2022-05-01

Development of an Automated Electronic Prototyping System

Cesar Yahir Sanchez Zambrano
The University of Texas at El Paso

DEVELOPMENT OF AN AUTOMATED ELECTRONIC PROTOTYPING SYSTEM

CESAR YAHIR SANCHEZ ZAMBRANO

Master's Program in Electrical Engineering

APPROVED:

David Zubia, Ph.D., Chair

Hector Erives-Contreras, Ph.D.

Eric MacDonald, Ph.D.

Stephen L. Crites, Jr., Ph.D.
Dean of the Graduate School

Dedication

I want to dedicate this work to my family, my loved one and Dr. David Zubia for all their support, dedication, patience and advice during this journey. Without your help this would not have been possible. Thank you.

DEVELOPMENT OF AN AUTOMATED ELECTRONIC PROTOTYPING SYSTEM

BY

CESAR YAHIR SANCHEZ ZAMBRANO, B.S

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfilment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

May 2022

Acknowledgements

I acknowledge the Texas Instruments Foundation Professorship, Professor David Zubia, Professor Hector Erives-Contreras, Eunice Tobias, Robby Ramirez, Angela Mendez, the department of Electrical and Computer Engineering and lastly The University of Texas at El Paso that made this work achievable.

Abstract

Prototyping systems with interconnected components can be a time and resource expensive process. The process consists of three main phases (design, build and analysis) with each having their own associated cost. For the case of electronic circuits, the building phase is the costliest phase among the three, being prone to human errors which causes the circuit to fail. All three phases of the prototyping process are important. However, often a disproportionate amount of time is spent on the build phase due to the difficulty of making and troubleshooting circuits by hand. In this thesis we will discuss a system that delivers students a fast and reliable method to prototype real electronic circuits in a personal laboratory. This system uses a modular hardware architecture that can interconnect electronic components automatically using a developed software. The circuit building system demonstrated that the building phase of a circuit took 17% of the total time spent on the entire prototyping process of such circuit. The automation of the building phase allows users to balance their time between the different phases of prototyping including design, build, and analysis.

Table of Contents

List of tables

List of figures

1. Introduction and Background

Prototyping systems that are made of interconnected components is a complex process that can be time and resource consuming. The process consists of three distinct phases as shown in Figure 1. Each of the three phases (design, build, and analysis) have their own associated costs. For the case of electronic circuits, the build phase is usually the most time and resource consuming due to the manual aspect of making circuits. This phase is prone to human error which can cause the prototype circuit to fail. Addressing these mistakes is often a challenge that is proportional to the complexity of the circuit. As the size of the circuit increases, the difficulty in finding and correcting errors also increases.

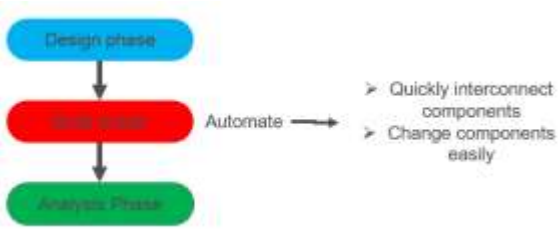


Figure 1 Prototyping process and how will the possible solution to reduce the cost of the build phase

Learning how to prototype electronic circuits is an important learning objective in electrical engineering academic programs. All three phases of the prototyping process are important. However, often a disproportionate amount of time is spent on the build phase due to the difficulty of making and troubleshooting circuits by hand. As a result, insufficient time is spent on circuit design and testing where much more knowledge can be gained about circuitry.

The seminal dissertation by Atalbe [1] analyzed student perspectives on virtual electronic laboratories and listed eleven key design guidelines for laboratories. The guidelines were based on the deployment of a simulation laboratory tool to gain insight into key features for diverse types of virtual laboratories. The guidelines are listed below: [1]

1. “Enable sharing of knowledge and real-time feedback”
2. “Enable options for individualized learning and group scheduling”
3. “Provide consistent and useful responses to errors”
4. “Provide access to tutors, preferably in real-time"
5. “Provide additional online help, in the form of tutorials and/or videos”
6. “Provide realism in the system”
7. “Ensure that the virtual laboratory supports learning in the physical laboratory”
8. “Involve students in the design from the beginning”
9. “Explicitly consider the desired learning objectives in the virtual laboratory design”
10. “Provide a user-interface that is intuitive, simple and easy to use, as well as easy to learn”
11. “Provide for speed and reliability of the system”

1.1 Available Systems

A brief survey is presented of physical electronic laboratory systems that are readily available. A common aspect of the systems is that they facilitate the building phase process by automating the wiring of a circuit or having a pre-built circuit with adjustable parameters.

1.1.1 Virtual Instrument System in Reality (VISIR)

VISIR is a remote system developed by the Blekinge technology institute in Sweden [2]. The system’s architecture allows users to reconfigure, build, and test physical circuits remotely. One deployment of the VISIR lab is at Universidad Estatal a Distancia in Costa Rica [3]. VISIR offers its users the ability to remotely connect to a central board shown in Figure 2(a) to start prototyping “basic analog experiments”. Users design and test circuits through a graphical interface shown in Figure 2(b). A strength of the system is that real circuits are automatically built

and tested. However, the system can support only a limited number of users at a time. If the system is saturated, users will need to wait until the system is available. Although this system is a great approach to automate the building phase of a circuit, another limitation is that the centralized component tower has a fixed set of hardware components that users cannot easily configure to their needs.

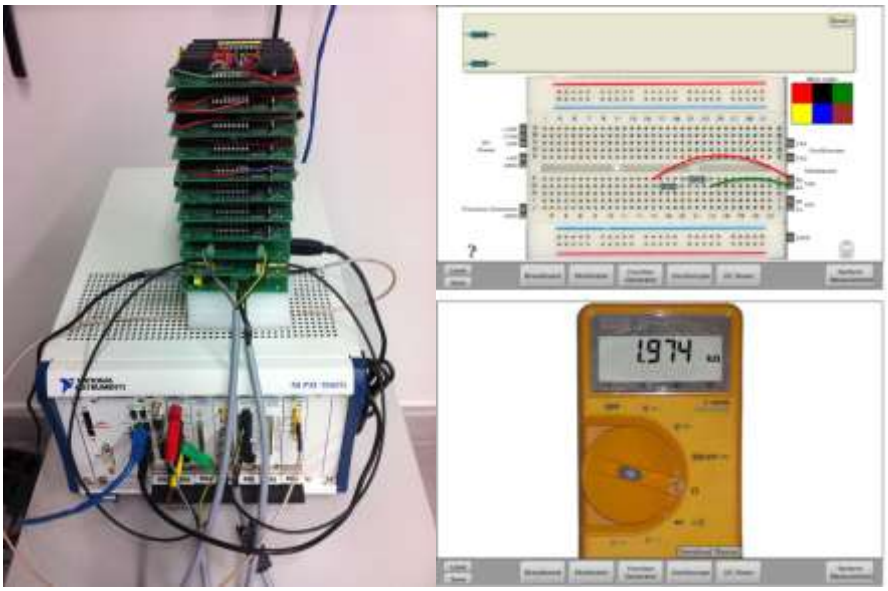


Figure 2 (a) VISIR central system. (b) VISIR user interface [3] This system is a great approach to automate the building phase of a circuit although their centralized component tower has a fix set of components and user cannot adjust the hardware configuration to their needs.

1.1.2 Analog Electronics Lab (AELabs) for NI Elvis

The Analog Electronics Lab developed by Quanser is another prototyping system that allows a fast connection between modules [4]. This system provides a set of cards with already built-in circuits which can connect one to another using a central board called “interface board” as shown in Figure 3. The system allows users to control the value of component parameters such as resistance, capacitance, and inductance using a graphical interface in a computer system. The system also provides several holes for probing different points of the circuit with measuring

instruments to study its response to changes. This system is a great option to prove basic concepts for analog circuits. However, it severely limits the degree of freedom from users to design their own circuits. User can only use the built-in circuits and cannot design their own circuit topologies.

Figure 3 Analog Electronics Lab (AELabs) for NI Elvis [4] This system is a great option to prove basic concepts for analog circuits although it leaves a tiny degree of freedom to the user meaning that user cannot design their own prototype or circuit configuration to study.

1.2 Contribution of This Thesis

This thesis focused on a software development and describes the algorithms to drive a modular hardware system following a procedure I composed to describe and track the hardware configuration. The motivation of combining hardware and software is to create a personal and reconfigurable electronic laboratory [5] using Altalbe design guidelines [1] to achieve key features in a real prototyping laboratory. This thesis describes the development of a major process needed to convert a SPICE netlist into a real circuit.

2. Overall System Approach

2.1 Netlist Representation of a Circuit

A typical way to represent a circuit is through a schematic diagram as shown in Figure 4(a) and many software circuit simulators use the schematic representation to visually show the details of a circuit. However, internally a computer uses a file called a "netlist" to specify the details of a circuit as shown in Figure 4(b). The netlist is a standard text representation of a circuit that a compiler can parse and translate all information into a series of 0's and 1's so a computer can process any required calculation. For example, the netlist shown in Figure 4(b) contains all necessary information regarding how to build the respective circuit shown in Figure 4(a). Each component of a circuit is divided into blocks and each block stores all the respective information about the component. The amount of information stored varies depending on the component category. In this thesis we divide components into three main categories: "generic", "specific", and "net element". We will discuss in more depth the component categorization and classification in later chapters of this thesis.

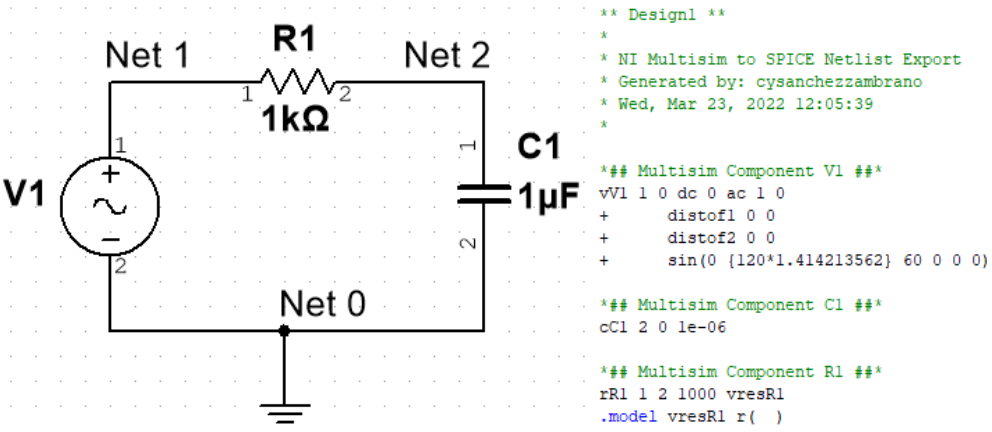


Figure 4. (a) Circuit schematic showing each component pin number and its respective (b) netlist specification.

2.2 Matrix Representation of a Circuit

An electrical circuit can also be represented as a matrix as shown in Figure 5, where the *"o's"* in the matrix cells represent electrical connections between the components of circuit. The *"x's"* represents self-connections. This type of representation facilitates arbitrarily manipulating the connections between components with a digital computer. For example, the interconnection between the components can be easily manipulated by changing the *"o's"* within the matrix as shown in Figure 5(b). This type of matrix representation emphasizes how components terminals connect one to another by having an identical set of components in both the rows and columns of the matrix and deemphasizes the nodes of the circuit. In this matrix, a connection between two terminals is expressed with an 'o' in the matrix cell that is common to both terminals. Notice that there is no need to represent any connection in the diagonal of the matrix since by definition "every terminal is connected to itself" [6]. Since the same components are used in rows and columns, the connections are symmetric above and below the major diagonal. This type of circuit matrix representation is called a component-to-component relation.

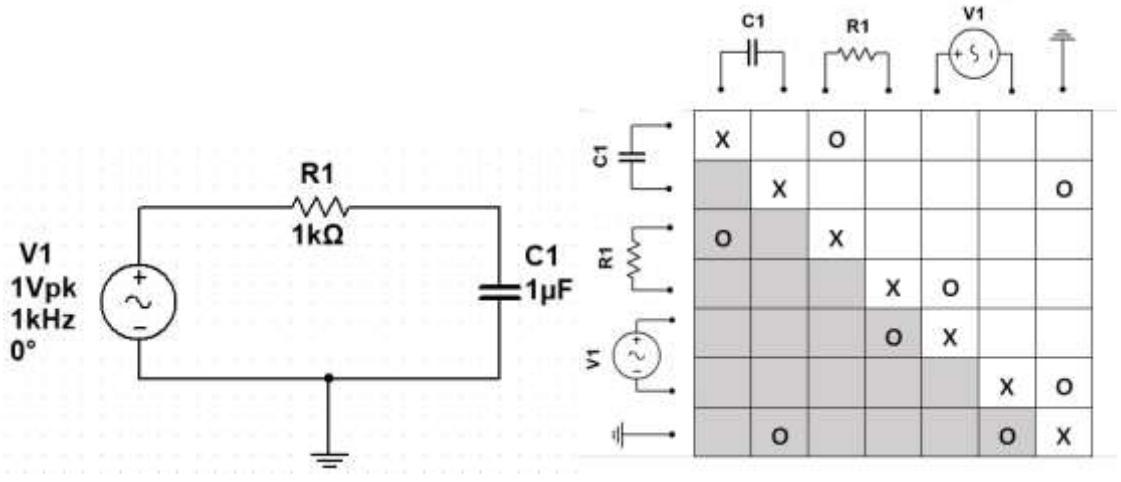
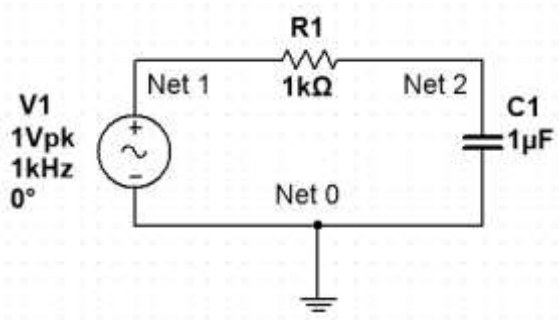


Figure 5 (a) High-pass filter circuit schematic. (b) Component-to-component relation matrix representation of circuit A

Another method for specifying a set of connections in a circuit is a component terminal-to-net relation as shown in Figure 6. In this matrix representation, the columns of the matrix represent a set of nets while components terminals are represented by the rows. In this case, every 'o' represents a component terminal connection to a net. Therefore, having two or more component terminals connected to the same net is equivalent to having them connected to each other. This allows us to quickly map any circuit schematic into a matrix.



