Documentation of the Automated Fan Control System

Dalianna M. Mojica Cotto

Universidad Ana G. Méndez, Recinto Gurabo- Electrical Engineering Department

Abstract- This project presents the design and implementation of an intelligent ventilation control system based on the Arduino Uno microcontroller. The system integrates multiple sensors—including a PIR motion detector (HC-SR501), a digital temperature sensor (DS18B20), and a real-time clock (DS3231)—to autonomously control a fan based on environmental and temporal conditions. A relay module safely switches the fan, while an LCD I2C display provides real-time feedback. The system was successfully prototyped and tested, demonstrating reliable operation, energy efficiency, and practical application of embedded systems principles. All challenges encountered were software-related and resolved through debugging and optimization.

I. Introduction

Efficient environmental control systems are essential for enhancing comfort, reducing energy consumption, and enabling autonomous operation. Traditional ventilation systems often rely on manual intervention, leading to inefficiency and inconvenience. This project addresses these limitations by developing an automated ventilation control system using an Arduino Uno as the central processing unit. The system responds dynamically to three independent triggers: motion detection, temperature thresholds, and scheduled time windows. By integrating sensor modules and a real-time clock, the project exemplifies the practical application of microprocessor-based systems in real-world scenarios, providing a scalable and adaptable solution for smart environmental management.

Objectives:

- Implement automated fan control using Arduino Uno.
- Integrate PIR motion, DS18B20 temperature, and DS3231 RTC modules.
- Develop a priority-based decision algorithm for relay activation.
- Provide real-time user feedback via an LCD display.
- Ensure system reliability and safety through thorough testing and debugging.
- Document the project for educational and developmental reference.

Circuit Design and Explanation

Components Used:

- Arduino Uno R3
- HC-SR501 PIR Motion Sensor
- DS18B20 Temperature Sensor
- DS3231 Real-Time Clock (RTC) Module
- JQC-3FF-S-Z Relay Module
- LCD I2C Display (0x27)
- Breadboard, jumper wires, 4.7kΩ resistor (pull-up for DS18B20)

Circuit Connections:

- **DS3231 RTC**: I2C interface (SDA → A4, SCL → A5, VCC → 5V, GND → GND)
- **DS18B20**: OneWire interface (Data → D4, VCC → 5V, GND → GND, 4.7kΩ pull-up between Data and VCC)
- **HC-SR501 PIR**: Digital input (OUT → D3, VCC → 5V, GND → GND)
- **Relay Module**: Control pin (IN → D7, VCC → 5V, GND → GND)
- **LCD I2C**: Shared I2C bus with RTC (SDA → A4, SCL → A5)
- **Power**: 5V and GND distributed via breadboard rails

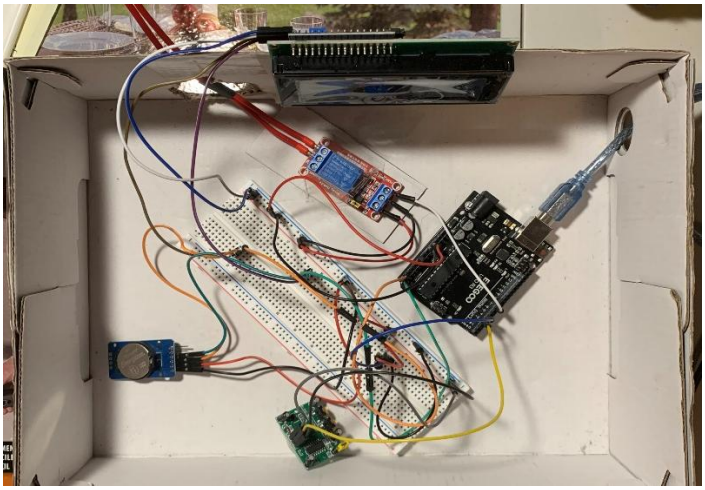

Figure 1: Circuit physically implemented on the breadboard

How the System Works:

1. Arduino continuously polls the PIR sensor, reads temperature via DS18B20, and checks time from DS3231.
2. The decision algorithm evaluates three activation conditions:
    - Temperature
    - Current time
    - Motion detected by PIR
3. If any condition is met, the relay is activated, turning on the fan.
4. The LCD displays real-time information (time, date, temperature) with optimized updates to prevent flickering.
5. Safety measures include insulated relay mounting and disconnecting AC power during wiring.

Future Improvements

- Hardware:

  Safety: Enclose the system in a protective casing and add overcurrent protection.

- Software:
    - Add an interactive LCD menu for parameter adjustment.
    - Implement WiFi/Bluetooth (ESP32/HC-05) for remote monitoring and control.Implement hysteresis and debounce logic to enhance sensor accuracy.
    - Introduce sleep modes to optimize power consumption.

Conclusion

This project successfully demonstrated the integration of multiple sensors and modules into a functional automated ventilation control system using Arduino. The system reliably activates a fan based on temperature, motion, and schedule, providing a practical example of embedded system design. All encountered issues—such as RTC library installation, time formatting errors, and LCD flickering—were software-related and resolved through systematic debugging and code optimization. The final prototype operates continuously and accurately, showcasing the potential for energy-efficient and autonomous environmental control.

