REAL-TIME EMBEDDED SYSTEM
EXPERIMENT NO. 2

REAL-TIME TRAFFIC LIGHT

Name: NAVARRO, ROD GERYK C.

Course/Section: CPE161P-4/C1

Group No.: N/A

Date of Performance: 12/12/2024
Date of Submission: 01/12/2025

CYREL O. MANLISES, PH.D.
Instructor

DISCUSSION

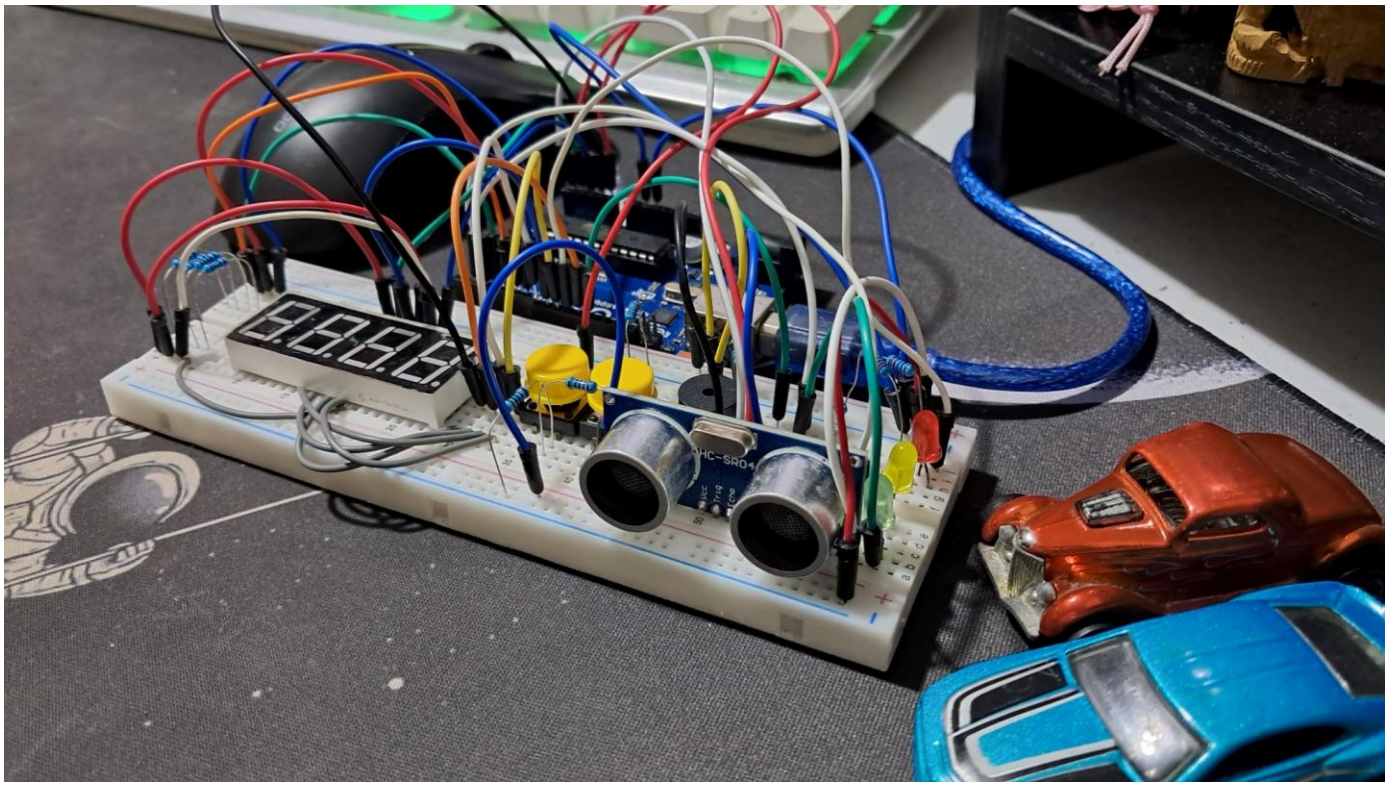

Figure 1: *The Connection of the 4-digit SSD, Buzzer, Buttons, Ultrasonic Sensor, and LEDs.*

In the experiment, I used a 4-digit seven-segment display (SSD), buttons, a buzzer, an ultrasonic sensor, and LEDs. The 4-digit SSD served as the display timer for both the stop signal timer and the go signal timer, with a maximum of 60 seconds for each timer. The first two digits on the left side of the SSD are for the stop timer, and the last two digits are for the go signal.

