CHAPTER-1

INTRODUCTION

1.1 BACKGROUND

Many university researchers and chip designers in small companies are faced with a difficult problem when it comes to developing new circuit intellectual products (IPs); the cost of prototyping. In order to verify their IP(s) they need to fabricate a prototype, test it and characterize its performance. With the current speeds of up to few Giga Hertz, these circuits would require very expensive testers and scopes. The high cost of such testing equipments is prohibitive for most universities. At the same time, trends in electronic design have converged to what is known as an IP-Based design. This is a design methodology based on re-using existing circuit blocks, namely the IP blocks.

These blocks are designed and verified (through prototyping and testing) by IP vendors and are then used and re-used by ASIC (application-specific integrated circuits) and SoC (system on chip) designers. Hence, developing a cost-effective method for testing and characterizing prototypes of circuit IPs is highly desirable. The IC tester simply determines which are workable gates and which are faulty. The main purpose of the project is to develop a digital IC tester that is very less expensive and handy than that of what are available in markets. The aim is to check the ICs in very due course of time and display results of ICs being good or faulty immediately. The necessary input signal conditions are applied to the inputs of the gate through microcontroller and output of each gate is monitored and compared with the truth table, and depending on that comparison IC is tested whether it is good or faulty.

The basic function of digital IC tester is to test the logic functioning of the ICs as described in the truth table/function table. The truth tables are stored in database while coding of the microcontroller. The test displays the good ICs and faulty ICs on LCD. The test is being accomplished with the ICs belonging to the basic logic gate IC series. There are many IC tester available in market but we have developed a tester that is very cheap, portable, and easy to handle as well.

1.2 LITERATURE SURVEY

[1] Arduino based 74-series integrated circuits testing system at gate level: Prof. D. G. Kanade, Nikhil Zambare and Krishna Rathode, Department of Electronics, Vishwakarma Institute of Technology, Pune, Maharashtra, India.

Integrated Circuit (IC) testing is a crucial step in the manufacturing process to ensure the functionality and reliability of electronic components. This paper presents the development of an IC tester using MATLAB, aimed at automating the process of testing and analyzing ICs to identify defects and ensure performance standards are met. The proposed IC tester leverages MATLAB's robust computational capabilities and versatile toolboxes to design, simulate, and implement test procedures for a variety of ICs.

[2] Microcontroller Based Design of Digital IC Tester with Multi-Testing and Loop Testing Functions: Maribelle D. Pabiania , Joanne T. Peralta, Leo Anthony T. De Luna,Phillipines.

The system integrates several key components: a user-friendly graphical user interface (GUI) for ease of operation, a database of standard test vectors for different types of ICs, and a suite of analysis tools to interpret test results. MATLAB's Signal Processing and Instrument Control toolboxes are utilized to generate and manage test signals, while data acquisition hardware interfaces with the IC under test.

[3] Microcontroller based IC tester: Mirza Shoaib Ahmed, Iqbal Muhammad Umair, Kashif Mehboob NED University of Engineering and Technology, Karachi.

Through a series of case studies, the IC tester is demonstrated to accurately identify common faults such as short circuits, open circuits, and parameter deviations. The tester's effectiveness is validated against industry-standard testing equipment, showing comparable accuracy and reliability. Additionally, the flexibility of the MATLAB environment allows for easy updates and modifications to test procedures, accommodating new IC designs and standards.

[4] Digital IC Tester using Arduino: Yasir Hashim, Marwa Awni, Abdullah Mufeed Computer Engineering Department, Faculty of Engineering, Tishk International University, Erbil, Iraq.

This work highlights the potential of MATLAB as a powerful platform for developing automated IC testing solutions, providing significant benefits in terms of cost reduction, ease of use, and adaptability. The presented IC tester serves as a foundation for further advancements in automated testing technologies, aiming to enhance the efficiency and precision of IC manufacturing processes.

[5] A low-cost Method for Test and Speed Characterization of Digital Integrated Circuit Prototypes Muhammad E. S. Elrabaa, Amran A. Al-Aghbari, and Mohammed A. Al-Asli Computer Engineering Department, King Fahd University for Petroleum and Minerals Dhahran, Saudi Arabia.

In this work, a novel low-cost test and characterization method has been developed. It allows functional testing and speed characterization of any number of IPs on the same chip. Users can specify their functional test procedure and data, capture the results, and specify the clock frequency. It can support any number of input/output ports per IP under test (IUT) with arbitrary port widths (number of bits). In section 2 an overview of the proposed method is presented followed by a detailed description of the support circuitry in section 3. Section 4 provides a description of the stand-alone test processor. The complete prototype of the proposed method that has been designed and implemented on FPGAs

1.3 OBJECTIVE OF PROJECT

- Automated Test Pattern Generation: Create and implement automated test sequences to reduce manual intervention and increase throughput.
- Automated Data Analysis: Use MATLAB to automatically analyze test data and extract meaningful insights.
- High-Precision Measurement: Utilize MATLAB's advanced numerical capabilities to ensure precise measurement and analysis of IC performance.
- Error Reduction: Implement robust error-checking and validation routines to minimize errors in testing.
- Wide Coverage: Ensure that tests cover all relevant aspects of IC performance, including functional, electrical, and thermal properties.
- Edge Case Analysis: Identify and test edge cases and extreme conditions to ensure IC reliability under all possible scenarios.

1.4 OVERVIEW OF PROJECT

Chapter-1: It contains background, literature survey and objective of the IC tester using MATLAB.

Chapter-2: It consists of block diagram of IC tester using MATLAB**.**

Chapter-3: It comprises of circuit diagram, schematic diagram and working principle of IC tester using MATLAB.

Chapter-4: It has description of specification of all components used in IC tester using MATLAB.

Chapter-5: It presents result, conclusion and future scope of IC tester using MATLAB.

Chapter-6: It comprises of advantages and disadvantages of IC tester using MATLAB.

CHAPTER-2

BLOCK DIAGRAM OF IC TESTER USING MATLAB

2.1 BLOCK DIAGRAM

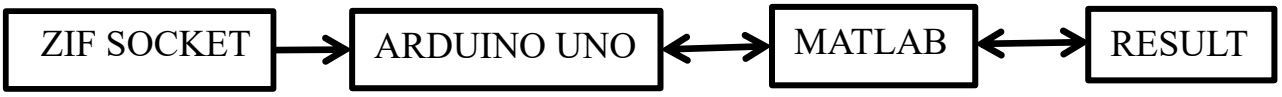

Fig 2.1: Block diagram of IC tester using MATLAB

Testing of digital electronic systems generally involves applying a set of test stimuli to inputs of the device-under-test (DUT) and analyzing responses of the system using a response analyzer. If the DUT generates correct output responses (also called the golden response) for all the input stimuli, the DUT is regarded as fault-free. Those DUTs that fail to meet the golden response are regarded as faulty or defective. This project describes a digital IC tester for testing 74xx series digital ICs using a MATLAB graphical user interface (GUI) drop-down menu based approach. Block diagram for testing a device is shown in Fig.

