PROJECT REPORT

CSE 360

COMPUTER INTERFACING

SPRING 2024

GROUP - 01

GROUP MEMBERS

NAME	ID
MD TARIQUL ISLAM	20301044
MOHAMMAD SHOWRAB HOSSN	20301081
ASHAKUZZAMAN ODREE	20301268

DRIVER DROWSINESS DETECTION AND ALERTING SYSTEM

Introduction; "Driver Drowsiness Detection & Alerting System" is a groundbreaking project aimed at enhancing road safety by employing cutting-edge technology. This innovative system utilizes Arduino, an open-source electronics platform, to monitor a driver's alertness level during long journeys or monotonous driving conditions. By integrating various sensors such as facial recognition cameras, heart rate monitors, and even steering wheel sensors, the system can accurately detect signs of drowsiness or fatigue in real-time.

Once the system identifies potential signs of driver fatigue, such as drooping eyelids or decreased heart rate variability, it triggers an immediate alert mechanism. This alert can take the form of auditory alarms, ensuring that the driver is promptly notified to regain focus or pull over for rest.

The significance of this project lies in its potential to prevent accidents caused by driver drowsiness, which is a leading factor in road accidents globally. By providing timely warnings to drivers, this system has the potential to save lives and reduce the risk of collisions on the road. Moreover, its Arduino-based design offers scalability, affordability, and accessibility, making it adaptable for implementation in various vehicles, from personal cars to commercial fleets. Overall, the Driver Drowsiness Detection & Alerting System represents a crucial step towards promoting safer and more responsible driving practices in today's fast-paced world.

APPLICATION AREA-

The "Driver Drowsiness Detection & Alerting System" serves the critical area of road safety by addressing one of the leading causes of accidents: Driver Fatigue. Here's how this project will make a significant impact in this area:

1. **Accident Prevention**: By continuously monitoring the driver's alertness level, the system can detect early signs of drowsiness or fatigue and issue timely alerts. This proactive approach helps prevent accidents caused by drivers falling asleep at the wheel or losing focus due to exhaustion.

2. **Real-time Monitoring**: The integration of sensors such as facial recognition cameras and heart rate monitors enables the system to monitor the driver's condition in real-time. This means that alerts are triggered immediately upon detecting any signs of drowsiness, providing prompt intervention before a potential accident occurs.

3. **Customizable Alerting Mechanisms**: The system offers various alerting mechanisms such as auditory alarms, visual cues, or haptic feedback, allowing for customization based on the driver's preferences or the vehicle's specifications. This versatility ensures that the alerts effectively capture the driver's attention without causing distraction.

4. **Universal Applicability**: The Arduino-based design of the system makes it adaptable for implementation in various vehicles, including personal cars, trucks, buses, and commercial fleets. Its scalability and affordability mean that it can be widely deployed, reaching a broad range of drivers and vehicles to maximize its impact on road safety.

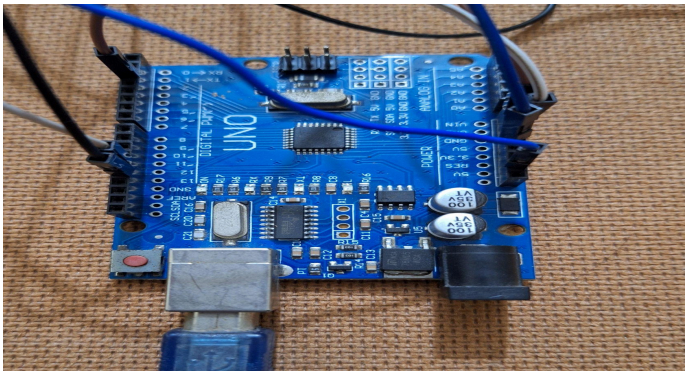
5. **Data Collection and Analysis**: The system can also collect valuable data on driver behavior and fatigue patterns, which can be analyzed to identify trends and potential risk factors. This data-driven approach enables policymakers and transportation authorities to develop targeted interventions and strategies for further improving road safety.

