# Cellular Automata Accelerator in RISC-V Processor

**Lucie Sharpe**
School of Engineering
University of Warwick

Supervised by Eduardo Wachter

15 July 2025

WARWICK
THE UNIVERSITY OF WARWICK

**Abstract**

Cellular automata (CA) can be a useful tool for modelling physical systems. However, as many CA use two or more dimensional boards the number of calculations needed to find the next state can scale greatly. If these calculations are performed serially, a considerable amount of computation time may be needed. Individual cell calculations in a CA are independent which provides an opportunity for massive parallelism allowing for constant time computation. However, even with the recent rise in core count, processors are not able to fully exploit this potential. This project aims to create an adaptable hardware accelerator for CA that can be directly connected to a RISC-V processor to allow for faster computation of states. Due to the adaptable nature any CA could be implemented. In this project Conway's Game of Life and Elementary CA were tested.

An FPGA will be used to create the hardware design as they allow for significantly faster development time. The functionality and performance of the accelerator was tested and compared to a software implementation running on the RISC-V core resulting is a significant decrease in clock cycles required to compute the next state. This demonstrative the advantage of having specialised hardware accelerators.

**Keywords:** Cellular Automata, RISC-V, FPGA, AXI, Accelerator

# Contents

# Chapter 1

# Introduction

As new applications are found the performance required from processors increases. This increase used to be gained primarily by increasing the clock speed of the processor. From 1987 to 2003 processor clock speed increased on average by 40% per year, shown in Figure 1.1, which was achieved by reducing transistor size, leading to decreased propagation delays but has the side effect of increasing power density.

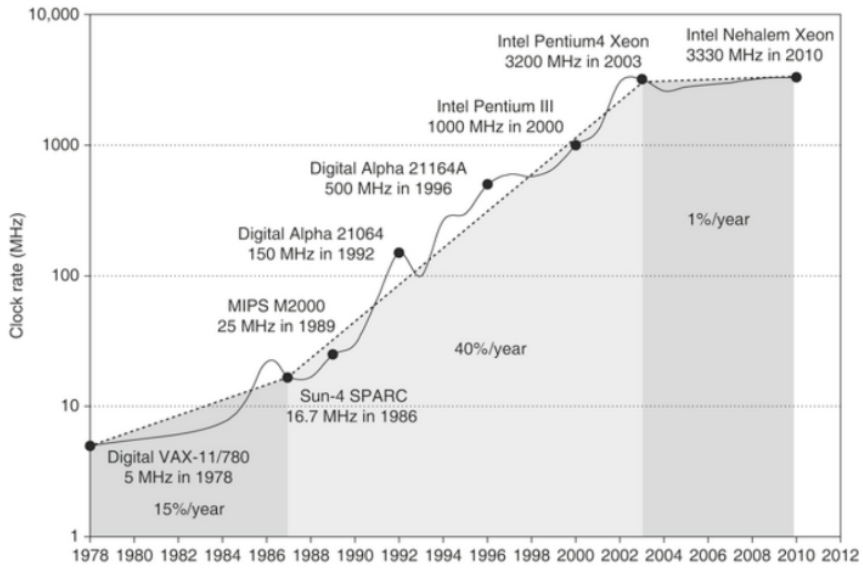


Figure 1.1: Graph showing trends in clock speed growth [9, Figure 1.11]

Issues are found when a limit is reached where the physical materials on the chip can not dissipate the heat faster than it is generated. From 2003 a drastic decrease in yearly clock speed gains showing the impact of this limit can be seen in Figure 1.2. Instead processor manufacturers started increasing core count to allow the use of parallelism for performance gain. General purpose parallelism is beneficial but has diminishing returns due to the ability of applications to harness multiple cores and the size requirements to add more cores. Another option is to use specificity designed hardware accelerators that can fully exploit the massive parallelism inherent in many different applications. Hardware accelerators also eliminate overhead such as the fetch-decode-execute cycle that is needed by a processor which can increase power efficiency.

