CircuitSim Design Document

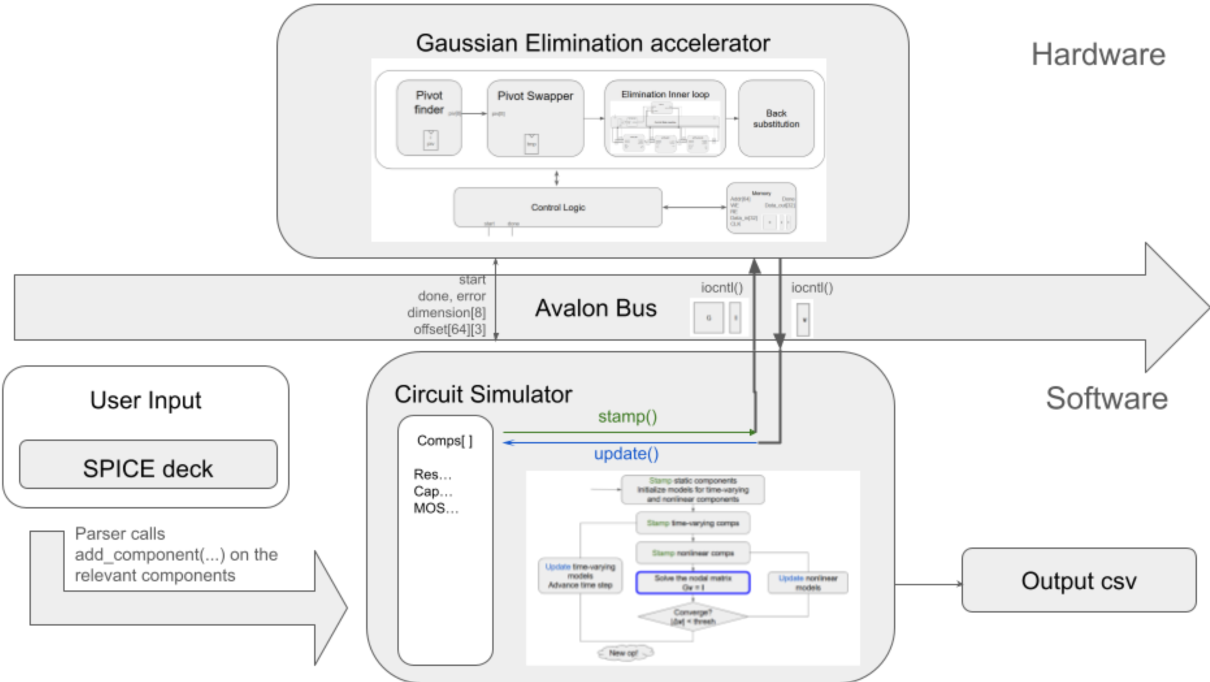
Andrew Yang (asy2130)
Case Schemmer (chs2164)
Faustina Cheng (fc2694)
Jary Tolentino (jt3577)
Ming Gong (mg4264)

April 2025

1 Introduction

We are developing an analog circuit simulator on the FPGA, inspired by professional tools like SPICE and Falstad. The simulator parses SPICE netlists into node and component representations. In each simulation step, the software linearizes the components and stamps their component models and stamps their contributions into matrix equations. These equations are then offloaded to a hardware module, which solves them using Gaussian elimination. The resulting node voltages are returned to the software for further processing or visualization.

2 System Block Diagram



Details of the Circuit Simulator are included in Section 3.3 and details/block diagrams of the Gaussian Elimination accelerator are included in Section 4.

3 Algorithms

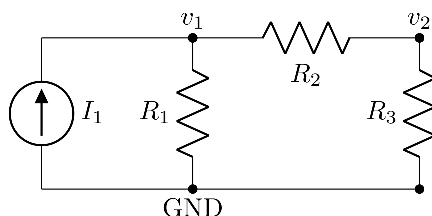
The user will provide input in the form of a SPICE-like netlist, specifying each component's type, node connections, and parameters (e.g., resistance, capacitance, etc.). The software parses the components into matrix equations using linear approximations and node voltage analysis. The FPGA performs Gaussian elimination to solve the resulting linear system, which operates in $O(n^3)$ time complexity. The simulation outputs the computed node voltages as a CSV file.