Overall, the "Driver Drowsiness Detection & Alerting System" serves the area of road safety by leveraging advanced technology to proactively address the risks

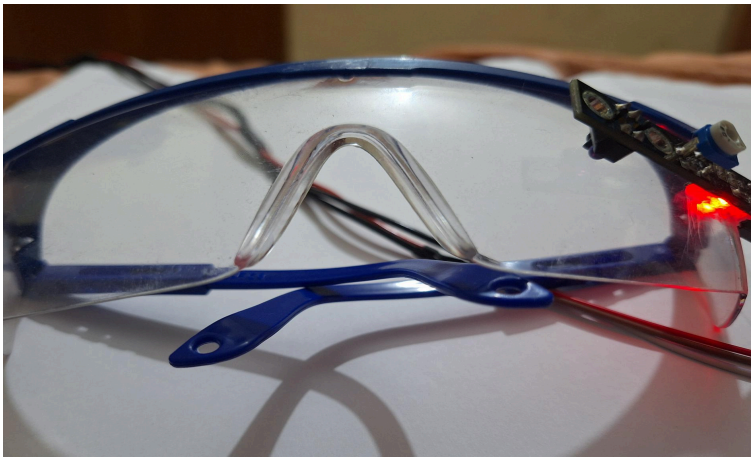
associated with driver fatigue, ultimately saving lives and preventing accidents on the road.

EQUIPMENT / TOOLS -

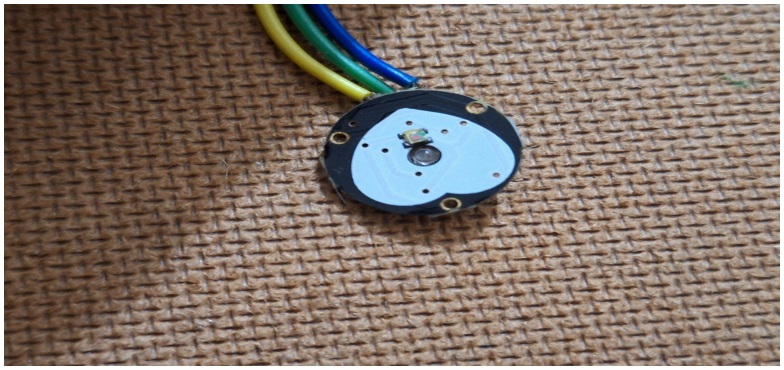
1. **Arduino Uno**: This is the main microcontroller board that serves as the brain of the system. It controls the operation of other components based on the input it receives from sensors and user interactions.



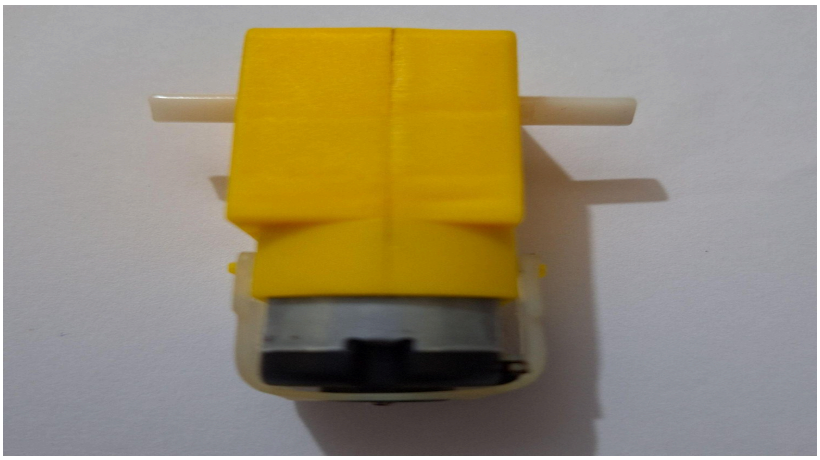
2. **Eyeblink Sensor**: This sensor detects eye blinks, which can be indicative of drowsiness or fatigue in the driver. It typically works by monitoring changes in light intensity caused by eyelid movements.



3. **Pulse Sensor**: This sensor measures the heart rate of the driver by detecting blood flow through capillaries in the fingertip. Changes in heart rate can indicate changes in alertness or stress levels.

