38 96.3	07
2790 108.	515 90.3	50 91.9	73 96.7	13
2800 108.	914 90.6	72 92.3	14 97.0	73
2810 109.	292 91.0	08 92.6	31 97.3	99
2820 109.	686 91.3	25 92.9	63 97.7	28
2830 110.	078 91.6	52 93.2	97 98.0	63
2840 110.	462 91.9	71 93.6	27 98.3	99
2850 110.	855 92.2	84 93.9	56 98.8	15
2860 111.	245 92.6	25 94.2	87 99.1	58
2870 111.	629 92.9	47 94.6	07 99.4	00
2880 112.	022 93.2	76 94.9	41 99.8	75
2890 112.	411 93.6	15 95.2	71 100.	112
2900 112.	793 93.9	16 95.6	00 100.	532

No. of instructions	Absolute	Immediat	e Zero Pag	e Mixed
2910 113.	189 94.2	57 95.9	35 100.	871
2920 113.	573 94.5	55 96.2	66 101.	187
2930 113.	962 94.9	06 96.5	96 101.	533
2940 114.	357 95.2	01 96.9	18 101.	891
2950 114.	744 95.5	42 97.2	44 102.	235
2960 115.	133 95.8	63 97.5	71 102.	603
2970 115.	515 96.1	89 97.9	12 102.	946
2980 115.	905 96.5	09 98.2	35 103.	238
2990 116.	294 96.8	27 98.5	65 103.	627
3000 116.	690 97.1	46 98.9	03 104.	015
3010 117.	070 97.4	77 99.2	26 104.	268
3020 117.	460 97.8	14 99.5	58 104.	701
3030 117.	859 98.1	17 99.8	93 105.	018
3040 118.	246 98.4	49 100.	222 105.	330
3050 118.	625 98.7	72 100.	553 105.	680
3060 119.	023 99.0	99 100.	874 106.	072
3070 119.	407 99.4	27 101.	205 106.	400
3080 119.	797 99.7	49 101.	532 106.	726
3090 120.	185 100.	077 101.	863 107.	096
3100 120.	573 100.	394 102.	193 107.	471
3110 120.	970 100.	724 102.	518 107.	813
3120 121.	347 101.	043 102.	850 108.	049
3130 121.	746 101.	364 103.	186 108.	545
3140 122.	128 101.	700 103.	513 108.	772
3150 122.	516 102.	022 103.	853 109.	242
3160 122.	909 102.	330 104.	175 109.	480
3170 123.	300 102.	656 104.	500 109.	841
3180 123.	688 102.	993 104.	834 110.	220
3190 124.	073 103.	314 105.	157 110.	576
3200 124.	471 103.	639 105.	488 110.	905
3210 124.	850 103.	951 105.	814 111.	229
3220 125.	235 104.	296 106.	145 111.	586
3230 125.	630 104.	614 106.	477 111.	967
3240 126.	017 104.	940 106.	810 112.	328
3250 126.	409 105.	269 107.	140 112.	560
3260 126.	798 105.	560 107.	479 113.	047
3270 127.	187 105.	898 107.	801 113.	258
3280 127.	578 106.	223 108.	131 113.	751
3290 127.	974 106.	553 108.	459 113.	998
3300 128.	356 106.	870 108.	787 114.	331
3310 128.	746 107.	194 109.	117 114.	701
3320 129.	130 107.	504 109.	449 115.	082
3330 129.	521 107.	842 109.	778 115.	408
3340 129.	912 108.	171 110.	117 115.	735
3350 130.	296 108.	486 110.	430 116.	132
3360 130.	688 108.	814 110.	770 116.	476
3370 131.	077 109.	145 111.	092 116.	790
3380 131.	472 109.	466 111.	419 117.	100
3390 131.	850 109.	794 111.	758 117.	549
3400 132.	245 110.	088 112.	075 117.	792

