KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRICAL/ELECTRONICS ENGINEERING

COE 381 - MICRPROCESSORS

GROUP 18

SMART TRAFFIC LIGHT SYSTEM BASED ON REAL TIME TRAFFIC DENSITY

1.0 INTRODUCTION

Reduction of traffic congestion has over the years been of great consideration to civil engineers when designing and building roads. They have resorted to using various techniques like lane management, interchange and intersection design, traffic signal coordination to ensure smooth flow of cars. As electrical engineers we can also contribute to traffic management by designing self-efficient systems to manage and coordinate traffic based on real time traffic density data to effectively improve traffic flow, safety and overall reduce waiting time during events of traffic.

This project is solely based on developing a smart traffic light system that not only frequently adjust the traffic lightning in various lanes to allow and disallow cars in and out of an intersection but uses the amount of cars in various lanes as a parameter for adjusting the traffic lightning .This parameter (amount of cars) we used is what we used to identify and determine whether the traffic density in a particular lane is high or low.

2.0 SYSTEM DESIGN AND DESCRIPTION

Components used for the project include

Hardware

- An ultrasonic sensor
- An Arduino mega board
- Traffic light display

- Resistors

Software

The main programming language was a C++.

REASONS FOR HARDAWARE COMPONENTS CHOSEN

- THE ARDUINO MEGA BOARD

The Arduino Mega was later chosen for the project as the initial Arduino UNO board had some limitations to our system design

The advantages of the mega board that led to its selection was

- More I\O pins (allows for interfacing with more hardware components)
- More memory (enables storage and complex programs)
- More serial ports (allows communication with multiple devices simultaneously)

- THE ULTRASONIC SENSOR

The ultrasonic sensor basically works on the principle of radio detection and ranging

- It emits a high frequency radio signal
- The radio waves bounce of target objects
- The sensor receives the and calculates the distance using flight time of the emitted wave

This gives the system the chance to be able to sense the density of cars within a lane utilizing the proximity profile.

For simplicity of the project only 4 sensors was used as more sensors would increase the complexity of the project.

- THE TRAFFIC LIGHT DISPLAY USING LEDS

This outputs a command encoded in a colour code all drivers and pedestrian understand

- THE RESISTOR

This supply required voltages to the various components connected to the micro controller

- **THE ALGORITHM USED FOR CONTROLLING THE TRAFFIC LIGHTNING**

Since the system is to manage traffic based on traffic density we resorted to using the amount of cars in the lane as a parameter to detect low or high density.

A range of distance was used to prompt the system of the presence or absence of a car

The less the total amount of reflected wave (longer the distance) means the less amount of cars and vice versa .This data is received by the micro controller for processing.

The traffic light begins it initialization from the lane with a higher reflection profile then proceeds to other lanes in descending order.

To ensure one lane doesn’t stay on green forever a maximum green light time was used to prevent this occurrence such that when that time elapses irrespective of the cars in the lane the traffic signal flow proceeds to the next lane .

□ SOFTWARE DESCRIPTION

The software component of our system is a C++ code for giving the micro controller command on the system processes.

Basically the code was to control how the micro controller analyzes and interprets the data fed to it. This enables the micro controller to generate an actuating signal that controls the traffic light (LEDS).

A detailed DESCRIPTION of the logic behind the code would be in our video presentation.

Below is the code we used to achieve the entire system operation

1. Defining pins

```
// Smart Traffic Light System with Dynamic Congestion Control

const int trigPins[4] = {8, 10, 12, A0};
const int echoPins[4] = {9, 11, 13,A1};
const int redPins[4] = {2, 5, A2, A7};
const int yellowPins[4] = {3, 6, A3, A6};
const int greenPins[4] = {4, 7, A4, A5};
```

These arrays define **four lanes**, each with:

- **Trigger and Echo pins** for ultrasonic sensors.
- **Red, Yellow, and Green light pins** for controlling the traffic signals.

The arrays make it easier to loop through the lanes without hard-coding each pin

Timing and Detection Parameters

```

const int greenTimeBase = 5000; // Minimum green time (5s)
const int extraTimePerCar = 500; // Additional time per car (0.5s)
const int maxGreenTime = 20000; // Max green time (20s)
const int detectionThreshold = 50; // Vehicle detection range (cm)
const int cycleTime = 30000; // 30s cycle time
```

These constants define how the traffic lights behave:

- **Green Time Base:** Minimum time (5 seconds) that a green light stays on.
- **Extra Time Per Car:** Extra green light time (0.5 seconds) per detected vehicle.
- **Max Green Time:** Maximum green light duration (20 seconds).

- **Detection Threshold:** Maximum distance (50 cm) for vehicle detection.
- **Cycle Time:** Total time (30 seconds) for one cycle of the traffic lights.