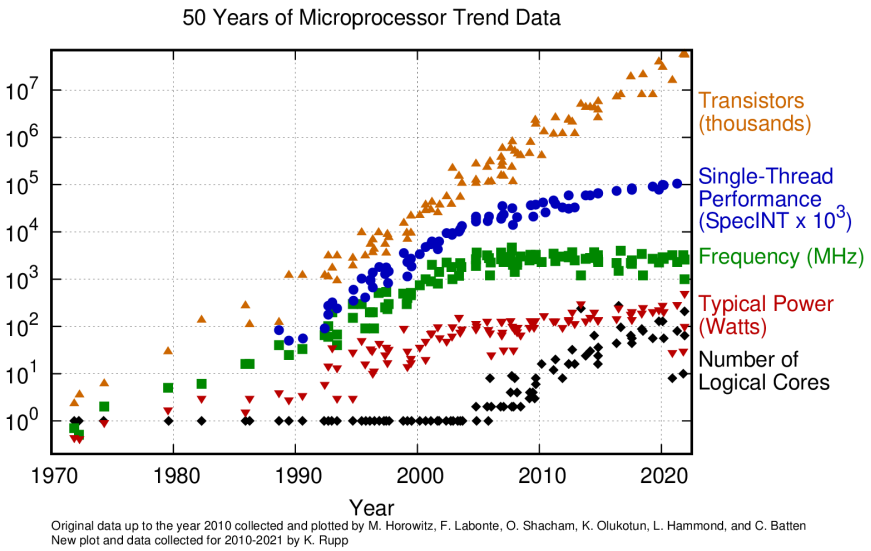


Figure 1.2: rends in processors transistor count, clock speed, power usage and core count [15]

One application for which hardware acceleration could be used is the execution of Cellular Automata (CA) which is a method of modelling complex phenomena based on simple rules. CA execution can utilize massive parallelisation as all calculations are independent, creating the possibility of constant time computation if sufficiently parallelised.

As these hardware accelerators are physical devices, development and iterations of designs can be slow and expensive if the hardware must be physically constructed each time. Field-Programmable Gate Arrays (FPGAs) provide a solution to this problem by providing a form of reconfigurable hardware that can be programmed to implement any design [20]. This project will use an FPGA chip to develop a CA hardware accelerator for a processor that can be used to evaluate the benefits of dedicated hardware.

RISC-V is an open standard instruction set architecture (ISA) that can be implemented when designing a processor [14]. Due to its open nature this ISA, has seen wide spread adoption in both high-performance products, such as SiFive boards, and in low-power microcontrollers such as Espressif's ESP32-C3. There is also an FPGA implementation [2] that can be modified and extended to add extra functionality. This provides an excellent foundation for this project to build off of as any developments would be just as applicable in the many real world application that utilize RISC-V.

## 1.1 Objectives

This project will develop a cellular automata hardware accelerator integrated into a RISC-V processor and evaluate it's performance when compared to other methods of execution. Below is a list of objectives for this project.

1. Generate and implement a RISC-V core on the FPGA that can run bare-metal code.
2. Develop hardware that can be integrated into the processor so data can be written to and read from the accelerator.
3. Develop hardware than can perform CA computations utilizing the inherent parallelisation of the application.
4. Implement a bare-metal driver for the hardware accelerator.
5. Evaluate the performance of the accelerator compared to computation through bare-metal code.

# Chapter 2

# Background

## 2.1 Cellular Automata

Cellular Automata (CA) are an array of cells capable of holding different states. The array can then be iterated by applying a rule set to each cell resulting in a new state being set. These rules take the cell's current state and its neighbour's states as inputs and then defines what the next state should be. CA were first used by von Neumann for his 29 state "universal constructor" in 1966 [12].

One of the simplest examples of CA are called elementary cellular automaton and were extensively studied by S. Wolfram [18]. These are one-dimensional, binary, nearest neighbour CA. A one-dimension array is used to hold the state of cells which can either be 1 or 0. The rules to update these cells use the two nearest neighbours to the cell and its current state to calculate the next state. This make 3 inputs to the rule giving a total eight possible input states. These input states can either be assigned to output 1 or 0 giving 256 possible forms of the elementary CA. Figure 2.1 shows an example rule set for this type of automaton, known as Rule 30. The diagram at the top shows every possible combination of the 3 inputs, centre cell and two nearest neighbours, and the output to that assigned pattern. This is known as Rule 30 due to the binary value made by the outputs. The grid below shows the effect this rule has when implemented on the row above. The top row at the top is the initial state and each row down is an iteration of the CA. These simple rules can lead to very

useful CA with Rule 30 being used for random number generation and Rule 184 for traffic modelling.