The software maintains two data structures:

- A component array that registers every circuit component, their connections, values, and memories
- The node matrix entries $G, \mathbf{v}, \mathbf{I}$.

3.1 Node Voltage Analysis

Node Voltage Analysis (NVA) is a method used to determine the voltage at various nodes in an electrical circuit. In the case of a purely resistor network, we can apply Kirchhoff's Current Law (KCL) at each node and use Ohm's Law to express the currents. Let us consider a simple resistor network with current sources.



In this circuit, we can use Node Voltage Analysis to find the voltages v_1 and v_2 at nodes 1 and 2, respectively. The steps are as follows:

1. Apply Kirchhoff's Current Law (KCL) to each node, which states that the sum of currents leaving a node is zero.

   - At node v_1, the sum of the currents **leaving** v_1 (through the resistors and the current source I_1) should be zero:

   $$\frac{v_1}{R_1} + \frac{v_1 - v_2}{R_2} - I_1 = 0$$

   - At node v_2, again, the sum should be zero

   $$\frac{v_2}{R_2} + \frac{v_2 - v_1}{R_3} = 0$$

2. Construct a system. Define $G_n = \frac{1}{R_n}$. Move all the G (resistance) term to the LHS, and all I (current) terms to the RHS, We now have a system of linear equations:

   $$\begin{cases} G_1 v_1 + G_2(v_1 - v_2) = I_1 \\ G_3 v_2 + G_2(v_2 - v_1) = 0 \end{cases}$$

   Turning into a matrix, we have:

   $$\begin{bmatrix} G_1 + G_2 & -G_2 \\ -G_2 & G_2 + G_3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} I_1 \\ 0 \end{bmatrix}$$

   $$G\mathbf{v} = \mathbf{I}$$

in matrix notation. We can then solve for v_1 and v_2.

Since the components are linear, we can iterate through the array of components and add their contributions to G and $\mathbf{I}$ one-by-one. This process is known as **stamping**.

Through nodal analysis, we can use Kirchhoff's Current Law to calculate the voltage at each node, when given n nodes with unknown voltage values. This works when a circuit has current sources, but voltage sources are different because we cannot determine the current flowing through it purely by looking at its voltage value.

As a result, we use modified nodal analysis, which solves this by adding the unknown current values to the $\mathbf{v}$ vector. As a result, the size of the $\mathbf{v}$vector is now n plus the number of unknown current values, which also means that the G and $\mathbf{I}$ vectors must increase in size accordingly so that the equations can be solved. This is solvable, since we also know that difference in voltage between the two nodes coinciding with the voltage source is equal to the voltage source's voltage. The matrix now has enough equations to solve for the unknowns, and is ready to be solved using any method that can solve simultaneous equations.

3.2 Software: Input Parsing

3.2.1 SPICE Parser

Our design supports a subset of circuits described using a SPICE netlist, specifically those that only have the components supported for this project. These components are:

- Current sources
- Voltage sources
- Resistors
- Capacitors
- Inductors
- Diodes

3.2.2 Circuit Interpretation

In order to convert the netlist into a mathematically solvable form, we reconfigure the netlist as components and nodes, which are then "stamped" one by one onto the G, V and I matrices.

For simple components such as resistors and current sources, this is simple using nodal analysis. Voltage sources are slightly more complicated, and require modified nodal analysis to derive the current flowing through the current source. For time-varying and non-linear components, simply applying Kirchhoff's Current Law is not sufficient, since Gaussian elimination cannot directly solve nonlinear terms. As a result, the software must translate these components into linear companion models, which closely approximate the original components, which is expanded upon in later sections.

