



UNIVERSITY OF KELANIYA
Department of Physics and Electronics
ELEC-33542 – Research Project

SMART COMPONENT TESTER FOR ELECTRONIC LABORATORY

By

PE/2018/015 – H P E Hettigoda

PE/2018/030 – K A M P Ravihara

PE/2018/025 – S A C Nirmal

PE/2018/004 – R M C S Bandara

March 2023

ACKNOWLEDGEMENT

ABSTRACT

The proper operation of electronic systems relies heavily on electronic components, and their accurate functioning is crucial for achieving maximum performance. The "Smart Component Tester for Electronic Laboratory" project aimed to develop an electronic testing device that could efficiently and accurately test a range of electronic components, including logical ICs, diodes, op amps, and transistors. The device's design was focused on affordability and versatility, making it accessible to a broad range of users, including small labs, educational institutions, and individual hobbyists. The device's user interface was designed to be easy to use, even for users without specialized knowledge, and implemented advanced testing techniques to provide quick and accurate feedback. The testing of the device confirmed its reliability and accuracy, and clear documentation and user manuals were created to assist users in operating the device effectively. In conclusion, this project delivered a valuable tool that simplified the testing of electronic components, enabling efficient troubleshooting of electronic systems. The project succeeded in achieving its objectives, creating an affordable and efficient device that made electronic testing accessible to a broad range of users.

Contents

1. Introduction	4
1.1. Introduction	4
1.2. Objectives	5
1.3. Literature Review.....	6
1.4. Scope and Limitations.....	12
2. Theory	14
3. Methodology.....	14
3.1. Overview	14
3.2. Software Simulations	16
3.3. Hardware Implementation	24
3.4. Optimization	28
4. Results.....	30
5. Discussion and Conclusion.....	36
5.1. Discussion.....	36
5.2. Conclusion.....	41
6. References	43
7. Appendixes.....	45

1. Introduction

1.1. Introduction

Electronic components are an integral part of the functioning of electronic systems and their correct functioning is essential for optimum performance. The Smart Component Tester project was launched to design and develop a device that could simplify and streamline the electronic testing process. This device is designed to test a wide range of electronic components such as logic ICs, diodes, op amps and transistors. It provides an efficient and reliable way to determine if components are working properly, making it easier for users to troubleshoot electronic systems.

One of the unique features of this device is its ability to detect malfunctioning pins of an IC, allowing users to quickly identify and address the issue. This feature was particularly useful for troubleshooting complex electronic systems, where identifying the source of a problem can be challenging. This information is valuable because, even if an IC has only one working gate, it may still be useful for a system that requires only one gate. This reduces e-waste and promotes sustainability.

The device is also equipped with the capability to send data to a local host and plot the characteristic curve of diode. This added functionality provided users with a more comprehensive understanding of the performance of electronic components, enabling them to make informed decisions when troubleshooting or designing electronic systems.

The project team aimed to create a device that is affordable, accessible, and easy to use. The device was designed using readily available components, making it accessible to a broad user base. The device was also thoroughly tested to ensure its reliability and accuracy, and clear documentation and user manuals were created to assist users in operating the device effectively.

1.2. Objectives

The main objective of this project is to create a reliable and efficient device that can test a wide range of electronic components commonly used in electronic laboratories. The device is designed to test the functionality of logical ICs, diodes, op amps, and transistors by applying a series of predefined test signals and analyzing the output response.

Specifically, the project aims to provide the following objectives:

- Testing Functionality: The device should be able to test the functionality of all logical ICs commonly used in electronic laboratories. The tester should identify faulty components and provide an accurate diagnosis of the problem. In the case of malfunctioning ICs, the device should be able to identify the specific faulty pins to aid in troubleshooting.
- Component Testing: The device should be able to test diodes, op amps, and transistors and analyze their output characteristics. It should be able to provide an accurate and reliable measurement of the component parameters such as current, voltage, gain, and frequency response.
- Data Logging: The device should be able to store test data for later analysis and comparison. It should have the capability to transfer test data to a local host for further analysis.
- Data Visualization: The device should have the capability to plot characteristic curve of diode to aid in understanding the behavior of the components.

In summary, the main objective of the project is to create a versatile and reliable device that can test electronic components commonly used in laboratories. The device should be able to provide accurate and reliable measurements, data logging, and visualization capabilities to aid in troubleshooting and understanding the behavior of electronic components.

1.3. Literature Review

In the field of electronics, the testing and analysis of electronic components are an essential part of the research process. There are different types of component testers available in the market, including manual and automatic testers. The manual testers are simple and easy to use, but they are time-consuming, and the testing process is not very accurate. On the other hand, automatic testers are more advanced, accurate, and efficient, but they are expensive.

Several existing solutions are available to test electronic components, with Arduino microcontrollers being a popular choice. However, these testers often only test ICs or specific types of components, such as 3-pin or 2-pin components. Additionally, some devices can test all components but lack the necessary details to provide useful information.

Over the years, several research studies have been conducted to develop smart component testers for laboratory use. A smart component tester is an advanced electronic device that uses microprocessors and software algorithms to automate the testing process. These testers are capable of detecting and diagnosing faults in electronic components quickly and accurately. They can also store test data and generate reports.

Research has identified different techniques to solve this problem, with varying levels of success. Here are some various methods that have been suggested from the literature.

DCA75 - Atlas DCA Pro Advanced Semiconductor Analyzer



Figure 1.1

Source: [Peak Atlas DCA Pro model DCA75 | Peak Electronic Design Limited](#)

The DCA75 will perform the following:

- Display component type (such as N-CH MOSFET, darlington transistor etc).
- Display component pinout (such as drain, source, gate etc).
- Display detailed parameter measurement.
- Plot characteristics curves on PC. (Windows 11, 10, 8, 7, XP)

But this device can only test all kinds of transistors and diodes. So, the device has a limited functionality and fails to test other electronic components besides transistors and diodes.

TES200 Integrated Circuit Tester

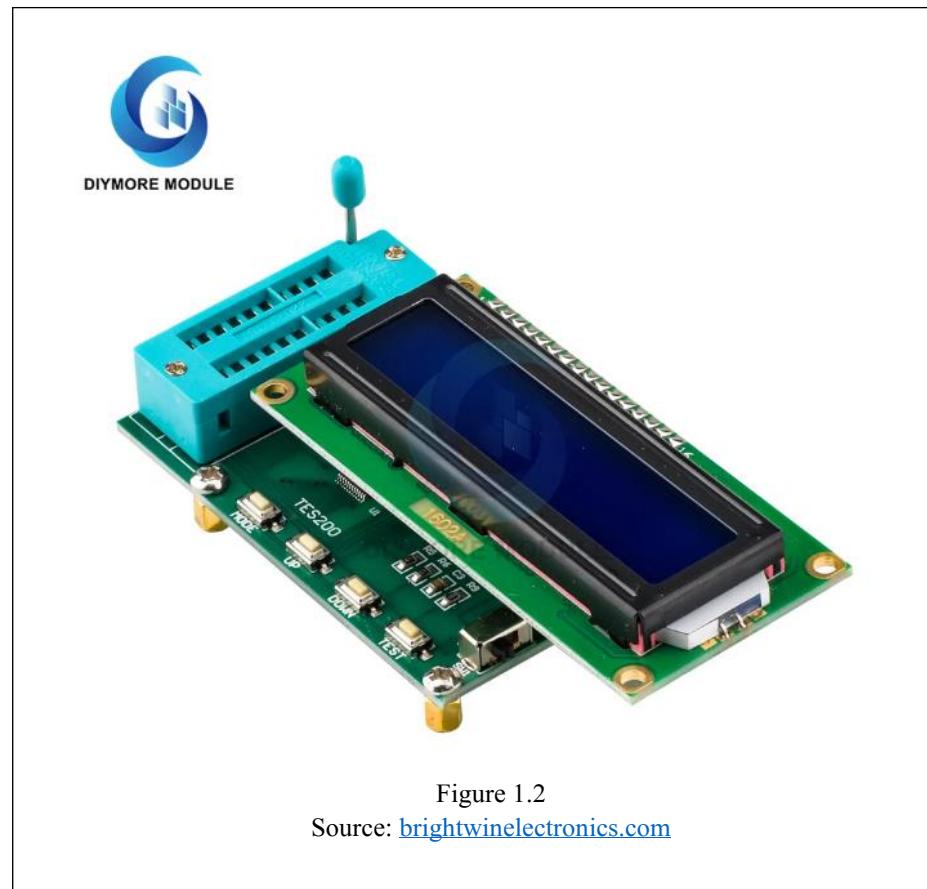


Figure 1.2

Source: brightwinelectronics.com

TES200 Digital IC Chip Tester is a digital integrated circuit chip tester checker with LCD screen, and mainly used to test 74 series and 40 series IC chips. When testing logic integrated circuit chip, it can diagnose and detect which logic is bad. Only by pressing four buttons, user can get the testing results in LCD screen easily. Its power supply is 7-12V wide range, low power consumption, high accuracy. It's LCD screen with back light can show IC models and testing results clearly. It is a low-cost essential IC testing kits and a DIY kit for technicians.

The device's function is limited to determining whether the IC is functional or not. Therefore, it cannot test any other electronic components apart from ICs, implying that its functionality is restricted.

MK-168 Transistor Tester



Figure 1.3
Source: [MK-168 Transistor - AliExpress Tools](#)

Features:

- Micro-controller with 8MHz external crystal, better measurement accuracy.
- Automatically detect NPN, PNP bipolar transistors, N-channel and P-channel MOSFET, JFET, diodes, dual diode, thyristor small power unidirectional and bidirectional thyristor.

But the device's testing capabilities are restricted to transistors, capacitors, resistors, and diodes, implying that it cannot test any other electronic components. Therefore, its functionality is limited.

TSH-06F Integrated Circuit Tester

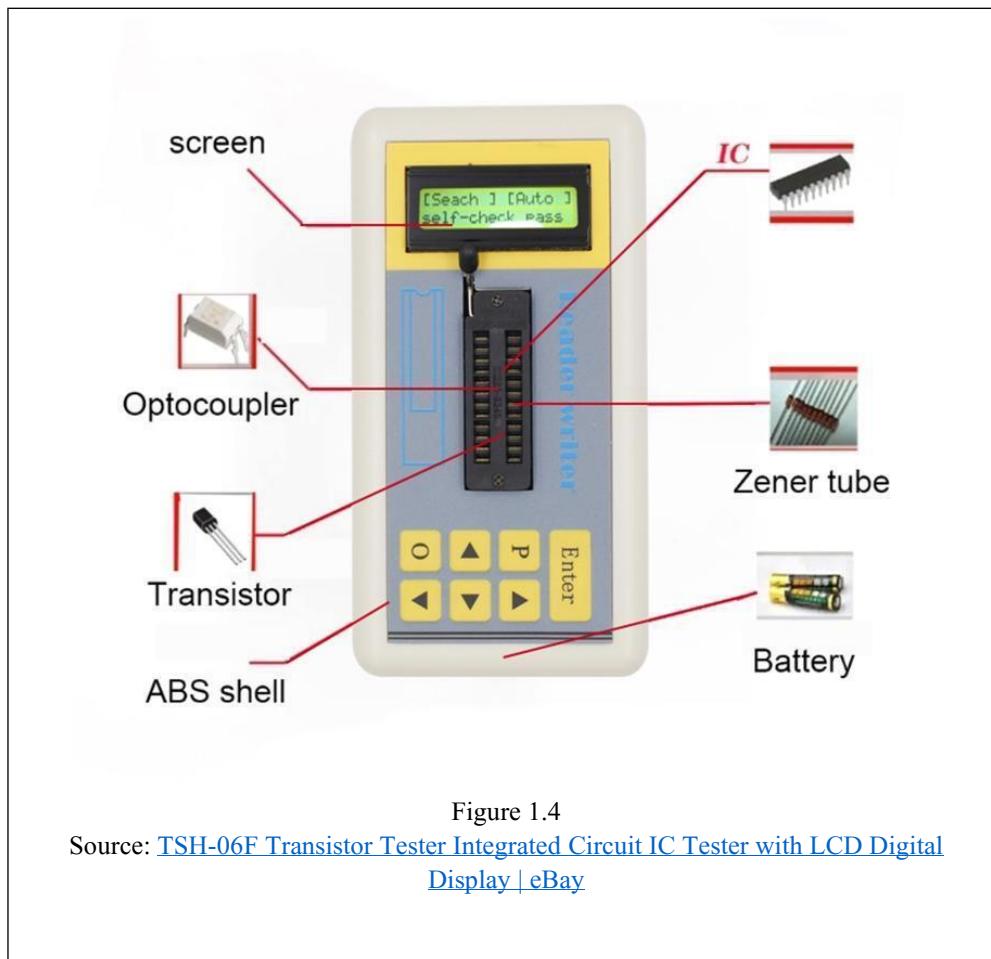


Figure 1.4

Source: [TSH-06F Transistor Tester Integrated Circuit IC Tester with LCD Digital Display | eBay](#)

The multi-functional integrated circuit tester is a professional instrument designed for microelectronic engineers and maintenance personnel.

Features:

- Tests ICs, transistors, diodes, thyristors, resistors and capacitors.
- Automatically detect NPN and PNP transistors
- Automatic identification of the transistor pinout
- Automatic shutdown if there is no operation after 60s.

Our device's main competitor at present is this one. But also, this device lacks the ability to plot characteristic curves and identify malfunctioning pins on a logic IC.

Overall, the literature review suggests that there is a need for smart component tester in the laboratory setting. These devices offer several advantages over manual testers, including accuracy, efficiency, and the ability to store and analyze test data. Therefore, the development of a smart component tester for laboratory use is a valuable contribution to the field of electronics.

Here is a comparison of our device with the currently available devices.

