PRO-C263

Parking Assist

What is our GOAL for this CLASS?

In this class, we learned how to create an ESP32 program that helps parking safely using distance measuring HC-SR04 Ultrasonic Sensors.

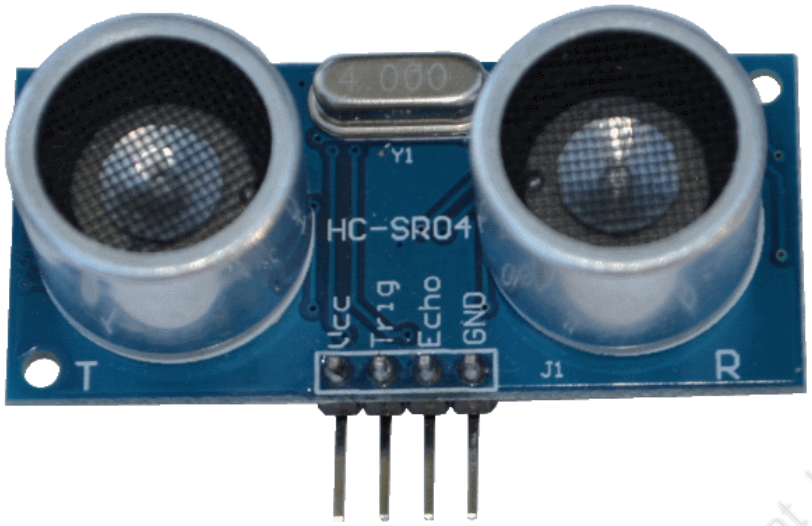
What did we ACHIEVE in the class TODAY?

- We understood the HC-SR04 Ultrasonic Sensor.
- We learnt how to connect a HC-SR04 with ESP32.
- We learnt how to write an ESP32 parking assist program using HC-SR04 Ultrasonic Sensor, LED Bar graph, Buzzer and Resistor.

How did we DO the activities?

1. **We learnt about HC-SR04 Ultrasonic Sensor:**
   Ultrasonic sensors that can measure the distance to a wide range of objects regardless of shape, color or surface texture. They are also able to measure an approaching or receding object. By using “non-contact” ultrasonic sensors, distances can be measured without damage to the object.
   It uses ultrasonic waves.

   One such sensor is HC-SR04. It has a range from 2cm to 400cm (about an inch to 13 feet).
   Below is how the sensor looks like.



https://s3-whjr-curriculum-uploads.whjr.online/5965a4b6-e37d-4278-b084-f838dad9684d.png

On either side the circular shapes represent T-Transmitter and R- Receiver. Transmitter emits the ultrasonic waves and the Receiver captures these ultrasonic waves.

It has 4 pins: TRIG,ECHO,VCC,GND

We have two pins other than **VCC** and **GND** on the HC-SR04. They are **TRIG** and **ECHO** pins. Let's understand and use them one by one.

TRIG pin is used to send the signal to the sensor that it can start transmitting the Ultrasonic wave.

ECHO pin is used for reception of the wave. As soon as the wave is received by the receiver, the ECHO pin sends the signal.

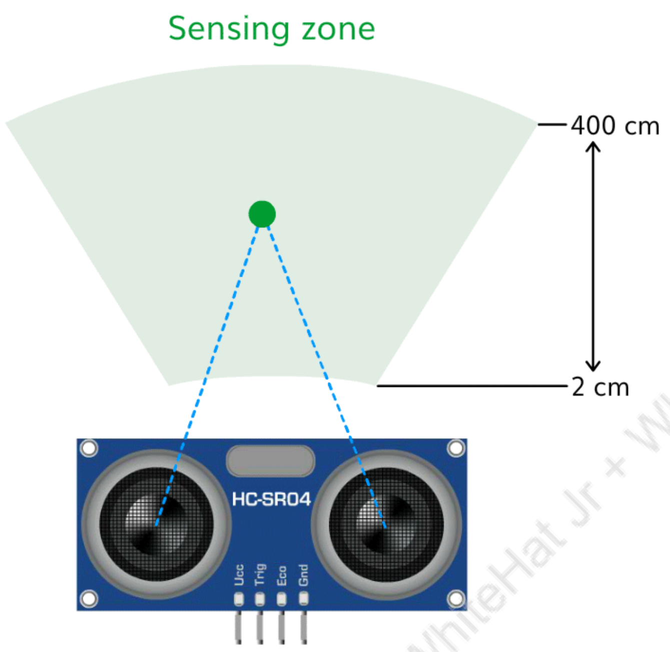
Below gif helps us understand the working of HC-SR04 and the TRIG and ECHO pins.

