Bandile Sebenele Dlamini

Arcadia, Pretoria 0007

Cell: (+27) 74 549 6293

Email: mbandidlamini@gmail.com

LinkedIn: www.linkedin.com/in/bandile-sebenele-dlamini-474926286

**Portfolio of Evidence**

The portfolio showcases a series of IoT projects using the ESP32 microcontroller, highlighting my skills in circuit design, embedded programming, and sensor integration. Each project shows practical applications, from traffic light control to environmental monitoring, showing real-world relevance. Through these projects, I gained hands-on experience in hardware-software integration, troubleshooting, and user-centric design.

**Project 01 - Traffic light controller**

This project simulates on how a basic traffic light we see on our roads work

**Objective:** Design and implement a traffic light controller using an ESP32 microcontroller programmed in Arduino C++ IDE. The controller should manage 3 LEDs representing red, yellow and green. When each LED is illuminated, a corresponding message should be displayed in the serial monitor of the Arduino IDE e.g. when a red LED is on a message displayed should be STOP, CAUTION for a yellow LED and GO for an illuminated green LED

**Instructions:** Use Wokwi to design and simulate the circuit diagram for the traffic light controller. Write a program in the Arduino IDE using C++ to control the traffic light sequence. Draw a circuit diagram

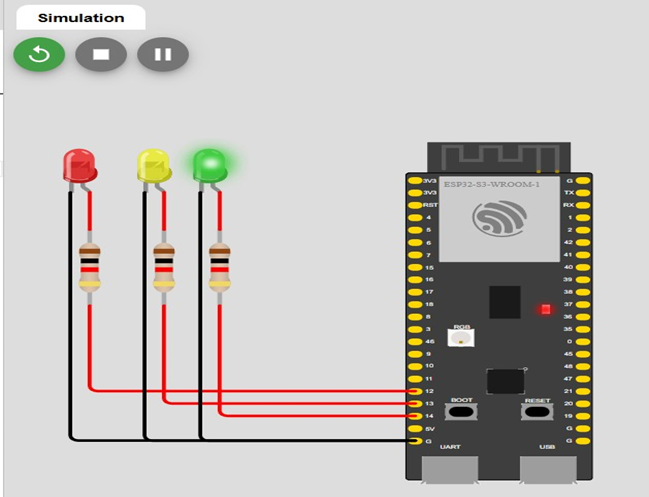


Figure 1: Circuit diagram for traffic light controller

// Define GPIO pins for LEDs

const int RED\_LED = 13;

const int YELLOW\_LED = 12;

const int GREEN\_LED = 14;

// Define delay times (in milliseconds) for each light state

const int GREEN\_LIGHT\_TIME = 15000; // Green light duration (15 seconds)

const int YELLOW\_LIGHT\_TIME = 5000; // Yellow light duration (5 seconds)

const int RED\_LIGHT\_TIME = 10000; // Red light duration (10 seconds)

void setup() {

// Initialize the LED pins as outputs

pinMode(RED\_LED, OUTPUT);

pinMode(YELLOW\_LED, OUTPUT);

pinMode(GREEN\_LED, OUTPUT);

}

void loop() {

// Green light on

digitalWrite(GREEN\_LED, HIGH);

digitalWrite(YELLOW\_LED, LOW);

digitalWrite(RED\_LED, LOW);

delay(GREEN\_LIGHT\_TIME);

// Yellow light on

digitalWrite(GREEN\_LED, LOW);

digitalWrite(YELLOW\_LED, HIGH);

digitalWrite(RED\_LED, LOW);

delay(YELLOW\_LIGHT\_TIME);

// Red light on

digitalWrite(GREEN\_LED, LOW);

digitalWrite(YELLOW\_LED, LOW);

digitalWrite(RED\_LED, HIGH);

delay(RED\_LIGHT\_TIME);

}

**Project 02 - LDR (Light Dependent Resistance) sensor Data display**

The project shows how an LDR sensor is used in industries to control light intensity