Features	Our Device	Atlas DCA Pro	TES200 IC Tester	MK-168 Component Tester	TSH-o6F Integrated Circuit (IC) Tester	TES200 Digital Universal IC Tester
Detect and identify component automatically.	✓	✓	✗	✓	✓	✗
Test transistors	✓	✓	✗	✓	✓	✗
Test diodes	✓	✓	✗	✓	✓	✗
Test thyristors	✗	✓	✗	✓	✓	✗
Test resistors	✗	✓	✗	✓	✓	✗
Test capacitors & measure capacitance	✗	✓	✗	✓	✓	✗
Characteristic curves	✓	✓	✗	✗	✗	✗
Test ICs	✓	✗	✓	✗	✓	✓
Show which pins are not working	✓	✗	✗	✗	✗	✗
Extra plugs to test components	✓	✗	✓	✓	✗	✓
Large and digital display	✓	✓	✓	✗	✓	✓
cost	Rs. 12000 (Prototype)	Rs. 31 879 (Market price)	Rs. 18109 (Market price)	Rs. 11,606 (Market price)	Rs. 41 988 (Market price)	Rs. 18 108 (Market price)

1.4. Scope and Limitations

Scope:

The scope of the smart component tester is to provide an efficient and reliable method for testing electronic components in the laboratory setting. The device is designed to test all types of logical ICs, diodes, op amps, and transistors. It can detect any malfunctioning pins of the ICs and provide the results for further analysis. The device is intended to be user-friendly and easily operable for both novice and experienced users.

The smart component tester is suitable for use in electronic laboratories, research and development centers, and educational institutions. It will be useful for electronic engineers, technicians, and hobbyists who require an efficient tool to test their electronic components. The device is portable and can be easily carried to different locations, making it ideal for fieldwork.

The device's ability to send data to a local host and plot the characteristic curve of diode adds to its usefulness in electronic testing. This feature provides a graphical representation of the characteristics of these electronic components, making it easy to analyze and interpret their behavior.

The scope of the project also includes designing and implementing a user interface that is easy to navigate, intuitive, and displays the test results clearly. The user interface will be designed to be responsive and allow for customization of the test parameters, providing users with greater control over the testing process.

The project also aims to make the device cost-effective by using readily available and affordable components. The device will be designed to be low maintenance, reducing the cost of ownership and ensuring its longevity. The project scope also includes testing and validating the device's performance to ensure its accuracy and reliability in testing electronic components.

Limitations:

- Op-amp pin layouts:

Op-amps come in various types and have different pin layouts. Since we have to provide + and - voltage to operate the op-amp, the device is limited to testing only the most commonly used 741/743 op-amps. This limitation may be significant for laboratories that use op-amps with different pin layouts.

- Limitations of testing op-amps:

The device can determine whether the op-amp is functioning or not, but it cannot calculate the gain due to the inability of the Arduino to output analog wave signals. This limitation may be problematic for laboratories that require more detailed testing of op-amps.

- Complexity:

Adding all op-amp testing methods using relays would make the device more complicated and larger in size. This may be a significant limitation for laboratories with limited space or resources.

- Arduino limitations:

The device relies on the capabilities of the Arduino, which has its own limitations in terms of memory, processing speed, and input/output capabilities. These limitations may affect the accuracy and speed of the device.

2. Theory and Methodology

2.1. Theory

The smart component tester is based on the fundamental principles of electronic circuit theory. Electronic components such as diodes, transistors, operational amplifiers, and logic integrated circuits are essential components of electronic circuits. These components are used to perform various functions, such as amplification, switching, and signal processing, in electronic systems.

The device works by applying a test signal to the component under test and analyzing the response of the component. For example, in the case of a diode, the device applies a test signal across the diode and measures the resulting voltage and calculating the current flow. By analyzing the current-voltage characteristics of the diode, the device can determine whether the diode is working correctly or not.

Also, in the case of a logical IC, the device will check whether the output is as expected for all possible inputs. If the output does not match the expected values, the device will display whether it is good or bad and if it is bad, indicate which pins of the IC are malfunctioning.

Similarly, in the case of a transistor, the device applies a test signal to the transistor's base and analyzes the resulting output signal. By analyzing the current-voltage characteristics of the transistor, the device can determine whether the transistor is working correctly or not.

To test the op-amp, the device applies a digital signal to the op-amp to test its functionality as a simple inverter, and by observing whether the output matches the desired result, the device can determine if the op-amp is operational or not.

The device also uses advanced techniques, such as curve fitting and data analysis, to plot the characteristic curve of diode. By plotting these curves, the device can provide valuable information about the component's performance, such as the breakdown voltage, current gain, and saturation voltage.

2.2. Overview of the Methodology

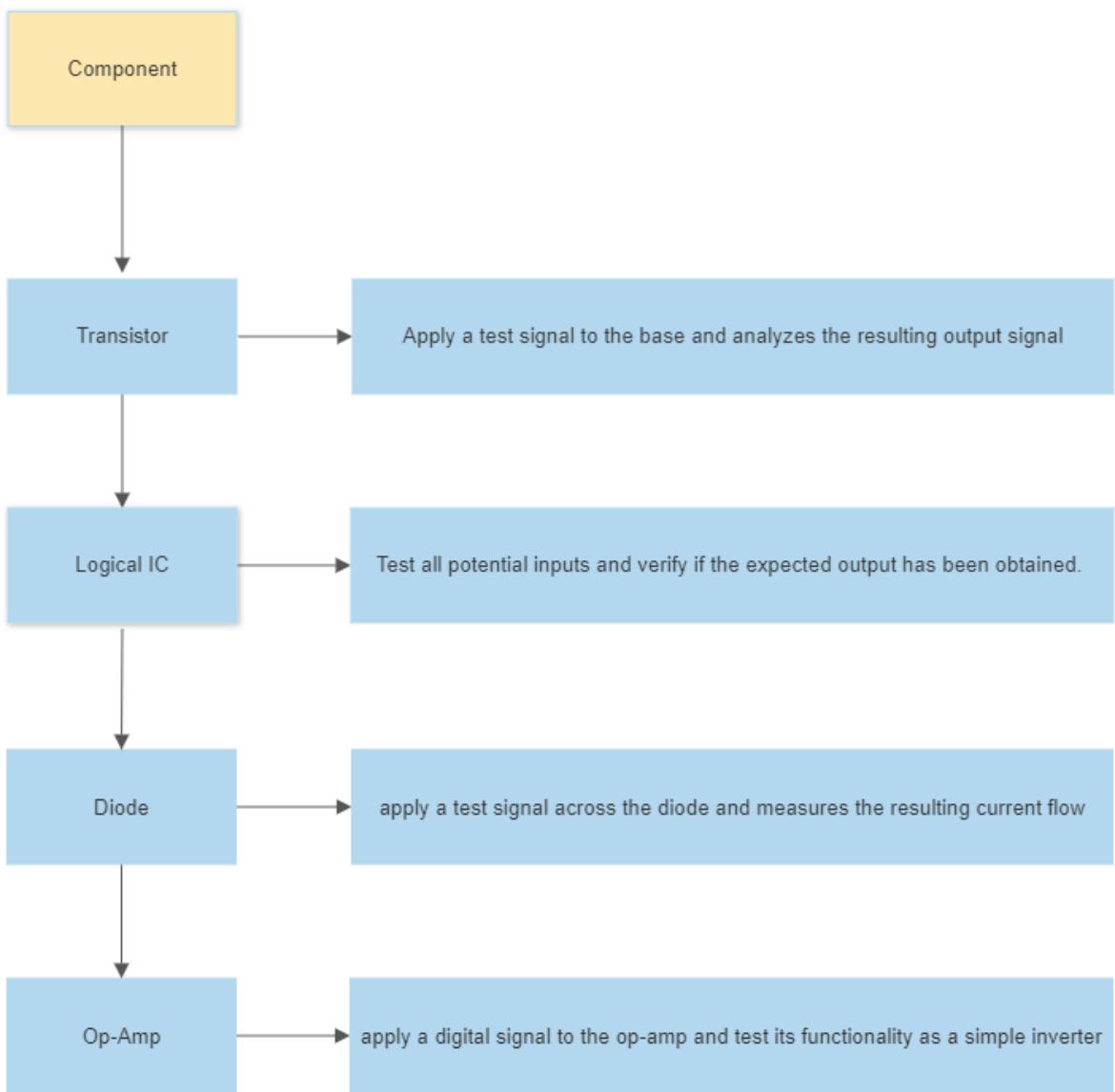


Figure 2.1

2.3. Software Simulations

Tinkercad is an online platform for creating 3D designs, simulations, and models. The platform is designed to be accessible to both beginners and professionals, and it has a wide range of features that make it suitable for a variety of applications. Our team found Tinkercad to be an ideal platform for creating simulations to test the fundamental operations of our project, due to its ease of use and versatile functionality.

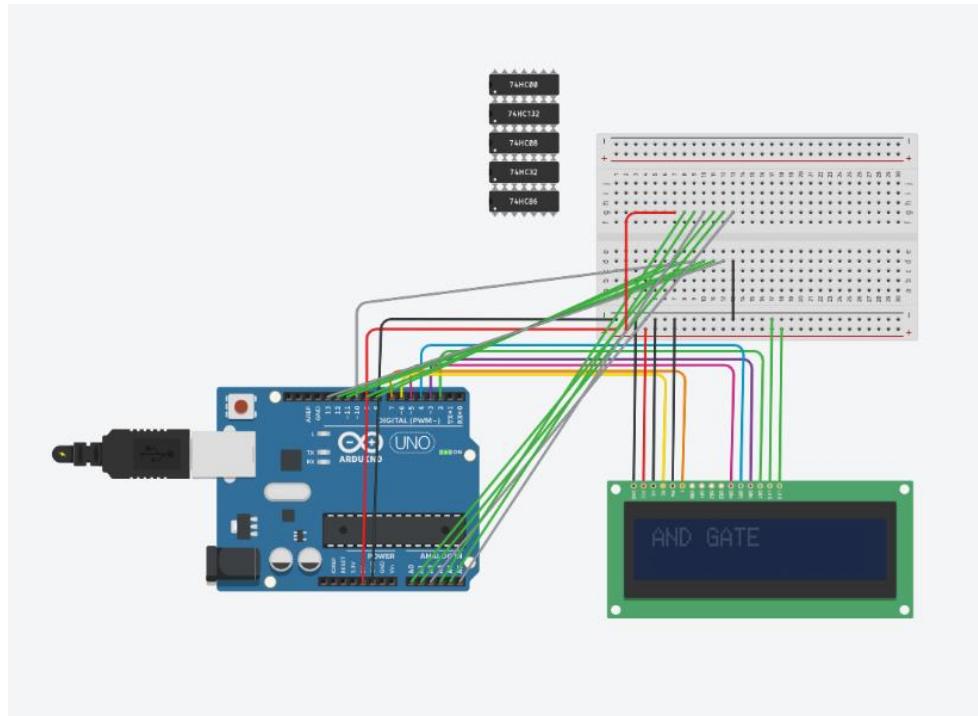


Figure 2.2
IC testing simulation (<https://www.tinkercad.com/>)

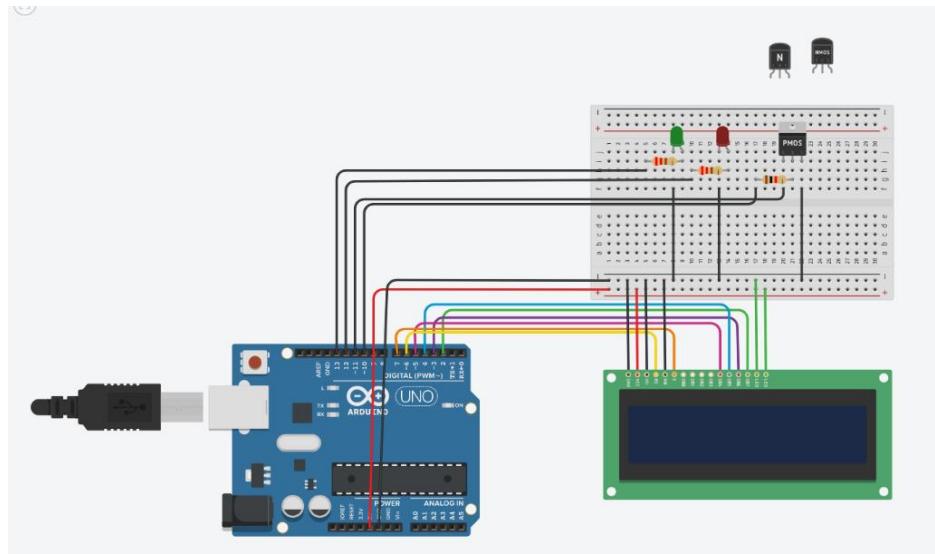


Figure 2.3
Transistor testing simulation (<https://www.tinkercad.com/>)

Following our decision to showcase the characteristic curve of diode in a PC via Wi-Fi, we made the choice to switch from Arduino Mega to ESP32 microcontroller. As “Tinkercad” did not offer ESP32 microcontroller for use on their platform, we relocated our simulations to www.wokwi.com.