Ic testera 24-pin IC socket, a keyboard unit, a display unit and indicators. To test a particular digital IC, one needs to insert the IC into the IC socket and enter the IC number using the keyboard and then press the “ENTER” key. The IC number gets displayed in the 7-segment display unit.

CHAPTER-3

CIRCUIT DIAGRAM OF IC TESTER USING MATLAB

3.1 CIRCUIT DIAGRAM

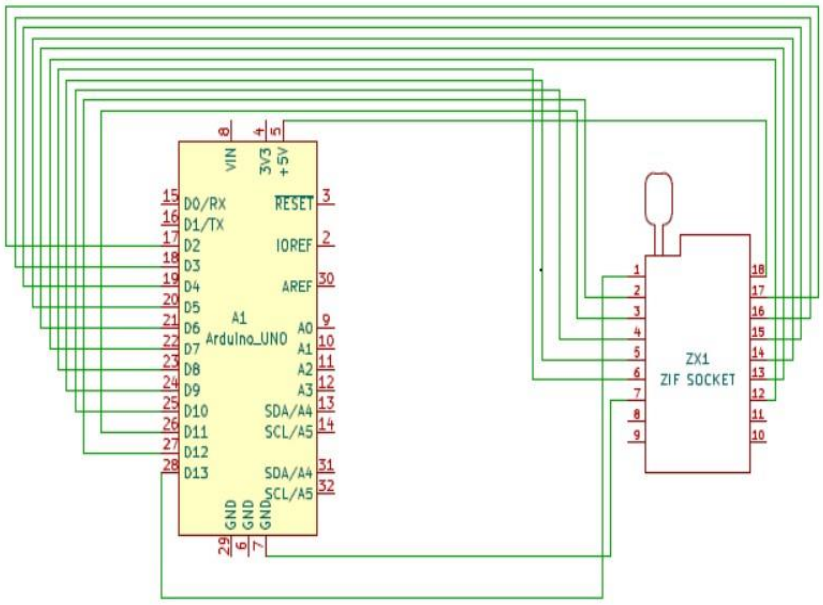

Fig 3.1: Circuit diagram of IC tester using MATLAB

Testing of digital electronic systems generally involves applying a set of test stimuli to inputs of the device-under-test (DUT) and analyzing responses of the system using a response analyzer. If the DUT generates correct output responses (also called the golden response) for all the input stimuli, the DUT is regarded as fault-free. Those DUTs that fail to meet the golden response are regarded as faulty or defective. This project describes a digital IC tester for testing 74xx series digital ICs using a MATLAB graphical user interface (GUI) drop-down menu based approach.

MATLAB acts as the test stimuli generator to the IC, which is the DUT. The GUI initiates communication with the Arduino and provides a user-friendly and interactive approach to conduct the test. The MATLAB source program (ic_tester.m) acts as the As mentioned earlier, MATLAB

is used to apply stimuli to the DUT (74xx series digital ICs) and also record the response of the DUT to stimuli. It then compares the response of the DUT with the correct/golden response to test whether the device is faulty or not. For a digital IC, the correct response is given in the form of a truth table. Acting as a response analyzer, the MATLAB verifies each and every possible outcome according to the truth table of a particular IC.74xx series ICs that can be tested by this project are 7400, 7402, 7404, 7408, 7432 and 7486. Truth tables for these ICs are shown on the next page.

3.2 METHODOLOGY

Creating an IC tester using MATLAB involves a comprehensive methodology that encompasses several key stages, each integral to ensuring the effectiveness and reliability of the testing process. It begins with a thorough requirement analysis to define the specific testing objectives, such as functional, electrical, and thermal tests, and to identify the necessary hardware and software tools, including MATLAB toolboxes and test equipment like signal generators and oscilloscopes.

The next phase involves designing and simulating the IC's behavior. Using MATLAB, engineers can create detailed behavioral models of the IC, allowing them to simulate expected performance under various conditions and to develop initial test scripts. These scripts are used in pre-test simulations to predict potential issues and refine test plans before physical testing begins. Following the design phase, the focus shifts to test pattern generation. MATLAB can automate the creation of test patterns based on the IC's design specifications, as well as develop custom patterns to address specific requirements or edge cases.

This automation significantly enhances the efficiency of the testing process. Integration with hardware is a critical step, where MATLAB interfaces with test equipment to control the hardware and acquire data in real-time. This integration allows for dynamic adjustment of test parameters based on intermediate results, ensuring more accurate and comprehensive testing. During test execution, the IC is subjected to the generated test patterns, and MATLAB scripts control the process while collecting data.

This data is systematically logged, including timestamps and relevant metadata, to facilitate detailed analysis later on. Data analysis is a major component, where MATLAB's

signal processing and statistical tools are employed to filter noise, extract relevant features, and perform in-depth statistical analysis to identify trends, correlations, and anomalies. Visualization tools in MATLAB create detailed plots and graphs, aiding in the interpretation of test results and highlighting key performance metrics of the IC.

Optimization is an ongoing process, where test procedures are continually refined to improve efficiency and coverage, and resource management ensures optimal use of testing time and hardware. The final stage involves rigorous validation and verification, ensuring that the IC meets all specified performance and reliability standards. This methodology leverages MATLAB's extensive capabilities in modeling, simulation, automation, data analysis, and visualization, making it a powerful tool for enhancing the accuracy, efficiency, and comprehensiveness of IC testing.

3.3 WORKING PRINCIPLE

An IC tester using MATLAB operates by interfacing MATLAB software with IC tester hardware to automate the testing and analysis of integrated circuits. The process begins with the development of test programs in MATLAB, where scripts are written to define the sequence of tests, the parameters for each test (such as voltage levels and timing), and the expected responses from the IC. The IC is connected to the tester hardware via a test fixture, ensuring correct connections. MATLAB scripts then configure the tester hardware to apply the specified stimuli to the IC.