To start, I built the circuit using two buttons and a voltage divider made of 220-ohm resistors connected to one analog input. The first button is used to subtract 15 seconds from the normal counting of the stop signal before the go signal, while the second button is used to add 30 seconds to the normal counting of the go signal before the stop signal.

I also added an ultrasonic sensor for detecting cars that beat the red signal. When a car is detected, a buzzer sounds an alarm to notify people that someone has violated the stop signal. Three LEDs are used in the experiment: red for indicating the stop signal, yellow for indicating the ready signal, and green for indicating the go signal.

Navarro_EXP2_CPE161P.ino

```
1
2   const int sPins[7] = {A3,A2, A1,12,8,9,10};
3   const int dPins[4] = {4,5,6,7};
4   const int buzzerAlarm = 13;
5   const int ledGreen = 2;
6   const int ledYellow = 3;
7   const int ledRed = 11;
8   const int triggerPin = A5;
9   const int echoPin = A4;
10
11  const int switchPins = A0;
12  int timeGo = 60;
13  int timeStop = 0;
14  int maxCountdown = 60;
15  bool settingStop = true;
16  bool isStopTimerRunning = true;
17
18  unsigned long previousMillis = 0;
19
20  const long interval  = 1000;
21  bool timerRunning = true;
22  int messageIndex = 0;
23  int cm = 0;
24
25  byte digitCodes[10]{
26    B00111111, B00000110, B01011011, B01001111,
27    B01100110, B01101101, B01111101, B00000111,
28    B01111111, B01101111
29  };
30
31
```

Figure 2: The Initialization of Variables and Components.

The initialization section of the code sets up all the constants, variables, and arrays needed for the experiment. The const int variables define the pins used for different components like the seven-segment display, buzzer, LEDs, ultrasonic sensor, and buttons. This ensures that each pin has a specific purpose, making the code easier to read and maintain. For example, sPins and dPins are arrays that group the segment and digit pins for the seven-segment display, allowing efficient control of the display. Other variables like timeGo and timeStop store the countdown timers, while maxCountdown defines the upper limit of the timer. Flags like settingStop and isStopTimerRunning manage the timer's current state.

The initialization also includes an array called digitCodes, which stores binary values for each digit (0 9) to control the segments of the display. These codes help illuminate the correct segments for displaying numbers. By defining these at the beginning, the code becomes modular and easier to debug if changes are needed. Overall, this section is crucial for ensuring all components are correctly identified and that the timers and states are properly initialized before the program runs.

Navarro_EXP2_CPE161P.ino

```
31
32
33  void setup() {
34    Serial.begin(9600);
35
36    pinMode(buzzerAlarm, OUTPUT);
37    pinMode(ledGreen, OUTPUT);
38    pinMode(ledYellow, OUTPUT);
39    pinMode(ledRed, OUTPUT);
40    pinMode(triggerPin, OUTPUT);
41    pinMode(echoPin, INPUT);
42
43    for(int i= 0; i<7; i++){
44      pinMode(sPins[i], OUTPUT);
45    }
46
47    for (int i=0; i<4; i++){
48      pinMode(dPins[i], OUTPUT);
49      digitalWrite(dPins[i], HIGH);
50
51    }
52
53    displayTimer(timeStop, timeGo);
54
55  }
56
```

Figure 3: The Setup of the Components.

The setup() function is where the hardware components are configured. It starts by initializing serial communication with Serial.begin(9600) for debugging purposes, allowing the user to monitor the program's behavior in real-time. Next, it sets the pin modes for all components. For example, the LEDs and buzzer are set as outputs because they will actively send signals, while the ultrasonic sensor's echo pin is set as an input to receive distance data.

A key part of the setup() function is the configuration of the seven-segment display. A loop iterates through the sPins and dPins arrays to set each pin as an output. The digitalWrite function ensures the digit pins are initialized to HIGH, turning off the display until it is needed. The displayTimer() function is also called here to show the initial timer values on the display. This ensures the hardware is ready and the user can see the starting state of the timers immediately.

```
void incrementTimer(){
 if (isStopTimerRunning == false) {
        timeGo += 30;
     } if (timeGo >= 60) {
        timeGo = 60;
   }
}

 void decrementTimer(){
 if (isStopTimerRunning == true) {
        timeStop -= 15;
     } if (timeStop <= 0) {
        timeStop = 0;
   }
}
```