Variables for Traffic Counting

```
14
15   int vehicleCounts[4] = {0, 0, 0, 0};
16   bool prevDetection[4] = {false, false, false, false};
17   unsigned long lastCheckTime = 0;
```

- **vehicleCounts:** Keeps track of the number of cars detected per lane.
- **prevDetection:** Tracks whether a vehicle was detected in the previous cycle.
- **lastCheckTime:** Records the last time the system checked vehicle counts.

2. Setup Fucntion

```
18
19   void setup() {
20       Serial.begin(9600);
21       for (int i = 0; i < 4; i++) {
22           pinMode(trigPins[i], OUTPUT);
23           pinMode(echoPins[i], INPUT);
24           pinMode(redPins[i], OUTPUT);
25           pinMode(yellowPins[i], OUTPUT);
26           pinMode(greenPins[i], OUTPUT);
27           digitalWrite(redPins[i], HIGH); // Start with all red
28       }
29       lastCheckTime = millis();
30   }
```

Serial Communication: Starts communication at 9600 baud for debugging.

Pin Configuration: Sets the pins as input or output as needed.

Initialize Lights: Sets all lights to **red** at startup to ensure safety.

3. Distance Measurement Function

```
31
32  long getDistance(int trigPin, int echoPin) {
33      digitalWrite(trigPin, LOW);
34      delayMicroseconds(2);
35      digitalWrite(trigPin, HIGH);
36      delayMicroseconds(10);
37      digitalWrite(trigPin, LOW);
38      long duration = pulseIn(echoPin, HIGH, 30000);
39      return (duration == 0) ? 9999 : duration * 0.034 / 2;
40  }
```

Purpose: Measures the distance from the ultrasonic sensor to the nearest object (vehicle).

How it Works:

- Sends a **sound pulse** and measures the time it takes to bounce back.
- Converts the time to distance using the speed of sound.

4. Vehicle Counting

```
41
42  void updateVehicleCounts() {
43      for (int i = 0; i < 4; i++) {
44          bool currentDetection = (getDistance(trigPins[i], echoPins[i]) <= detectionThreshold);
45          if (currentDetection && !prevDetection[i]) {
46              vehicleCounts[i]++;
47          }
48          prevDetection[i] = currentDetection;
49      }
50  }
```

- **Purpose:** Updates the count of detected vehicles for each lane.
- **Logic:**
  - Checks if a vehicle is detected within the threshold distance.
  - Increments the count only if it was **not detected** previously (to avoid double-counting).

5. Display Vehicle Counts

```
51
52  void displayVehicleCounts() {
53      Serial.println("\nVehicle Counts:");
54      for (int i = 0; i < 4; i++) {
55          Serial.print("Lane "); Serial.print(i + 1);
56          Serial.print(": "); Serial.println(vehicleCounts[i]);
57      }
58  }
```

- **Purpose:** Prints the current vehicle counts to the **serial monitor** for debugging and monitoring.

6. **Determining the Most Congested Lanes**

```
59
60  void getMostCongestedLanes(bool lanes[4]) {
61      int maxCount = -1;
62      for (int i = 0; i < 4; i++) {
63          if (vehicleCounts[i] > maxCount) {
64              maxCount = vehicleCounts[i];
65          }
66      }
67      for (int i = 0; i < 4; i++) {
68          lanes[i] = (vehicleCounts[i] == maxCount && maxCount > 0);
69      }
70  }
```

- **Purpose:** Identifies lanes with the **highest congestion** to prioritize green light time.
- **Logic:** Finds the **maximum count** and marks lanes with the same count as congested.

7. **Resetting Traffic Lights**

```
71
72  void resetTrafficLights() {
73      for (int i = 0; i < 4; i++) {
74          digitalWrite(redPins[i], LOW);
75          digitalWrite(yellowPins[i], LOW);
76          digitalWrite(greenPins[i], LOW);
77      }
78  }
```

Purpose: Turns off all the traffic lights before making a new decision.

8. **Controlling the Traffic Lights**

```
79
80  void controlTraffic() {
81      bool lanes[4] = {false, false, false, false};
82      getMostCongestedLanes(lanes);
83
84      bool anyLaneActive = false;
85      for (int i = 0; i < 4; i++) {
86          if (lanes[i]) {
87              anyLaneActive = true;
88              int greenTime = min(greenTimeBase + vehicleCounts[i] * extraTimePerCar, maxGreenTime);
89              resetTrafficLights();
90              digitalWrite(redPins[i], LOW);
91              digitalWrite(greenPins[i], HIGH);
92              delay(greenTime);
93              digitalWrite(greenPins[i], LOW);
94              digitalWrite(yellowPins[i], HIGH);
95              delay(3000);
96              digitalWrite(yellowPins[i], LOW);
97              digitalWrite(redPins[i], HIGH);
98          }
99      }
100     if (!anyLaneActive) {
101         Serial.println("No vehicles detected. Turning off all lights.");
102         resetTrafficLights();
103     }
104 }
```

- **Purpose:** Controls the lights to give priority to the most congested lane.
- **Logic:**
    - Calculates **green light duration** based on congestion.
    - Manages **transitions** from green to yellow and then to red.
    - If no vehicles are detected, all lights are turned off.