3.2.3 Matrix Solver

To solve the system of equations, our design uses Gaussian elimination as the algorithm of choice. This will be done on the FPGA, which will take the matrices and return the solved unknowns.

3.2.4 Data Visualization

The simulation will output the node voltages at each timestep (the time duration is specified within the SPICE netlist) to a CSV file, which can be easily visualized using any plotting tool.

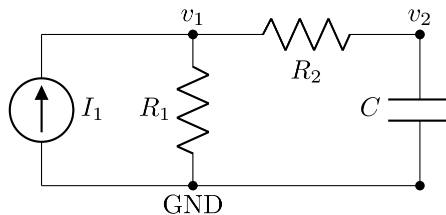
If we have time, we would also like to construct a simple animation of the circuit as a graph, so that we can approximately visualize the circuit described in the netlist with the state of the circuit at each timestamp.

3.3 Modeling Time-varying and Non-linear Components

Now we have a general workflow for static, linear components. For more complicated components, we will need to employ linear companion models.

3.3.1 Capacitors: Backwards Euler's Method

Let's replace R_2 with a capacitor with capacitance C.



Using backward Euler's method,

$$i(t) = C\frac{dv_C}{dt} = C\frac{d(v_2 - 0)}{dt}$$
$$i(t_0 + \Delta t) \approx C\frac{v_2(t_0 + \Delta t) - v_2(t_0)}{\Delta t}$$
$$v_2(t_0 + \Delta t) = v_2(t_0) + \frac{\Delta t}{C} i(t_0 + \Delta t)$$

Let $V_{eq} = v_2(t_0)$ and $R_{eq} = \frac{\Delta t}{C}$, the equation becomes

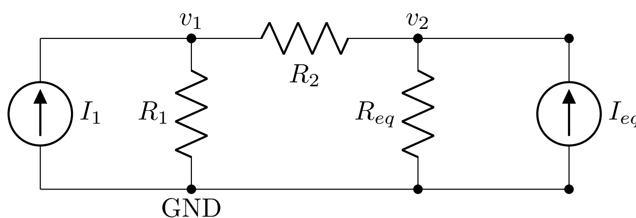
$$v_2(t_0 + \Delta t) = V_{eq} + R_{eq} i(t_0 + \Delta t)$$

This capacitor is converted into a voltage source and a resistor in series, at this particular timestamp.

Voltage sources introduce additional nodes, so we convert it to a current source in parallel (Norton equivalent). Let $I_{eq} = \frac{V_{eq}}{R_{eq}}$

Our simulation becomes

1. At t_0, stamp all static components, and R_{eq}, I_{eq} calculated from the initial condition, or from the previous iteration
2. Solve the matrix
3. Update time step, and update the values of R_{eq}, I_{eq} in the component array based on $\mathbf{v}(t_0 + \Delta t)$



3.3.2 Diodes: Newton-Raphson Method

Diodes are nonlinear, with the following current-voltage relation:

$$i_d = I_s(e^{v_d/V_t} - 1)$$

where I_s, V_t are constants.

The Newton-Raphson method makes a linear model at the current estimate, and updates our model until it converges.

Take the derivative at step 0, where $i_d = i_{d0}$, $v_d = v_{d0}$:

$$G_{eq} \equiv \frac{di_d}{dv_d} = \frac{I_s}{V_t} e^{v_{d0}/V_t}$$

$$
\begin{aligned}
i_d &\approx i_{d0} + G_{eq}(v_d - v_{d0}) \\
&= (i_{d0} - G_{eq}v_{d0}) + G_{eq}v_d
\end{aligned}
$$

