ECE 241 Final Project Report

Project Name: Electrical Circuit Solver and Synthesizer
Completed by: Bingran Hu
Course Term: Fall 2019
Report Author: Bingran Hu Email: bingran.hu@mail.utoronto.ca

I. Introduction and Project Motivation

The term IP Core, also known as Semiconductor Intellectual Property Core, refers to a block of logical architecture that is reusable for future designs. It lies at the boundary between hardware and software, since it functions as a primitive software program but is built of actual wires and fundamental logical elements. This project aims to accomplish a similar goal: a good portion of the source code can be referenced or reused in similar circuit-related program designs, such as the Matrix Solver module, the Video Graphics Array(VGA) drawing CPU, the Supernode Analyzer, and various arithmetic converters.

While the scope of the project limits the types of synthesizable circuit elements to only independent voltage/current sources and simple, fixed resistors, the project serves as a basic prototype for a more general electrical circuit synthesizer/solver that may operate upon more complicated elements such as dependent sources, capacitors, inductors, and even operational amplifiers.

II. Design Process and Program Architecture

This project uses circuit nodal analysis as its main strategy and follows a hierarchical design. The analysis, calculation, and diagram synthesis processes are carried out sequentially in several steps, which are referred to as "Stages". Hence, the main controller implements a Finite State Machine(FSM) that controls the specific stage of the program that should be run. In addition, different stages of the design may end up using the same HEX decoders, Random-access Memory(RAM) and Read-only Memory(ROM), so the main datapath contains some multiplexers that monitor the reads and writes of these RAMs and ROMs based on the current stage of operation. The design is thus hierarchical since each stage also has its own controller and datapath to carry out the required operations.

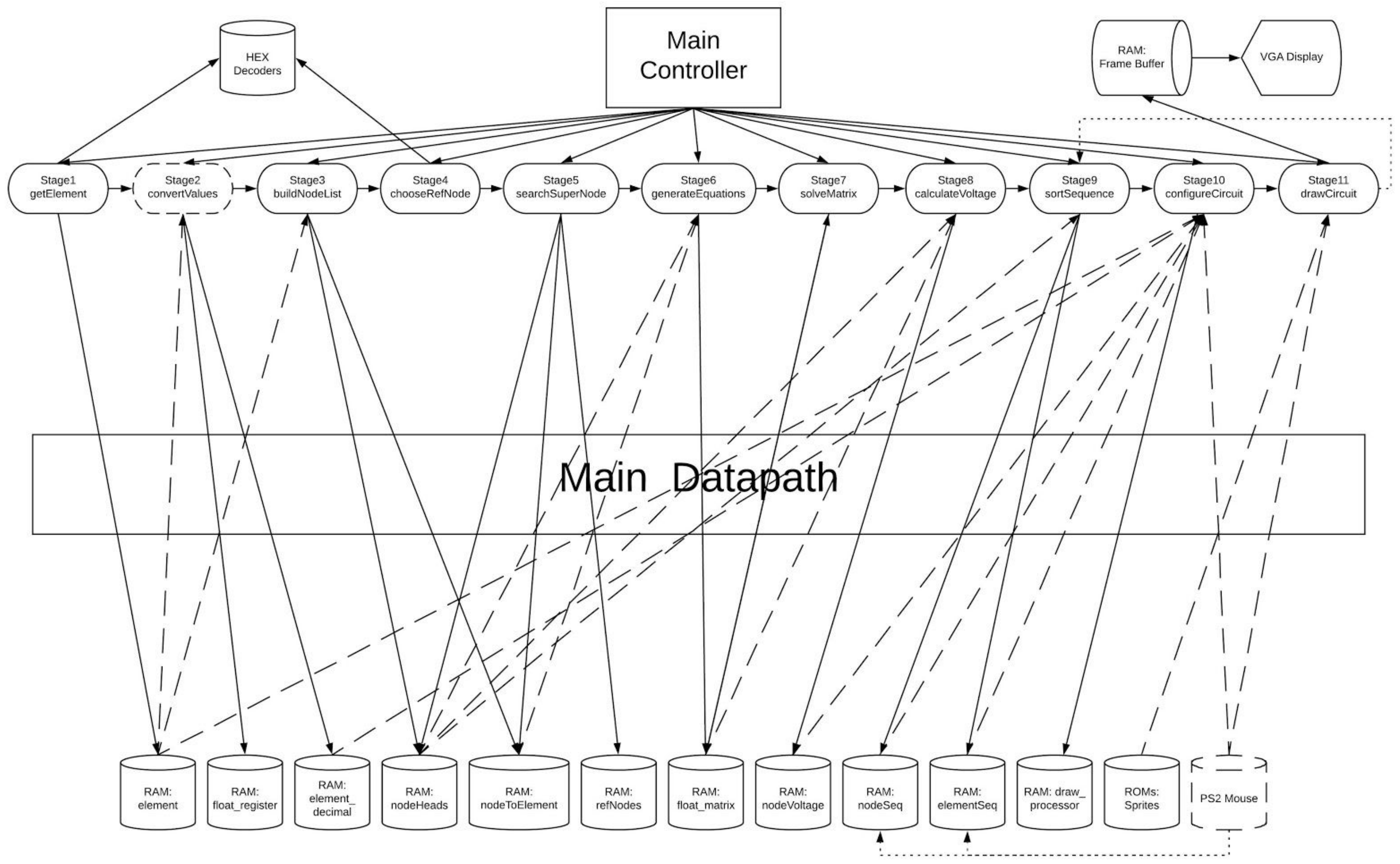
Due to the page limitation of the report, one single main schematic with all the major program components (excluding arithmetic IP cores, specific sprite ROMs, etc.) as well as their interconnections is shown on page 2. Here I present a detailed description of this diagram:

Connections:

- Filled Lines: The process writes new data into the RAM
- Dashed Lines: The process only reads existing data from the RAM
- Dotted Lines: Connections that were planned but not realized due to the project deadline

Stages:

1. getElement:
    a. Read the user's input using FPGA switches and keys. The user inputs circuit elements one-by-one, giving enough info to synthesize and calculate the circuit.
    b. The input sequence: The node type, the node value(base+exponent), the number tag for node A and node B. For the voltage/current source, **node A is the positive end.**
    c. SW represents the current value in binary. Press KEY[3] to confirm value, KEY[1] to reset current element input, and KEY[2] to finish inputting elements.


Main Controller
HEX Decoders
RAM: Frame Buffer
VGA Display
Stage1 getElement
Stage2 convertValues
Stage3 buildNodeList
Stage4 chooseRefNode
Stage5 searchSuperNode
Stage6 generateEquations
Stage7 solveMatrix
Stage8 calculateVoltage
Stage9 sortSequence
Stage10 configureCircuit
Stage11 drawCircuit
Main Datapath
RAM: element
RAM: float_register
RAM: element_ decimal
RAM: nodeHeads
RAM: nodeToElement
RAM: refNodes
RAM: float_matrix
RAM: nodeVoltage
RAM: nodeSeq
RAM: elementSeq
RAM: draw_ processor
ROMs: Sprites
PS2 Mouse

2. convertValues:
    a. Convert the element values into floating point for further calculations. Resistor values' are taken reciprocals to facilitate the generation of matrix equations (eliminate divisions).
    b. Convert the binary number into decomposed decimal digits for the VGA display.
    c. The algorithm that is used to convert the binary number to decimal digits deserves a special mention. The idea is to use a hardware-efficient way to divide a number by 10, and then subtract the result multiplied by 10 from the original number. Repeating this process will produce the decimal digits one by one, and this is shown to be done without the need of synthesizing any multiplier/division circuits[1]. Refer to the **Appendix A**.
3. buildNodeList:
    a. Build a linked list using the user's element inputs. All circuit elements on one single linked list have at least one node in common.
    b. To build the list, two nodes of each element are examined (in the order of user's input, and first A, then B). For the first element, neither of its nodes have been seen before, so two empty lists will be created.
    c. Starting from the second element, if either one of its nodes has already been seen before, this element will be added to the end of the existing list of the corresponding node.
    d. Memory is used to store the current end address of each list so that new elements can be added successfully. Once added, the list updates its current end address accordingly.
    e. E.g. the input sequence is:
        i. Voltage Source, 10V, A-1, B-0
        ii. Voltage Source, 2V, A-3, B-1
        iii. Current Source, 2A, A-2, B-0
        iv. Resistor, 4Ω, A-1, B-2
        v. Resistor, 8Ω, A-3, B-2
        vi. Resistor, 3Ω, A-0, B-3
    f. The resulting linked list is:
        i. Node 0: 10V(#1, B)-->2A(#5, B)-->3Ω(#10, A)
        ii. Node 1: 10V(#0, A)-->2V(#3, B)-->4Ω(#6, A)
        iii. Node 2: 2A(#4, A)-->4Ω(#7, B)-->8Ω(#9, B)
        iv. Node3: 2V(#2, A)-->8Ω(#8, A)-->3Ω(#11, B)
4. chooseRefNode:
    a. Choose the ground reference node that will have 0 voltage. This will affect the supernode searching process as the ground node's supernode will be searched first.
5. searchSuperNode:
    a. Each supernode is a collection of nodes that are connected by voltage sources. As long as the value of one of them is set, the values of other nodes can be calculated directly.
    b. Apply a Depth-first Search(DFS) on the node linked lists. Whenever a voltage source is encountered within a node list, a supernode extension is detected and the value difference is recorded down unless the node on the side of the element has already been explored before. Contact the author or refer to the source code for a detailed explanation/implementation.
6. generateEquations:

