SANTA CLARA UNIVERSITY

Department of Electrical and Computer Engineering

I HEREBY RECOMMEND THAT THE THESIS PREPARED
UNDER MY SUPERVISION BY

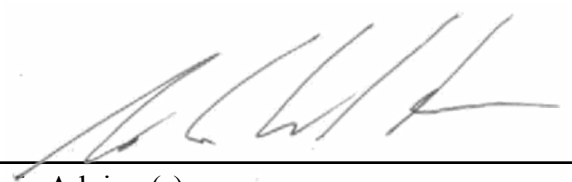
Sean Shao, Devin Hill

ENTITLED

Open Source Implementation of Cortex M0

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

BACHELOR OF SCIENCE
IN
ELECTRICAL AND COMPUTER ENGINEERING



6/14/23

Thesis Advisor(s) date

Shoba Krishnan

Jun 16, 2023

Department Chair(s) date

Open Source Implementation of Cortex M0

By

Sean Shao, Devin Hill

SENIOR DESIGN PROJECT REPORT

Submitted to
the Department of Electrical and Computer Engineering

of

SANTA CLARA UNIVERSITY

in Partial Fulfillment of the Requirements
for the degree of
Bachelor of Science in Electrical and Computer Engineering

Santa Clara, California

2023

Abstract

The Cortex-M series is an immensely popular family of processors optimized for low cost, size, and power. The smallest of the M-family, the Cortex-M0, uses a 3-stage pipelined design. Despite their popularity, however, no publicly available open-source designs exist for the Cortex-M series. Thus, for our senior design project, we chose to create an open-source implementation of the Cortex-M0 processor in Verilog. This enables users to learn more about, experiment on, and make changes to the design of the processor.

Though our goal for this project was to support all the functionality of the Cortex-M0 processor (as defined in the instruction set architecture ARMv6-M), we were unable to do so in the limited timeframe of a senior design project due to various challenges encountered, although most functionality is supported and the design is able to execute assembly programs. As an open-source design, however, others who wish to improve and expand upon our project can easily do so.

Acknowledgments

We would like to acknowledge our Senior Design advisor Dr. Wolfe for the guidance and support he has provided us over the course of this project.

We would also like to thank Santa Clara University's Electrical and Computer Engineering Department for providing us with the knowledge and tools to pursue this project

Table of Contents

Page

Page

List of Figures

Page

Chapter 1 — Introduction

1.1 Background

While Intel's x86 architecture may be more widely known amongst the general population for powering most desktops and laptops, the ARM instruction set architecture is the most produced and used microprocessor architecture, outselling all other microprocessor instruction set architectures combined [1]. As the world's reliance on digital electronics grows, ARM's usage continues to grow, even in fields where they were previously less dominant, such as for personal computers and data centers. A few years ago, Apple — which had already used ARM-based processors for their iPhones — announced they would transition to ARM-based processors for all their Mac computers [2], and Nvidia announced their development of an ARM-based server-class CPU [3].

Though they are experiencing significant growth in other fields, ARM's mostly widely used products are by far its Cortex-M family of microprocessors [1], which are designed for low cost, low size, and high efficiency. As a result, they are incredibly popular for microcontrollers and embedded systems applications [5].

Despite the immense popularity of the Cortex-M series, as far as we are able to tell, there are no publicly available open-source register-transfer level designs for any members of the M-family. ARM themselves do offer obfuscated designs for some processors in the Cortex-M series, but the nature of these designs makes it nearly impossible to experiment with changes to the design itself.

1.2 Motivation

Since no publicly available designs exist, one of our primary motivations for this project is to provide a tool that would enable users to learn about and experiment with the massively popular Cortex-M series. It also serves as an excellent way for us to better understand the ARM microprocessor architecture.

Of the M-family, the Cortex-M0 embodies the defining characteristics of the M-family the most, being the processor with the lowest cost, smallest size, and lowest power consumption, causing it to be one of ARM's most popular microcontrollers [5]. To enable these characteristics, the Cortex-M0 has a very limited instruction set, containing primarily basic data processing and control instructions, as illustrated by Figure 1. The limited instruction set is not an issue for our project, however, as we just intend to lay the foundations for a tool that can be expanded in the future as needed (e.g. to larger instruction sets). In fact, the smaller instruction set made the implementation process slightly simpler.