https://s3-whjr-curriculum-uploads.whjr.online/835baf56-250c-43c8-af9a-0f02086601d6.gif

Blind Zone:

Blind spots are areas or zones that cannot be seen by a viewer while looking at the

rear view. The person must turn his or her head in order to see it.
"The Blind Zone" is the shortest permissible sensing range. This means that no objects or targets are permitted within the minimum working area ("Blind Zone") as this would false trigger the sensor.



https://s3-whjr-curriculum-uploads.whjr.online/ab36b7b8-3433-40bb-8412-350a582eee41.gif

2. **We learnt about LED Bar graphs and Resistors:**

   LED Bar Graph:
   The bar graph - a series of LEDs in a line is made up of a series of LEDs in a row to allow usage of multiple LEDs together controlled by a little code. Read more about it here.

   Resistors:
   Resistors are used to limit the amount of current going to certain components in the circuit, such as LEDs and integrated circuits.

3. **We learnt how to connect and use HC-SR04 Ultrasonic Sensor with ESP32:**
   a. Open the wokwi simulator and create a new ESP32 project.
   b. Create the circuit diagram.

Step 1: Select the components:

- 1 x **ESP32**
- 1 x **HC-SR04 Ultrasonic Distance Sensor**

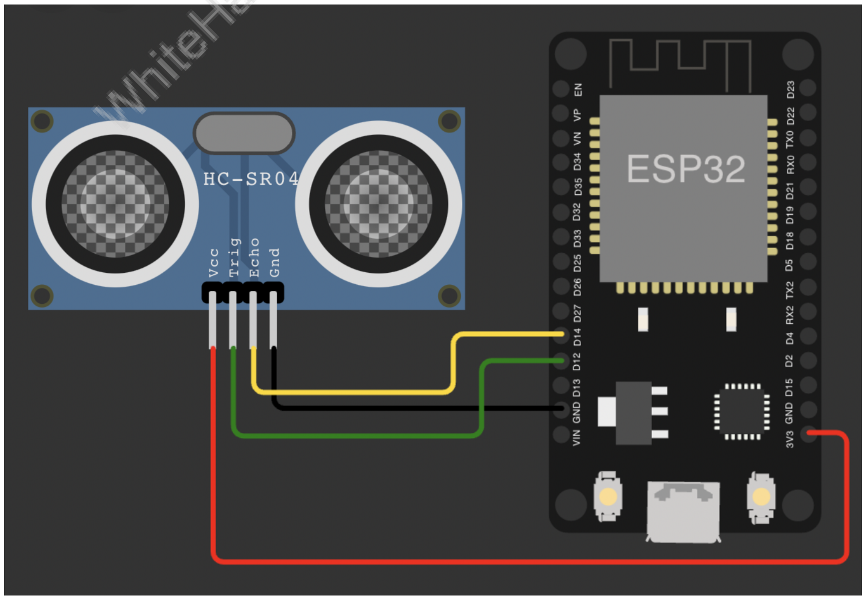
Step 2: Make the connections:

The circuit of this project consists of an **ESP32** Controller, and a **HC-SR04 Ultrasonic Distance Sensor.**