			Nets			
			N0	N1	N2	N3
Components		GND	o			
	V1	V1+		o		
		V1-	o			
	R1	R1.1		o		
		R1.2			o	
	C1	C1.1			o	
		C1.2	o			

Figure 6 (a) High-pass filter circuit schematic. (b) Component terminal-to-net relation matrix representing circuit A.

2.3 Implementation of Matrix via Cross-point Analog Switch Array

One of the advantages of the component terminal-to-net matrix shown in Figure 6 is that it can be easily implemented with hardware using a cross-point analog switch array as shown in Figure 7. A cross-point analog switch array is a programmable analog array that has a set of vertical and horizontal lines (or wires) that can be arbitrarily interconnected. If components are physically connected to the horizontal lines and the vertical lines are interpreted as nets as shown in Figure 7, then the components can be interconnected to physically implement a circuit. For example, the RC circuit shown in Figure 6 (a) can be physically produced by interconnecting the cross-points in the analog array corresponding to the cells marked 'o' elements in Figure 6 (b). It is noted that

the use of analog arrays grants the system the ability to create any desired connection and can therefore produce many different circuit configurations.

However, to automate the build process of a circuit so that the interconnections are achieved without human interaction, software and a microcontroller also are needed. Software is needed to translate a circuit design into a set of addresses on the cross-point analog switch array. A microcontroller is needed to program the cross-point analog switch array with the appropriate input/output signals and the addresses created by the software. Together, the cross-point analog switch array with the software and microcontroller comprises a circuit building system.

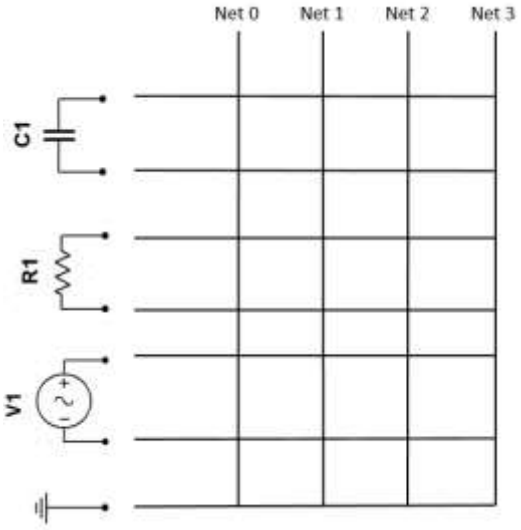


Figure 7 Cross-point analog switch capable of interconnecting components

2.4 Modularization of Circuit Building System

The circuit-building matrix shown in Figure 6 has several drawbacks. One is that the class of circuits that can be built is limited to the components that are hardwired to the cross-point analog switch array. Another is that the number of components is constraint to the number of ports

available on the cross-point analog switch array. This section describes how these limitations can be overcome by modularizing the circuit building system.

The main idea of modularizing the hardware is to divide it into three major modules that can be connected as shown in Figure 8. One module contains solely the components and is called the "component card". Another module is composed of the cross-point analog switch array and is called the "port board". The third module is a microcontroller. As shown in Figure 8, each of the modules are detachable. To make a circuit, the microcontroller programs the port board to make the required component interconnections for the circuit.

Software is needed to send instructions to the microcontroller and is the focus for this thesis. The software combines information about the hardware and circuit and sends instructions to the microcontroller. The software will be described in detail in the chapters below.

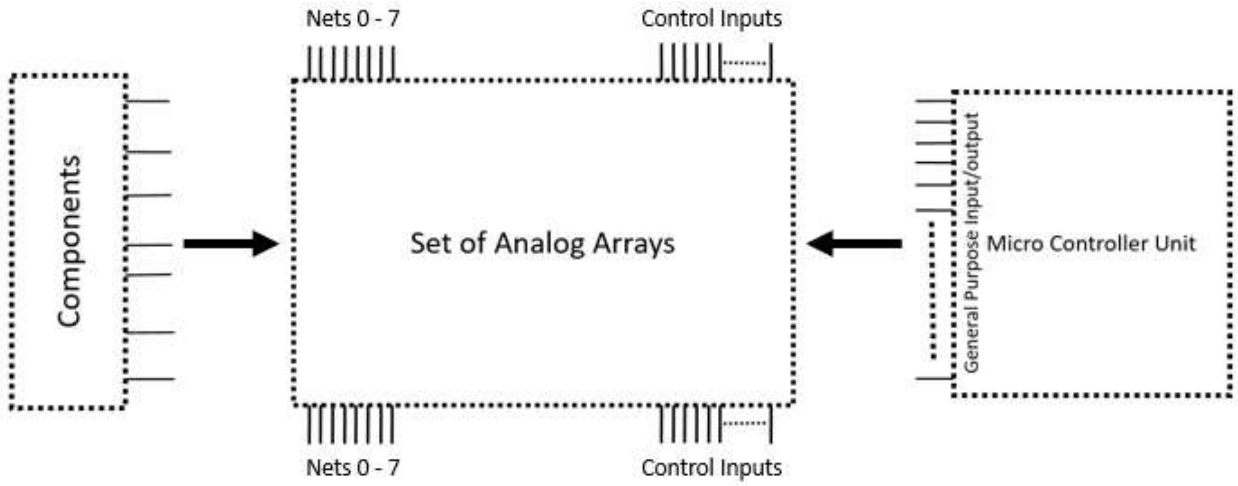


Figure 8 Collection of modules comprehending a section of a modular hardware architecture for an Automated Circuit Build System

One benefit of this modular architecture is that it allows components to be interchanged simply by exchanging a component card with another set of components. Another benefit is that it allows the number of components and size of the cross-point analog switch array (number of port

boards) to be expanded as shown in Figure 9. Overall, the benefits of this modular architecture allow for a wider variety and higher complexity of circuits to be built. Reed's Law can be used to estimate the number of ways in which a set of components can be interconnected [7]. In a system with forty, 2-terminal elements, there are 2^{40} = one trillion configurations possible! In general Reed's Law states that the number of configurations is given by

Equation 1 Reed's Law

$$c = n^y$$

where c is the number of configurations, n is the number of terminals of the components, and y is the number of components. The variety increases hyper-exponentially due to the combinatorial manner in which the components can be interconnected. In conclusion, a modular architecture allows the hardware to be easily changed or expanded according to user needs. The role of each module is described in more detail in the following sections.

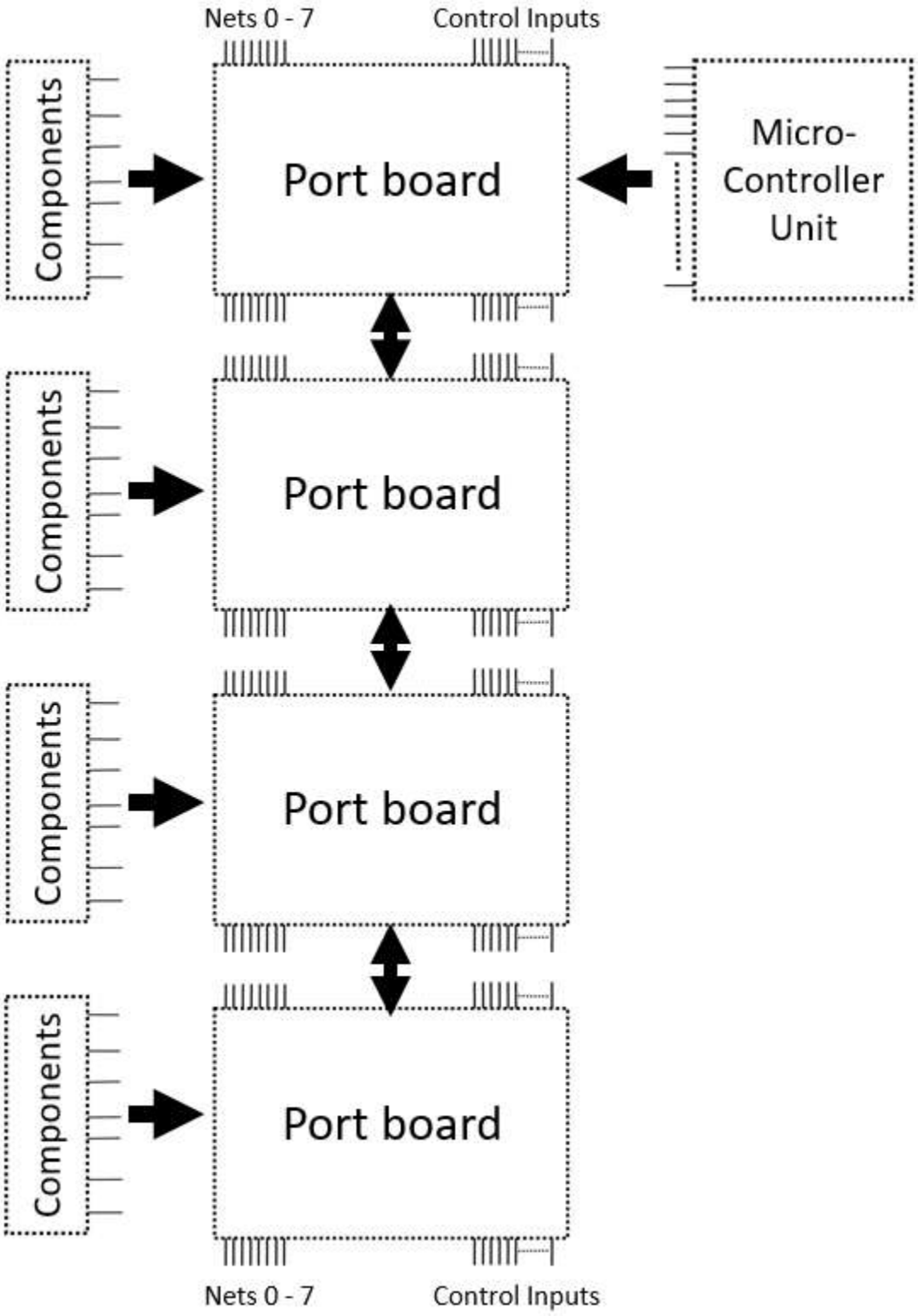


Figure 9 Automated circuit build system hardware architecthre

2.4.1 Component Card

A component card contains a set of components that can be used to physically implement a wide variety of circuits. A component card can be thought of as a “bag of components” that can be used to make a variety of circuits. Although the variety of circuits that can be made is constrained by the set of components, the variety can be easily changed or expanded since the component cards are detachable. In other words, the set of components can be easily exchanged and/or expanded by connecting a component card with different set of components and/or connecting more cards. This allows different and more complex circuits to be built.

The components are physically connected to a multi-pin connector but not to each other as shown in Figure 10. The terminals of each component are connected to unique pins on the connector. In turn, the pins of the connector correspond to the horizontal lines or “ports” on the port board. In this way the components are made available for interconnection using the port board.

The multi-pin connector may also be made available to connect electrical instruments such as power sources and meters. A connector can be placed and routed to the multi-pin connector depending on the type of electrical element that is going to be placed in the component card.

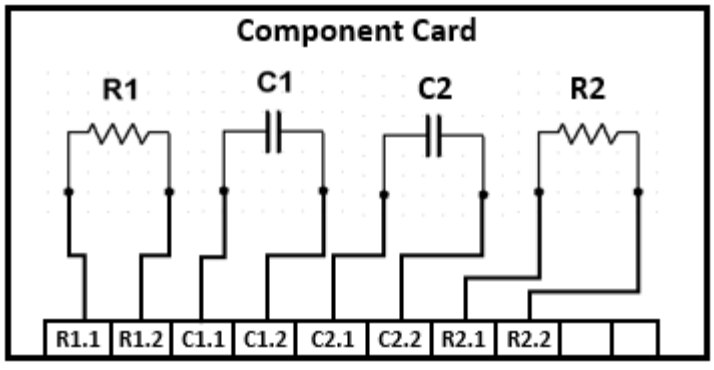


Figure 10 Component card routing concept

2.4.2 Port Board

A port board is a cross-point analog switch array that is used to interconnect components on component cards. The port board is a central board of the modular circuit building system and in cooperation with a microcontroller and component cards is used to make the interconnections needed to create physical circuits as shown in Figure 9. Essentially, the port board has three roles. One role is to provide ports so that components can be connected. The second role is to provide electrical lines that can be used as nets for a circuit. The third role is to interconnect the components as needed to make a circuit based on instructions from a microcontroller.

Two advantages of the modular architecture of the circuit building system shown in Figure 9 are that its configuration can be easily changed and expanded. For example, the system can be expanded by connecting two or more port boards together and connecting different component cards to each port board. This gives a circuit designer flexibility to work on a wide variety of circuits. However, the physical configuration of the system including the number of ports, the list of components and their location on each port board must be precisely accounted for. Figure 11 shows how the number of ports doubles when two port boards with each 64 ports and 16 lines are connected to make a larger system with 128 ports and 16 lines. In the larger system, different component cards can be attached to each port board. All these configuration changes must be precisely described and recorded. This is accomplished through software with the help of a file called “Portlist” and is discussed in section 2.5 and Chapter 3.

The port board distributes power among the cross-point analog switches that comes from any external source such as a microcontroller unit or any other power source. The power lines are also distributed among the any other port boards connected together.

Each port board contains a set of switches to assign the port board a unique position in the system. The port board can select among 4 distinct positions using two, two-state switches working as binary selectors to denote their position. For example, 00 in these switches will mark the first position or that the port board is the first port board in the system and a 11 will mark the last position as shown in Table 1.

Table 1 Board selection truth table

Switch 1	Switch 2	Board
0	0	Board 1
0	1	Board 2
1	0	Board 3
1	1	Board 4

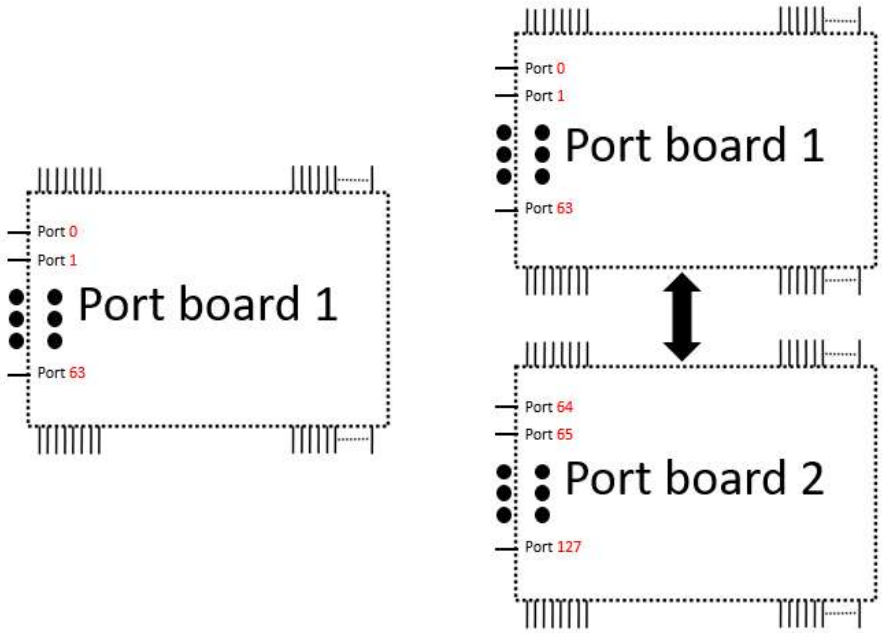


Figure 11 . (a) System using only one port board having 64 ports. (a) System using two port boards having 128 ports.