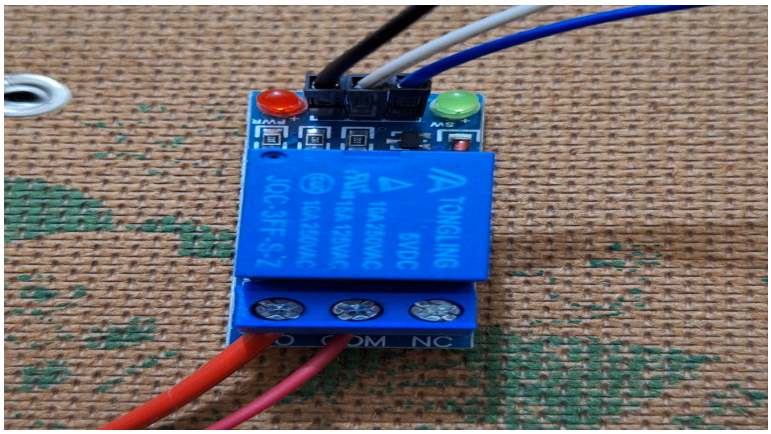


4. **DC Motor**: This component can be used to simulate vehicle movement or to create haptic feedback for the driver, such as vibration, to alert them of potential drowsiness.



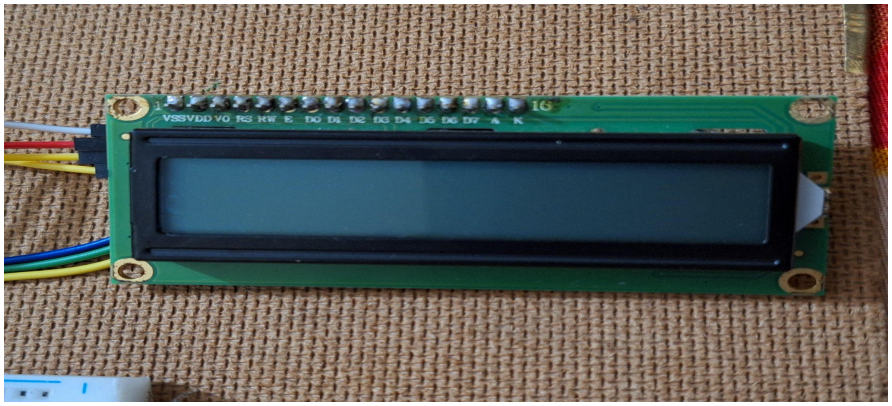
5. **Steering Wheel**: This can be a dummy steering wheel or a real one equipped with sensors to detect changes in steering behavior, which can indicate drowsiness or distraction.

6. **Relay Module**: This module is used to control high-voltage devices like lights or alarms using the low-voltage signals from the Arduino.

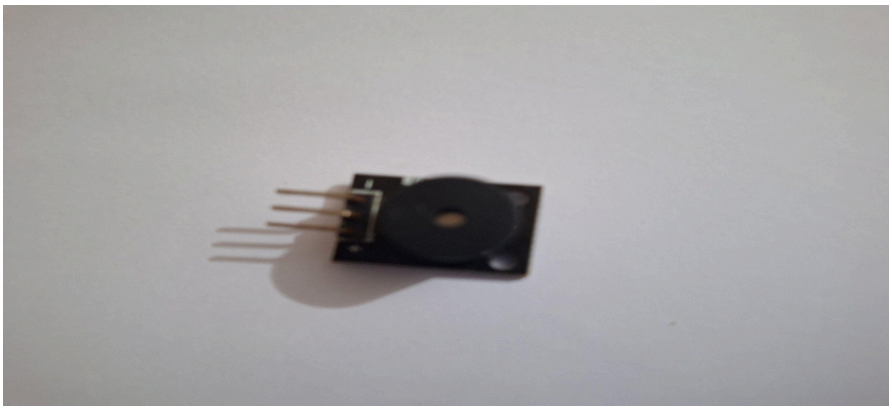


7. **9V Battery**: This provides power to the system, particularly useful for portable or standalone applications where a power outlet is not available.

8. **LCD DISPLAY**: This display provides visual feedback to the driver, such as alert messages or real-time sensor readings, enhancing their awareness of their condition.

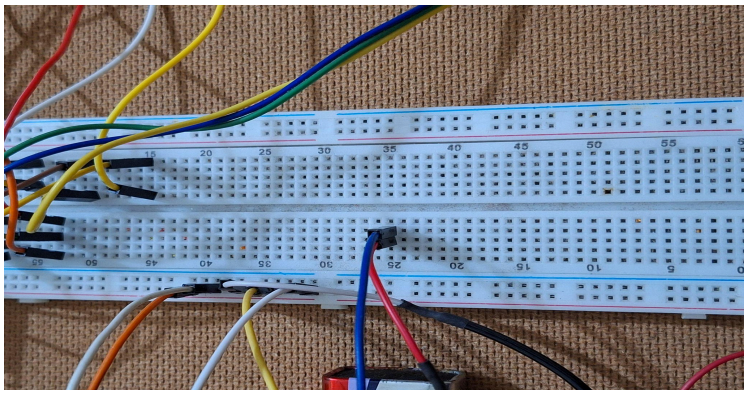


9. **Buzzer**: This component emits audible alerts to the driver when drowsiness or fatigue is detected, ensuring that they are alerted even if they are not looking at the display.