The hardware measures the IC's responses to these stimuli and sends the data back to MATLAB for analysis. MATLAB algorithms process the collected data to determine whether the IC meets the desired specifications, enabling efficient and accurate verification of the IC's functionality. This setup allows for the automation of repetitive tasks, reducing human error and increasing testing throughput. Additionally, MATLAB's extensive library of functions facilitates complex data analysis, visualization, and reporting, making it easier to diagnose issues and verify compliance with performance

3.4 COMPONENTS REQUIRED

3.4.1 ARDUINO

Fig3.4.1.1 :ARDUINO UNO

The Arduino UNO is a standard board of Arduino. Here UNO means 'one' in Italian. It was named as UNO to label the first release of Arduino Software. It was also the first USB board released by Arduino. It is considered as the powerful board used in various projects. Arduino.cc developed the Arduino UNO board. Arduino UNO is based on an ATmega328P microcontroller. It is easy to use compared to other boards, such as the Arduino Mega board, etc. The board consists of digital and analog Input/Output pins (I/O), shields, and other circuits. The Arduino UNO includes 6 analog pin inputs, 14 digital pins, a USB connector, a power jack, and an ICSP (In-Circuit Serial Programming) header. It is programmed based on IDE, which stands for Integrated Development Environment. It can run on both online and offline platforms.

- **ATmega328 Microcontroller**- It is a single chip Microcontroller of the ATmel family. The processor code inside it is of 8-bit. It combines **Memory (SRAM, EEPROM, and Flash), Analog to Digital Converter, SPI serial ports, I/O lines, registers, timer, external and internal interrupts, and oscillator.**
- **ICSP pin** - The In-Circuit Serial Programming pin allows the user to program using the
- **Power LED Indicator**- The ON status of LED shows the power is activated. When the power is OFF, the LED will not light up.
- **Digital I/O pins**- The digital pins have the value HIGH or LOW. The pins numbered from D0 to D13 are digital pins.
- **TX and RX LED's**- The successful flow of data is represented by the lighting of these LED's.
- **AREF-** The Analog Reference (AREF) pin is used to feed a reference voltage to the Arduino UNO board from the external power supply.
- **Reset button**- It is used to add a Reset button to the connection.
- **USB**- It allows the board to connect to the computer. It is essential for the programming of the Arduino UNO board.
- **Crystal Oscillator**- The Crystal oscillator has a frequency of 16MHz, which makes the Arduino UNO a powerful board.
- **Voltage Regulator**- The voltage regulator converts the input voltage to 5V.
- **GND**- Ground pins. The ground pin acts as a pin with zero voltage.
- **Vin**- It is the input voltage.
- **Analog Pins**- The pins numbered from A0 to A5 are analog pins. The function of Analog pins is to read the analog sensor used in the connection. It can also act as GPIO (General Purpose Input Output) pins.

3.4.2 IC's

Fig 3.4.2.1:7402 Nor gate

7402 IC is a device containing four independent gates each of which performs the logic NOR function. 7402 package options include: plastic small outline, ceramic chip carriers, flat packages, plastic and ceramic DIPs. The SN7402, SN74LS02 and SN74S02 are characterized for operation from 0°C to 70°C.

Features

- Four Independent 2-Input NOR Gates
- Outputs Directly Interface to CMOS, NMOS and TTL
- Large Operating Voltage Range
- Wide Operating Conditions

Fig 3.4.2.2:7404 Not gate

NOT gate IC. It consists of six inverters which perform logical invert action. The output of an inverter is the complement of its input logic state, i.e. when input is high its output is low and vice versa.Six Hex Inverters in a 14-Pin DIP Package Outputs Directly Interface to CMOS, NMOS and TTL Large Operating Voltage Range Wide Operating Conditions

Operating Condition of IC 7404:

1. The power supply should be given to the IC from 4.5V DC to 5.25V DC
2. The IC will consider a signal as high when the voltage of the signal is above 2V
3. The IC will consider a signal as low when the voltage of the signal is below 0.8V
4. The operating temperature of the IC should be below the 70-degree centigrade

Characteristics:

1. IC 74LS04 can deliver -0.4 mA current when the output is high.
2. It can deliver 16 mA current when the output is low.
3. When the Vcc is 5V and the input signal is 5V then the IC draws 1 mA current.
4. When the Vcc is 5V and the input signal is 2.7V then the IC draws 20 to 40 micro-ampere currents.
5. When the Vcc is 5V and the input signal is 0.4V then the IC draws -1.6 mA current.

Application of IC 7404:

1. IC 7404 is mostly used for digital electronics projects.
2. They are also used in some electronic devices.
3. They are also used in Space instruments.

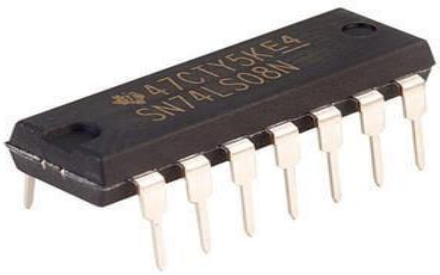

Fig 3.4.2.3:7408 And gate

7408 is Quad 2-Input AND Gate 14 Pin IC. The 74HC08 provides provides 4 independent 2-input AND gates with standard push-pull outputs. The device is designed for operation with a power supply range of 2.0V to 6.0V. It utilize advanced silicon-gate CMOS technology to achieve operating speeds similar to LS-TTL gates with the low power consumption of standard CMOS integrated circuits. The HC08 has buffered outputs, providing high noise immunity and the ability to drive 10 LS-TTL loads. The 74HC logic family is functionally as well as pin-out compatible with the standard 74LS logic family. All inputs are protected from damage due to static discharge by internal diode clamps to VCC and ground.

Specification:

- Type: DIP Bipolar
- Voltage Rating: 2.0 to 6.0V
- Temperature Rating: 0 to 70Deg C
- Number of: 14 Pins
- Mounting: Through Hole

Features:

- 4 AND gates in a 14-Pin DIP Package
- Outputs Directly Interface to CMOS, NMOS and TTL
- Large Operating Voltage Range
- Wide Operating Conditions

Fig 3.4.2.4:7486 X-Or gate

7486 is based on EXCLUSIVE-OR logic gate function which is very important and useful circuit which uses many types of computational circuit to perform EXCLUSIVE-OR logic function in that circuit.

7486 is a 14pin IC which is use to perform **E**XCLUSIVE-OR gate logic function in circuit, 7486 having 1 VCC and 1 GND pin, and 8 input pins and 4 output pins.

Input/output pins are placed in pair of 3 pins(2 pin for input and 1 for result/output), one IC can connect up-to 4 devices.

7486 works on 5volt(max 7volt) DC power supply, and it's input pins work on3.3volt(max 5.5volts), and its storage temp is -65C to +150C.