No. of instructions	Absolute	Immediat	e Zero Pag	e Mixed
3410 132.	630 110.	443 112.	408 118.	179
3420 133.	018 110.	753 112.	743 118.	567
3430 133.	406 111.	081 113.	084 118.	849
3440 133.	802 111.	400 113.	403 119.	200
3450 134.	187 111.	726 113.	743 119.	565
3460 134.	572 112.	055 114.	065 119.	917
3470 134.	969 112.	387 114.	398 120.	286
3480 135.	354 112.	675 114.	725 120.	621
3490 135.	741 113.	027 115.	049 120.	938
3500 136.	138 113.	350 115.	381 121.	275
3510 136.	517 113.	680 115.	709 121.	701
3520 136.	905 113.	988 116.	041 121.	925
3530 137.	294 114.	312 116.	368 122.	411
3540 137.	684 114.	646 116.	707 122.	647
3550 138.	075 114.	963 117.	033 122.	990
3560 138.	464 115.	291 117.	358 123.	403
3570 138.	865 115.	608 117.	686 123.	756
3580 139.	241 115.	947 118.	018 124.	075
3590 139.	638 116.	262 118.	346 124.	408
3600 140.	016 116.	576 118.	683 124.	740
3610 140.	412 116.	908 119.	006 125.	104
3620 140.	795 117.	231 119.	346 125.	492
3630 141.	183 117.	565 119.	665 125.	816
3640 141.	580 117.	872 119.	992 126.	103
3650 141.	968 118.	203 120.	322 126.	560
3660 142.	356 118.	530 120.	657 126.	803
3670 142.	737 118.	850 120.	980 127.	174
3680 143.	130 119.	174 121.	308 127.	600
3690 143.	521 119.	499 121.	647 127.	878
3700 143.	911 119.	827 121.	975 128.	221
3710 144.	297 120.	154 122.	303 128.	568
3720 144.	687 120.	467 122.	633 128.	928
3730 145.	080 120.	791 122.	970 129.	282
3740 145.	464 121.	125 123.	289 129.	584
3750 145.	858 121.	449 123.	620 129.	927
3760 146.	242 121.	757 123.	948 130.	353
3770 146.	632 122.	087 124.	278 130.	608
3780 147.	020 122.	431 124.	612 131.	059
3790 147.	413 122.	732 124.	944 131.	298
3800 147.	795 123.	054 125.	272 131.	746
3810 148.	190 123.	388 125.	599 132.	059
3820 148.	577 123.	721 125.	921 132.	374
3830 148.	971 124.	027 126.	250 132.	734
3840 149.	348 124.	357 126.	583 133.	077
3850 149.	745 124.	689 126.	919 133.	432
3860 150.	130 125.	007 127.	240 133.	788
3870 150.	525 125.	343 127.	578 134.	109
3880 150.	914 125.	661 127.	906 134.	434
3890 151.	299 125.	989 128.	243 134.	862
3900 151.	687 126.	319 128.	565 135.	107

No. of instructions	Absolute	Immediat	e Zero Pag	e Mixed
3910 152.	079 126.	623 128.	905 135.	561
3920 152.	463 126.	945 129.	220 135.	805
3930 152.	856 127.	266 129.	553 136.	226
3940 153.	240 127.	617 129.	877 136.	577
3950 153.	635 127.	939 130.	220 136.	881
3960 154.	019 128.	251 130.	545 137.	235
3970 154.	420 128.	571 130.	875 137.	588
3980 154.	805 128.	896 131.	204 137.	942
3990 155.	183 129.	224 131.	539 138.	292
4000 155.	572 129.	531 131.	856 138.	570
4010 155.	967 129.	850 132.	188 138.	994
4020 156.	360 130.	194 132.	516 139.	369
4030 156.	746 130.	506 132.	852 139.	601
4040 157.	130 130.	830 133.	181 140.	062
4050 157.	525 131.	164 133.	517 140.	344
4060 157.	909 131.	480 133.	830 140.	672
4070 158.	300 131.	795 134.	171 141.	079
4080 158.	689 132.	138 134.	505 141.	441
4090 159.	074 132.	459 134.	822 141.	759
4100 159.	464 132.	769 135.	154 142.	085
4110 159.	848 133.	100 135.	488 142.	413
4120 160.	240 133.	438 135.	818 142.	754
4130 160.	634 133.	750 136.	151 143.	165
4140 161.	027 134.	095 136.	490 143.	522
4150 161.	413 134.	414 136.	805 143.	760
4160 161.	802 134.	729 137.	134 144.	226
4170 162.	189 135.	045 137.	461 144.	469
4180 162.	580 135.	372 137.	798 144.	900
4190 162.	964 135.	693 138.	128 145.	229
4200 163.	357 136.	012 138.	460 145.	546
4210 163.	746 136.	338 138.	783 145.	886
4220 164.	129 136.	683 139.	112 146.	248
4230 164.	518 136.	977 139.	438 146.	610
4240 164.	912 137.	302 139.	764 146.	968
4250 165.	295 137.	635 140.	097 147.	308
4260 165.	686 137.	959 140.	434 147.	599
4270 166.	070 138.	281 140.	767 147.	988
4280 166.	471 138.	609 141.	095 148.	385
4290 166.	864 138.	928 141.	418 148.	621
4300 167.	248 139.	255 141.	755 149.	059
4310 167.	632 139.	584 142.	079 149.	366
4320 168.	018 139.	899 142.	395 149.	683
4330 168.	413 140.	226 142.	741 150.	038
4340 168.	804 140.	572 143.	070 150.	434
4350 169.	187 140.	873 143.	404 150.	755
4360 169.	581 141.	211 143.	726 151.	093
4370 169.	964 141.	529 144.	064 151.	466
4380 170.	356 141.	835 144.	387 151.	831
4390 170.	741 142.	175 144.	718 152.	176
4400 171.	131 142.	487 145.	052 152.	416

