EMBEDDED TESTING

AN INDUSTRIAL TRAINING REPORT

Submitted by

EZHILARASI M(513121106025)
MANIMARAN V(513121106302)
NARMADHA S(513121104712)
RAJESHKUMAR V(513121106075)
SARUMATHI E(513121104038)
SUSHMITHA S(513121106097)

in complete fulfillment for the award of the certificate

of

COURSE COMPLETION

IN THE AREA OF
"AGNI"

VAAYUSASTRA AEROSPACE PRIVATE LIMITED

IITM Research Park, Chennai.

VAAYUSASTRA AEROSPACE PRIVATE LIMITED
IITM Research Park, Chennai.

BONAFIDE CERTIFICATE

Certified that this industrial training report **"EMBEDDED TESTING"** is the bonafide work of **"EZHILARASI M", "MANIMARAN V", "NARMADHA S", "RAJESHKUMAR V", "SARUMATHI E", "SUSHMITHA S"** who carried out of the Industrial Training & Project Work under my supervision.

Mr. JAGADEESH KANNA, M.E.,
Founder & CEO,
Vaayusastra Aerospace Pvt. Ltd,
IITM Research Park,
Tharamani,
Chennai – 600 113.

Mr. P.GOWTHAM
R&D Engineer
Vaayusastra Aerospace Pvt. Ltd,
IITM Research Park,
Tharamani,
Chennai – 600 113.

Submitted for the Industrial Training Report and Evaluation held on _____

Mr. JAGADEESH KANNA, M.E.,
CEO & Chief Examiner
Vaayusastra Aerospace Pvt. Ltd,
Chennai – 600 113.

Mr.Karthikeyan Sundaresan,
Chief Operations Officer,
Vaayusastra Aerospace Pvt. Ltd,
Chennai – 600 113.

ACKNOWLEDGEMENT

I am expressing my prime gratitude to the LORD ALMIGHTY and MY PARENTS for giving me the confidence and strength for the successful completion of this project. However, it would not have been possible without the kind support and help of many individuals and organizations.

I would like to thank, **Mr.JAGADEESH KANNA**, Founder & CEO, Vaayusastra Aerospace Pvt. Ltd., for giving us this opportunity and necessary advice and guidance andarrangement of all facilities to provide us wonderful learning experience.

I would like to express my special gratitude and thanks to our beloved Principal

Dr.P.K.PALANI,M.E**.,** Ph.D., for his zealous support and for providing constructive feedback and approval of Industrial Training.

I am highly indebted to pay my sincere gratitude to our Head of Department

Dr.S. LETITIA ,M.E., Ph.D., for her guidance, constant supervision and also her support in completing the project.

I am highly indebted to pay my Sincere gratitude to our Training Guide & Coordinator **Mr. P.GOWTHAM** for his guidance, constant supervision and also his support incompleting the Training.

EZHILARASI M(513121106025)
MANIMARAN V(513121106302)
NARMADHA S(513121104712)
RAJESHKUMAR V(513121106075)
SARUMATHI E(513121104038)
SUSHMITHA S(513121106097)

CHAPTER NO	CONTENT	PAGE NO
	ABSTRACT	**1**
1 1.1 1.2 1.3	OVERVIEW OF EMBEDDED SYSTEMS Embedded components Unit level testing Programming exercise	**2**
2 2.1 2.2 2.3	CHIP LEVEL TESTING Microprocessor and microcontroller Active and Passive Components Arduino IDE	**17**
3 3.1 3.2 3.3	BOARD LEVEL TESTING Sensor and Actuators Sensor Testing Integration of Sensors	**27**
4 4.1	CASE STUDY Assistive Technology For Visual Impairment	**52**
5 5.1 5.2 5.3	PROJECT DEVELOPMENT Problem Statement Implementation Output	**56**
6	CONCLUSION REFERENCES	**66**

ABSTRACT

This report documents the completion of an internship in Embedded Testing at Vaayusastra Aerospace, a leading organization in the development and testing of embedded systems. The internship provided hands-on experience in various aspects of embedded system testing, including test plan creation, execution, automation, and result analysis.

The primary focus was on understanding the intricacies of embedded systems and how they interact with both software and hardware components. My responsibilities included designing and implementing test cases to validate system functionality, performance, and reliability. The internship also involved the use of specialized testing tools and equipment, such as oscilloscopes, logic analyzers, and simulation software, to assess the performance of embedded modules under different conditions.

One of the key projects I worked on involved automating the testing process for a critical embedded module, significantly reducing manual testing time and increasing the accuracy of the results. This project allowed me to deepen my understanding of test automation frameworks and scripting languages such as Python, which were used to create and execute test scripts.

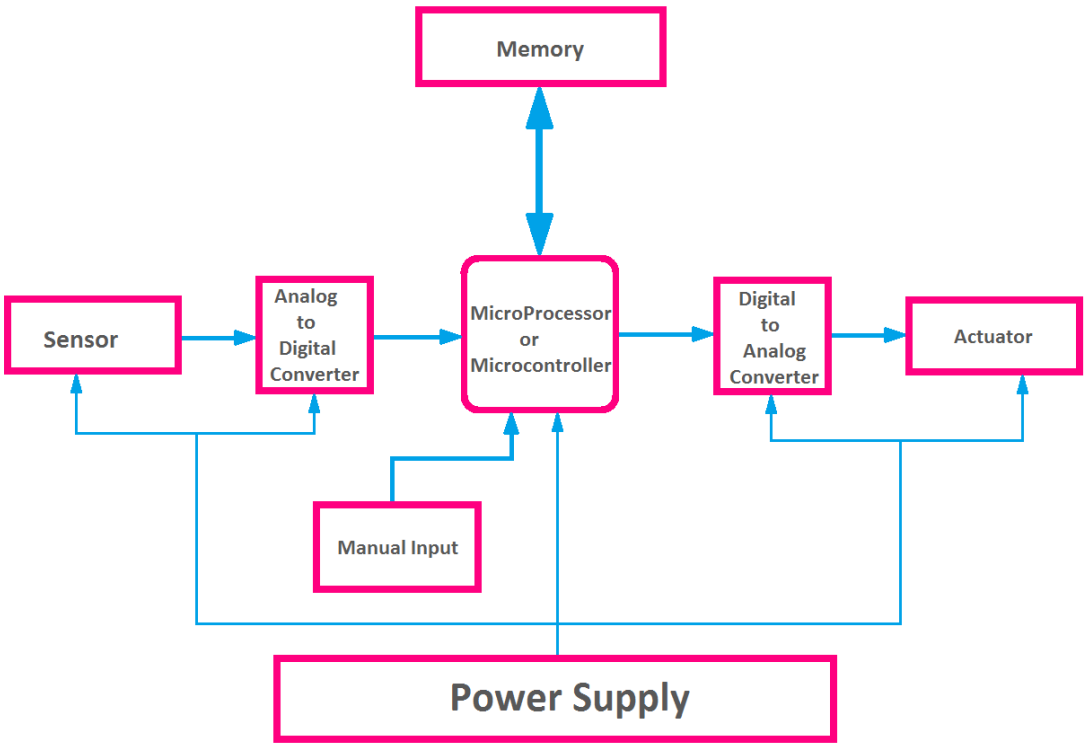
Collaboration with cross-functional teams was another integral part of the internship, providing insight into the importance of communication and teamwork in a real-world engineering environment. Regular interactions with software developers, hardware engineers, and quality assurance teams helped me gain a comprehensive view of the product development lifecycle and the role of testing within it.

Overall, the internship was a valuable learning experience that enhanced my technical skills in embedded systems testing and provided practical exposure to industry-standard tools and methodologies. The knowledge and experience gained during this period have significantly contributed to my professional development and have prepared me for future challenges in the field of embedded systems engineering

OVERVIEW OF EMBEDDED SYSTEMS

Embedded systems are specialized computing units designed to perform specific functions within larger systems, often with strict requirements for efficiency, reliability, and real-time operation. Unlike general-purpose computers, embedded systems are optimized to handle particular tasks, operating with limited resources such as processing power, memory, and energy.These systems typically consist of a microcontroller or microprocessor, memory, and input/output interfaces, all integrated with firmware that dictates their operations. The hardware and software work together seamlessly to execute tasks like controlling machinery, monitoring environmental conditions, or managing communications.

A crucial aspect of embedded systems is their ability to function in real-time, meaning they must process inputs and generate outputs within stringent time constraints. This capability is vital in applications such as automotive safety systems, medical devices, and industrial automation, where delays can have serious consequences.



Embedded systems vary in complexity, from small-scale systems used in simple devices like digital watches and household appliances to medium-scale systems with

networking capabilities, such as smart home devices, to large-scale systems that power advanced applications in aerospace, telecommunications, and defense.The design of an embedded system involves careful selection of hardware components and development of efficient firmware, ensuring the system meets specific performance, reliability, and cost requirements. Designers must consider factors like power efficiency, physical size, and the ability to withstand environmental stresses.

Development of embedded systems typically utilizes specialized tools, including Integrated Development Environments (IDEs), cross-compilers, and debugging equipment. Hardware testing might involve using oscilloscopes, logic analyzers, and in-circuit emulators to ensure the system operates correctly.

Embedded systems are ubiquitous in modern technology, with applications spanning consumer electronics, automotive controls, industrial machinery, healthcare devices, and telecommunications infrastructure. As technology advances, these systems are becoming increasingly complex, with emerging trends such as the integration of artificial intelligence (AI), the proliferation of Internet of Things (IoT) devices, and the need for ultra-low-power operation. These advancements are driving the future of embedded systems, making them even more critical to the functioning of modern society.

1.1 Embedded components

Embedded systems are integral to modern electronics, comprising various hardware and software components that work together to perform specific tasks. When creating a report on embedded systems, the following components should be highlighted:

Microcontroller:

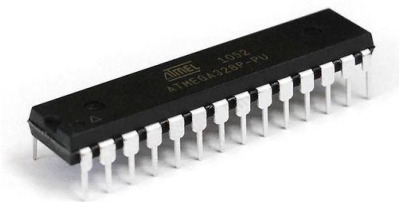
A microcontroller is a compact integrated circuit designed to govern a specific operation in an embedded system. Unlike general-purpose processors, microcontrollers are optimized for embedded applications, where they control processes or functions within a larger system. They typically contain a processor, memory, and input/output peripherals on a single chip, making them self-sufficient for their tasks.

In an embedded system, a microcontroller is tasked with real-time operations, such as monitoring sensor inputs, controlling actuators, and managing communications. For instance, in an automotive system, microcontrollers manage engine control, airbags, and infotainment systems, while in consumer electronics, they can be found in devices like washing machines, microwaves, and remote controls.

These devices operate within tight constraints on power, processing speed, and memory, which is why microcontrollers are designed for efficiency and low power consumption. Programming microcontrollers typically involves using languages like C or C++ and is often done through specialized development environments.

Additionally, they can be programmed to respond to external events or operate autonomously based on pre-defined logic. Popular microcontroller families include AVR, PIC, ARM Cortex, and ESP32, each offering different features and capabilities tailored to specific application needs.

The versatility and integration of microcontrollers make them essential components in the modern world, where they enable the development of smart, connected, and automated devices across various industries. Their role in embedded systems is crucial, providing the intelligence that drives everything from simple gadgets to complex industrial machines.



Microprocessor:

A microprocessor, unlike a microcontroller, focuses primarily on processing tasks and is often used in more complex embedded systems where processing power is a priority. It lacks integrated peripherals like memory or I/O, requiring external components.

Memory:

RAM (Random Access Memory): RAM in an embedded system is used for temporary storage of data and program execution. It's crucial for the system's real-time operation, allowing quick read/write access to data during processing.

ROM (Read-Only Memory): ROM stores the firmware or permanent software that controls the embedded system's operations. This non-volatile memory ensures the system retains its instructions even when powered off.

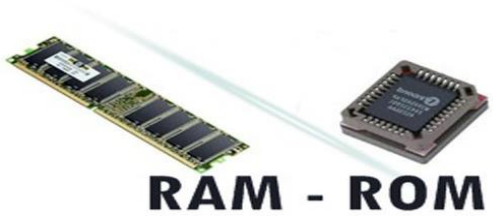
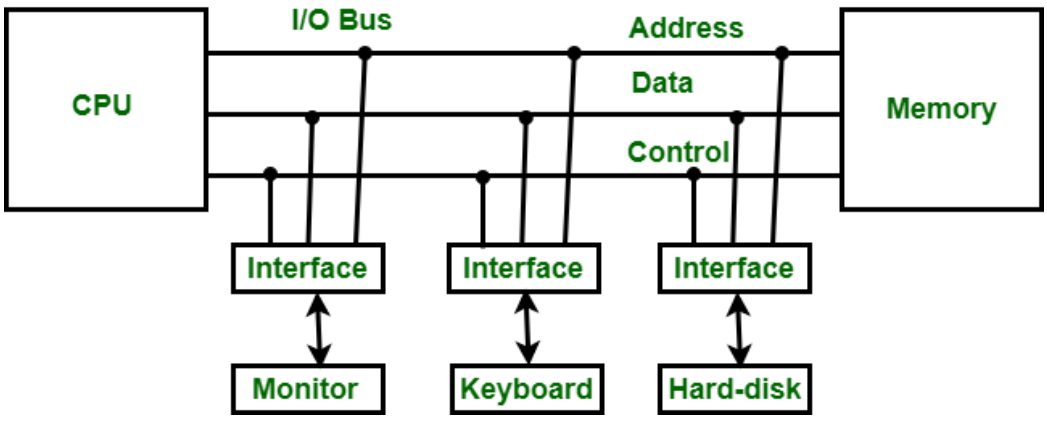


FIG 1.4

Input/Output Interfaces:

Digital/Analog I/O: These interfaces allow the embedded system to interact with the external environment. Digital I/O handles binary data, while analog I/O deals with continuous signals, such as voltage or current, which are converted into digital signals for processing.