Features

- Four 2-Input Exclusive OR Gates
- Outputs Directly Interface to CMOS, NMOS and TTL
- Large Operating Voltage Range
- Wide Operating Conditions
- Not Recommended for New Designs Use 74LS86

CHAPTER-4

SOFTWARE - MATLAB

4.1 MATLAB

MATLAB (Matrix Laboratory) is a high-performance language and environment for numerical computation, visualization, and programming. Developed by MathWorks, it is widely used in academia, research, and industry for a variety of applications. MATLAB excels in matrix and vector computations, making it ideal for linear algebra, statistical analysis, and optimization problems. It offers powerful tools for visualizing data through 2D and 3D plots, histograms, and heat maps, aiding in data analysis and presentation.

It is used to design, implement, and test algorithms in a user-friendly environment, supporting rapid prototyping and experimentation. It is commonly used for simulating dynamic systems, such as control systems, signal processing, and financial models, using built-in functions and toolboxes. MATLAB provides a wide range of specialized toolboxes for various domains, including signal processing, image processing, machine learning, and robotics. It can generate C, C++, and HDL code from MATLAB code, enabling integration with other programming environments and hardware platforms.

MATLAB supports integration with other programming languages like Python, C/C++, Java, and .NET, allowing for flexible and extended functionality. Users can create custom graphical user interfaces (GUIs) to simplify interaction with programs and models. MATLAB is widely used in educational institutions for teaching mathematics, engineering, and science courses, as well as in research for developing and testing new theories and models.

In industry, MATLAB is used for tasks such as data analysis, system modeling, and automation in sectors like aerospace, automotive, finance, telecommunications, and biotechnology. Overall, MATLAB's combination of numerical computing power, versatile visualization capabilities, and extensive toolboxes makes it a valuable tool for a wide range of scientific and engineering applications.

4.2 MATLAB INTEGRATED WITH IC

Integrating MATLAB with an Integrated Circuit (IC) tester can significantly enhance the efficiency and capability of IC testing processes. This integration leverages MATLAB's robust computational and visualization tools alongside the hardware capabilities of the IC tester. Through MATLAB's Instrument Control Toolbox, users can establish communication with the IC tester via various interfaces such as GPIB, VISA, USB, Ethernet, or serial ports. This setup allows for sending test commands and receiving measurement data directly into MATLAB. Once the data is acquired, MATLAB's extensive array of numerical, statistical, and visualization functions can be employed to analyze and interpret the results.

Automation of test sequences can be achieved by scripting procedures that control the IC tester, perform measurements, and process the data, significantly reducing manual intervention and potential for error. Additionally, error handling mechanisms can be incorporated to manage communication issues and ensure the reliability of the tests. The integration also facilitates data storage, reporting, and further analysis by connecting MATLAB to databases or exporting results to other formats. This seamless integration results in a powerful and flexible testing environment that enhances the accuracy, efficiency, and depth of IC testing and analysis.

Integrating MATLAB with an Integrated Circuit (IC) tester further extends its utility by allowing for real-time monitoring and control of the testing process. MATLAB's interactive environment enables engineers to tweak test parameters on-the-fly and immediately observe the impact on test results. This real-time interaction is crucial for debugging and optimizing IC designs. Additionally, MATLAB's advanced machine learning and artificial intelligence toolboxes can be applied to the test data to identify patterns and anomalies that might not be apparent through traditional analysis methods.

Finally, MATLAB’s capabilities in reporting and documentation help in generating detailed, professional reports that can include plots, statistical summaries, and other critical insights derived from the test data. These reports can be automatically generated and customized to meet the specific needs of stakeholders, such as design engineers, quality assurance teams, or management.

