ADJUST FAN SPEED USING PWM BASED ON TEMPERATURE RANGES

PS:

Part 2:(moderate task) Enhance the system to dynamically adjust fan speed using PWM based on temperature ranges, and display temperature and fan speed level on a 16x2 LCD.

Solution:

Adjust fan speed using PWM based on temperature ranges

This project automatically controls the fan speed using PWM according to the surrounding temperature. A TMP36 temperature sensor is used to monitor the ambient temperature. The analog voltage from the sensor is converted to a temperature value in Celsius using the formula:

Temperature (°C) = (Voltage - 0.5) × 100

The fan is controlled using a PWM signal generated on analog pin A1 with the `analogWrite()` function. The temperature range from 24°C to 50°C is mapped to a PWM value between 0 and 255 using the `map()` function. The result is constrained within the 0–255 range to ensure safe operation.

In addition, a servo motor (connected to pin 9) is used to visually represent the fan speed level. The PWM voltage is converted to a logic level from 0 to 5, and this level is multiplied by 18 to set the servo angle between 0° and 90°. This provides a simple and intuitive way to show how fast the fan is spinning.

The system behavior is as follows:

- Below 24°C – the fan remains off

- Between 24°C and 50°C – the fan speed increases proportionally with temperature
- Above 50°C – the fan runs at full speed (255)
- The servo angle increases from 0° to 90° depending on the PWM output level

This implementation provides both functional cooling and visual feedback, making it a practical demonstration of temperature-based control using PWM in embedded systems.
Three push buttons are introduced to allow **manual control of fan speed**, overriding the automatic behavior:

- **Button 1 (Pin 6)** → Fan Level 1
- **Button 2 (Pin 7)** → Fan Level 3
- **Button 3 (Pin 8)** → Fan Level 5

When any of these buttons is pressed:

- The fan runs at a fixed PWM speed mapped from the selected level.
- The **servo angle** is set accordingly.
- **Heater and buzzer** are turned off to avoid conflict with automatic control.
- LCD displays “Manual Fan L:x” and “Manual Override” screens in rotation.

This mode improves flexibility by letting the user override the temperature-based behavior in specific scenarios.

LCD display explanation

The project uses a 16x2 I2C liquid crystal display to show real-time status of temperature, fan speed, and system state. The LCD is initialized using the I2C address `0x27`, and controlled through the `LiquidCrystal_I2C` library. This allows communication with only two wires (SDA and SCL), saving digital pins for other components.

The display is divided into two screens that toggle every two seconds:

1. **First screen:**
    - Shows current temperature in Celsius on the first row (e.g., “Temp: 32.56”)
    - The second row displays fan output PWM value (“Fan: 178”) and fan level (“L: 4”), which is derived from the output voltage.
    -
2. **Second screen:**
    - Shows system status based on temperature thresholds
    - For example:
        - “HOT” if temperature is above 35°C
        - “FREEZE” if below 0°C
        - “HEATING...” if below 10°C and heater is on
        - “COOLING...” if fan is active
        - “NORMAL” if temperature is in safe range

The LCD is periodically cleared using `lcd.clear()` to refresh the content, and cursor positions are set using `lcd.setCursor()`. The backlight is enabled for clear visibility using `lcd.backlight()`.

This real-time display provides user-friendly feedback and makes the system easy to monitor without a computer connection.
The first LCD screen shows:

- “Manual Fan L: x”
- “Speed: yyy”

The second screen shows:

- “Manual Override”
- “Servo: angle”

This makes it easy for the user to know when the system is in manual mode and what the settings are.