2.4.3 Role of Microcontroller Unit

The role of the microcontroller unit is to program the cross-point analog switch arrays contained in the port boards. The microcontroller receives instructions from a software program

that contains information about which analog switches needed to be turned on and details about the timing sequence to execute the programming. To accomplish the programming, an addressing scheme is used to identify each of the cross-point switches within the whole circuit building system. The addressing scheme used in this work consists of a 12-bit address stream as shown in *Table 2*. The address is used to target a port and line corresponding to a cross-point. A total of 8 bits are used to address each port on the circuit building system. Two of the eight bits are used to select a port board. The remaining six bits are used to address ports on each port board. This gives a total of 256 physical ports available for components. The system has 16 physical lines that are addressed using 4 bits. The microcontroller receives a stream of 12-bit addresses from software and uses it to turn on all the cross-point switches needed to make a desired circuit.

Table 2. 12-bit address stream

4-bits Lines				2-bits Board select		6-bits Ports					
CS2	AY2	AY1	AY0	BE1	BE0	CS1	CS0	AX3	AX2	AX1	AX0

2.5 Role of Software

The overall role of the software is to combine information about the hardware and circuit design and create instructions for the microcontroller to make the circuit physically as shown in Figure 12. In other words, the task of software is to create a “mapping” from a circuit schematic to a set of addresses for the analog-switches needed to make the circuit. However, to accomplish this task several subtasks need to be performed.

First, the software needs to recognize the components present in the component cards and all their corresponding ports on the port board. The software must recognize the physical configuration of the hardware. This is done through a file called the “Portlist” which is shown in

Figure 12. The Portlist's key role is to list and specify the components present on each component card with their respective port numbers. In other words, the Portlist specifies what components are physically present on the component cards and at which ports they are located. As the name implies, the Portlist acts similar to a netlist because it contains a list components. The difference is that the Portlist specifies the ports, in contrast to nets, to which the components are connected. The Portlist can be thought of as a generalized description of the hardware. In this subtask, the software enters information from the Portlist into a relational database file that is used in other subtasks. The relational database is shown in Figure 12.

The second subtask is to determine whether the creation of a circuit is physically viable for a given Portlist and netlist. A relation between the Portlist and netlist is performed to determine; (1) what is needed, (2) is it available, (3) where it is located and (4) where it will be connected. This relation is performed using logic control and stored in the relational database. This is the first stage of viability; knowing if the circuit can be physically built.

The third subtask is to perform a design rule check to protect the system from damage. The purpose of the rules is to avoid physical damage to the hardware. The output of this rule check is either a pass or fail. The software will determine how many and which design rule checks are performed depending on the type of components required by the netlist. The tests are performed using conditional statements; for example, if a power line is directly connected to a ground line the test will raise a "fail" flag marking the circuit as an invalid circuit to build. Similar to the previous subtasks, the output from the test is written to the relational database. This is the second stage of viability; knowing if the circuit is safe to build.

Finally, once it is determined that it is possible to build a circuit in a safe manner, a translation is performed from the netlist required components, connections, and locations to an

address stream in accordance with the address programing scheme of the hardware. Address streams are arranged as a set with a specific order and timing to ensure that each cross-point analog array switch is targeted properly, and the circuit is guaranteed to be built.

Throughout the entire process, information is written into and read from the relational database with all the necessary information. This makes accessing information fast and easy at any stage of the process. This information is crucial since all component port locations, line connections, address streams, design rule check results and finally circuit-build viability are stored within this database. At the same time, this database facilitates the comparison process through different conditional logic control tests, reducing the number of instructions needed per stage.

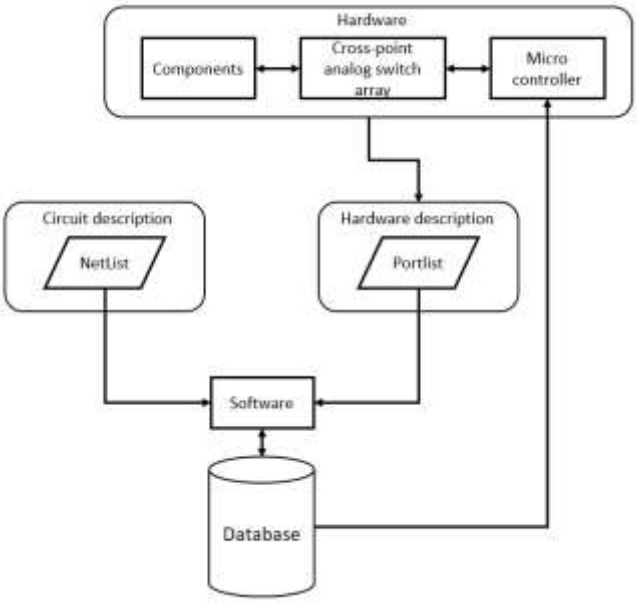


Figure 12 Overall role of the software

3. Method to Describe and Track Hardware Configuration

Because the circuit building system is modular in the sense that it can be easily changed and expanded, a method is needed to conveniently describe and keep track of the hardware configuration. For example, if a component card is exchanged with another card that has a different

set of components, a method is needed to take this change into account. In general, a method is needed to describe the configuration of the hardware including the name, type, part number, value and corresponding port of each component attached to the port board(s).

The method should allow the software to identify what is physically present and where it is located. It should also store or keep the configuration information within the software's reach. There are two complementary methods to achieve this. One method is to use a (2 – 4 KB) variable that can be read anytime as required. However, a disadvantage is that all variables are deleted and "dumped" at the software's end process i.e., when the main window of a program is closed. This causes the hardware configuration information to be lost and requires a time consuming "set up" process each time the program is started.

An alternative solution is to use a file stored in nonvolatile memory that can be accessed to obtain the hardware configuration. This allows a single "configuration set up" if there are no changes in the hardware configuration. The downside to this method is that parsing through a file several times, even if it is a text file, can saturate the reader memory making the process slower with long periods of usage. Thus, a program reset may be required to free memory and start over. Table 3 lists the advantages and disadvantages of the two methods. The main advantage of a nonvolatile memory file is that the information can be stored and modified. On the other hand, the main advantage of a software variable is that the information can be accessed very quickly and modified on the fly.

The approach taken in this work is to combine both methods to make use of their complementary advantages. Essentially this is accomplished by storing a description of the hardware configuration into a text file similar in format to a netlist. Then the information in the text file is read and translated into a software variable.

The sections below narrate how the text file is used to describe and keep track of the details of the hardware configuration. The information in the text file can be categorized into different levels. At the lowest level, a method to describe individual components is presented. This is followed by a method to describe component cards. The highest level is a method to describe the set of components in the whole system.

Table 3 Comparison between two methods of storing hardware configuration information for software use.

Software Variable		Memory stored file	
Advantages	Disadvantages	Advantages	Disadvantages
• Fast memory accessing • Information can be modified on the fly	• Volatile memory • Max allocated memory may saturate depending on information size	• Non-volatile memory • File size can be as big as the free ROM memory	• Higher CPU process required at long periods • File must be copied into local memory to be modified

3.1 Description of Individual Components – Redefine Nets as Ports

This section explains the method used to describe hardware at the individual component level. The method is similar to the format used in netlists. At the component level, a netlist uses blocks of text in a structured way to describe details about individual components and how they are interconnected to make a circuit as shown at the top of Figure 13. One aspect of the block describes the electronic characteristics of the component. Another aspect specifies the net that each terminal of a component is connected to. The method in this work is like the netlist but changes the meaning of *net* to *port* as shown at the bottom of Figure 13. In other words, by changing the significance of terminal-related elements for each component from net to port, a new information block is obtained that contains both the electronic characteristics of each component and the ports that its terminals are connected to as shown in Figure 13. Unlike nets, ports can be used only once.

Although this change in meaning appears minor, it is a key aspect to modularizing the circuit builder. For example, this arrangement can be used to produce a file that will serve as a resource file to identify the electronic components and their port locations on a respective component card.

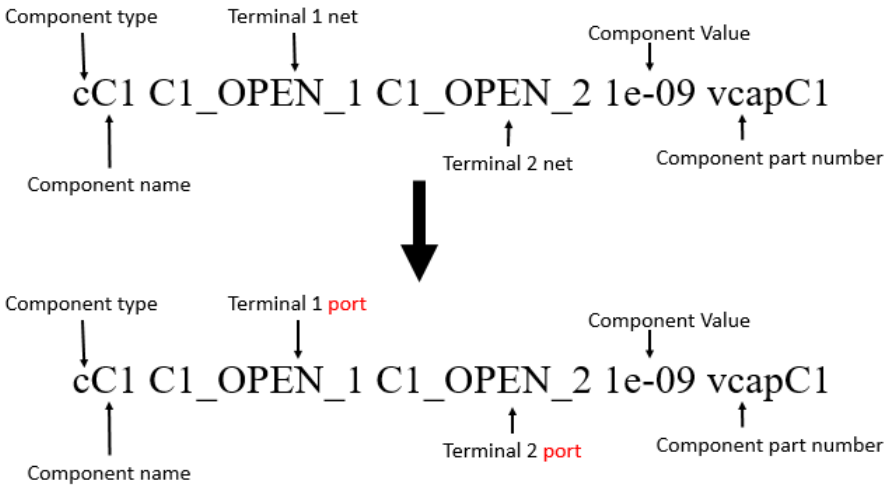


Figure 13. Changing the meaning of the terminal-related elements from net to port.

3.2 Component Card Resource File

A feature of the circuit building system is the ability to easily interchange component cards. However, for the circuit building system to operate correctly it is important to ensure that the information in the software matches the configuration of the physical resources. Changes in the hardware must be accurately tracked in the software. This requires a validation step to ensure that the information in the software matches the hardware configuration. At the component card level if the components are fixed or hardwired, a read-only resource file can be used to describe all the components on the card as shown in Figure 14. The resource file can be in the form of a text file in a ROM chip on the component card. This will ensure that the correct resource file is associated with each component card.

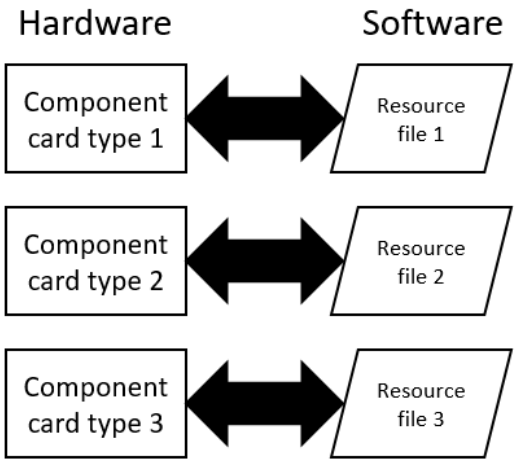


Figure 14 Component card type and corresponding resource file.

3.3 Portlist

When component cards are exchanged or multiple component cards are used in a circuit building system, a method is required to keep track of all components and their unique port locations connected to the system via the component cards. This is achieved using a non-volatile information file called the “Portlist” which contains information about the type, name, terminal port locations, value parameter, and part number of all components. The Portlist is a hardware description of the circuit building system. It is important that the information in the Portlist be validated to ensure it matches the physical configuration of the system. An advantage of using a Portlist is that it be saved as a text file and reused without having to create it or validate it each time. However, if there is a physical change in the hardware (meaning a component card is replaced with a different component card) a new configuration/validation process is required to ensure a match between the Portlist information and the new hardware configuration.

When configuring a circuit building system it is possible to use the same type of component card multiple times as shown in Figure 15. This means that identical sets of components will be connected to the system via the port boards. However, each component is still required to be

connected to unique ports and therefore assigned unique ports in the Portlist. The scheme to accomplish this is described next.

Each component card allocates a total of 64 ports for component terminals which are connected to a port board. Two port boards connected one to another may accept two component cards of the same type. This means that in software to recognize these two component cards the same resource file is used, but to properly adapt the port routing to the corresponding port board we use

Equation 2 Port board-port number equation

$$PBN = \big((PB - 1) * 64\big) + CCP$$

were "PBN" indicates the port number of each port board; "PB" represents the port board number and "CCP" the component card port number. We can observe in Figure 15 how resistor "R1" terminals belong to port 0 and port 1 inside two component cards but, when a component card is connected to port board 2 the resistor "R1" terminals port numbers turn to be the port 64 and 65. This differentiation helps when building the Portlist file to remark which component cards belong to which port board and route their corresponding port number in accordance with the port board they are attached to. Once this distinction is made, we can append all resource files for all required component cards to create a Portlist.