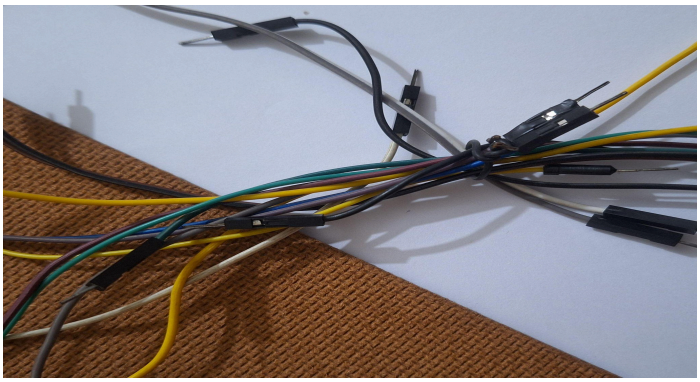


10. **Switch**: This is a simple on/off switch that can be used to control the system's power supply, allowing the user to turn the system on or off as needed.

11. **Breadboard**: This is a prototyping board used for building and testing electronic circuits without the need for soldering. It allows for easy connections and modifications during the development process.



12. **Jumper Wires**: These wires are used to make electrical connections between components on the breadboard or between the breadboard and the Arduino, facilitating the setup and configuration of the system.



The programming language we've used here is C.

