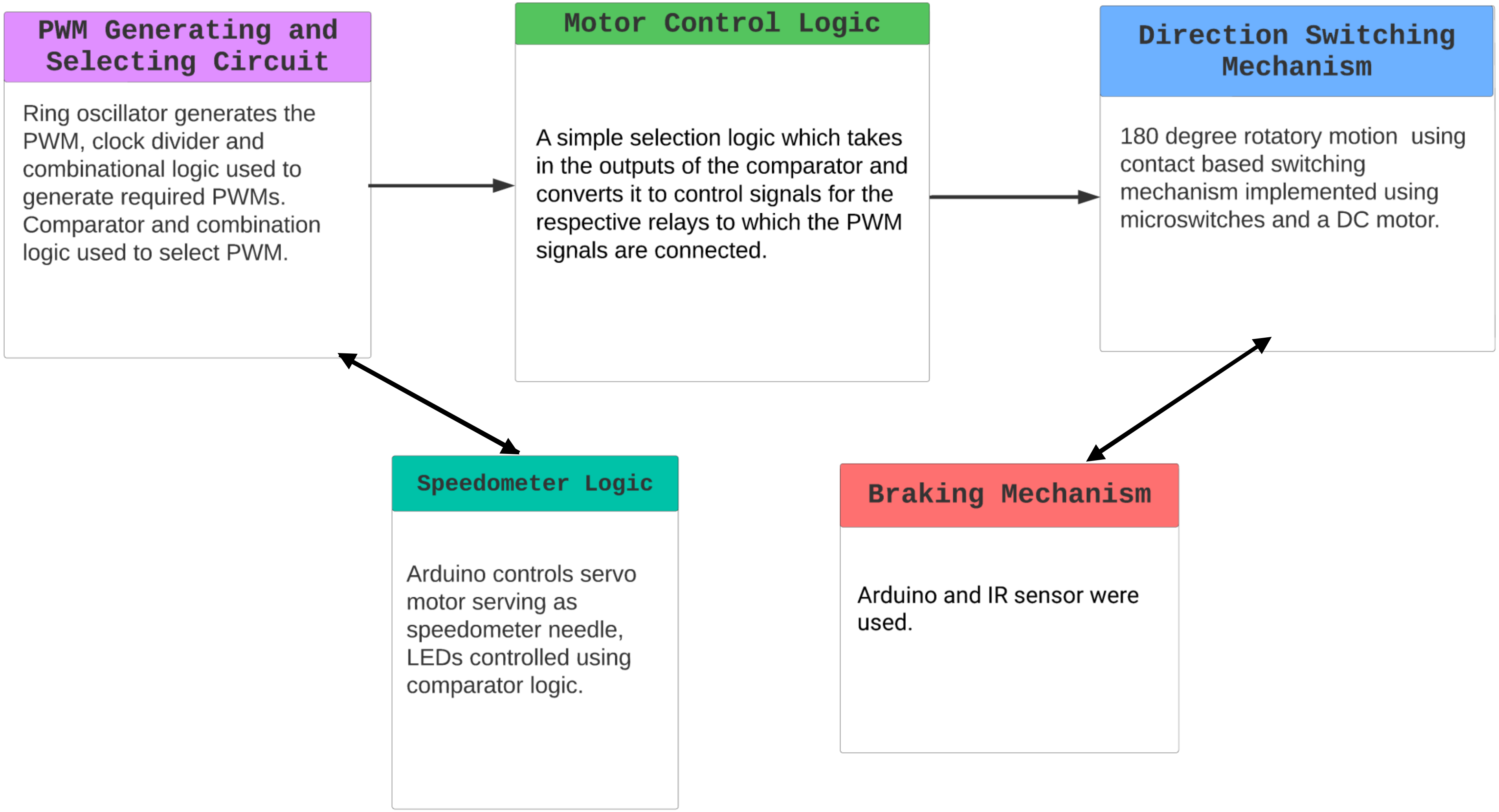
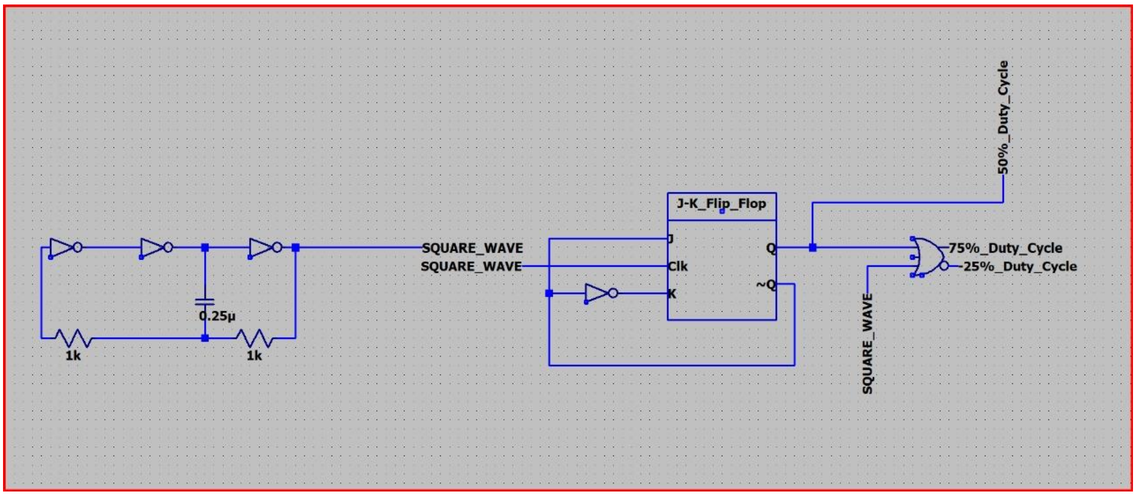
Overview