Key Achievements:

- Successful sensor integration and real-time data processing.
- Implementation of a priority-based control algorithm.
- Optimized LCD output for stable visual feedback.
- Enhanced safety with proper relay insulation and wiring practices.

References

- Maxim Integrated. (2015). *DS18B20: Programmable resolution 1-Wire digital thermometer* (Rev. 5). Maxim Integrated. [https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf]
- Datasheet PDF desde MPJA: [https://www.mpja.com/download/31227sc.pdf]
- Ficha técnica / pinout en Components101: [https://components101.com/sensors/hc-sr501-pir-sensor]
- Monk, S. (2016). *Programming Arduino: Getting started with sketches* (2nd ed.). McGraw-Hill Education.
- OpenAI. (2025). *ChatGPT (GPT-5.1)* [Large language model]. https://chat.openai.com/

Link to the simulation in Tinkercad:

https://www.tinkercad.com/things/dQ5MVYcterv-simulation-of-the-automated-fan-control-system/editel?returnTo=https%3A%2F%2Fwww.tinkercad.com%2Fdashboard%2Fdesigns%2Fall&sharecode=yGtfq80yOyNQlIjVbL33nbsBdQp3_76VTnQbnYWmDdo

Appendix A – Program Code

```
/*
Librerias para el proyecto
*/

#include <Wire.h>
#include <RtcDS3231.h>
#include <LiquidCrystal_I2C.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// ----- DS3231 RTC -----
static RtcDS3231<TwoWire> RTC(Wire);

// ----- LCD -----
LiquidCrystal_I2C lcd(0x27, 16, 2);

// ===== PIR Sensor =====
#define PIR_PIN 3

// ----- DS18B20 Temperature Sensor -----
#define ONE_WIRE_BUS 4
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

// ----- Relay for Ventilator -----
#define RELAY_PIN 7
int tempThresholdF = 70; // Cambiado de 80 a 70

// Array para días de la semana en inglés
const char* daysOfWeek[] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};

// Variables para control de visualización
char lastTimeString[9] = "";
char lastDateString[14] = "";
char lastDayOfWeek[4] = "";
float lastTempF = -100;
bool lastIsPM = false;

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

  // Initialize RTC
  RTC.Begin();
  RTC.SetSquareWavePin(DS3231SquareWavePin_ModeNone);

  // AJUSTAR HORA Y FECHA CORRECTAS - Miércoles 26 de noviembre de 2025, 4:13 PM
  //RtcDateTime newTime(2025, 11, 26, 20, 53, 0); // Año, Mes, Día, Hora, Minuto,
Segundo
  //RTC.SetDateTime(newTime);

  // Initialize LCD
```

```
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("System Ready");
  delay(1000);

  // Initialize temperature sensor
  sensors.begin();

  // PIR
  pinMode(PIR_PIN, INPUT);

  // Initialize relay pin
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, LOW);

  Serial.println("System Ready");
  Serial.println("Hora ajustada a: 2025-11-26 16:13:00 (4:13 PM)");
}

// Función para ajustar la hora desde el Monitor Serial
void checkSerialForTimeAdjust() {
  if (Serial.available() > 0) {
    String input = Serial.readStringUntil('\n');
    input.trim();

    if (input.startsWith("SET,")) {
      input = input.substring(4); // Remover "SET,"

      int year, month, day, hour, minute, second;
      if (sscanf(input.c_str(), "%d,%d,%d,%d,%d,%d", &year, &month, &day, &hour,
&minute, &second) == 6) {
        // Validar los valores
        if (year >= 2000 && year <= 2100 &&
            month >= 1 && month <= 12 &&
            day >= 1 && day <= 31 &&
            hour >= 0 && hour <= 23 &&
            minute >= 0 && minute <= 59 &&
            second >= 0 && second <= 59) {

          RtcDateTime newTime(year, month, day, hour, minute, second);
          RTC.SetDateTime(newTime);

          Serial.println("Hora ajustada correctamente!");
          Serial.print("Nueva hora: ");
          Serial.print(year);
          Serial.print("-");
          Serial.print(month);
          Serial.print("-");
          Serial.print(day);
          Serial.print(" ");
          Serial.print(hour);
```

```
          Serial.print(":");
          Serial.print(minute);
          Serial.print(":");
          Serial.println(second);

          // Mostrar confirmación en LCD
          lcd.clear();
          lcd.setCursor(0, 0);
          lcd.print("Hora Ajustada");