LDRs are used wherever light-sensitive control or measurement is needed because they are simple, reliable, and inexpensive. Examples include Street Lighting: LDRs are commonly used in streetlights to detect daylight. When it gets dark, the resistance of the LDR decreases, triggering the streetlights to turn on automatically. Security Systems: LDRs are often included in alarm systems. For instance, if an intruder disrupts a light source, the change in light can trigger an alarm. Automatic Brightness Control in Screens: Some smartphones, laptops, and TVs use LDRs to adjust screen brightness based on ambient light conditions. Industrial and Scientific Instruments: LDRs are used in equipment that measures light levels, such as in agricultural applications to monitor sunlight exposure for plants just to mention a few.

**Objective:**  Design and implement a system using an ESP32 microcontroller to read data from an LDR sensor and display on the Arduino IDE serial monitor. The system should accurately measure the light intensity detected by the sensor and provide real time feedback through the serial interface.

**Instructions:**

1. Connect the LDR sensor to one of the analog input pins of the ESP32
2. Connect an LED indicator of the output that will illuminate based on predefined light intensity
3. When light intensity exceeds a certain threshold, an LED should blink
4. Write a program in the Arduino IDE in C++ language to read data from the LDR sensor
5. Draw the circuit diagram

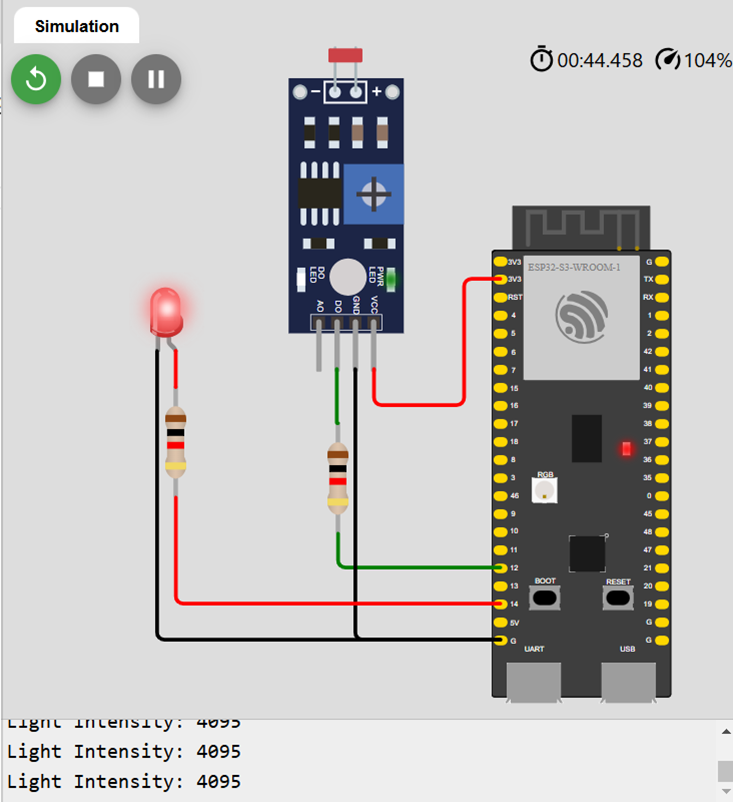


Figure 2: Circuit diagram for LDR sensor

// Define pins

const int LDR\_PIN = 12; // Analog pin connected to LDR (GPIO 12 on ESP32)

const int LED\_PIN = 14; // Digital pin connected to LED (GPIO 14 on ESP32)

// Threshold value for light intensity

const int LIGHT\_THRESHOLD = 2000; // Adjust this value based on testing

void setup() {

// Initialize the LED pin as an output

pinMode(LED\_PIN, OUTPUT);

// Initialize serial communication for monitoring

Serial.begin(115200);

}

void loop() {

// Read the analog value from the LDR

int lightIntensity = analogRead(LDR\_PIN);