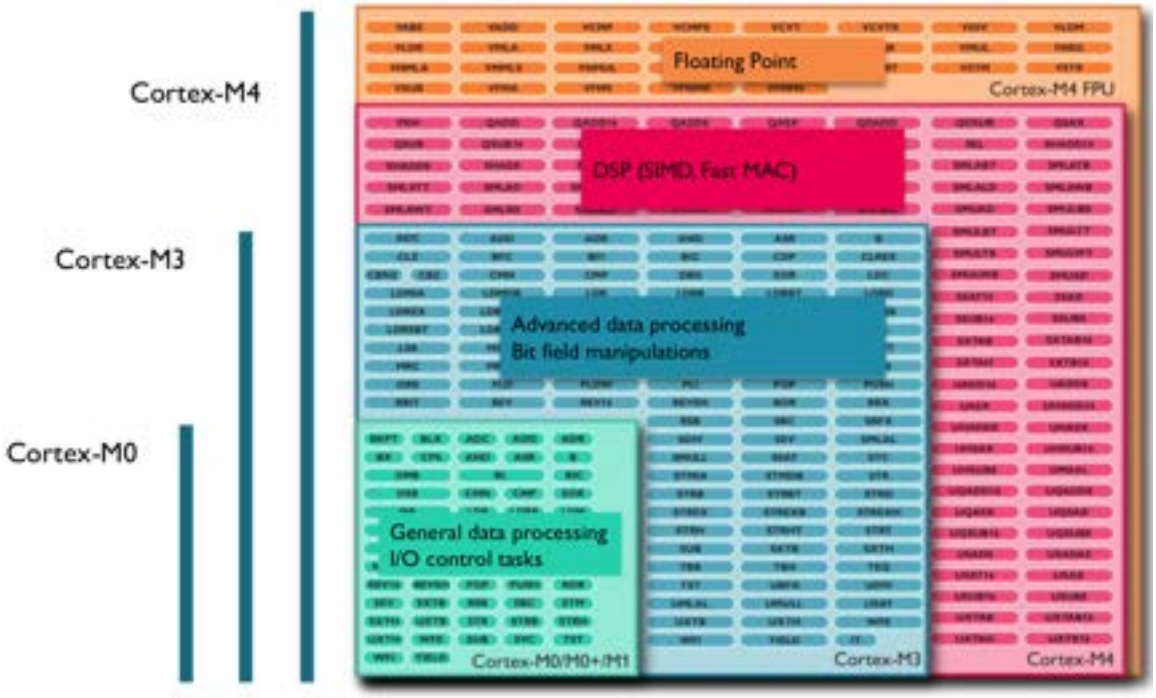


Figure 1: ARM Cortex-M instruction set architectures [6]

1.3 Project Objective

Our objective for our senior design project is to create an open-source implementation of a pipelined processor in Verilog that enables users to experiment with and make changes to the design of the processor. Our end result will be a register-transfer level design only, which means our project does not involve any physical design such as creating a layout.

As a processor, our end result will need to be able to read, decode, and execute assembly instructions. For these assembly instructions, we chose to follow the instruction set architecture ARMv6-M, which is the instruction set architecture of the Cortex-M0, both because of its popularity and simplicity. In the future, it is possible that ourselves or a different group may wish to expand this project to support the instruction set architecture of later Cortex-M cores, such as the advanced data processing instructions found in the Cortex-M3 and later.

Though we are only making an register-transfer level design, not providing a physical design or layout, and thus cannot make promises of efficiency or speed, we are aiming to support the full instruction set of the Cortex-M0 processor with the same behavior for each instruction, allowing us to achieve a 100% functional match. Additionally, we seek to make it synthesizable on a field-programmable gate array for both testing and demonstration purposes.

Chapter 2 — Design and Implementation

2.1 Pipelined Implementation

Pipelining is a technique that attempts to utilize more of a processor at once by dividing the processor into stages. Each stage handles a certain part of executing an instruction. Doing so enables multiple instructions to execute in parallel, as once the first part of one instruction is executed, the section of the processor used to execute that part can begin executing the first part of the next instruction. In the example in Figure 2, in a 3-stage pipelined processor, although each instruction takes 3 clock cycles to finish executing, pipelining enables a throughput of 1 instruction per clock cycle.