          lcd.setCursor(0, 1);
          lcd.print("OK!");
          delay(2000);
          lcd.clear(); // Limpiar para volver a la visualización normal
        } else {
          Serial.println("Error: Valores fuera de rango válido");
        }
      } else {
        Serial.println("Error: Formato incorrecto");
        Serial.println("Usa: SET,YYYY,MM,DD,HH,MM,SS");
      }
    }
  }
}

void loop() {
  // Verificar si hay comando para ajustar hora
  checkSerialForTimeAdjust();

  // ========== PIR DETECTION ==========
  int pirState = digitalRead(PIR_PIN);

  // ----- Read RTC -----
  RtcDateTime now = RTC.GetDateTime();

  // Convertir a formato 12 horas
  uint8_t hour = now.Hour();
  bool isPM = false;

  if (hour == 0) {
    hour = 12; // 12 AM
  } else if (hour == 12) {
    isPM = true; // 12 PM
  } else if (hour > 12) {
    hour = hour - 12;
    isPM = true;
  }

  // Obtener día de la semana (0=Dom, 1=Lun, ..., 6=Sab)
  uint8_t dayOfWeek = now.DayOfWeek();

  // Formatear hora en 12 horas
  char timeString[9];
  sprintf(timeString, "%02u:%02u:%02u", hour, now.Minute(), now.Second());
```

```
  // Formatear fecha completa con año de 4 dígitos (YYYY)
  char dateString[14];
  sprintf(dateString, "%02u-%02u-%04u", now.Month(), now.Day(), now.Year());

  // ----- Read Temperature -----
  sensors.requestTemperatures();
  float tempC = sensors.getTempCByIndex(0);
  float tempF = sensors.getTempFByIndex(0);

  // ----- Display on LCD -----
  // NO usar lcd.clear() - actualizar solo lo que cambia

  // Línea superior: Día de semana y fecha completa (MM-DD-YYYY)
  if (strcmp(daysOfWeek[dayOfWeek], lastDayOfWeek) != 0 || strcmp(dateString,
lastDateString) != 0) {
    lcd.setCursor(0, 0);
    lcd.print(daysOfWeek[dayOfWeek]); // Mon, Tue, etc.
    lcd.print(" ");
    lcd.print(dateString); // MM-DD-YYYY
    // Limpiar el resto de la línea si es necesario
    int charsPrinted = strlen(daysOfWeek[dayOfWeek]) + 1 + strlen(dateString);
    for (int i = charsPrinted; i < 16; i++) {
      lcd.print(" ");
    }

    // Actualizar variables de control
    strcpy(lastDayOfWeek, daysOfWeek[dayOfWeek]);
    strcpy(lastDateString, dateString);
  }

  // Línea inferior: Hora (12h), AM/PM y temperatura
  if (strcmp(timeString, lastTimeString) != 0 || tempF != lastTempF || isPM !=
lastIsPM) {
    lcd.setCursor(0, 1);
    lcd.print(timeString); // HH:MM:SS en 12h
    lcd.print(isPM ? " PM" : " AM");
    lcd.print(" ");
    lcd.print(tempF, 1);
    lcd.print((char)223);
    lcd.print("F");
    // Limpiar el resto de la línea si es necesario
    int charsPrinted = strlen(timeString) + 3 + (tempF >= 100 ? 6 : 5);
    for (int i = charsPrinted; i < 16; i++) {
      lcd.print(" ");
    }

    // Actualizar variables de control
    strcpy(lastTimeString, timeString);
    lastTempF = tempF;
    lastIsPM = isPM;
  }
```

```
  // ----- Optional Serial Output -----
  Serial.print(daysOfWeek[dayOfWeek]);
  Serial.print(" ");
  Serial.print(dateString);
  Serial.print(" ");
  Serial.print(timeString);
  Serial.print(isPM ? " PM" : " AM");
  Serial.print(" - ");
  Serial.print(tempC);
  Serial.print("C / ");
  Serial.print(tempF);
  Serial.print("F - ");
  Serial.println(pirState == HIGH ? "MOTION" : "NO MOTION");

  // ----- Relay Control -----
  bool shouldTurnOn = false;

  // Condición 1: Temperatura mayor a 70°F
  if (tempF > tempThresholdF) {
    shouldTurnOn = true;
    Serial.println("Ventilador ON por temperatura");
  }

  // Condición 2: Horario entre 5:19 PM y 5:20 PM (17:19 a 17:20 en 24h)
  if (now.Hour() == 17 && now.Minute() >= 19 && now.Minute() <= 20) {
    shouldTurnOn = true;
    Serial.println("Ventilador ON por horario");
  }

  // Condición 3: Detección de movimiento
  if (pirState == HIGH) {
    shouldTurnOn = true;
    Serial.println("Ventilador ON por movimiento");
  }

  // Control final del relay
  if (shouldTurnOn) {
    digitalWrite(RELAY_PIN, HIGH);
    Serial.println("--- VENTILADOR ENCENDIDO ---");
  } else {
    digitalWrite(RELAY_PIN, LOW);
    Serial.println("--- VENTILADOR APAGADO ---");
  }

  delay(1000);
}
```