CHAPTER-5

RESULT AND DISCUSSION

5.1 RESULT

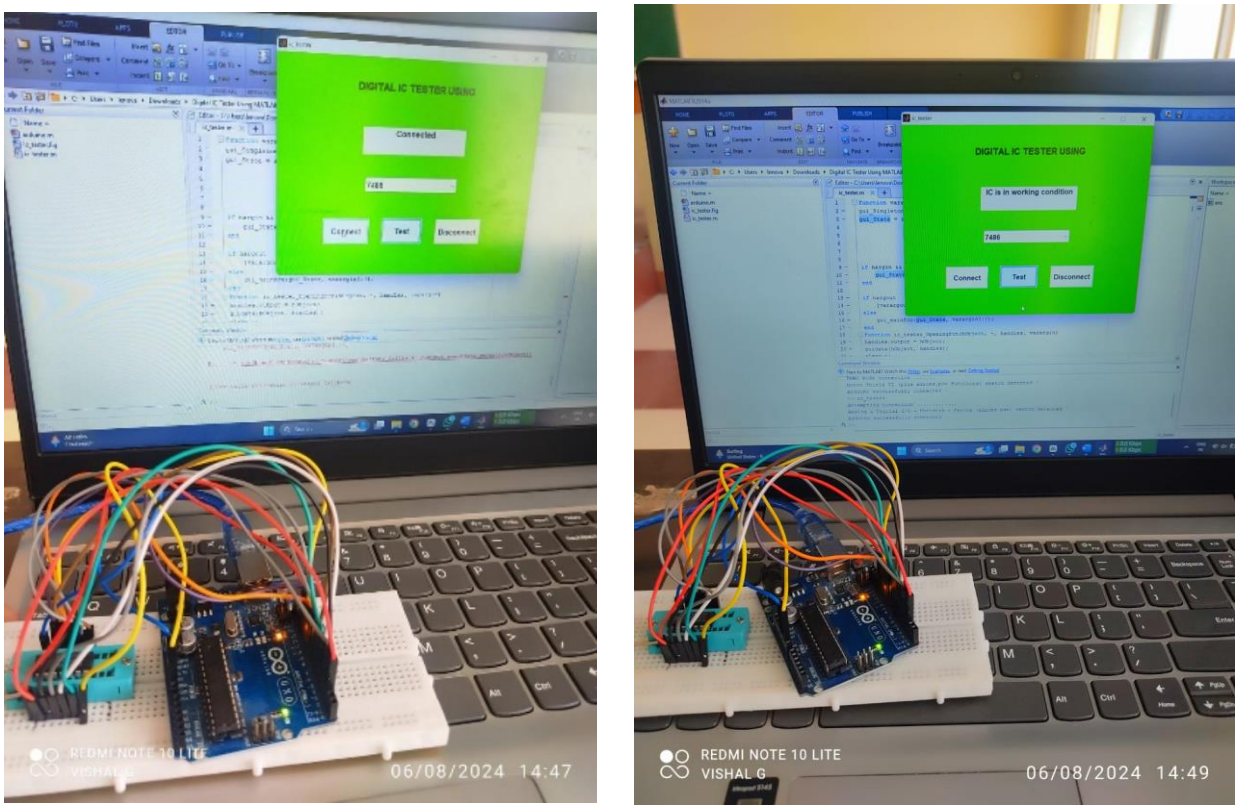

Fig 5.1: IC tester using MATLAB

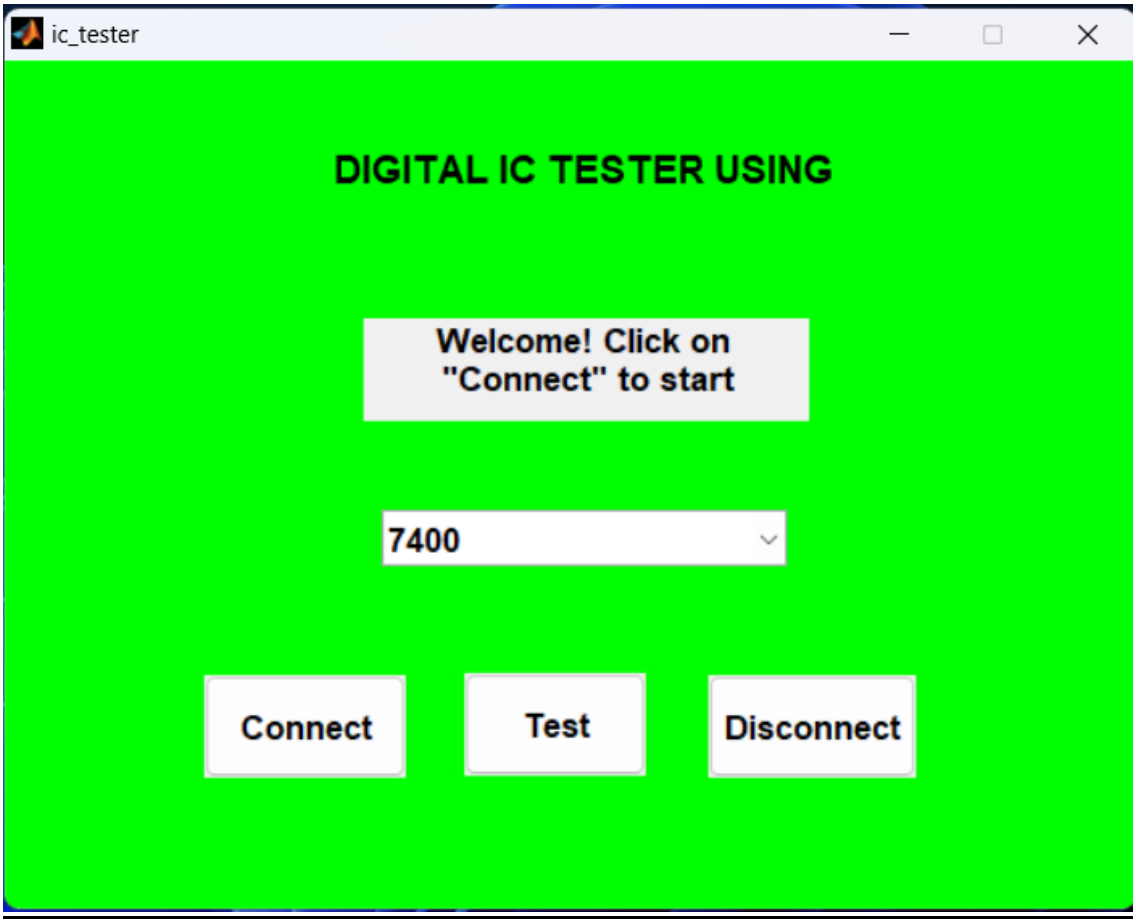

Fig 5.4:Output screen of IC tester using MATLAB

TRUTH TABLES FOR 74 SERIES DIGITAL ICs

7400 Quad Two-Input NAND Gate

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

7402 Quad Two-Input NOR Gate

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

7404 Hex Inverter

A	Y
0	1
1	0

7408 Quad Two-Input AND Gate

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

7432 Quad Two-Input OR Gate

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

7486 Quad Two-Input XOR Gate

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Fig 5.3: Truth-table for 74 series digital IC's

5.2 CONCLUSION

A new low-cost platform for the test and characterization of circuit IP prototypes has been developed. The new platform would allow IP developers to test the functionality and speed of their IP prototypes. It supports unlimited number of IPs per IC and unlimited number of ports/port widths per IP with constant interfaces. Hence the platform itself does not need any configuration or design change to test any circuit. The new concept has been successfully verified using a complete FPGA prototype and specially-developed user-interface software. Also a complete layout of the support circuitry that is included on the chip with the IUTs was generated using a150 nm technology. It shows that the area overhead of this circuitry is insignificant.

5.3 FUTURE SCOPE

- Predictive Maintenance: Using machine learning algorithms in MATLAB, predictive models can be developed to anticipate failures and reduce downtime.
- Anomaly Detection: MATLAB can be used to implement sophisticated anomaly detection algorithms that identify defects or unusual patterns in IC behavior.
- Automated Test Pattern Generation: MATLAB can automate the generation of test patterns for ICs, making the testing process more efficient.
- Test Optimization: Optimization algorithms in MATLAB can be used to minimize the number of tests required while maximizing coverage.
- Hardware-in-the-Loop (HIL) Testing: MATLAB can integrate with HIL systems to test ICs in real-time with actual hardware, improving the accuracy and reliability of tests.
- Behavioral Modeling: MATLAB can simulate and model the behavior of ICs under different conditions to predict performance and identify potential issues before physical testing.

- Thermal and Power Analysis: MATLAB can simulate thermal and power characteristics of ICs, crucial for ensuring they operate within safe limits.
- Noise Analysis and Filtering: Advanced signal processing techniques in MATLAB can be used to analyze and filter out noise in the signals from ICs, ensuring more accurate test results.
- Frequency Response Analysis: MATLAB can perform frequency response analysis to assess the performance of ICs in different frequency ranges.
- EDA Tools Integration: MATLAB can integrate with Electronic Design Automation (EDA) tools to streamline the testing process from design to implementation.
- Cloud and IoT Integration: MATLAB's capabilities can be extended to cloud platforms and IoT devices, enabling remote and scalable IC testing.
- Advanced Visualization: MATLAB’s powerful visualization tools can create detailed plots and graphs to better understand IC performance and test results.
- Automated Reporting: Generate automated reports with MATLAB, summarizing test results, trends, and insights for easy interpretation.