Looks familiar! A current source in parallel with a resistor (conductor). Let $I_{eq} = (i_{d0} - G_{eq}v_{d0})$. We have:

$$i_d = I_{eq} + G_{eq}v_d$$

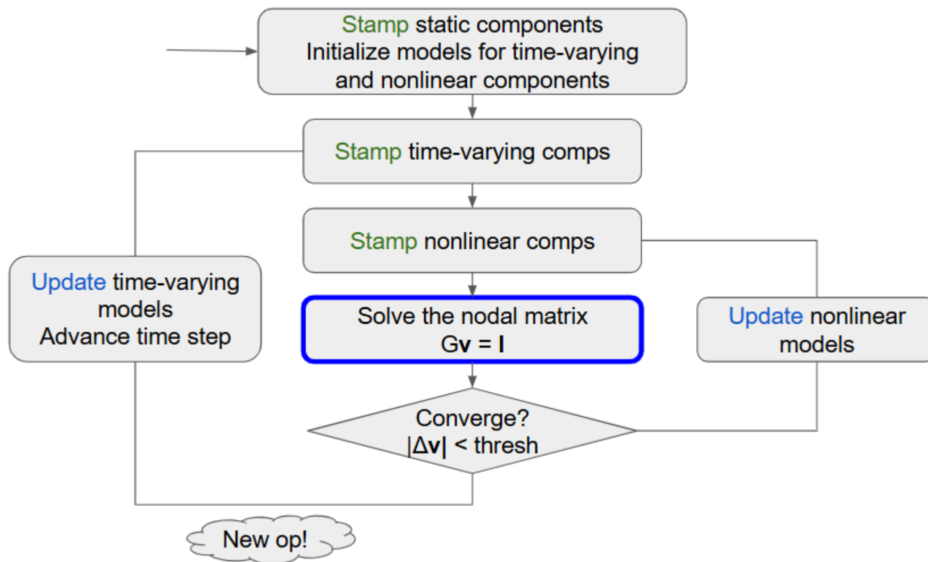
At a particular time (operating point), we perform the following loop:

1. `stamp()` all components. For nonlinear components,
   (a) Make an initial guess on v_{d0}.
   (b) Using v_{d0}, calculate I_{d0}, G_{eq}, I_{eq}
   (c) Add G_{eq} and I_{eq} onto G and $\mathbf{I}$
2. Solve the matrix $G\mathbf{v} = \mathbf{I}$
3. `update()` the new v_{t0} from the new $\mathbf{v}$
4. Check for convergence: find the maximum change of $\mathbf{v}$ from the previous iteration. Finish if it's less than a threshold.

This solves an operating point.

3.3.3 Simulation Summary

Below is the flow chart for the entire simulation loop



- `stamp()` loads the components from the list to $G, \mathbf{I}$
- Hardware solves the matrix
- `update()` updates the components' memory from the results of $\mathbf{v}$

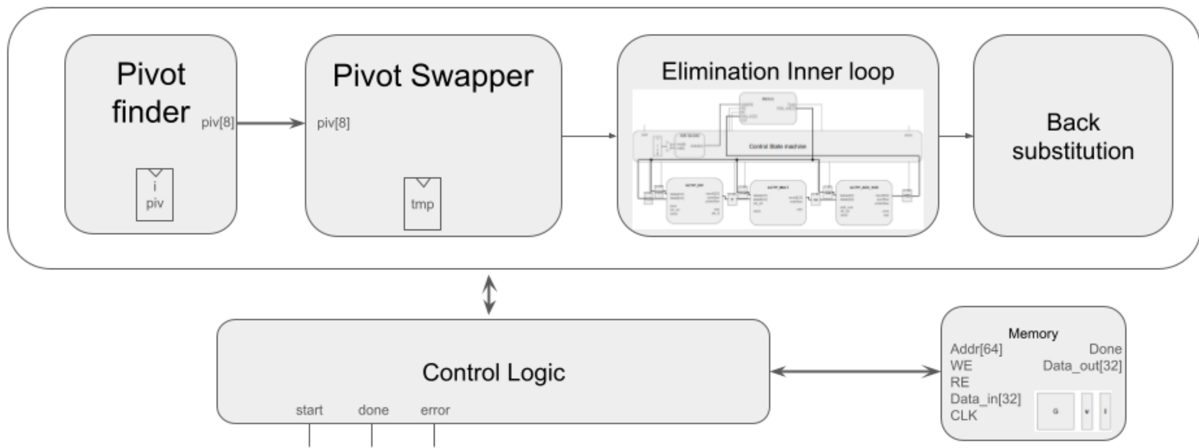
With a similar idea, we can simulate ideal voltage sources, finite-gain opamps, and MOSFETS. Below are some "interesting circuits" that we hope to impress Stephen Edwards