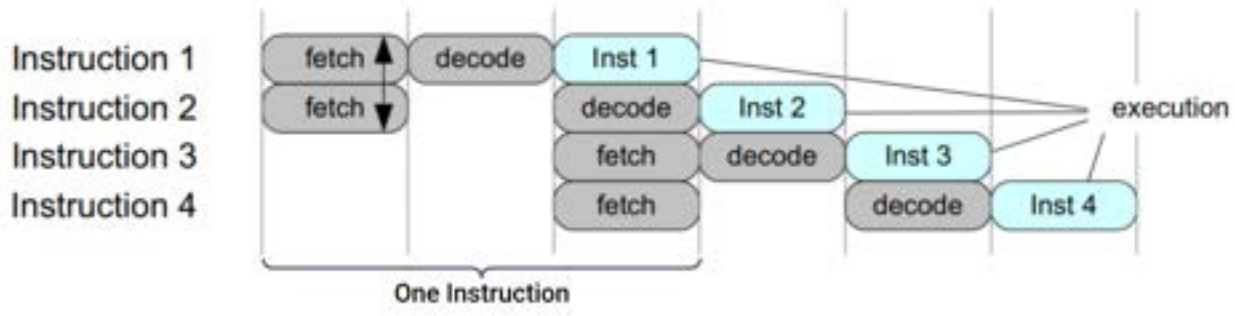


Figure 2: Execution of instructions in a 3-stage pipeline [7]

However, the longer the pipeline, stalling (e.g. when branching) is more costly as the entire pipeline must be flushed, and power consumption is higher.

The Cortex-M0 uses a 3-stage pipeline, as shown in Figure 3, due to the performance benefits it provides. The purpose of these stages is as follows:

Fetch stage: The instruction to be executed is read from instruction memory.

Decode stage: The instruction is interpreted then relevant registers are read from the register file.

Execute stage: Arithmetic operations are performed as needed for the instruction, and relevant data is written back to registers or memory.

A few other members of the Cortex-M series use non-3-stage pipelines, such as the Cortex-M0+, which only has 2 stages for improved power usage [8], Cortex-M7, which has a significantly

longer 6-stage pipeline because it is more optimized for performance than power consumption compared the rest of the M-family [9].

We elected to use a similar 3-stage pipeline for our design, both because of the performance benefits such a pipelined design offers, but also to stay as true to the Cortex-M0 as possible.

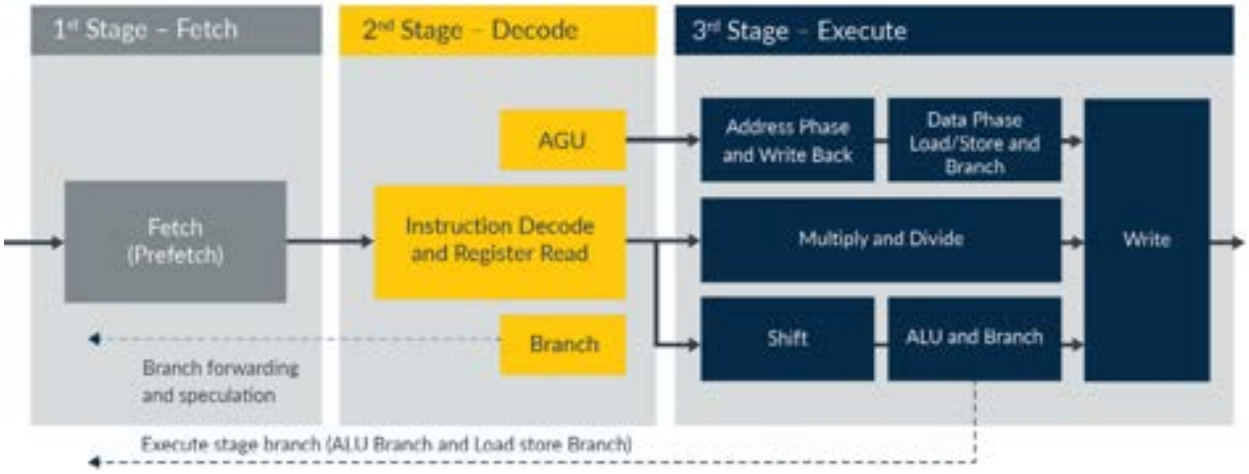


Figure 3: Block diagram of the Cortex-M0's pipeline [10]

In order to implement our pipelined design, we chose the hardware description language Verilog both because it is widely used and what we learned in our courses here at Santa Clara University. Each section of the pipeline is implemented as an individual Verilog module, which is a self-contained section of code that describes the intended behavior of that section of the digital system. These Verilog modules are then connected in a separate module to form the complete pipelined design. Figure 4 illustrates the primary modules in our design, as well as the main signals connecting them. The purpose of these modules is as follows:

Program Counter: Holds the address in memory of the next instruction to be executed.