Communication Interfaces: These include protocols like UART, SPI, I2C, and CAN bus, enabling communication between the embedded system and other devices or networks. These interfaces are essential for data exchange in applications like automotive systems, sensor networks, and industrial automation.



Timers and Counters:

Timers and counters are crucial for managing time-dependent tasks in embedded systems. They are used for generating precise time delays, counting events, or triggering specific actions at predetermined intervals. These components are vital for real-time operations, such as controlling motors, managing tasks in an operating system, or measuring sensor data.

Power Supply:

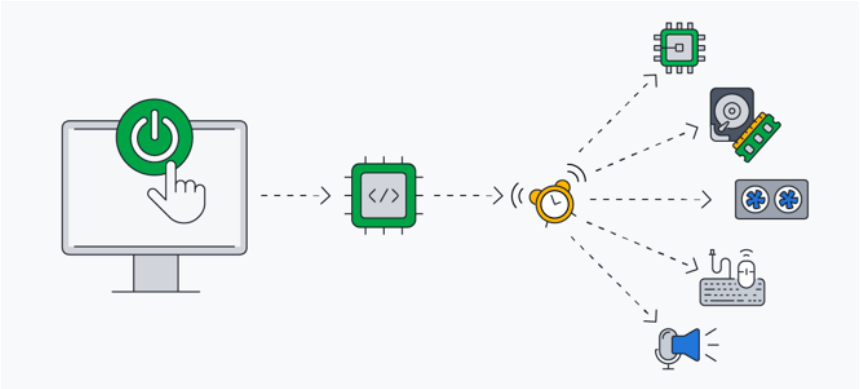
Embedded systems require a stable power supply to function correctly. The power supply unit (PSU) converts and regulates the power from a source (e.g., battery, AC adapter) to the appropriate voltage and current levels required by the system. In portable or low-power applications, power management is critical to extending battery life and ensuring the system operates efficiently.

Clocks and Oscillators:

Clocks and oscillators provide the timing signals necessary for synchronizing the operations of the microcontroller or microprocessor. They ensure that the embedded system's components work together in harmony, with precise timing critical for tasks like data processing, communication, and real-time control.

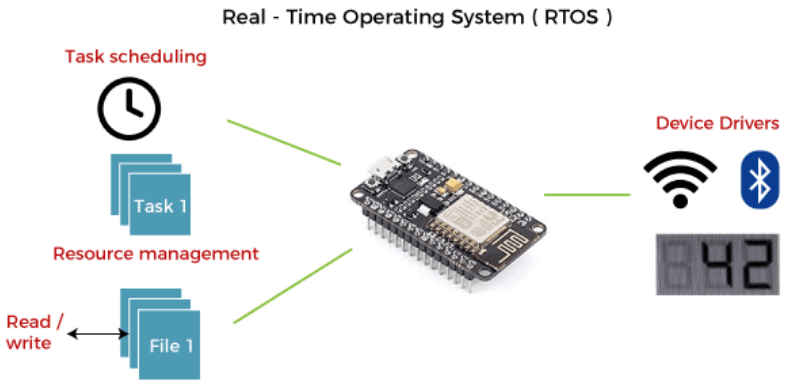
Firmware:

Firmware is the software embedded within the system, stored in ROM or flash memory. It controls the hardware components, enabling the system to perform its specific tasks. Firmware is usually written in low-level programming languages like C or assembly, allowing direct control over hardware resources.



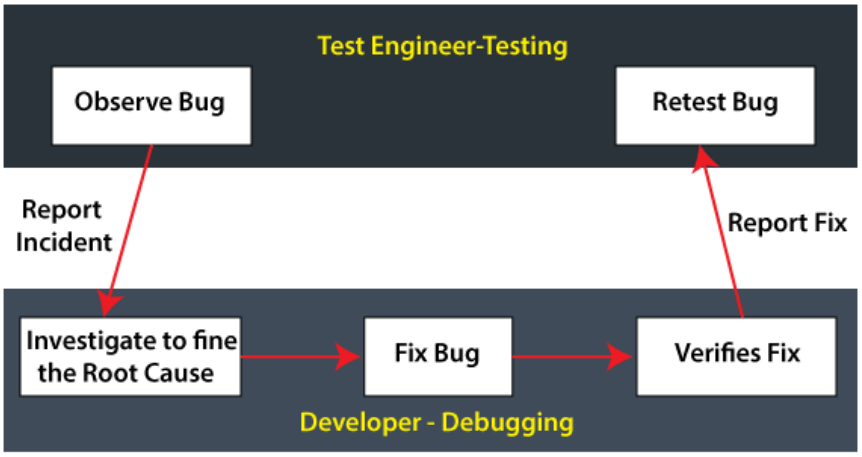
Real-Time Operating System (RTOS):

An RTOS manages the system's resources and ensures that tasks are executed within their time constraints, crucial for real-time embedded systems. It provides multitasking, task scheduling, and inter-task communication, making it easier to manage complex, time-sensitive operations.



Debugging and Testing Interfaces:

Debugging and testing interfaces, such as JTAG or SWD, allow developers to test and debug the embedded system during development. These interfaces enable step-by-step execution, monitoring of variables, and analysis of system performance, which are critical for ensuring the system functions correctly under all conditions.



1.2 Unit testing simulation:

Unit level testing in embedded systems involves verifying the functionality of individual components or modules before integrating them into the overall system. This testing process is crucial for identifying and resolving issues early in the development cycle, ensuring that each unit performs its intended function correctly.

In embedded systems, unit testing typically focuses on individual functions, methods, or code modules within the firmware. Given the resource constraints and real-time requirements of embedded systems, unit testing is often performed using both simulation environments and hardware-in-the-loop (HIL) setups.

Simulation-based unit testing allows developers to test the code in a controlled, virtual environment. This approach enables thorough testing without the need for physical hardware, which can be time-consuming and expensive. Simulators can replicate the behavior of microcontrollers, peripherals, and other hardware components, providing a

platform to validate code logic, error handling, and boundary conditions. Simulations also allow for the injection of faults and extreme conditions that might be challenging to reproduce in a physical setup.

The test cases in simulation environments are typically written in languages like C or C++ and run on the simulated hardware. These tests validate the functionality of individual modules, ensuring they meet specified requirements. For instance, a test case might verify that a temperature sensor module correctly converts analog signals to digital values within an expected range. By running these tests in a simulated environment, developers can quickly identify issues like incorrect logic, memory leaks, or timing errors.

Unit testing in embedded systems also involves using test harnesses, which are frameworks that simulate the environment in which the unit operates. These harnesses provide the necessary inputs, manage the execution of the test cases, and verify the outputs against expected results. This approach allows for automated testing, which is essential for efficiently handling the large number of test cases required to thoroughly verify embedded software.

The outcomes of unit testing are usually documented in detailed reports that include test case descriptions, expected and actual results, and any deviations or failures encountered. These reports help developers track the quality of the code and make informed decisions about further development or debugging.

Overall, unit level testing in embedded systems plays a critical role in ensuring the reliability and functionality of the system. By identifying and resolving issues at the unit level, developers can reduce the risk of defects propagating to higher levels of integration, ultimately leading to a more robust and reliable embedded system.

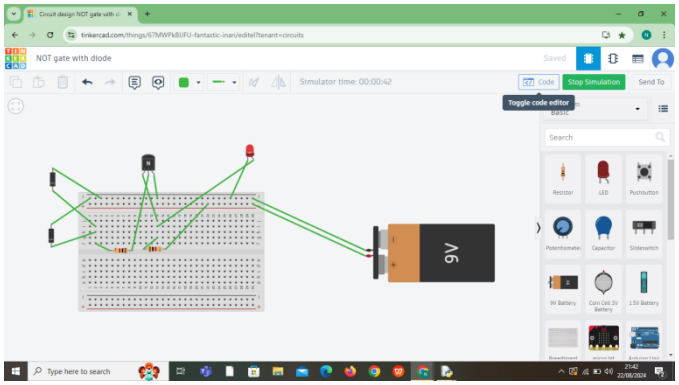
1.3 Programming exercise in simulation:

Tinkercad simulation provides an intuitive platform for designing and testing electronic circuits virtually. It allows users to create circuits using a drag-and-drop interface, connecting components like resistors, LEDs, microcontrollers, and sensors. Users can write code to program microcontrollers such as the Arduino directly within the platform and observe how the circuit behaves in real-time. Tinkercad also offers features to simulate

various input conditions, enabling users to test and debug their designs before implementing them in physical hardware. This makes it a valuable tool for learning, prototyping, and experimenting with embedded systems and electronics.

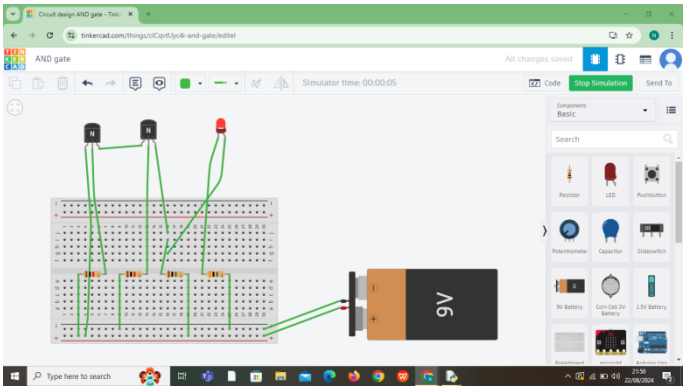
NOT GATE:

We can simulate a NOT gate by connecting a transistor in series with a resistor. The input signal is applied to the transistor's base through a current-limiting resistor, and the output is taken from the collector. When the input is high, the transistor switches on, pulling the output low, and when the input is low, the transistor remains off, allowing the output to be high.



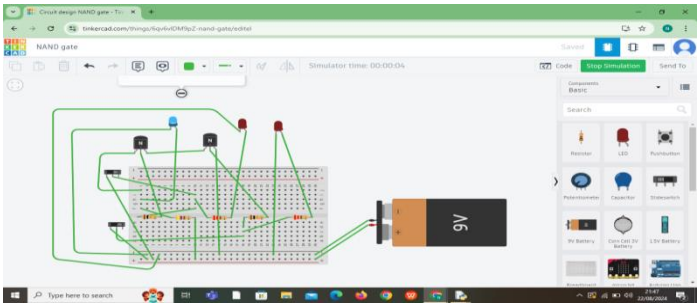
AND GATE:

We can simulate an AND gate using two transistors. Connect the emitters of both transistors to ground, with their collectors connected to a pull-up resistor leading to the output. Apply the input signals to the bases of the transistors; the output will be high only when both inputs are high, turning on both transistors and pulling the output low.



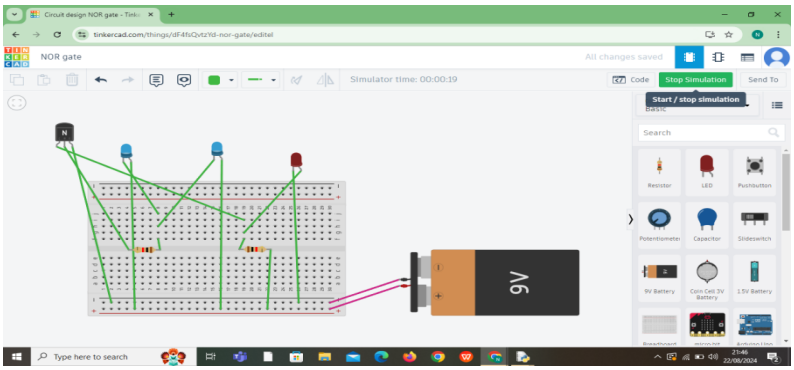
NAND GATE:

To simulate a NAND gate in Tinkercad using transistors and resistors, connect two transistors in series, with their emitters connected to ground and the output taken from the junction between the transistors and a pull-up resistor. Apply the input signals to the bases of each transistor. The output will be high unless both inputs are high, which turns on both transistors and pulls the output low, thereby inverting the AND gate operation to create a NAND gate.



NOR GATE:

In Tinkercad, to simulate a NOR gate using transistors and resistors, connect two transistors in parallel, with their collectors connected to the output and a pull-up resistor, and their emitters connected to ground. Apply the input signals to the bases of the transistors. The output will be high only when both inputs are low, as either input going high will turn on its respective transistor, pulling the output low, thus creating the NOR operation.



Basic C Programming exercise:

1. Even or Odd

This C program checks if an integer is even or odd. The program starts by prompting the user to input a number. It then calculates the remainder when dividing the number by 2 using the modulus operator (%). If the remainder is 0, the program prints that the number is even; otherwise, it prints that the number is odd. The program ends by returning 0 to indicate successful execution.

PROGRAM:

```
#include <stdio.h>
int main() {
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);
   if (number % 2 == 0)
        printf("%d is even.\n", number);
    else
        printf("%d is odd.\n", number);
 return 0;
}
```

2. Age Problem(Voting):

This C program determines if a person is eligible to vote based on their age. The user is prompted to enter their age, which is then checked against the voting age threshold of 18. If the entered age is 18 or older, the program outputs that the user is eligible to vote. If the age is below 18, it indicates that the user is not eligible. The program concludes by returning 0, signaling successful execution.