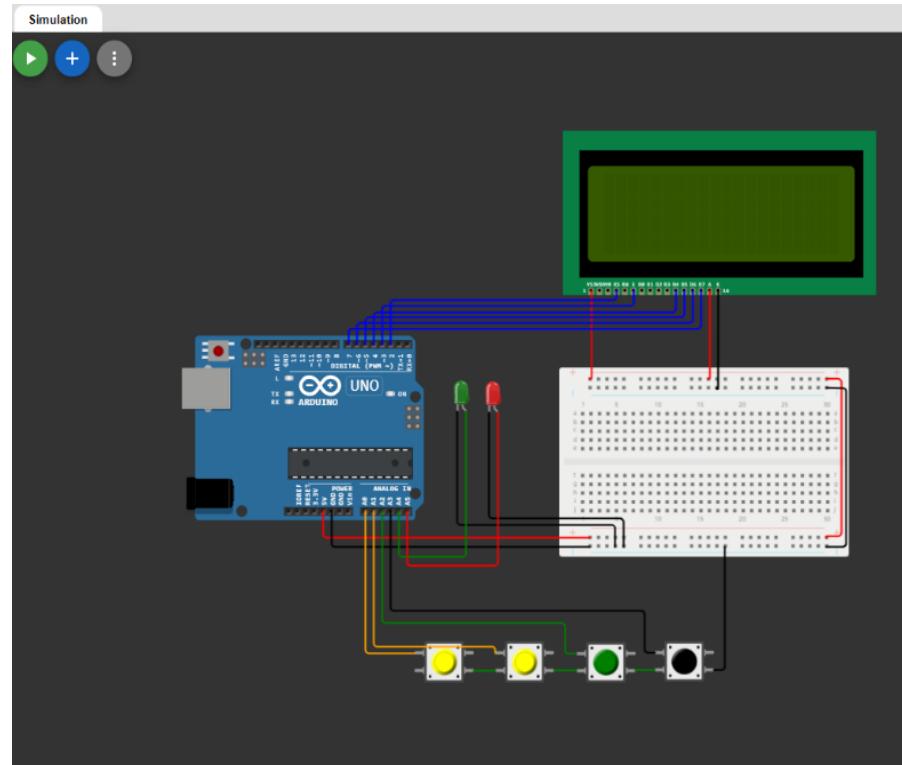


Figure 2.4
Completed setup using Arduino Uno (<https://www.wokwi.com/>)

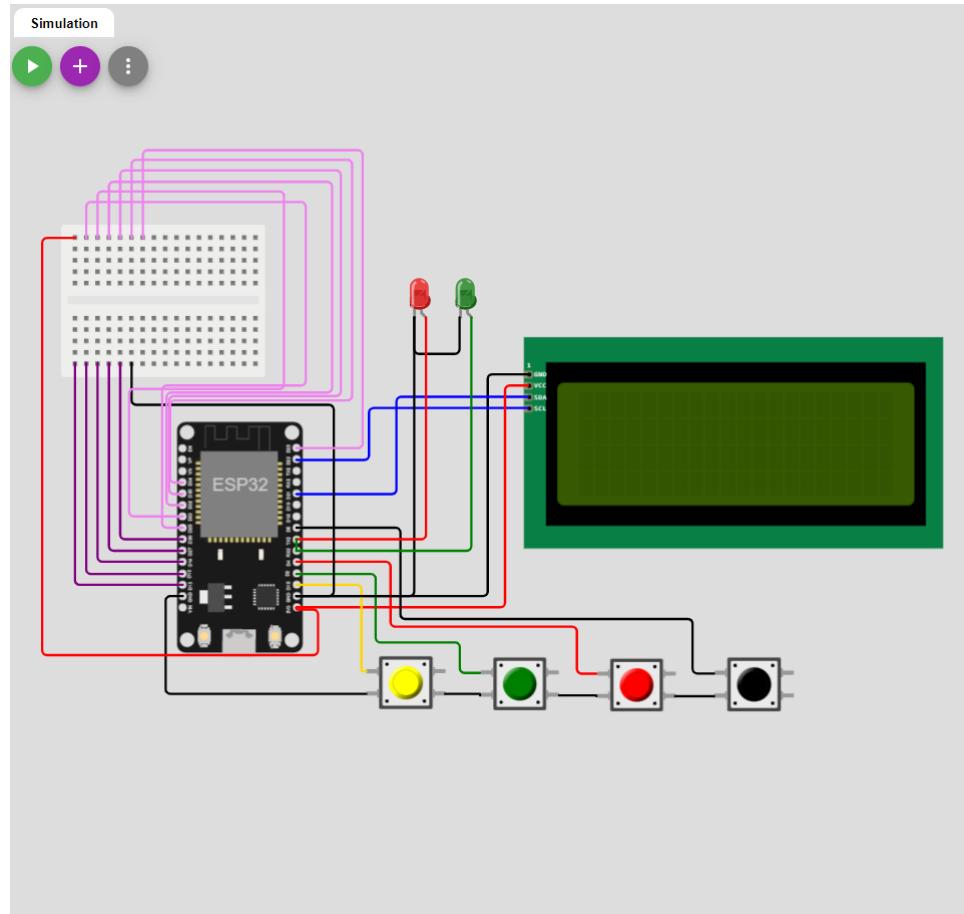


Figure 2.5
Completed setup using ESP32 (<https://www.wokwi.com/>)

After making the switch from Arduino Mega to ESP32 microcontroller and moving our simulations to www.wokwi.com, we encountered an issue during testing. It was discovered that the display we were using was not compatible with the ESP32 microcontroller as it required a voltage of 5V, whereas the board's maximum voltage output was only 3.3V.

As a result, we had to reassess our options and decided to switch back to the Arduino Mega in order to continue with our project. The Arduino Mega was capable of providing the required 5V output to power the display, which made it the more suitable choice for our needs.

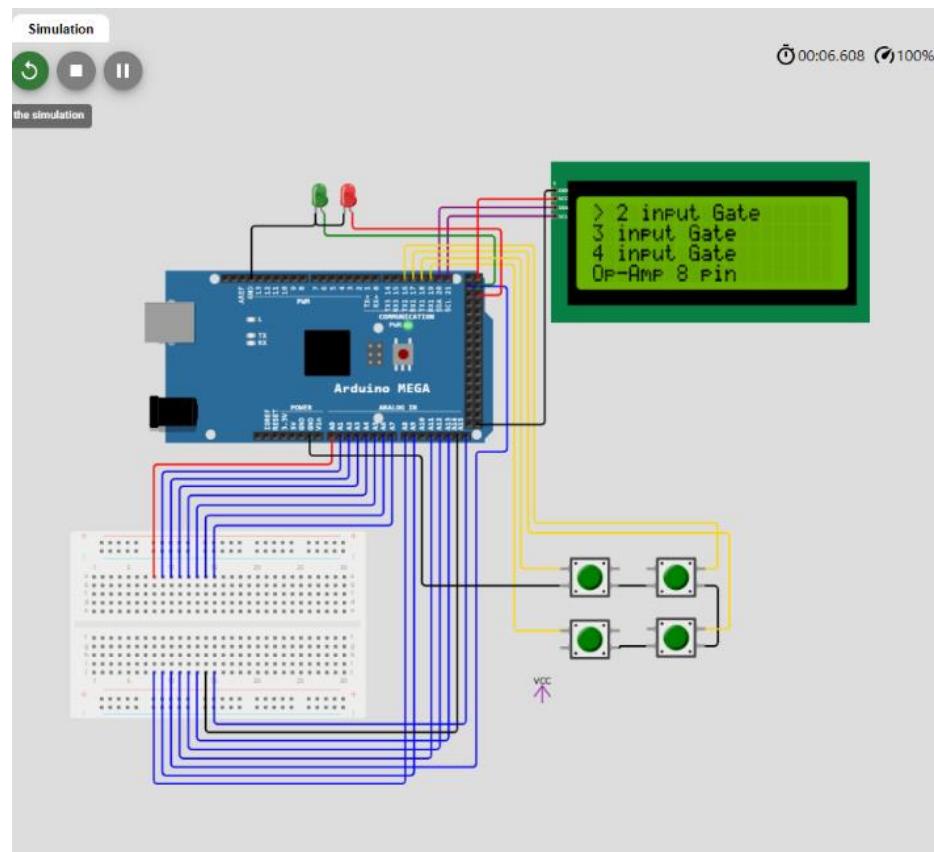


Figure 2.6
The Arduino Mega microcontroller was used to simulate the completed project in the final stage.

Despite the setback we experienced with the display's compatibility with the ESP32 microcontroller, we remained determined to achieve our goal of displaying the characteristic curve of diode in a PC via Wi-Fi.

To accomplish this, we decided to develop the characteristic curve plotting feature in a separate circuit, utilizing the ESP32 microcontroller. We began by designing and building the circuit, carefully selecting the necessary components and ensuring compatibility between them.

For that we used XAMPP for our database deploying to a live server. XAMPP is free and open source software that allows users to run and test web applications locally on their own computer. It making easy for us to create and manage databases locally because XAMPP includes MySQL database server.

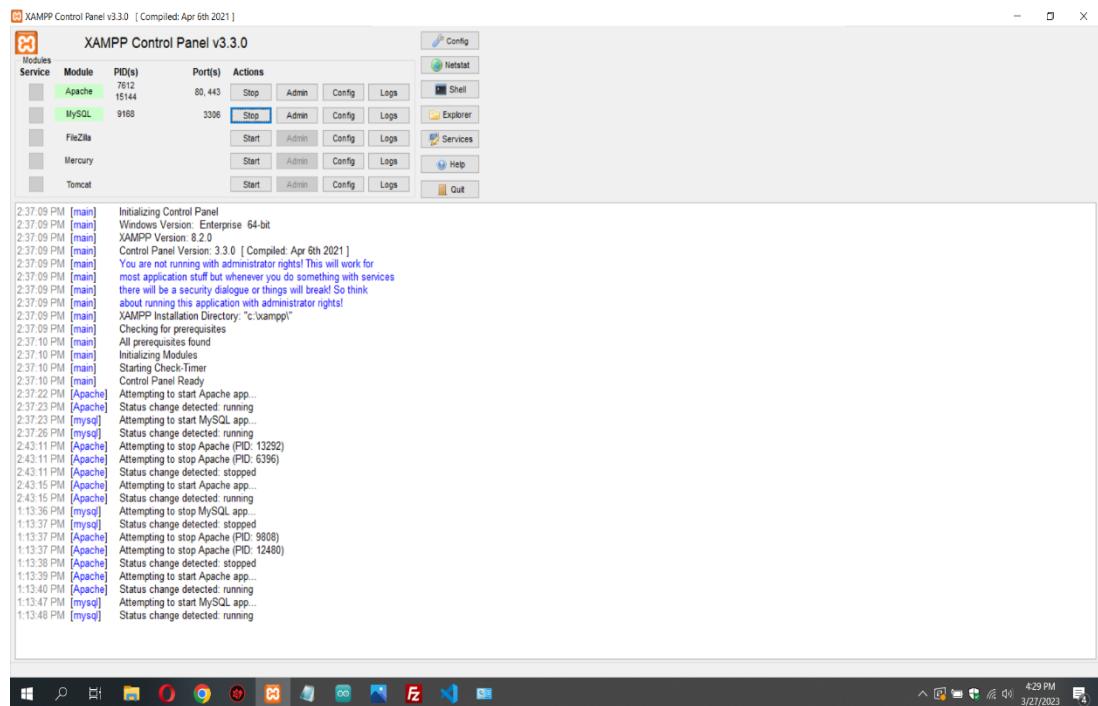


Figure 2.7

We used **Visual Studio** to develop HTML, CSS and PHP codes which is a powerful Integrated Development Environment (IDE) created by Microsoft that supports a wide range of programming languages, platforms, and technologies. Visual Studio provides extensive support for web development, including tools for building websites, web applications, and cloud services.

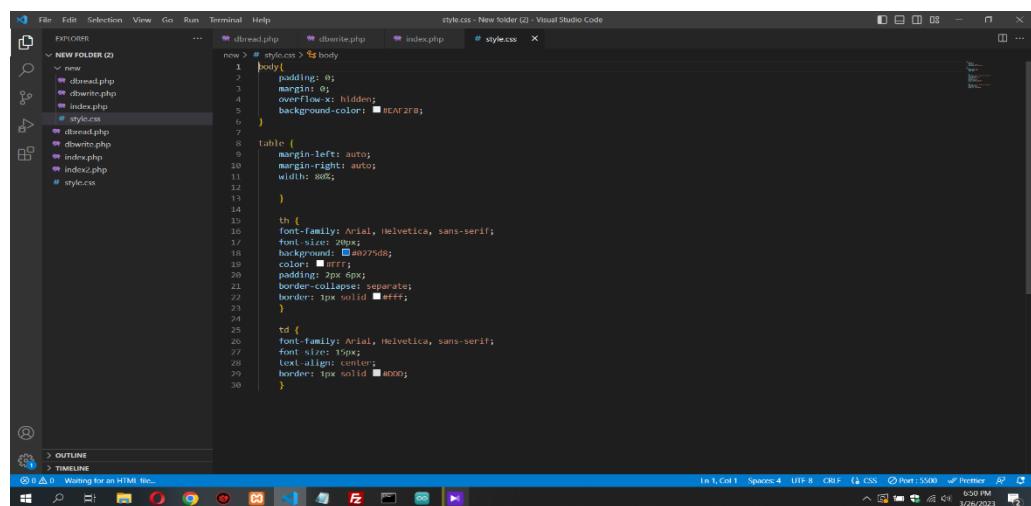


Figure 2.8

The screenshot shows the Visual Studio Code interface with the file 'index.php' open. The code is a JavaScript script for a chart. It includes imports for 'dbread.php', 'dbwrite.php', and 'style.css'. The main part of the code defines a chart object 'myChart' with various properties like 'type: 'line'', 'data: [{}]', and 'options: { scales: { xAxes: [{}], yAxes: [{}]} }'. The code uses 'graphData.map' to generate data points.

```

<?php
require('dbread.php');
require('dbwrite.php');
require('style.css');

new > index.php > html > body > script > yValues > graphData.map() callback
new > index.php > style.css
new > dbread.php
new > dbwrite.php
new > index.php
new > index2.php
# style.css

now > index.php > body > script > yValues > graphData.map() callback
101 // get the canvas element
102 var ctx = document.getElementById('myChart').getContext('2d');
103 // Create a new Chart object
104 var myChart = new Chart(ctx, {
105   type: 'line',
106   data: [
107     {
108       labels: graphData.map(function(val) { return val[0]; }),
109       data: graphData.map(function(val) { return val[1]; })
110     }
111   ],
112   options: {
113     scales: {
114       xAxes: [
115         {
116           display: true,
117           scaleLabel: {
118             display: true,
119             labelString: 'Data Points'
120           }
121         }
122       ],
123       yAxes: [
124         {
125           display: true,
126           scaleLabel: {
127             display: true,
128             labelString: 'Values'
129           }
130         }
131       }
132     }
133   });
134 });

</script>

```

Figure 2.9

Then we created and managed tables within our databases using PhpMyAdmin which is a free and open-source web-based application that provides a graphical user interface (GUI) for managing and administering MySQL databases. In addition to using PhpMyAdmin to import data from our database, we also modify the table structure by adding or removing columns and changing data types. This feature is particularly useful when transferring data from one database to another or when creating backups of important data.

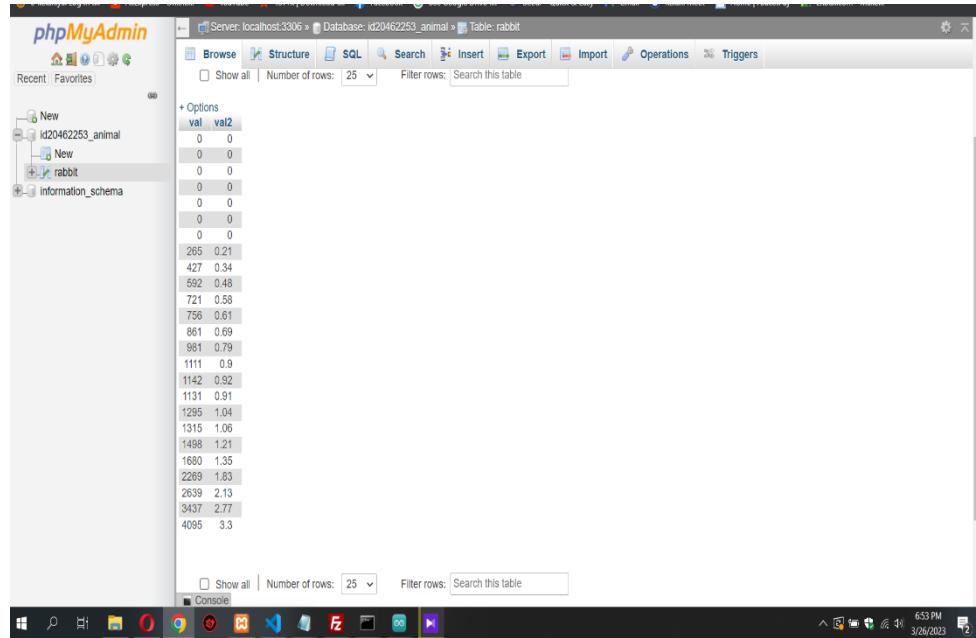


Figure 2.10

We have used **000webhost.com** to create an online free host which is a web hosting service provider that offers free web hosting services.