CHAPTER-6

ADVANTAGES AND DISADVANTAGES

6.1 ADVANTAGES

1. Versatility: MATLAB's extensive toolboxes and functions allow for testing various IC types and parameters.
2. Rapid prototyping: The high-level language enables quick implementation and modification of testing algorithms.
3. Powerful data analysis: MATLAB excels at processing and interpreting complex data from IC tests.
4. Excellent visualization: Built-in plotting tools make it easy to create clear, informative graphs of test results.
5. Hardware integration: MATLAB supports communication with various hardware, including Arduino Uno.
6. Signal processing capabilities: Advanced signal generation and analysis functions are readily available.
7. User interface development: MATLAB's App Designer facilitates the creation of user-friendly GUIs.
8. Automation: Easy scripting of test sequences for efficient, repeatable testing.
9. Educational value: MATLAB's widespread use in academia makes the project valuable for learning.
10. Extensive documentation: MATLAB offers comprehensive documentation and examples for reference.
11. Community support: A large user base provides resources and solutions to common problems.
12. All-in-one environment: MATLAB combines coding, analysis, and visualization in a single platform.

6.2 DISADVANTAGES

1. Performance limitations: MATLAB may be slower than lower-level languages for real-time testing of high-speed ICs.
2. Deployment challenges: Distributing MATLAB-based applications to users without MATLAB installed can be complicated.
3. Limited low-level hardware control: MATLAB may not provide as fine-grained control over hardware as some lower-level languages.
5. Dependency on toolboxes: Some advanced features may require additional toolboxes, increasing costs.
6. Limited real-time capabilities: For very high-speed IC testing, MATLAB's real-time performance may be insufficient.
7. Debugging complexity: Debugging large MATLAB projects can sometimes be more challenging than in traditional programming environments.
8. Hardware interfacing limitations: Some specialized IC testing hardware may lack direct MATLAB support.

CHAPTER-7

APPLICATIONS

1. MATLAB can be used to analyse the quality of electrical signals in ICs, checking for noise, crosstalk, and other signal integrity issues.
2. MATLAB helps in characterizing the performance of ICs by analysing parameters like speed, power consumption, and thermal performance under different operating conditions.
3. MATLAB can automate the functional testing of ICs, ensuring that all components perform as expected and meet design specifications.
4. MATLAB's data analysis and machine learning tools can be used to identify and diagnose faults in ICs, improving the accuracy and efficiency of the testing process.
5. MATLAB can simulate the behavior of IC designs before fabrication, verifying that the design meets the required specifications and functionality.
6. MATLAB can automate the measurement and analysis of various electrical parameters (such as voltage, current, and resistance) to ensure they fall within acceptable ranges.
7. MATLAB can analyse data from multiple ICs to identify trends and patterns that impact yield, helping in process optimization and quality control.
8. MATLAB can be used to monitor and analyse the results of burn-in tests, where ICs are subjected to elevated stress conditions to weed out early failures.
9. MATLAB can help design and analyse reliability tests, predicting the long-term performance and failure rates of ICs under different conditions.
10. MATLAB can interface with ATE systems to control test sequences, acquire data, and analyse results, streamlining the overall test process.
11. MATLAB can perform detailed power analysis on ICs, helping to optimize power consumption and identify areas for improvement.
12. MATLAB can be used to analyse the performance of ICs under various environmental conditions, such as temperature, humidity, and vibration.

REFERENCES

[1] Prof. D. G. Kanade, Nikhil Zambare and Krishna Rathode, Department of Electronics, Vishwakarma Institute of Technology, Pune, Maharashtra, India.

[2] Maribelle D. Pabiania , Joanne T. Peralta, Leo Anthony T. De Luna,Phillipines.

[3] Mirza Shoaib Ahmed, Iqbal Muhammad Umair, Kashif Mehboob NED University of Engineering and Technology, Karachi**.**

[4] Yasir Hashim, Marwa Awni, Abdullah Mufeed Computer Engineering Department, Faculty of Engineering, Tishk International University, Erbil, Iraq.

[5] Muhammad E. S. Elrabaa, Amran A. Al-Aghbari, and Mohammed A. Al-Asli Computer Engineering Department, King Fahd University for Petroleum and Minerals Dhahran, Saudi Arabia.