   a. Generate the matrix equations for each reference node in the circuit. Note that the number of reference nodes is equal to the number of supernodes. As in the previous case, there are four reference/super nodes in total. Hence, the augmented matrix is 4 by 5.
   b. The voltage values general go into the numerators, the resistor values into the denominators, and the current values into the constant vector section. Voltage differences specified by the supernodes also contribute to the constant vector section.
7. solveMatrix:
   a. Starting from the first column, repeat the following steps to convert a matrix to Reduced Row-echelon Form (RREF):
      i. Find a non-zero leading number in the current column (the nth column)
      ii. Swap this row with the nth row if they are different
      iii. Divide the current nth row by its leading number to obtain a leading one
      iv. Clear this column using the leading one
      v. Repeat the process for the next column, i.e. the (n+1)th column
   b. E.g. The first column has been processed. Now process the second one:

$$\left[\begin{array}{cccc|c} 1 & 3 & 4 & 2 & 3 \\ 0 & \mathbf{0} & 2 & 4 & 4 \\ 0 & \mathbf{3} & 9 & 6 & 3 \\ 0 & 2 & 4 & 5 & 2 \end{array}\right] \longrightarrow \mathit{swap} \longrightarrow \left[\begin{array}{cccc|c} 1 & 3 & 4 & 2 & 3 \\ 0 & \mathbf{3} & \mathbf{9} & \mathbf{6} & \mathbf{3} \\ 0 & 0 & 2 & 4 & 4 \\ 0 & 2 & 4 & 5 & 2 \end{array}\right] \longrightarrow \mathit{divide} \longrightarrow \left[\begin{array}{cccc|c} 1 & \mathbf{3} & 4 & 2 & 3 \\ 0 & 1 & 3 & 2 & 1 \\ 0 & \mathbf{0} & 2 & 4 & 4 \\ 0 & \mathbf{2} & 4 & 5 & 2 \end{array}\right] \longrightarrow \mathit{clear} \longrightarrow \left[\begin{array}{cccc|c} 1 & \mathbf{0} & -5 & -4 & 0 \\ 0 & 1 & 3 & 2 & 1 \\ 0 & \mathbf{0} & 2 & 4 & 4 \\ 0 & \mathbf{0} & -2 & 1 & 0 \end{array}\right]$$

   c. Refer to the source code or contact the author for implementation details
8. calculateVoltage:
   a. Use the matrix results and supernode relationship to calculate the voltage value for each node, regardless if it is a reference node or not.
   b. Convert the values from floating-point to integer, and obtain the decimal form for display
9. sortSequence:
   a. For this project, all circuit elements are drawn vertically, as this is the best way to synthesize a circuit diagram.
   b. This stage first initializes the display sequence of elements (from left to right) and nodes lines (from bottom to top) according to user inputs, and then change the sequence according to the drag movement of the user's PS2 mouse *(not implemented)*.
10. configureCircuit:
    a. Retrieve the information about the display sequence of the nodes/elements (Stage9), and the nodal voltage values and circuit element values in their decimal forms (Stage8, Stage2) for VGA display.
    b. Encode all the drawing operations into 64-bit command lines for the draw_processor to process during the next stage. Refer to the Appendix for a detailed description of how the sprite information is encoded in each command line.
11. drawCircuit:
    a. Carry out the command lines stored in the draw_processor CPU during the configureCircuit stage

This concludes the discussion of all of the major stages in the program. Everything else is left to the discussion of implementation details, which is mostly beyond the scope of this report.

III. Report on Success

There are two major criteria for measuring the success of this project: the ability to calculate the nodal voltages correctly, and the ability to synthesize a circuit diagram that can be readily understood. Please refer to the **Appendix B** section for an abundance of examples that can prove that the program is fully functional if:

- The User only inputs information of a single circuit
- The element values that do not exceed an absolute magnitude of 10^6 (More extreme values would require more information storage registers and more complicated hardware configurations)
- The number of nodes and/or elements does not exceed 12.

In addition, the program provides the following features:

- The ability to directly display node voltage values and element values on the VGA screen
- The ability to choose an arbitrary ground node as the whole circuit's reference node
- The ability to invert voltage/current source symbols for a more intuitive display

IV. Lessons Learned

For this project, the available time ran out before the PS2 mouse has got a chance to get fully implemented. In the original design concept discussed during the uniqueness-approval meeting was that the users can use the mouse to click on specific nodes and elements to retrieve their numeric data. However, this goal is still achieved by displaying their values directly onto the VGA display using sprites. Hence, the functionalities and the user experience of this program remains intact.