HC-SR04	ESP32 PIN
VCC	VCC 3V
GND	GND
TRIG	GPIO 12
ECHO	GPIO 14

Note: Wire color can be changed by ***clicking over it*** *and selecting the color, or via* ***diagram.json*** *file. Go to the diagram.json wire and change the color of the wire. Any design changes or color changes can be done via the diagram.json file. Keep the track of the component and change the design settings.*

Reference image for connections:



c. Define the **TRIG** and **ECHO** pin. In this case they are the pins number 12 and

14 on the ESP32 Board and they are named trig_pin and echo_pin.

```
const byte trig_pin = 12;

const byte echo_pin = 14;
```

d. Next, in the **setup()** we define the trig_pin as an output and the echo_pin as an Input and also start the serial communication for showing the results on the serial monitor.

```
pinMode(trig_pin , OUTPUT);

pinMode(echo_pin , INPUT);
```

e. Next, we create a function to calculate the distance to ease the work.

```
int get_distance(){


 return distance;


}
```

f. We set the trig_pin on HIGH State for 10 µs for sending the ultrasonic wave.

```
 //  sending trigger burst
 digitalWrite(trig_pin , HIGH);
 delayMicroseconds(10);
 digitalWrite(trig_pin , LOW);
```

Using the **pulseIn**() function we read the travel time and put that value into the variable "duration". This function has 2 parameters, the first one is the name of the Echo pin and for the second is the state of the pulse we are reading, either HIGH or LOW.

```
//  waiting for response : checking HIGH duration of echo pin
 long int duration = pulseIn(echo_pin , HIGH);
 long int distance = 0;
```

In this case, we need this set to HIGH, as the HC-SR04 sensor sets the Echo pin to High after sending the 8 cycle ultrasonic burst from the transmitter. This actually starts the timing and once we receive the reflected sound wave the Echo pin will go to Low which stops the timing. At the end the function will return the length of the pulse in microseconds.

g. Also clear the noise before sending the burst

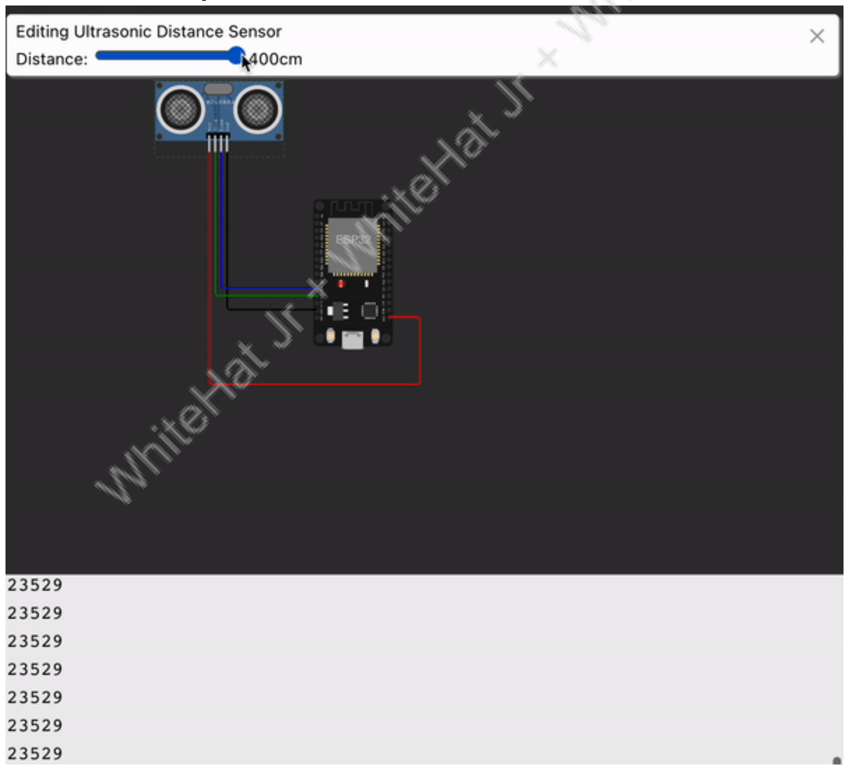
```
//  getting ready before sending tigger burst
digitalWrite(trig_pin , LOW);
delayMicroseconds(2);
```

h. Next, we call the function that is calculating the distance in the loop().

```
int distance = get_distance();
```

We checked the output using Serial.println() and saw the output in duration(milliseconds).

