# Computer Architecture Final: Conway's Game of Life

Samuel Valencia Cabrera | Alex Butler | Aydin O'Leary

December 19, 2020

## Abstract

Logic design hardware tools seem like an interesting space for cellular automata design. Due to the nature of computer hardware being binary, finite state machines are easily implemented; and once defined, can be replicated with ease to produce incredibly fast and inexpensive cellular automaton at a hardware level. Conway's Game of Life, being one of the most iconic for its deliberate rules and unique emergent features, inspired us to create it.
Our process of building John Conway's Game of Life consisted of two parts. The first, software and logic-gate level models that would act as guides for functionality and further implementations. The second, we expanded the models into System Verilog before attempting an FPGA and LED matrix implementation. Along the path to developing our models and implementations, there were plenty unique learning experiences in addition to the plethora of difficulties that we came across along the way.

## Contents

# Why Conway's Game of Life

Our interest in cellular automata predated our involvement with the Computer Architecture class; however, throughout our progression of the course we realized the inherent benefits that a hardware implementation would offer. Since each cell operates as an Finite State Machine (FSM) with the same four rules every clock cycle and doesn't require any inputs once the clock starts ticking, each cell can operate in parallel.

With structures as simple as a rule set, cellular automata are capable of producing truly amazing and unexpected emergent features. Often these features can be found within the system completely separate from the original intentions of the creation, some of which, such as Conway's Game of Life, are even Turing complete.

## What is Conway's Game of Life?

John von Neumann's (creator of Game Theory) definition of life had two qualifications; the entity must have the ability to reproduce itself and be able to simulate a Turing machine. John Conway designed a system that was aimed to meet this definition as simply as he could using only four rules. Thus by meeting this definition, he titled his work the "Game of Life."

#### Rules of Conway's Game of Life

1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

# Implementation

The production cycle was broken into two parts: the preliminary stage that comprised most of our work modeling & breaking down of the underlying mechanics in the system. The secondary stage was our progression to a hardware implementation, this was largely comprised of working in Verilog and troubleshooting through the FGPA's quirks.

## Modeling & Theory

While we understood there were parallels between digital logic circuit design and Conway's Game of Life (GOL) that would allow for a GOL implementation to be relatively intuitive, we needed to flesh this out.

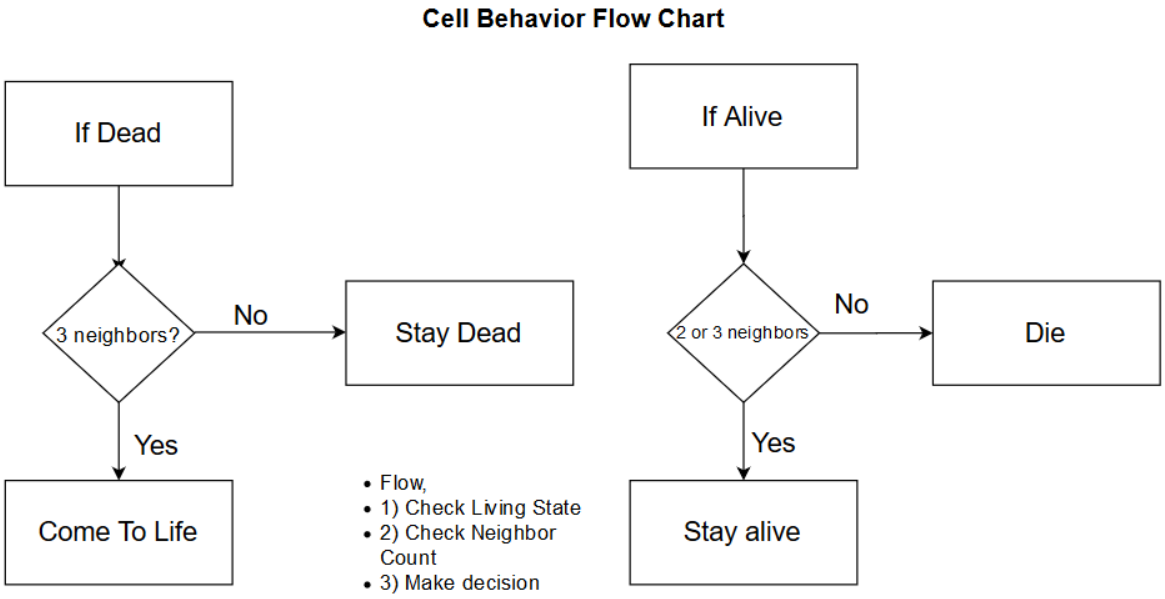


Figure 1: A flow chart depicting the behavior of a single cell.