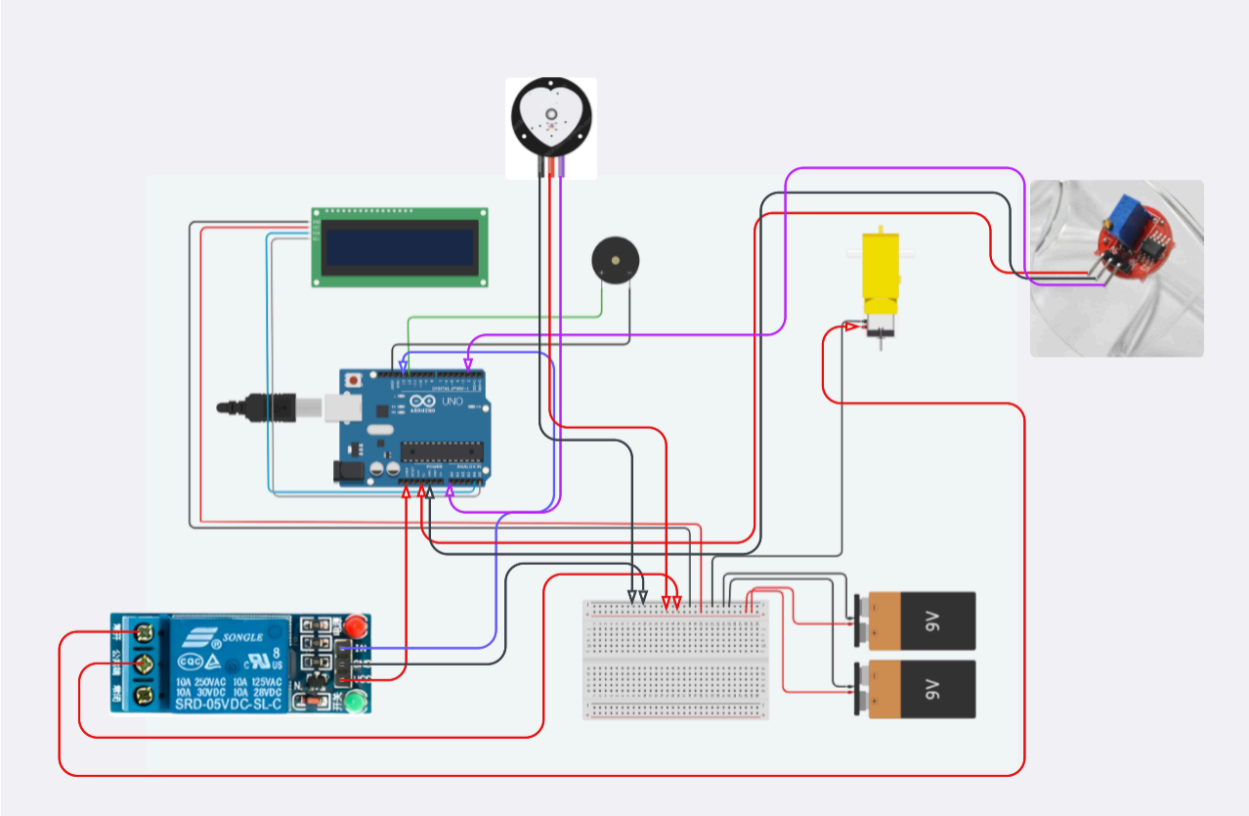
SENSORS

1. **Eyeblink Sensor:**
    - The Eyeblink Sensor typically consists of an infrared (IR) emitter and receiver pair.
    - The emitter emits infrared light towards the eye, and the receiver detects the reflected light.

- When the eyelid is open, the reflected light intensity detected by the receiver is relatively high.
- However, when the eyelid blinks or closes, the amount of reflected light decreases abruptly due to obstruction by the eyelid.
- This change in light intensity is detected by the sensor, and it can be interpreted as an eyeblink event.
- By monitoring the frequency and duration of these blinks, the sensor can infer the level of alertness or drowsiness of the individual.

2. **Pulse Sensor:**
    - The Pulse Sensor typically includes an infrared LED and a photodetector (photodiode) placed on the fingertip.
    - The infrared LED emits light into the fingertip, and the photodetector detects the amount of light that is transmitted or reflected back.
    - As blood pulses through the arteries in the fingertip, it absorbs or reflects varying amounts of infrared light.
    - These fluctuations in infrared light intensity detected by the photodetector correspond to changes in blood volume caused by the heartbeat.
    - The Pulse Sensor amplifies and filters these variations to isolate the pulsatile component, representing the heart rate.
    - By analyzing the time intervals between these pulsatile signals, the sensor can accurately measure the heart rate of the individual.

Circuit Diagram



Communication Protocols (I2C and UART)

In this driver drowsiness detection and alert system project, UART (Universal Asynchronous Receiver-Transmitter) and I2C (Inter-Integrated Circuit) protocols facilitate communication between sensors, devices, and interfacing ICs.

1. **UART Protocol:**
    - UART is employed for serial communication between the Arduino Uno board and the eyeblink sensor.
    - The Arduino sends commands to the eyeblink sensor via UART to initiate measurements and receive data.
    - Data packets containing eyeblink sensor readings, such as eye open/closed status and duration, are transmitted back to the Arduino Uno via UART.
    - This bidirectional data flow allows the Arduino to continuously monitor the eyeblink sensor's output and detect prolonged eye closures indicative of drowsiness.

2. **I2C Protocol:**
    - I2C is utilized for communication between the Arduino Uno board and the pulse sensor, as well as any other interfacing ICs.
    - The Arduino acts as the master device in the I2C bus, controlling communication with the pulse sensor and other connected devices.
    - Through I2C commands, the Arduino instructs the pulse sensor to initiate pulse rate measurements.
    - The pulse sensor responds by transmitting pulse rate data back to the Arduino over the I2C bus.
    - This data flow enables real-time monitoring of the driver's pulse rate, aiding in the detection of drowsiness based on physiological indicators.

3. **Interfacing ICs:**
    - Interfacing ICs, such as level shifters or multiplexers, may be employed to ensure compatibility and efficient communication between the Arduino Uno and the sensors/devices.
    - For example, level shifters may be used to adapt voltage levels between the Arduino Uno (which typically operates at 5V) and sensors/devices operating at different voltage levels.
    - Multiplexers may be utilized to expand the number of sensors/devices that can be connected to the Arduino Uno by enabling the sharing of communication lines.

So, UART and I2C protocols facilitate seamless communication between the Arduino Uno board, eyeblink sensor, pulse sensor, and any other interfacing ICs. This data flow enables the Arduino to monitor eyeblink patterns and pulse rate, detect drowsiness, and trigger appropriate alert mechanisms for enhanced driver safety.