9. **Main Loop**

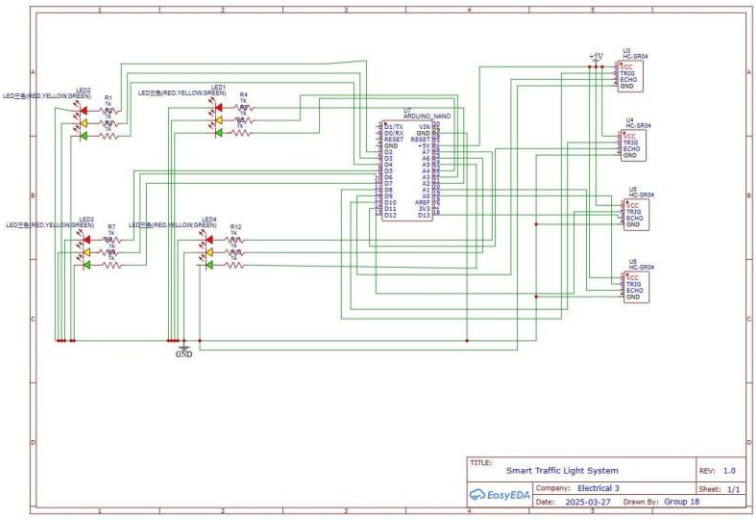
```
105
106 void loop() {
107     updateVehicleCounts();
108     if (millis() - lastCheckTime >= cycleTime) {
109         displayVehicleCounts();
110         controlTraffic();
111         memset(vehicleCounts, 0, sizeof(vehicleCounts));
112         lastCheckTime = millis();
113     }
114 }
115
```

- **Purpose:** Continuously monitors the lanes and manages the light changes.
- **Timing Control:** Uses the millis() function to manage the timing of operations.

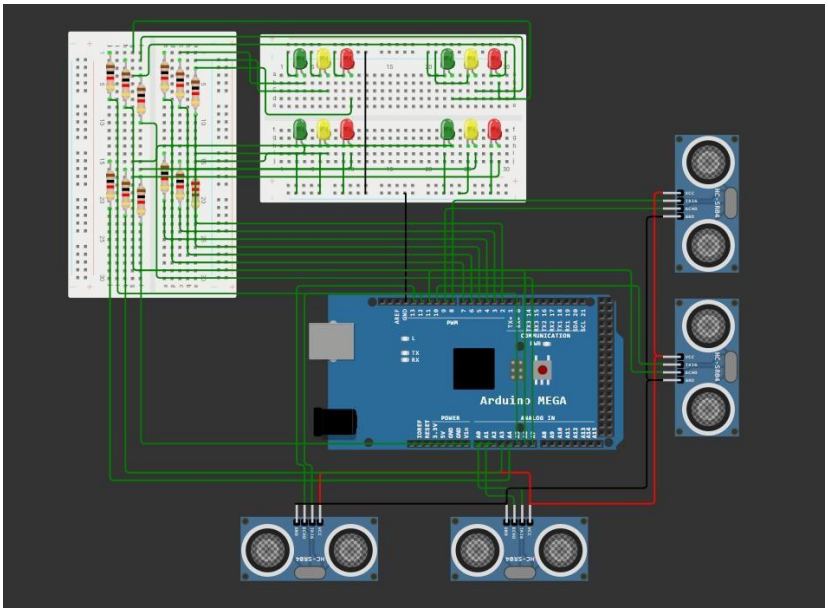
THE SYSTEM OPERATION

Basically the system operation can be stated as follows

- Data collection by the sensors
- Data processsing by the microcontrollers
- execution via the program command on the micro controller
- Visual display of the execution buy the LEDS which conveys a message that road users comprehend
- PICTURE OF SHEMTATIC



PICTURE OF SIMULATION



3.0 CONCLUSION

Through our research for the project we discovered that the concept of a smart traffic lightning system is an area in traffic management is an area that is gaining and making major breakthroughs through the usage of complex algorithms for traffic density detection.

Our algorithm based on ranging of the moving cars is one of the basic one of its kind and prone to limitations.

This system has much more room for potential improvement in areas of being able to accurately

determine the presence or absence of a vehicle since pedestrians in close proximity would be added as car count per our system design , the type of vehicle be it an emergency vehicle in a lane and further prioritizing the lane irrespective of the traffic density in the various lanes