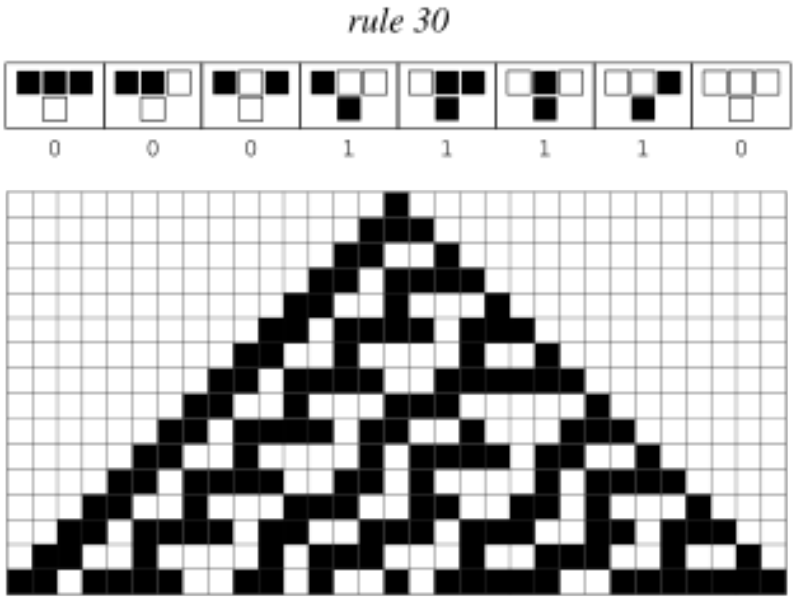


Figure 2.1: Rule 30 and its iterations [17]

Another example of CA is Conway's Game of Life (CGOL) [7] which uses a two-dimensional array of binary cell's but with a totalistic rule set meaning the average of neighbours is used. CGOL uses the Moore neighbourhood, square of closest neighbours to cell, with the following rules to determine the cells next state.

1. Set state to 0 if sum of Moore neighbours is less than 2 or greater than 3.
2. Set state to 1 if sum of Moore neighbours is 2 or 3 and current state is 1.
3. Set state to 1 if sum of Moore neighbours is 3 and current state is 0.

When calculating the next iteration of CA the rules are applied independently to each cell creating an inherently parallel process. This creates an excellent opportunity to use a hardware accelerator to exploit this parallelism allowing for any size CA to be computed in constant time if a sufficiently large accelerator is used.

### 2.1.1 Applications

CA have many real world applications such as modelling physical or chemical systems. One useful application is simulating forest fires [11]. This simulation can then be used to test how effective various fire fighting strategies are and plan responses for future forest fires. Another use case is traffic modelling. Rule 184 is a particularly useful basis for this as it models traffic jams in a one dimensional stream. The Biham, Middleton and Levine Model extends this to act in two dimensions and can then be applied to predict real time traffic jams [10].

## 2.2 FPGAs

A Field Programmable Gate Array (FPGA) is a semiconductor device based around a matrix of Configurable Logic Blocks (CLB), Input/Output Blocks and specialized blocks such as for Digital Signal Processing (DSP) [21]. Logic can take two forms, combinatorial and behavioural. Combinatorial logic takes in a set of new input and based on a set of rules determines the output. Behavioural logic takes in new input but also uses previous outputs to determine what the next output is. To achieve this, data must be stored for use in future computations. CLB use Lookup Tables (LUT) for combinatorial logic and Flip-Flops (FF) to store values for behavioural logic. A LUT is a small memory block where the inputs are used to address a bit of memory and the values stored there represents the output of the logic. Any truth table can be stored in these LUTs and if more inputs are required multiple LUTs can be combined. Figure 2.2 shows the structure of a CLB.

While CLB could achieve any task, they sacrifice efficiency for adaptability. So some very common and useful functions are given dedicated blocks such as the DSP which allows for operations such as addition and multiplication to be completed with greater efficiency. Blocks are connected together through configurable switches allowing signals to take any route through the blocks. This structure is show in Figure 2.3.