Instruction Decoder: Decodes the instruction being executed and sends control signals to the rest of the pipeline accordingly.

Register File: Holds the registers, which are made up of 13 general purpose registers, the stack pointer, link register, and program status register.

ALU: Performs arithmetic and bitwise operations.

Pre-memory Logic: Based on the given address, determines which bits of a memory access should be written.

Memory: Stores the instructions and data used.

Branch Control: Determines if a branch should be taken, and if so, calculates the target address for the branch.

Stall Controller: Determines if the pipeline needs to be stalled, and if so, sets the relevant control signals in the pipeline.

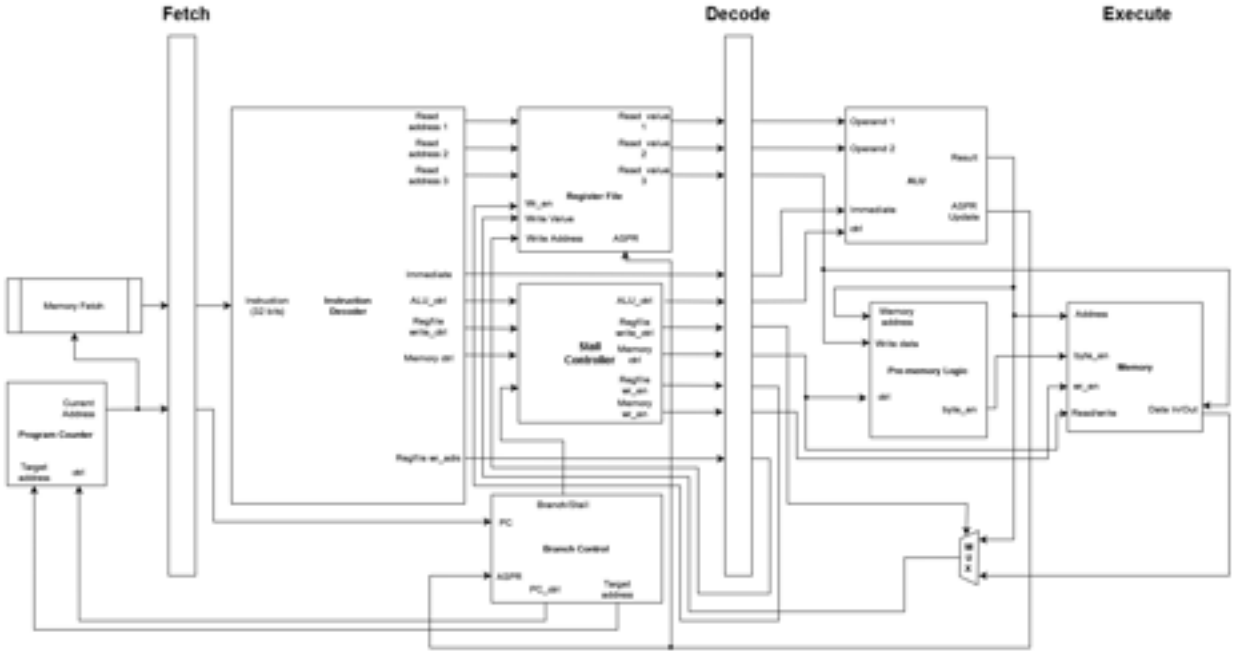


Figure 4: Block diagram of our 3-stage pipeline

2.2 Tools

The only tools we needed for this project, besides a working computer, were a development environment to write and simulate Verilog and a source control service to keep track of and sync code changes between the two of us.

For our development environment, we chose to use Xilinx Vivado, as we had experience using it in prior courses. Since it was difficult to use Vivado on our laptops, we ran Vivado on Santa Clara University's Engineering Computer Center. However, in the early stages of development, we sometimes used the website EDA Playground to test small code snippets, especially when we were unable to connect to the Engineering Computing Center to run Vivado.

For source control, we initially considered using GitHub. However, between the fact that we were writing our code remotely and we were unsure if we would run into issues using it with Vivado, we decided that we didn't need all the features GitHub provides and only needed a way to sync the code between the two of us. As a result, we simply used a shared Google Drive folder to hold our code files. At the beginning of the day, we could download the latest version of the code off of Google Drive, make changes as needed, then once finished reupload the files into the shared folder with a timestamp added to the file names. The older versions would be placed in an archive folder, in case we ever wanted to reverse changes. Since we would often both make changes to the same files on the same day, we also needed to manually merge our changes when this occurred, but since we usually divided up our work to be on different sections, the merging process was rarely much of an issue.