Working of the system

The system monitors temperature using a TMP36 sensor connected to the analog pin A0. The sensor output is converted to Celsius temperature using the formula:

Temperature (°C) = (Voltage - 0.5) × 100

Based on this temperature value, the system performs three key tasks:

1. **Fan control:**
    - If temperature is below 24°C → fan remains off
    - From 24°C to 50°C → fan speed is increased gradually using PWM
    - Above 50°C → fan runs at full speed (PWM = 255)
    - The fan PWM value is calculated using `map()` and constrained between 0 and 255
    - A servo angle from 0° to 90° is used to visually represent the fan level

2. **Heater control:**
    - If temperature drops below 10°C (low temp threshold), the heater (pin 10) is activated
    - If temperature rises above 10°C, the heater is turned off

3. **Buzzer control:**

    - If temperature < 10°C → buzzer produces a low-frequency tone (300–800 Hz)
    - If temperature > 35°C → buzzer produces a high-frequency tone (1000–2000 Hz)
    - Between 10°C and 35°C → buzzer is silenced

4. **Manual Control Override:**

- When any of the three buttons are pressed:
    - The system switches to manual mode
    - Fan speed is fixed based on the level (1, 3, 5)

- Servo angle reflects the manual level
- Buzzer and heater are turned off
- Temperature sensing continues in the background
- Returns to auto mode when no buttons are pressed

Temperature to fan speed table (approx):

Temp (°C)	Fan PWM (auto mode)	Fan Speed Level	Notes
< 10	0	0	Heater ON, Buzzer (800–300 Hz)
10–24	0	0	Normal
25	~9	0	Fan starts gradually
30	~58	1	Light cooling
35	~113	2	Medium cooling
40	~170	3	Strong cooling
45–50	~226–255	4–5	Max cooling
>50	255	5	Max cooling, Buzzer (1000–2000 Hz)

Manual Mode Fan Mapping (via Buttons):

Button Pressed	Manual Fan Level	Fan PWM	Servo Angle
None	0	Auto Mode	Auto Mode
btn1 (D6)	1	51	18°
btn2 (D7)	3	153	54°
btn3 (D8)	5	255	90°

LCD display:

- Shows current temperature, fan speed, and fan level
- Displays system status like "HOT", "COOLING", "HEATING...", "NORMAL", or "FREEZE" based on temperature thresholds

This system continuously monitors and responds to environmental temperature, automating fan speed, heater control, and user feedback through the LCD and buzzer.

Buzzer control explanation

The buzzer in this system is used to generate audio alerts based on how extreme the temperature is. It provides an additional layer of feedback to signal when the environment is either too cold or too hot.

The buzzer is connected to a digital pin (pin 11), and is controlled using the `tone()` and `noTone()` functions. The system maps the temperature to specific sound frequencies, creating different tones depending on how far the temperature deviates from the safe range.

Here's how the buzzer behaves:

- **If the temperature is below 10°C** (low threshold):
  The system treats this as too cold and maps the temperature range (0°C to 10°C) to a decreasing frequency range (from 800 Hz to 300 Hz). The colder it gets, the lower the pitch of the sound.
  This helps indicate a freezing condition or the need for heating.

- **If the temperature is above 35°C** (high threshold):
  The system considers this too hot and maps the temperature range (35°C to 50°C) to a higher frequency range (from 1000 Hz to 2000 Hz). The hotter it gets, the higher the tone.
  This is useful for warning about overheating or high ambient temperatures.

- **If the temperature is between 10°C and 35°C**:
  The buzzer is turned off using the `noTone()` function, as this is considered a normal and safe operating range.

- Note: In **manual mode**, the buzzer is always **turned off** to avoid unnecessary alerts when user is in control.

This implementation makes the system more interactive and useful in environments where visual indicators (like an LCD) might not always be seen. It provides immediate audio feedback for temperature anomalies without requiring the user to look at the display.