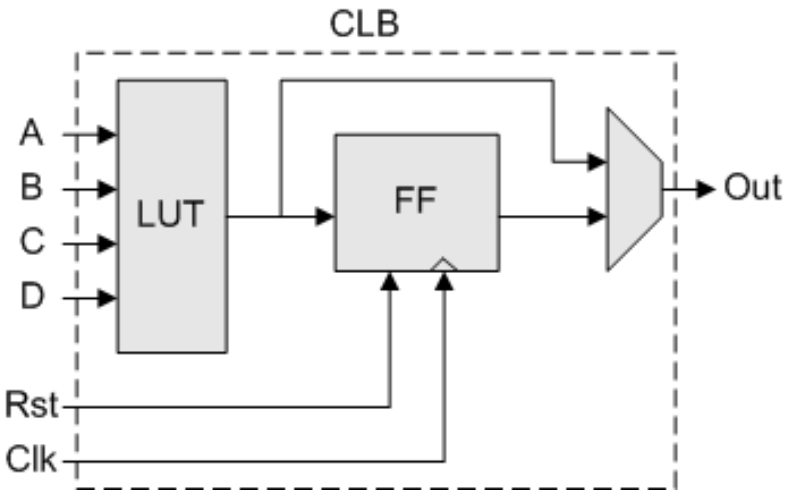


Figure 2.2: Block diagram of a CLB [6]

The ability to program the FPGA to act as any hardware makes it an excellent solution for rapid development of hardware as new designs can be tested and iterated on without incurring any additional costs. For this project a Xilinx Nexys A7 100T development board was used.

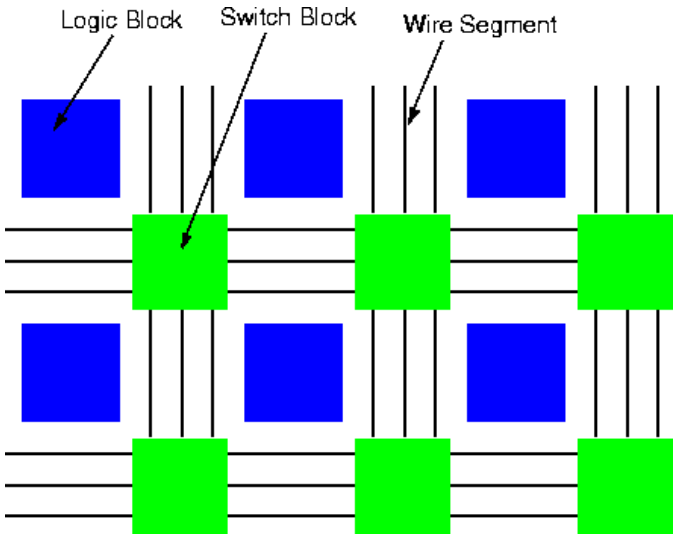


Figure 2.3: Structure of an FPGA block matrix with configurable routing [3]

### 2.2.1 Verilog

FPGAs are programmed using a Hardware Description Language (HDL) which describes the logic that is run. This project uses Verilog as it is a relatively low level HDL giving fine control over the implemented design. Verilog code can then be synthesised into a form that optimises the logic for use on FPGAs in general. It can then be implemented for the specific FPGA used and mapped onto the cells inside the FPGA. As a Xilinx FPGA is used in this project the Vivado Design Suite 2023.2 was used to perform these steps.

## 2.3 RISC-V

As an open standard ISA, RISC-V provides a royalty free base that can be used when creating custom solutions which require a processor. This guarantees the design will function with the plethora of publicly available software and tools designed for RISC-V processors reducing the work required to get a fully functional product. These tools include a RISC-V GNU Compiler Toolchain with GCC for compiling C code [4] and a port of Debian [5]. The RISC-V GNU Compiler Toolchain will be particularly useful for this project as it will be used to compile bare-metal code. RISC-V's open nature and compatibility has led to wide spread adoption with an estimated shipments of RISC-V based System-on-chip (SoC) at 16.2B units in 2030, with revenues reaching $92B [8].

### 2.3.1 Rocket Chip

Rocket Chip is an open-source SoC generator that uses the RISC-V ISA [2]. This tool can be used to configure and generate a RISC-V SoC into synthesizable RTL for use on FPGAs. A second project called vivado-risc-v uses Rocket Chip to generate a set of cores that are directly compatible with the Nexys A7-100T FPGA board used in this project [16]. It also provides scripts and documentation to setup and run the Linux kernel and compile bare-metal code. In its current state the bootable Linux environment this project creates has many issues, the biggest being exceedingly long boot times. However, it should

be possible to add the custom hardware created in this project as a preferential in the device tree allowing for its use within Linux.