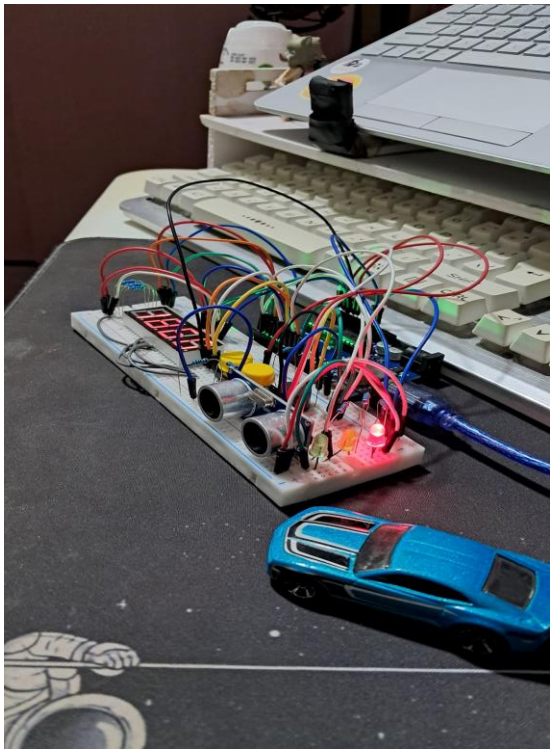

Figure 4: Fixing the Errors.

One issue encountered during the experiment was with the ultrasonic sensor. The sensor sometimes returned inaccurate readings, especially when no object was in range. This caused the buzzer to sound unexpectedly. The problem was due to insufficient filtering of the sensor's output, which sometimes registered noise as valid data. To fix this, a condition was added in the detectUltrasonic() function to check if the distance is less than or equal to 10 cm. This filter reduces false positives and ensures the buzzer only activates when an actual object is detected within the specified range.

Another issue occurred with the timers when the increment or decrement functions were called repeatedly in quick succession. This caused the timers to exceed their maximum or minimum values. To address this, bounds were added in the incrementTimer() and decrementTimer() functions. For example, timeGo is capped at 60 seconds, and timeStop cannot go below 0. These fixes ensure the program behaves predictably, even if buttons are pressed rapidly.

Navarro_EXP2_CPE161P.ino

```
56
57  void loop() {
58    int buttonValue = analogRead(switchPins);
59    Serial.print("Analog value: ");
60    Serial.print(buttonValue);
61
62    int allButtons = digitalRead(switchPins);
63
64    if (buttonValue < 5){
65      allButtons = LOW;
66    } else if (buttonValue < 25){
67      Serial.println("Button 1 Pressed");
68      decrementTimer();
69      delay(500);
70    } else{
71      incrementTimer();
72      Serial.println("Button 2 Pressed");
73      delay(500);
74    }
75
76      unsigned long currentMillis = millis();
77
78    if (currentMillis - previousMillis >= interval) {
79      previousMillis = currentMillis;
80
81      if (isStopTimerRunning) {
82
83        if (timeStop < 60) {
84         timeStop++;
85         detectUltrasonic();
86         digitalWrite(ledRed, HIGH);
87           if(timeStop >= 30){
88            digitalWrite(ledYellow, HIGH);
```

Figure 5: Loop Function.

The loop() function is the heart of the program, where all the real-time operations take place. It starts by reading the value of the analog pin connected to the buttons using analogRead(). Depending on the value, it determines which button was pressed and either increments or decrements the timers accordingly. A short delay is added after each button press to debounce the input and prevent multiple unintended actions.

The function also manages the timers and their transitions between the stop and go states. Using the millis() function, the program tracks elapsed time and updates the timers every second. When the stop timer finishes, it switches to the go timer, and vice versa. LEDs are used to indicate the current state: red for stop, green for go, and yellow as a transition indicator. The displayTimer() function continuously updates the seven-segment display to show the current timer values. This ensures that the system provides clear feedback to users while running smoothly in real time.