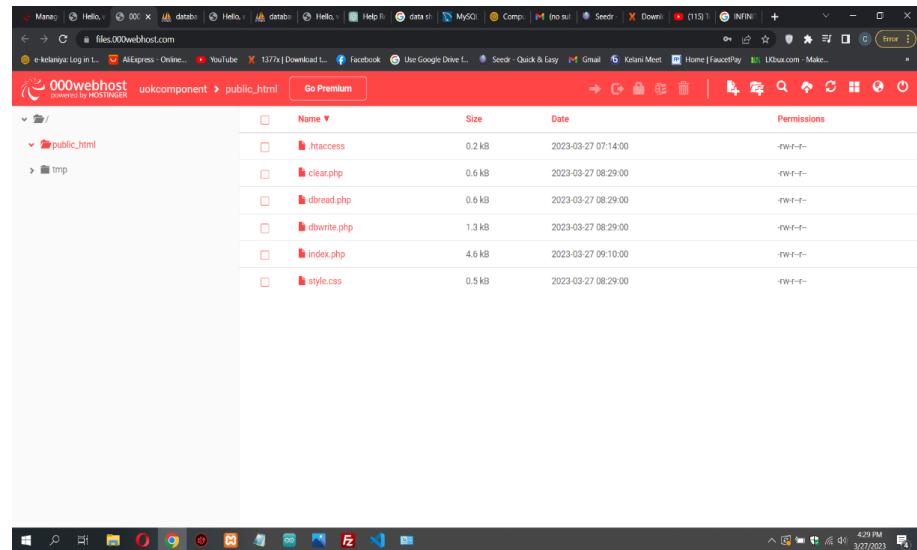


Figure 2.11

Once the pc connector circuit is completed, we integrate it into our existing project, using an ESP32 microcontroller to communicate between the circuit and the computer. With the circuit in place, we were able to successfully plot the characteristics of the diodes in real time as needed.

Afterwards, there was a problem with the size of the device as using the Arduino Mega would make the circuit board more complex. As a solution, we decided to switch to the MEGA 2560 Pro Mini instead of the Arduino Mega, which was a smaller and more compact option.

Then we started developing the PCB layout using easyEDA (<https://www.easyyeda.com>), a web-based PCB design tool.

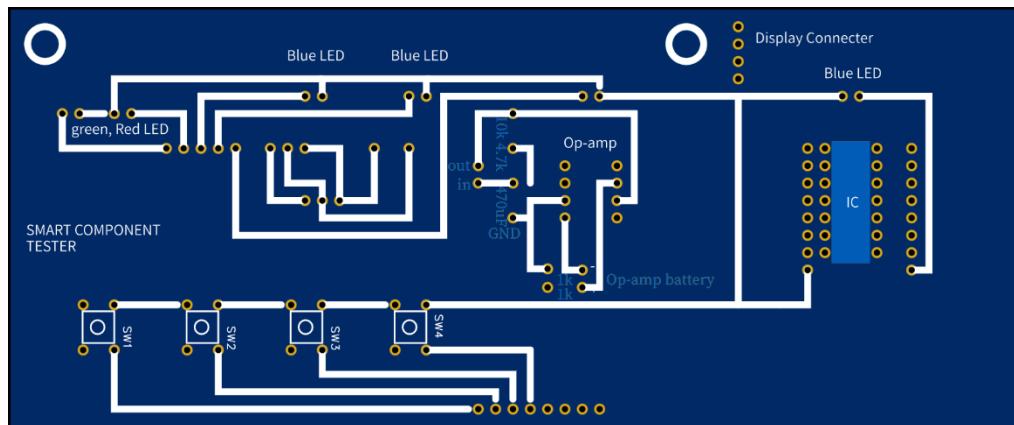


Figure 2.12
PCB layer 1

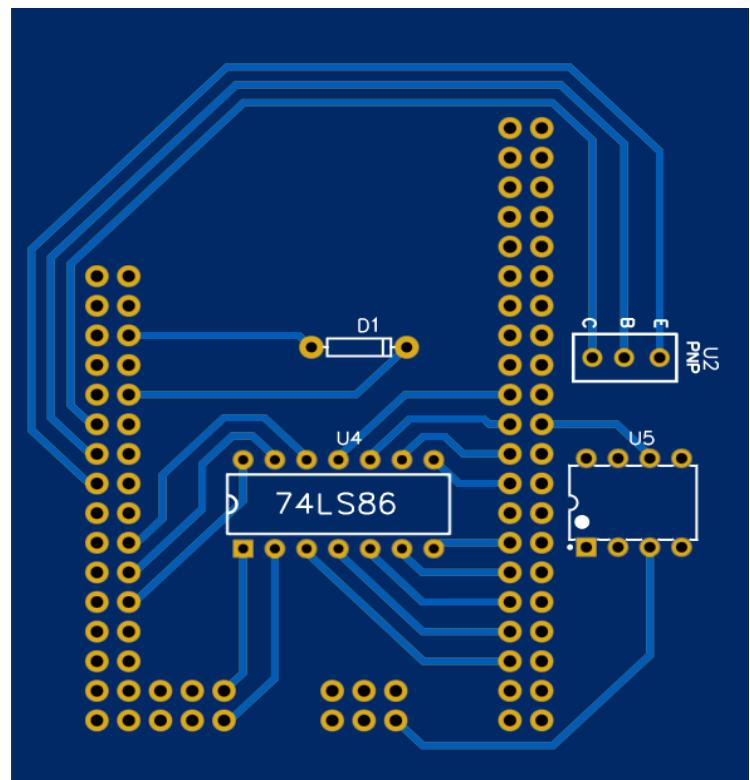


Figure 2.13
PCB layer 2

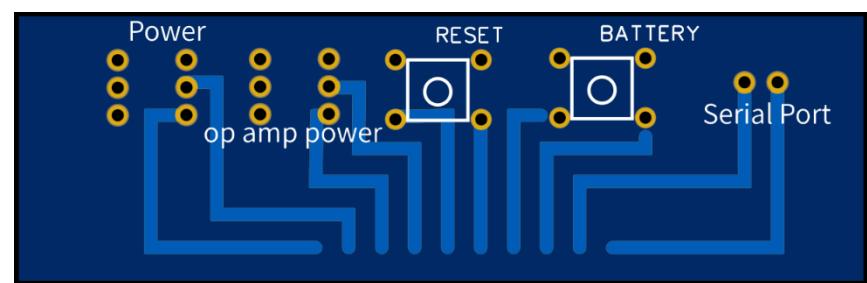


Figure 2.14
PCB layer 3

2.4. Hardware Implementation

Step 1

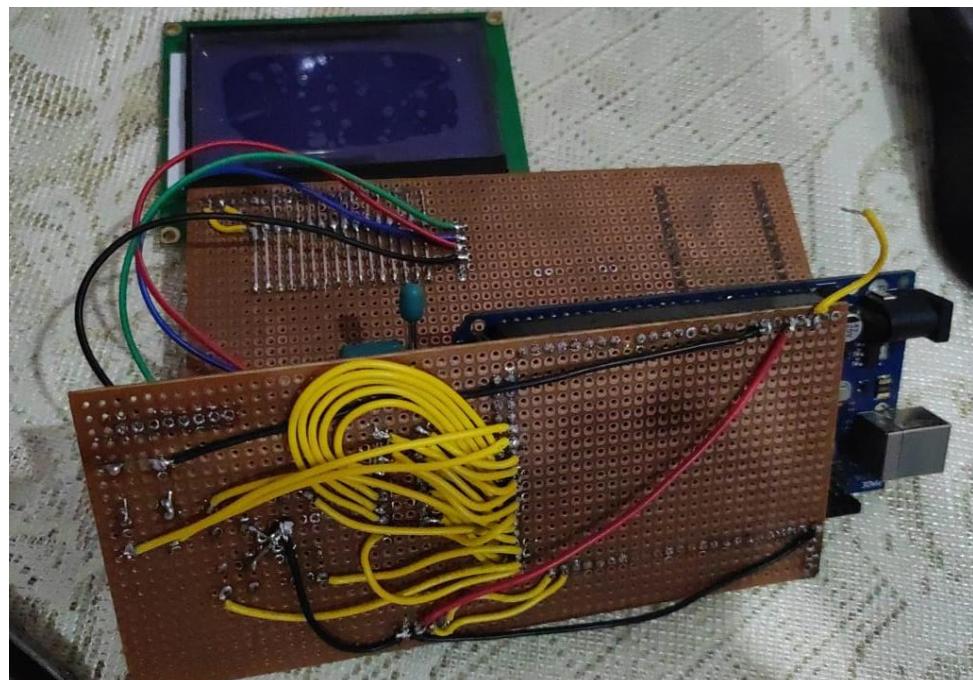


Figure 2.15

A basic hardware setup was created to test the functionality of logical ICs. The setup utilized an Arduino MEGA board, an I2C module, a 128 x 64 graphical LCD display, 4 buttons and a ZIF socket. All the pins of the IC were connected to the analog pins of the Arduino MEGA board. This setup allowed for easy testing and evaluation of the IC's logic circuits, as well as providing a graphical display of the results on the LCD screen.

Step 2

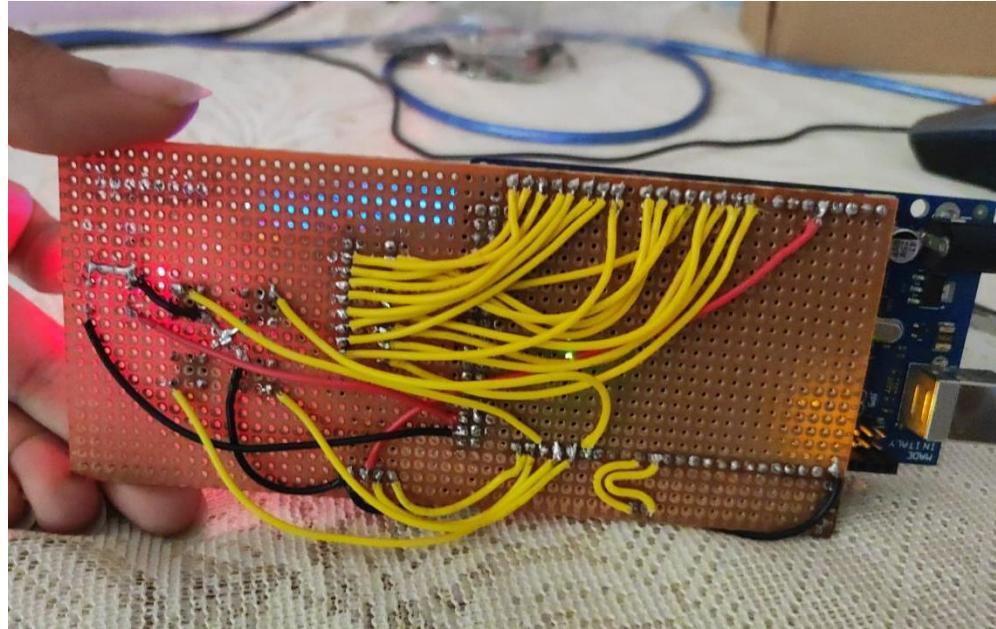


Figure 2.16

In the previous implementation (Figure 2.6), all the pins of the logical IC were connected to the analog pins of the Arduino MEGA board for testing the IC's functionality. However, in this new implementation, the digital pins of the Arduino MEGA board are used instead of the analog pins. This means that the connections between the IC and the Arduino MEGA board are made using digital pins instead of analog pins. This may require a different wiring scheme, as well as different coding and programming for the Arduino MEGA board to communicate with the IC properly.

Step 3

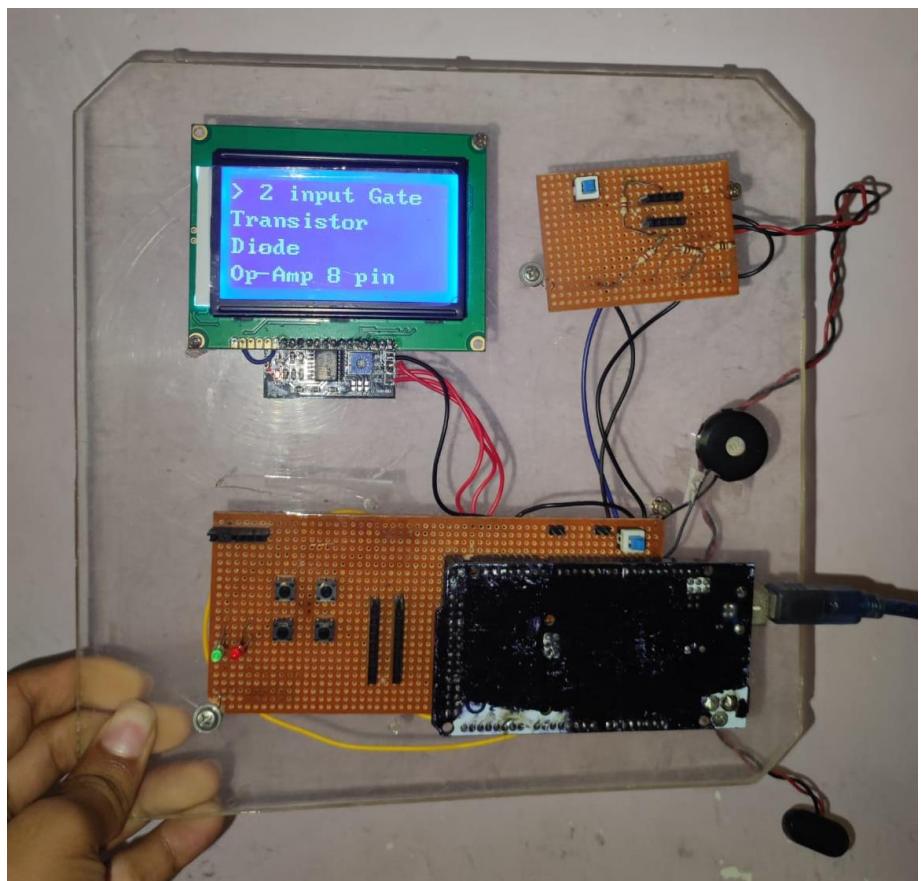


Figure 2.17

At this stage, the prototype component tester is capable of verifying the functionality of various electronic components including logical ICs, transistors, diodes, and operational amplifiers.

