Naan Mudhalvan

Public Transportation Optimization

Phase-5

DEVELOPMENT PHASE 2

This code is for an ESP32-based system that uses ultrasonic sensors and an LCD display to monitor the presence of objects, like passengers, within a certain proximity. It integrates the Blynk platform for remote monitoring of passenger counts. Here's a step-by-step explanation of what's happening in the code:

1. **Blynk Setup:**

   - The code starts by defining Blynk-related constants:

     - `BLYNK_TEMPLATE_ID`, `BLYNK_TEMPLATE_NAME`: Template ID and name used for Blynk.

     - `BLYNK_AUTH_TOKEN`: The Blynk authentication token that allows the device to communicate with the Blynk server.

2. **Library Inclusion:**

   - The code includes necessary libraries:

     - `WiFi.h`: For Wi-Fi connectivity.

     - `WiFiClient.h`: For the Wi-Fi client.

     - `BlynkSimpleEsp32.h`: Blynk library for ESP32.

     - `LiquidCrystal_I2C.h`: A library for controlling the I2C-connected LCD.

3. **Variable Declarations:**

- Various variables are declared:
  - `distanceThreshold`: A threshold distance (in centimeters) to determine when an object (passenger) is near.
  - Variables for timing and distance measurements.
  - `entered` and `left`: Counts for passengers who entered and left.
  - `boarding`: A boolean variable to track if the boarding process is in progress.
  - `LED` and pin definitions for ultrasonic sensors (trigger and echo pins).
  - `BlynkTimer`: A timer object for scheduling tasks.
  - `LiquidCrystal_I2C` object for controlling the I2C LCD.
  - `auth`, `ssid`, and `pass`: Authentication token and Wi-Fi network credentials.

4. **Distance Measurement Functions:**

- Two functions, `get_distance1` and `get_distance2`, are defined to measure distances using ultrasonic sensors. These functions send trigger signals, measure echo signal durations, and calculate distances based on the speed of sound.

5. **`myTimer` Function:**

- The `myTimer` function is scheduled to run at intervals by Blynk.
- It calls the distance measurement functions (`get_distance1` and `get_distance2`) to get the distances from two ultrasonic sensors.

- It checks if no boarding process is in progress and if the distances from both sensors are less than the `distanceThreshold`. If so, it sets `boarding` to true (boarding is in progress).

- If `boarding` is true and either of the distances exceeds the `distanceThreshold`, it means the boarding is complete. The code sets `boarding` to false and increments both `entered` and `left` counts.

- It updates the LCD display with the "Passengers" count and sends this count to Blynk on virtual pins `V0` and `V1`.

6. **Setup Function:**

- The `setup` function is called once at the start of the program.

- It initializes the serial communication for debugging.

- Sets up pins for the LED and the ultrasonic sensors (trigger as output and echo as input).

- Initializes the I2C-connected LCD and turns on the backlight.

- Begins the Blynk connection with the provided authentication token, Wi-Fi credentials, and Blynk server address.

- Schedules the `myTimer` function to run every 1000 milliseconds (1 second) using the Blynk timer.

7. **Loop Function:**

- The `loop` function is called continuously.