Reference Output:

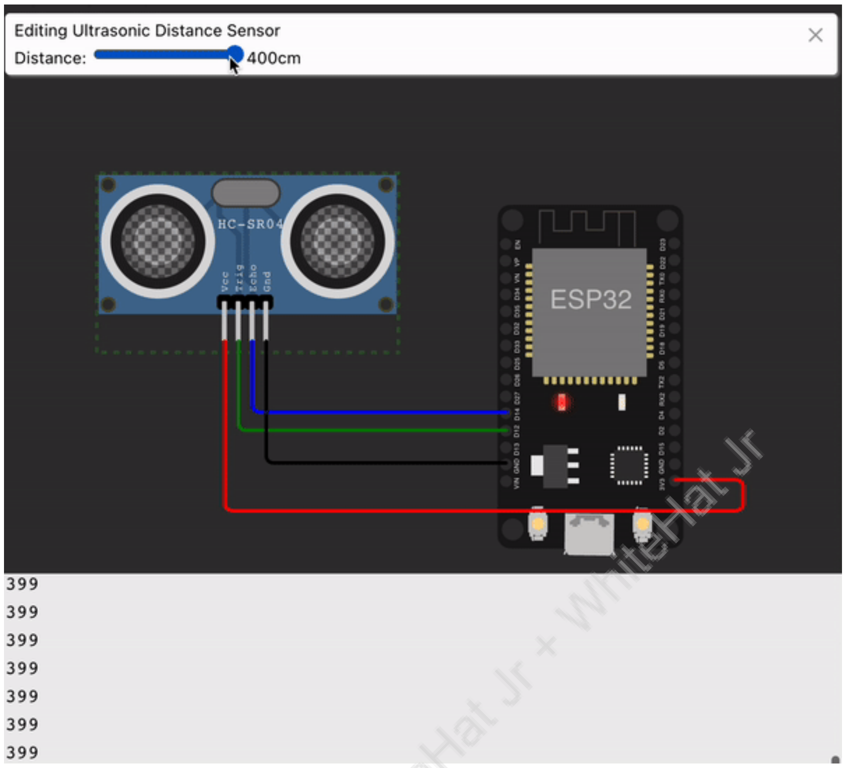


```
23529
23529
23529
23529
23529
23529
23529
```

For getting the distance we will multiply the duration by 0.034(speed of Ultrasonic wave) and divide it by 2 as the time known is for both to and fro.