Step 4

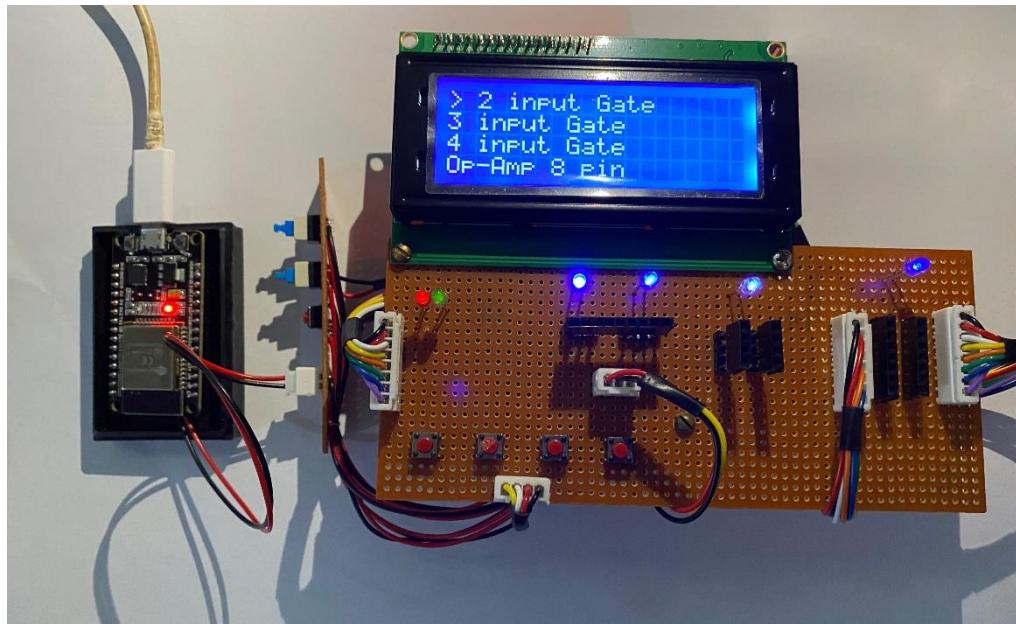


Figure 2.18

This is the final stage of the Smart component tester and this model is capable of testing logical ICs, transistors, diodes and op amps. Also, this model can detect any malfunction pins of a logical IC and plot the characteristic curves of diode. The setup is mainly powered by Arduino MEGA 2560 PRO MINI and the PC connector is utilized by ESP32.

2.5. Optimization

- Micro Controller
 - We replace the Arduino Mega 2560 with identical board Arduino Mega Pro mini 2560V to reduce the space and make device more portable.

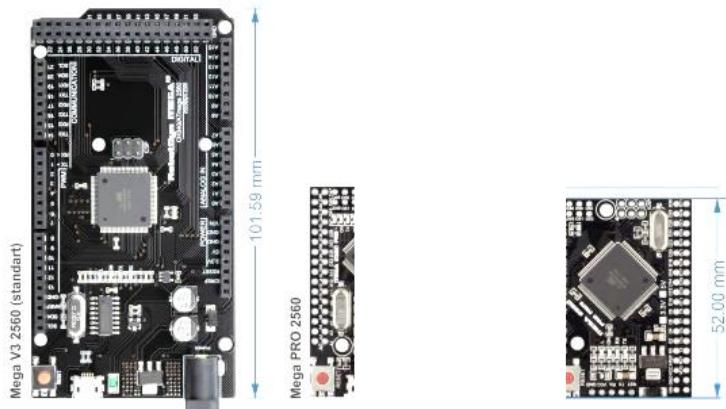


Figure 2.19

- We added a buzzer,
 - To give user to feel the buttons were pressed.
 - To indicate Component not functioning well by three beep sound.
 - One beep sound Component is functioning well
 - Lagging beep for indicating serial communication.
- LED indicator
 - Added blue led indicator to identify the which component user selected.
 - Added green led to indicate component is functioning well and red to if component has an error.



Figure 2.20

- User Interface and Device Selection
 - Added simple options to select which component user wanted to test, when user need to check IC jst need to select which IC wanted to test.

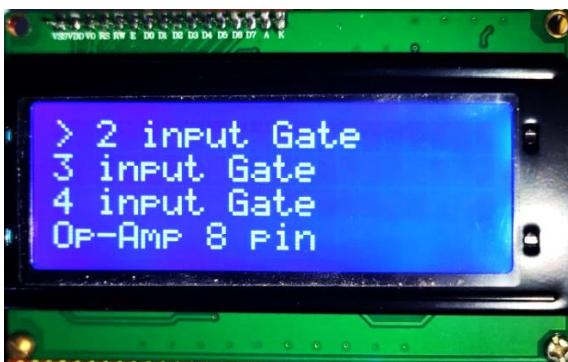


Figure 2.21

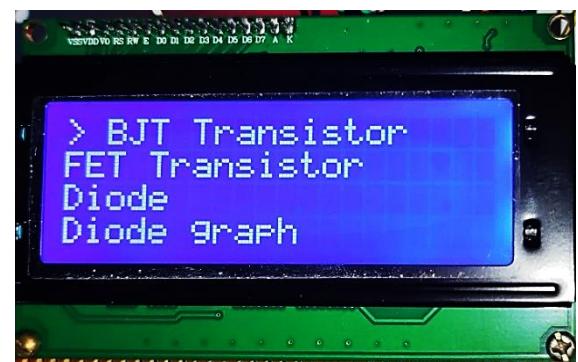


Figure 2.22

- Placed testing results the way that user can easily understand.



Figure 2.23



Figure 2.24



Figure 2.25

3. Results

Display outputs

- IC testing outputs



Figure 3.1

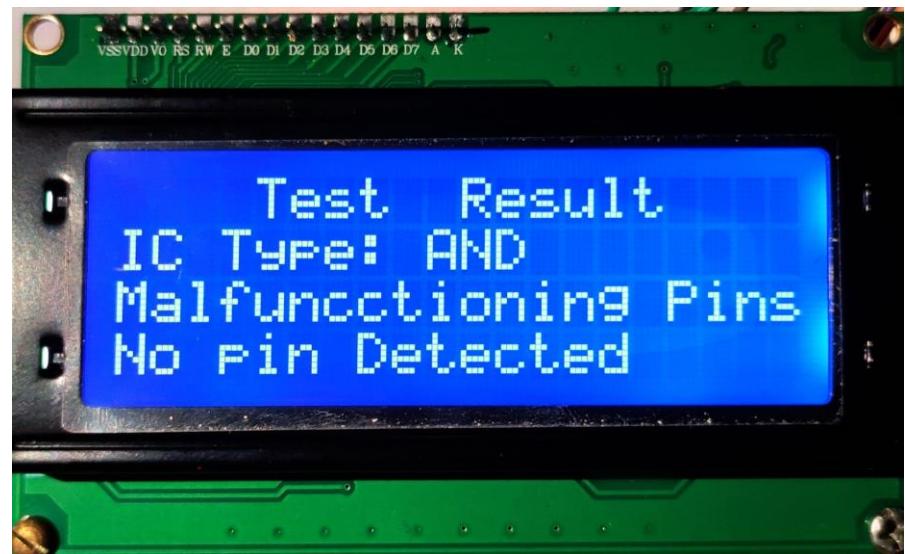


Figure 3.2

- Diode testing outputs



Figure 3.3

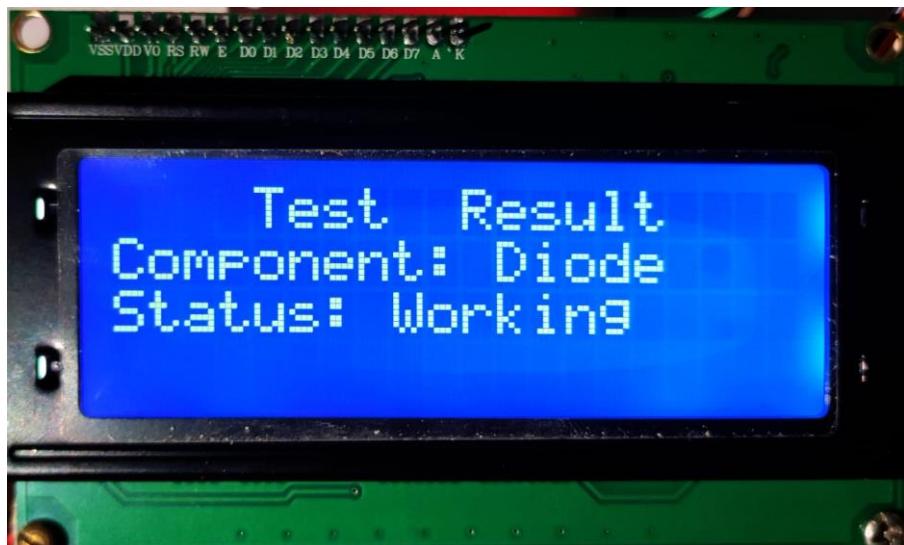


Figure 3.4

- Op amp testing outputs



Figure 3.5



Figure 3.6

- Transistor testing outputs

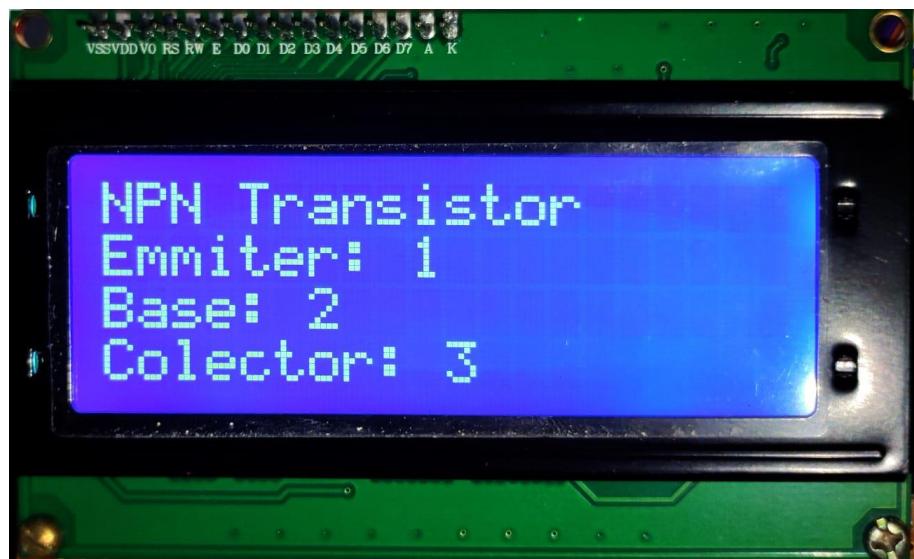


Figure 3.7



Figure 3.8

- Error message outputs



Figure 3.9



Figure 3.10

PC connector outputs

- Test results

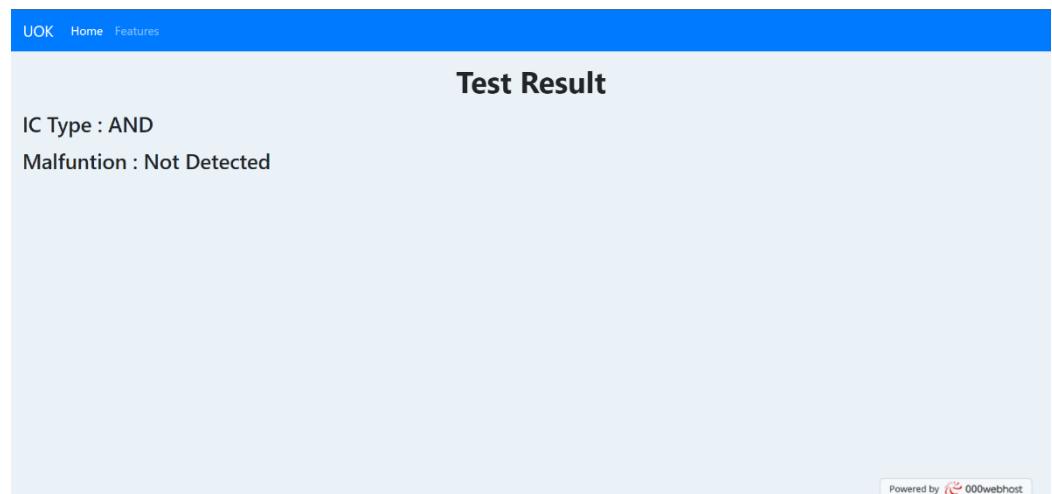


Figure 3.11

- Characteristic curve

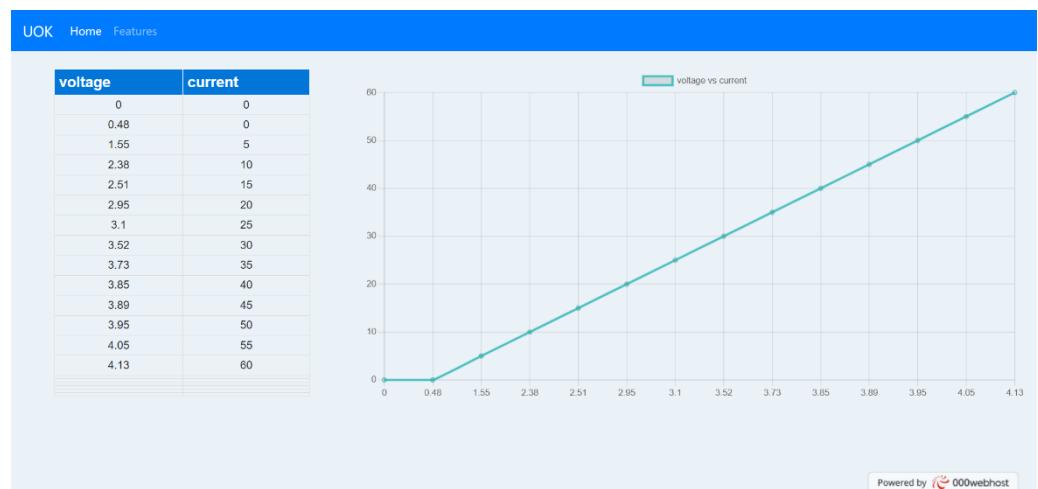


Figure 3.12

4. Discussion and Conclusion

4.1. Discussion

In the initial stages of the project, we began developing the circuit using an Arduino Uno microcontroller. We chose this particular microcontroller due to its versatility and ease of use, as it is widely used in a variety of electronic projects.