Appendix B – Flowcharts, circuit design and photos of the operation

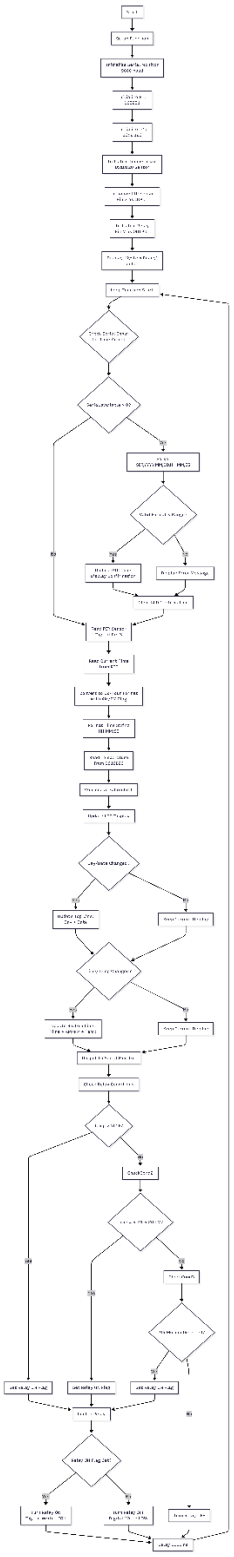

Figure 2: Main Program Flowchart

The main program flowchart illustrates the complete operational flow of an Arduino-based environmental control system. The process begins with system initialization, where all hardware components are sequentially set up: the Serial Monitor at 9600 baud, the DS3231 Real-Time Clock (RTC), a 16x2 I2C LCD display, the DS18B20 temperature sensor, the PIR motion sensor configured on pin 3 as an input, and finally the relay on pin 7 as an output to control the ventilator fan. Once initialized, the system displays a "System Ready" message and enters its main continuous loop.

Within this main loop, the program first checks for any incoming serial commands that might adjust the system time, accepting properly formatted "SET" commands with date and time parameters. It then reads the PIR sensor to detect motion. Simultaneously, it retrieves the current time from the RTC, converts it to a 12-hour format with AM/PM designation, and formats it as a displayable string. Next, it reads the temperature from the DS18B20 sensor and converts it to Fahrenheit. The LCD display is updated efficiently, only refreshing lines when the date or temperature data actually changes to avoid screen flickering. All information is also output to the Serial Monitor for debugging.

The system then evaluates three independent conditions to determine whether to activate the ventilator relay: if the temperature exceeds 70°F, if the current time falls within the scheduled window of 5:19-5:20 PM, or if motion was detected by the PIR sensor. If any of these conditions is true, the relay is activated (HIGH signal); otherwise, it remains deactivated (LOW signal). Each activation triggers a corresponding log message. Finally, the system pauses for 1000 milliseconds before repeating the entire loop, creating a continuous monitoring and control cycle that runs once per second.