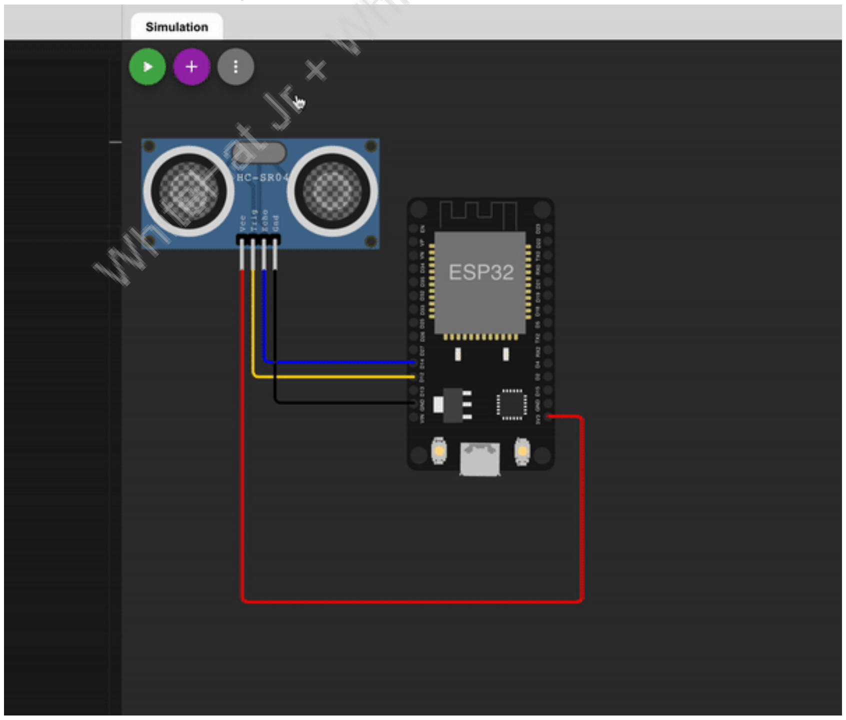
```
float sound_speed = 0.034;
long int distance = (duration * sound_speed) / 2;
```

Output Reference:

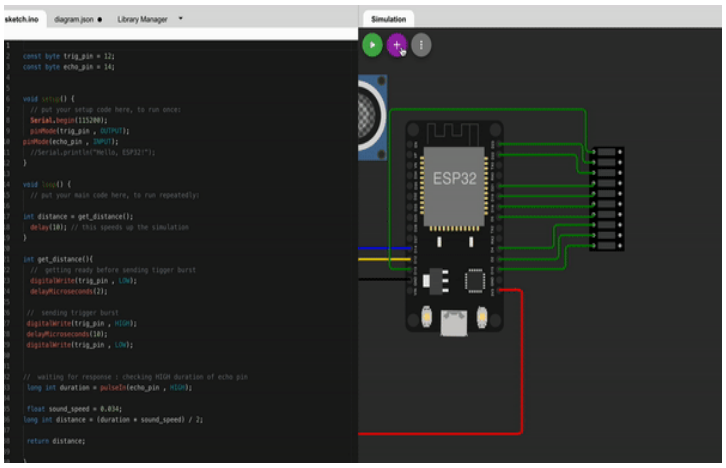


```
399
399
399
399
399
399
399
```

i. We add LED Bar graph and make its connections:

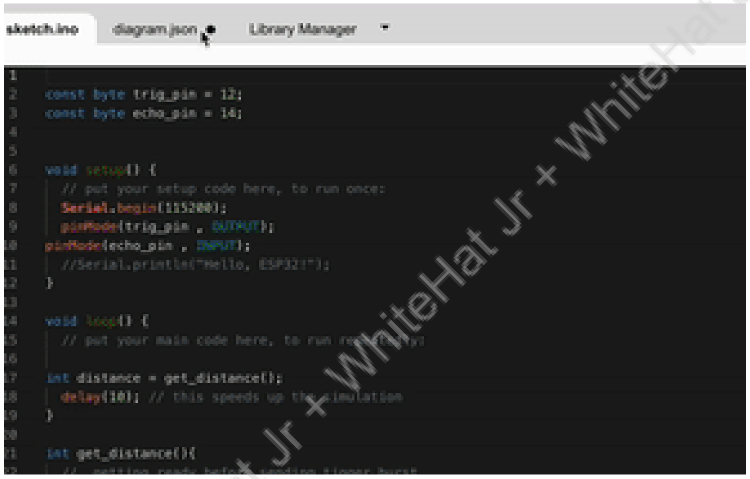


j. Also, add Resistor and make its connection:



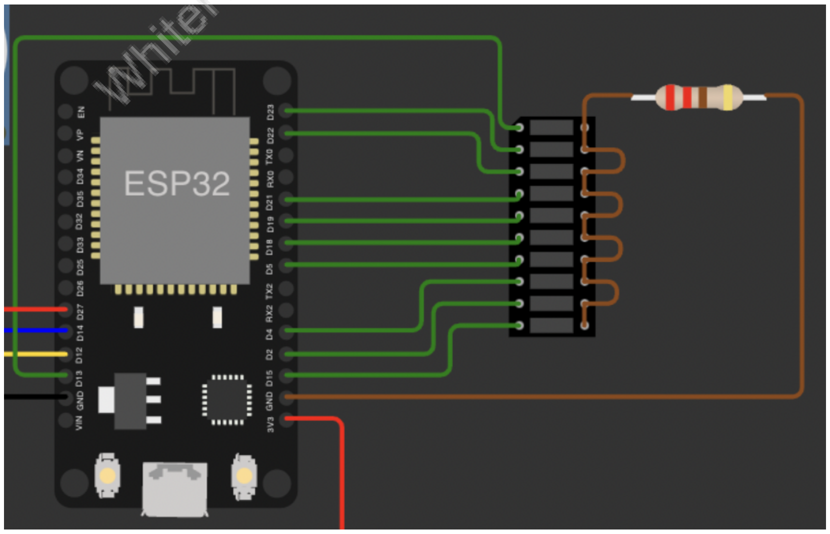
```
const byte trig_pin = 12;
const byte echo_pin = 14;


void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(trig_pin , OUTPUT);
pinMode(echo_pin , INPUT);
  //Serial.println("Hello, ESP32!");
}

void loop() {
  // put your main code here, to run repeatedly:

int distance = get_distance();
  delay(10); // this speeds up the simulation
}

int get_distance(){
  //  getting ready before sending tigger burst
  digitalWrite(trig_pin , LOW);
  delayMicroseconds(2);

 //  sending trigger burst
 digitalWrite(trig_pin , HIGH);
 delayMicroseconds(10);
 digitalWrite(trig_pin , LOW);


//  waiting for response : checking HIGH duration of echo pin
 long int duration = pulseIn(echo_pin , HIGH);

 float sound_speed = 0.034;
long int distance = (duration * sound_speed) / 2;

 return distance;

}
```

k. Update Resistor value:



```
const byte trig_pin = 12;
const byte echo_pin = 14;


void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(trig_pin , OUTPUT);
pinMode(echo_pin , INPUT);
  //Serial.println("Hello, ESP32!");
}

void loop() {
  // put your main code here, to run repeatedly:

int distance = get_distance();
  delay(10); // this speeds up the simulation
}

int get_distance(){
  //  getting ready before sending tigger burst
```

Final image for reference:



l. Declare an array of led_pins to register the pin numbers.

```
// array of led_pins
byte led_pins[] = {13, 15, 2, 4, 5, 18, 19, 21, 22, 23};
```

Note: The numbers correspond to the GPIO pins they are connected to from bottom to top approach.

m. We assign the pin mode for each of these LED pins. As we need to for each of them, we use loops.

```
// declaring all led pins as output
 for (int i = 0; i < sizeof(led_pins); i++){
  pinMode(led_pins[i] , OUTPUT);
 }
```

n. Next, we check if these LED pins are working or not. So, we create a function to do so and call it in loop().