PROGRAM:

```
#include <stdio.h>
int main() {
   int age;
   printf("Enter your age: ");
   scanf("%d", &age);
   if (age >= 18)
      printf("You are eligible to vote.\n");
   else
      printf("You are not eligible to vote.\n");
return 0;
}
```

3. Swap two numbers:

This C program swaps the values of two integers entered by the user. After prompting the user to input two numbers, the program uses a temporary variable temp to hold the value of the first number while the second number is assigned to the first. Then, the value stored in temp is assigned to the second number. Finally, the program prints the swapped values of the two numbers and returns 0 to indicate successful execution.

PROGRAM:

```
#include <stdio.h>
int main() {
   int a, b, temp;
   printf("Enter two numbers: ");
   scanf("%d %d", &a, &b);
```

```
    temp = a;
    a = b;
    b = temp;
    printf("After swapping: a = %d, b = %d\n", a, b);
    return 0;
}
```

4. Factorial:

This C program calculates the factorial of a given number using a loop to multiply each integer up to the input number.

PROGRAM:

```
#include <stdio.h>
int main() {
    int n, i;
    unsigned long long fact = 1;
    printf("Enter a number: ");
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
        fact *= i;
    printf("Factorial of %d is %llu.\n", n, fact);
    return 0;
}
```

5.Fibonacci Series:

This C Program calculates the Fibonacci series up to a specified number of terms

PROGRAM:

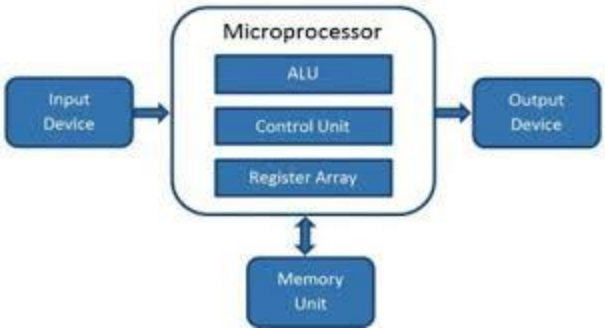
```
#include <stdio.h>

void fibonacci(int n) {
    int a = 0, b = 1, next;
    printf("Fibonacci series up to %d terms:\n", n);
    for (int i = 0; i < n; i++) {
        if (i <= 1) {
            next = i;
        } else {
            next = a + b;
            a = b;
            b = next;
        }
        printf("%d ", next);
    }
    printf("\n");
}
int main() {
    int num_terms;
    printf("Enter the number of terms: ");
    scanf("%d", &num_terms);
    if (num_terms <= 0) {
        printf("Please enter a positive integer.\n");
    } else {
        fibonacci(num_terms);
```

}return 0;}

CHIP LEVEL TESTING

2.1 Microprocessor and Microcontroller:

A **microprocessor** is a central processing unit (CPU) that is fabricated on a single integrated circuit (IC), containing millions of transistors and other components. It serves as the brain of a computer system, executing a wide range of tasks by performing arithmetic, logic, control, and input/output (I/O) operations. The microprocessor's primary function is to fetch, decode, and execute instructions from memory, enabling it to process data and control peripherals. This process involves interpreting program instructions, performing calculations, making decisions, and controlling the flow of information between the computer's memory, storage, and connected devices.



1. Arithmetic Logic Unit (ALU)

- Function: The ALU is responsible for performing all arithmetic and logical operations within the microprocessor. It handles basic operations like addition, subtraction, multiplication, and division, as well as logical operations such as AND, OR, NOT, and XOR. The ALU is the core of data processing in the microprocessor.
- Components: It consists of circuits that can perform arithmetic operations (like adders and subtractors) and logic operations (like comparators).

2. Control Unit (CU)

- Function: The control unit directs the operation of the processor by fetching instructions from memory, decoding them, and then executing them by sending control signals to other components of the microprocessor. It acts as the brain's "traffic controller," ensuring that data and instructions are correctly processed.
- Components: The control unit consists of a decoder, a sequence counter, and control logic circuits. It generates control signals to manage the execution of instructions and operations within the microprocessor.

3. Register Array

- Function: Registers are small, fast storage locations within the microprocessor used to hold data, instructions, and addresses temporarily. The register array is a collection of these registers. Registers facilitate quick access to data and instructions that the ALU needs to process.
- Types: There are various types of registers, such as the accumulator, data registers, address registers, and special-purpose registers (like the program counter and status register).

4. Memory Unit

- Function: The memory unit stores data and instructions required for processing. It consists of both primary memory (RAM and ROM) and secondary storage (hard drives, SSDs). The microprocessor fetches instructions and data from memory, processes them, and stores the results back in memory.
- Types:
    - RAM (Random Access Memory): Used for temporary storage and is volatile, meaning data is lost when power is turned off.
    - ROM (Read-Only Memory): Used for permanent storage of instructions that are not meant to be altered, such as the firmware.

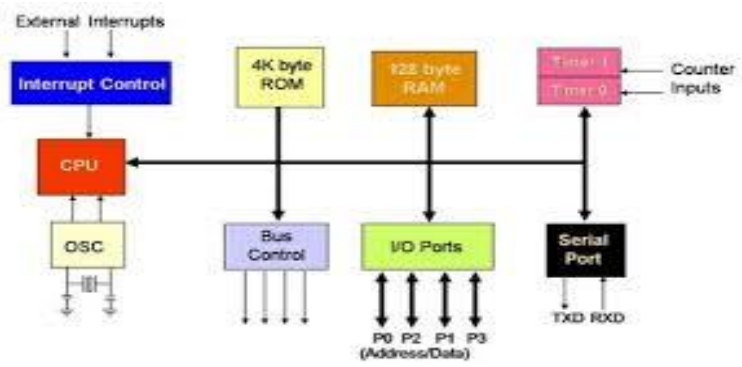
5. Input and Output Devices (I/O)

- Function: I/O devices allow the microprocessor to communicate with the external world. Input devices (like keyboards, mice, and sensors) send data to the

microprocessor, while output devices (like displays, printers, and LEDs) receive data from the microprocessor.

- Interaction: The microprocessor uses ports and interfaces (like USB, serial, and parallel ports) to connect with these devices, enabling the execution of tasks that require external interaction.

A **microcontroller** is an integrated circuit designed to perform specific control tasks within embedded systems. Unlike a microprocessor, which is primarily focused on processing data, a microcontroller is a complete system on a chip (SoC) that includes a CPU, memory (RAM and ROM), I/O peripherals, and other essential components. It is used in various applications, such as home appliances, automotive systems, medical devices, and industrial automation, where it controls processes and interacts with external devices. The microcontroller is engineered to be highly efficient and cost-effective, allowing it to perform tasks like monitoring sensors, driving motors, managing user interfaces, and more, all within a compact footprint.The key difference lies in their application: microprocessors are suited for complex, high-performance tasks requiring extensive computing power, while microcontrollers are optimized for specific, real-time control tasks within embedded systems.



1. Central Processing Unit (CPU)

- **Function**: The CPU is the brain of the microcontroller, responsible for executing instructions from the program stored in memory. It performs arithmetic and logic operations, controls the timing and sequencing of processes, and manages data flow within the microcontroller.
- **Operations**: The CPU fetches instructions from ROM, decodes them, and executes them to perform tasks like reading sensor data, controlling actuators, or communicating with other devices.

2. Random Access Memory (RAM)

- **Function**: RAM is a volatile memory used to store data temporarily while the microcontroller is running. It holds variables, buffers, and intermediate data that the CPU needs during program execution.
- **Usage**: RAM is used for operations like storing sensor readings, maintaining counters, or holding the stack for function calls.

3. Read-Only Memory (ROM)

- **Function**: ROM is non-volatile memory that stores the firmware or program code that the microcontroller executes. It retains data even when the power is off.
- **Types**: Common types of ROM in microcontrollers include Flash memory, EPROM, and EEPROM, which can be used to store the bootloader, application code, and sometimes calibration data.

4. Interrupt System

- **Function**: Interrupts are signals that temporarily halt the CPU's current operations to attend to a high-priority task. The interrupt system enables the microcontroller to respond immediately to external events (e.g., a button press or sensor trigger) without waiting for the current program loop to reach that point.
- **Components**: The interrupt system includes interrupt vectors, interrupt service routines (ISRs), and priority levels to manage multiple interrupts.

5. Oscillator

- **Function**: The oscillator provides the clock signal that drives the timing of the microcontroller. It determines the speed at which the CPU executes instructions.
- **Types**: Microcontrollers may use internal oscillators or external crystals, with frequencies typically ranging from a few kilohertz to several megahertz, depending on the application's timing requirements.

6. Timers and Counters

- **Function**: Timers and counters are used to measure time intervals, generate precise delays, or count events. They are essential for tasks like generating PWM signals, timing operations, or implementing watchdog timers.
- **Operation**: Timers can operate in various modes, such as up-counting, down-counting, or capturing time events, and can trigger interrupts when specific conditions are met.

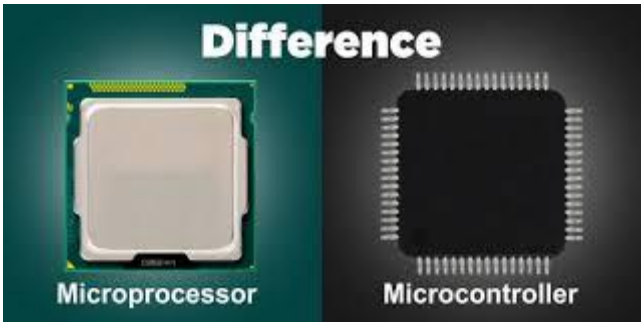
7. Bus System

- **Function**: The bus system is the communication pathway that connects the CPU, memory, and peripherals within the microcontroller. It allows the transfer of data, instructions, and control signals between different components.
- **Types**: Common buses in a microcontroller include the data bus, address bus, and control bus, each serving a specific role in the operation of the microcontroller.

Difference between microcontroller and microprocessor:

Microcontrollers and processors serve distinct roles in computing systems, each tailored for specific applications. A microcontroller is a compact, integrated circuit that combines a CPU, memory (both RAM and ROM), and various peripherals (such as timers and ADCs) into a single chip. This integration makes microcontrollers ideal for embedded systems where dedicated, low-power solutions are needed, such as in household appliances and automotive controls. They are generally low-cost and designed for efficiency in real-time tasks.

In contrast, a processor (or CPU) is focused solely on computation and requires external components like RAM, ROM, and I/O interfaces to function. Processors are used in general-purpose computing devices like desktops and servers, offering high performance and flexibility but typically consuming more power and costing more due to the need for additional components. While microcontrollers are optimized for specific, embedded applications with lower complexity and power consumption, processors are built for handling complex, multitasking environments with higher computational demands.



2.2 Active and Passive components:

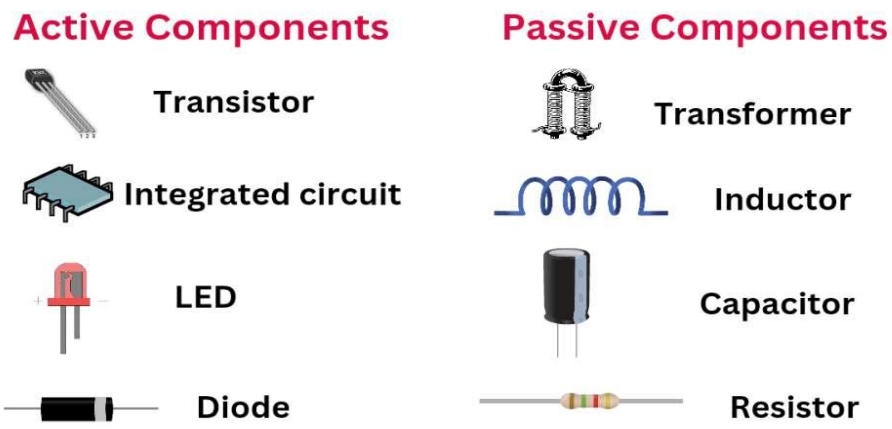
In embedded testing, both active and passive components play crucial roles in ensuring the functionality and reliability of electronic systems.

Active components are electronic devices that can control the flow of electricity and amplify signals within a circuit. These components require an external power source to operate. Common examples include transistors, diodes, and integrated circuits (ICs). In embedded systems, active components are essential for processing signals, switching, and providing power management. During testing, these components are examined for their ability to perform tasks such as amplification, switching, and signal modulation. For instance, in a microcontroller circuit, the transistors might be tested for proper switching behavior to ensure that digital signals are accurately processed.

Passive components, on the other hand, do not require an external power source to operate and cannot amplify signals. They include resistors, capacitors, inductors, and diodes (when used for functions like rectification). These components are used to control current flow, filter signals, store energy, and manage voltage levels within a circuit. In embedded testing, passive components are tested for their ability to perform these tasks under various conditions. For example, resistors are tested to ensure they provide the correct resistance value, which is crucial for controlling current and protecting sensitive components in the circuit.

Testing embedded systems involves verifying the functionality of both active and passive components, ensuring they operate correctly within their specified ranges and under different environmental conditions. This helps identify potential issues such as signal distortion, power inefficiency, or failure under stress, which could lead to system

malfunctions. Proper testing of these components is vital for the overall reliability and performance of the embedded system.



2.3 Arduino IDE:

The Arduino Integrated Development Environment (IDE) is a user-friendly software platform designed for writing, compiling, and uploading code to Arduino microcontroller boards. It is widely used in the maker community, education, and by professionals for developing embedded systems and Internet of Things (IoT) applications.

The Arduino IDE is open-source and cross-platform, meaning it can be run on Windows, macOS, and Linux. It provides a simple and intuitive interface that is accessible to both beginners and experienced developers. The primary programming language used in the Arduino IDE is based on C and C++, but the IDE abstracts much of the complexity, making it easier for users to focus on developing their projects.