Chapter 3 — Testing and Results

3.1 Testing Process

A testbench is a Verilog module which sends a sequence of signals to a module under test and displays the behavior of that corresponding module. As we wrote the Verilog modules seen in Figure 4 that make up our design, we wrote a corresponding testbench module to ensure proper operation. Isolating tests to each module allows us to more quickly pinpoint the sources of errors.

To test the completed pipeline, we write a testbench module for the complete pipeline. This testbench loads the instruction memory with the assembly instruction of a program to be executed, then has the pipeline enable the right sequence of input signals to have the processor begin execution. Similarly to how we write testbenches for each module, we write individual programs to test all the core functionality of our design, so we can quickly determine if a certain feature is working. In the code for the testbench module, we use a flag to enable each functionality check as needed, so we can test a set of features at once rather than needing to rerun the simulation multiple times. These functionality checks include ensuring proper operation when reading from/branching to non-32-bit aligned instructions, branch operations take the correct number of clock cycles, and byte/half-word read/write memory accesses function as intended.

In order to test our device in a physical system, we can synthesize the completed pipelined design onto a field-programmable gate array, which is a circuit that can be reprogrammed to implement different logic functions. We can use this synthesized result to run programs as needed.

3.2 Results To Date

At the time of writing, almost the entirety of the pipeline has been implemented. All the modules from Figure 4 have been implemented, and the design is able to read and execute assembly programs. However, the decoder is not 100% complete — certain instructions that require special state machines, and system-level instructions, are not fully supported. Additionally, though it was

one of our testing goals, we did not have sufficient time to actually synthesize our design on an FPGA in order to run some basic tests.

Chapter 4 — Challenges and Constraints

While working on our project, we ran into a multitude of challenges, both because of the tools we were using and the design itself.

4.1 Challenges with the Tools

Because we were running Vivado remotely on the Engineering Computing Center, many tasks were more time consuming than they should have been. If we were off-campus, remoting in involved first connecting to Santa Clara University's VPN. If we were on-campus, remoting in often wouldn't work at all on our personal laptops. Because we were remoting into the Engineering Computer Center, we had to re-sign into our accounts every time, including our school emails when we needed to download or upload off the shared drive. On top of all this, when the internet was slow, as it often was when we were off-campus, doing anything through the remote desktop was difficult and slow.

Despite our past experience using it, we still ran into several issues using Vivado. Besides the issues stemming from having to connect to the Engineering Computer Center, it would often get stuck completely when performing certain operations. We eventually narrowed down the issue to the enhanced syntax highlighter, and the only workaround we found was disabling the enhanced syntax highlighter (which was enabled by default). However, even after switching to the "basic" syntax highlighter, the text editor would still often hang for seconds at a time. We tried turning off features such as the syntax highlighter completely and autocomplete, but this did not resolve it. We tried increasing the maximum number of threads Vivado would use from 2 (the default) to 8 (the maximum supported), but this also did not resolve it. Ultimately, the only workaround we were able to find was using an external text editor such as Notepad++ to write the code and only using Vivado to run the simulations. This meant we were unable to use many features such as autocomplete and syntax highlighting would be limited compared to Vivado. Yet another issue we ran into with Vivado is that, after relaunching the simulation several times, Vivado would crash without warning. Though we were unable to pin down a reason, we were luckily able to

find a workaround by entering the close and relaunch command using the TCL console rather than buttons in the GUI or keyboard shortcuts.

4.2 Challenges with the Design

As our design follows ARMv6-M, the instruction set architecture used by the Cortex-M0, memory accesses present unique challenges not faced by some other instruction set architectures. According to the specifications of ARMv6-M, memory accesses must be 32-bit aligned, yet ARMv6-M supports both 16 and 32-bit instructions, so extra logic is needed to support instruction fetches. For example, it is possible to have two 16-bit instructions in one 32-bit access (in Figure 2, you'll see that instructions 1 and 2 are fetched at once), branches to non-32-bit aligned addresses, or even a 32-bit instruction split across two memory locations because there is a 16-bit instruction in the first half of the first memory address. Moreover, though sometimes shown separately in block diagrams, instruction and data memory are part of the same memory block, the pipeline has to stall instruction fetch for every data memory access. Additionally, ARMv6-M supports reading and writing signed and unsigned bytes (8 bits), half-words (16 bits), and words (32 bits) to memory, even though memory accesses, as previously mentioned, must be 32-bit aligned. Thus, to support reading bytes and half-words, we need logic to mask the irrelevant bits of memory output, and select and extend only the bits we are interested in. Then, to support writing bytes and half-words, we need the memory controller to select which bits of memory are written, and ensure the memory module supports writing only to specific bits of a memory access to avoid overwriting the other 24 or 16 bits of the 32-bit memory access for an 8 or 16 bit write, respectively.