Once the circuit had been developed, we began the process of unit testing for each component. This involved testing each individual component of the circuit to ensure that it was functioning correctly and as expected. By conducting unit testing in this way, we were able to identify and address any potential issues or problems with the circuit, prior to integrating it into the larger project.

During the unit testing phase of our project, we began by testing the integrated circuits (ICs) that we planned to use in our circuit. Our first step in this process was to understand the pin layout of different ICs. This was important because the correct connection of pins is crucial for the IC to function properly. For an example, the pin layouts of 7400 Quad 2-input NAND gate and 7402 Quad 2-input NOR gate has given below.

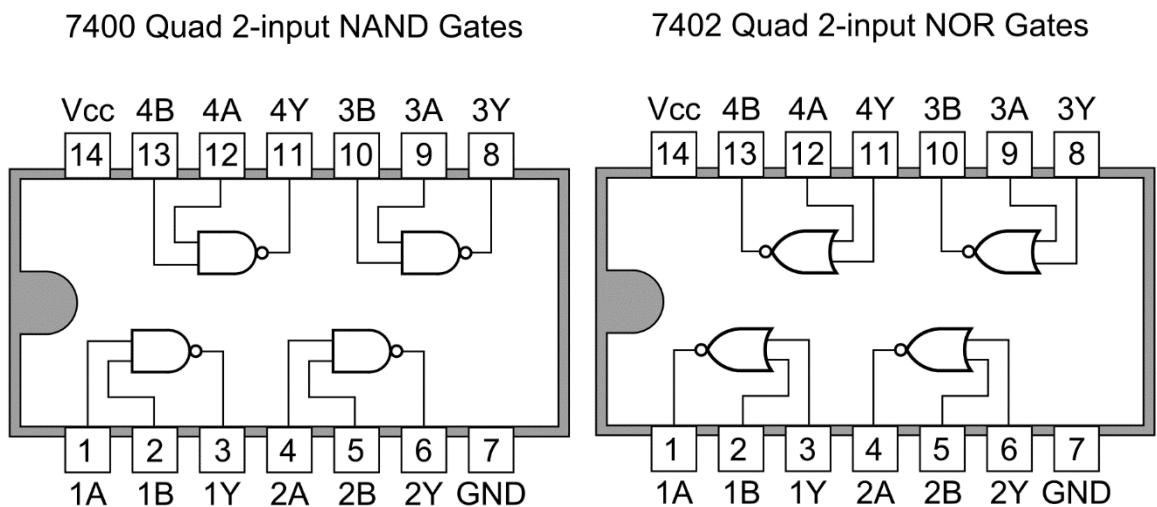


Figure 4.1

Initially, we used a DIP (dual in-line package) to connect the ICs to our circuit. However, we found that when removing the IC from the DIP, the connections became messy and difficult to sort out. This made it challenging to carry out further testing on the IC.

To address this issue, we decided to switch to a ZIF socket. This type of connector allowed us to easily insert and remove the IC without causing any damage to the pins or making the connections messy. This made it easier to

carry out further testing and helped to ensure that our ICs were functioning properly in our circuit.

When we were conducting unit testing for the transistor, we decided to develop a pin detection feature to ensure that we could accurately identify each terminal. To accomplish this, we analyzed the voltage between each of the transistor's terminals, carefully observing the readings to determine which terminal corresponded to which pin. By comparing the voltage measurements to the transistor's datasheet, we were able to confirm our findings and accurately identify each terminal. This process allowed us to effectively test the transistor's functionality and ensure that it was operating correctly within our circuit.

During the unit testing phase for the diode, we applied a test signal across the diode and measured the resulting voltage while also calculating the current flow. By measuring the voltage drop across the diode and applying Ohm's Law, we were able to calculate the current flow through the diode. This allowed us to determine whether the diode was functioning as expected or not.

Additionally, we also checked for any possible reverse bias voltage that could potentially cause damage to the diode. Overall, this testing process helped us ensure the functionality and reliability of the diode component in our circuit.

When we were conducting unit testing for the op-amp, our initial approach was to generate a sine wave and observe whether the output was inverted, as this is a common functionality of op-amps. However, we soon realized that Arduino microcontrollers are not capable of generating a true sine wave. As a result, we had to come up with an alternative method for testing the op-amp's functionality.

To achieve this, we decided to apply a digital signal to the op-amp and use it as a simple inverter. By analyzing the output signal and comparing it to the expected result, we were able to determine whether the op-amp was operational or not. This process involved carefully observing the signal waveforms and ensuring that the output signal matched our desired expectations. Through this method, we were able to successfully test the functionality of the op-amp and confirm that it was working as intended.

After completing the unit testing phase for each component of the circuit, we proceeded to integrated testing. During the integrated testing phase, we realized that the number of pins on the Arduino Uno was not sufficient to meet the requirements of our project. As a result, we decided to use an I2C module for the display, which helped to reduce the number of connections with the Arduino Uno.

However, as we continued with our integrated testing, we found that even with the I2C module, the pins on the Arduino Uno were still insufficient for our needs. After some consideration and discussion among team members, we decided to switch to the Arduino Mega microcontroller, which had more pins and could better support our project requirements.

As we were making the switch to the Arduino Mega, we also decided to add a new feature to our device - the ability to show the characteristic curve diode on a PC. To achieve this feature, we needed to incorporate a WIFI module into our device.

As we decided to incorporate the feature of displaying characteristic curve of diode on a PC through Wi-Fi, we realized that we needed a more advanced microcontroller than Arduino Mega. After some discussions, we decided to switch to ESP32 microcontroller, which offers better connectivity options and supports Wi-Fi connectivity.

Once we switched from the Arduino Mega to the ESP32 microcontroller to enable showcasing characteristic curve of diode in a PC via Wi-Fi, we encountered an issue during testing. It was observed that the display we were using was not compatible with the ESP32 microcontroller as it required a voltage of 5V, whereas the board's maximum voltage output was only 3.3V. To resolve this issue, we explored various options and concluded that switching back to the Arduino Mega was a more suitable choice as it could provide the required 5V output to power the display. Therefore, we switched back to the Arduino Mega and continued with our project.

Although we faced an obstacle when we discovered the incompatibility between the display and the ESP32 microcontroller, we persisted in our efforts to showcase the characteristic curve of diode on a PC via Wi-Fi. We decided to pursue a different approach by developing a separate circuit for the characteristic curve plotting feature, utilizing the ESP32 microcontroller.

We carefully selected compatible components and built the circuit from scratch. After completion, we integrated the circuit into our existing project and used the ESP32 microcontroller to establish communication between the circuit and the PC. This enabled us to successfully plot the characteristic curve of diode in real-time, which was our ultimate objective.

After completing our project, we realized that the size of the device was becoming an issue. As we were using the Arduino Mega microcontroller, the circuit board was becoming more complex and larger in size. We needed to find a solution that would allow us to maintain the functionality of our device while also reducing its size.

After researching various options, we came across the MEGA 2560 Pro Mini, which was a smaller and more compact version of the Arduino Mega. This microcontroller was designed specifically for projects that require a smaller form factor and lower power consumption.

We decided to switch to the MEGA 2560 Pro Mini to reduce the size of our circuit board without compromising on its functionality. The new microcontroller offered the same features and functionality as the Arduino Mega, but in a smaller package.

Making the switch required some adjustments to our existing design, as the pin layout and size of the MEGA 2560 Pro Mini were different from that of the Arduino Mega. However, with some modifications to our circuit board

layout and wiring, we were able to successfully integrate the new microcontroller into our project.

PC connector

To begin our project, we first had to explore ways to send data to a computer. We chose to use the ESP32, which has built-in Wi-Fi capabilities, and then we had to determine how to send data through Wi-Fi. Connecting to the Wi-Fi network required us to enter the SSID and password in the source code, and once connected, the IP address was displayed in the serial monitor. To ensure that the connection was stable and functional, we printed some text.

Next, we worked on establishing a connection between the ESP32 and a local web page. We utilized one of the ESP32's features, which allows it to act as a web server that serves web pages to clients. Using the Webserver library in the Arduino IDE, we were able to create a web server on the ESP32.

An API serves as a set of rules defining how distinct components within a system communicate with each other. Our team has established an API on the ESP32, providing a web page to interact with the ESP32 via HTTP requests. We utilized JavaScript on the web page to send HTTP requests to the API on the ESP32, employing the XMLHttpRequest object in JavaScript to facilitate the process.

Upon receiving an HTTP request from the web page, the ESP32 processed the request and executed any necessary actions. Subsequently, the ESP32 sent a response back to the web page. We employed a bulb to verify the connection throughout these procedures.

To gain a comprehensive understanding of data transfer via WI-FI, we proceeded to control an LED bulb using the webpage. To accomplish this, we utilized a relay module to connect the bulb to the ESP32. The relay module acted as a switch capable of toggling the bulb on or off, and we connected it to one of the digital pins of the ESP32.

To facilitate the process, we created a basic HTML page featuring a button that toggled the state of the bulb. When pressed, the button sent an HTTP request to the ESP32. Additionally, we established an API on the ESP32 that listened for HTTP requests from the web page. This API featured a route that toggled the state of the bulb. Upon calling the route, the API turned the relay module on or off, depending on the current state of the bulb. Following the bulb's state change, the API sent a response to the web page, indicating the new state of the bulb. Upon receiving the response from the API, the web page updated the button on the page to reflect the new state of the bulb.

Our team's next objective was to display the output of the serial monitor on a webpage in real time. To achieve this, we implemented WebSocket communication between the ESP32 and the web page. We utilized the ESPAsyncWebServer and

AsyncWebSocket libraries to establish a web server and WebSocket on the ESP32, allowing for bidirectional communication between the ESP32 and the web page.

Within our ESP32 sketch, we printed data to the serial port using the Serial.print() or Serial.println() functions. To transmit this data to the web page via the WebSocket, we first loaded the ESP32's IP address in our web browser to view the web page. The output of the serial monitor is displayed on the web page.

We then encountered a significant hurdle in our efforts to obtain live observations through the ESP32. We utilized a potentiometer to obtain dynamic data (live observations), first connecting the potentiometer to the ESP32. Within our ESP32 sketch, we utilized the analogRead() function to read the potentiometer value, which returned a value between 0 and 4095, representing the voltage on the ADC pin. We then employed the WebSocket to transmit the potentiometer values to the web page, utilizing JavaScript in the process. After altering our programming codes on numerous occasions, we ultimately succeeded in obtaining our full observations on the webpage.

Our most significant obstacle was how to store our dynamic data in a database. Initially, we attempted to use XAMPP to establish a local host and create an offline database using phpMyAdmin. After multiple attempts, we ultimately decided to send data to an Excel sheet. Unfortunately, this method proved to be somewhat inaccurate. Ultimately, we decided to create a free host and transmit the data online. Although this was a challenging task, we managed to send the data through the URL to the database using the GET method, as it proved to be easier to handle data in this manner.

Finally, we plotted the graph on the webpage using the data from the database. To do so, we first retrieved the data from the SQL database using visualize the dynamic data we collected from the ESP32, we plotted a graph on the web page using the data stored in the database. To achieve this, we first retrieved the data from the SQL database using a server-side programming language such as PHP. Next, we organized the data into arrays in the format expected by the charting library we planned to use. We decided to use the chart.js library to plot the graph, as it is widely used and easy to implement.

With the data properly formatted, we used the chart.js library to create a graph that updated in real-time as new data was added to the database. The graph provided a visual representation of the data, which made it easier to analyze and draw insights. Additionally, we added interactive features to the graph, such as zooming and panning, to provide users with more control over the data visualization.

In the end, the switch to the MEGA 2560 Pro Mini allowed us to significantly reduce the size and complexity of our device, while maintaining its full functionality. It was a valuable lesson in finding creative solutions to design challenges and adapting to new technologies as they become available.

4.2. Conclusion

In conclusion, the smart component tester is a device designed to provide an efficient and reliable method for testing electronic components. The device is suitable for use in various settings such as electronic laboratories, research and development centers, and educational institutions. It is intended to be user-friendly and easily operable for both novice and experienced users, providing a valuable tool for electronic engineers, technicians, and hobbyists who require an efficient tool to test their electronic components.

The device's ability to detect any malfunctioning pins of the Logic ICs and provide results for further analysis, as well as its ability to send data to a local host and plot the characteristic curve of diode, adds to its usefulness in electronic testing. The feature provides a graphical representation of the characteristics of these electronic components, making it easy to analyze and interpret their behavior.

The project has also focused on designing and implementing a user interface that is easy to navigate, intuitive, and displays test results clearly. The user interface is responsive and allows for customization of test parameters, providing users with greater control over the testing process.

Furthermore, the project has aimed to make the device cost-effective by using readily available and affordable components. The device is designed to be low maintenance, reducing the cost of ownership and ensuring its longevity. The project scope also includes testing and validating the device's performance to ensure its accuracy and reliability in testing electronic components.

Overall, the smart component tester is an efficient and reliable device designed to provide a valuable tool for electronic testing. With its user-friendly interface, advanced features, and cost-effective design, the device is sure to make a significant contribution to the field of electronic testing.