Key features of the Arduino IDE include:

Code Editor: The IDE features a basic code editor with syntax highlighting, automatic indentation, and simple text search functionality. This editor allows users to write and edit sketches, which are the programs that run on Arduino boards.

Sketchbook: Arduino sketches are stored in a folder called the Sketchbook, which acts as a project manager for organizing and accessing your projects.

Libraries: The Arduino IDE supports a wide range of libraries, which are collections of code that provide additional functionality, such as controlling sensors, motors, displays, and communication modules. Users can easily include these libraries in their sketches to expand the capabilities of their projects.

Serial Monitor: The IDE includes a built-in Serial Monitor that allows users to communicate with the Arduino board via serial communication. This feature is particularly useful for debugging, as it enables the display of real-time data from the board, such as sensor readings or diagnostic messages.

Compiler and Uploader: The Arduino IDE includes a built-in compiler that converts the written code into a format that the Arduino board can understand. After compilation, the IDE automatically uploads the compiled code to the connected Arduino board via USB or other communication interfaces.

Board and Port Selection: Users can easily select the type of Arduino board they are working with from a drop-down menu, ensuring compatibility between the code and the hardware. The IDE also allows users to select the appropriate communication port to connect to their Arduino board.

Extensions and Plugins: The Arduino IDE supports additional plugins and extensions, which can enhance its functionality. This includes support for different types of boards beyond the standard Arduino models, enabling a broader range of hardware compatibility.

ARDUINO HARDWARE :

Arduino is an open-source electronics platform based on easy-to-use hardware and software, designed for creating interactive projects. The core of the Arduino system revolves around various hardware components that enable users to build and control devices in a wide range of applications. Below are the key hardware components commonly used in Arduino projects:

1.Arduino Boards

- **Microcontroller:** The heart of any Arduino board is the microcontroller, which serves as the brain of the system. Popular boards include the Arduino Uno (based on the ATmega328P microcontroller), Arduino Nano, and Arduino Mega.
- **Digital and Analog Pins:** Arduino boards come with a variety of digital and analog I/O pins used for interfacing with external components such as sensors, LEDs, and motors.
- **Power Supply:** Arduino boards can be powered via USB connection, battery, or external power adapters, providing flexibility in various project environments.

2.Sensors

- **Temperature and Humidity Sensors:** Components like the DHT11 and DHT22 are used to measure environmental conditions.
- **Motion Sensors:** PIR (Passive Infrared) sensors detect movement, while accelerometers and gyroscopes measure orientation and acceleration.
- **Light Sensors:** Photoresistors and photodiodes measure light intensity, enabling light-sensitive applications.

3.Actuators

- **Motors:** DC motors, servos, and stepper motors are used to add movement to projects. The Arduino can control motor speed and direction using motor drivers or shields.
- **Relays:** These act as electrically operated switches, allowing Arduino to control high-power devices like lamps and appliances.
- **LEDs and Displays:** LEDs provide visual feedback, while LCD displays and OLED screens can show information and status updates directly on the device.

4. Communication Modules

- **Wi-Fi Modules:** Components like the ESP8266 or ESP32 allow Arduino to connect to wireless networks, enabling IoT applications.

- **Bluetooth Modules:** Modules like HC-05 or HC-06 enable wireless communication between the Arduino and smartphones or other devices.
- **RFID and NFC Modules:** These modules are used for proximity-based applications, such as access control systems and contactless payment.

5.Power Components

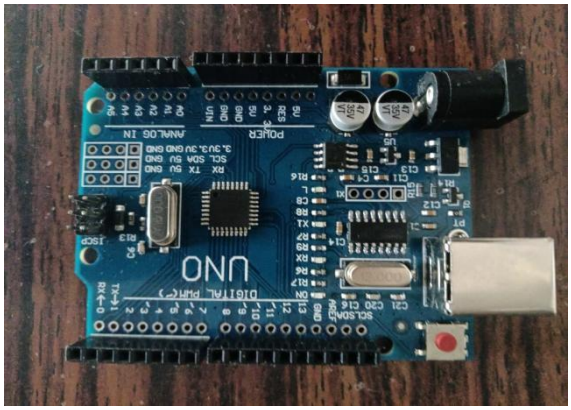
- **Voltage Regulators:** Ensure that the Arduino and its components receive a stable voltage, protecting them from power fluctuations.
- **Batteries:** Portable power sources, including AA batteries, lithium-ion batteries, and coin cells, allow for untethered projects.

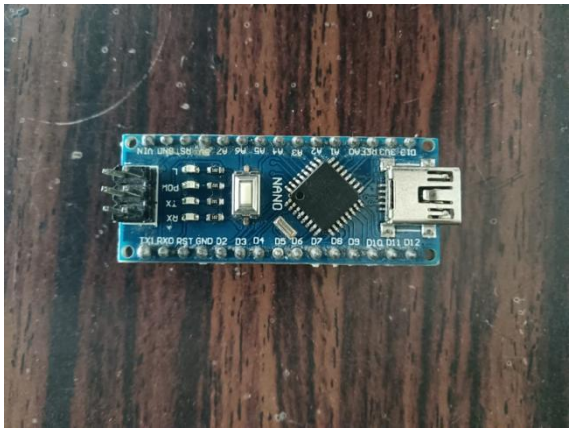
6.Shields

- **Motor Shields:** These are used to easily interface with motors, providing a plug-and-play solution for motor control.
- **Relay Shields:** Enable easy control of multiple high-voltage devices through Arduino.
- **Sensor Shields:** Offer easy connection of multiple sensors to the Arduino, streamlining complex projects.

7. Prototyping Accessories

- **Breadboards:** Essential for creating temporary circuits without soldering, allowing easy experimentation.
- **Jumpers and Connectors:** Used to connect different components on the breadboard or to the Arduino.
- **PCB Boards:** Used for creating more permanent circuits once a prototype is finalized.





BOARD LEVEL TESTING

3.1 Sensors and Actuators:

Sensors are devices that detect and respond to physical changes in the environment, converting them into electrical signals for measurement or control. They are essential components in modern technology, enabling systems to monitor temperature, pressure, humidity, light, motion, and other parameters. Sensors are used in a wide range of applications, from everyday devices like smartphones and thermostats to industrial automation, healthcare, and automotive systems. They can be analog or digital, depending on the type of output they produce. Sensors play a critical role in the Internet of Things (IoT), where they collect data for analysis and decision-making. Their accuracy, sensitivity, and reliability are key factors in ensuring the effectiveness of the systems they are part of.

Ultrasonic Sensor:

An ultrasonic sensor works by emitting ultrasonic sound waves (typically above 20 kHz) and measuring the time it takes for the sound to reflect back from an object. The sensor calculates the distance based on the speed of sound in the air. It's widely used in applications like parking sensors, robotics for obstacle detection, and level measurement in tanks and silos.

DHT Sensor:

The DHT sensor (Digital Humidity and Temperature) is a low-cost, easy-to-use sensor that provides digital output for both temperature and humidity readings. It consists of a thermistor for temperature measurement and a capacitive humidity sensor. It's popular in DIY electronics, weather stations, and IoT projects where monitoring environmental conditions is essential.

Gas Sensor:

A gas sensor, or gas detector, is designed to detect and quantify the presence of specific gases in the environment. It works by converting the concentration of a gas into a corresponding electrical signal. Gas sensors are crucial in industrial safety systems, environmental monitoring, and residential gas leak detectors to prevent dangerous situations by triggering alarms or ventilation systems.

PIR Sensor:

The Passive Infrared (PIR) sensor detects motion by measuring changes in infrared radiation emitted by objects in its detection area. When a warm object, like a human body, passes in front of the sensor, it triggers a signal. PIR sensors are commonly used in security alarms, automatic lighting systems, and energy-saving applications where detecting human presence is important.

BMP Sensor:

The BMP (Barometric Pressure) sensor measures atmospheric pressure with high precision. It can also calculate altitude and temperature based on the pressure readings. This

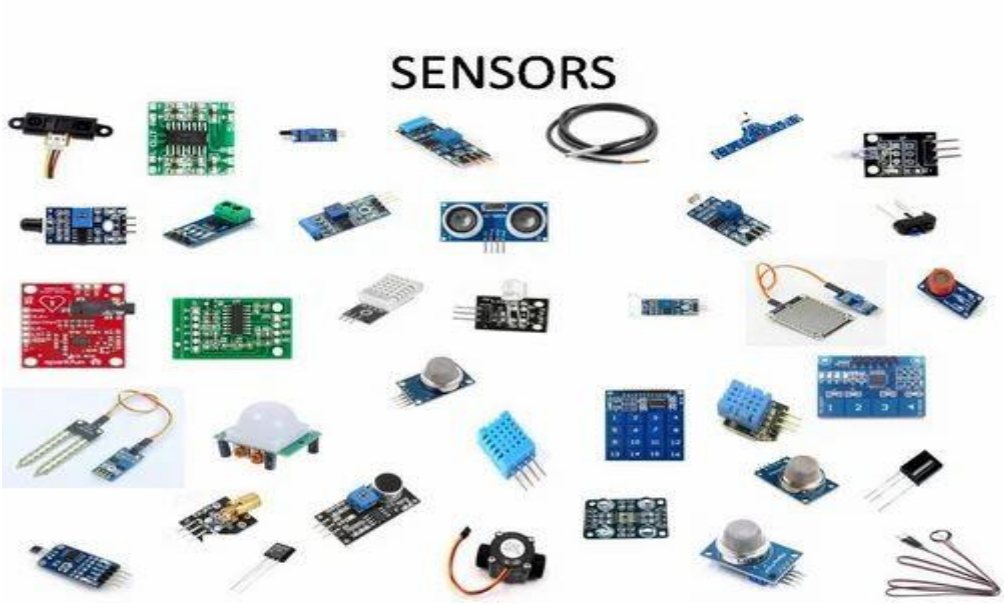
sensor is commonly found in weather stations, smartphones for altitude measurement, and drones for altitude stabilization, providing essential data for various environmental and navigation applications.

Soil Moisture Sensor:

A soil moisture sensor measures the volumetric water content in soil, providing critical information for managing irrigation. It usually consists of probes that are inserted into the soil to measure the resistance or capacitance, which varies with moisture levels. This sensor is indispensable in agriculture, gardening, and smart irrigation systems to ensure plants receive the right amount of water.

IR Sensor:

An Infrared (IR) sensor detects infrared radiation from objects, often used for proximity sensing, motion detection, and remote control applications. It can emit and detect IR light, making it versatile in applications like automatic doors, obstacle detection in robotics, and touchless control interfaces in consumer electronics.



Actuators, on the other hand, are devices that convert electrical signals into physical actions, effectively bridging the digital and physical worlds. Actuators receive commands from the embedded system and execute them by performing tasks such as moving a motor, opening a valve, or adjusting a display. For instance, a DC motor controlled by an embedded system might rotate in response to a signal, driving mechanical movement in a robot or conveyor belt.

In many applications, sensors and actuators work together in a feedback loop. The sensor collects data from the environment, which is processed by the embedded system to determine an appropriate response, and the actuator then carries out that response. For example, in an automated temperature control system, a temperature sensor detects the room's temperature, and if it falls outside the desired range, the system activates a heater or cooling fan via an actuator to adjust the temperature.

3.2 SENSOR TESTING:

LED BLINK USING ARDUINO UNO:

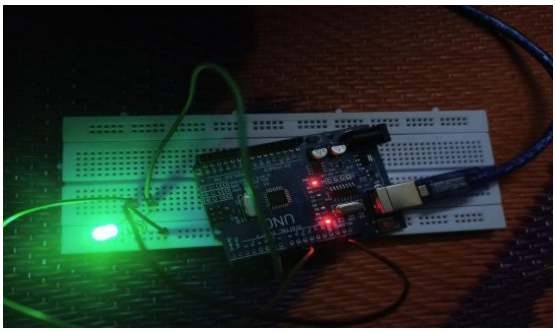
To blink an LED using an Arduino Uno, connect the LED's anode to digital pin 13 and its cathode to ground through a resistor. In the Arduino IDE, write a sketch to set pin 13 as an output, then toggle the pin HIGH to turn the LED on and LOW to turn it off, with a delay of 1000 milliseconds between each state change. Upload the sketch to the Arduino Uno, and the LED will blink on and off every second.

CODE:

```
int led1_pin = 9;
int led2_pin = 3;
int led3_pin = 5;
int brightness1 = 0, brightness2 = 100, brightness3 = 255;
void setup() {
    Serial.begin(9600);
```

```
  pinMode(led1_pin, OUTPUT);
  pinMode(led2_pin, OUTPUT);
  pinMode(led3_pin, OUTPUT);
}
void loop() {
  analogWrite(led1_pin, brightness1);
  analogWrite(led2_pin, brightness2);
  analogWrite(led3_pin, brightness3);
  delay(20000);
  analogWrite(led1_pin, brightness2);
  analogWrite(led2_pin, brightness3);
  analogWrite(led3_pin, brightness1);
  delay(20000);
  analogWrite(led1_pin, brightness3);
  analogWrite(led2_pin, brightness1);
  analogWrite(led3_pin, brightness2);
  delay(20000);
}
```

OUTPUT:



DHT SENSOR USING ARDUINO NANO:

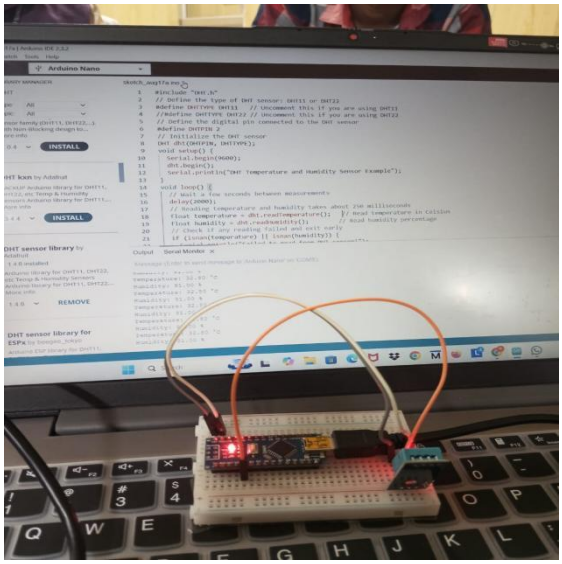
To use a DHT sensor with an Arduino Nano, connect the sensor's data pin to a digital pin on the Nano, and connect the VCC and GND pins to the Arduino's 5V and GND, respectively. In the Arduino IDE, include the DHT library and write a sketch to initialize the sensor and read temperature and humidity data. Upload the code to the Arduino Nano, and the sensor will provide readings that can be displayed on the Serial Monitor.