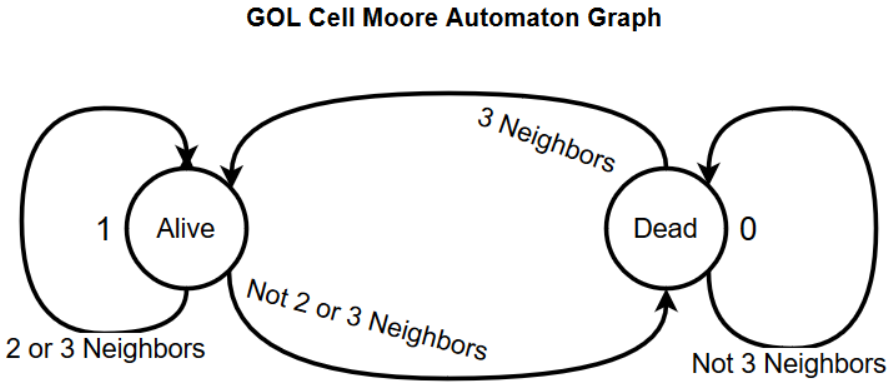


Figure 2: A Moore automaton depiction of a single cell.

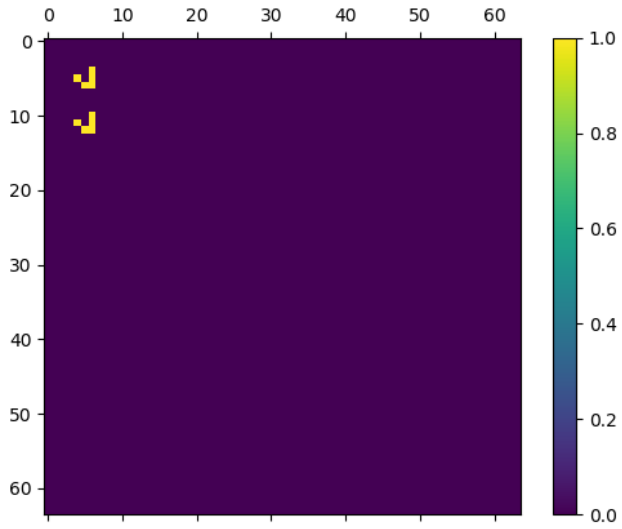


Figure 3: Two gliders in our Python implementation of Game of Life.

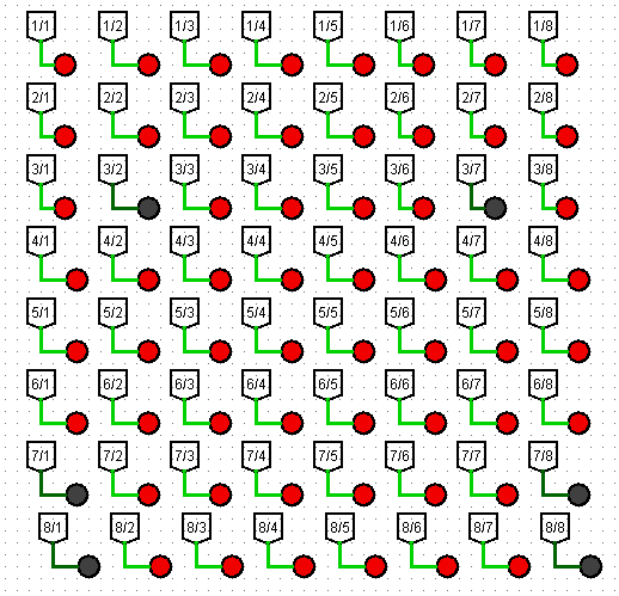


Figure 4: A hard-to-see pattern in our Logisim implementation of Game of Life. This was more of a proof of concept and a way of testing hardware-level implementation than an actual deliverable.

Starting out, we knew the complexity cellular automata could bog down our ability to debug problems early on, and even minuscule errors that have a small chance of occurring would propagate throughout the whole system, and effectively ruin everything. To prevent these errors from stacking up, we created "golden egg" models to compare our system; these included Python and LogiSim implementations, which can be seen in figs. 3 and 4.