However, when the idea of this project first came into mind, the immediate concern was over devising appropriate methods for drawing/synthesizing "elegant" circuits. In order to make the circuit "elegant", crossovers between wires and circuit elements should be minimized, or even forced to zero. At the early stage of the design process, I came up with a complicated drawing heuristic that would work for 95% of the circuits (planar circuits). The details of this heuristic are similar to the DFS search techniques used for the supernode detections and will not be elaborated here. But after the introduction of the PS2 mouse, a second idea emerged: instead of putting the burden of the circuit diagram synthesis on the programmer, why not allow users to freely re-configure their circuit diagram as they wish?

Hence, with this idea in mind, the mouse module is designed to connect to the "sortSequence" stage of the program (as shown in the main block diagram by the dotted lines) and provide constant feedback of users' mouse movements and clicks. The dragging of the mouse will update the sequence of the node/element that will appear on the VGA display. A frame buffer is also used so that only necessary parts of the screen are erased and redrawn every time there is an update to the diagram.

Sadly, this implementation failed to be realized before the project deadline. The chosen compromise is to draw the node lines using the dashed lines in order to better distinguish them from the sprites of various circuit elements. In some ways, the circuit diagram now looks like a real protoboard used in the ECE labs, and it serves the purpose of the project well with its clear presentation style.

Another problem is that the max frequency Fmax dropped to 25.6MHz in the final version of the program while the 50MHz clock is still being used. This might generate garbage values in various registers. However, the program is simply too complicated for me to fix this timing issue, at least for now.

Last but not the least, it was fun and exciting experience to solo the whole ECE241 project. But if I were going to start everything over, I wouldn't do it again.

V. References

[1] “Divide by 10 using bit shifts?” [Online]. Available: https://stackoverflow.com/questions/5558492/divide-by-10-using-bit-shifts. [Accessed: 28-Nov-2019].
[2] Tomasz S, Czajkowski, “VGA Adapter Source Code”, July 10, 2007 [Online]. Available: https://piazza.com/utoronto.ca/fall2019/ece241h1f/resources [Accessed: 27-Oct-2019].
[3] ECE241 Staff, “IP cores for the Altera DE1-SoC board” [Online]. Available: http://www.eecg.toronto.edu/~pc/courses/241/DE1_SoC_cores/ [Accessed: 25-Nov-2019].

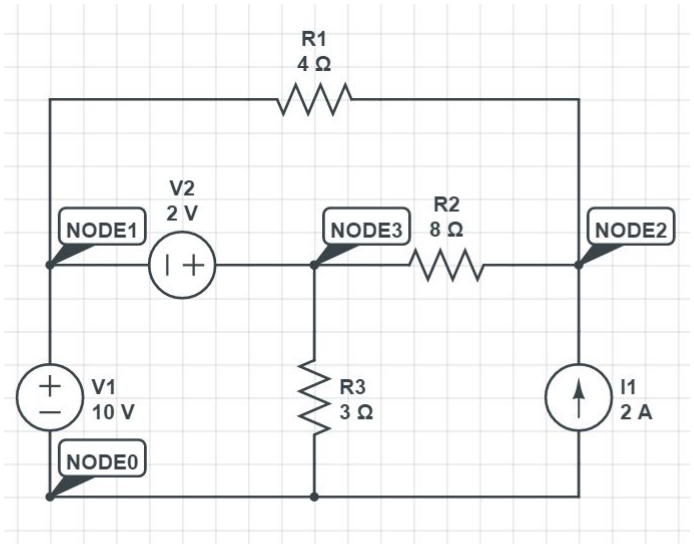
VI. Appendices

Appendix A: Fast Algorithm for Division by 10 [1]

1. Given the original number N in binary form, do the following steps:
2. q = (N >> 1) + (N >> 2);
3. q = q + (q >> 4);
4. q = q + (q >> 8);
5. q = q + (q >> 16);
6. q = q>>3;
7. r = n - (((q << 2) + q) << 1)
8. q = q + (r > 9 ? 1 : 0)
9. The decimal residue is N - (q << 3) - (q << 1) (q is multiplied by 10)