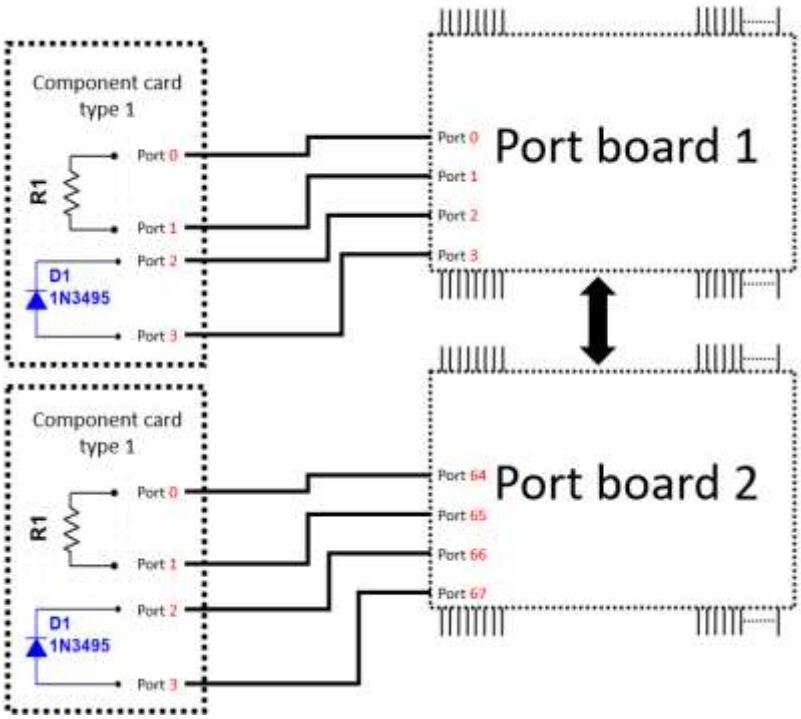


Figure 15 Component Card port numbering vs Port board port numbering

Opposite to a net, which are repeating elements, ports are unique elements. Therefore, each terminal related element in a Portlist must not be repeated through the entire Portlist as we can observe in Figure 16.

```
rR1 0 1 2200 vres
dD1 2 3 D1N4002RL/ON__DIODE__1
rR1 64 65 2200 vres
dD1 66 67 D1N4002RL/ON__DIODE__1
```

Figure 16 Section of a Portlist denoting port location for Figure 15

4. Component Database

The Portlist file is a convenient method to describe and store the hardware configuration and track changes. However, the Portlist is an unarranged information file that uses information blocks that are simply a string of characters. To make the best use of the information in the Portlist, the software rearranges and divides each information block into its essential information elements, creates a structured variable called “database”, and then inputs the element information into the database. A graphical representation of the structured variable is shown in Figure 17. The single variable is structured into three levels of information denoted: (1) component, (2) component parameters and (3) safety parameters. Essentially these levels provide information about the number of components and their electronic characteristics, their location within the system, how they are to be interconnected, and whether it is safe to interconnect them. These information levels can be used to manipulate the arrangement of the database and thus reduce the search space, as well as log (write) required information such as what component is being assigned and in which port.

The component information level is the highest level of interaction between the software logic and the Portlist stored data. The number of components contained in the Portlist determines the size of the database since each vertical structure (Comp x) in Figure 17 corresponds to each component. The structures (Comp x) are created in the same order as the components are listed in the Portlist. This level of information is particularly useful to move an entire component informational set. However, the information extracted from this level is poor to determine which component is being selected. This is where the second level of information is highly useful.

The second level of interaction is the component parameters information level. At this level, the system can retrieve specific information from each component including ID, type, name,

value, part number, and usage as shown on the *left* side of each vertical branch in Figure 17. This information allows the system to obtain identification characteristics of each component. In contrast the *right* side of each vertical branch provides 64 port parameters to store the port location of each terminal of a component. It is important to note that the port parameters have their own subset (3rd level) of information.

The decision to provide 64 parameters was due to the specific size of the hardware created for this thesis since each port board and component card has 64 ports. Null statements (instead of port numbers) are placed in excess parameters when components have less than 64 terminals. The null statements are useful because they signal a port searching process to stop as soon as a null port parameter is reached.

The third information level is a port parameter subset that stores to which line (or net) that port should be connected and if such connection is safe for the hardware to create. Essentially this level provides information of how the components will be interconnected to make a circuit. In addition, it provides information regarding any potential electrical damage that may occur if such a circuit is built. This level of information is only accessed during a design rule check and information is not written until the component is marked as used by the software.

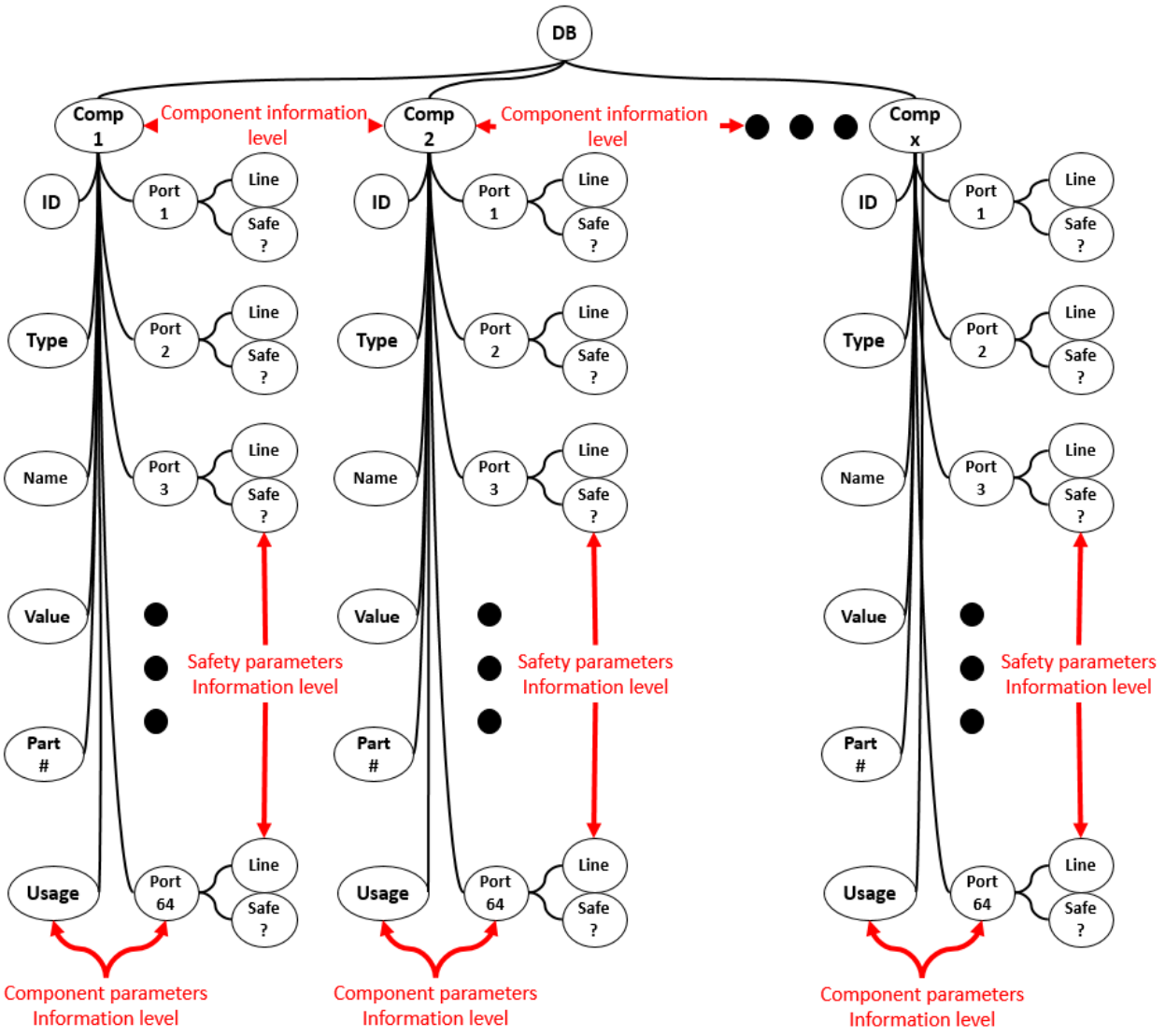


Figure 17 Component database structure

4.1 Portlist-to-Database Data Transfer Process

The algorithm used to fill the contents of the database is shown in Figure 18. Information in the Portlist is structured such that each line of text represents a component. When the database is created, its size or number of "Comp" structures (Figure 17) is set to the number of lines or components found in the Portlist. Moreover, each line is delimited by the "enter" character and

this allows the software to recognize each line of the Portlist divide it into component information blocks.

Taking each line of text or component at a time, the software transfers the Portlist data in the database. The first step is to create a null data set to override any information the data set may have in the case of a previous existence of another database. This is similar to a clear or reset function. The next step is to split the component information block into its elemental or second level of information that contains the type, name, ports, value and part number. The ID parameter is filled by assigning the same value as the initial position this component has in the database. Additionally, the usage parameter is initialized as “false” since no circuit has been created at this point. Similarly, the third level of information is a subset added by software and initialized as null by the fact that no circuit has being created yet. Once each element from the component information block is written into its corresponding field in the database, the process repeats if there are any more components.

Since the size of the database is based on the number of components present in the Portlist, it is guaranteed that no data set will remain null at the end of the database filling process. Therefore, when all data sets are full the software knows that no other component block exists within the Portlist and proceeds to save the database into ram memory.

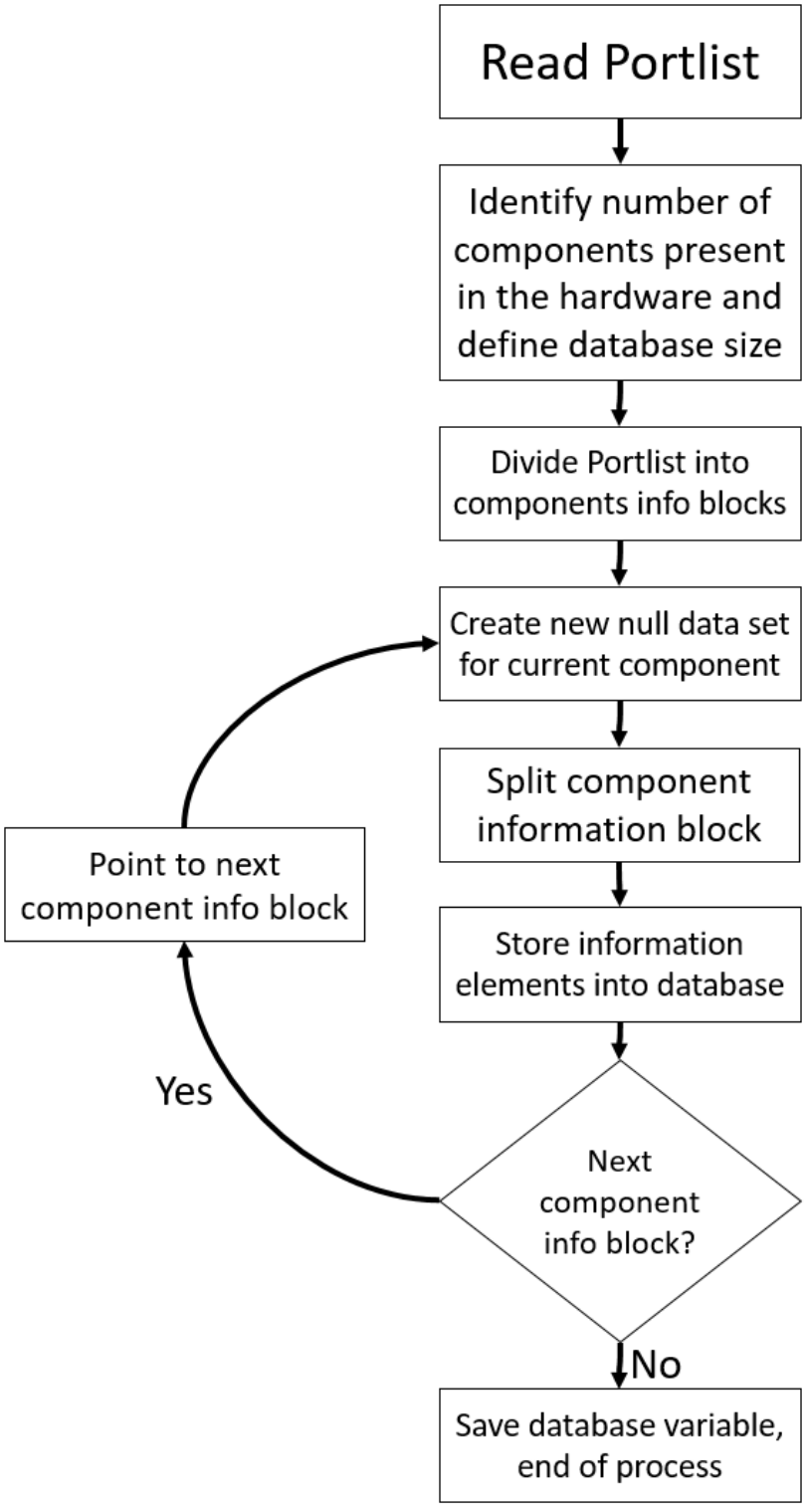


Figure 18 Database filling algorithm

4.2 Database Special Conditions

The database presents two special conditions that can occur and need to be taken into consideration for an identification process that will be explained in more detail in chapter 5.

4.2.1 Tracking Order of Component Terminals

For many components the function of each terminal is unique. This creates the need to track the location of individual terminals of each component. In a SPICE netlist this is accomplished using a standard order in which a component's terminals are listed. For example, in a netlist for a diode the third element is always the anode, and the fourth element is the cathode. Figure 19 shows the netlist for the 1N4002G diode indicating that the third element is reserved for the anode and the fourth element is reserved for the cathode. Therefore, when storing this component in the software's database it is important to maintain the position of each terminal. Also, the Portlist file needs to adhere to the SPICE netlist standard regarding the order of the terminals. For example, if the anode and cathode of a diode happen to be in port numbers 15 and 14, respectively, then the values "15" and "14" should be stored in the component parameters "port 1" and "port 2", respectively, to preserve the diode terminal arrangement information.

Figure 19 Diode 1N4002G Netlist information block. First terminal shown correspond to the Anode of the diode. Second terminal shown correspond to the Cathode of the diode.

4.2.2 Case Sensitivity

Another key factor to consider is that in a netlist file, component types are case-sensitive. An example netlist is shown in Figure 20 where a capital 'X' represents an oscilloscope instrument

and a lower case ‘x’ represents an Op-Amp. This distinction is crucial to identify components in future processes.

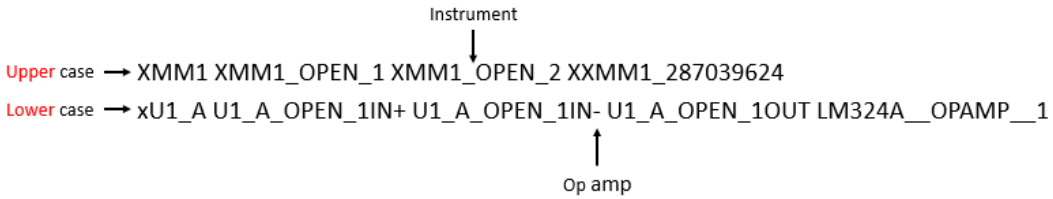


Figure 20 Component type difference between an op-amp and an instrument

5. Netlist to Hardware Programming Code

The pervious chapter delt with creating a database of physically available components in the circuit building system without considering how they are to be interconnected to make a circuit. In contrast, this chapter presents an algorithm to interconnect a set of components into a circuit using the information from a SPICE netlist and the database. It is useful to note here that the SPICE netlist has information about the circuit to be built. Whereas, the database has information about the components physically present in the circuit building system. The circuit building system combines these two pieces of information as shown in Figure 12 to make a circuit.