ANNEXURE

```
function varargout = ic_tester(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ic_tester_OpeningFcn, ...
                   'gui_OutputFcn',  @ic_tester_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
function ic_tester_OpeningFcn(hObject, ~, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
clear n;
set(handles.text2,'String','Welcome! Click on "Connect" to start');
function varargout = ic_tester_OutputFcn(~, ~, handles)
varargout{1} = handles.output;
function popupmenu1_Callback(hObject, ~, ~)
global n;
contents = get(hObject,'Value');
switch contents

    case 2
        n=1;

    case 3
        n=2;

    case 4
        n=3;

    case 5
        n=4;

    case 6
        n=5;
```

```
    case 7
        n=6;
end
function popupmenu1_CreateFcn(hObject, ~, ~)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function pushbutton1_Callback(~, ~, handles)
set(handles.text2,'String','Connecting...');
clear a;
global a;
a = arduino('COM7');
set(handles.text2,'String','Connected');
function pushbutton2_Callback(~, ~, handles)
global a;
global n;
if (n==1)
    a.pinMode(13,'output');
    a.pinMode(2,'output');
    a.pinMode(3,'input');
    a.pinMode(4,'output');
    a.pinMode(5,'output');
    a.pinMode(6,'input');
    a.pinMode(7,'input');
    a.pinMode(8,'output');
    a.pinMode(9,'output');
    a.pinMode(10,'input');
    a.pinMode(11,'output');
    a.pinMode(12,'output');
    a.digitalWrite(13,0);
    a.digitalWrite(2,0);
    a.digitalWrite(4,0);
    a.digitalWrite(5,0);
    a.digitalWrite(8,0);
    a.digitalWrite(9,0);
    a.digitalWrite(11,0);
    a.digitalWrite(12,0);
    pause(0.1);
    t11=a.digitalRead(3);
    t12=a.digitalRead(6);
    t13=a.digitalRead(7);
    t14=a.digitalRead(10);
    if (t11==1 && t12==1 && t13==1 && t14==1)
        a.digitalWrite(13,0);
```

```
        a.digitalWrite(2,1);
        a.digitalWrite(4,0);
        a.digitalWrite(5,1);
        a.digitalWrite(8,0);
        a.digitalWrite(9,1);
        a.digitalWrite(11,0);
        a.digitalWrite(12,1);
        pause(0.1);
t21=a.digitalRead(3);
        t22=a.digitalRead(6);
        t23=a.digitalRead(7);
        t24=a.digitalRead(10);
        if (t21==1 && t22==1 && t23==1 && t24==1)
            a.digitalWrite(13,1);
            a.digitalWrite(2,0);
            a.digitalWrite(4,1);
            a.digitalWrite(5,0);
            a.digitalWrite(8,1);
            a.digitalWrite(9,0);
            a.digitalWrite(11,1);
            a.digitalWrite(12,0);
            pause(0.1);
            t31=a.digitalRead(3);
            t32=a.digitalRead(6);
            t33=a.digitalRead(7);
            t34=a.digitalRead(10);
            if (t31==1 && t32==1 && t33==1 && t34==1)
                a.digitalWrite(13,1);
                a.digitalWrite(2,0);
                a.digitalWrite(4,1);
                a.digitalWrite(5,0);
                a.digitalWrite(8,1);
                a.digitalWrite(9,0);
                a.digitalWrite(11,1);
                a.digitalWrite(12,0);
                pause(0.1);
                t41=a.digitalRead(3);
                t42=a.digitalRead(6);
                t43=a.digitalRead(7);
                t44=a.digitalRead(10);
                if (t41==1 && t42==1 && t43==1 && t44==1)
                    set(handles.text2,'String','IC is in working condition');
                else
                    set(handles.text2,'String','Faulty IC');
                end
            else
```

```
                set(handles.text2,'String','Faulty IC');
            end
        else
            set(handles.text2,'String','Faulty IC');
        end
    else
        set(handles.text2,'String','Faulty IC');
    end
end
if (n==2)
    a.pinMode(13,'input');
    a.pinMode(2,'output');
    a.pinMode(3,'output');
    a.pinMode(4,'input');
    a.pinMode(5,'output');
    a.pinMode(6,'output');
    a.pinMode(7,'output');
    a.pinMode(8,'output');
    a.pinMode(9,'input');
    a.pinMode(10,'output');
    a.pinMode(11,'output');
    a.pinMode(12,'input');
    a.digitalWrite(2,0);
    a.digitalWrite(3,0);
    a.digitalWrite(5,0);
    a.digitalWrite(6,0);
    a.digitalWrite(7,0);
    a.digitalWrite(8,0);
    a.digitalWrite(10,0);
    a.digitalWrite(11,0);
    pause(0.1);
    t11=a.digitalRead(13);
    t12=a.digitalRead(4);
    t13=a.digitalRead(9);
    t14=a.digitalRead(12);
    if (t11==1 && t12==1 && t13==1 && t14==1)
        a.digitalWrite(2,0);
        a.digitalWrite(3,1);
        a.digitalWrite(5,0);
        a.digitalWrite(6,1);
        a.digitalWrite(7,0);
        a.digitalWrite(8,1);
        a.digitalWrite(10,0);
        a.digitalWrite(11,1);
        pause(0.1);
        t21=a.digitalRead(13);
```

```
        t22=a.digitalRead(4);
        t23=a.digitalRead(9);
        t24=a.digitalRead(12);
        if (t21==0 && t22==0 && t23==0 && t24==0)
            a.digitalWrite(2,1);
            a.digitalWrite(3,0);
            a.digitalWrite(5,1);
            a.digitalWrite(6,0);
            a.digitalWrite(7,1);
            a.digitalWrite(8,0);
            a.digitalWrite(10,1);
            a.digitalWrite(11,0);
            pause(0.1);
            t31=a.digitalRead(13);
            t32=a.digitalRead(4);
a.digitalWrite(5,1);
    a.digitalWrite(8,1);
    a.digitalWrite(10,1);
    a.digitalWrite(12,1);
    pause(0.1);
    t11=a.digitalRead(2);
    t12=a.digitalRead(4);
    t13=a.digitalRead(6);
    t14=a.digitalRead(7);
    t15=a.digitalRead(9);
    t16=a.digitalRead(11);
    if (t11==0 && t12==0 && t13==0 && t14==0 && t15==0 && t16==0)
        a.digitalWrite(13,0);
        a.digitalWrite(3,0);
        a.digitalWrite(5,0);
        a.digitalWrite(8,0);
        a.digitalWrite(10,0);
        a.digitalWrite(12,0);
        pause(0.1);
        t11=a.digitalRead(2);
        t12=a.digitalRead(4);
        t13=a.digitalRead(6);
        t14=a.digitalRead(7);
        t15=a.digitalRead(9);
        t16=a.digitalRead(11);
        if (t11==1 && t12==1 && t13==1 && t14==1 && t15==1 && t16==1)
            set(handles.text2,'String','IC is in working condition');
        else
            set(handles.text2,'String','Faulty IC');
        end
    else
```

```
        set(handles.text2,'String','Faulty IC');
    end
end

if (n==4)
    a.pinMode(13,'output');
    a.pinMode(2,'output');
    a.pinMode(3,'input');
    a.pinMode(4,'output');
    a.pinMode(5,'output');
    a.pinMode(6,'input');
    a.pinMode(7,'input');
    a.pinMode(8,'output');
    a.pinMode(9,'output');
    a.pinMode(10,'input');
    a.pinMode(11,'output');
    a.pinMode(12,'output');
    a.digitalWrite(13,0);
a.digitalWrite(2,0);
    a.digitalWrite(4,0);
    a.digitalWrite(5,0);
    a.digitalWrite(8,0);
    a.digitalWrite(9,0);
    a.digitalWrite(11,0);
    a.digitalWrite(12,0);
    pause(0.1);
    t11=a.digitalRead(3);
    t12=a.digitalRead(6);
    t13=a.digitalRead(7);
    t14=a.digitalRead(10);
    if (t11==0 && t12==0 && t13==0 && t14==0)
        a.digitalWrite(13,0);
        a.digitalWrite(2,1);