Appendix B: Measure of Success Examples

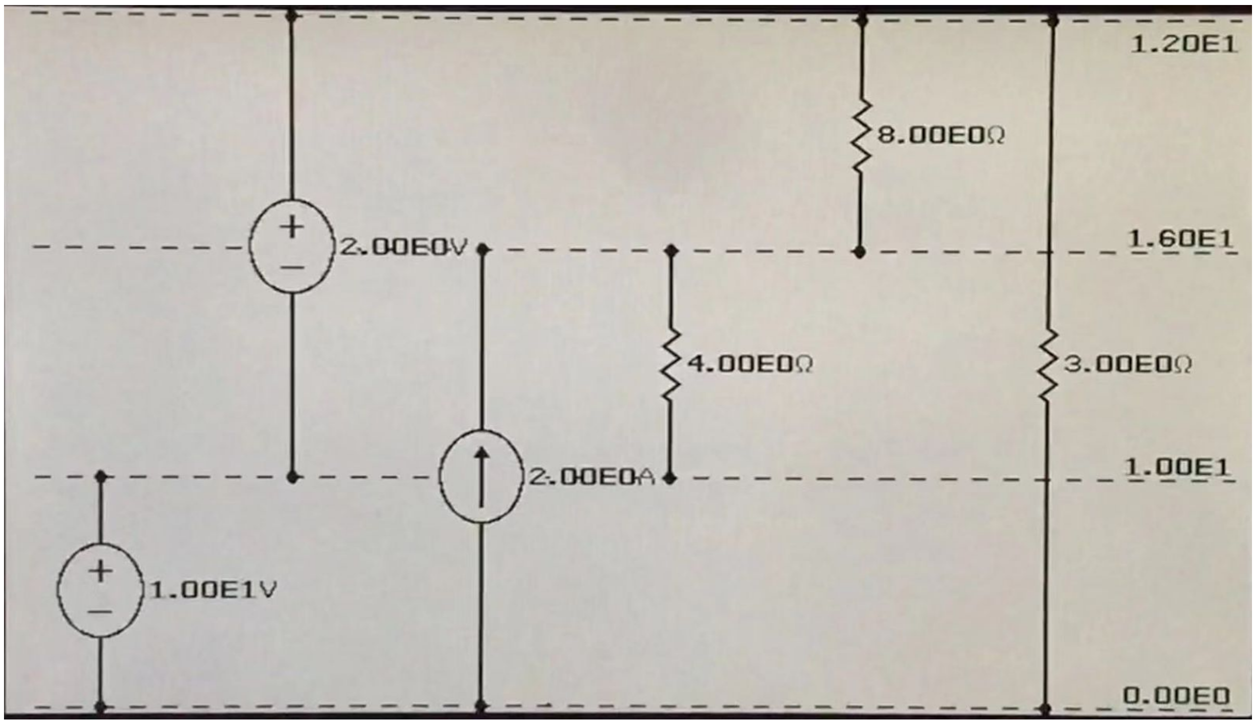
Example 1: Solving a circuit with all three elements + 2nd degree supernode (Node 0, 1, 3)



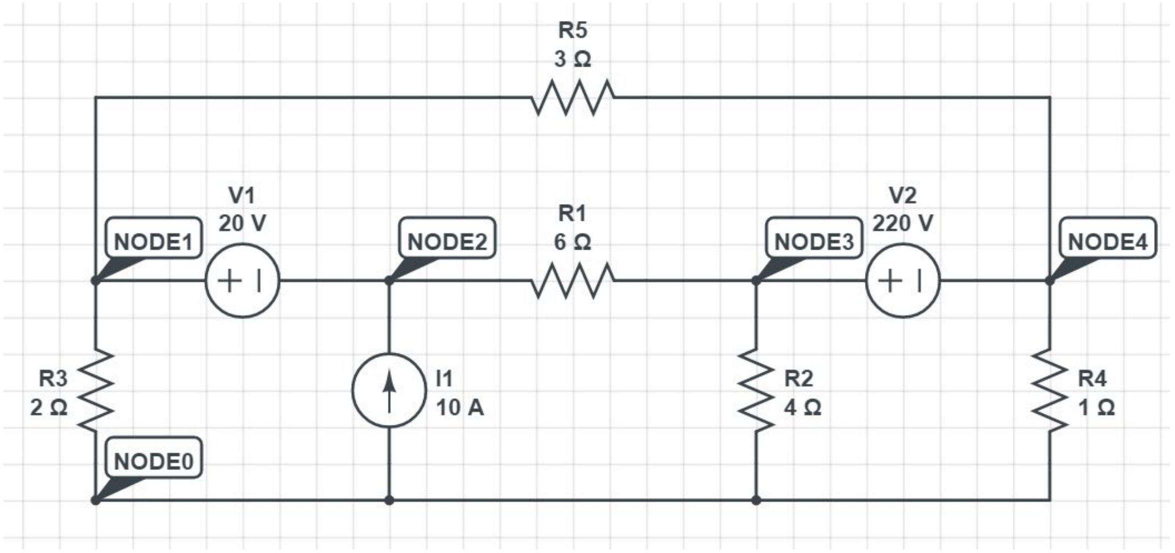
User Input:

1. Voltage Source, 10V, A-1, B-0
2. Voltage Source, 2V, A-3, B-1
3. Current Source, 2A, A-2, B-0
4. Resistor, 4Ω, A-1, B-2
5. Resistor, 8Ω, A-3, B-2
6. Resistor, 3Ω, A-0, B-3

VGA Diagram:



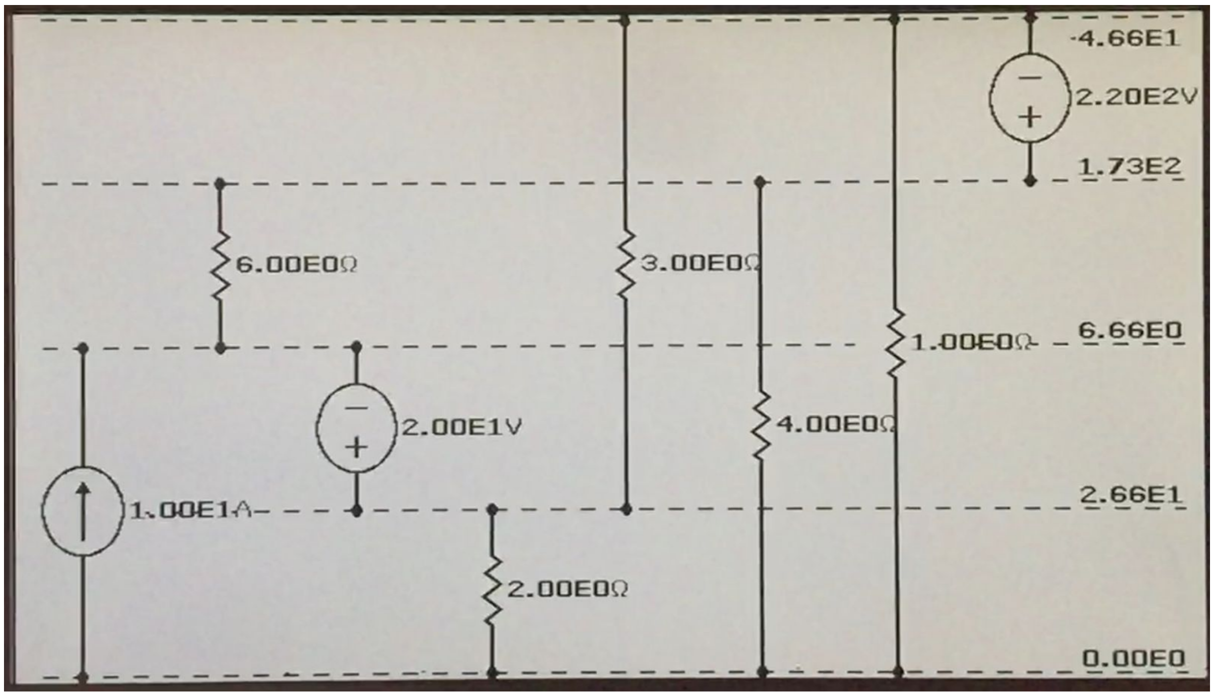
Example 2: Solving a circuit with all three elements + 2 supernodes (Node 1, 2) & (Node3, 4)



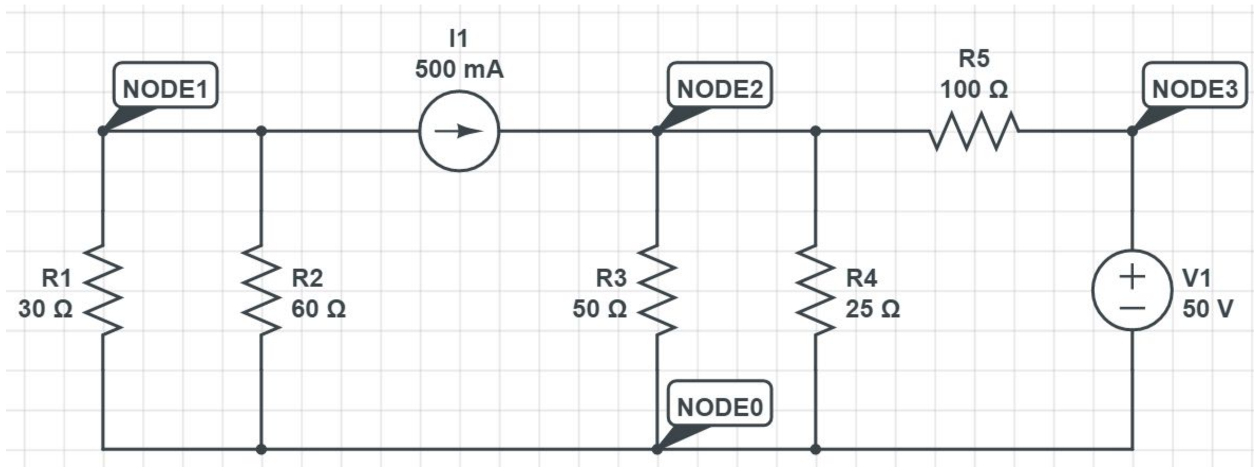
User Input:

1. Current Source, 10A, A-2, B-0
2. Resistor, 6Ω, A-2, B-3
3. Voltage Source, 20V, A-1, B-2
4. Resistor, 2Ω, A-1, B-0
5. Resistor, 3Ω, A-1, B-4
6. Resistor, 4Ω, A-3, B-0
7. Resistor, 1Ω, A-0, B-4
8. Voltage Source, 220V, A-3, B-4

VGA Diagram:

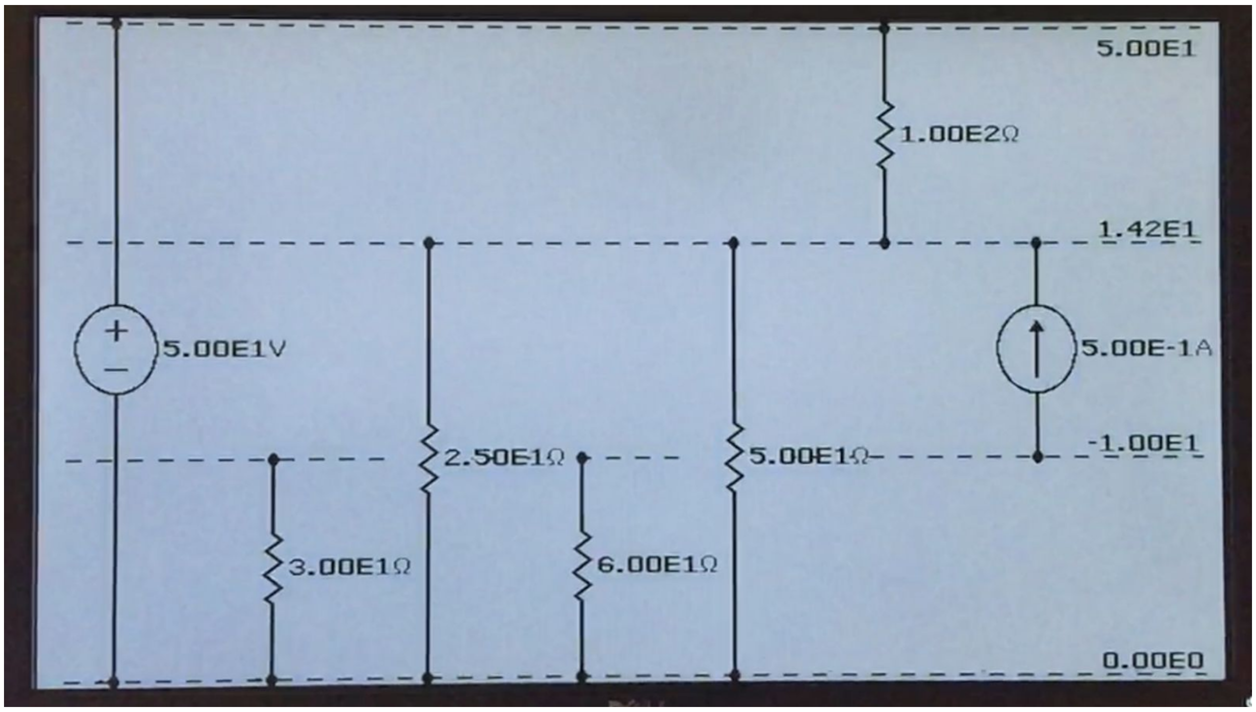


Example 3: Solving a circuit with all three elements + 1 supernode (Node 0, 3)



User Input:

1. Voltage Source, 50V, A-3, B-0
2. Resistor, 30Ω, A-1, B-0
3. Resistor, 25Ω, A-2, B-0
4. Resistor, 60Ω, A-1, B-0
5. Resistor, 50Ω, A-2, B-0
6. Resistor, 100Ω, A-2, B-3
7. Current Source, 500mA, A-2, B-1