In general the algorithm consists of first identifying the components listed in a netlist as shown in Figure 21. This is followed by a scan to ensure that the needed component and nets are physically available in the circuit building system. If availability of a component and required nets are confirmed, the component is reserved and its port connections to the circuit nets is encoded into the database. A circuit design rule check is then performed. Finally, if the circuit to be built passes the component and net availability, and design rule checks, a hardware binary address is created for all the interconnections needed to make the circuit. Otherwise the software will generate an error code and the process will be stopped. The sections below describe each of the major steps of the algorithm.

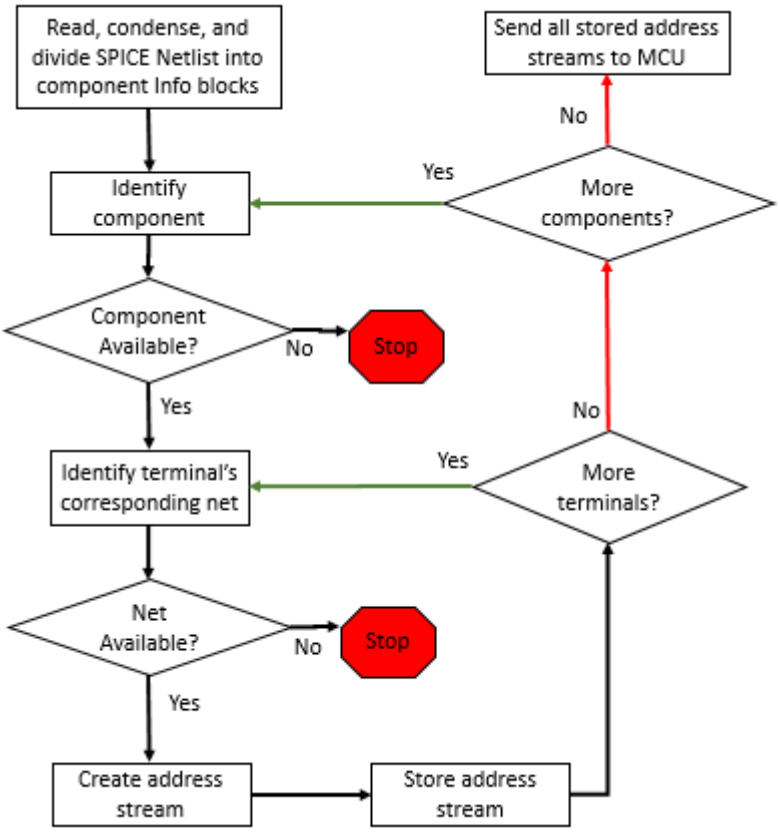


Figure 21 Automated circuit build algorithm

5.1 Classification of Components on Netlist

The first step of the algrorithm is to identify the characteristics of each of the components in the SPICE netlist and encode the information into a structured variable. The purpose of encoding the characteristics of individual components into a structured variable is to facilitate comparing with the database of the circuit building system.

SPICE netlists contain critical as well as non-critical information that need to be separated. In this work, the SPICE netlists are condensed by removing non-critical content that do not give information regarding a component. This is achieved by analizing a SPICE netlist line-by-line and removing lines in which the first character is not a component type. Component types are identified with the following characters: *"r, c, l, d, x, X, a, v,"* and a special condition for power instruments

where the first three characters are identified as *"VSS"* or *"VDD"*. Figure 22 shows an example of how the SPICE netlist looks like after deleting all non-critical information. This condensed format of a SPICE netlist still contains all necessary information of the required components to build a circuit. After the condensing process, each line in the netlist corresponds to a single comopnent and the information is called a "component information block".

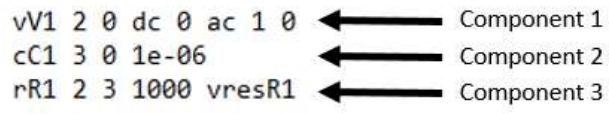


Figure 22 SPICE netlist cleared from non-critical information

After the SPICE netlist has been condensed, the next step is split each component information block into "elements" that describe the characteristics and the terminal-to-net connections of the components as shown in Figure 23. Figure 23 shows a component block for a resistor broken down into information elements to identify such resistance. The elements are ordered from left to right and are delimited by a space character. The only exception is the first two elements *"Component type"* and *"Component name"* which are not delimited by any character. However, since the *"Component type"* is a single character it can be easily identified as the first character of the entire line.

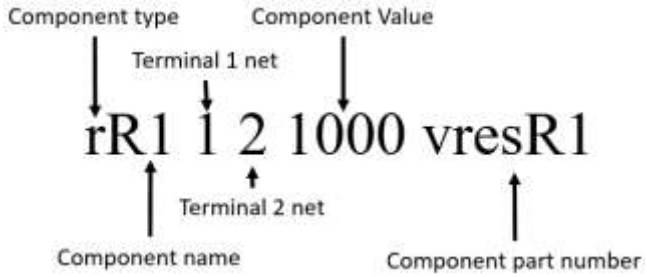


Figure 23 SPICE Netlist component R1 information block representing R1 in Figure 22 and its corresponding elements

The elements are then stored in a structured variable called “ND” as shown in Figure 24, which is similar to the database structure (Figure 17) discussed in chapter 4. The differences between this new data and the database previously discussed are that the ND variable can only store one component at the time and the *right* side of the ND contains the *nets* that the terminals of the component are connected to instead of the *ports*. Moreover, the ND variable does not contain a third level of information.

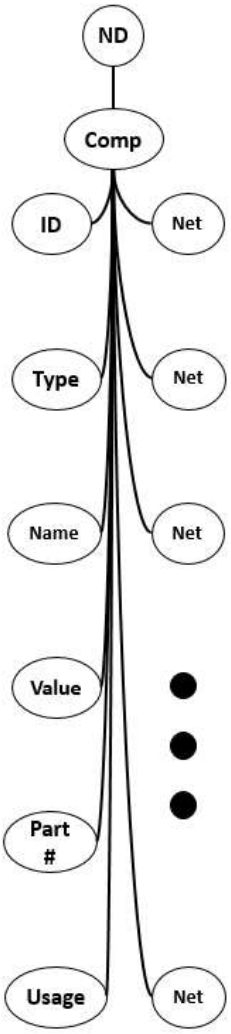


Figure 24 ND structure variable

After the ND variable is created, the next step of the identification process is to classify the component into one of the three categories as shown in Table 4. The classification is based on the component type. Classification is important because it will dictate the process used to search for components and confirm their existence on the circuit building system as described in the next section.

Table 4 Components and elements classifications

Component type	**Classification**
r	Generic component
l	
c	
d	Specific component
x	
X	
a	
v	
VSS	Net element
VDD	

5.2 Search for Existence of Components on Circuit Building System

In general, the search for the existence of a required component on the circuit building system consists of comparing the ND variable with the database variable as shown in Figure 25. A match in the parameters indicates that the component required by the circuit is physically present on the circuit building system.

During the search process, the database variable is arranged using a bubble sort algorithm [8]. For example, if the component type being searched in the database is ‘r’, the database is sorted in such a way that all components with ‘r’ as their type of parameter are the first elements in the database. This reduces the search space and thus avoids searching the entire database each time a

search process takes place. Other details of the search depend on the classification of the component. Once the classification of the component is determined the software executes one of three searches as listed in Table 5.

Table 5 Search process based on the component classification

Component classification	Search process
Generic	Value based
Specific	Part number based
Power	Net element based

5.2.1 Value Based Search

Components that fall under the 'generic' classification have a parameter value that is either resistance, inductance, or capacitance. For this class of components, the software executes a research based on the component type and parameter value as emphasized in Figure 25 with the red (type) and green (value) traces.

5.2.2 Part Number Based Search

If the component falls into the 'specific' classification; a search based on the part number is performed. This is emphasized in Figure 25 with the red (type) and blue (part number) traces.

5.2.3 Net Element Search

If the component is classified as a 'power element', the software performs a net search process where the only requirement is to determine the parameter name of the power element. Since it is a difficult task to determine through software the amount of voltage or amperage that will be applied by the source, the software searches for a non-used line in the hardware and reserves it as a net for the power element. It also renames the line as the name obtained from the parameter

in ND variable. Currently, the software only handles DC sources as net elements. This means that other power sources such as AC wave generators are classified as specific components and are handled as components even if they are instruments.

Regardless of the type of comparison performed the software always looks for exact coincidences although, some coincidences may be interpreted as exact depending on the parameter is being compared. For example, in the case where a generic component a value parameter of "2.2E3" is required but the database variable value parameter is "2200", the system will try to parse the scientific notation and compare both values as integers and in the case of a match the existence of the component is validated.

In the event that no component is not found in the database, the software terminates the "Netlist to hardware programming" process and prompts the user with a message regarding the error. However, future versions of the software could include a new process to search for "similar" components.

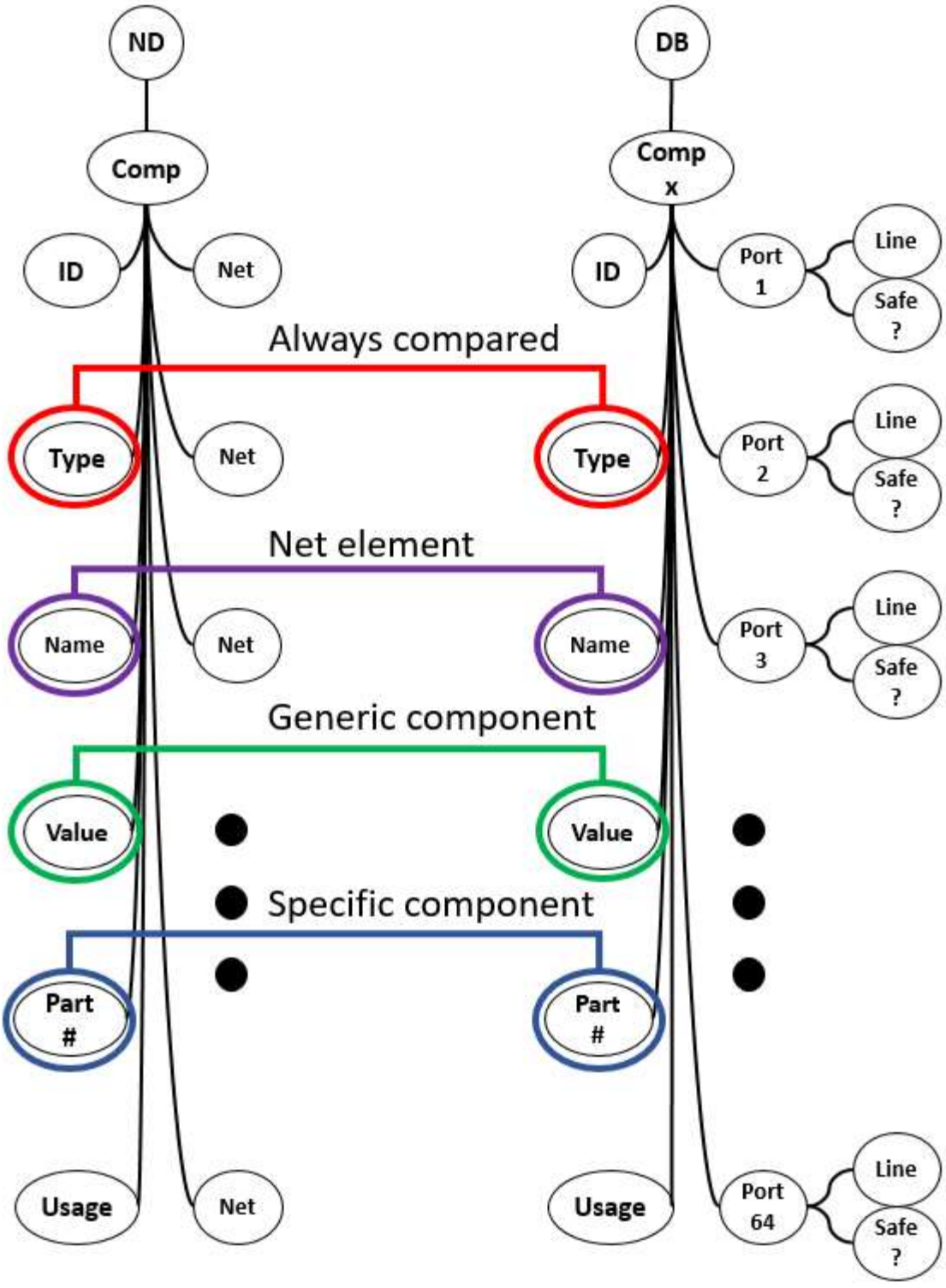


Figure 25 Physical validation comparing the ND variable and the DB variable.

5.3 Search for the Existence of Lines in Circuit Building System

In the event of the database having an instance of a component that match the required searching criteria as described above, the software implements a validation to verify if the lines to where a component is to be connected exists. The process is a comparison between the required net and one of 16 possible cases, this 16 cases are accessed by the software depending on the name of the required net by the SPICE netlist ranging from the strings “0” to “15” as a SPICE netlist list its net names automatically. Users can rename the net name in the schematic capture software and thus confusing the software to not recognize a renamed net. Since the hardware has no expansion towards the lines the software does not adapt to the possibility of having more cases available to identify lines.

A future process can be implemented in the software to adapt the possibility of having a SPICE netlist with personalized net names as well as a future implementation for line number adaptability.

5.4 Availability and Selection of Components and Lines for Circuit

A successful search for the existence of a component means that one or more instances of a component required by the circuit are physically present in the circuit building system. The next step is to determine whether any of the instances are available. The distinction between the existence and availability searches is that the former determines whether one or more instance of a component required by a circuit is/are physically present in the system and the latter determines whether at least one instance is available (or not) for use. The determination of availability consists of reading the contents of the usage parameter in the database as shown in Figure 26. If an instance of a component is available, the usage parameter will contain “false”: the algorithm will then

immediately select it and make it unavailable for any subsequent search by writing “true” into usage parameter of the database.

On the other hand, if the content is “true” the component is not available. The usage parameter ensures that physical components are assigned to at most one instance in a circuit. In other words, it avoids selecting the same physical component two or more times.

Once an available instance of a component is found, the next step is to determine whether enough lines and/or nets are available for each of the terminals of the component. If enough lines and/or nets are available, the net information from ND is written into the corresponding port parameter in the database as shown in Figure 26. Importantly, this is the key step in defining how the components will be interconnected with lines to make a circuit specified by a netlist. Once this step is accomplished for all the components on the netlist, how the physical components are interconnected with lines in the circuit building system to make a circuit is completely defined. At this point, the database contains all the information needed to make a circuit specified on a netlist. (As a reminder, lines are the vertical wires of the cross-point analog switch array. A used line becomes a net of a circuit.)