- Diode voltage rectifier

- 4th order Sallen-Key filter
- MOSFET amplifiers
- Digital gates
- Flip-Flop

4 Gaussian Elimination Hardware Accelerator

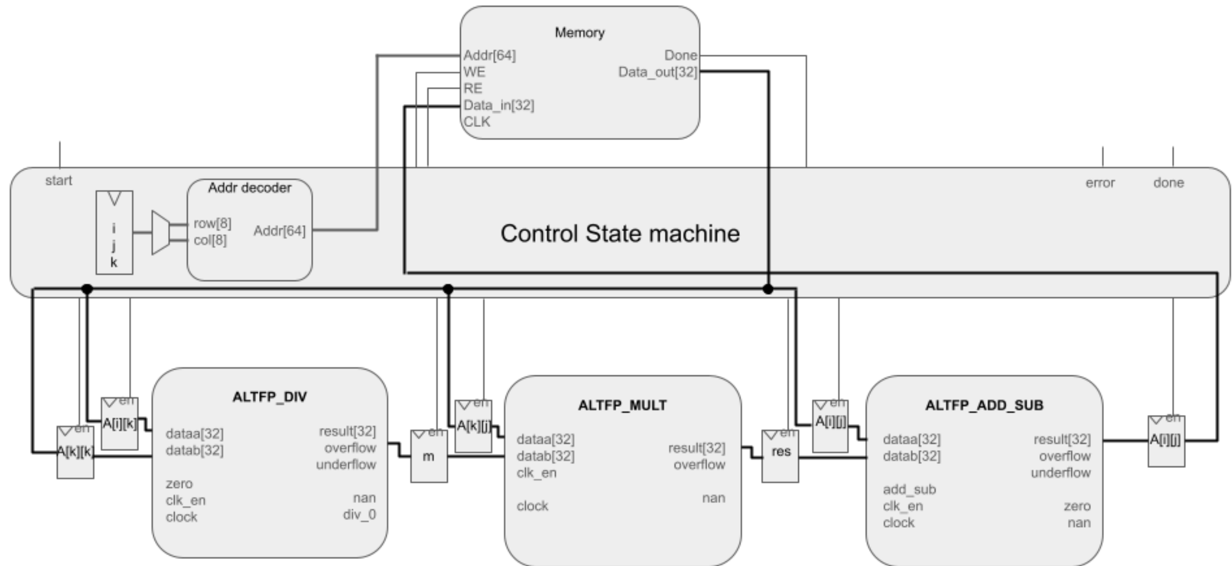
4.1 Outer block design



This is our block diagram for Gaussian elimination. Here, we find a pivot for the Gaussian process. In the pivot finder block, there is internal logic. From here, we check if we need to swap by seeing if `piv=k`. If so, perform a pivot swap using the tmp register. After this, the control is passed to the inner loop (more details below). Finally, we do back substitution for the last step to determine the actual values in the GI-V matrix.

All blocks will read/write from/to the memory to its internal registers, whose addresses and destinations are controlled by a control logic.

4.2 Inner block design



For the inner loop, the elimination process works as follows:

To access memory of a specific entry, we have address decoders to take rows and columns and provide the exact memory address for the value, and then access this memory in "read mode". We used the available Altera Floating Point cores, e.g. ALTFP_MULT, ALTFP_ADD_SUB, and ALTFP_DIV to perform the arithmetic for computing values i and j within the inner block of the Gaussian Elimination. The counters allow us to determine the next state logic for the state flip flop.