Appendix C: All major RAMs' architectures

RAM: draw_processor

- Stores the encoded command lines that are executed to draw each shape/sprite on the VGA display in a sequential manner
- 47-bit wide, 1024 words, 47x1024 (can handle 1024 commands)
- bit[47]: 1-do this command; 0-skip this command
- bit [46:38]: the code for the sprite/shape being drawn
- bit [37:29]: height of the shape
- bit [28:19]: width of the shape
- bit [18:10]: topmost position (the y_position where the drawing starts)
- bit [9:0]: leftmost position (the x_position where the drawing starts)
- Here is the list of that explains the correspondence between command codes and shapes
    - 9'b111111111: clear screen(white) (currently disabled/bit[47]=0)
    - 9'd0: dashed node lines
    - 9'd1: straight wires connecting different nodes
    - 9'd2: upright voltage source
    - 9'd3: upright current source
    - 9'd4: upright resistor element
    - 9'd5: node connection dot
    - 9'd6: mouse cursor (currently disabled/bit[47]=0)
    - 9'd7: minus sign '-'
    - 9'd8: exponent sign 'E'
    - 9'd9: decimal point '.'
    - 9'd10~9'd19: digits '0'~'9'
    - 9'd20: inverted voltage source
    - 9'd21: inverted current source
    - 9'd22: inverted resistor element
    - 9'd23: volt unit sign 'V'
    - 9'd24: ampere unit sign 'A'
    - 9'd25: ohm unit sign 'Ω'

RAM: element

- Stores basic information about each element according the user's input
- 32-bit wide, 32 words, 32x32
- bit [31:27]: node A number
- bit [26:22]: node B number
- bit [21:20]: element type (00-Voltage Source, 01-Current Source, 10-Resistor, 11-NULL)
- bit [19:10]: element exponent
- bit [9:0]: element value

RAM: float_register

- Stores the element value that is converted to the floating-point form
- 32-bit wide, 32 words, 32x32
- bit [31:0]: the floating point number
- Note: Resistor values are stored as their reciprocals, aka conductance

RAM: element_decimal

- Stores the decomposed decimal-digit form of the element value
- 16-bit wide, 32 words, 16x32
- bit [15:12]: second number after the decimal point
- bit [11:8]: first number after the decimal point
- bit [7:4]: single number before the decimal point
- bit [3:0]: the signed exponent (ranges from -8 to 7)

RAM: nodeHeads

- Stores the information about each node. Also serves as the head of the node linked list since it stores the address of the first element in the linked list that corresponds to this node
- 64-bit wide, 32 words, 64x32
- bit [63]: 0-not builded, 1-builded
- bit [62:50]: unused
- bit [51:47]: current `nodeToElement` address for searching/traversing
- bit [46:42]: address of the first element in linked list `nodeToElement`
- bit [41:37]: index of the corresponding node row in the system of linear equations
- bit [36:32]: address of the reference node in `nodeHeads`
- bit [31:0]: voltage value difference with respect to the ref node (in floating-point)

RAM: nodeToElement

- Stores the linked list of each node. Used to search supernodes and generate matrix equations
- 64-bit wide, 32 words
- bit [63]: 0-is not the end of the list; 1-is the end of the list
- bit [62:58]: address of the next element
- bit [57:45]: unused
- bit [44]: the current node is the node 0-A/1-B of this element
- bit [43:39]: address of the other node of the element
- bit [38:34]: address of the current node (the one that corresponds to this list)
- bit [33:32]: element type (00-Voltage Source, 01-Current Source, 10-Resistor, 11-NULL)
- bit [31:0]: element value in floating-point form

RAM: float_matrix

- Stores the system of linear equations to be solved by the Matrix Solver module.
- 31-bit wide, 4096 words, 31x4096
- The largest number of equations that can be solved is 11. So the circuit allows at most 11 reference nodes.
- bit [31:0]: stores the floating-point value

RAM: nodeVoltage

- Stores the decomposed decimal digit form of the calculated values of each node for display
- 32-bit wide, 32 words, 32x32
- bit [31:28]: fifth number after the decimal point
- bit [27:24]: fourth number after the decimal point
- bit [23:20]: third number after the decimal point
- bit [19:16]: second number after the decimal point
- bit [15:12]: first number after the decimal point
- bit [11:8]: single number before the decimal point
- bit [7:4]: the exponent value (0~7)
- bit [3:0]: the sign of the exponent (4'b0000-positive, 4'b1111-negative)

RAM: nodeSeq

- Stores the sequence of nodes that appear on the VGA display (from bottom to top)
- 5-bit wide, 32 words, 5x32
- bit[4:0]: the node number in the sequence

RAM: elementSeq

- Stores the sequence of elements that appear on the VGA display (from left to right)
- 5-bit wide, 32 words, 5x32
- bit[4:0]: the element number in the sequence