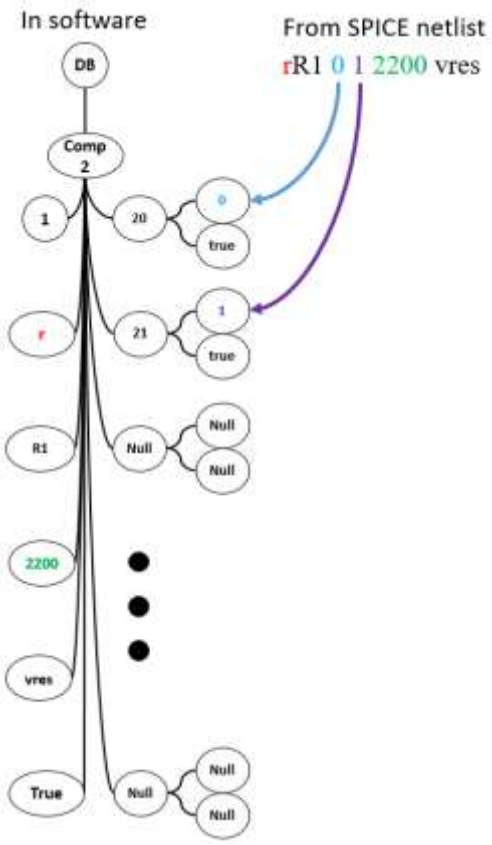


Figure 26 Assignation of nets to corresponding ports

5.5 Circuit Design Rule Checks

As a safety and system protection measure, circuit design rule checks can be performed before a circuit is built. After the interconnections of all the needed components have been defined as shown in Figure 26, the software performs conditional tests based on circuit design rules to determine if there are any connections that might create an electrical hazard or cause damage to the system. A list of design rule checks and their level of importance are shown in Table 6.

Table 6 Design rule checks based on the components used and their degree of importance

Classification	Components	Rule checks performed	Importance level
NA	Ground	Floating ground	Medium
		Ground to power	High
Generic	Resistance	Short circuit test	High
		Floating test	Low
Generic	Capacitors	Short circuit test	High
		Floating test	Low
Generic	Inductors	Short circuit test	High
		Floating test	Low
Specific	Op-Amp	Ground to power	High
		Output to ground	High
		Inverted power supplies	High
		Floating test	Low
Net Element	Power supplies (Wave gen, DC power supply, etc.)	Ground to power	High
		Floating test	Low
Specific	Multimeter (Readers)	Short circuit test	High

Each rule check has its own importance level depending on the severity of consequence of a failed rule check. For low-level failed rule checks, the software will prompt users of a possible error made in the circuit design and proceed its program as normal. In the case of medium-level failed rule check, the software will warn the user regarding the error and stop its process, leaving files created incomplete. Finally, in the case of high-level failed rule check, the software will immediately terminate any process, warn the user of prevented damage and proceed to delete all files created during the process.

The design rule checks are based on electrical circuit theory. For example, if a resistor has two of its terminals connected to net number 2, it is concluded that this component is shorted. In another example, if a power supply is connected to net 0 which is reserved for ground, a failure flag will rise indicating a ground to power fault. The software will then terminate the netlist-to-hardware programming process and delete the resulting file from the process due to its high level

of importance. If no circuit design rule is violated, the software proceeds to the next step which is to create a set of binary addresses.

5.6 Hardware Binary Address Streams

In this final step, the system will produce a set of binary addresses corresponding to the port-line intersections the system needs to make the circuit. The addresses are used to program the cross-point analog switch array. Each address consists of two words: one corresponding to the port number and the other to the line number. In general, the addressing scheme is hardware independent and points to the intersections where ports and lines interconnect.

The software generates addresses for each of the components that have a “true” in the usage parameter of the database. For example, Figure 27 shows a component with “true” in the usage parameter. This component is located at ports 20 and 21 and these two ports are to be connected to lines 0 and 1, respectively. The software will translate these numbers into their binary form as shown in Table 7 and Table 8. In this example, the binary word corresponding to the ports is 8-bits long, and the word corresponding to the lines is 4-bits long. The set of 12-bit addresses are then sent to a microcontroller unit to program the cross-point analog switch array and create the required interconnections to make a circuit.

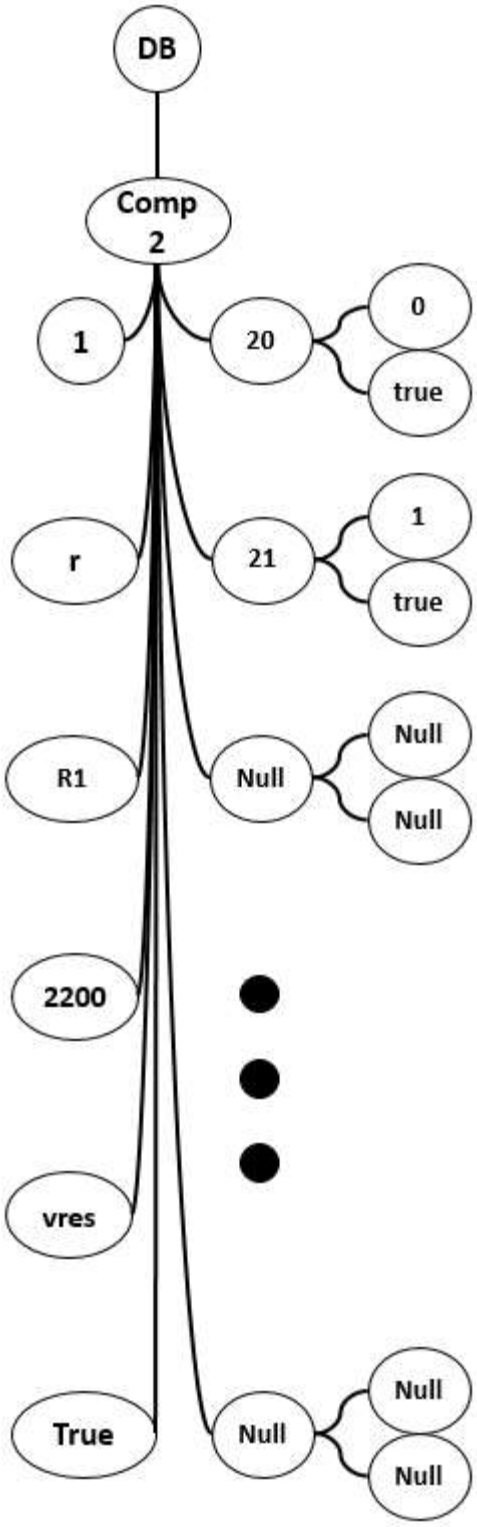


Figure 27 Required component for binary address stream

Table 7 Address stream for port parameter 1 shown in Figure 27

4-bits Lines				8-bits Ports							
CS2	AY2	AY1	AY0	BE1	BE0	CS1	CS0	AX3	AX2	AX1	AX0
0	0	0	0	0	0	0	1	0	1	0	0

Table 8 Address stream for port parameter 2 shown in Figure 27

4-bits Lines				8-bits Ports							
CS2	AY2	AY1	AY0	BE1	BE0	CS1	CS0	AX3	AX2	AX1	AX0
0	0	0	1	0	0	0	1	0	1	0	1

5.7 Machine Programing Code Synthesis

Once all address streams are created for all the required components, a final product file called “the machine programming code” is created which can be loaded into a microcontroller unit. This part of the code is hardware dependent and the for the purpose of this thesis was created for the Digilent Analog Discovery II platform. However, further code can be added to the software to support other microcontrollers. As an example, for the Digilent platform, the file consists of a set of instructions for the microcontroller to drive its general-purpose input/output peripherals and a portion of the file is shown in Figure 28.

In conclusion, the user can repeat the netlist to hardware programming code process with different SPCIE netlists allowing the user to create different machine programming codes thus creating several circuits in a matter of seconds. Like a netlist, a machine programming code can be stored in nonvolatile memory and used whenever needed without having to repeat the component identification, component availability, circuit design rule check, and address creation steps.

```
StaticIO.Channel1.DIO14.text = "0";
StaticIO.Channel1.DIO15.text = "1";
StaticIO.Channel0.DIO0.text = "1";
StaticIO.Channel0.DIO1.text = "0";
StaticIO.Channel0.DIO2.text = "1";
StaticIO.Channel0.DIO3.text = "0";
StaticIO.Channel0.DIO4.text = "1";
StaticIO.Channel0.DIO5.text = "0";
StaticIO.Channel0.DIO6.text = "0";
StaticIO.Channel1.DIO14.text = "0";
StaticIO.Channel1.DIO15.text = "0";
StaticIO.Channel0.DIO7.text = "0";
StaticIO.Channel1.DIO8.text = "0";
StaticIO.Channel1.DIO9.text = "0";
wait(0.001*speed);
StaticIO.Channel1.DIO13.text = "1";
wait(0.003*speed);
StaticIO.Channel1.DIO13.text = "0";
wait(0.001*speed);
StaticIO.Channel1.DIO14.text = "1";
StaticIO.Channel1.DIO15.text = "1";
```

Figure 28 Portion of machine programing code file connecting one terminal

6. Hardware Programming

It is typical that cross-point analog switch arrays are programmed using timed voltage signals. The programming specifications will depend on the specific cross-point analog switch array part numbers. However, the software's source code is configured to allow an adaptation in the case of a different cross-point analog switch arrays are used. In this thesis, the Mitel MT8816 cross-point analog switch arrays are used, and their programming method and timing will be used as an example to explain how the software can adapt to a different cross-point.

6.1 Timed Signal

The cross-point analog switch arrays require an address in combination with control signals to program a connection. In the case of the MT8816 the control signal are chip select (CS), data (DATA), strobe (STROBE), and reset (RESET). These control signals and their protocols are shown in Table 9the integrated circuit. This control memory signals are shown in Table 9 and Figure 29 [9].

Table 9 Control and I/O Timings [9]

	Characteristics	Sym	Min	Typ‡	Max	Units	Test Conditions
1	Control Input crosstalk to switch (for CS, DATA, STROBE, Address)	CX_{talk}		30		mVpp	V_{IN}=3V squarewave, R_{IN}=1kΩ, R_L=10kΩ. See Appendix, Fig. A.6
2	Digital Input Capacitance	C_{DI}		10		pF	f=1MHz
3	Switching Frequency	F_O			20	MHz	
4	Setup Time DATA to STROBE	t_{DS}	10			ns	R_L= 1kΩ, C_L=50pF ⓘ
5	Hold Time DATA to STROBE	t_{DH}	10			ns	R_L= 1kΩ, C_L=50pF ⓘ
6	Setup Time Address to STROBE	t_{AS}	10			ns	R_L= 1kΩ, C_L=50pF ⓘ
7	Hold Time Address to STROBE	t_{AH}	10			ns	R_L= 1kΩ, C_L=50pF ⓘ
8	Setup Time CS to STROBE	t_{CSS}	10			ns	R_L= 1kΩ, C_L=50pF ⓘ
9	Hold Time CS to STROBE	t_{CSH}	10			ns	R_L= 1kΩ, C_L=50pF ⓘ
10	STROBE Pulse Width	t_{SPW}	20			ns	R_L= 1kΩ, C_L=50pF ⓘ
11	RESET Pulse Width	t_{RPW}	40			ns	R_L= 1kΩ, C_L=50pF ⓘ
12	STROBE to Switch Status Delay	t_S		40	100	ns	R_L= 1kΩ, C_L=50pF ⓘ
13	DATA to Switch Status Delay	t_D		50	100	ns	R_L= 1kΩ, C_L=50pF ⓘ
14	RESET to Switch Status Delay	t_R		35	100	ns	R_L= 1kΩ, C_L=50pF ⓘ

† Timing is over recommended temperature range. See Fig. 3 for control and I/O timing details. Digital Input rise time (tr) and fall time (tf) = 5ns.
‡ Typical figures are at 25°C and are for design aid only; not guaranteed and not subject to production testing.
ⓘ Refer to Appendix, Fig. A.7 for test circuit.

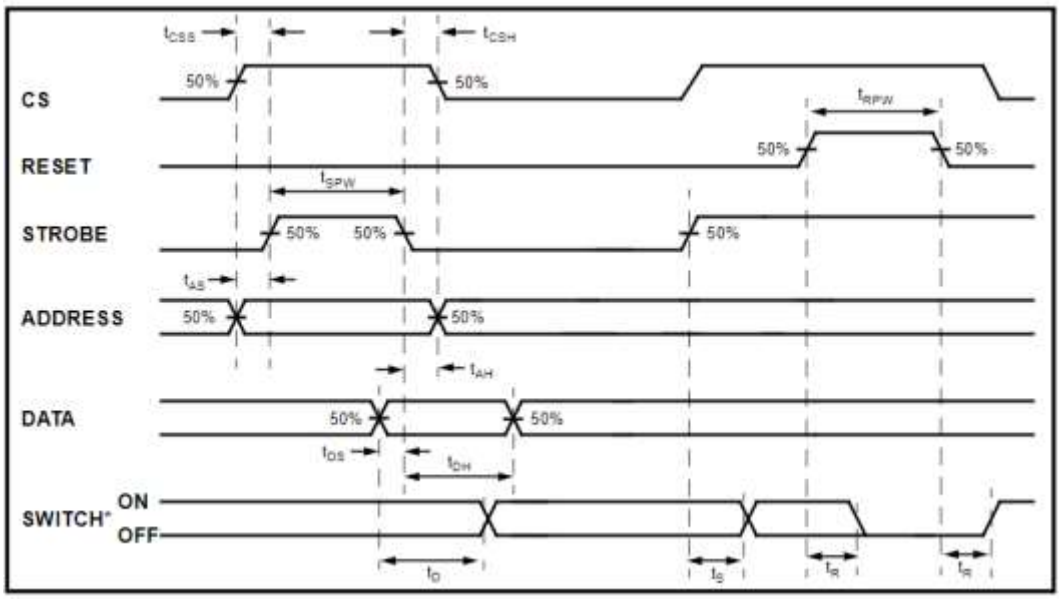


Figure 29 Control memory timing diagram [9]

The STROBE signal is the one in charge of pushing the selected address into the internal circuitry of the cross-point analog switch arrays. The CS, DATA and ADDRESS signals need to be stable for a minimum of 10 nanoseconds at the rise and fall of the STROBE signal. Additionally, for the STROBE signal to accomplish its duty, it must remain high for a minimum time of 20 nanoseconds. Leaving the DATA signal on for the entire process, we can only focus on the three remaining signals CS, ADDRESS and STROBE. However, as mentioned in previous chapters the address scheme already includes the CS signal therefore, the address scheme is a combination of the CS and ADDRESS signals which leaves only the need of controlling the STROBE signal once the address scheme is loaded in the system.