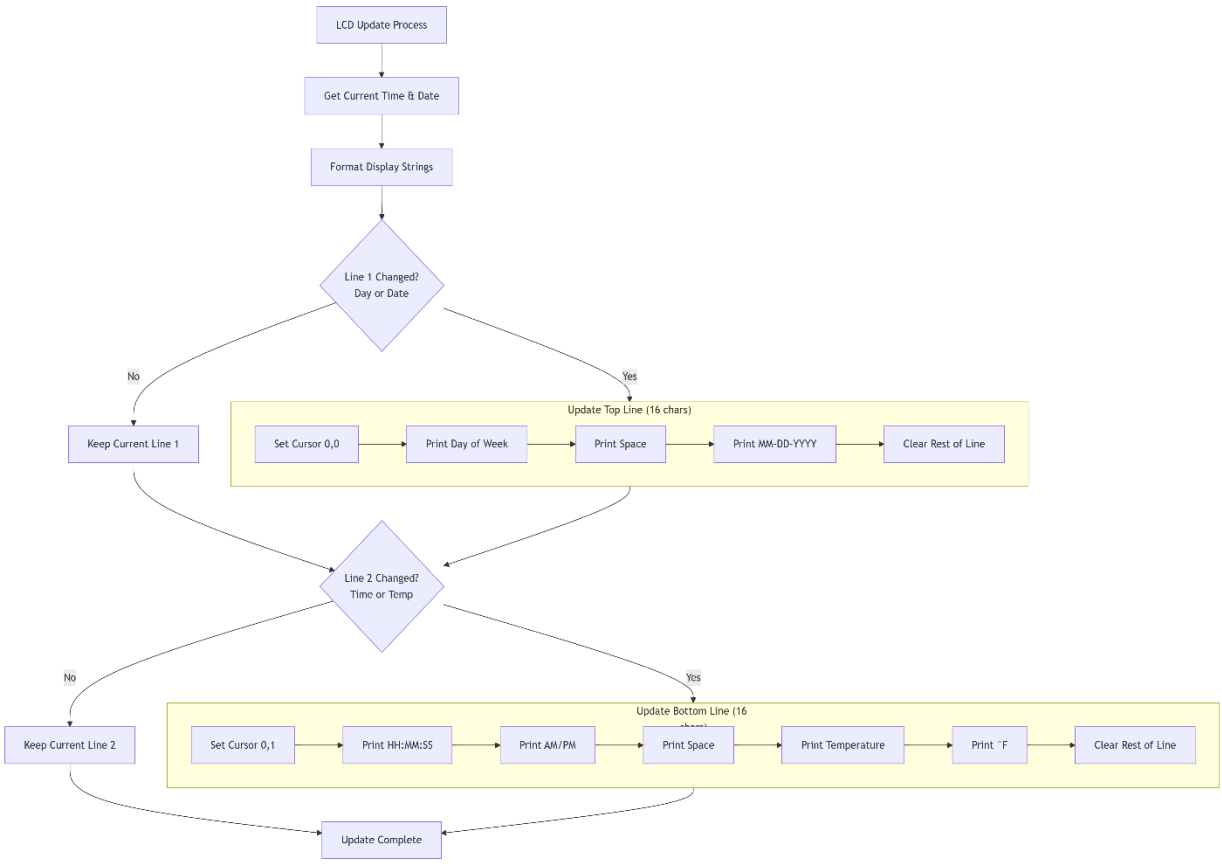

Figure 3: LCD Display

The LCD display update process is designed for efficiency and minimal screen flickering. When updating the display, the system first retrieves the current time and date from the RTC and formats them into display-ready strings. It then intelligently checks whether the content of each LCD line has actually changed before performing any updates. For the top line (Line 1), it compares both the day of the week and the complete date (in MM-DD-YYYY format) against previously stored values. Only if either the day or date has changed does it proceed to update the entire top line—first positioning the cursor, then printing the three-letter day abbreviation, a space, and the full date, finally clearing any remaining characters to ensure a clean display.

Similarly, for the bottom line (Line 2), the system verifies whether the time string (in HH:MM:SS format), the AM/PM indicator, or the temperature reading has changed since the last update. If any of these values differ, it updates the entire second line by printing the formatted time, the appropriate AM/PM indicator, a space, the temperature value with one decimal place, the degree symbol, and the "F" for Fahrenheit, again clearing any leftover characters. This selective update approach prevents unnecessary screen refreshes, maintains display stability, and conserves processing resources while ensuring the displayed information remains current and accurate.

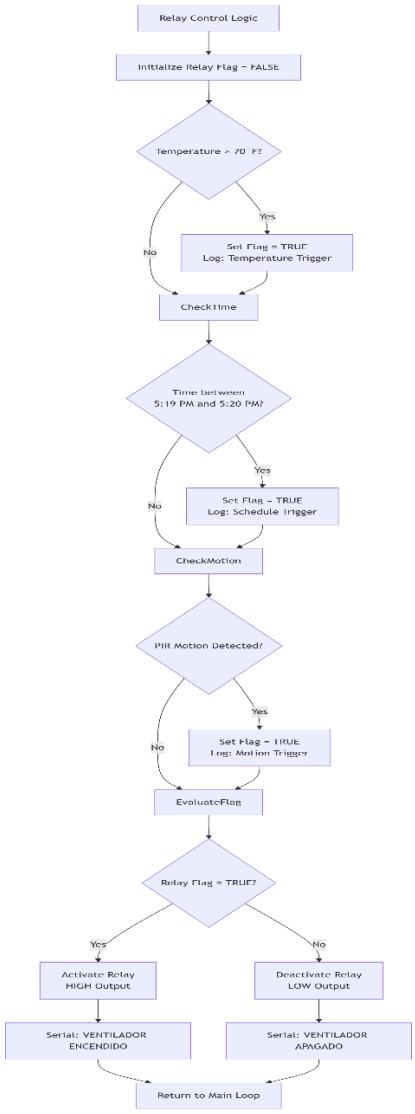

Figure 4: Relay Control Decision Logic

The relay control system employs a logical OR decision structure where any one of three independent conditions can trigger ventilator activation. The process begins by initializing a relay flag to FALSE (off state). The system then sequentially evaluates three distinct criteria. First, it checks if the current temperature reading exceeds the 70°F threshold; if true, it sets the relay flag to TRUE and logs a temperature-based activation message. Second, regardless of the temperature outcome, it proceeds to verify if the current time falls within the specifically scheduled window of 5:19 PM to 5:20 PM (17:19 to 17:20 in 24-hour format); meeting this condition also sets the flag to TRUE with a corresponding schedule-based log entry. Third, it evaluates whether motion has been detected by the PIR sensor, and any detected motion similarly sets the flag to TRUE with a motion-based activation message.

After evaluating all three conditions, the system examines the final state of the relay flag. If the flag is TRUE (indicating at least one triggering condition was met), it activates the relay by sending a HIGH signal to the control pin, which physically powers the ventilator, and logs "VENTILADOR ENCENDIDO" to the serial monitor. If the flag remains FALSE (meaning none of the three conditions were satisfied), it deactivates the relay with a LOW signal, turning off the ventilator, and logs "VENTILADOR APAGADO." This design ensures the ventilator operates whenever environmental, temporal, or occupancy conditions warrant activation, providing automated climate and air circulation control through a simple but effective multi-criteria decision system.