4.3. Future Work

There are several potential future plans for the Smart Component Tester project. Here are some of them:

- Expansion of component testing: Currently, the device is designed to test logical ICs, diodes, op amps, and transistors. However, there are many other types of electronic components that could potentially be added to the device's testing capabilities. In the future, the project team could research and develop methods for testing additional types of components, such as capacitors, resistors, and inductors.
- Integration with other tools: While the Smart Component Tester is a powerful standalone tool, it could also be integrated with other electronic testing tools to provide a more comprehensive testing solution. For example, the device could be connected to an oscilloscope to provide more detailed waveform analysis, or it could be integrated with a signal generator to provide more complex testing scenarios.
- Cloud connectivity: Currently, the device can send data to a local host, but it could also be designed to connect to cloud-based platforms for data analysis and sharing. This could be especially useful for large-scale testing projects or for remote teams who need to collaborate on testing results.
- AI-based testing: As artificial intelligence (AI) continues to advance, there may be opportunities to integrate AI-based testing capabilities into the Smart Component Tester. This could potentially enable the device to automatically identify and diagnose faults in electronic components, or to provide more advanced analysis of testing results.
- Miniaturization: While the current design of the Smart Component Tester is portable, there may be opportunities to make the device even smaller and more lightweight by using a custom designed circuit instead of Arduino microcontroller. This could be useful for fieldwork where space and weight are at a premium, or for applications where the device needs to be integrated into other equipment.

5. References

1. Yinghua Min. Jishun Kuang . Xiaoyan Niu (2003) At-speed current testing [logic IC testing] - ResearchGate, ResearchGate. Available at: https://www.researchgate.net/publication/4047391_At-speed_current_testing_logic_IC_testing (Accessed: March 27, 2023).
2. T. Brandon; B. Cockburn; M. Hume (2004) A reconfigurable digital IC Tester implemented using the ARM integrator rapid prototyping system, IEEE Xplore. Available at: <https://ieeexplore.ieee.org/document/1347590> (Accessed: March 27, 2023).
3. Micaela Serra (2005) *Digital IC Testing: An introduction* - uvic.ca, ResearchGate. Dept. of Computer Science University of Victoria Victoria, B.C. Canada. Available at: https://webhome.cs.uvic.ca/~mserra/Publications/Chapter_2004d.pdf (Accessed: March 27, 2023).
4. Chirayu Darji (2021) *Design and implementation of microcontroller based Digital Logic Gate ...*, ResearchGate. Dharmsinh Desai University. Available at: https://www.researchgate.net/publication/350580762_DESIGN_AND_IMPLEMENTATION_OF_MICROCONTROLLER_BASED_DIGITAL_LOGIC_GATE_IC_TESTER (Accessed: March 27, 2023).
5. Be'atrice Pradarelli; Laurent Latorre; Marie-Lise Flottes; Yves Bertrand; Pascal Nouet. (20n.d.) *Remote Labs for Industrial IC Testing*, ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5306065> (Accessed: March 27, 2023).
6. *The Best Integrated Circuit Testers (testing any ICS, 2022)* (2022) Yaman Electronics. Available at: <https://www.yamanelectronics.com/best-integrated-circuit-testers/> (Accessed: March 27, 2023).
7. *Diodes* (no date) Futurlec. Available at: <https://www.futurlec.com/Diodes.shtml> (Accessed: March 27, 2023).
8. Josna (2022) *9 best transistor tester reviews in 2022*, Electronics Hub. Available at: <https://www.electronicshub.org/best-transistor-tester/> (Accessed: March 27, 2023).
9. *MOSFET transistor series* (no date) Futurlec. Available at: <https://www.futurlec.com/TransMos.shtml> (Accessed: March 27, 2023).
10. *Operational amplifiers (Op-Amps)* (no date) Futurlec. Available at: <https://www.futurlec.com/ICLinearSeries.shtml> (Accessed: March 27, 2023).
11. www.futurlec.com (no date) General Purpose Transistors, Futurlec. Available at: <https://www.futurlec.com/TransGen.shtml> (Accessed: March 27, 2023).

12. www.futurlec.com (no date) *power transistor series*, *Futurlec*. Available at: <https://www.futurlec.com/TransPower.shtml> (Accessed: March 27, 2023).
13. Jesse (2022) ESP32 insert data into mysql database using PHP and Arduino Ide, Microcontrollers Lab. Available at: <https://microcontrollerslab.com/esp32-mysql-database-php/> (Accessed: March 28, 2023).

6. Appendixes

- Gate IC Testing

2 input gate IC

```
320 //Gate IC testing function
321 void gate2in(){
322     int outputpins[8] = {pin1,pin2,pin4,pin5,pin9,pin10,pin12,pin13}; //output from bord, inout to gate IC
323     int output2d[4][2] = {{pin1,pin2},{pin4,pin5},{pin9,pin10},{pin12,pin13}};
324     int inputpins[4] = {pin3,pin6,pin8,pin11}; //input to bord, output from gate IC
325     int inputs[4][2]={{{LOW,LOW},{LOW,HIGH},{HIGH,LOW},{HIGH,HIGH}}};
326     int type[4];
327
328     pinMode(VCC,OUTPUT);
329     pinMode(GND,OUTPUT);
330     digitalWrite(VCC,HIGH); //set Vcc to 5V
331     digitalWrite(GND,LOW); //set ground
332
333     for(i=0;i<8;i++){pinMode(outputpins[i],OUTPUT);} //set output pins
334     for(i=0;i<4;i++){pinMode(inputpins[i],INPUT);} //set input pins
335
336     //check whenever output has, if it is system recognize as ic is connected otherwise not.
337     j=0;
338     for(i=0;i<4;i++){
339         for(j=0;j<4;j++){
340             digitalWrite(output2d[i][0], inputs[j][0]);
341             digitalWrite(output2d[i][1], inputs[j][1]);
342             output[j] = digitalRead(inputpins[i]);
343             delay(100);
344         }
345         type[i] = icCheck2In(inputpins[i]);
346     }
347     filterResult(type);
348 }
```

As shown in the above code segment (gate2in () function),

first, we have initialized arrays and 2D arrays to easily access the input and output pins of the Gate IC (line 322, 323, 324), Store 2 input gate input variations (line 325) and store results (line 326).

then we can use simple for loop to get things done as shown in

- line 333 & 334 – initialize pin modes,
- line 338 to 346 – select inputs and store outputs for each gate in IC.

Then the result sent to the icCheck2In () function to check with input pin that we recorded output from and record gate type according to the output.

After that we will call filterResult () function and we will feed it with type array to check if there are any unmatched gate type or pins.

```

349 int icCheck2In(int pin){
350     int gatetype;
351     if(output[0]==0 && output[1]==0 && output[2]==0 && output[3]==1){ gatetype=1; }
352     else if(output[0]==0 && output[1]==1 && output[2]==1 && output[3]==1){ gatetype=2; }
353     else if(output[0]==0 && output[1]==1 && output[2]==1 && output[3]==0){ gatetype=3; }
354     else if(output[0]==1 && output[1]==1 && output[2]==1 && output[3]==0){ gatetype=4; }
355     else if(output[0]==1 && output[1]==0 && output[2]==0 && output[3]==0){ gatetype=5; }
356     else if(output[0]==1 && output[1]==0 && output[2]==0 && output[3]==1){ gatetype=6; }
357     else{gatetype=pin;}
358
359     return gatetype;
360 }
```

In this icCheck2IN (int pin) function simply check which gate type of each input pins have and return it to the gate2in () function. If the gate output not matched with any of these it means that pin not working. If that happens it simply return pin number.

```

361 void filterResult(int t[]){
362     int size=sizeof(t)/sizeof(t[0]);
363     int gate=0;
364     int maxcount = 0;
365     int malpin[size]={0};
366     for(i = 0; i < size; i++){ //get most presetn gate type; incase if there are malfunctining gate that gives
367         if(t[i] == 1 || t[i] == 2 || t[i] == 3 || t[i] == 4 || t[i] == 5 || t[i] == 6){
368             int count = 0;
369             for(j = 0; j < size; j++){
370                 if(t[i] == t[j]){
371                     count++;
372                 }
373             }
374             if(count > maxcount){
375                 maxcount = count;
376                 gate = t[i];
377             }
378         }
379         else{
380             malpin[i] = t[i];
381         }
382     }
383     typeTest(gate);
384     malFunPinDetect(malpin,size) ;
385 }
```

After the all the gates output checked the gate types for each gate stored in type array. To check every gate output matched with all the gates we used this function. Simply we check which gate type most present in the array and assign a integer to gate type,

- AND Gate = 1
- OR Gate = 2
- XOR Gate = 3
- NAND Gate = 4
- NOR Gate = 5
- XNOR Gate = 6

At the end this function calls the typeTest () function and feed the selected gate type number.

After that if there any detected mal functioning pins, we call malFunPinDetect () function and feed it with detected mal functioning pins (malpin array) and array size.

```

388 void typeTest(int r){ //give test result to display final output
389     |   buzzer(100);
390     |   lcd.setCursor(0,0);
391     |   lcd.print("Test Result");
392     |   lcd.setCursor(0,1);
393     |   lcd.print("IC Type:");
394     switch (r) {
395         |   case 0:
396             |       break;
397         |   case 1: //chek array has malfunctioning pin
398             |       lcd.print("AND");
399             |       Serial.print("AND");
400             |       digitalWrite(GREEN, HIGH);
401             |       break;
402         |   case 2: //chek array has malfunctioning pin
403             |       lcd.print("OR");
404             |       Serial.print("OR");
405             |       digitalWrite(GREEN, HIGH);
406             |       break;
407         |   case 3: //chek array has malfunctioning pin
408             |       lcd.print("XOR");
409             |       Serial.print("XOR");
410             |       digitalWrite(GREEN, HIGH);
411             |       break;
412         |   case 4: //chek array has malfunctioning pin
413             |       lcd.print("NAND");
414             |       Serial.print("NAND");
415             |       digitalWrite(GREEN, HIGH);
416             |       break;
417         |   case 5: //chek array has malfunctioning pin
418             |       lcd.print("NOR");
419             |       Serial.print("NOR");
420             |       digitalWrite(GREEN, HIGH);
421             |       break;
422         |   case 6: //chek array has malfunctioning pin
423             |       lcd.print("XNOR");
424             |       Serial.print("XNOR");
425             |       digitalWrite(GREEN, HIGH);
426             |       break;
427         |   default:
428             |       lcd.setCursor(0,1);
429             |       lcd.print("error:Test Again");
430             |       Serial.print("error:Test Again");
431             |       digitalWrite(RED, HIGH);
432             |       buzzer(100);
433             |       break;
434     }
435 }
```

In above code segment simple display, the gate type according to the input integer in user-friendly way and send the data through to serial communication to PC.

```

436 void malFunPinDetect(int r[],int arysize){ //give test result to display final output
437     buzzer(500);
438     lcd.setCursor(0,2);
439     lcd.print("Malfunctioning:");
440     Serial.print("Malfunctioning");
441     lcd.setCursor(0,3);
442     for(i=0;i<arysize;i++){
443         switch (r[i]) {
444             case 0:
445                 digitalWrite(GREEN, HIGH);
446                 break;
447             case 31: //chek array has malfunctioning pin
448                 lcd.print("Pin3 ");
449                 Serial.print("Pin3");
450                 digitalWrite(RED, HIGH);
451                 break;
452             case 33: //chek array has malfunctioning pin
453                 lcd.print("Pin4 ");
454                 Serial.print("Pin4");
455                 digitalWrite(RED, HIGH);
456                 break;
457             case 37: //chek array has malfunctioning pin
458                 lcd.print("Pin6 ");
459                 Serial.print("Pin6");
460                 digitalWrite(RED, HIGH);
461                 break;
462             case 34: //chek array has malfunctioning pin
463                 lcd.print("Pin8 ");
464                 Serial.print("Pin8");
465                 digitalWrite(RED, HIGH);
466                 break;
467             case 30: //chek array has malfunctioning pin
468                 lcd.print("Pin10 ");
469                 Serial.print("Pin10");
470                 digitalWrite(RED, HIGH);
471                 break;
472             case 28: //chek array has malfunctioning pin
473                 lcd.print("Pin11 ");
474                 Serial.print("Pin11");
475                 digitalWrite(RED, HIGH);
476                 break;
477             case 26: //chek array has malfunctioning pin
478                 lcd.print("Pin12 ");
479                 Serial.print("Pin12");
480                 digitalWrite(RED, HIGH);
481                 break;
482             default:
483                 lcd.print("error:Test Again");
484                 Serial.print("error:Test Again");
485                 digitalWrite(RED, HIGH);
486                 buzzer(100);
487                 break;
488         }
489     }
}

```

In filterResult () calls malFunPinDetect () function detected mal functioning pins (malpin array) and array size this will run each input entry in given array and display pin name according to the pin number send the data through to serial communication to PC.

After all these code segment users will be able to see gate type and malfunctioning pins withing 1 to 2 seconds. if have any. To develop this algorithm, we assumed that if IC working, the type array that has most count in the type array is the gate type, if there any other types or array contain pin numbers, it will indicate malfunctioning pin.

The testing and find gate type and identify mal functioning pins of 3 input gate IC and 4 input gate IC, is same as above explained and

- gate2in () function have different pin arrays.
- icCheck2IN (int pin) function check only 3 or 4 outputs.
- filterResult (), typeTest () and malFunPinDetect () functions are common for all types of gate ICs.

• Op-Amp 741

```
775 //op-amp testing function
776 int opAmp(){
777     pinMode( opinput, OUTPUT ); //input of the op-amp
778     pinMode( opoutput, INPUT ); //output of the op-amp
779
780     lcd.setCursor(0,0);
781     lcd.print( "    Test Result    " );
782     lcd.setCursor(0,1);
783     lcd.print( "Component: Op-Amp" );
784     lcd.setCursor(0,2);
785     lcd.print( "Testing Op-Amp" );
786     delay( 1000 );
787     lcd.setCursor(0,2);
788     lcd.write("                ");
789
790     digitalWrite( opinput, LOW ); //op-amp input set to low
791     delay(10); //let stabilize the voltage
792     if( digitalRead(opoutput) == HIGH ){ //op-amp output should be high, first check high 'case not giving any
793         digitalWrite( opinput, HIGH ); //op-amp input set to high
794         delay(10);
795         if( digitalRead(opoutput) == LOW ){ //op-amp output should be low
796             lcd.setCursor(0,2);
797             lcd.print( "Status: Working" );
798             buzz(100);delay(100);buzz(100); //make two beep
799         }
800     }
801     else {
802         lcd.setCursor(0,2);
803         lcd.print("Status: Not Working");
804         buzz(500);
805     }
806     digitalWrite( opled, LOW );
807 }
```

In here we used digital signal to check op-amp is output inverting signal or not. That means when input signal is LOW, output should be HIGH and input signal is HIGH, output should be LOW.

In line 790 we set opinput to LOW and wait 10 milliseconds to stabilize the voltage and check whether opoutput is HIGH. If it is then we changed opinput to HIGH and wait 10 milliseconds to stabilize the voltage and check whether opoutput is LOW. If it worked, it would display Status: Working". If it not, then it will display "Status: Not Working." All these display data will send the data through to serial communication to PC.