Code:

```
const int sensorPin = 2;

const int motorPin = 8;

const int buzzerPin = 9;


long time; // Variable to store time
```

```
void setup() {
  pinMode(motorPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT); // Set buzzerPin as an OUTPUT
  digitalWrite(motorPin, HIGH); // Turn on the motor initially
  pinMode(sensorPin, INPUT);
}
void loop() {
   // Check if the IR sensor (sensorPin) is triggered (LOW means it's triggered)
  if (!digitalRead(sensorPin)) {
    time = millis(); // Record the current time
    // Keep checking if the IR sensor is still triggered
    while (!digitalRead(sensorPin)) {
       // When the IR sensor is triggered, turn off the buzzer and turn on the motor and wait for 1 second
       digitalWrite(buzzerPin, HIGH); // Turn on the buzzer when eyes are closed
      digitalWrite(motorPin, HIGH);
      delay(1000);
    }
  } else {
    // If the IR sensor is not triggered (eyes open)
    // Check the time elapsed since the sensor was last triggered
    if (TimeDelay() >= 3) {
       digitalWrite(buzzerPin, LOW); // Turn off the buzzer when eyes are open
      if (TimeDelay() >= 4) {
        digitalWrite(motorPin, LOW); // If 4 seconds have passed, turn off the motor
      }
    }
  }
}
```

```
int TimeDelay() {
   long t = millis() - time; // Calculate the time elapsed since the IR sensor was triggered
  t = t / 1000; // Convert milliseconds to seconds
  return t; // Return the elapsed time in seconds
}
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2); // to check your i2c address  upload i2c scanner to the board
int pulsePin = A0;                  // Pulse Sensor purple wire connected to analog pin A0
int blinkPin = 13;                // pin to blink led at each beat
// Volatile Variables, used in the interrupt service routine!
volatile int BPM;                   // int that holds raw Analog in 0. updated every 2mS
volatile int Signal;                // holds the incoming raw data
volatile int IBI = 600;             // int that holds the time interval between beats! Must be seeded!
volatile boolean Pulse = false;     // "True" when User's live heartbeat is detected. "False" when not a "live beat".
volatile boolean QS = false;        // becomes true when Arduoino finds a beat.
static boolean serialVisual = true;   // Set to 'false' by Default. Re-set to 'true' to see Arduino Serial Monitor ASCII Visual Pulse
volatile int rate[10];                    // array to hold last ten IBI values
volatile unsigned long sampleCounter = 0;          // used to determine pulse timing
volatile unsigned long lastBeatTime = 0;           // used to find IBI
volatile int P = 512;                     // used to find peak in pulse wave, seeded
volatile int T = 512;                     // used to find trough in pulse wave, seeded
volatile int thresh = 525;                // used to find instant moment of heart beat, seeded
volatile int amp = 100;                   // used to hold amplitude of pulse waveform, seeded
```

```
volatile boolean firstBeat = true;        // used to seed rate array so we
startup with reasonable BPM

volatile boolean secondBeat = false;      // used to seed rate array so we
startup with reasonable BPM


void setup()

{

   pinMode(blinkPin,OUTPUT);              // pin that will blink to your
heartbeat!

  Serial.begin(115200);

  // we agree to talk fast!

  interruptSetup();                 // sets up to read Pulse Sensor signal
every 2mS

                                         // IF YOU ARE POWERING The Pulse
Sensor AT VOLTAGE LESS THAN THE BOARD VOLTAGE,

                                      // UN-COMMENT THE NEXT LINE AND APPLY
THAT VOLTAGE TO THE A-REF PIN

                                     //   analogReference(EXTERNAL);

  lcd.init();//if you can not upload this code lcd.init(); and change to
lcd.begin();

  lcd.backlight();

}

//  Where the Magic Happens

void loop()

{

   serialOutput();


  if (QS == true) // A Heartbeat Was Found

    {

      // BPM and IBI have been Determined

      // Quantified Self "QS" true when arduino finds a heartbeat

        serialOutputWhenBeatHappens(); // A Beat Happened, Output that to
serial.

      QS = false; // reset the Quantified Self flag for next time

    }
```

```
  delay(20); //  take a break
}
void interruptSetup()
{
  // Initializes Timer2 to throw an interrupt every 2mS.
  TCCR2A = 0x02;      // DISABLE PWM ON DIGITAL PINS 3 AND 11, AND GO INTO CTC MODE
  TCCR2B = 0x06;      // DON'T FORCE COMPARE, 256 PRESCALER
  OCR2A = 0X7C;       // SET THE TOP OF THE COUNT TO 124 FOR 500Hz SAMPLE RATE
  TIMSK2 = 0x02;      // ENABLE INTERRUPT ON MATCH BETWEEN TIMER2 AND OCR2A
  sei();              // MAKE SURE GLOBAL INTERRUPTS ARE ENABLED
}
void serialOutput()
{   // Decide How To Output Serial.
 if (serialVisual == true)
  {
      arduinoSerialMonitorVisual('-', Signal);   // goes to function that makes Serial Monitor Visualizer
  }
 else
  {
        sendDataToSerial('S', Signal);       // goes to sendDataToSerial function
   }
}
void serialOutputWhenBeatHappens()
{
 if (serialVisual == true) //  Code to Make the Serial Monitor Visualizer Work
   {
     Serial.print(" Heart-Beat Found ");  //ASCII Art Madness
     Serial.print("BPM: ");
     Serial.println(BPM);
```

```
      lcd.print("Heart-Beat Found ");