### 2.3.2 AXI

The Advanced eXtensible Interface (AXI) is a royalty-free communication bus protocol that is part of the ARM Advanced Microcontroler Bus Architectrue (AMBA) standard [1]. The AXI protocol has been adopted by Xilinx for use with FPGA IP block designs [19] and is common in many other SoCs. There are 3 interfaces: AXI4, AXI4-Lite and AXI4-Stream. AXI4 provides high-performance memory-mapped communication and AXI4-Stream allows for high-speed streaming of data. For this project, AXI4-Lite will be used as it is a simpler interface for low-throughput memory-mapped communication. AXI4-Lite uses five channels. Two for read transactions consisting of address and data and three for write transaction which uses an additional write response channel [13]. The AXI4 specification uses a VALID/READY handshake to allow both the master and slave to control the transfer rate. The VALID signal is set when the address or data is ready to be sent and the READY signal is set when the destination is ready to accept the data. The transaction completes when both signals are set during a rising clock edge. Figures 2.4 and 2.5 shows the activity of the signals used in AXI read and write transactions. These show the handshake being performed and data getting transferred.

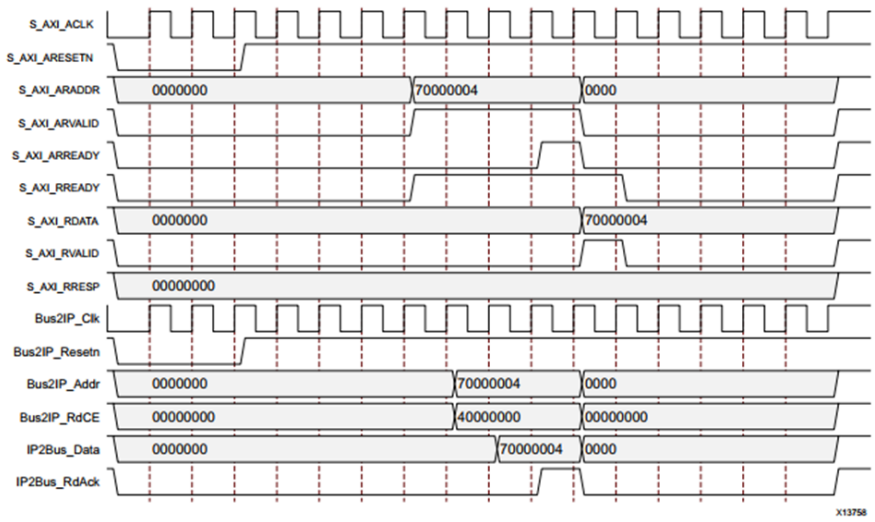


Figure 2.4: Signals used in AXI read transactions [13]