• BJT Transistor

```

474 //Transistor testing function
475 void bjtPinDetect(){
476     lcd.clear();
477     pinMode(basePin, OUTPUT);
478     pinMode(collectorPin, OUTPUT);
479     pinMode(emitterPin, INPUT);
480
481     // Set base/gate to low and collector/drain to high
482     digitalWrite(basePin, LOW);
483     digitalWrite(collectorPin, HIGH);
484     delay(10);
485
486     // Measure voltage on emitter/source
487
488     int emitterVoltage = analogRead(emitterPin);
489
490     // Set base/gate to high and collector/drain to low
491     digitalWrite(basePin, HIGH);
492     digitalWrite(collectorPin, LOW);
493     delay(10);
494
495     // Measure voltage on emitter/source
496     int emitterVoltage2 = analogRead(emitterPin);
497
498     // Determine which pin is which based on voltage measurements
499     if(!(emitterVoltage2 < 100) || !(emitterVoltage < 100)){
500         if (emitterVoltage > emitterVoltage2) {
501             lcd.setCursor(0,0);
502             lcd.print("PNP Transistor");
503             Serial.println("PNP Transistor");
504             lcd.setCursor(0,1);
505             lcd.print("C: 1 ");
506             Serial.println("C: 1 ");
507             lcd.print("B: 2 ");
508             Serial.println("B: 2 ");
509             lcd.print("E: 3 ");
510             Serial.println("E: 3 ");
511         }
512     else {
513         lcd.setCursor(0,0);
514         lcd.print("NPN Transistor");
515         Serial.println("NPN Transistor");
516         lcd.setCursor(0,1);
517         lcd.print("E: 1 ");
518         Serial.println("E: 1 ");
519         lcd.print("B: 2 ");
520         Serial.println("B: 2 ");
521         lcd.print("C: 3 ");
522         Serial.println("C: 3 ");
523     }
524 } else {
525     lcd.print("Transistor Not detected");
526 }
527 }
```

In this code segment we test whether BJT transistor is PNP or NPN. First we set collector pin to HIGH and base pin to LOW and measure the emitter voltage as emitterVoltage. Next base pin set to HIGH and collector pin to LOW. And measured the emitter voltage as emitterVoltage2. If emitterVoltage and emitterVoltage2 has value greater than 100 it means that component is connected to the pins If not it will display "Transistor Not detected".

If emitterVoltage > emitterVoltage2 it means connected Transistor is PNP Transistor and it display Collector is 1st pin, Base is 2nd pin and Emitter is 3rd pin.

If above statement is wrong it means connected Transistor is NPN Transistor and it display Emitter is 1st pin, Base is 2nd pin and Collector is 3rd pin.

• FET Transistor

```
529 void fetPinDetect(){
530     lcd.clear();
531     pinMode(basePin, OUTPUT);
532     pinMode(collectorPin, OUTPUT);
533     pinMode(emitterPin, OUTPUT); // For MOSFET detection, set emitter/source pin as output
534
535     // Set base/gate to low and collector/drain to high
536     digitalWrite(basePin, LOW);
537     digitalWrite(collectorPin, HIGH);
538     digitalWrite(emitterPin, LOW); // For MOSFET detection, set emitter/source pin to low
539     delay(10);
540
541     // Measure voltage on emitter/source
542     int emitterVoltage = analogRead(emitterPin);
543
544     // Set base/gate to high and collector/drain to low
545     digitalWrite(basePin, HIGH);
546     digitalWrite(collectorPin, LOW);
547     digitalWrite(emitterPin, HIGH); // For MOSFET detection, set emitter/source pin to high
548     delay(10);
549
550     // Measure voltage on emitter/source
551     int emitterVoltage2 = analogRead(emitterPin);
552
553     // Determine which pin is which based on voltage measurements
554     if(emitterVoltage2 > emitterVoltage) { // Detect N-channel MOSFET
555         lcd.setCursor(0,0);
556         lcd.print("N-channel MOSFET");
557         Serial.println("N-channel MOSFET");
558         lcd.setCursor(0,1);
559         lcd.print("D: A2 ");
560         Serial.println("D: 1 ");
561         lcd.print("G: A1 ");
562         Serial.println("G: 2 ");
563         lcd.print("S: A0 ");
564         Serial.println("S: 3 ");
565     } else if (emitterVoltage > emitterVoltage2) { // Detect P-channel MOSFET
566         lcd.setCursor(0,0);
567         lcd.print("P-channel MOSFET");
568         Serial.println("P-channel MOSFET");
569         lcd.setCursor(0,1);
570         lcd.print("S: A2 ");
571         Serial.println("S: 1 ");
572         lcd.print("G: A1 ");
573         Serial.println("G: 2 ");
574         lcd.print("D: A0 ");
575         Serial.println("D: 3 ");
576     } else {
577         lcd.print("Transistor Not detected");
578     }
579 }
```

This code segment used to determine n-channel MOSFET and p-channel MOSFET same as above instead this used for FET.

• Diode Tester

```
581 //Diode testing function
582 void diodeTest(){
583     pinMode( diodeinput, OUTPUT ); //set diode pin modes
584     pinMode( diodeoutput, INPUT );
585
586     lcd.setCursor(0,0);
587     lcd.print( "    Test Result    " );
588     Serial.println("Test Result");
589     lcd.setCursor(0,1);
590     lcd.print( "Component: Diode" );
591     Serial.println("Component: Diode");
592     lcd.setCursor(0,2);
593     lcd.print( "Testing Diode" );
594     Serial.println("Testing Diode");
595     delay( 1000 );
596     lcd.setCursor(0,2);
597     lcd.write("                ");
598
599     digitalWrite( diodeinput, HIGH ); //set dioe input pin to high
600     if( digitalRead( diodeoutput ) == HIGH ){ //check output pin is also high if it is, diode is working
601         lcd.setCursor(0,2);
602         lcd.print( "Status: Working" );
603         Serial.println("Status: Working");
604         buzzer(100);delay(100);buzzer(100); //make two beep
605     }
606     else{ //diode is not working
607         lcd.setCursor(0,2);
608         lcd.print("Status: Not Working");
609         Serial.println("Status: Not Working");
610         buzzer(500); //not working beep
611     }
612     digitalWrite( diodeled, LOW );
613 }
```

This simply test the given signal is receive from other end of the diode. If it not diode is not working.

• Diode Curve

```
617 void drawDiodeGraph(){
618     pinMode(dacPin, OUTPUT); // set DAC pin as output
619     float voltage;
620     float current;
621     lcd.print("Data Sending to PC")
622     for(int i = 0; i <= 20; i++) {
623         voltage = i * 0.25; // increment voltage in steps of 0.25V
624         analogWrite(dacPin, voltage * 51); // convert voltage to DAC value (0-255)
625         current = analogRead(currentSensorPin) * 0.0049; // convert current sensor reading to current (A)
626         Serial.print(voltage); // send voltage reading through serial port
627         Serial.print(",");
628         Serial.println(current); // send current reading through serial port
629         delay(100); // wait 100ms before next voltage increment
630     }
631 }
```

Using DAC (digital to Analogue Converter) we supply voltage by 0.2v and measure current using current sensor and send the data through serial port to PC connector.

- Serial Connector

```

139  if (Serial.available()) {
140    | Serial.write(Serial.read()); // echo received data back
141  }

```

Using this code segment in PC Connecting Module reads data from Smart Component Tester and display it in website.

- PC connector

- index.php

```

$host="localhost";
$dbname="id20514385_compdatabase";
$username="id20514385_component";
$password="Uoktester@1100";

$conn = new mysqli($host, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
else {

```

This is a PHP code snippet that establishes a connection to a MySQL database using the mysqli extension. The first four lines define the necessary credentials to connect to the MySQL server. Then, the mysqli() function is used to create a new connection object with the provided credentials.

```

$sql = "SELECT voltage,current FROM component";
$readData = $conn->query($sql);
echo '<table cellspacing="3" cellpadding="2" width="200" height="200">
<tr>
<th>voltage</th>
<th>current</th>
</tr>';
if ($result = $conn->query($sql)) {
    $i = 0;
    $graph= array(array());
    while ($row = $result->fetch_assoc()) {
        $row_value1 = $row["voltage"];
        $row_value2 = $row["current"];
        $graph[$i][0] = $row_value1;
        $graph[$i][1] = $row_value2;
        $i++;
    }
    for($i=0;$i<20;$i++) {
        echo '<tr>
<td>' . $graph[$i][0] . '</td>
<td>' . $graph[$i][1] . '</td>
</tr>';
    }
    $result->free();
}
$conn->close();

```

```
?>
</table>

</div>
<div class="col-md-8 col-sm-12">
    <canvas id="myChart"></canvas>
</div>
</div>

<script>
```

This is a PHP code that retrieves data from a MySQL database, formats it into an HTML table, and creates a graph using the Chart.js library. The first line of code defines a SQL query that selects the "voltage" and "current" columns from a MySQL table named "component." The next line executes the query using the `query()` method of the `mysqli` object, and stores the result in the `$readData` variable. The following lines of code create an HTML table with two columns: "voltage" and "current." The while loop fetches each row of the result set and stores the "voltage" and "current" values in a two-dimensional array called `$graph`. After the loop, a for loop is used to iterate over the first 20 rows of `$graph` and display them in the HTML table. Finally, a canvas element with id "myChart" is created, which will be used to render the chart. The Chart.js library is used to create a chart with `$graph` data, which can be customized further using various configuration options provided by the library.

```
var graphData = <?php echo json_encode($graph); ?>;
var yValues = graphData.map(function(data) {
    return data[1];
});
var ctx = document.getElementById('myChart').getContext('2d');
```

This code extracts the y-values from the graph data by using the `map()` method to iterate over each element in the `graphData` array and return the second element (index 1) of each subarray. The resulting array of y-values is stored in the `yValues` variable.

Finally, the code selects a canvas element with the ID `myChart` and gets its 2D rendering context using the `getContext` method. The `myChart` canvas element is presumably used to render a chart or graph using a library such as Chart.js, and the `ctx` variable holds a reference to the rendering context that can be used to draw on the canvas.

```
var myChart = new Chart(ctx, {
    type: 'line',
    data: [
        labels: graphData.map(function(voltage) { return voltage[0]; }),
        datasets: [
            label: 'voltage vs current',
            data: graphData.map(function(voltage) { return voltage[1]; }),
            fill: false,
            borderColor: 'rgb(75, 192, 192)',
            tension: 0.1
        ]
    ],
    options: {
        scales: {
            xAxes: [
                display: true,
                scaleLabel: {
                    display: true,
                    labelString: 'Data Points'
                }
            ],
            yAxes: [
                display: true,
                scaleLabel: {
                    display: true,
                    labelString: 'Values'
                }
            ]
        }
    }
});
</script>
```

This code creates a line chart using the Chart.js library. The chart displays voltage vs current data, with voltage values on the x-axis and current values on the y-axis. The data is provided in the `graphData` variable, which is an array of arrays, where each inner array contains two values: the voltage and current values.

The `newChart()` function creates a new instance of a Chart object and takes two arguments: a reference to the canvas element where the chart will be rendered (`ctx`), and an object that contains configuration options for the chart.

The `type` option specifies that the chart will be a line chart. The `data` option contains the data and formatting information for the chart. The `labels` array specifies the values for the x-axis ticks, which are the voltage values from the `graphData` array. The `datasets` array contains an object that specifies the data points for the chart, which are the current values from the `graphData` array. The `label` option specifies the label for the dataset, and the `borderColor` option specifies the color of the line on the chart.

The `options` object contains formatting options for the chart. The `scales` object specifies the formatting for the x- and y-axes. The `display` option specifies whether the axis should be displayed, and the `scaleLabel` object specifies the label for the axis.

o dbwrite.php

```
$sql = "SELECT voltage,current FROM component";
$readData = $conn->query($sql);
if ($readData->num_rows > 0) {
    echo "
        <table border='1'>
        <thead class='thead-light'>
        <tr>
            <th scope='col'>#</th>
            <th scope='col'>Last Month</th>
        </tr>
        </thead>
        <tbody>
            ";
    while ($row = $readData->fetch_assoc()) {
        echo "
            <tr>
                <th scope='row'>{$row['voltage']}</th>
                <td>{$row['current']}
```

The code checks if the query returned any rows using the **num_rows** property of the result object. If the result contains more than 0 rows, the code creates an HTML table to display the retrieved data.

The code then enters a loop and fetches each row of the result set using the **fetch_assoc()** method. It then displays the "voltage" and "current" values for each row in a table row.

If the query returns no results, the code displays the message "0 result". Finally, the code closes the database connection using the **close()** method.

○ Dbread.php

```
<?php

$host="localhost";
$dbname="id20514385_compdatabase";
$username="id20514385_component";
$password="Uoktester@1100";

$conn = new mysqli($host, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
else { echo"Connected My SQL.";}

$val=$_GET['v1'];
$v2=$_GET['v2'];
echo $val;
echo $v2;
// $val =$_POST['sendval'];
// $val2=$POST['sendval2'];

$sql ="INSERT INTO component(voltage,current) VALUES ('".$val."','".$.$v2."')";
$result = mysqli_query($conn,$sql);
$conn->close();

?>
```

This code establishes a connection to a MySQL database using the mysqli extension in PHP. It then retrieves two variables, \$val1 and \$val2, through the GET method, which are presumably voltage and current values obtained from a user. These variables are then inserted into a MySQL database table named "component" using an SQL INSERT statement. Finally, the connection is closed using the mysqli_close() function.

○ clear.php

```
<?php

$host="localhost";
$dbname="id20514385_compdatabase";
$username="id20514385_component";
$password="Uoktester@1100";

$conn = new mysqli($host, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Clear the database table
$sql = "DELETE FROM component"; // Replace with your actual table name
if ($conn->query($sql) === TRUE) {
    echo "Database cleared successfully";
} else {
    echo "Error clearing database: " . $conn->error;
}

$conn->close();

?>
```

First, the code defines the database server hostname, database name, username, and password required to connect to the MySQL database server. Then, it creates a new mysqli object and uses it to connect to the database server. If the connection is successful, the code executes a SQL query to delete all rows in the "component" table. If the query is successful, the code prints "Database cleared successfully" to the screen. Otherwise, it prints an error message that includes the error returned by the MySQL server.

Finally, the code closes the database connection.