CODE:

```
#include "DHT.h"

#define DHTTYPE DHT11

#define DHTPIN 2

DHT dht(DHTPIN, DHTTYPE);

void setup() {

  Serial.begin(9600);

  dht.begin();

  Serial.println("DHT Temperature and Humidity Sensor Example");

}

void loop() {

  // Wait a few seconds between measurements

  delay(2000);

   float temperature = dht.readTemperature();

  float humidity = dht.readHumidity();            if (isnan(temperature) || isnan(humidity))
{

    Serial.println("Failed to read from DHT sensor!");

    return;

  }
```

```
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
}
```

OUTPUT:



DHT SENSOR USING ESP32:

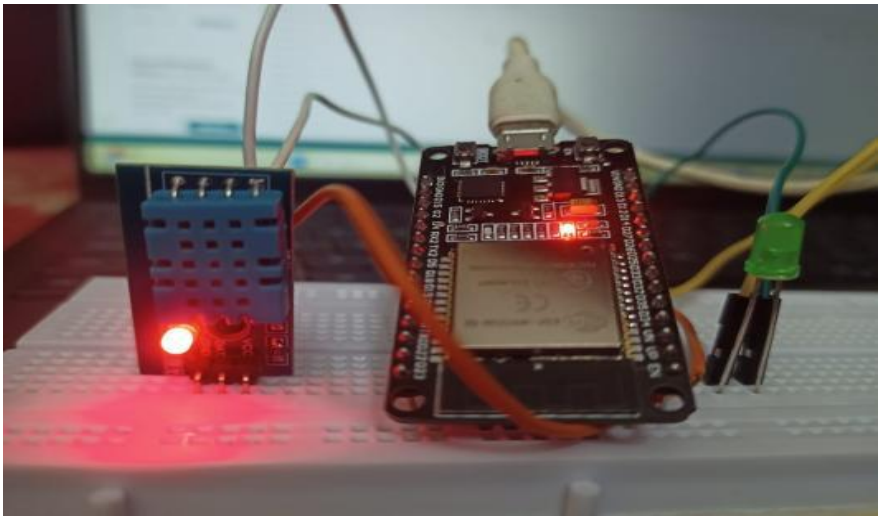
To use a DHT sensor with an ESP32, connect the sensor’s data pin to a digital GPIO pin on the ESP32, and connect the VCC and GND pins to the ESP32's 3.3V and GND, respectively. In the Arduino IDE, include the DHT library and write a sketch to initialize the sensor and read temperature and humidity data. Upload the code to the ESP32, and the readings will be available in the Serial Monitor or can be used in other applications.

CODE:

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#define DHTPIN 15  #define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
const int ledPin = 13;
void setup() {
  Serial.begin(115200);
  dht.begin();
  pinMode(ledPin, OUTPUT);
}
void loop() {
    digitalWrite(ledPin, HIGH); // Turn the LED on
  delay(100);                 // LED on for 100ms
  digitalWrite(ledPin, LOW);  // Turn the LED off
  float humidity = dht.readHumidity();
   float temperature = dht.readTemperature();
 if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
```

```
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" *C");
  delay(2000);
}
```

OUTPUT:



BMP 180 WITH ESP32:

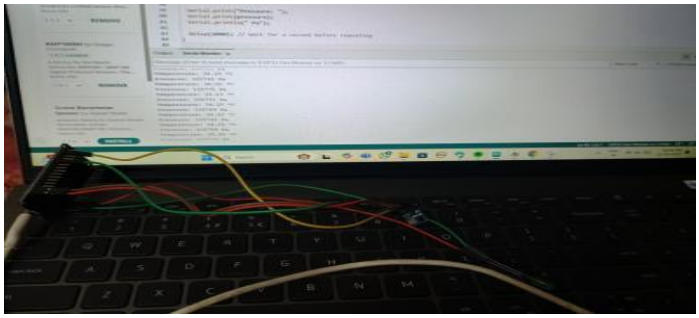
To use a BMP180 sensor with an ESP32, connect the sensor's SDA pin to a GPIO pin designated for I2C data, and the SCL pin to a GPIO pin designated for I2C clock, along with the VCC and GND pins to the ESP32's 3.3V and GND, respectively. In the Arduino IDE, include the BMP180 library and write a sketch to initialize the sensor and read temperature and pressure data. Upload the code to the ESP32, and the sensor readings will be available in the Serial Monitor or can be used in your application.

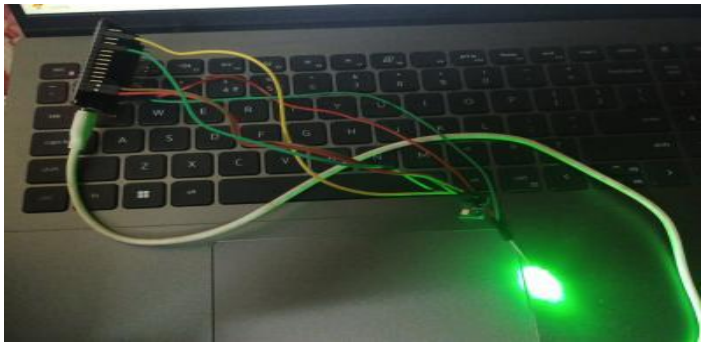
CODE:

```
#include <Wire.h>
#include <Adafruit_BMP085.h>
Adafruit_BMP085 bmp;
const int ledPin = 13;
```

```
void setup() {
  Serial.begin(115200);
  if (!bmp.begin()) {
    Serial.println("Could not find a valid BMP085 sensor, check wiring!");
    while (1) {}
  }
  pinMode(ledPin, OUTPUT);
}
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(100);
  digitalWrite(ledPin, LOW);
  float temperature = bmp.readTemperature();
  int32_t pressure = bmp.readPressure();
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" *C");
  Serial.print("Pressure: ");
  Serial.print(pressure);
  Serial.println(" Pa");
  delay(1000);
}
```

OUTPUT:





INSTALLATION OF RASPBERRY PI:

To install and set up a Raspberry Pi, follow these steps:

1. Download and Install Raspberry Pi Imager

- **Download Raspberry Pi Imager** from the official Raspberry Pi website.
- **Install** it on your computer.

2. Prepare the Storage Device

- **Insert** the USB drive or microSD card into your computer.

3. Launch Raspberry Pi Imager

- **Open** Raspberry Pi Imager.

4. Select the Operating System

- **Click** on “Choose OS.”

- **Select** “Raspberry Pi (64-bit)” for Raspberry Pi 4, or choose another version based on your Raspberry Pi model.

5. Select Storage

- **Click** on “Choose Storage.”
- **Select** your USB drive or microSD card from the list of available drives.

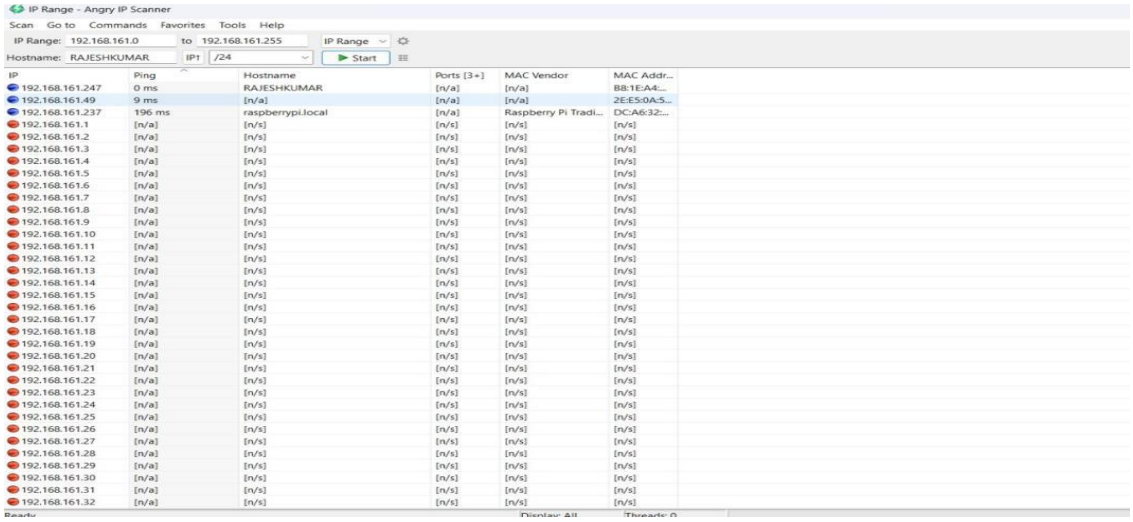
6. Configure Advanced Options (Optional)

- **Click** the gear icon or the advanced options button (if available).
- **Set up** your preferences:
  - **Hostname**: Enter a name for your Raspberry Pi.
  - **Username**: Set a username.
  - **Password**: Set a password.
  - **Wi-Fi**: Enter your Wi-Fi network name (SSID) and password if you’re using Wi-Fi.
  - **Enable SSH**: Check the box to enable SSH for remote access.

7. Write OS to Storage

- **Click** “Write” to start writing the OS to the storage device.
- **Wait** for the process to complete. This may take a few minutes.

- ++



11. Connect via RealVNC Viewer

- **Download** and **install** RealVNC Viewer from the official VNC website.
- **Open** RealVNC Viewer.
- **Click** “New Connection” and enter the Raspberry Pi’s IP address.
- **Enter** the username and password you set up earlier.
- **Connect** to your Raspberry Pi.

12. Configure Raspberry Pi via SSH

- **Open** a terminal on your computer.

Type :ssh rpi@your_ip

Replace your_ip with the Raspberry Pi’s IP address.

Enter the password you set up for SSH access.

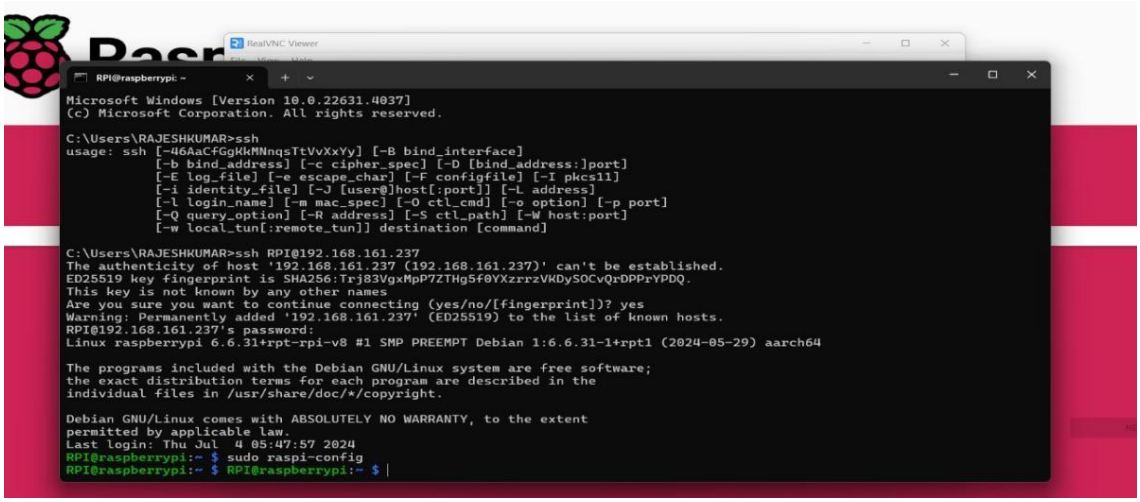
Configure the raspberry pi using the command :

sudo raspi-config

This will open the Raspberry Pi configuration tool where you can adjust settings like locale, time zone, and additional options.

13. Complete Configuration

- **Follow** the prompts in raspi-config to complete the setup.
- **Reboot** your Raspberry Pi if necessary.



SENSOR TESTING WITH RASPBERRY PI:

LED BLINK USING RASPBERRY PI:

To make an LED blink using a Raspberry Pi, connect the LED's positive leg to a GPIO pin (e.g., GPIO 18) and the negative leg to a ground pin via a resistor. Write a Python script that toggles the GPIO pin on and off with a delay in between using the RPi.GPIO library. The script continuously turns the LED on and off, creating a blinking effect. Finally, run the script on the Raspberry Pi to observe the LED blinking.

CODE:

import RPi.GPIO as GPIO

import time

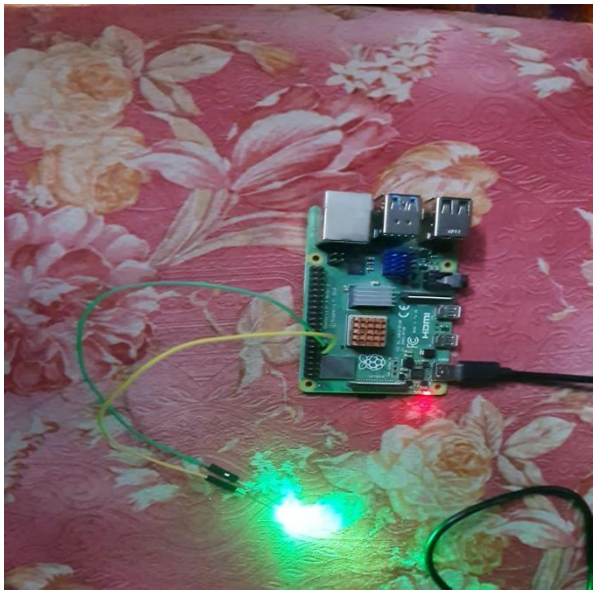
Set up the GPIO mode

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