4.3 Constraints

One of the few constraints of our project was, since one of the goals for our project was to run and test our design on a FPGA, we had to ensure that the design of our processor is synthesizable on an FPGA. Certain Verilog constructs, such as delays, cannot be directly implemented on an FPGA, and thus are not synthesizable. We can still use those constructs for our testbenches, as

the testbenches are only used for testing purposes, not implemented on the FPGA, so this constraint applies only to the design of the processor itself.

Besides this, we just had to ensure that our code followed the definition of Verilog in IEEE Std 1364™-2005 [11], and our design attempted to remain as true to the Cortex-M0 as possible.

Chapter 5 — Reason for Open-Source Design

Why make our design open-source? There are two primary reasons.

5.1 Improving Design

Because of the limited timeframe of our project, testing of some sections of our design is incomplete, and some edge functionality is missing. By making our design open-source, others can help expand on our project by finding the edge cases where our design may not work as intended and fully rounding out the functionality of our design. Some features which are currently missing from our design that future groups may wish to work on include:

Full Decoder Support: Add the necessary logic to fully support all instructions in the ARMv6-M instruction set.

NVIC: Implement the Nested Vectored Interrupt Controller, which facilitates exception and interrupt handling.

Better Tests: Develop both more thorough and more complex test applications — make our design more reliable. Properly synthesize and use design on an FPGA.

Additionally, as seen back in Figure 1, the Cortex-M0 has the smallest instruction set and least functionality of the M-family. Making our design open-source opens the door for others to expand our project to other processors in the M-family with broader instruction sets.

5.2 Academic Utility

This project served as an excellent academic exercise for us, as it helped us refresh and expand upon our understanding of computer architecture. We hope that, by making our design open-source, our design can serve as a similar academic tool for others. Students can explore our design, not for research purposes, but just to better understand the components of a processor core. Similarly, teachers can utilize our design to supplement their teaching. For example, when we took ELEN 122 (Computer Architecture) here at Santa Clara University, our lab project was to design a basic pipelined CPU using the MIPS architecture, which is no longer supported. With our design, a lab could task the student with creating parts of a CPU using the ARM architecture,

which is massively popular in the embedded systems space and quickly growing in other fields as well.

Chapter 6 — Ethical and Professional Issues

In his 1921 Nobel Peace Prize acceptance speech, Christian Lous Lange commented, “Technology is a useful servant, but a dangerous master [12].” Over a century later, Lange’s remark on technological progress continues to ring true. It is all too easy to become enamored with the idea of creating new technological marvels without giving enough consideration to the impact they will have on the human beings it is meant to serve. We made sure to ensure that we do not make this blunder, and fully consider our project’s impact on the people and world around us.

6.1 Ethical Justification

In 1867, Alfred Nobel, a Swedish chemist and engineer, invented dynamite, a mixture of dynamite and an absorbent, as a tool for blasting tunnels, mines, and roads [13]. Prior to its invention, people were aware of the utility of nitroglycerin as an explosive, but it was far too volatile and thus unsafe to use. In fact, 3 years prior, Nobel’s factory, where he was manufacturing nitroglycerin, had exploded, killing several people including Nobel’s younger brother [13]. The invention of dynamite should have saved lives by being safer than its alternatives, but it was so effective that dynamite went from tool for mining to instrument of war. According to a popular story, Alfred Nobel was dismayed that his invention had been corrupted, and as a result decided to establish the Nobel Prizes in order to redeem himself [14]. Rather than simply being awarded for the greatest scientific advancement, the Nobel Prizes, in accordance to Alfred Nobel’s will, are given to those who “have conferred the greatest benefit to humankind” [15], which according to utilitarianism (and several other ethical theories) is the mark of ethicality.

One might argue that there is no distinction between the greatest scientific advancement and that which has “conferred the greatest benefit to humankind” because the “greatness” of scientific advancements is based on how they are able to benefit humanity. However, the way in which scientific advancements benefit humanity may vary. Some advancements, such as the invention of Penicillin, have obvious benefits that are immediately relevant to the general public, but the majority of technological development and research has no obvious effect on the well-being of

the broader public, but rather “benefits humanity” by expanding the collective pool of human knowledge. Even if you don’t value the expansion of our knowledge, these “inapplicable” advancements are what enable the development of advancements that are “applicable” to the general public.