The time needed to connect a component terminal to a line can be estimated. Each address stream is a 12-bit signal and for the case of the Digilent microcontroller each bit represents an instruction in the hardware programming code. However, assuming ideal conditions where all the instructions are performed instantaneously, the programming duration will depend on the requirements for the strobe signal. In this case the following three timings will determine the

programming duration; (1) a wait time of 10 nanoseconds is needed once the address stream is loaded into the system, (2) the STROBE signal rises and stays high for 20 nanoseconds and then falls to low voltage afterwards, (3) subsequently, the address stream is cleared after another 10 nanoseconds. In this estimate each component terminal can be connected in 40 nanoseconds as shown in Table 10. As mentioned in previous chapters an address stream is the equivalent of one component terminal, meaning that a component terminal is connected in 40 nanoseconds using this system as shown in Table 10.

Table 10 Control timing assuming ideal conditions where the load and clear of the address stream occurs instantaneously.

Step	Signal	Time	Total time
1	Set up time Address Stream to STROBE (t_{AS})	10 ns	10 ns
2	STROBE Rise and Fall (t_S)	20 ns	30 ns
3	Hold time Address Stream to STROBE (t_{AH})	10 ns	40 ns

In more realistic conditions, a microcontroller clock speed determines the time rate for each instruction. Assuming a clock speed of 1MHz, we can estimate each instruction is performed in one microsecond meaning that to initially load and clear the address stream a total of 12 microseconds are needed. If this new timing required to load and clear the address stream is added to the time required to connect the cross-point analog switch array, 24.04 microseconds are required to connect one terminal.

The key aspect of this timing signal is not the time the microcontroller takes to load and clear the address stream but the three timing steps that are required to "properly strobe" the address stream into the cross-point analog switch arrays. In Figure 30 we can observe a portion of the code that generates the hardware programming code which in its bottom portion we can see that there

are three dedicated lines with a “wait()” command which instruct the micro-controller to do nothing for the specified amount of time. To adapt the wait times for different cross-point analog switch array models, these three lines can be modified or replaced with a different wait method of choice.

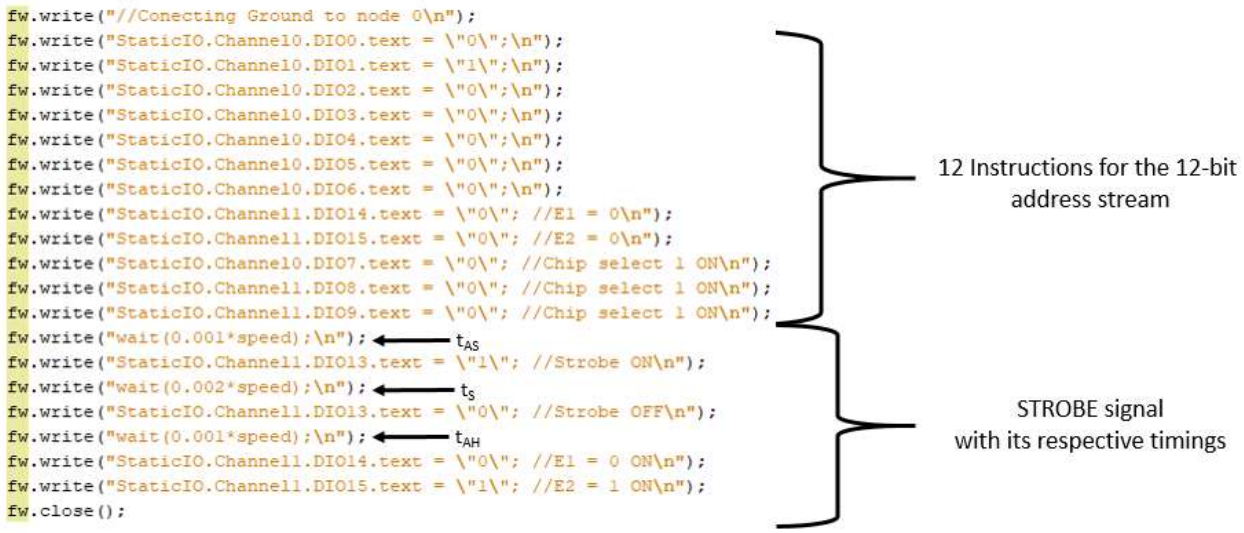


Figure 30 Hardware programming code portion to connect ground to node 0.

7. Results

7.1 Software Interface

A user interface as shown in Figure 31 was created to manipulate the software's functions as described in previous chapters. The main window of the user interface is called "RCB" and contains two submenus called "File" and "Setup". The "File" submenu has several buttons and text fields. The "OPEN" button will cause a file navigator window to open to search and select the desired SPICE netlist that will be converted into machine code as shown in Figure 32. The text field called "Scrip Name" allows users to enter the name of the resulting machine programming code. The "Create Script" button that will start the SPICE netlist to machine programming code process. The text area supplies feedback to the user about any errors that occurred during the conversion process or warnings from the circuit design rule checks. The check box called "Include Clear Function?" is only for debugging purposes. Finally, the "Exit" button will end the program.

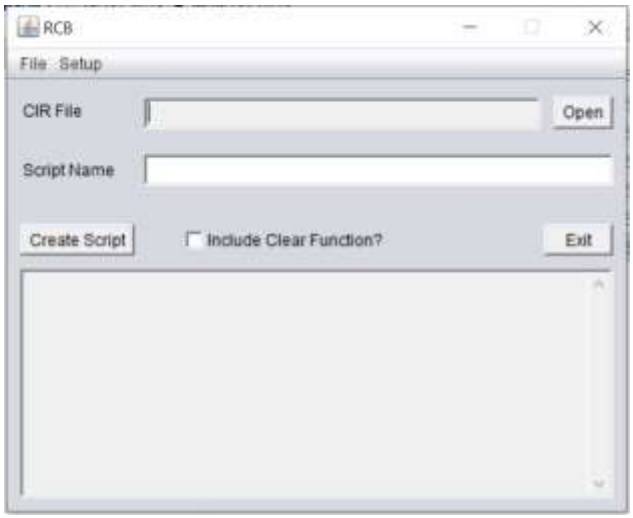


Figure 31 Software's main window

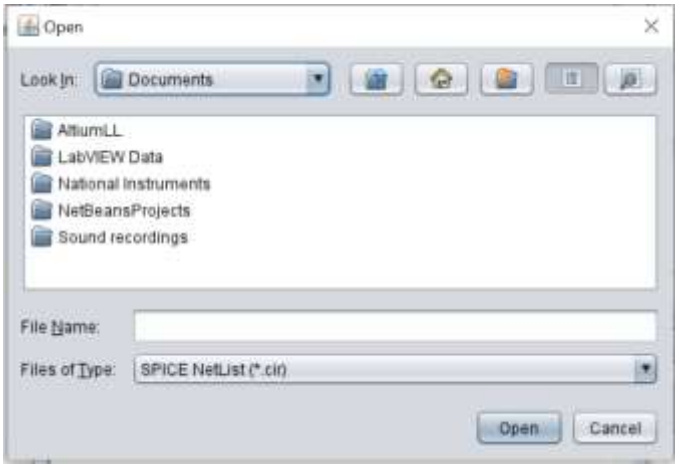


Figure 32 Software's file explorer

The “Setup” submenu is used to input the configuration of the hardware into the software. This task needs to be performed every time the hardware configuration is changed. The “Setup” submenu will open a window as shown in Figure 33 where the user can select the number of boards in the system and their corresponding component cards. Inside each panel there is a drop box to select the component card resource file for each port board. The “Configure” button will cause the software to read the component card resource files and generate and store a Portlist for that specific configuration. The Portlist will be saved in the same directory as the main program application.

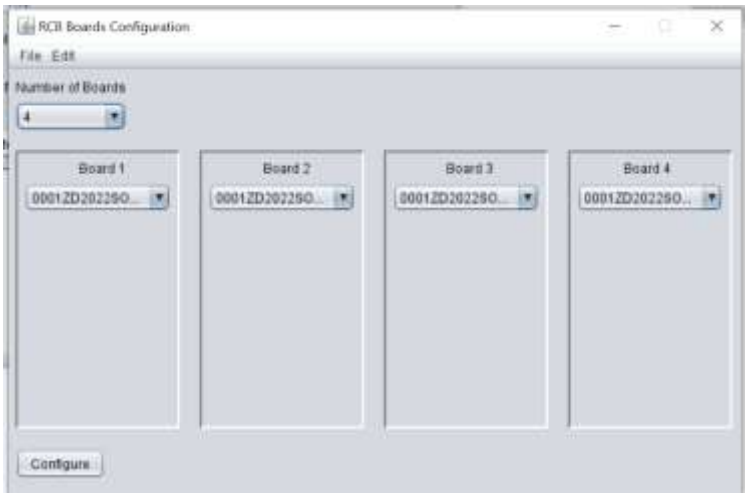


Figure 33 Board Configuration window

7.2 Circuit Building System Evaluation

Preliminary versions of the circuit building system were tested to evaluate how it would perform in practice. In this evaluation, two users collaborating remotely tested the system following the workflow diagram shown in Figure 34. The test procedure mimicked an educational laboratory exercise in which users collaborate with distinct roles to carry out a prototyping task. The task consisted of designing, building and testing an electronic filter using the system and recording the time spent on each prototyping phase. No other instructions were given regarding the type of filter.

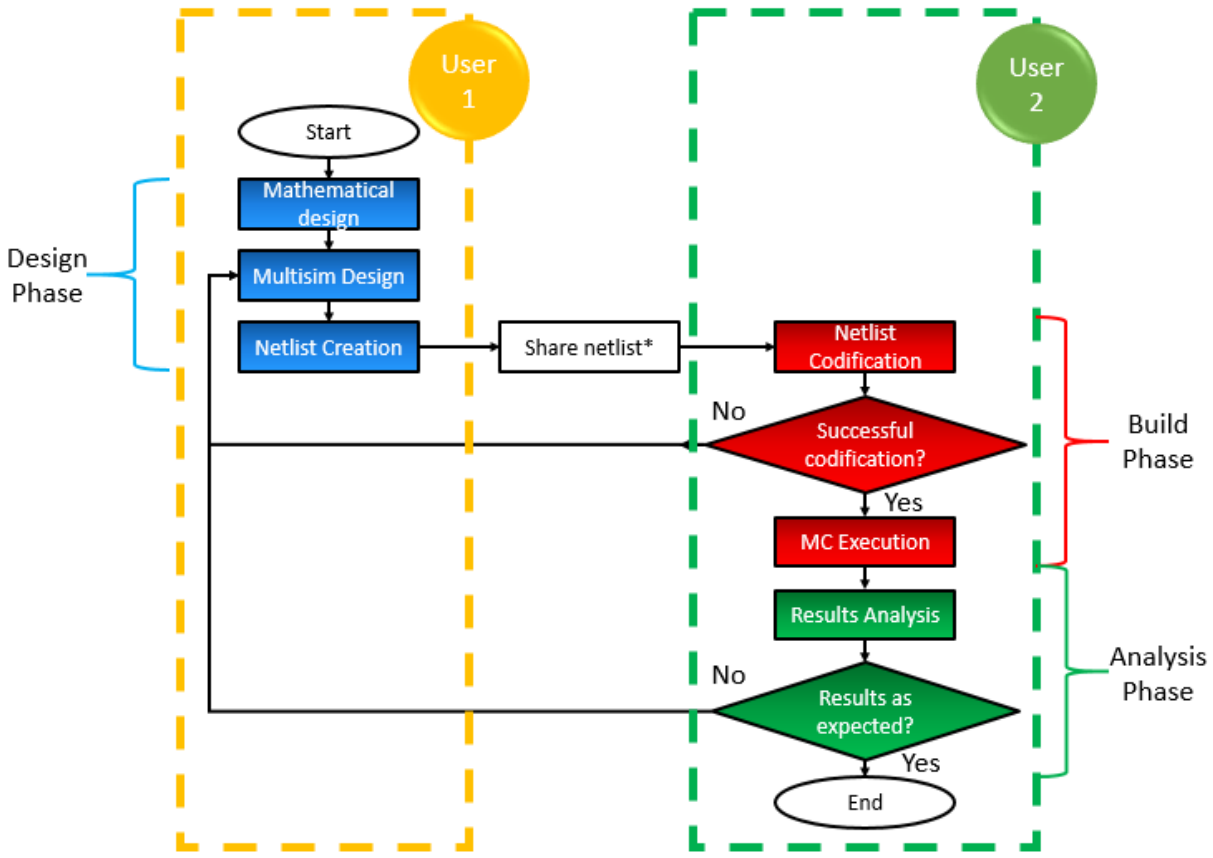


Figure 34 Prototype test workflow

The users opted to design a Twin-T notch filter as shown in Figure 35. During the design phase, user one recognized that not all the components physically present in the system matched the component values from the calculations. To account for the discrepancy, a combination of components was used to approximate the calculated values. This can be observed in Figure 35 where some parts of the circuit have two components while others have only one like in the case of “C1”, “C2” and “C4”.

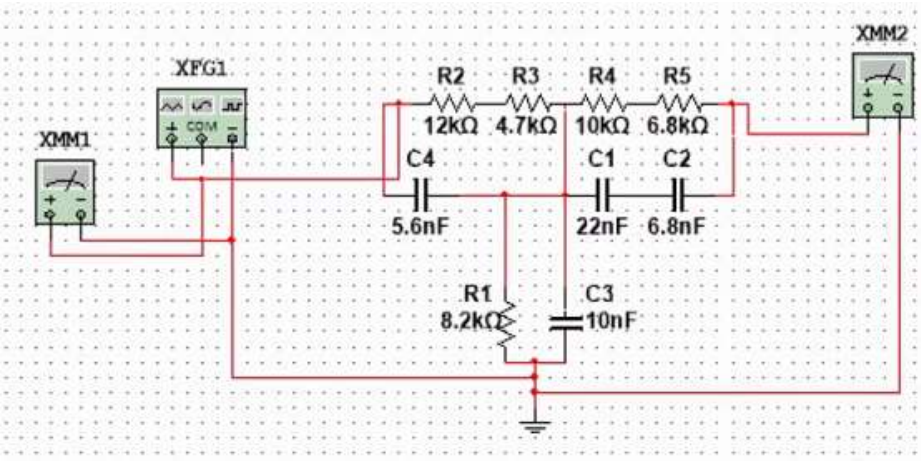


Figure 35 Twin-T notch filter designed using NI Multisim software

User-one then used the schematic capture software to produce a SPCIE netlist file and send it to user-two through email. Upon receiving the SPICE netlist file, user-two then used the software interface to create a set of instructions for the microcontroller. However, the circuit design rule checker was able to detect a mistake in the schematic of the circuit and prompted the user that resistor R4 had a terminal that was open. Resistor R4 appeared to be connected but there was no wire between R4 and R5 meaning that in fact the second terminal of R4 was open. User-one proceeded to revise the schematic design to fix the error as shown in Figure 36 and sent a new SPICE netlist to user-two. User-two then used the software to translate the SPICE netlist from the corrected circuit design without issues on the second attempt. The resulting machine programming code was then loaded into the Digilent microcontroller to build and analyze the circuit.