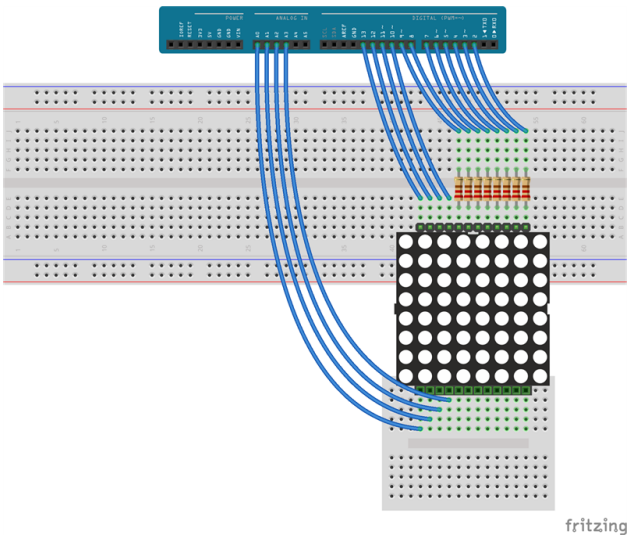


Figure 5: A sample setup using an 1588A 8x8 24-pin LED matrix.

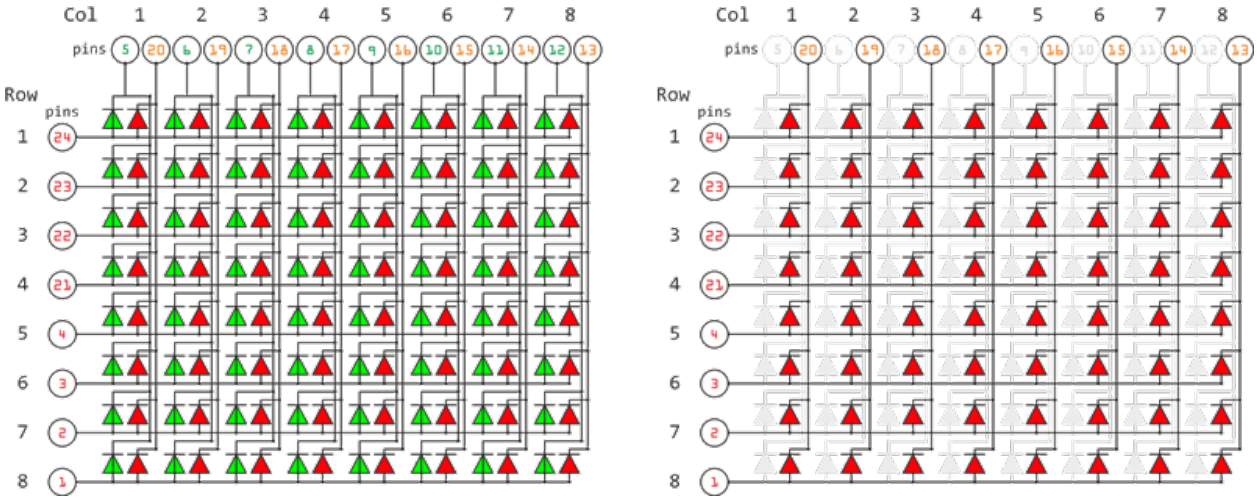


Figure 6: A schematic for a 1588A board similar to ours. Note the lack of a driver chip of any kind, which enables the FPGA to drive the LEDs directly.

# Our Project

## Code

Our code is available at the github repository `zbwrm/game-of-life`. The repository contains implementations of our Game of Life design in Logisim, Python, and Verilog, as well as a Vivado project containing our current work towards FPGA system integration.

The Verilog implementation is structured around a `CELL` module that takes in inputs from its eight Moore neighbors and then decides its next state using a simple state machine. These cells are then packed into a matrix and connected to one another to create the Game of Life.

## Hardware

We used a Digilent Zybo Zynq-7000 connected with a 1588A 8x8 24-pin LED matrix. The only documentation we could find regarding driving this array without a MAX7219 driver chip was in Russian; we used Google Chrome's inbuilt translation function.

We found that there is a large amount of variation on the 1588A LED matrix, so if you decide to use one of those you may need to spend a decent amount of time testing it and documenting which pins are which ends of each diode, and which pins correspond to rows and columns. See fig. 5 for a sample setup using this matrix and fig. 6 for a sample pinout of a 1588A matrix. Ours was extremely similar to this setup: we connected each column and row to an output pin on the FPGA, using 330Ω resistors in series with each diode to limit current. Additionally, we loaded SystemVerilog files onto the Zybo and enabled the Zybo's output pins using the Xilinx Vivado Design Suite.