```
# Set up GPIO 17 as an output
GPIO.setup(17, GPIO.OUT)
# Blink the LED
try:
    while True:
        GPIO.output(17, GPIO.HIGH)  # Turn LED on
        time.sleep(1)               # Wait for 1 second
        GPIO.output(17, GPIO.LOW)   # Turn LED off
        time.sleep(1)               # Wait for 1 second
except KeyboardInterrupt:
    GPIO.cleanup()  # Clean up GPIO on CTRL+C exit
finally:
    GPIO.cleanup()  # Clean up GPIO on normal exit
```

OUTPUT:



HEART RATE SENSOR WITH RASPBERRY PI:

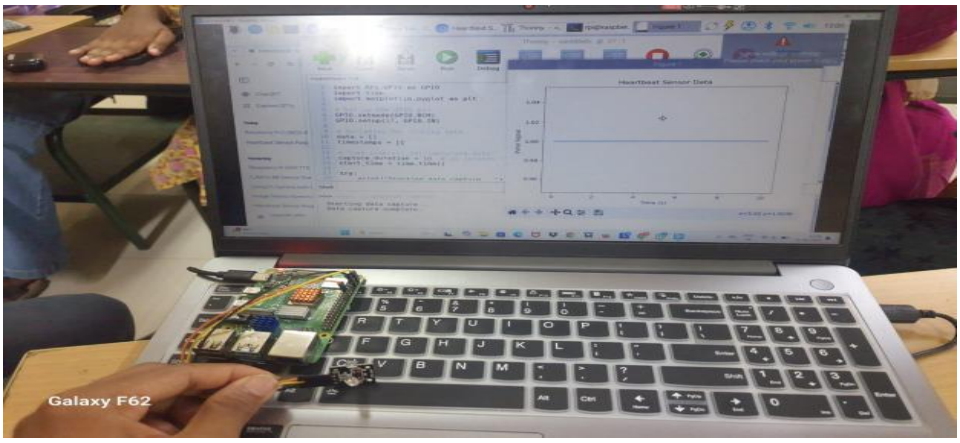
To use a heart rate sensor with a Raspberry Pi, connect the sensor's output pin to a GPIO pin on the Raspberry Pi, and connect the VCC and GND pins to the Raspberry Pi's 3.3V and GND, respectively. Install any necessary libraries and write a Python script to read the sensor data from the GPIO pin. Run the script to process and display the heart rate information on the Raspberry Pi.

CODE:

```
import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BCM)

PULSE_SENSOR_PIN = 17

GPIO.setup(PULSE_SENSOR_PIN, GPIO.IN)

pulse_count = 0

last_time = time.time()

def pulse_callback(channel):

    global pulse_count
```

```
    pulse_count += 1

GPIO.add_event_detect(PULSE_SENSOR_PIN,GPIO.RISING,
callback=pulse_callback)

try:

    while True:

        current_time = time.time()

        elapsed_time = current_time - last_time

        if elapsed_time >= 1:

            heart_rate = pulse_count * 60

            print(f"Heart Rate: {heart_rate} BPM")

            pulse_count = 0

            last_time = current_time

        time.sleep(0.1)

except KeyboardInterrupt: print("Program terminated")
```

OUTPUT:



CAMERA MODULE USING RASPBERRY PI:

The Raspberry Pi Camera Module is a high-quality camera that connects to the Raspberry Pi via the CSI (Camera Serial Interface) port. It is capable of capturing still

images and video, making it ideal for various projects like time-lapse photography, video recording, and even computer vision applications.

To use a camera module with a Raspberry Pi, connect the camera module to the dedicated CSI interface on the Raspberry Pi board. Enable the camera interface by running sudo raspi-config and selecting the camera option. Install any necessary libraries or tools, such as raspistill or raspivid, to capture images or video. Use command-line tools or Python scripts to interact with the camera and capture media.

TO ENABLE CAMERA MODULE:

· Connect the camera module to the CSI interface on the Raspberry Pi.

· Run ‘sudo raspi-config’ in the terminal.

· Navigate to **Interfacing Options** and select **Camera**.

· Choose **Enable** and confirm the changes.

· Reboot the Raspberry Pi with ‘sudo reboot’ to apply the settings.

STEPS FOR USING RASPBERRY PI CAMERA WITH LIBCAMERA COMMAND:

1.Update System Packages Open a terminal and run:

sudo apt update
sudo apt upgrade

2.Install libcamera and related utilities:

sudo apt install libcamera-apps

3. Ensure the camera interface is enabled:

sudo raspi-config

Navigate to Interface Options -> Camera and enable it. Reboot the Raspberry Pi if prompted

4. Check if the camera is recognized:

libcamera-hello

5. Use libcamera-vid to record a video. For example, to record a 10-second video:

libcamera-vid -t 10000 -o video.h264

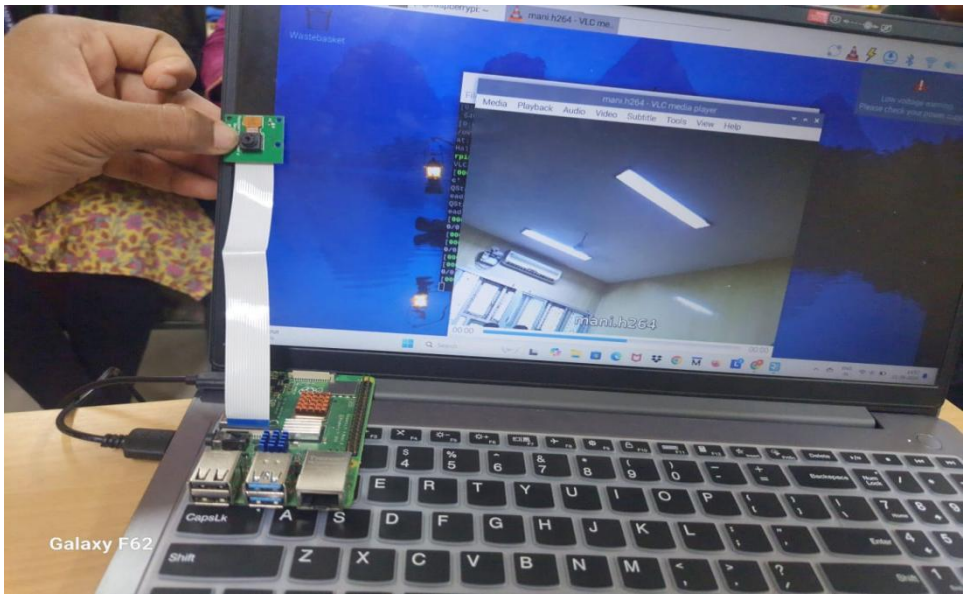
6. To view the recorded video using VLC:

vlc video.h264

7. To capture still images, use

libcamera-still -o image.jpg

OUTPUT:



3.3 Integration of sensors:

Integration of Ultrasonic with SD card using Arduino Nano:

To integrate an ultrasonic sensor with an SD card using an Arduino Nano, connect the ultrasonic sensor's trig and echo pins to digital I/O pins on the Arduino. Connect the SD card module to the Arduino using SPI pins (MOSI, MISO, SCK, and CS). Write a sketch to read distance measurements from the ultrasonic sensor and save them to a file on the SD card. Upload the code to the Arduino Nano to log distance data.

CODE:

```
#include <SD.h>

#include <SPI.h>
```

```
int CS_PIN = 10;
const int trigPin = 9;   // Trigger pin for the Ultrasonic Sensor
const int echoPin = 8;   // Echo pin for the Ultrasonic Sensor
File file;
void setup()
{
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  initializeSD();
 // Clear the file and write three ultrasonic sensor readings to it
  createFile("sush.txt");  // This clears the file before writing new data
  writeToFile(String(getUltrasonicDistance()) + " cm");
  writeToFile(String(getUltrasonicDistance()) + " cm");
  writeToFile(String(getUltrasonicDistance()) + " cm");
  closeFile();
  // Read the stored sensor values from the file
  openFile("sush.txt");
  Serial.println("Reading back the sensor values:");
  Serial.println(readLine());
  Serial.println(readLine());
  Serial.println(readLine());
  closeFile();
}
void loop()
```

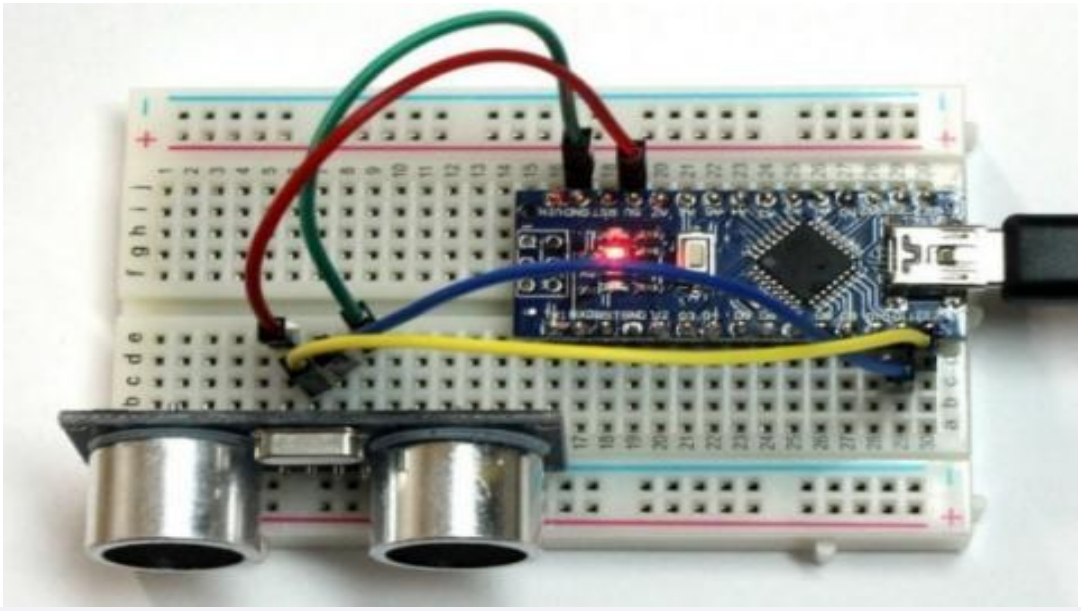
```
{
}
void initializeSD()
{
  Serial.println("Initializing SD card...");
  pinMode(CS_PIN, OUTPUT);
  if (SD.begin(CS_PIN))
  {
    Serial.println("SD card is ready to use.");
  }
  else
  {
    Serial.println("SD card initialization failed");
    while (true);
  }
}
// Updated createFile function to clear the file before writing
int createFile(char filename[])
{
  file = SD.open(filename, FILE_WRITE | O_TRUNC);  // O_TRUNC clears the file
  if (file)
  {
    Serial.println("File created (or cleared) successfully.");
    return 1;
  }
```

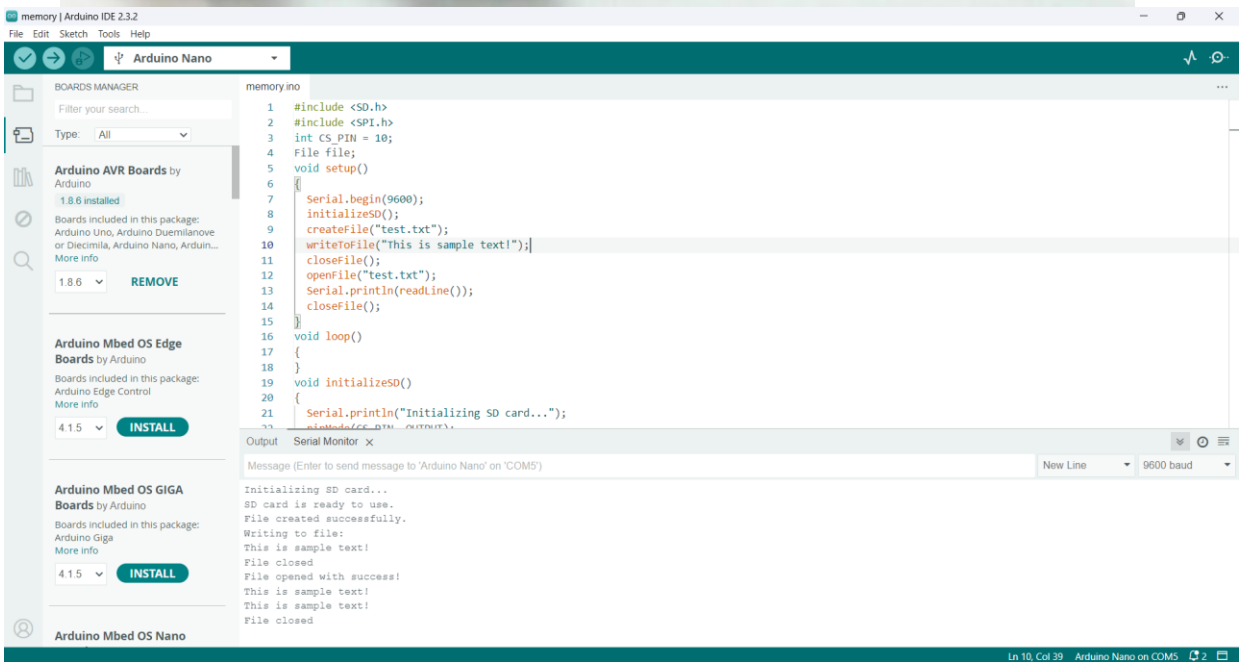
```
  else
  {
    Serial.println("Error while creating file.");
    return 0;
  }
}
int writeToFile(String text)
{
  if (file)
  {
    file.println(text);
    Serial.println("Writing to file: ");
    Serial.println(text);
    return 1;
  }
  else
  {
    Serial.println("Couldn't write to file");
    return 0;
  }
}
void closeFile()
{
  if (file)
  {
```