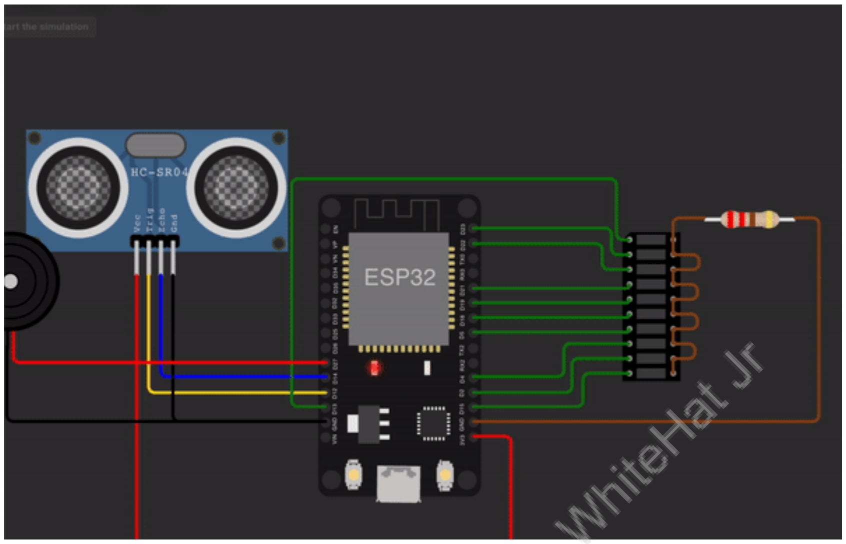
```
indicator_testing();
```

o. We define this method.

Starting from bottom to top, we will turn it ON and OFF one by one. So, the first loop will turn it ON and the next loop will turn it OFF. But turning OFF will happen in reverse order i.e. top to bottom.

```
void indicator_testing(){
 // turning all leds on
 for (int i = 0; i < sizeof(led_pins); i++){
   digitalWrite(led_pins[i] , HIGH);
   delay(70);
 }
 // all leds off
 for (int i = sizeof(led_pins) - 1; i >= 0; i--){
   digitalWrite(led_pins[i] , LOW);
   delay(70);
 }
}
```

Reference output:



p. Create a function to turn on the desired number of LEDs.
It accepts an integer/byte as an input for the number of LEDs.
Now, depending on the led_count, we will run a loop to start lighting the LEDs from bottom.
And also at the same time, we make sure the other LEDs are turned OFF in case they were turned on previously based on the distance.

```
void led_on(byte led_count){
 // total led which are going to be on
 for (int i = 0; i < led_count; i++){
  digitalWrite(led_pins[i] , HIGH);
 }

 // total leds which are going to be off
 for (int i = led_count; i < sizeof(led_pins); i++){
  digitalWrite(led_pins[i] , LOW);
 }
}
```

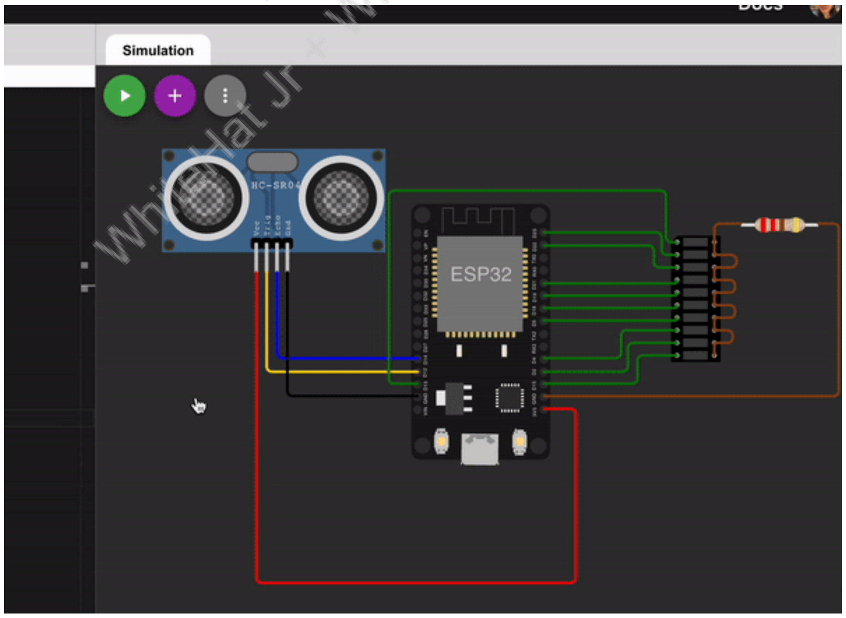
q. Next, we create a function indicator() that will check the distance using conditionals and turn on the desired number of LEDs.

```
void indicator(int distance){
    if (distance <= 150 && distance > 135)led_on(1);
    else if (distance <= 135 && distance > 120)led_on(2);
    else if (distance <= 120 && distance > 105)led_on(3);
    else if (distance <= 105 && distance > 90)led_on(4);
    else if (distance <= 90 && distance > 75)led_on(5);
    else if (distance <= 75 && distance > 60)led_on(6);
    else if (distance <= 60 && distance > 45)led_on(7);
    else if (distance <= 45 && distance > 30)led_on(8);
    else if (distance <= 30 && distance > 15)led_on(9);
    else if (distance <= 15 && distance > 0)led_on(10);
    else led_on(0);
  }
```