Similarly, the IEEE Code of Ethics calls on its members to “improve the understanding by individuals and society of…technologies” [16], not just because IEEE is an engineering organization that naturally prefers more people learn about and research technology, but also because it is the ethical choice. Our project, though very inapplicable to the general public and likely only used, or even seen, by our fellow electrical and computer engineers, is still ethically justified. By allowing users to learn more about and experiment with the design of the Cortex-M0, can help improve our understanding of this technology. The IEEE Code of Ethics also calls on its members to “maintain and improve our technical competence” [16]. Only by keeping our knowledge and skills up to date can we ensure that we are able and qualified to undertake future tasks which may have a more significant effect on the general public. Even if our project is not used for learning or experimentation purposes by others, it is still ethically justified as it serves to improve our personal ability.

6.2 Risks and Safety

Even if the underlying concept of our project is ethically justified, we still must be careful to ensure the ethical application of our design. In a vacuum, there is no risk to the general public, as we have no intention of creating a physical design or implementing our design into other projects. Unfortunately, because our project is available to the public, others may decide to incorporate our design into their own projects where they expect our design to perform as intended. Due to the limited timeframe of our project, we have not performed enough testing on our design to ensure that our design will always function as intended.

To ensure the safe use of our project, we must ensure that the risks of our project — that our design might not be unerring — are properly communicated to any potential users regardless of their technological knowledge. Of course, due to the nature of our project, who would have access to it, and who would have the technological fluency to use it, it would be safe to assume a

certain degree of prerequisite knowledge from any users of our product. Nevertheless, if someone decided to make something using our design, we must make it clear that the users of their project (the "public") must similarly be informed of the possible unreliability of their project, originating from our design.

To communicate these risks, we place warnings and disclaimers at the beginning of both the documentation and code of the design itself to ensure that a user only reading one or the other still properly understands the risks associated with our project not being fully polished.

6.3 Legal Considerations

Since our design is based on a trademarked product, and we intend to make it open-source rather than solely an academic project, we have been careful to try to avoid legal issues surrounding our project. As stated on the title page (page ii), we are not sponsored by, endorsed by, or affiliated with the owner of the trademark in any way.

The Semiconductor Chip Protection Act of 1984 established that "the predetermined, three-dimensional pattern of…material present or removed from the layers of a semiconductor chip product" – the chip's layout – is protected from reproduction, importing, and distribution [17]. However, our design is not based on the layout of the chip itself, but rather the functionality of the chip (as defined by the instruction set architecture). The Chip Protection Act also specifies that "in no case does protection…extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery" [17]. Even if the functional design of the chip were covered, reverse engineering is permitted by the Chip Protection Act. Analyzing the "circuitry, logic flow, or organization of components" is not an infringement, and the knowledge gained from that analysis can be "used to create an original chip having a different design layout, but which performs the same or equivalent function as the existing chip" [17]. Thus, even if someone were to create a layout based on our design, it would not be an infringement of the Chip Protection Act because it is based on our analysis of the "logic flow" of the ARM Cortex-M0, and would not have the exact same layout as the Cortex-M0 (though they would need to provide the evidence and paperwork that their layout is original and not based on the layout of the Cortex-M0 itself).

During our work on this project, we never referenced any information which required creating an account with ARM, including the obfuscated design of the ARM Cortex-M0. All the documents we referenced regarding the design and architecture of the Cortex-M0 are publically available on the ARM website. As far as we are able to tell, the “Proprietary Notice” and “Confidentiality Status” of the documents we referenced do not state anything regarding the use of them for the design of a processor compatible with the instruction set architecture of the Cortex-M0 (ARMv6-M). Though they suggest that sections may be protected by patents, they do not specify which sections of the document, if any, those patents cover. Additionally, we have not made, used, imported, or offered to sell a processor so regardless the likelihood of patent infringement is miniscule.

6.4 Civic Engagement

As far as we are aware, there are no regulatory agencies or professional societies from which our project requires approval, besides the requisite approval we needed for our project and thesis from Santa Clara University. If, in the future, we wish to publish our design in a more established publication or present our project at a conference, we would need to receive approval from that organization (likely IEEE), but at the moment we have no need to influence the approval of the general public due to the nature of the project.