PWM Generating and Selecting Circuit
Ring oscillator generates the PWM, clock divider and combinational logic used to generate required PWMs. Comparator and combination logic used to select PWM.
Motor Control Logic
A simple selection logic which takes in the outputs of the comparator and converts it to control signals for the respective relays to which the PWM signals are connected.
Direction Switching Mechanism
180 degree rotatory motion using contact based switching mechanism implemented using microswitches and a DC motor.
Speedometer Logic
Arduino controls servo motor serving as speedometer needle, LEDs controlled using comparator logic.
Braking Mechanism
Arduino and IR sensor were used.

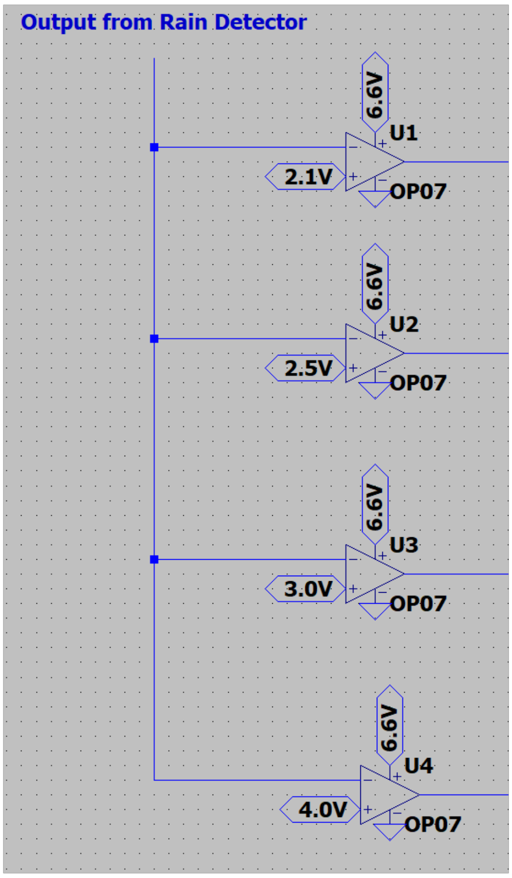
PWM Generating Circuit:



We created a **PWM generating circuit** using a **ring oscillator**, it outputs the pulse width modulated (PWM) signal with a frequency of **980Hz, 50% duty cycle** and an amplitude of 5V.

Further we used a **clock divider (using JK FF)** to generate PWM of frequency **490Hz at 50% duty cycle**, combination logic was used to further generate **75% and 25% duty cycle** PWM at the halved frequency (490Hz).