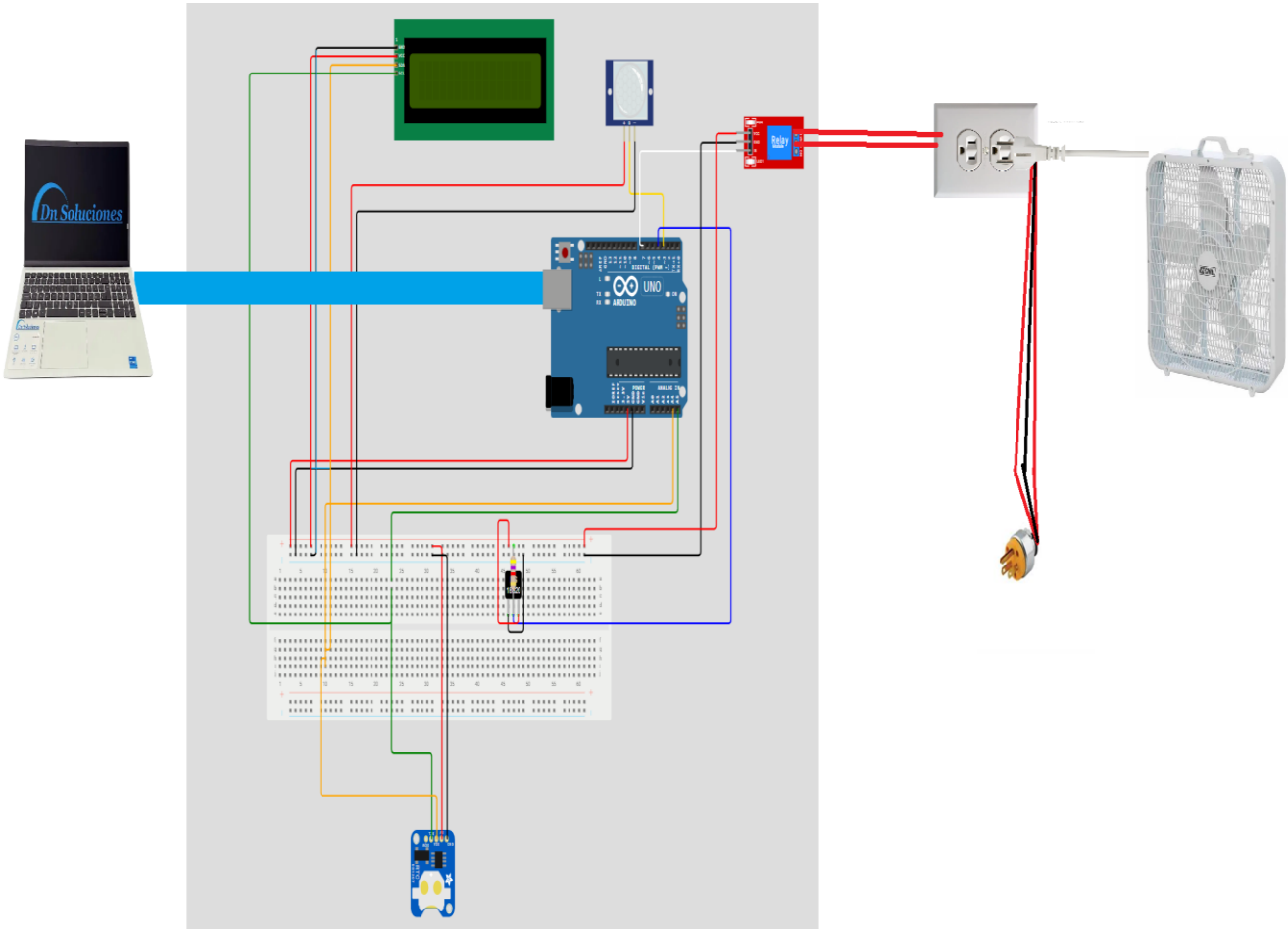

Figure 5: Circuit Design

The circuit consists of a DS18B20 temperature sensor, an HC-SR501 PIR motion sensor, an RTC DS3231 real-time clock, and a microcontroller that controls a fan through a relay or transistor. The DS18B20 measures ambient temperature, and the PIR detects motion. The RTC DS3231 provides accurate time, allowing the microcontroller to follow a user-defined schedule. The fan turns on if the temperature exceeds a set threshold, if motion is detected, or if the current time matches the scheduled period. Power lines are shared among components, and the DS18B20 uses a pull-up resistor for reliable communication. The microcontroller activates the relay or transistor to power the fan, completing the automated system.