      lcd.setCursor(1,1);
      lcd.print("BPM: ");
      lcd.setCursor(5,1);
      lcd.print(BPM);
      delay(300);

      lcd.clear();

    }
 else
    {
      sendDataToSerial('B',BPM);   // send heart rate with a 'B' prefix
       sendDataToSerial('Q',IBI);    // send time between beats with a 'Q'
prefix
    }
}
void arduinoSerialMonitorVisual(char symbol, int data )
{
   const int sensorMin = 0;         // sensor minimum, discovered through
experiment
   const int sensorMax = 1024;      // sensor maximum, discovered through
experiment
   int sensorReading = data; // map the sensor range to a range of 12
options:
  int range = map(sensorReading, sensorMin, sensorMax, 0, 11);
  // do something different depending on the
  // range value:
}
void sendDataToSerial(char symbol, int data )
{
   Serial.print(symbol);
   Serial.println(data);
}
```

```
ISR(TIMER2_COMPA_vect) //triggered when Timer2 counts to 124
{
  cli();                                      // disable interrupts while
we do this
  Signal = analogRead(pulsePin);              // read the Pulse Sensor
  sampleCounter += 2;                         // keep track of the time in
mS with this variable
  int N = sampleCounter - lastBeatTime;       // monitor the time since
the last beat to avoid noise
                                              //  find the peak and trough
of the pulse wave
  if(Signal < thresh && N > (IBI/5)*3) // avoid dichrotic noise by waiting
3/5 of last IBI
    {
      if (Signal < T) // T is the trough
      {
        T = Signal; // keep track of lowest point in pulse wave
      }
    }

  if(Signal > thresh && Signal > P)
    {          // thresh condition helps avoid noise
      P = Signal;                             // P is the peak
    }                                             // keep track of highest
point in pulse wave

  //  NOW IT'S TIME TO LOOK FOR THE HEART BEAT
  // signal surges up in value every time there is a pulse
  if (N > 250)
  {                                   // avoid high frequency noise
    if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)*3) )
      {
        Pulse = true;                               // set the Pulse flag
when we think there is a pulse
        digitalWrite(blinkPin,HIGH);                // turn on pin 13 LED
```

```
          IBI = sampleCounter - lastBeatTime;           // measure time
between beats in mS
         lastBeatTime = sampleCounter;               // keep track of time
for next pulse


        if(secondBeat)
          {                                 // if this is the second beat, if
secondBeat == TRUE
          secondBeat = false;                   // clear secondBeat flag
             for(int i=0; i<=9; i++) // seed the running total to get a
realisitic BPM at startup
          {
            rate[i] = IBI;
          }
        }
           if(firstBeat) // if it's the first time we found a beat, if
firstBeat == TRUE
        {
          firstBeat = false;                    // clear firstBeat flag
          secondBeat = true;                    // set the second beat flag
          sei();                                // enable interrupts again
           return;                               // IBI value is unreliable
so discard it
        }
      // keep a running total of the last 10 IBI values
       word runningTotal = 0;                     // clear the runningTotal
variable
      for(int i=0; i<=8; i++)
        {                // shift data in the rate array
          rate[i] = rate[i+1];                  // and drop the oldest IBI
value
          runningTotal += rate[i];              // add up the 9 oldest IBI
values
        }
```

```
      rate[9] = IBI;                          // add the latest IBI to the
rate array
       runningTotal += rate[9];                // add the latest IBI to
runningTotal
       runningTotal /= 10;                     // average the last 10 IBI
values
       BPM = 60000/runningTotal;               // how many beats can fit
into a minute? that's BPM!
      QS = true;                              // set Quantified Self flag
      // QS FLAG IS NOT CLEARED INSIDE THIS ISR
    }
  }


  if (Signal < thresh && Pulse == true)
    {   // when the values are going down, the beat is over
      digitalWrite(blinkPin,LOW);            // turn off pin 13 LED
      Pulse = false;                         // reset the Pulse flag so we
can do it again
      amp = P - T;                           // get amplitude of the pulse
wave
       thresh = amp/2 + T;                    // set thresh at 50% of the
amplitude
      P = thresh;                            // reset these for next time
      T = thresh;
    }
  if (N > 2500)
    {                           // if 2.5 seconds go by without a beat
      thresh = 512;                          // set thresh default
      P = 512;                               // set P default
      T = 512;                               // set T default
       lastBeatTime = sampleCounter;          // bring the lastBeatTime up
to date
      firstBeat = true;                      // set these to avoid noise
       secondBeat = false;                    // when we get the heartbeat
back
    }
```

```
  sei();                                    // enable interrupts when youre
done!

}// end isr
```