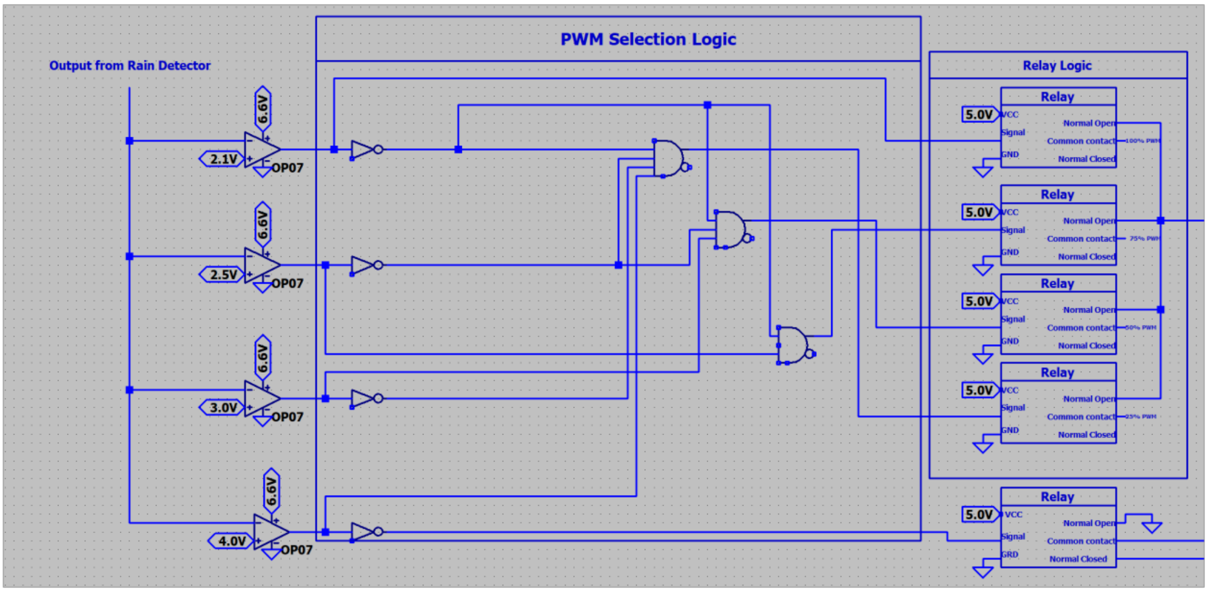
COMPARATOR LOGIC :



We designed a comparator logic circuit using an **operational amplifier (Op-Amp - LM324)** with positive saturation voltage and negative saturation voltage fixed at +6.6V (generated by buck booster) and 0V respectively. Operational amplifier outputs a high (5V) when the **analog output signal** from the **rain sensor** goes below the values fixed at the positive terminal of the Op-Amp.

The values were fixed at the positive terminal (generated by buck boosters) of the Op-Amp based on extensive experiments done on the rain sensor at **different levels of rain intensity.**

PWM selection Logic:

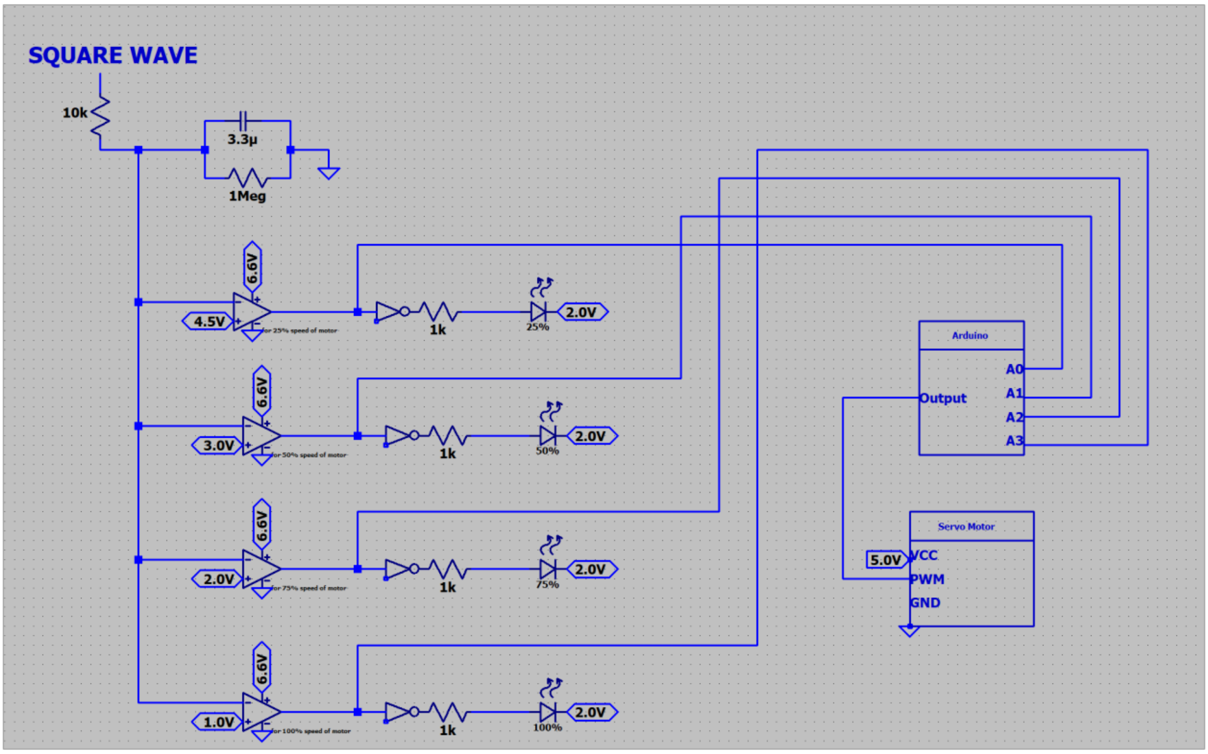


We then implemented a simple selection logic which takes in the outputs of the comparator and converts it to control signals for the respective relays to which the PWM signals are connected.

The output of the four relays is then connected to another relay which has the control input of whether the rain sensor is on or off.

The output of this last relay then goes into the breaking mechanism.

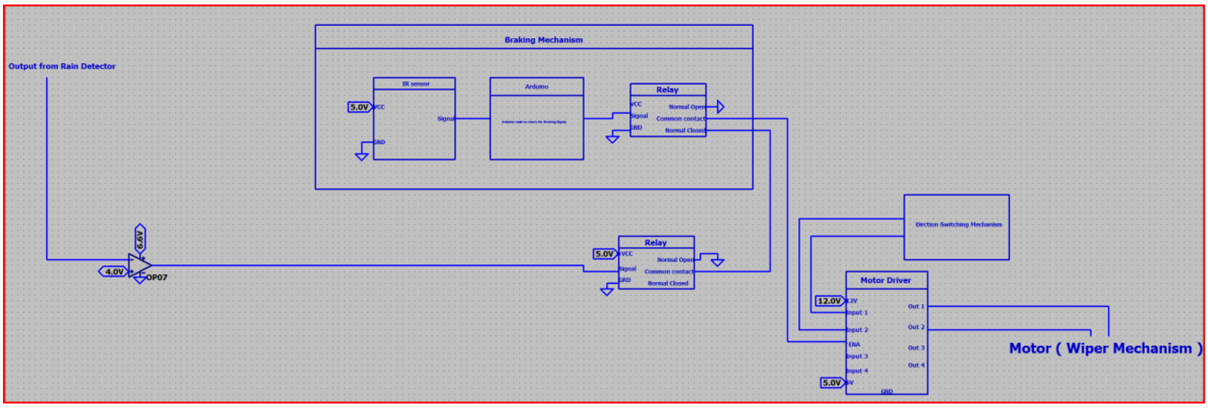
Speedometer Config:



We used an **RC filter** to convert the PWM signal into a **DC voltage**, incorporating a **1MΩ resistor** to prevent unwanted capacitor charging.