// Print the LDR reading to the Serial Monitor

Serial.print("Light Intensity: ");

Serial.println(lightIntensity);

// Check if the light intensity is below the threshold

if (lightIntensity < LIGHT\_THRESHOLD) {

// Turn on LED if it's dark

digitalWrite(LED\_PIN, HIGH);

} else {

// Turn off LED if there's enough light

digitalWrite(LED\_PIN, LOW);

}

// Small delay for stability

delay(500);

}

**Project 3 - Digital temperature humidity data display with LED indicators (DHT22)**

The project shows how a DHT22 sensor is used in industries to measure temperature and humidity

The DHT22 sensor, which measures temperature and humidity, is widely used in various real-world applications due to its accuracy, affordability, and ease of integration. Weather Stations: DHT22 sensors are often used in DIY and commercial weather stations to monitor local environmental conditions. Data Centers and Server Rooms: These sensors help maintain ideal environmental conditions to protect sensitive electronic equipment by monitoring temperature and humidity levels. Greenhouses and Agriculture: In greenhouses, gardens, and farms, DHT22 sensors help regulate and monitor the temperature and humidity levels, which are critical for plant growth.

**Objective:** Design and implement a system using an ESP32 to read data from a DHT22 temperature and humidity sensor. Display both temperature and humidity readings on a serial monitor. Use LEDs to alert users when temperature or humidity exceeds certain limits

**Instructions:** Connect the DHT22 to the ESP32.Write an Arduino C++ code to interface with the DHT22 sensor and read temperature and humidity data. Draw the circuit diagram for the above project

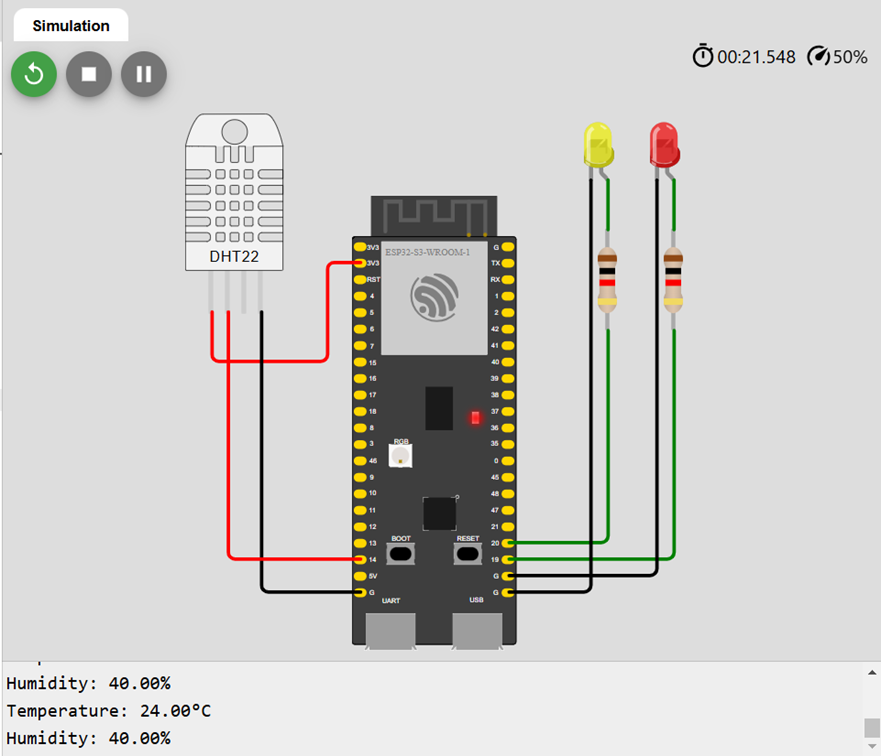


Figure 3:Circuit diagram for DHT22 temperature and humidity sensor

#include <DHT.h>

#include <DHT\_U.h>

// Define the DHT22 pin and type

#define DHTPIN 14 // GPIO pin connected to DHT22 data pin

#define DHTTYPE DHT22 // DHT 22 (AM2302)