Photos of the operation

Figure 6: LCD displays automatic power-on time

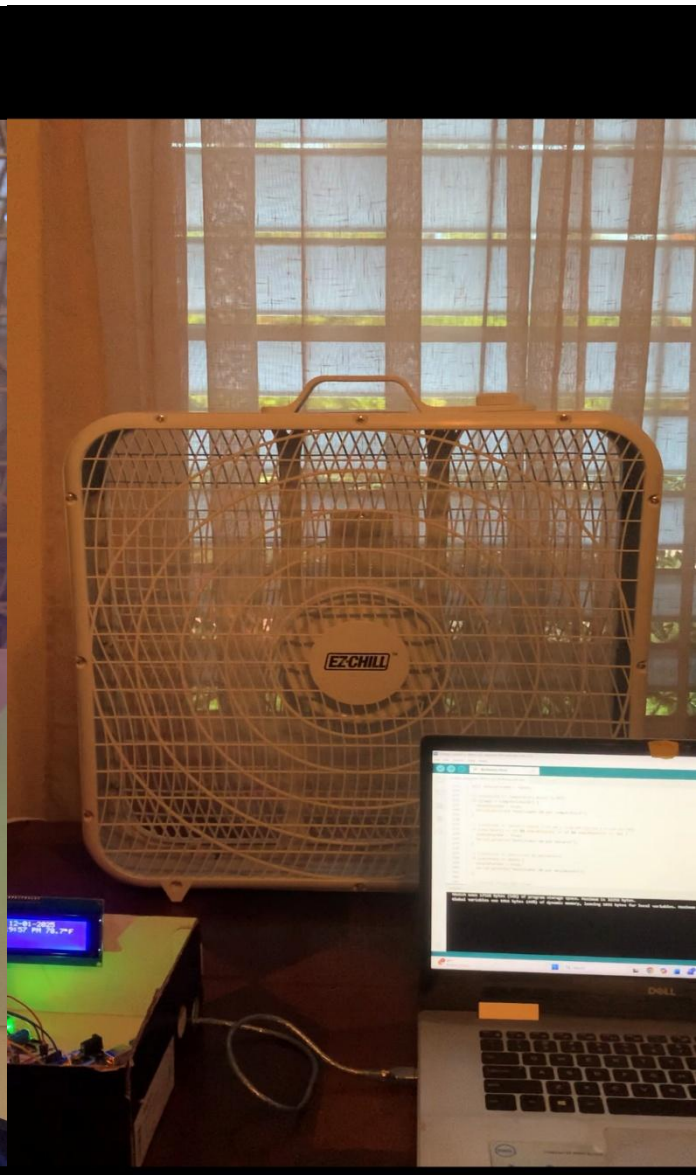

Figure 7: Fan On

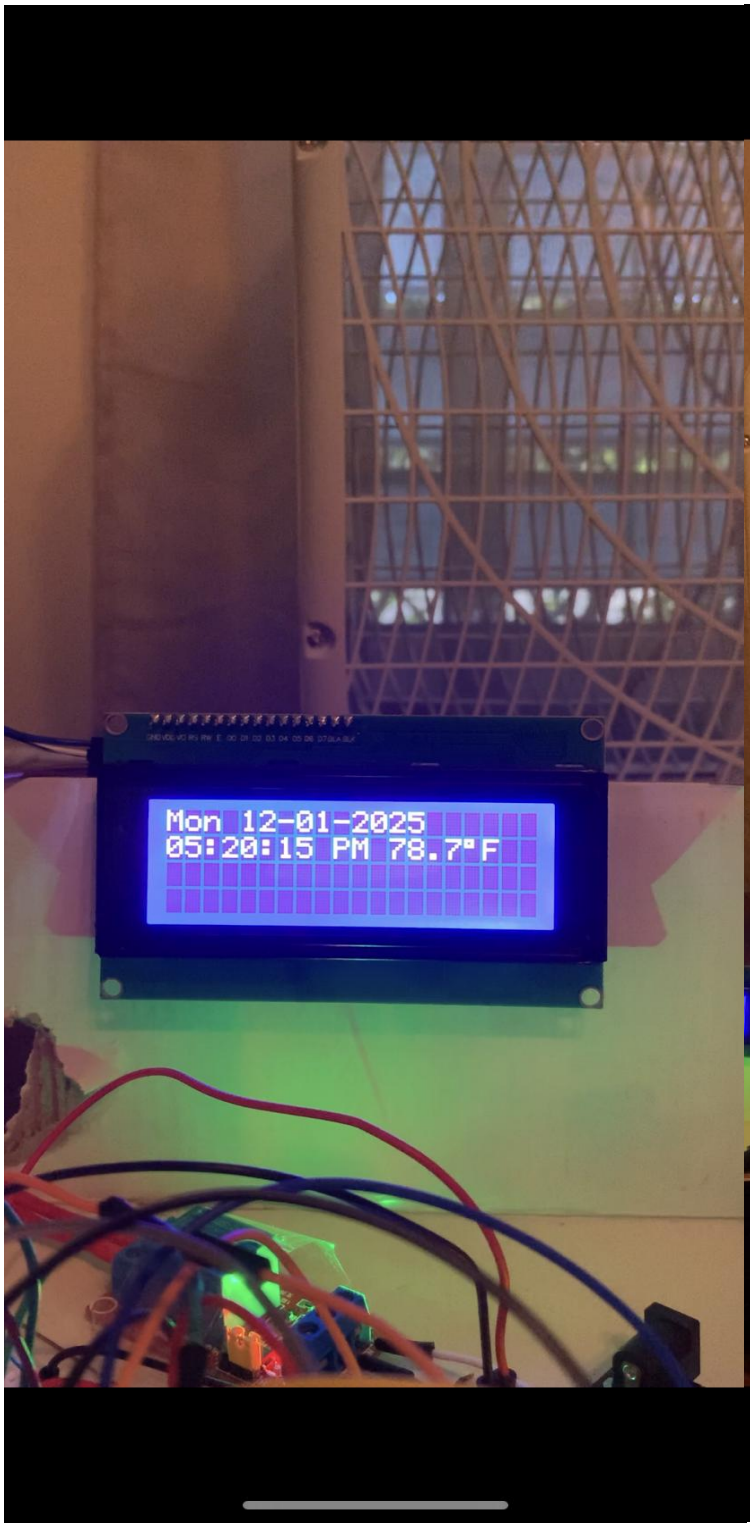