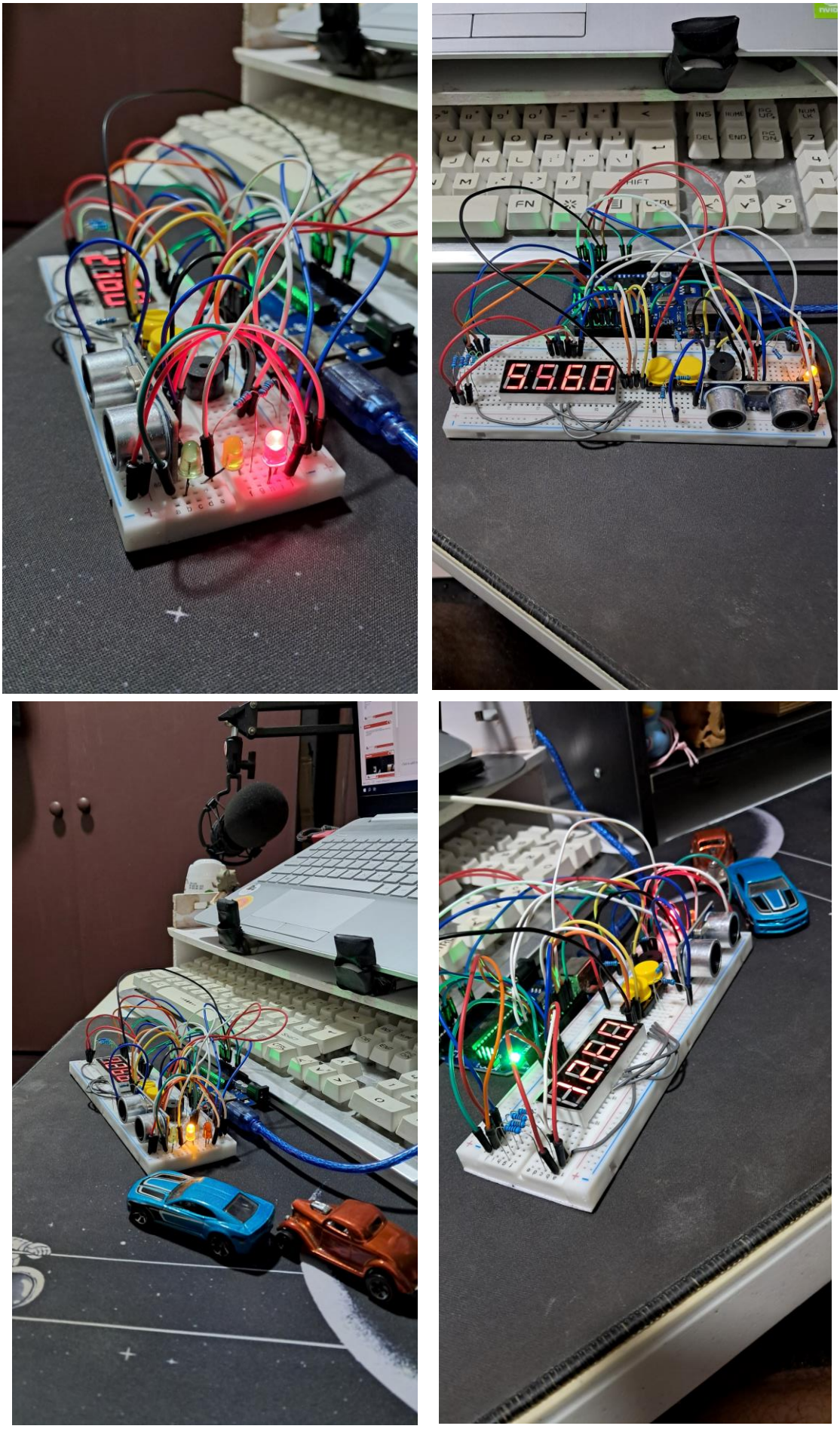

Figure 6: Working Prototype.

This is the final working prototype for this experiment, which successfully meets all the requirements.

CONCLUSION

The real-time traffic light experiment demonstrated the integration of various electronic components to simulate a functional traffic light system. The system used a 4-digit seven-segment display to show timers for stop and go signals, with buttons to adjust these timers. An ultrasonic sensor was included to detect vehicles that violated the red light, triggering a buzzer to alert nearby individuals. LEDs were used to visually represent the stop (red), ready (yellow), and go (green) signals, making the setup intuitive and practical. Through careful coding and circuit design, the experiment successfully managed the timers, states, and transitions between signals, providing a clear and interactive demonstration of real-time control systems.

Despite some challenges, such as inaccurate readings from the ultrasonic sensor and timer limits being exceeded during rapid button presses, these issues were resolved through coding adjustments. Filtering conditions were added to improve sensor accuracy, and boundaries were implemented to keep timers within acceptable limits. The experiment highlighted the importance of proper initialization, error handling, and real-time monitoring. The project was a success, demonstrating the potential of combining sensors, displays, and microcontrollers to create a responsive and reliable traffic light system.