// Define LED pins

#define RED\_LED 19 // GPIO pin connected to red LED

#define YELLOW\_LED 20 // GPIO pin connected to yellow LED

// Initialize DHT sensor

DHT dht(DHTPIN, DHTTYPE);

// Temperature and humidity thresholds

const float TEMP\_THRESHOLD = 22.0;

const float HUMIDITY\_THRESHOLD = 30.0;

void setup() {

// Begin serial communication

Serial.begin(115200);

Serial.println(F("DHT22 Sensor Initialization"));

// Start the DHT sensor

dht.begin();

// Initialize LED pins as outputs

pinMode(RED\_LED, OUTPUT);

pinMode(YELLOW\_LED, OUTPUT);

}

void loop() {

// Wait a few seconds between measurements

delay(2000);

// Read temperature in Celsius

float temperature = dht.readTemperature();

// Read humidity

float humidity = dht.readHumidity();

// Check if readings are valid

if (isnan(temperature) || isnan(humidity)) {

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

return;

}

// Print temperature and humidity values to Serial Monitor

Serial.print(F("Temperature: "));

Serial.print(temperature);

Serial.println(F("°C"));

Serial.print(F("Humidity: "));

Serial.print(humidity);

Serial.println(F("%"));

// Check temperature threshold

if (temperature > TEMP\_THRESHOLD) {

// Blink red LED if temperature exceeds threshold

digitalWrite(RED\_LED, HIGH);

delay(500);

digitalWrite(RED\_LED, LOW);

}

// Check humidity threshold

if (humidity > HUMIDITY\_THRESHOLD) {

// Blink yellow LED if humidity exceeds threshold

digitalWrite(YELLOW\_LED, HIGH);

delay(500);

digitalWrite(YELLOW\_LED, LOW);

}}

**Key takeaways**

1. **Intergration of Hardware components**- became skilled at integrating various hardware components, including sensors, keypads, LEDs, resistors just to mention a few with a microcontroller (ESP32). This helped me better understand circuit design and troubleshooting.
2. **Software Development for Embedded Systems**: Writing software to manage hardware inputs and outputs, particularly using Arduino IDE, this gave me practical experience in embedded systems programming.
3. **Problem-Solving Skills:** I have encountered and overcame challenges, such as component compatibility and detection accuracy, by using modular design and advanced algorithms, demonstrating adaptability.
4. **User-Centric Design:** I focused on creating a user-friendly interface, demonstrating a practical approach to designing for end-users.
5. **Technical Depth in Sensor Integration:** This project deepened my knowledge in sensor technology, microcontrollers, and IoT. The experience i gained here forms a strong foundation for future work in embedded systems or IoT, where sensors and microcontrollers play a central role

**Areas of improvement**

1. **Energy Efficiency**: Since the above projects have a potential to be implemented in remote areas, prioritizing power optimization techniques would enhance their practicality and sustainability.
2. **Cost Management:** Exploring lower-cost alternatives for some hardware components could make solution more scalable.
3. **Privacy Measures:** Developing more explicit data security measures to address privacy concerns could improve system compliance with privacy standards and increase user trust.
4. **Testing and Iteration:** Continuous testing in different environmental conditions (lighting, temperature) will further enhance detection reliability.
5. **Advanced Detection Algorithms:** While projects include basic detection systems, implementing more sophisticated algorithms, like AI-based image recognition, could boost detection accuracy and reduce false results.
6. **Scalability for Larger Systems**: Building the system to handle larger, multi-site environments could broaden its applications, like integrating with centralized databases or monitoring hubs. Adding features like remote management through a cloud interface or app would also expand usability.
7. **Environmental Adaptability**: Modifying the system to adapt to varying environmental conditions (e.g., different lighting, weather, or movement patterns) could reduce error rates and improve detection reliability in diverse locations.