No. of instructions	Absolute	Immediat	e Zero Pag	e Mixed
4410 171.	519 142.	816 145.	376 152.	904
4420 171.	912 143.	155 145.	713 153.	119
4430 172.	301 143.	475 146.	037 153.	599
4440 172.	693 143.	807 146.	368 153.	842
4450 173.	077 144.	110 146.	697 154.	200
4460 173.	468 144.	434 147.	022 154.	592
4470 173.	854 144.	770 147.	355 154.	938
4480 174.	239 145.	076 147.	683 155.	259
4490 174.	633 145.	405 148.	018 155.	596
4500 175.	024 145.	734 148.	342 155.	949
4510 175.	410 146.	051 148.	676 156.	313
4520 175.	804 146.	373 149.	006 156.	678
4530 176.	194 146.	707 149.	328 156.	919
4540 176.	578 147.	026 149.	660 157.	398
4550 176.	967 147.	349 149.	983 157.	628
4560 177.	352 147.	688 150.	323 158.	113
4570 177.	739 148.	010 150.	653 158.	359
4580 178.	133 148.	313 150.	984 158.	698
4590 178.	520 148.	657 151.	305 159.	065
4600 178.	905 148.	970 151.	642 159.	434
4610 179.	302 149.	301 151.	970 159.	767
4620 179.	692 149.	604 152.	299 160.	092
4630 180.	074 149.	954 152.	627 160.	497
4640 180.	466 150.	234 152.	963 160.	838
4650 180.	853 150.	578 153.	291 161.	159
4660 181.	248 150.	915 153.	619 161.	467
4670 181.	630 151.	241 153.	946 161.	897
4680 182.	018 151.	559 154.	274 162.	152
4690 182.	418 151.	898 154.	614 162.	539
4700 182.	800 152.	194 154.	942 162.	934
4710 183.	191 152.	531 155.	267 163.	218
4720 183.	580 152.	867 155.	596 163.	555
4730 183.	963 153.	176 155.	926 163.	916
4740 184.	356 153.	504 156.	259 164.	281
4750 184.	742 153.	825 156.	587 164.	639
4760 185.	134 154.	156 156.	921 164.	975
4770 185.	522 154.	477 157.	246 165.	293
4780 185.	911 154.	804 157.	574 165.	628
4790 186.	300 155.	120 157.	907 166.	054
4800 186.	691 155.	444 158.	230 166.	293
4810 187.	078 155.	775 158.	563 166.	771
4820 187.	464 156.	100 158.	895 167.	010
4830 187.	857 156.	425 159.	224 167.	353
4840 188.	249 156.	745 159.	559 167.	773
4850 188.	632 157.	062 159.	878 168.	114
4860 189.	029 157.	389 160.	205 168.	443
4870 189.	413 157.	718 160.	544 168.	765
4880 189.	798 158.	032 160.	868 169.	095
4890 190.	190 158.	357 161.	209 169.	453
4900 190.	580 158.	688 161.	534 169.	863

No. of instructions	Absolute	Immediat	e Zero Pag	e Mixed
4910 190.	976 159.	007 161.	862 170.	174
4920 191.	357 159.	337 162.	182 170.	469
4930 191.	740 159.	660 162.	506 170.	919
4940 192.	138 159.	985 162.	848 171.	158
4950 192.	526 160.	299 163.	180 171.	528
4960 192.	915 160.	636 163.	507 171.	957
4970 193.	298 160.	975 163.	833 172.	223
4980 193.	695 161.	283 164.	170 172.	575
4990 194.	080 161.	611 164.	495 172.	932
5000 194.	465 161.	926 164.	828 173.	285

Table 9: Measured time to run 1000 instructions with increasing frequency of the interrupt service routine (ISR) to run, measured in ms