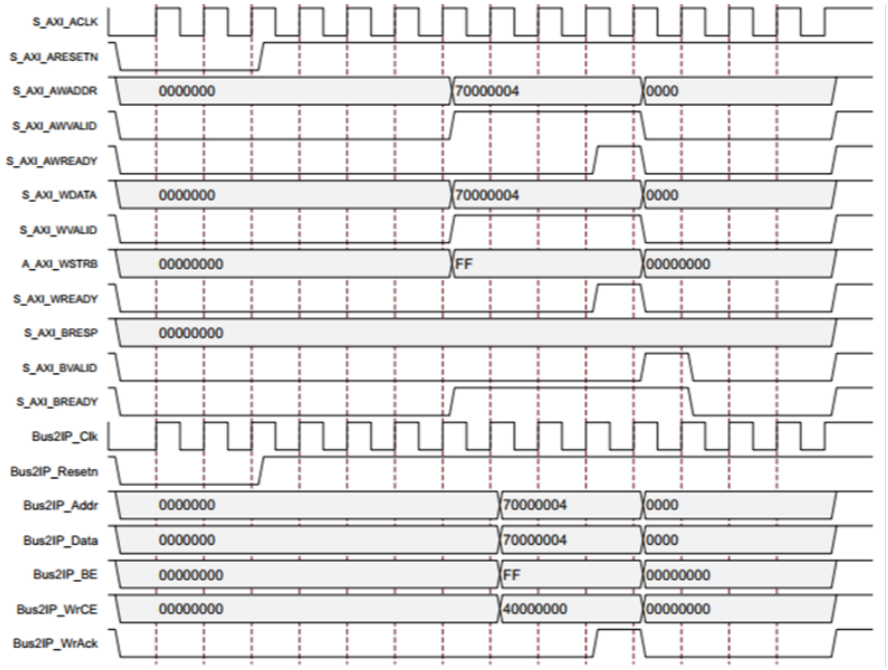


Figure 2.5: Signals used in AXI write transactions [13]

# Chapter 3

# Related Works

## 3.1 Implementation of a RISC-V Processor with Hardware Accelerator

This paper covers the development of a hardware matrix multiplier for a RISC-V processor. A ZedBoard development board, using the same Artix-7 FPGA chip as the Nexys A7-100T, is used. The RISC-V core used in this project is generated by PULPino which creates a 32-bit, single-core microprocessor [**?** ] This is a simpler core and requires less FPGA resources to implement, freeing more space for custom hardware. Our project uses a 64-bit core generated with Rocket Chip which reduces the available resources. However, due to the configurability of Rocket Chip the core can be changed to a smaller 32-bit core if needed.They used the APB bus to communicate between the accelerator and PULPino microprocessor due to its interface being simpler than the AXI bus. In our project, the AXI interface will be used as it provides the option for higher-performance transfers and is more often used when connecting internal hardware. When tested their accelerator reduced the number of clock cycles required from 603 to 134. This is a good improvement but due to the processor having a clock speed of 5 MHz, it is unable to compete with modern processors. They encountered many errors when trying to build C programs for the PULPino and required much trial and error. The process of compiling code in our project has proved to be far simpler.

## 3.2 Design and Implementation of Cellular Automata on FPGA for Hardware Acceleration

This project implements an accelerator for Conway's Game Of Life. They use a DE1-SoC development board with a Cyclone V FPGA chip. Their design uses a lattice of cell modules that read the neighbours states from a register file and write their next state back to it. Our design is similar in its use of a board register but to provide more adaptability all cells states are sent to each cell rather than only the 8 neighbours. This allows the cell to use any neighbourhood. They also include the load state function in the cell module using a multiplexer in the same way as our design. They integrate a mouse and VGA controller into their design to allow for control and result output. Another feature implemented is a save manager that stores previous board states. A similar feature to this could be added to our design to allow intermediate states to be pulled as required while the accelerator continues to compute removing the need to wait for each one. Their design achieved a speed boost of 36.7x against a GPU implementation and 2908x against a software implementation. This shows the benefit of accelerators such as even when compared to modern GPU parallelization.

## 3.3 Cellular Automata Hardware Implementations - an Overview

This paper explains the requirements for general cellular automata hardware that should be able to implement any local rule. It then reviews the methods this hardware could be implemented. FPGAs can be reconfigured to fit a particular CA from a hardware description. A programmable cellular automata allows the same hardware to be used for any local rule. This form of general cellular automata has been implemented on FPGAs by the University of Porto (Portugal) [**?** ] Cellular automata machines are a different approach which use pipelined serial processing. Finally it mentions the uses of systolic arrays for parallel processing of CA.

# Chapter 4

# Design

In this chapter, we describe the overall design of our solution to the problem identified in Chapter 1, building on work described in Chapter 2.

# Chapter 5

# Implementation

In this chapter, we describe the implementation of the design we described in Chapter 4. You should **not** describe every line of code in your implementation. Instead, you should focus on the interesting aspects of the implementation: that is, the most challenging parts that would not be obvious to an average Computer Scientist. Include diagrams, short code snippets, etc. for illustration.

# Chapter 6

# Evaluation

Describe the approaches you have used to evaluate that the solution you have designed in Chapter 4 and executed in Chapter 5 actually solves the problem identified in Chapter 1.

While you can discuss unit testing etc. you have carried here a little bit, that is the minimum. You should present data here and discuss that. This might include *e.g.* performance data you have obtained from benchmarks, survey results, or application telemetry / analytics. Tables and graphs displaying this data are good.

# Chapter 7

# Conclusions

The project is a success. Summarise what you have done and accomplished.