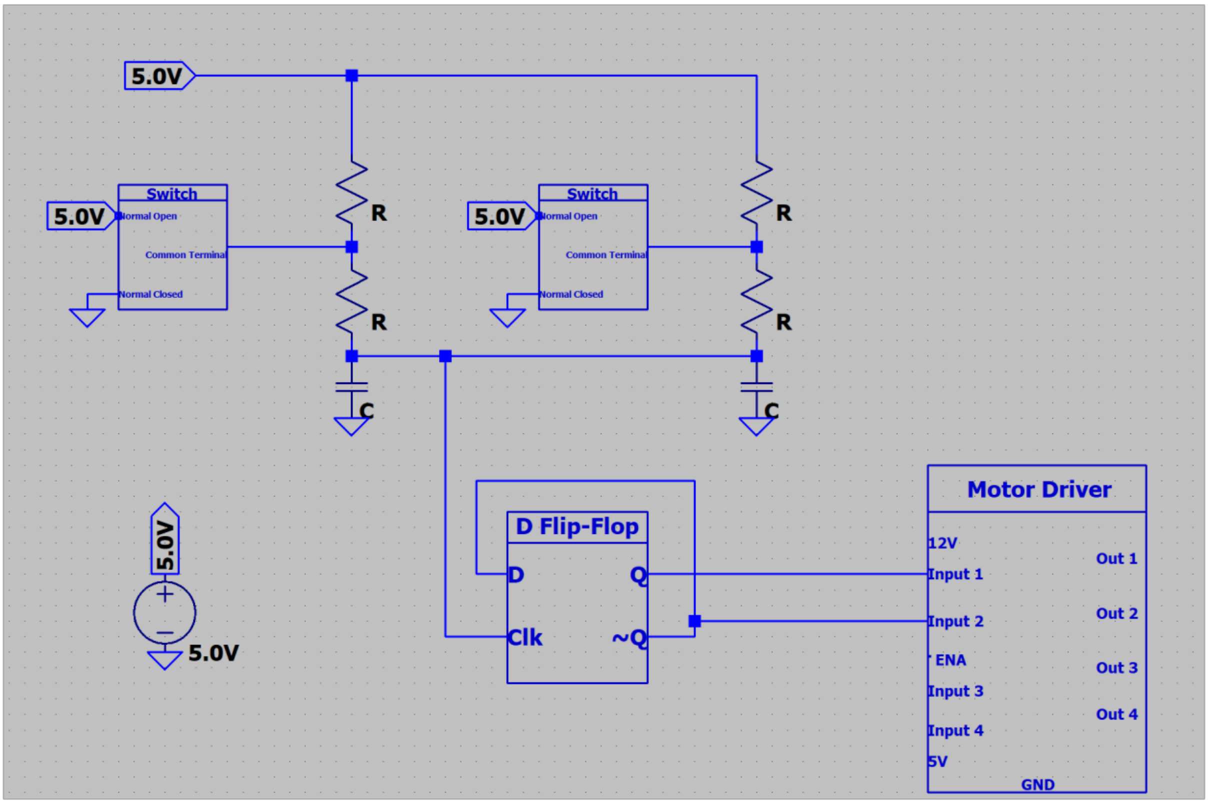
Next, we designed an **LED control logic** using **Op-Amps** and implemented an **Arduino-based logic** to control a **servo motor**, which adjusts to specific angles to indicate different speed levels.

Braking Mechanism:



We implemented the **Wireless Braking Mechanism** using **IR senor, Arduino** and a relay. Control input comes from IR sensor which decides to let the signal from the PWM selection logic pass or not.

Direction Switching Mechanism:



We used two **microswitches** placed opposite to each other at 0° and 180° to develop a **direction-switching mechanism**. When the wiper hits either of the two switches, the **debounced** output of the microswitches are used as the clock of a **D FF** that toggles at the **rising edge of the microswitch output**. I1 and I0 inputs of the motor driver are driven by the Q and Q' outputs of the D FF, allowing the DC motor's motion to be constricted between 0° and 180°.

Arduino Code for Servo and Braking Mechanism:

The photos attached below show the entire micro-controller code we've used in our solution, principally this code achieves 2 objectives, **servo motor control** for implementing the bonus task of a **speedometer** and **wireless braking** using an IR remote.

```
#include <IRremote.h>
#include <Servo.h>

// IR receiver object on pin 7
IRrecv IR(7);

// Servo object
Servo speedometerServo;

#define BRAKE_PIN   8  // Output pin to brake mechanism (if still needed)

// Define input pins
const int bit0Pin = 2;  // Least significant bit (rightmost)
const int bit1Pin = 3;
const int bit2Pin = 4;
const int bit3Pin = 5;  // Most significant bit (leftmost)

int servoAngle = 180; // Default servo angle
bool brakeApplied = false; // Flag to track brake state

// IR code that activates the brake
const unsigned long BRAKE_CODE = 0xE619FF00;
const unsigned long NO_BRAKE_CODE = 0xBA45FF00;

void setup() {
  Serial.begin(9600);

  // Initialize IR reception
  IR.enableIRIn();

  // Set brake pin as output
  pinMode(BRAKE_PIN, OUTPUT);
  digitalWrite(BRAKE_PIN, LOW); // Default brake off

  // Configure input pins with internal pull-up resistors
  pinMode(bit0Pin, INPUT);
  pinMode(bit1Pin, INPUT);
  pinMode(bit2Pin, INPUT);
  pinMode(bit3Pin, INPUT);

  // Attach servo to pin 9
  speedometerServo.attach(9);
  speedometerServo.write(180);
}

void loop() {
  // IR decoding
  if (IR.decode()) {
    Serial.print("IR Code: 0x");
    Serial.println(IR.decodedIRData.decodedRawData, HEX);

    if (IR.decodedIRData.decodedRawData == BRAKE_CODE) {
      Serial.println("BRAKE CODE RECEIVED -> Activating brake");
      digitalWrite(BRAKE_PIN, HIGH);
      brakeApplied = true; // Set brake flag
    } else if (IR.decodedIRData.decodedRawData == NO_BRAKE_CODE) {
      Serial.println("NO BRAKE CODE RECEIVED -> Releasing brake");
      digitalWrite(BRAKE_PIN, LOW);
      brakeApplied = false; // Clear brake flag
    }

    IR.resume(); // Ready for next IR signal
  }
```

```
      brakeApplied = false; // Clear brake flag
    }

    IR.resume(); // Ready for next IR signal
  }

  // If brake is applied, force speedometer to 180 degrees
  if (brakeApplied) {
    speedometerServo.write(180);
    return; // Skip further execution to ensure no speed is set
  }

  // Read all digital inputs (inverted because we're using pull-ups)
  int b0 = digitalRead(bit0Pin);
  int b1 = digitalRead(bit1Pin);
  int b2 = digitalRead(bit2Pin);
  int b3 = digitalRead(bit3Pin);

  // Combine bits into a single value
  int inputValue = b0 + (b1 << 1) + (b2 << 2) + (b3 << 3);

  // Determine servo angle based on input combination
  switch (inputValue) {
    case 0b0000:  // 0000 - Stop
      servoAngle = 180;
      break;
    case 0b0001:  // 0001 - Speed 1
      servoAngle = 135;
      break;
    case 0b0011:  // 0011 - Speed 2
      servoAngle = 90;
      break;
    case 0b0111:  // 0111 - Speed 3
      servoAngle = 45;
      break;
    case 0b1111:  // 1111 - Max Speed
      servoAngle = 0;
      break;
    default:      // Invalid combination (use previous state)
      return;
  }

  // Move servo to selected position
  speedometerServo.write(servoAngle);

  delay(50); // Small delay for stability
}
```

LOGICS USED IN ARDUINO CODE

A) SPEEDOMETER

Our solution uses a **servo motor** and **LEDs** mounted on a cardboard sheet for mimicking the function of a speedometer as requested in the bonus section of this PS.

For driving the LEDs, the **PWM signal** corresponding to various speeds is first passed through an **RC high pass filter** to get the **DC value** which is different for different speeds. This DC value is then used as the reference for a **comparator chain** consisting of 4 comparators, the output of these comparators is used to drive the 4 LEDs. This ensures the LEDs get lit up incrementally for various speeds as required by the PS.

The servo motor is programmed to rotate to 4 discrete angles, namely **45°, 90°, 135°, 180°** using a case statement block which takes 4-bit string as an input argument. The 4-bit string is obtained by **reading 4 digital**

signals which are the comparator outputs, and corresponding to each combination of 4 inputs the servo is set to rotate to the desired angle.

B) WIRELESS BRAKING

On pressing a certain button on the IR remote, the **Hex-Code** is recorded and upon receiving that particular Hex-code, a relay is triggered which cuts off the PWM signal to the "**ENABLE**" pin of the motor driver, the speedometer is also set to 0° which indicates the motor has come to a stop.