No. of timeouts per second	Absolute	Immediate	Zero Page	Mixed
100.04 38.475	31.463	31.924	33.713	
101.07 38.465	31.451	31.932	33.728	
102.12 38.469	31.449	31.944	33.795	
103.20 38.468	31.468	31.937	33.770	
104.30 38.471	31.434	31.926	33.707	
105.42 38.465	31.452	31.942	33.720	
106.56 38.465	31.458	31.937	33.724	
107.74 38.472	31.447	31.934	33.783	
108.93 38.469	31.453	31.929	33.796	
110.16 38.468	31.460	31.934	33.715	
111.41 38.473	31.462	31.941	33.735	
112.69 38.478	31.470	31.928	33.717	
114.00 38.463	31.464	31.925	33.764	
115.34 38.464	31.455	31.929	33.802	
116.71 38.469	31.450	31.936	33.714	
118.12 38.467	31.447	31.936	33.720	
119.56 38.476	31.441	31.932	33.732	
121.04 38.480	31.457	31.941	33.709	
122.55 38.478	31.443	31.943	33.830	
124.10 38.479	31.467	31.933	33.754	
125.69 38.475	31.455	31.935	33.703	
127.32 38.471	31.463	31.938	33.737	
129.00 38.475	31.462	31.945	33.747	
130.72 38.479	31.456	31.933	33.805	
132.49 38.473	31.469	31.934	33.768	
134.30 38.478	31.455	31.939	33.730	
136.17 38.477	31.457	31.935	33.729	
138.08 38.485	31.459	31.934	33.735	
140.06 38.475	31.450	31.931	33.792	
142.09 38.469	31.455	31.935	33.831	
144.18 38.484	31.464	31.940	33.735	
146.33 38.468	31.467	31.939	33.742	
148.54 38.470	31.465	31.937	33.719	

No. of timeouts per second	Absolute	Immediate	Zero Page	Mixed
150.83 38.475	31.464	31.937	33.739	
153.19 38.473	31.452	31.940	33.806	
155.62 38.471	31.463	31.940	33.737	
158.13 38.475	31.458	31.948	33.709	
160.72 38.479	31.453	31.937	33.695	
163.40 38.481	31.469	31.938	33.731	
166.17 38.476	31.457	31.942	33.829	
169.03 38.482	31.470	31.949	33.764	
172.00 38.476	31.460	31.949	33.715	
175.07 38.486	31.464	31.945	33.724	
178.25 38.482	31.462	31.943	33.710	
181.55 38.486	31.480	31.942	33.787	
184.98 38.491	31.462	31.941	33.783	
188.54 38.489	31.464	31.944	33.741	
192.23 38.483	31.483	31.951	33.720	
196.08 38.495	31.473	31.955	33.702	
200.08 38.485	31.476	31.937	33.773	
204.25 38.489	31.479	31.946	33.819	
208.59 38.488	31.476	31.945	33.763	
213.13 38.486	31.470	31.951	33.740	
217.86 38.479	31.472	31.933	33.729	
222.82 38.490	31.478	31.952	33.730	
228.00 38.491	31.469	31.942	33.820	
233.43 38.494	31.464	31.951	33.795	
239.12 38.500	31.468	31.958	33.742	
245.10 38.500	31.484	31.953	33.730	
251.38 38.493	31.471	31.954	33.743	
258.00 38.495	31.488	31.952	33.791	
264.97 38.498	31.470	31.955	33.815	
272.33 38.500	31.475	31.954	33.748	
280.11 38.508	31.493	31.955	33.721	
288.35 38.505	31.465	31.961	33.747	
297.09 38.506	31.483	31.958	33.750	
306.37 38.510	31.497	31.953	33.824	
316.26 38.508	31.493	31.955	33.782	
326.80 38.503	31.510	31.955	33.752	
338.07 38.515	31.502	31.961	33.733	
350.14 38.515	31.496	31.960	33.743	
363.11 38.514	31.484	31.979	33.840	
377.07 38.528	31.493	31.969	33.826	
392.16 38.522	31.492	31.963	33.742	
408.50 38.527	31.495	31.971	33.746	
426.26 38.531	31.514	31.970	33.766	
445.63 38.518	31.490	31.979	33.811	
466.85 38.540	31.520	31.978	33.836	
490.20 38.544	31.506	31.979	33.792	
516.00 38.542	31.514	31.993	33.776	
544.66 38.557	31.536	31.979	33.762	
576.70 38.563	31.538	32.001	33.811	
612.75 38.565	31.518	31.999	33.868	