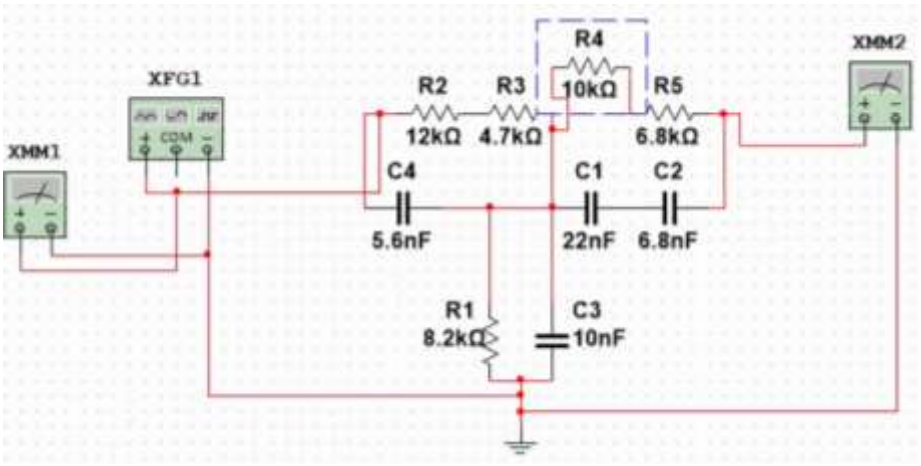


Figure 36 Circuit schematic correction

Using an oscilloscope instrument, user-two performed two tests consisting of an AC sweep analysis and a network analysis. The AC sweep analysis showed that the amplitude of the output (blue line) was minimum at a frequency corresponding to 1.9 kHz as shown in Figure 37. In contrast, the amplitude input signal (yellow line) remained constant throughout the AC sweep.

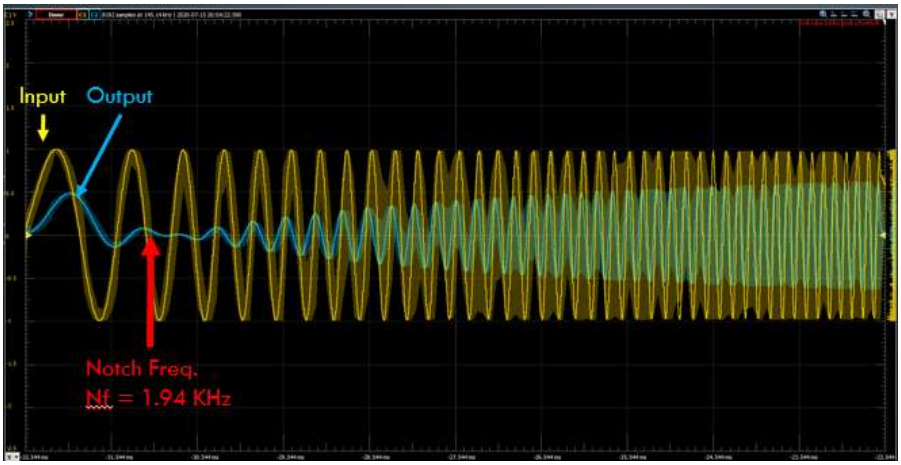


Figure 37 Twin-T notch filter AC sweep analysis

A network analysis was also performed to further verify that the constructed circuit in the system was acting as a notch filter. The bode plot is shown in Figure 38 and shows the output of the circuit (blue line) dropped by 40 dB at approximately 1.9 kHz in good agreement with calculations of the notch filter.

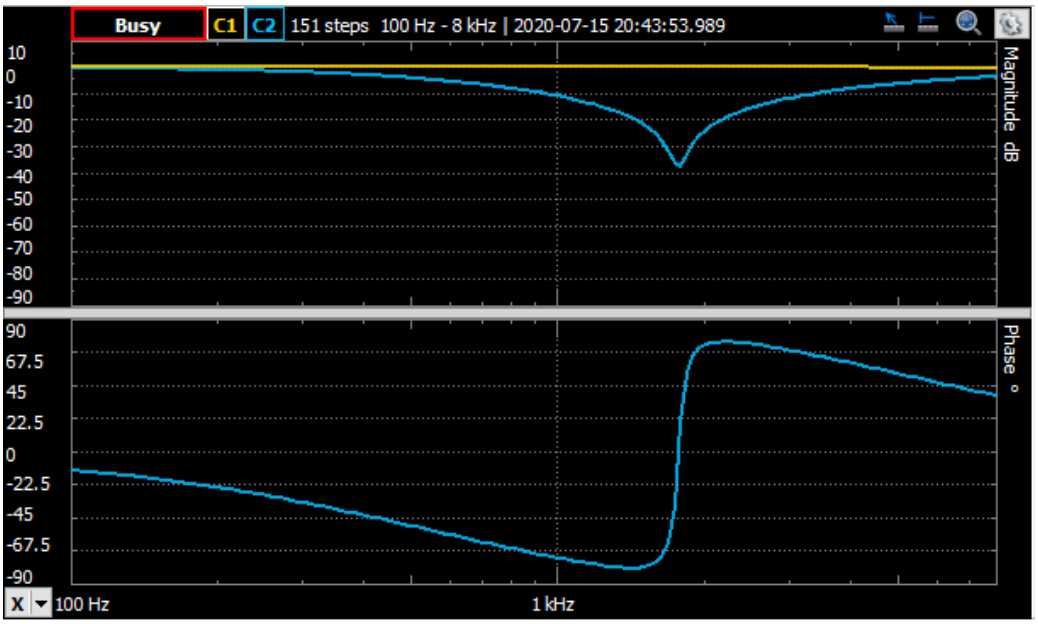


Figure 38 Twin-T notch filter bode plot

Figure 39 shows the timeline of the prototyping workflow to visualize the amount of time spent on each of the prototyping phases. User-one took around 35 minutes to design the twin-T notch filter. User-two needed approximately 10 minutes to build the circuit. Finally, user-two spent another 15 minutes analyzing the circuit behavior. The complete prototyping process lasted 60 minutes as shown in Figure 39 showing that the building phase of the process used only 17% of the total time.

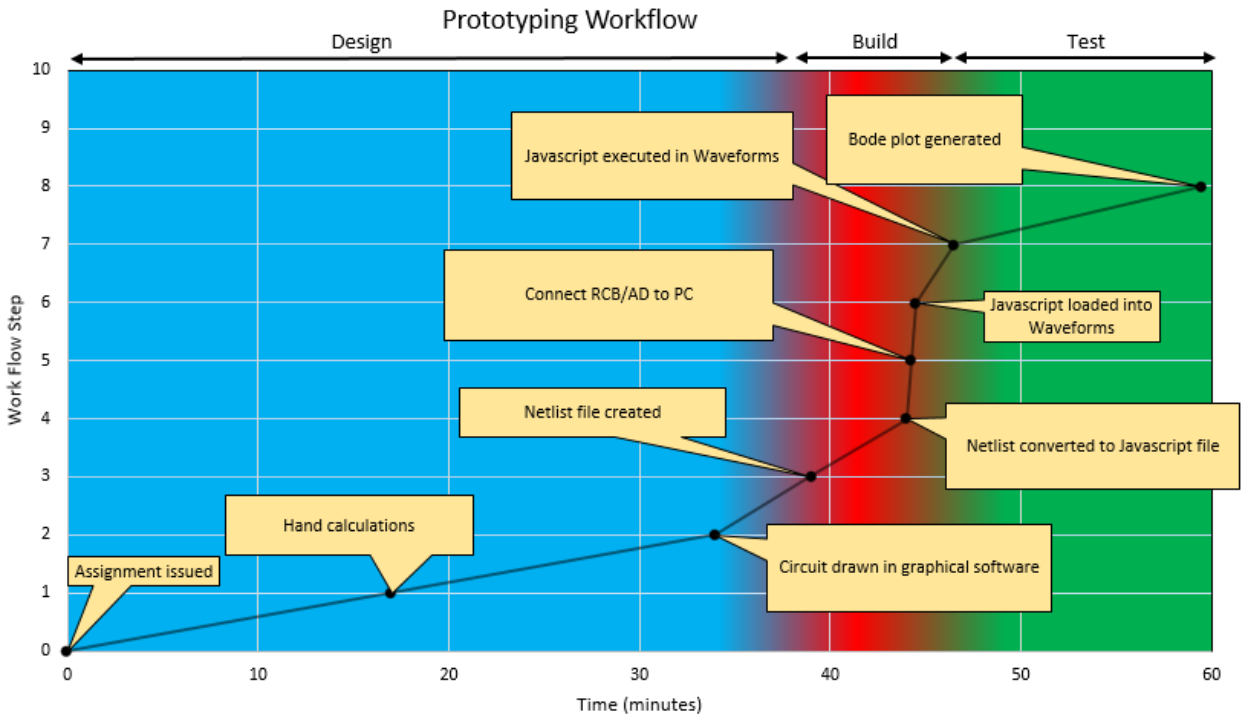


Figure 39 Prototyping workflow time chart

8. Conclusion

A method and system were developed that automated the building of circuits in a time efficient manner. The automation of the building phase allows users to balance their time between the different phases of prototyping including design, build, and analysis. This will enrich their learning experience and enhance their knowledge gained.

The modularity of the system formed by the port board, component card, microcontroller, and software offer users many options to expand and configure the system to their needs. For example, users can tailor the type and number of components they want to employ. Moreover, they can easily prototype circuits based on their own topological designs using any schematic capture software that produces a SPICE netlist.

Overall, the circuit building system addressed many of the guidelines listed by Altalbe [1]. For example, the system supplied reliable and fast circuit building abilities using physical electrical components offering the user a real-time response of real physical circuits. Since the system uses schematic capture software, errors can be easily detected in the design phase. Moreover, the system also provides feedback prior to the building phase of various errors using the circuit design rule checks. Finally, the system is designed as a portable laboratory which provides users with the opportunity to enhance their individual learning.

An evaluation of the system showed that it streamlined the prototyping workflow. Many important capabilities were demonstrated in the evaluation. For example, the system allowed users to remotely collaborate in real-time and balance their time to design, build and analyze a complicated real circuit.

Bibliography

[1] A. Altalbe, "Virtual Laboratory for Electrical Engineering Students: Student Perspectives and Design Guidelines," *The University of Queensland Australia,* vol. I, no. 2, p. 188, 2018.

[2] T. Mohamed, E. Sncristobal, S. Martin, R. Gil, G. Diaz, A. Colmenar, J. Peire, M. Castro, K. Nilson, J. Zackrisson, L. Ha˚ kansson and I. Gustavsson, "Virtual Instrument Systems in Reality (VISIR) for Remote Wiring and Measurement of Electronic Circuits on Breadboard," *IEEE Transactions on Learning Technologies,* vol. 6, no. 1, pp. 60-72, 2013.

[3] E. y. d. C. Departamento de Ingenieria Electrica, "DIEEC," UNED, [Online]. Available: http://www.ieec.uned.es/Labs_remotos.htm. [Accessed 13 May 2022].

[4] Quanser, *Analog Electronics Labs (AELabs) for NI Elvis,* Markham: Quanser, 2017.

[5] D. Zubia and S. F. Almeida Loya, "SELF-CONTAINED RECONFIGURABLE PERSONAL LABORATORY". United States of America Patent US2017/0363678 A1, 21 December 2017.

[6] Ž. R, J. O and I. F, "Analog Circuit Topology Representation for Automated Synthesis and Optimization," *Journal of Microelectronics, Electronic Components and Materials,* vol. Vol. 48, no. No. 1, pp. 29-40, 03 06 2018.

[7] D. P. Reed, "That sneaky exponential-Beyond Metcalfe's law to the power of community building.," *Context magazine,* vol. I, no. 2, 1999.

[8] GeeksforGeeks, "Bubble sort," GeeksforGeeks, 2022 April 22. [Online]. Available: https://www.geeksforgeeks.org/sorting-algorithms/. [Accessed 30 April 2022].

[9] Mitel, "MT8816 Data sheet," *MT8816,* vol. 3, no. 2, pp. 45-50, 1988.

[10] A. Small, "Automated breadboard wiring assembly". United States of America Patent US11079742B2, 03 08 2021.

[11] U. Hernandez and J. Garcia Zubia, "LabsLand," LabsLand, 2020. [Online]. Available: https://labsland.com/pub/docs/experiments/electronics/en/index.html. [Accessed 07 May 2022].

Vita

Cesar Yahir Sanchez Zambrano was born in Chihuahua, Chihuahua, the capital city of the state of Chihuahua, Mexico. He was raised at his born city but moved to Cd. Juarez at the age of 10 years where he continued his basic studies up to his high school. Ever since elementary school Cesar showed a big curiosity in the STEM field and even when his grades were not perfect he managed to stay as the top 3 highest GPA's of his class. During his high school education, he desired to know how education was in the neighbor country the United States of America where after the first tour through The University of Texas at El Paso his desire for being an Electrical Engineer begin. With help of his family he started his education as a first gen student in a foreign country in The University of Texas at El Paso, where he earned his degree in B.S. in Electrical and Computer Engineering in May 2019 with a Cum Laude. During his undergraduate career Cesar started a part time job as a lifeguard to help his family pay his tuition and daily expenses, he often used his free time to work and summer vacations to work and volunteered in a research projects with an available professor. He then started his dual credit, fast track program at the university to start earning credits towards his master degree.

During his senior year Dr. David Zubia recognized his talent and programming skills and offered him a research position in his team. Not even the COVID-19 pandemic was enough to stop him from continue his research project with Dr. Zubia. He is currently working on developing a new prototyping system and pursuing his Master's degree under Dr. Zubia's mentoring in the NanoMIL laboratory at UTEP. As part of the center, an intensive collaboration was performed regularly and resulted in a possible invention disclosure and a funding proposal to continue the project. His project was to program and test a system designed to facilitate EE students access to laboratory practices which was develop in the university's labs.