Figure 8: LCD displays automatic power-off time

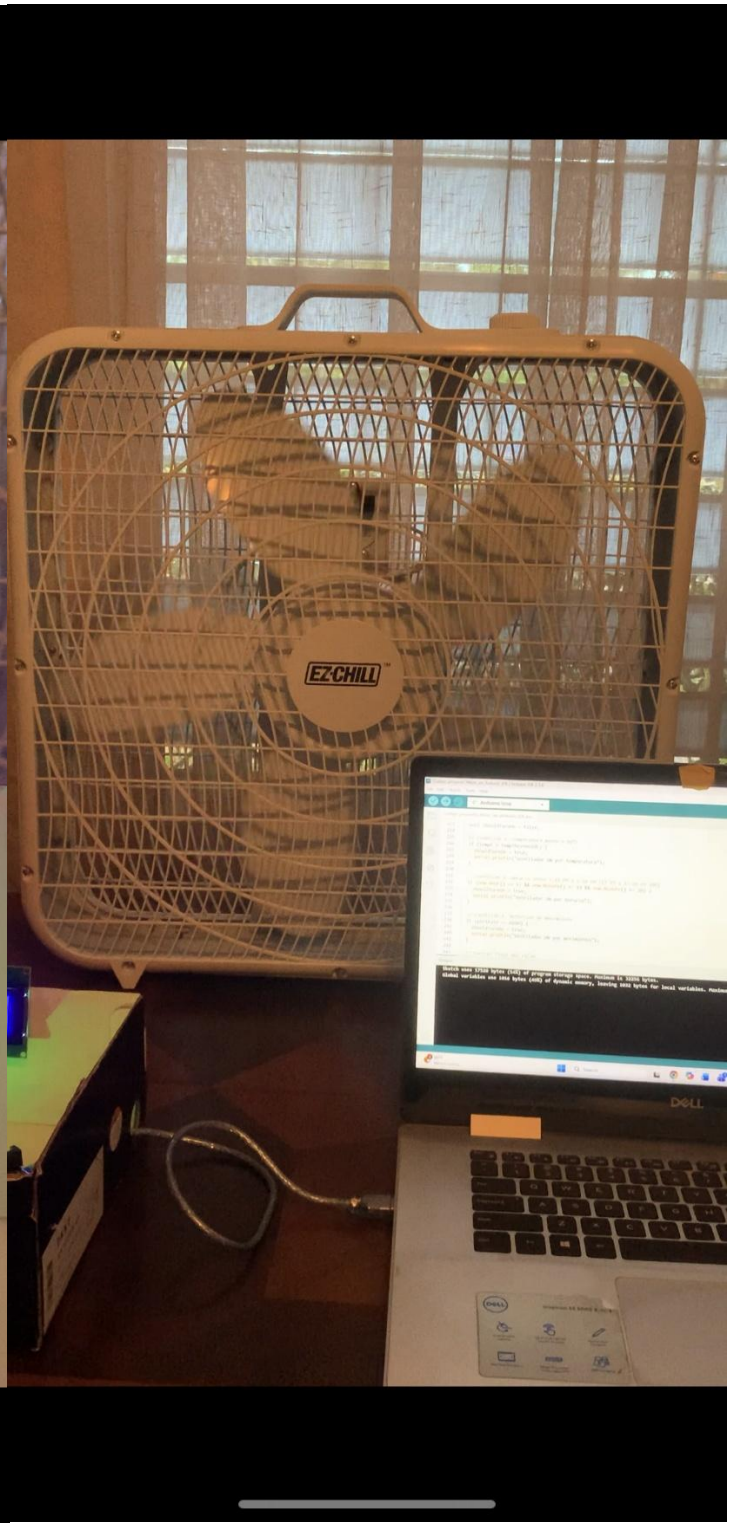

Figure 9: The fan automatically turns off at the set time.