In order to light up an LED, pass a high signal of 3.3V to the entire row, and pass low signals of ground to the individual LEDs in that row you want to activate.

In order to light up the whole matrix, we planned to implement a rolling signal that lit up each row in quick succession, in a form of pulse-width modulation. This would allow us to switch rows faster than the human eye could see, creating the illusion that the entire grid was active at the same time.

## External Resources

In addition to the documentation linked above, we also made extensive use of Xilinx tutorials such as the introductory tutorial and the Xilinx Community Forums to establish a good workflow. For Game of Life information, we used the websites conwaylife.com and playgameoflife.com. Finally, we used a myriad of online resources found through Google to bolster our knowledge of Python and Verilog syntax in the process of writing and debugging our models.

## Difficulties

There were two main subsets of difficulties that we faced.

### Telework

During this project, Alex and Aydin were living in a commune in Vermont while Sam was living in a Miami apartment. Because a large part of this project was hardware-based, this created unique challenges. We received Zybo FPGAs sent by Olin College of Engineering to test designs with, but only Sam had a version of the LED matrix that we intended to display the Game of Life on. This pushed us into a workflow of less-than-rapid iteration over Zoom, and made it difficult for us to work asynchronously on the hardware aspect of the project. This being said, after multiple checkins with the course instructure, Jonathan Tse, we were elucidated that our experience was all to common in industry, even outside of a global pandemic, due to the distribution of the global industry assets

### System Integration

Another major roadblock was the difficulty of system integration across our project. There were three main subsections of the project that needed to be integrated with one another: the Verilog Game of Life implementation, the Vivado Design Suite and Zybo FPGA, and the 1588A LED matrix.

The Verilog Game of Life implementation was arguably the easiest part of the project. Our initial prototype was a gate-level reconstruction of the Logisim model. Once we had built a Verilog wrapper that let the FPGA interact with the matrix of `CELL`s that our Game of Life consisted of, we began testing our early prototype and quickly realized it was not calculating each cell's number of Moore neighbors properly. We then reimplemented the `is_2` and `is_3` modules in behavioral Verilog for proper neighbor detection, which gave us a fully-functioning Verilog Game of Life model.

The FPGA interface was a bit of a pain, to be honest. The initial licensing and download process was confusing, and Aydin spent about two work days simply downloading, licensing, and establishing a workflow on the Vivado Design Suite. The Suite is not intuitive at best and has spotty documentation, so there was a large amount of trial and error (although the Xilinx tutorial linked above helped a lot). At one point, the team fixed a major pin-assignment issue by backing up our Verilog source files, creating an entirely new Vivado project, and discarding the old one. Critical settings are hidden beyond several sub-menus, and error messaging is unhelpful at best and misleading at worst. Additionally, and this is more the team being used to software development than a shortcoming of the design software, the long compile and distribution times made testing incremental changes incredibly time-consuming.

The difficulties in our final stage, the LED matrix, were largely our own fault. We only spec'd out our LED matrix, a random 1588A board that Sam had laying around, towards the end of our project, after we had already established an FPGA workflow. This created a lot of confusion, as noted in the Telework section, as Sam had access to the matrix while Alex and Aydin did not.

This created an onus on Sam to make up for the lack of documentation on the matrix by testing it, which imbalanced the work load of this project.

## Reflection

If we were to redo this project, we would front-load our actual model implementation more, leaving a lot more time for system integration. No one ever expects system integration to take a long time, but a complicated project combined with licensed industry software makes it an almost Herculean task.

We would also pick out a final spec for the LED matrix earlier so we could ensure that everyone in the team can work on that stage of the project concurrently across Zoom. Besides the difficulty of working remotely on hardware, it's annoying having all the members of the team but the one with the hardware be able to meet and therefore not having a very productive meeting.

## To-dos

Although the work period on this project for this class is finished, our entire team is living together next semester and will have access to FPGAs as well as the passive electrical components necessary to drive the LED matrix. As such, we plan to get a functional implementation going during that period of time. After that, we'd like to do an even more comprehensive and "cleaner" write-up of the project online to add to our portfolios.
Further to-dos include diving deeper into automata theory, as well as breaking up the computing of the each cell into chunks of 4, 9, 16, etc. and performing a pareto analysis of what effect this would have on the overall system.