CODE:

```
#include <Wire.h>

#include <Servo.h>

#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);
Servo fanServo;


const int tempPin = A0;
const int fanPin = A1;
const int heaterPin = 10;
const int buzzerPin = 11;
const int servoPin = 9;

const int btn1Pin = 6;
const int btn2Pin = 7;
const int btn3Pin = 8;


const float lowTempThreshold = 10.0;
const float highTempThreshold = 35.0;
const float coldAlert = 0.0;
```

```
void setup() {
  Serial.begin(9600);
  lcd.init();
  lcd.backlight();

  fanServo.attach(servoPin);

  pinMode(tempPin, INPUT);
  pinMode(fanPin, OUTPUT);
  pinMode(heaterPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);

  pinMode(btn1Pin, INPUT_PULLUP);
  pinMode(btn2Pin, INPUT_PULLUP);
  pinMode(btn3Pin, INPUT_PULLUP);
}

void loop() {

  bool btn1 = digitalRead(btn1Pin) == LOW;
  bool btn2 = digitalRead(btn2Pin) == LOW;
  bool btn3 = digitalRead(btn3Pin) == LOW;

  int manualLevel = 0;

  if (btn1) manualLevel = 1;
  else if (btn2) manualLevel = 3;
  else if (btn3) manualLevel = 5;

  int fanSpeed;
  int angle;
  float temperatureC = 0;
```

```
if (manualLevel > 0) {

  fanSpeed = map(manualLevel, 0, 5, 0, 255);
  angle = manualLevel * 18;

  analogWrite(fanPin, fanSpeed);
  fanServo.write(angle);
  digitalWrite(heaterPin, LOW);
  noTone(buzzerPin);


  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Manual Fan L:");
  lcd.print(manualLevel);
  lcd.setCursor(0, 1);
  lcd.print("Speed: ");
  lcd.print(fanSpeed);
  delay(2000);

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Manual Override");
  lcd.setCursor(0, 1);
  lcd.print("Servo: ");
  lcd.print(angle);
  delay(2000);
```

```
  Serial.print("Manual Override | Fan Level: ");
  Serial.print(manualLevel);
  Serial.print(" | Fan Speed: ");
  Serial.print(fanSpeed);
  Serial.print(" | Servo: ");
  Serial.println(angle);
} else {

  int sensorValue = analogRead(tempPin);
  float voltage = sensorValue * (5.0 / 1023.0);
  temperatureC = (voltage - 0.5) * 100.0;

  fanSpeed = map(temperatureC, 24, 50, 0, 255);
  fanSpeed = constrain(fanSpeed, 0, 255);
  analogWrite(fanPin, fanSpeed);

  float fanVoltage = fanSpeed * (5.0 / 255.0);

  int level = (int)(fanVoltage);
  angle = level * 18;

  if (temperatureC > 24) {
    fanServo.write(angle);
  } else {
    fanServo.write(0);
  }


  if (temperatureC < lowTempThreshold){
    digitalWrite(heaterPin, HIGH);
  } else {
    digitalWrite(heaterPin, LOW);
  }
```

```
  if (temperatureC < 10) {
    int freq = map((int)temperatureC, 0, 10, 800, 300);
    tone(buzzerPin, freq);
  } else if (temperatureC > 35) {
    int freq = map((int)temperatureC, 35, 50, 1000, 2000);
    tone(buzzerPin, freq);
  } else {
    noTone(buzzerPin);
  }


  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Temp: ");
  lcd.print(temperatureC);
  lcd.setCursor(0, 1);
  lcd.print("Fan:");
  lcd.print(fanSpeed);
  lcd.print(" L:");
  lcd.print(level);
  delay(2000);
```

```
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Status:");
  lcd.setCursor(0, 1);
  if (temperatureC > highTempThreshold){
    lcd.print("HOT ");
  } else if (temperatureC < coldAlert) {
    lcd.print("FREEZE ");
  } else if (temperatureC < lowTempThreshold){
    lcd.print("Heating...");
  } else if (fanSpeed > 0){
    lcd.print("Cooling...");
  } else {
    lcd.print("Normal");
  }
  delay(2000);


  Serial.print("Temp: ");
  Serial.print(temperatureC);
  Serial.print(" °C | Fan: ");
  Serial.print(fanSpeed);
  Serial.print(" | Level: ");
  Serial.print(level);
  Serial.print(" | Heater: ");
  Serial.print(digitalRead(heaterPin));
  Serial.print(" | Buzzer: ");
  Serial.println(digitalRead(buzzerPin));
 }

 delay(1000);
}
```

SIMULATION LINK AND VIDEO ATTACHED