r. We call this indicator() function in the loop() after calculating the distance.

```
indicator(distance);
```

s. We add a new component(buzzer) and make its connections.



t. Also, add code for pin configuration at the correct places.

```
int buzzerPin = 26;
```

```
pinMode(buzzerPin, OUTPUT);
```

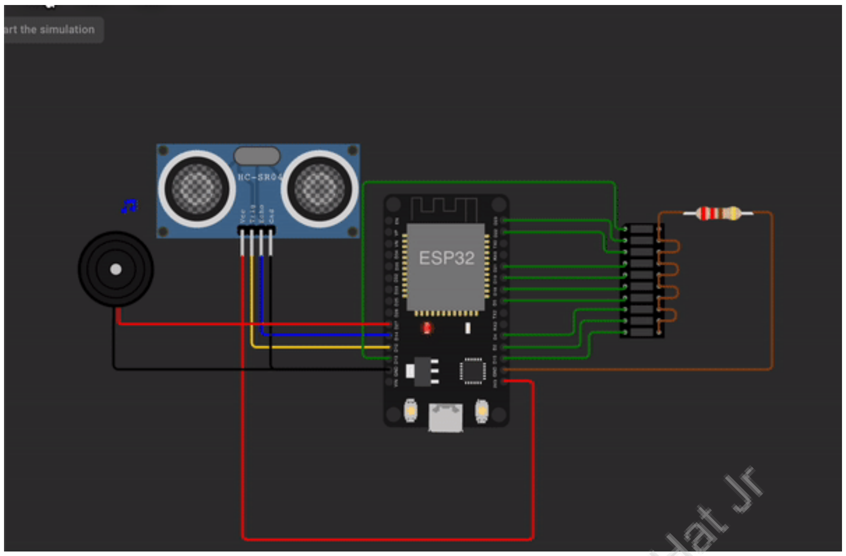
u. Create a function to buzz the sound:

```
void buzzer(){
 for (int j = 0; j < 3; j++)
  {
    digitalWrite(buzzerPin, HIGH);
    delay(2);
    digitalWrite(buzzerPin, LOW);
    delay(2);
  }
}
```

v. In **indicator()** method, we add the below code to play the buzzer if the distance is less than 150cm.
Also, we have added the buzzer interval. buzzer_interval is the time gap between the beeps. It should be less when the gap between the vehicle and the obstacle is less thereby beeping very fast to indicate to the user. Hence it is mapped as per the distance using map instruction. The distance from 0 to 150 cm is mapped between 0 to 1000 milliseconds.

```
int buzzer_interval = 0;
if(distance <= 150){
  //  calculating buzzer interval, mapping with distance
  buzzer_interval = map(distance , 0 , 150 , 0 , 1000);
  buzzer();
  delay(buzzer_interval);
 }
```

Reference Output:



What's NEXT?

In the **next class**, we will learn about a new microcontroller Arduino and also about serial communication.

Expand Your Knowledge

Read more about the working of HC-SR04 here and its application here

Read more about LED Bar graphs here and Resistor here.