## 7.1 Future work

Suggest what projects might follow up on this. The suggestions here should **not** be small improvements to what you have done, but more substantial work that can now be done thanks to the work you have done or research questions that have resulted from your work.

# Bibliography

[1] ARM, . *AMBA AXI Protocol Specification*. URL https://developer.arm.com/documentation/ihi0022/latest/. (Accessed April 2023).

[2] Asanović, Krste & Avizienis, Rimas & Bachrach, Jonathan & Beamer, Scott & Biancolin, David & Celio, Christopher & Cook, Henry & Dabbelt, Daniel & Hauser, John & Izraelevitz, Adam & Karandikar, Sagar & Keller, Ben & Kim, Donggyu & Koenig, John & Lee, Yunsup & Love, Eric & Maas, Martin & Magyar, Albert & Mao, Howard & Moreto, Miquel & Ou, Albert & Patterson, David A. & Richards, Brian & Schmidt, Colin & Twigg, Stephen & Vo, Huy & Waterman, Andrew. The rocket chip generator. Technical Report UCB/EECS-2016-17, Apr 2016. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html.

[3] Betz, Vaughn. Fpga architecture for the challenge. URL https://www.eecg.utoronto.ca/~vaughn/challenge/fpga_arch.html. (Accessed April 2024).

[4] Collaboration, RISC-V. riscv-gnu-toolchain. URL https://github.com/riscv-collab/riscv-gnu-toolchain. (Accessed April 2023).

[5] Debian, . Debian riscv64 port. URL https://wiki.debian.org/Ports/riscv64. (Accessed April 2023).

[6] FPGAKey, . Clb. URL https://www.fpgakey.com/wiki/details/51. (Accessed April 2024).

[7] Gardner, M. Mathematical games: The fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 1970. URL http://dx.doi.org/10.1038/scientificamerican1070-120.

[8] Group, SHD. Risc-v market report. URL https://theshdgroup.com/market-reports/. (Accessed April 2023).

[9] Hennessy, John L. *Computer architecture: a quantitative approach*. Morgan Kaufmann/Elsevier, 5th edition, 2012.

[10] Hu, Wenbin & Yan, Liping & Wang, Huan & Du, Bo & Tao, Dacheng. Real-time traffic jams prediction inspired by biham, middleton and levine (bml) model. *Information Sciences*, 381:209–228, 2017. ISSN 0020-0255. doi: 10.1016/j.ins.2016.11.023. URL https://www.sciencedirect.com/science/article/pii/S0020025516318308.

[11] Mutthulakshmi, K. & Wee, Megan Rui En & Wong, Yew Chong Kester & Lai, Joel Weijia & Koh, Jin Ming & Acharya, U. Rajendra & Cheong, Kang Hao. Simulating forest fire spread and fire-fighting using cellular automata. *Chinese Journal of Physics*, 65:642–650, 2020. ISSN 0577-9073. doi: 10.1016/j.cjph.2020.04.001. URL https://www.sciencedirect.com/science/article/pii/S0577907320300873.

[12] Neumann, John Von & Burks, Arthur W. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[13] RealDigital, . Introduction to axi4-lite (advanced extensible interface). URL https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081. (Accessed April 2023).

[14] RISC-V, . About risc-v. URL https://riscv.org/about/. (Accessed April 2024).

[15] Rupp, Karl. Microprocessor trend data, 2022. URL https://github.com/karlrupp/microprocessor-trend-data. (Accessed April 2024).

[16] Tarassov, Eugene. vivado-risc-v. URL https://github.com/eugene-tarassov/vivado-risc-v. (Accessed October 2023).

[17] Weisstein, Eric W. Cellular automaton. URL https://mathworld.wolfram.com/CellularAutomaton.html. (Accessed April 2024).

[18] Wolfram, Stephen. *A New Kind of Science*. Wolfram Media, 2002. ISBN 1579550088. URL https://www.wolframscience.com.

[19] Xilinx, . *AXI Reference Guide*, . URL https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf. (Accessed December 2023).

[20] Xilinx, AMD. What is an fpga, . URL https://xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. (Accessed April 2024).

[21] Xilinx, AMD. Fpga architecture, . URL https://docs.amd.com/r/en-US/ug1291-viv/FPGA-Architecture. (Accessed April 2024).