```
    file.close();
    Serial.println("File closed");
  }
}

int openFile(char filename[])
{
  file = SD.open(filename);
  if (file)
  {
    Serial.println("File opened with success!");
    return 1;
  }
  else
  {
    Serial.println("Error opening file...");
    return 0;
  }
}
String readLine()
{
  String received = "";
  char ch;
  while (file.available())
  {
```

```
    ch = file.read();
    if (ch == '\n')
    {
     return received;
    }
    else
    {
      received += ch;
    }
  }
  return "";
}
long getUltrasonicDistance()
{
  long duration, distance;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;
  return distance;
}
```

OUTPUT:




memory | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Arduino Nano
BOARDS MANAGER
Filter your search...
Type: All
Arduino AVR Boards by Arduino
1.8.6 installed
Boards included in this package: Arduino Uno, Arduino Duemilanove or Diecimila, Arduino Nano, Arduin...
More info
1.8.6
REMOVE
Arduino Mbed OS Edge Boards by Arduino
Boards included in this package: Arduino Edge Control
More info
4.1.5
INSTALL
Arduino Mbed OS GIGA Boards by Arduino
Boards included in this package: Arduino Giga
More info
4.1.5
INSTALL
Arduino Mbed OS Nano
memory.ino
1 #include <SD.h>
2 #include <SPI.h>
3 int CS_PIN = 10;
4 File file;
5 void setup()
6 {
7 Serial.begin(9600);
8 initializeSD();
9 createFile("test.txt");
10 writeToFile("This is sample text!");
11 closeFile();
12 openFile("test.txt");
13 Serial.println(readLine());
14 closeFile();
15 }
16 void loop()
17 {
18 }
19 void initializeSD()
20 {
21 Serial.println("Initializing SD card...");
Output Serial Monitor
Message (Enter to send message to 'Arduino Nano' on 'COM5')
New Line
9600 baud
Initializing SD card...
SD card is ready to use.
File created successfully.
Writing to file:
This is sample text!
File closed
File opened with success!
This is sample text!
This is sample text!
File closed
Ln 10, Col 39 Arduino Nano on COM5

CASE STUDY

4.1 ASSISTIVE TECHNOLOGY FOR VISUAL IMPAIRMENT - ADAPTATIONS IN HOUSEHOLD APPLIANCES

Introduction:

Visual impairment significantly affects daily activities, and household tasks such as laundry can be particularly challenging without proper assistance. Technological advancements in embedded systems have led to the development of assistive devices that help visually impaired individuals perform these tasks more independently. This case study focuses on an embedded system designed to assist users in operating a washing machine, providing features like real-time color recognition and voice synthesis to guide users through the process.

The primary objective of the system is to ensure that users can sort their laundry by color and operate the washing machine without requiring external help. This system's design leverages an ARM STR711FR0 microcontroller, which serves as the core processing unit, managing inputs from sensors and driving the output devices that deliver auditory feedback to the user.

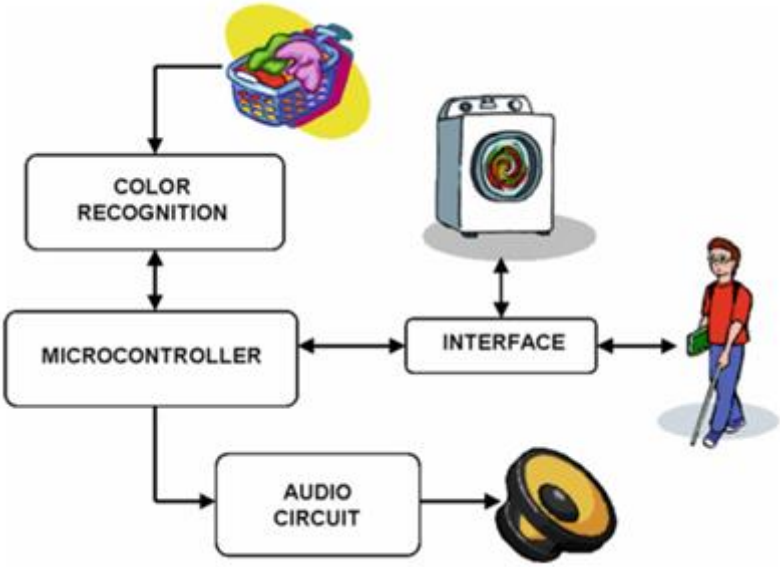
System Overview:

The embedded system integrates several components, each chosen for its specific role in making the washing machine accessible to visually impaired users. The key components include:

Microcontroller*: The ARM STR711FR0 microcontroller is selected for its balance between performance and resource availability, with 16kB of RAM and 64kB of flash memory. It is responsible for processing data from the color sensor and controlling the voice synthesis subsystem.

Color Sensor: The system uses a TCS230 color sensor, which can detect the red, green, and blue components of reflected light. This sensor is essential for distinguishing between different colors of clothing, ensuring that users do not mix incompatible colors in the wash.

Voice Synthesis: The auditory feedback mechanism is built on phonetic voice synthesis, allowing the system to provide real-time, spoken instructions to the user, thus making the washing machine's operation more intuitive.

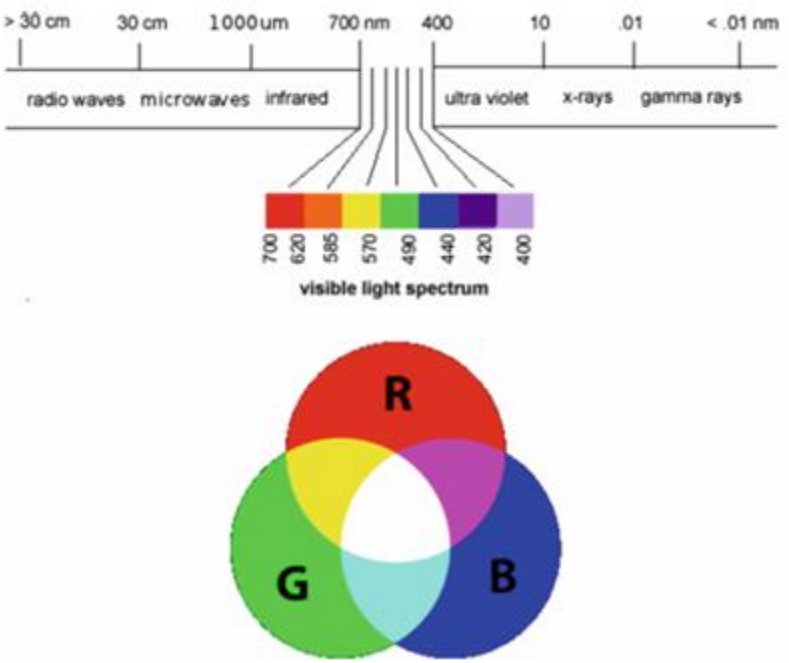


Color Recognition:

The device employs the HSV color model, which is more suited for color recognition tasks than the traditional RGB model due to its separation of chromatic content (hue) from intensity (saturation and value). This model allows the system to more accurately distinguish between different colors, a crucial feature for users who need to prevent color mixing during laundry.

A TCS230 color sensor is used to measure the red, green, and blue components of the reflected light from clothing items. The microcontroller converts the sensor's RGB outputs into the HSV model and matches the detected hue against a predefined

color sample set, simplifying the color recognition process. This ensures reliable identification of colors, aiding the user in correctly sorting their laundry

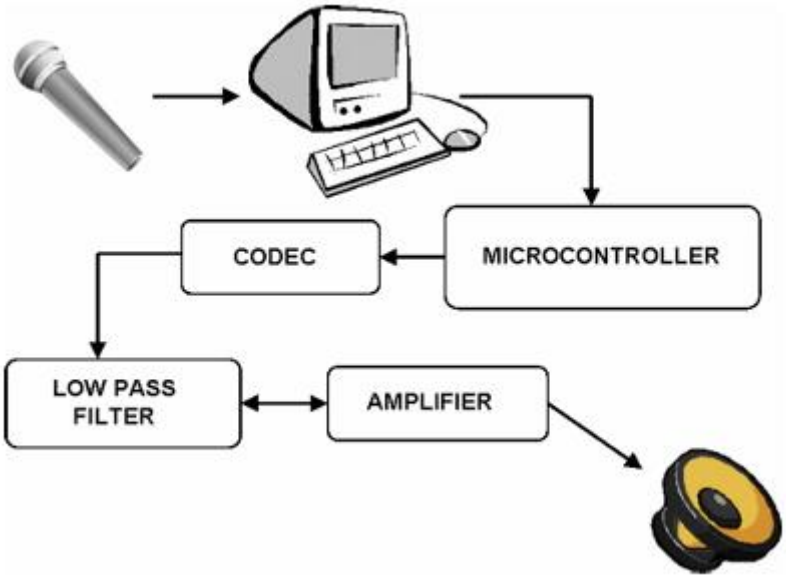


.

Voice Synthesis:

Voice synthesis is implemented using pre-recorded messages and phonetic synthesis. Phonetic synthesis was chosen for its memory efficiency, allowing the system to generate a wide range of spoken feedback without consuming excessive memory space.

The microcontroller retrieves and processes audio data from external flash memory, converting it into an analog signal via a codec for playback. This system provides real-time, clear auditory instructions and feedback to the user, improving the accessibility of the washing machine.



Conclusion:

This case study demonstrates the successful integration of color recognition and voice synthesis in an embedded system designed for visually impaired users. The careful selection of components and algorithms results in a cost-effective, reliable device that significantly enhances the usability of a standard household appliance. Future improvements could include more advanced voice synthesis and additional sensors to further increase the functionality and user experience of the device.

PROJECT DEVELOPMENT

IMAGE-TEXT TO SPEECH CONVERSION USING RASPBERRY PI

5.1 Problem Statement:

Develop a system using a Raspberry Pi to convert text from images into spoken words to enhance accessibility for individuals with visual impairments. The system will involve capturing images containing text using a camera module or USB webcam, extracting the text with Optical Character Recognition (OCR) technology, and converting the extracted text into speech using text-to-speech (TTS) technology. The challenge lies in ensuring accurate text extraction and efficient speech synthesis to provide a seamless user experience. The goal is to create a practical and effective tool that makes printed information accessible through auditory means.

Literature Review:

The integration of Optical Character Recognition (OCR) and Text-to-Speech (TTS) technologies plays a crucial role in converting text from images into spoken words, enhancing accessibility for individuals with visual impairments. OCR tools such as Tesseract and Google Cloud Vision API offer robust text extraction capabilities, leveraging advanced machine learning techniques to improve accuracy (Smith, 2007; Google Cloud, 2021). TTS systems like Festival and eSpeak convert text into natural-sounding speech, with ongoing research focusing on improving speech quality and naturalness (Taylor et al., 1998; eSpeak, 2021). Existing solutions like Seeing AI and OrCam MyEye demonstrate the practical application of these technologies, providing real-time text-to-speech conversion (Microsoft, 2020; OrCam, 2020). Challenges in the field include enhancing text extraction accuracy under varying conditions and optimizing real-time processing to ensure seamless user experiences (Li et al., 2018; Lee et al., 2020). Ongoing advancements continue to address these issues, striving to deliver more effective and user-friendly systems.

Hardware Components:

· Raspberry Pi 4
· Camera Module or USB Webcam
· Speaker (USB or analog)
· MicroSD Card
· Power Supply (5V USB-C)
· Case and Heat Sinks
· Keyboard and Mouse (optional for setup)
· Internet Connection (Wi-Fi or Ethernet)

Software Components:

· Raspberry Pi OS (operating system)
· OpenCV (for image processing)
· Tesseract OCR (for text extraction)
· pytesseract (Python wrapper for Tesseract OCR)
· gTTS (for text-to-speech conversion)
· pygame (for audio playback)
· Python 3 (programming language)

5.2 IMPLEMENTATION:

The implementation process of image-text to speech conversion using raspberry pi 4 include the following steps:

1.Setup the Raspberry Pi:

- Install Raspberry Pi OS on a microSD card and boot the Raspberry Pi 4.
- Connect the Raspberry Pi to a network via Wi-Fi or Ethernet.
- Update the system to ensure all packages are current.

2.Install Required Software:

- Install OpenCV for capturing and processing images.
- Install Tesseract OCR for recognizing and extracting text from images.

- Install gTTS (Google Text-to-Speech) to convert the extracted text into speech.
- Install pygame to handle audio playback on the Raspberry Pi.

3. Capture Image:

- Connect a camera module or USB webcam to the Raspberry Pi.
- Use the camera to capture images containing the text you want to convert.

4. Text Extraction:

- Use Tesseract OCR to extract text from the captured images.
- Ensure that the text extraction process is accurate by testing with different image qualities.

5. Convert Text to Speech:

- Convert the extracted text into speech using the gTTS library.
- Generate an audio file that contains the spoken text.

6. Audio Playback:

- Play the generated audio file through a speaker connected to the Raspberry Pi.
- Test the system to ensure that the text-to-speech conversion is clear and accurate.

7. Testing and Refinement:

- Test the entire process with different images to ensure reliability.
- Refine the image capture, text extraction, and speech synthesis as needed to improve performance.

CODE:

```
from picamera2 import Picamera2, Preview

import time

import subprocess
```

```
import cv2
def process_and_extract_text(image_path):
    # Read the image using OpenCV
    image = cv2.imread(image_path)
    # Convert to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Apply adaptive thresholding to highlight the text
    processed_image = cv2.adaptiveThreshold(
        gray_image,
        255,
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
        cv2.THRESH_BINARY,  11,2
    )
    # Save the processed image (optional, for debugging)
    processed_image_path = image_path.replace('.jpg', '_processed.jpg')
    cv2.imwrite(processed_image_path, processed_image)
  # Convert the processed image to text using Tesseract
    text_output_path = image_path.replace('.jpg', '')
    subprocess.run(['tesseract', processed_image_path, text_output_path])

    # Read the extracted text from the file
    with open(text_output_path + '.txt', 'r') as file:
        extracted_text = file.read()
  return extracted_text
# Initialize the camera
```

```
picam2 = Picamera2()

config = picam2.create_still_configuration(main={"size": (1280, 720)})

picam2.configure(config)

picam2.start()

# Allow the camera to adjust (warm-up time)

time.sleep(2)

# Capture and save the image

image_path = '/home/RPI4/Documents/balaji.jpg'

picam2.capture_file(image_path)

print(f"Image captured and saved as {image_path}")

# Process and extract text

extracted_text = process_and_extract_text(image_path)

print("Extracted Text:")

print(extracted_text)

# Stop the camera

picam2.stop()

# Convert final text to speech using espeak

subprocess.run(['espeak', extracted_text])

print('Text to speech conversion with espeak done')
```

CODE EXPLANATION:

This Python code uses the PiCamera on a Raspberry Pi to capture an image, process it for text extraction, and then convert the extracted text to speech. The PiCamera is initialized, configured, and a still image is captured and saved to a specified file path. The image is then processed using OpenCV, where it is converted to grayscale and undergoes adaptive thresholding to enhance text visibility. Tesseract OCR is used

to extract text from the processed image, with the text being saved to a file and then read back into the program. Finally, the extracted text is converted to speech using the eSpeak command, which reads the text aloud. The code ends by stopping the camera.

This project involves writing a Python script to capture images, process them for text extraction, and then convert the extracted text into speech using the Raspberry Pi. Below is a detailed explanation of the code, covering each key step and its functionality:

1.Importing Libraries:

The script begins by importing necessary libraries: Picamera2 for controlling the Raspberry Pi camera, time for handling delays, subprocess for running shell commands, and cv2 from OpenCV for image processing.

2.Function Definition (process_and_extract_text**):**

This function takes an image file path as input and processes the image to extract text using Tesseract OCR.

Image Loading:

The image is loaded using cv2.imread(), which reads the image into a format that OpenCV can process.

Grayscale Conversion:

The image is converted to grayscale using cv2.cvtColor(). Grayscale conversion simplifies the image, making it easier for OCR to detect text by removing color information.

Adaptive Thresholding:

To enhance the text's visibility, the grayscale image undergoes adaptive thresholding using cv2.adaptiveThreshold(). This technique highlights text areas by adjusting the threshold based on local pixel intensities, making the text stand out against the background.

Saving Processed Image:

The processed image is optionally saved for debugging purposes. This step is useful for verifying that the image preprocessing is working correctly.

Text Extraction:

The processed image is passed to Tesseract OCR via a subprocess call, which generates a text file containing the extracted text. Tesseract is run as a command-line tool, and the text output is saved to a file.

Reading Extracted Text:

The extracted text is read from the generated file using Python's file handling. This text is then returned by the function for further processing.

3.Camera Initialization:

A Picamera2 object is created to control the camera. The camera is configured for still image capture with a resolution of 1280x720 pixels, providing a balance between image quality and processing speed.The camera is started with picam2.start(), allowing it to warm up and adjust to the lighting conditions.

4.Image Capture:

After a short delay (2 seconds) to allow the camera to stabilize, an image is captured and saved to a specified file path on the Raspberry Pi. This image contains the text that will be processed and converted to speech.

5.Text Processing and Extraction:

The captured image is processed by calling the process_and_extract_text() function. The function applies the aforementioned steps to enhance the image and extract the text using OCR.

6.Text-to-Speech Conversion:

The extracted text is then passed to the espeak command via a subprocess call. espeak is a text-to-speech synthesis tool that converts the text into audible speech and plays it through the connected speaker.The subprocess call ensures that the speech synthesis is handled externally by the system's command-line interface, allowing the script to focus on image processing and OCR.

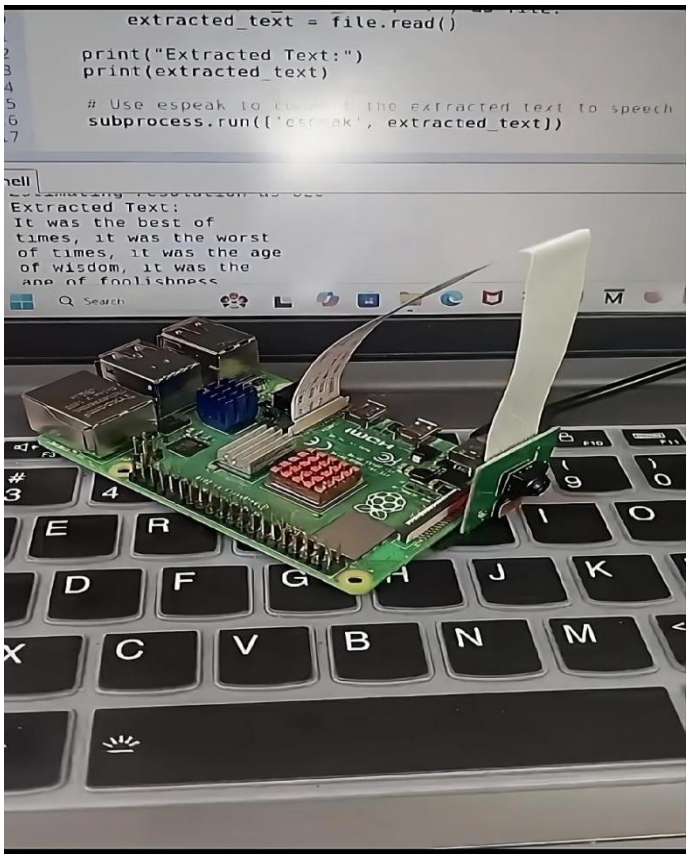
7.Stopping the Camera:

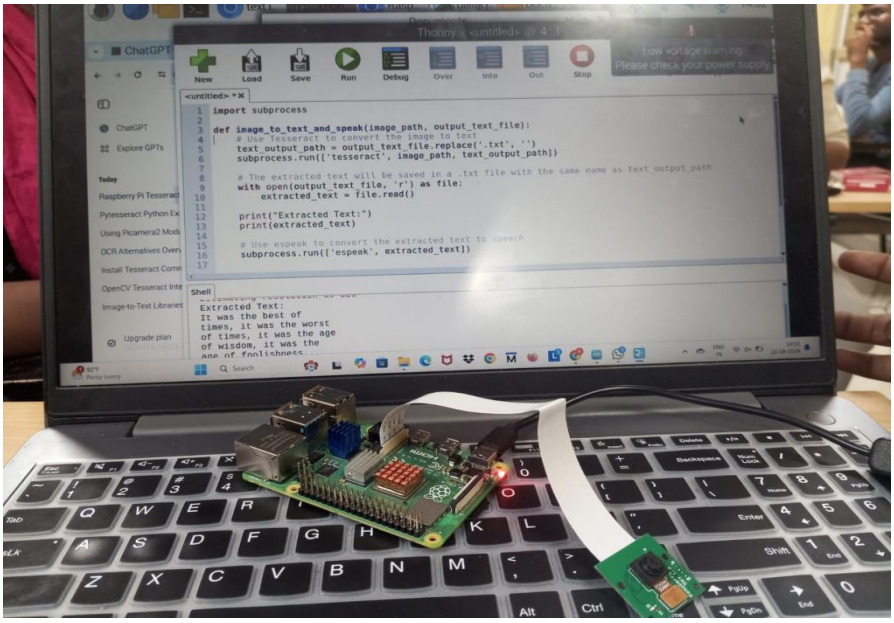
Once the image has been processed and the text has been converted to speech, the camera is stopped using picam2.stop(). This step is crucial for releasing the camera resources, preventing any conflicts with other applications that might need access to the camera.

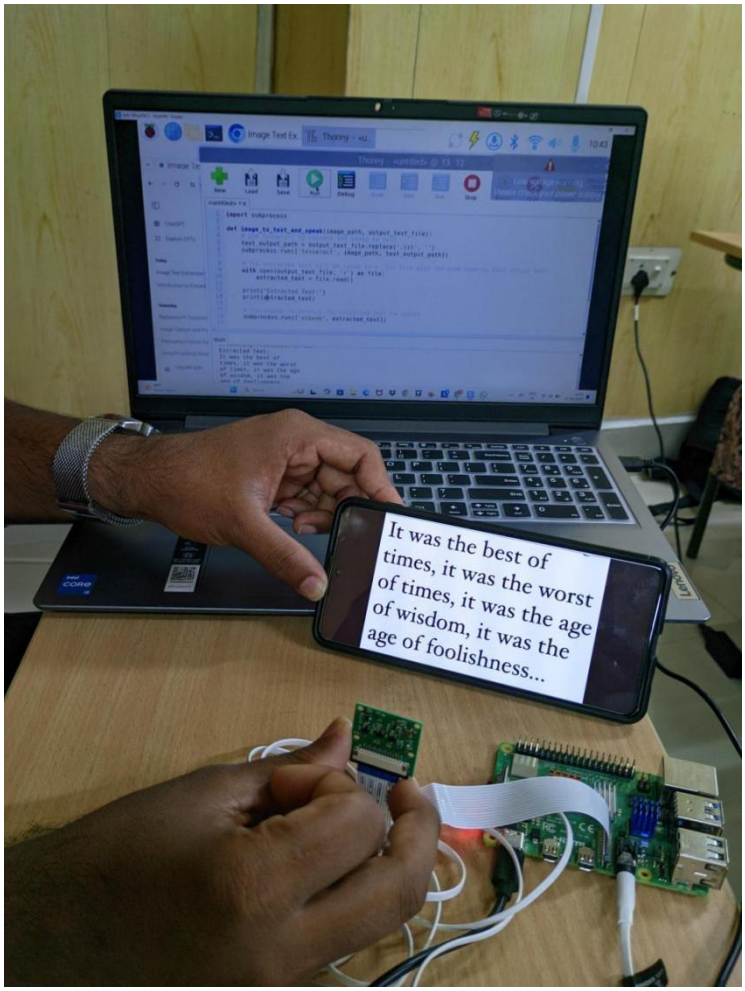
8.Final Output:

The script prints the captured image's file path, the extracted text, and confirms that the text-to-speech conversion has been completed. These messages provide feedback to the user, indicating the status of each step in the process.

5.3 OUTPUT:


extracted_text = file.read()
print("Extracted Text:")
print(extracted_text)
# Use espeak to
the extracted text to speech
subprocess.run(['espeak', extracted_text])
Extracted Text:
It was the best of
times, it was the worst
of times, it was the age
of wisdom, it was the


import subprocess
def image_to_text_and_speak(image_path, output_text_file):
# Use Tesseract to convert the image to text
text_output_path = output_text_file.replace('.txt', '')
subprocess.run(['tesseract', image_path, text_output_path])
# The extracted text will be saved in a .txt file with the same name as text_output_path
with open(output_text_file, 'r') as file:
extracted_text = file.read()
print("Extracted Text:")
print(extracted_text)
# Use espeak to convert the extracted text to speech
subprocess.run(['espeak', extracted_text])
Extracted Text:
It was the best of
times, it was the worst
of times, it was the age
of wisdom, it was the


It was the best of
times, it was the worst
of times, it was the age
of wisdom, it was the
age of foolishness...

CONCLUSION

The Image Text-to-Speech Conversion project using Raspberry Pi successfully demonstrates the integration of image processing, optical character recognition (OCR), and text-to-speech (TTS) technologies in a compact and efficient system. Throughout the project, a Raspberry Pi 4 was used to capture images, extract text from these images using Tesseract OCR, and convert the text into audible speech via the eSpeak utility. This system provides an accessible solution for individuals with visual impairments or reading difficulties, allowing them to interact with printed materials through auditory feedback.

The project highlighted the importance of image preprocessing for improving OCR accuracy and the seamless integration of different software components to achieve the desired functionality. Although the system performed well in controlled conditions, further enhancements could include improving text extraction accuracy with more advanced preprocessing techniques and incorporating natural language processing (NLP) to better interpret and vocalize the extracted text.

In conclusion, this project lays a solid foundation for creating assistive technologies that leverage the Raspberry Pi's versatility. It demonstrates how a low-cost, compact device can be used to create practical solutions that address real-world accessibility challenges. Future work could explore scaling this solution for broader applications, including multi-language support and real-time processing capabilities.

REFERENCES:

1. https://docs.arduino.cc/
2. https://www.raspberrypi.com/documentation/