# CircuitSim Project Proposal

Andrew Yang (asy2130)
Case Schemmer (chs2164)
Faustina Cheng (fc2694)
Jary Tolentino (jt3577)
Ming Gong (mg4264)

March 2025

## 1 Introduction

Our plan for the final project is to use our FPGA as a hardware accelerator for simulating analog circuits, similar to SPICE and the Falstad Circuit Simulator (`https://www.falstad.com/circuit/`). We aim to recreate the circuit simulation and user interface. Specifically, we would like our project to do the following:

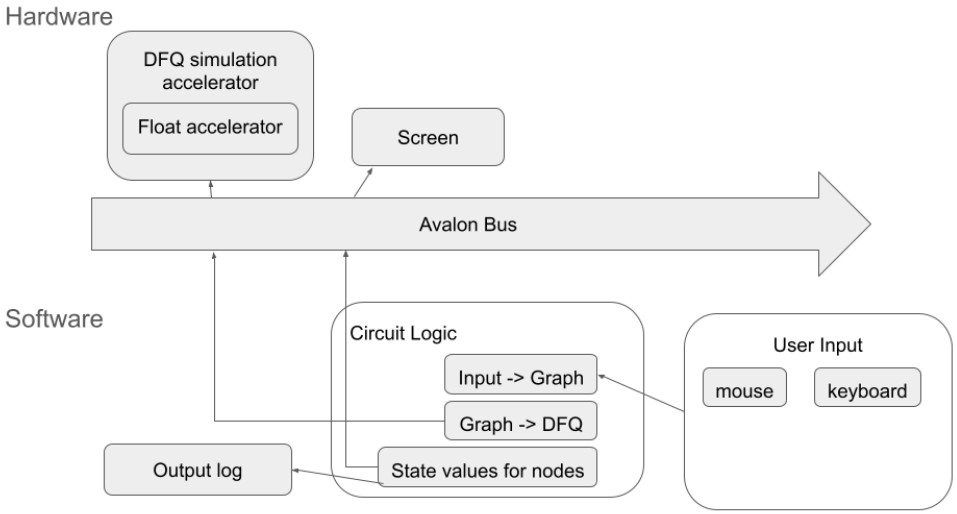
1. The user can use the keyboard and mouse to place circuit components on a grid
2. Software translates the sketch to a graph with nodes and edges (wires and components)
3. Software translates the graph to differential equations
4. Hardware simulates the DFQs and returns the results
5. The node results are exported, or displayed back to the VGA screen

This project will use:

- Peripherals: keyboard and mouse interface for the user to craft a circuit
- Hardware: acceleration for differential equation simulator
  - Floating point arithmetic acceleration
- Software: graph data structure to represent a circuit

## 2 System Block Diagram

Display optional, may not have enough memory




# 3 Algorithms

## 3.1 User input

The screen is divided into squares in a grid, for example a 20 * 20 grid

- Keyboard to select component (wire, resistor, delete)
- Keyboard to specify the component values (may need a UI for this. It can be simple as a text line at the bottom)
- Mouse to drag around and place components

## 3.2 Circuit Graph

Sparse directed graph with nodes as voltages and edges as components (resistor, capacitor, diode, etc.)

Representing the graph using 3-tuples, for example, `(0, 1, 'r')` means node 0 and 1 are connected by a resistor.

For multi-terminal devices, such as op amp, we can create 3-terminal edges: `(0, 1, 2, 'opamp')` and make internal simplifications.

We also want to provide constraints for the input circuit to prevent unpredictable behavior, so we plan to ensure that the user doesn't input things like short circuits.

## 3.3 Differential Equation

From the graph, we need to declare state variables (voltages and/or currents) and write differential equations relating them.

- This part can go in the software or hardware

We then need hardware to do the DFQ simulation

- Algorithm: Euler or Range-Kutta
- Need a lot of float arithmetic. This is VERY important for this project

### 3.4 Output

Alternatively, we need to export the data structure and node values for more detailed analyses.

- Edwards asked for a 4th-order filter, which we have to export and plot the data

If we have memory, a VGA display will be awesome!

- Voltages are represented as colored components (green for high, red for low, similar to Falstad)
- (optional) Current represented as moving electrons

Since our grid is 20 * 20, we just need to display the colored component symbols, which should be similar to video game displays.

## 4 Resource Budgets

Want to simulate at least 20 * 20 sparse graphs. There would be 400 nodes then, each of which can have up to 4 (upper bound, as corners and edges have slightly less) connections, which are components. Each node would be indexed (requiring 9 bits of memory to represent 400 numbers), and have four connections to other nodes. Assuming 16 or less distinct circuit elements (resistors, capacitors, inductors, diodes, BJTs, MOSFETs, OP-Amps, switches, etc), each of these connections would take 4 bits. There is no need to describe what node it is pointing to, as this can be done in a simple transition function (i.e. every adjacent node can be calculated easily). As such, we have encoded our 400 nodes in roughly $400 \cdot (9+4 \cdot 4) = 10000$ bits $= 1250$ Bytes $\approx 1.2$kB, which is much less than the 500kB max.

For accelerators, the main consideration is how much input and how much output need to reside in the FPGA at one time, and whether the FPGA can accommodate it all. Fortunately, the memory for this system is not too complex. Even in the more precise Runge-Kutta algorithm, it requires less than 10 floating point values to be stored at a time, taking up only 40 Bytes of memory. It's always possible to move data into and out of the FPGA as it is being processed, or even give the FPGA direct access to HPS memory, but these are much more difficult. Fundamentally, the memory of the system should not be too steep. Since this isn't any real vector calculus simulation, it only requires a small piece of memory for position, velocity, and their angular counterparts.

## 5 Hardware/Software Interface

The Hardware will communicate with the software via commands. Unlike other circuit simulation, this will be a "lazy" program that computes only what the user requests. As such when a user is in "circuit description mode", the software will convert the graph data structure into a set of usable pixel information and send this to the screen. In "simulation mode", the software will read the calculations performed from hardware, and then send this to the hardware in a set of usable data points to plot.

## 6 Milestone

1. Hardware accelerator for floating point numbers and DFQ simulation. This is the key to the project (according to Edwards)
2. Software prototype of the graph data structure, and functions to modify and simulate the graph
3. User interface with mouse and keyboard
4. Export (and/or) display the data

# 7 Tweaks

1. We can build a digital simulator. The hardware part should be easier, but we need a bigger scale simulation for this to be non-trivial
    - We may not need a display for this, as a larger grid may use too much memory
    - We can also add digital components on top of the analog framework
2. If floating point arithmetic is too expensive, shall we use fixed-point numbers in hardware (?)
3. We have a lot of room for the components and functions to include, such as
    - Minimum: voltage source (DC/AC), resistor, capacitor, (opamp)
    - Nonlinear components: diode
    - Dependent sources
    - Digital gates
    - More functions in the UI: move/copy components