No. of timeouts per second	Absolute	Immediate	Zero Page	Mixed
653.59 38.569	31.531	32.005	33.831	
700.28 38.582	31.548	32.020	33.790	
754.15 38.600	31.522	32.013	33.805	
816.99 38.612	31.504	32.037	33.782	
891.27 38.613	31.561	32.041	33.874	
980.39 38.641	31.608	32.050	33.929	
1089.32 38.646	31.603	32.075	33.849	
1225.49 38.676	31.618	32.085	33.883	
1400.56 38.716	31.644	32.110	33.913	
1633.99 38.755	31.680	32.129	33.986	
1960.78 38.810	31.761	32.192	34.069	
2450.98 38.906	31.874	32.260	34.047	
3267.97 39.067	31.912	32.386	34.162	
4901.96 39.401	32.180	32.592	34.312	

[1] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy information systems: Issues and directions," *IEEE Software*, vol. 16, no. 5, pp. 103–111, 1999.

[2] H. M. Sneed, *Annals of Software Engineering*, vol. 9, no. 1/4, pp. 293–313, 2000.

[3] "Jaffe-/nesizer2." https://github.com/Jaffe-/NESizer2; GitHub.

[4] Comella-Dorda, Wallnau, Seacord, and Robert, "A survey of black-box modernization approaches for information systems," in *Proceedings international conference on software maintenance ICSM-94*, 2000.

[5] R. C. seacord, D. Plakosh, and G. A. Lewis, *Modernizing legacy systems: Software technologies, engineering process and business practices*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[6] M. Dowson, "The ariane 5 software failure," *SIGSOFT Softw. Eng. Notes*, vol. 22, no. 2, pp. 84, Mar. 1997.

[7] "N° 33–1996: Ariane 501 - presentation of inquiry board report." https://www.esa.int/For_Media/Press_Releases/Ariane_501_-_Presentation_of_Inquiry_Board_report.

[8] "Poison pcs and toxic tvs, californias biggest environmental crisis that you've never heard of." http://svtc.org/wp-content/uploads/ppc-ttv1.pdf.

[9] "Examensarbete t5 läkarprogrammet: Göra litteraturstudie." http://libguides.lub.lu.se/c.php?g=297461&p=4112444.

[10] D. Ary, L. C. Jacobs, A. Razavieh, and C. K. Sorensen, *Introduction to research in education*. Wadsworth Publishing, 2009.

[11] A. Håkansson, "Portal of research methods and methodologies for research projects and degree projects," in *Proceedings of the international conference on frontiers in education : Computer science and computer engineering fecs'13*, 2013, pp. 67–73.

[12] Merriam-Webster.com, "Legacy." https://www.merriam-webster.com/dictionary/legacy; Merriam Webster.

[13] "Little sound dj." https://www.littlesounddj.com/.

[14] "IEEE chip hall of fame: MOS technology 6502 microprocessor." https://spectrum.ieee.org/tech-history/silicon-revolution/chip-hall-of-fame-mos-technology-6502-microprocessor.

[15] I. MOS Technology, "MOS mcs6500 microcomputer family programming manual." MOS Technology, Inc., 1976.

[16] A. Corporation, "Atmega328/p." http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf; Microchip, 2016.

[17] J. L. Alberi, "A method of interprocessor communication for a multiprocessor environment," *Nuclear Science, IEEE Transactions on*, vol. 29, pp. 84–86, Mar. 1982.

[18] Y. N. Patt, "Methodologies for experimental research in computer architecture and performance measurement," in *Twenty-third annual hawaii international conference on system sciences*, 1990, vol. 1, pp. 2–5 vol.1.

[19] Merriam-Webster.com, "Agile." https://www.merriam-webster.com/dictionary/agile; Merriam Webster.

[20] "Manifesto for agile software development." http://agilemanifesto.org/, 2001.

[21] M. Kropp and A. Meier, "Teaching agile software development at university level: Values, management, and craftsmanship," in *2013 26th international conference on software engineering education and training (csee t)*, 2013, pp. 179–188.

[22] K. Schwaber and J. Sutherland, http://https://www.scrumguides.org/, 2016.

[23] H. Kniberg, 2nd ed. C4Media, InfoQ.com, 2015.

[24] "Nesdev wiki." http://wiki.nesdev.com/w/index.php/Nesdev_Wiki.

[25] "How to quickly get a 6502 cpu running." http://lateblt.tripod.com/bit63.txt.

[26] "AVRDUDE." https://savannah.nongnu.org/projects/avrdude.

[27] "Avr-gcc." https://gcc.gnu.org/wiki/avr-gcc.

[28] "AVR libc." https://www.nongnu.org/avr-libc/user-manual/index.html.

[29] "USBasp." https://www.fischl.de/usbasp/.