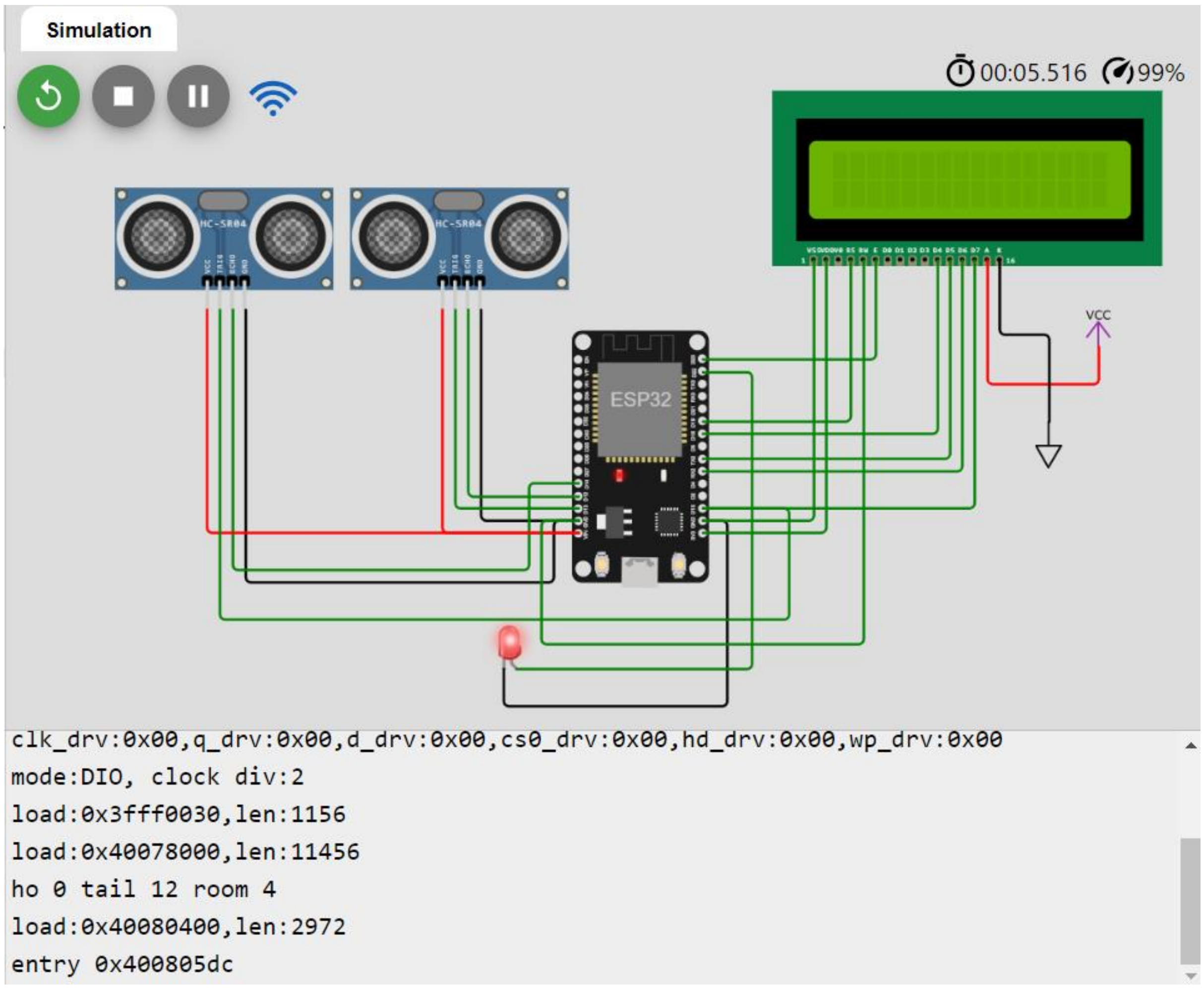
- It keeps the Blynk connection active and runs scheduled tasks via the Blynk timer.

In summary, this code creates an ESP32-based system that uses ultrasonic sensors to monitor the presence of objects within a certain proximity and displays the passenger count on an LCD. It also sends this count to the Blynk platform for remote monitoring. The code frequently checks the distances, updates the count, and informs Blynk about the passenger count changes.

```
#define BLYNK_TEMPLATE_ID "Your_Template_ID"
#define BLYNK_TEMPLATE_NAME "Your_Template_Name"
#define BLYNK_AUTH_TOKEN "XEHxNF_Ur1Nt2p7wB5B20dNI1ZUwj34P"
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include <LiquidCrystal_I2C.h>

int distanceThreshold = 20;  // Adjust this value as needed
int duration1 = 0;
int distance1 = 0;
int duration2 = 0;
int distance2 = 0;
int dis1 = 0;
int dis2 = 0;
int entered = 0;
int left = 0;
bool boarding = false; // Track if boarding is in progress
#define LED 2
#define PIN_TRIG1 15
#define PIN_ECHO1 14
#define PIN_TRIG2 13
#define PIN_ECHO2 12
BlynkTimer timer;
```