Cost Estimation:

SL NO.	Components	Price
1.	Arduino UNO	1050
2.	Eyeblink Sensor	705
3.	Pulse Sensor	297
4.	BO motor (2 piece)	170
5.	BO motor Wheel	75
6.	9V battery(2 Piece)	150
7.	LCD display	340
8.	Relay module	85
9.	Buzzer	20
10.	Switch	20
11.	Breadboard	150
12.	Jumper Wire	120
13.	Battery connector	40
14.	Base	100
		Total= 3322

Brief description of the project

For our project, in the driver drowsiness detection and alert system Arduino project, we use an Arduino Uno to serve as the central controller. Initially, the positive terminal of a

buzzer is connected to the Arduino Uno, with the negative terminal grounded. Additionally, a relay module is utilized to control the rotation of a wheel, powered by a 9V battery. An eyeblink sensor is connected to pin 2 of the Arduino Uno, while the remaining pins are connected to a breadboard. A switch is employed to manage power connection. Upon initialization and code execution, the wheel begins rotating, and the buzzer remains silent. However, when the eye remains closed for 4 seconds, the Arduino triggers the relay module to halt wheel rotation and activates the buzzer to emit an audible alert. Concurrently, a pulse sensor measures the driver's pulse rate, initially registering high values while driving but decreasing when the eye is closed, a known indicator of drowsiness. This comprehensive system effectively detects and alerts drivers of potential drowsiness, thus enhancing road safety.

Conclusion

In conclusion, the driver drowsiness detection and alert system implemented using Arduino Uno, UART, and I2C protocols represents a significant advancement in enhancing road safety. By effectively monitoring eyeblink patterns and pulse rate, the system can accurately detect signs of driver drowsiness in real-time. Through the integration of sensors, devices, and interfacing ICs, the system provides timely alerts to drivers, prompting them to take necessary actions to mitigate the risk of accidents caused by fatigue. This project underscores the potential of technology-driven solutions in addressing critical safety concerns on the road, ultimately contributing to safer driving experiences for all.

Contribution Of Member:

ID	NAME	CONTRIBUTION
20301044	MD. TARIQUL ISLAM	Interface the pulse sensor to detect the heart rate of the driver while

		driving and show it in the LCD Display. Contribute the programming code of the pulse sensor and its workings.
20301081	MOHAMMAD SHOWRAB HOSSAN	Interface the Arduino UNO, BO motor setup, connect the circuit diagram and breadboard connection. Will combine the overall structure and assist in programming.
20301268	ASHAKUZZAMAN ODREE	Interface the eyeblink sensor to analyze the driver's sleepiness while driving and give an alarm in the buzzer. Contribute the eyeblink sensor's programming code.

References:

1. Rajasekar, R., Pattni, V. B., & Vanangamudi, S. (2014). Drowsy Driver Sleeping Device and Driver Alert System. *International Journal of Science and Research (IJSR)*, 3(4). Retrieved from http://www.ijsr.net
2. Ahmed, I. (2022). Developing an Arduino Based Anti-sleep Device for Driver (Bachelor's project report). Khulna University of Engineering & Technology.https://www.researchgate.net/publication/374534905_Developing_an_Arduino_Based_Anti-sleep_Device_for_Driver