        a.digitalWrite(4,0);
        a.digitalWrite(5,1);
        a.digitalWrite(8,0);
        a.digitalWrite(9,1);
        a.digitalWrite(11,0);
        a.digitalWrite(12,1);
        pause(0.1);
        t21=a.digitalRead(3);
        t22=a.digitalRead(6);
        t23=a.digitalRead(7);
        t24=a.digitalRead(10);
        if (t21==0 && t22==0 && t23==0 && t24==0)
            a.digitalWrite(13,1);
```

```
            a.digitalWrite(2,0);
            a.digitalWrite(4,1);
            a.digitalWrite(5,0);
            a.digitalWrite(8,1);
            a.digitalWrite(9,0);
            a.digitalWrite(11,1);
            a.digitalWrite(12,0);
            pause(0.1);
            t31=a.digitalRead(3);
            t32=a.digitalRead(6);
            t33=a.digitalRead(7);
            t34=a.digitalRead(10);
            if (t31==0 && t32==0 && t33==0 && t34==0)
                a.digitalWrite(13,1);
                a.digitalWrite(2,1);
                a.digitalWrite(4,1);
                a.digitalWrite(5,1);
                a.digitalWrite(8,1);
                a.digitalWrite(9,1);
                a.digitalWrite(11,1);
                a.digitalWrite(12,1);
                pause(0.1);
                t41=a.digitalRead(3);
                t42=a.digitalRead(6);
t43=a.digitalRead(7);
                t44=a.digitalRead(10);
                if (t41==1 && t42==1 && t43==1 && t44==1)
                    set(handles.text2,'String','IC is in working condition');
                else
                    set(handles.text2,'String','Faulty IC');
                end
            else
                set(handles.text2,'String','Faulty IC');
            end
        else
            set(handles.text2,'String','Faulty IC');
        end
    else
        set(handles.text2,'String','Faulty IC');
    end
end

if (n==5)
    a.pinMode(13,'output');
    a.pinMode(2,'output');
    a.pinMode(3,'input');
```

```
a.pinMode(4,'output');
a.pinMode(5,'output');
a.pinMode(6,'input');
a.pinMode(7,'input');
a.pinMode(8,'output');
a.pinMode(9,'output');
a.pinMode(10,'input');
a.pinMode(11,'output');
a.pinMode(12,'output');
a.digitalWrite(13,0);
a.digitalWrite(2,0);
a.digitalWrite(4,0);
a.digitalWrite(5,0);
a.digitalWrite(8,0);
a.digitalWrite(9,0);
a.digitalWrite(11,0);
a.digitalWrite(12,0);
pause(0.1);
t11=a.digitalRead(3);
t12=a.digitalRead(6);
t13=a.digitalRead(7);
t14=a.digitalRead(10);
if (t11==0 && t12==0 && t13==0 && t14==0)
    a.digitalWrite(13,0);
    a.digitalWrite(2,1);
    a.digitalWrite(4,0);
    a.digitalWrite(5,1);
    a.digitalWrite(8,0);
    a.digitalWrite(9,1);
    a.digitalWrite(11,0);
    a.digitalWrite(12,1);
    pause(0.1);
    t21=a.digitalRead(3);
    t22=a.digitalRead(6);
    t23=a.digitalRead(7);
    t24=a.digitalRead(10);
    if (t21==1 && t22==1 && t23==1 && t24==1)
        a.digitalWrite(13,1);
        a.digitalWrite(2,0);
        a.digitalWrite(4,1);
        a.digitalWrite(5,0);
        a.digitalWrite(8,1);
        a.digitalWrite(9,0);
        a.digitalWrite(11,1);
        a.digitalWrite(12,0);
        pause(0.1);
```

```
            t31=a.digitalRead(3);
            t32=a.digitalRead(6);
            t33=a.digitalRead(7);
            t34=a.digitalRead(10);
            if (t31==1 && t32==1 && t33==1 && t34==1)
                a.digitalWrite(13,1);
                a.digitalWrite(2,1);
                a.digitalWrite(4,1);
                a.digitalWrite(5,1);
                a.digitalWrite(8,1);
                a.digitalWrite(9,1);
                a.digitalWrite(11,1);
                a.digitalWrite(12,1);
                pause(0.1);
                t41=a.digitalRead(3);
                t42=a.digitalRead(6);
                t43=a.digitalRead(7);
                t44=a.digitalRead(10);
                if (t41==1 && t42==1 && t43==1 && t44==1)
                    set(handles.text2,'String','IC is in working condition');
                else
                    set(handles.text2,'String','Faulty IC');
                end
            else
                set(handles.text2,'String','Faulty IC');
            end
        else
            set(handles.text2,'String','Faulty IC');
        end
    else
        set(handles.text2,'String','Faulty IC');
    end
end

if (n==6)
    a.pinMode(13,'output');
    a.pinMode(2,'output');
    a.pinMode(3,'input');
    a.pinMode(4,'output');
    a.pinMode(5,'output');
    a.pinMode(6,'input');
    a.pinMode(7,'input');
    a.pinMode(8,'output');
    a.pinMode(9,'output');
    a.pinMode(10,'input');
    a.pinMode(11,'output');
```

```
    a.pinMode(12,'output');
    a.digitalWrite(13,0);
    a.digitalWrite(2,0);
    a.digitalWrite(4,0);
    a.digitalWrite(5,0);
    a.digitalWrite(8,0);
    a.digitalWrite(9,0);
    a.digitalWrite(11,0);
    a.digitalWrite(12,0);
    pause(0.1);
    t11=a.digitalRead(3);
    t12=a.digitalRead(6);
    t13=a.digitalRead(7);
    t14=a.digitalRead(10);
    if (t11==0 && t12==0 && t13==0 && t14==0)
        a.digitalWrite(13,0);
        a.digitalWrite(2,1);
        a.digitalWrite(4,0);
        a.digitalWrite(5,1);
        a.digitalWrite(8,0);
        a.digitalWrite(9,1);
        a.digitalWrite(11,0);
        a.digitalWrite(12,1);
        pause(0.1);
        t21=a.digitalRead(3);
        t22=a.digitalRead(6);
        t23=a.digitalRead(7);
        t24=a.digitalRead(10);
        if (t21==1 && t22==1 && t23==1 && t24==1)
            a.digitalWrite(13,1);
            a.digitalWrite(2,0);
            a.digitalWrite(4,1);
            a.digitalWrite(5,0);
            a.digitalWrite(8,1);
a.digitalWrite(9,0);
            a.digitalWrite(11,1);
            a.digitalWrite(12,0);
            pause(0.1);
            t31=a.digitalRead(3);
            t32=a.digitalRead(6);
            t33=a.digitalRead(7);
            t34=a.digitalRead(10);
            if (t31==1 && t32==1 && t33==1 && t34==1)
                a.digitalWrite(13,1);
                a.digitalWrite(2,1);
                a.digitalWrite(4,1);
```

```
            a.digitalWrite(5,1);
            a.digitalWrite(8,1);
            a.digitalWrite(9,1);
            a.digitalWrite(11,1);
            a.digitalWrite(12,1);
            pause(0.1);
            t41=a.digitalRead(3);
            t42=a.digitalRead(6);
            t43=a.digitalRead(7);
            t44=a.digitalRead(10);
            if (t41==0 && t42==0 && t43==0 && t44==0)
                set(handles.text2,'String','IC is in working condition');
            else
                set(handles.text2,'String','Faulty IC');
            end
        else
            set(handles.text2,'String','Faulty IC');
        end
    else
        set(handles.text2,'String','Faulty IC');
    end
else
    set(handles.text2,'String','Faulty IC');
end
end
function pushbutton3_Callback(~, ~, handles)
set(handles.text2,'String','Disconnected');
clear all;
function edit3_Callback(~, ~, handles)
function edit3_CreateFcn(hObject, ~, ~)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```