After `start`, the control state machine reads the memory and loads the floating point numbers into the corresponding data registers (`en` signal). After a successful computation, the result is written back to memory.

The control state machine will generate control signals for the FP units, and handle their outputs and exceptions (e.g. division by zero)

4.3 Floating Point IP

We intend to use the Floating Point IP provided by Altera. The general overview of all blocks is that they take in two inputs, a clock, and an enable switch. The result appears some number of clock cycles later. This is preset by the user and will allow for slower or quicker clock frequencies. To ensure the result is read at the exact right clock cycle, we use a counter with a threshold signal to read the result. There is an async clear set that works with our IP's reset signal. Finally, there are additional signals for overflow/underflow, zero, NaN, or division by zero (DIV), which will be handled by the state machine.

5 Resource Budget

Our resource budget is constrained by the memory, which in this case is less than half a megabyte. For this project, the memory usage is dominated by the matrices, the sizes of which are determined by the size of the circuit. For this project, we plan to use 32-bit floats, and will have a G matrix of dimension n by n, a v vector of size n, and an I vector of size n. As a result, we should be able to safely solve matrices up to size 256 by 256, which would require approximately 262 kilobytes, safely under the limit. However, this means that we can only support circuits with fewer than 256 unknowns. The nodes will be managed in the

software, and if the software parses the netlist and finds that there are too many unknowns, it will raise an error to the user before the data is passed to the FPGA.

For this project, the representation of the matrix will be under the assumption that it is dense, even if it isn't, because the implementation of Gaussian elimination is significantly easier when all the data is contiguous.

6 Hardware/Software Interface

The main interactions between the hardware and software revolve around the transfer of the G matrix and V/I vectors to the FPGA and sending the result of the Gaussian elimination algorithm from the hardware to software. In order to facilitate this, the software will need to allocate space for the matrix and the two vectors, write to it, and then send the memory addresses to the FPGA so that it can read the data. In order to do this, the hardware must also send a signal telling the FPGA that it is ready to start, as well as the dimension of the matrix, since that decides how the data is read.

The register map will have:

- Byte offset 0: Start flag to notify the FPGA that it can start execution and read from memory, only one bit is used
- Byte offset 4: Done and Error flag that marks when the FPGA is done with the algorithm, or runs into an error, only two bits are used
- Byte offset 8: Matrix dimension, since the max is 256, only 8 bits are used
- Byte offset 12: Lower 32 bits of the memory address that the G matrix starts at
- Byte offset 16: Upper 32 bits of the memory address that the G matrix starts at
- Byte offset 20: Lower 32 bits of the memory address that the V matrix starts at
- Byte offset 24: Upper 32 bits of the memory address that the V matrix starts at
- Byte offset 28: Lower 32 bits of the memory address that the I matrix starts at
- Byte offset 32: Upper 32 bits of the memory address that the I matrix starts at

Once the Gaussian elimination is complete, the FPGA will raise a done flag, signaling to the software that it is okay to read from the v vector.

7 Milestones

1. 4/5 Working prototype for node voltage analysis
2. 4/12 Complete simulation algorithm with time-varying and non-linear components
3. 4/18 Hardware designed
4. 4/30 Hardware complete

8 References

- SPICE algorithm overview: `https://www.ecircuitcenter.com/SpiceTopics/Overview/Overview.htm`
- Nodal analysis: `https://www.ecircuitcenter.com/SpiceTopics/Nodal%20Analysis/Nodal%20Analysis.htm`

- Modified nodal analysis: https://lpsa.swarthmore.edu/Systems/Electrical/mna/MNA2.html#Example_3
- Dependent sources: https://qucs.sourceforge.net/tech/node60.html
- Netwon-Raphson: https://www.ecircuitcenter.com/SpiceTopics/Non-Linear%20Analysis/Non-Linear%20Analysis.htm
- Backwards Euler: https://electronics.stackexchange.com/questions/272012/companion-capacitor-model-in-circuit-simulation