6.5 Environmental Impact

Beyond the environmental impact of the computers used to design, code, and test our design, the only environmental impact of our project we were able to identify is the cost needed to publicly host our design due to its open-source nature. Of course, the cost of storing data on the “cloud,” even if monetarily free, is far from negligible. Though exact numbers vary depending on which study you consult, an article in Stanford Magazine estimates that hosting 100 gigabytes in the cloud results in a carbon footprint of roughly 0.2 tons [18]. For scale, the global average carbon footprint per person is 4 tons [19]. Our project will take far less than 100 gigabytes, as it simply consists of text files and some documentation, so our carbon footprint will be significantly less

than 0.2 tons, but it is still an important factor to consider. We believe the benefits of making our project open-source, as outlined in Chapter 5, make hosting our project on the internet worth the cost, but it is a difficult argument to make when the welfare of our planet's environment is in the balance.

Chapter 7 — Conclusion

7.1 Conclusion

Our Senior Design Project is an open-source synthesizable implementation of the Cortex-M0 processor in Verilog. Our design follows the M0's 3-stage architecture, divided into individual Verilog modules with testbenches for each module. Though we ran into several challenges along the way, and there are several avenues of improvement for future work, our project provides other students and researchers with a new tool to further study the ARM Cortex-M0's architecture. Additionally, thanks to the open-source nature of our project, improvements can be made and the design may be optimized.

References

[1] A. Shilov, “842 chips per second: 6.7 billion arm-based chips produced in Q4 2020,” Tom’s Hardware, 2021, tomshardware.com/news/arm-6-7-billion-chips-per-quarter.

[2] “Apple announces Mac transition to Apple silicon,” Apple Newsroom, 2020, apple.com/newsroom/2020/06/apple-announces-mac-transition-to-apple-silicon.

[3] “NVIDIA Announces CPU for Giant AI and High Performance Computing Workloads,” Nvidia Newsroom, 2021, nvidianews.nvidia.com/news/nvidia-announces-cpu-for-giant-ai-and-high-performance-computing-workloads.

[4] P. Zuo, “An overview of the Arm Cortex-M processor family,” Arm Community, 2016, community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/white-paper-cortex-m-for-beginners-an-overview-of-the-arm-cortex-m-processor-family-and-comparison.

[5] I. Cutress, “PlasticArm: Get Your Next CPU, Made Without Silicon,” AnandTech, 2021, anandtech.com/show/16837/plasticarm-get-your-next-cpu-without-silicon.

[6] A. L. Shimpi, “ARM's Cortex M: Even Smaller and Lower Power,” AnandTech, 2014, anandtech.com/show/8400/arms-cortex-m-even-smaller-and-lower-power-cpu-cores.

[7] “Application Note 321 ARM Cortex-M Programming Guide to Memory Barrier Instructions,” Arm Developer, 2012, developer.arm.com/documentation/dai0321/a.

[8] “ARM Cortex®-M0+ Pipeline,” Microchip Developer Help, microchipdeveloper.com/32arm:m0-pipeline.

[9] “Cortex-M7,” Arm Developer, Arm, developer.arm.com/Processors/Cortex-M7.

[10] “Arm Cortex-M0 Processor Datasheet,” Arm, arm.com/-/media/Arm%20Developer%20Community/PDF/Processor%20Datasheets/Arm_Cortex-M0_Processor_Datasheet.pdf.

[11] “IEEE Standard for Verilog Hardware Description Language,” IEEE Std 1364-2005, IEEE Xplore, 2006, ieeexplore.ieee.org/document/1620780.

[12] “Christian Lange,” The Nobel Prize, nobelprize.org/prizes/peace/1921/lange/lecture.

[13] “Alfred Nobel,” Encyclopaedia Britannica, 2023, britannica.com/biography/Alfred-Nobel.

[14] F. Golden, “The Worst And The Brightest,” TIME, content.time.com/time/subscriber/article/0,33009,998209,00.html.

[15] “Alfred Nobel’s will,” The Nobel Prize, nobelprize.org/alfred-nobel/alfred-nobels-will.

[16] “IEEE Code of Ethics,” IEEE, ieee.org/about/corporate/governance/p7-8.html.

[17] “Brooktree Corporation, Plaintiff/cross-appellant, v. Advanced Micro Devices, Inc.,” Justia, law.justia.com/cases/federal/appellate-courts/F2/977/1555/304802.

[18] J. Adamson, “Carbon and the Cloud,” Stanford Magazine, stanfordmag.org/contents/carbon-and-the-cloud.

[19] “Calculate Your Carbon Footprint,” The Nature Conservancy, nature.org/en-us/get-involved/how-to-help/carbon-footprint-calculator.