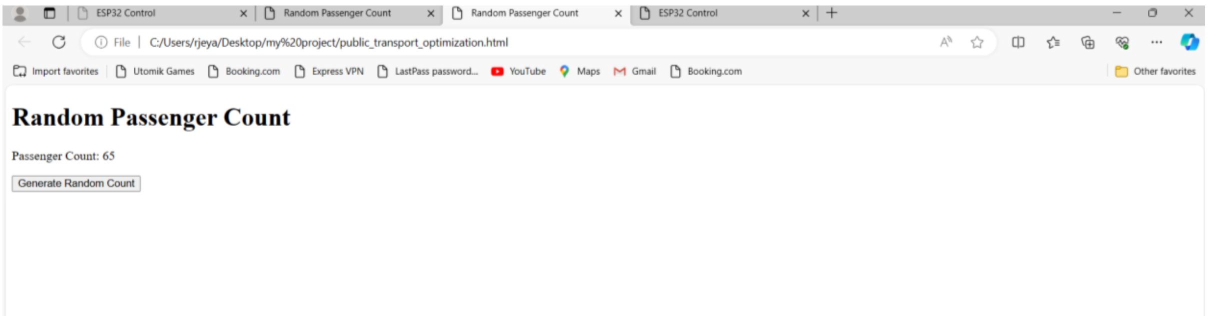
```
LiquidCrystal_I2C lcd(0x27, 16, 2);  // Adjust the I2C address if needed
char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "Wokwi-GUEST";   // your network SSID (name)
char pass[] = "";
#define BLYNK_PRINT Serial

long get_distance1() {
  // Start a new measurement:
  digitalWrite(PIN_TRIG1, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG1, LOW);

  // Read the result:
  duration1 = pulseIn(PIN_ECHO1, HIGH);
  distance1 = duration1 / 58;
  return distance1;
}

long get_distance2() {
  // Start a new measurement:
  digitalWrite(PIN_TRIG2, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG2, LOW);

  // Read the result:
  duration2 = pulseIn(PIN_ECHO2, HIGH);
  distance2 = duration2 / 58;
  return distance2;
}
```

```
void myTimer() {
  dis1 = get_distance1();
  dis2 = get_distance2();

  if (!boarding && dis1 < distanceThreshold && dis2 < distanceThreshold) {
    boarding = true; // Boarding in progress
  } else if (boarding and (dis1 > distanceThreshold || dis2 > distanceThreshold)) {
    boarding = false; // Boarding is complete
    entered++;
    left++;
  }

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Passengers: ");
  lcd.print(entered);

  Blynk.virtualWrite(V0, entered);
  Blynk.virtualWrite(V1, left);
}

void setup() {
  Serial.begin(115200);
  pinMode(LED, OUTPUT);
  pinMode(PIN_TRIG1, OUTPUT);
  pinMode(PIN_ECHO1, INPUT);
  pinMode(PIN_TRIG2, OUTPUT);
  pinMode(PIN_ECHO2, INPUT);
  // Initialize the LCD in the setup function
  lcd.init();
  lcd.backlight();

  Blynk.begin(auth, ssid, pass, "blynk.cloud", 8080);
  timer.setInterval(1000L, myTimer);
}

void loop() {
  Blynk.run();
  timer.run();
}
```



```
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:1156
load:0x40078000,len:11456
ho 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc
```

HTML WEBPAGE



Random Passenger Count

Passenger Count: 65

Generate Random Count

```
<!DOCTYPE html>
<html>
<head>
  <title>Random Passenger Count</title>
</head>
```

```
<body>
  <h1>Random Passenger Count</h1>
  <p>Passenger Count: <span id="passengerCount">0</span></p>
  <button onclick="generateRandomCount()">Generate Random Count</button>

  <script>
    function generateRandomCount() {
      // Generate a random passenger count between 0 and 100
      var randomCount = Math.floor(Math.random() * 101);

      // Display the random count on the web page
      document.getElementById("passengerCount").textContent = randomCount;
    }
  </script>
</body>
</html>
```

This HTML code creates a simple web page for displaying a random passenger count and allows the user to generate a new random count with the click of a button. Here's an explanation of the code:

1. **HTML Structure:**

  - `<!DOCTYPE html>`: This declaration defines the document type and version of HTML being used (HTML5 in this case).

  - The `<html>` element encloses the entire HTML document.

  - The `<head>` section typically contains metadata about the document, but in this code, it includes the page title.

2. **Page Title:**

   - `<title>`: This element sets the title of the web page, which is displayed in the browser's title bar or tab.

3. **Body Content:**

   - `<body>`: This element contains the visible content of the web page.

4. **Heading and Passenger Count Display:**

   - `<h1>`: This is a top-level heading that displays the text "Random Passenger Count."

   - `<p>`: This is a paragraph element that contains the text "Passenger Count:".

   - `<span id="passengerCount">0</span>`: Within the paragraph, a `<span>` element is used to display the passenger count. The `id` attribute is set to "passengerCount" to uniquely identify this element.

5. **Generate Random Count Button:**

   - `<button>`: This is a button element that allows the user to interact with the page.

   - `onclick="generateRandomCount()"`: The `onclick` attribute specifies a JavaScript function to run when the button is clicked. In this case, it calls the `generateRandomCount` function when the button is clicked.

6. **JavaScript Function:**

  - `<script>`: This element is used to include JavaScript code in the HTML document.

  - `function generateRandomCount() { ... }`: This defines a JavaScript function named `generateRandomCount`. When called, it does the following:

    - Generates a random passenger count between 0 and 100 using the `Math.random()` function, which returns a random number between 0 (inclusive) and 1 (exclusive).

    - `Math.floor()` is used to round the random number down to the nearest integer, ensuring it's within the specified range.

    - Updates the text content of the `<span>` element with the ID "passengerCount" to display the new random count.

In summary, this code creates a web page with a title, a heading, a display area for the passenger count, and a button. When the button is clicked, a JavaScript function generates a random passenger count and updates the display. This simple web page demonstrates how HTML and JavaScript can be used together to create interactive content for users.