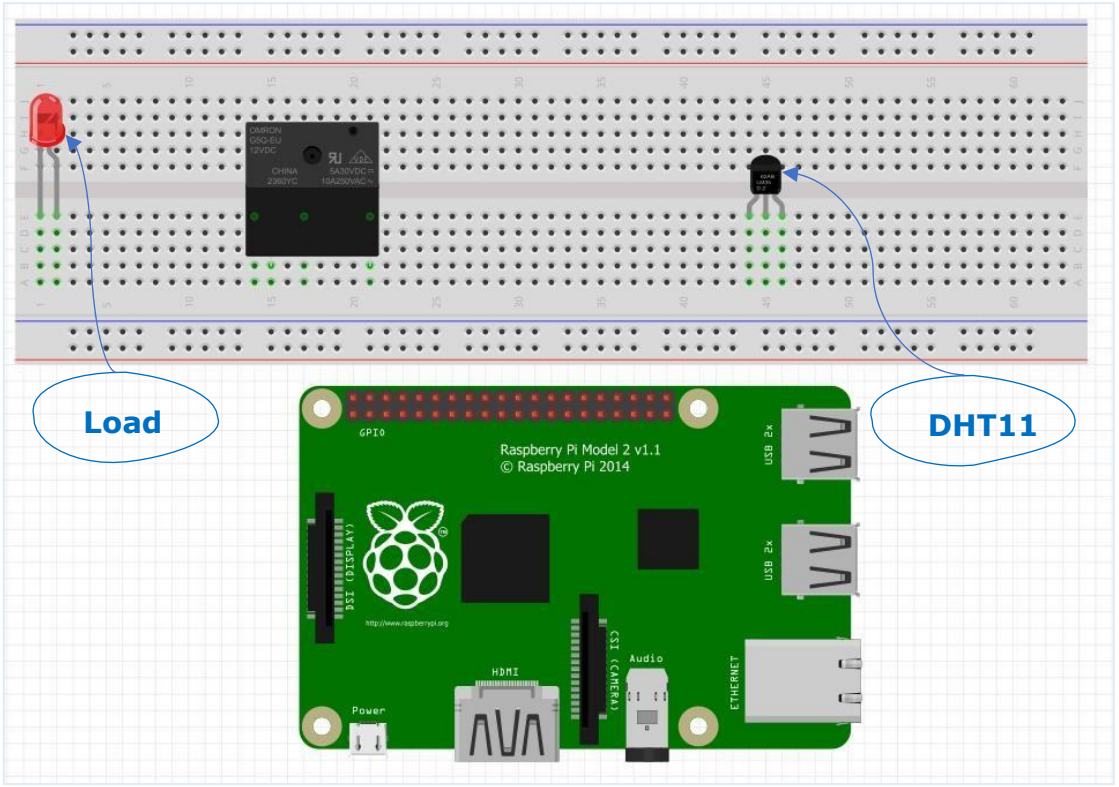
Cooling System Control

Introduction

The purpose of this project is to obtain a control system for a room cooling system, by using the DHT11 sensor with Raspberry Pi.

Overview

In the following figure, we will see a simple structure equivalent to the project.



Components

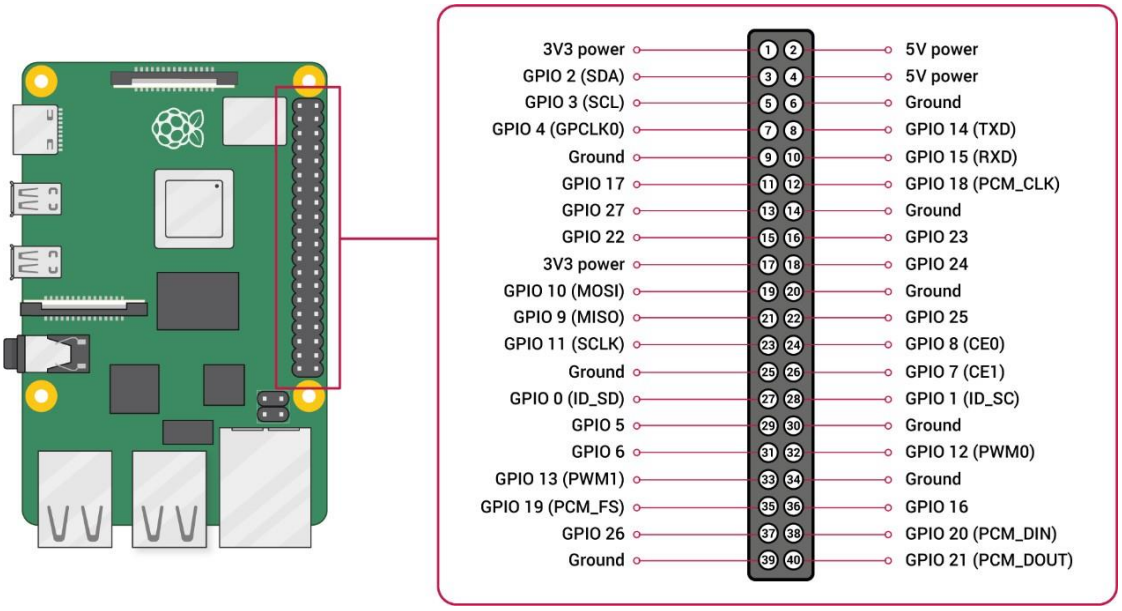
1- **Raspberry Pi** is a credit card-sized single-board computer that has gained popularity for its versatility and affordability. It was developed by the Raspberry Pi Foundation and is widely used for various projects, including home automation, robotics, media centers, and IoT (Internet of Things) applications.

The Raspberry Pi board features a set of GPIO (General Purpose Input/Output) pins that allow you to connect and control external devices and components. These pins can be programmed to read inputs from sensors, control actuators, communicate with other devices, and perform other digital operations.

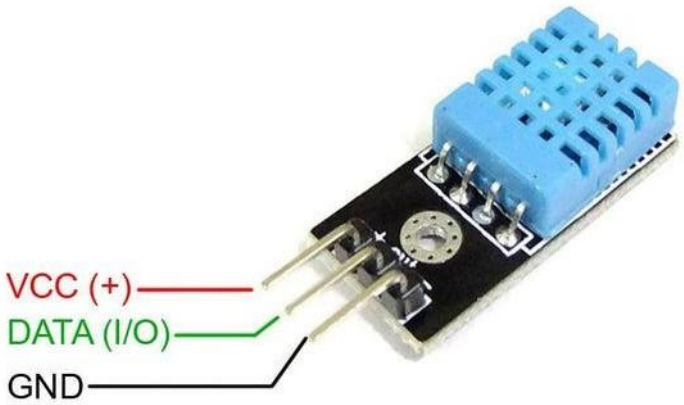
Here's a brief overview of the different types of pins on the Raspberry Pi:

- **Power Pins:** These pins provide power to the board and connected peripherals. The common power pins are 5V (power supply), 3.3V (lower voltage power supply), and ground (GND).
- **GPIO Pins:** The Raspberry Pi offers a number of GPIO pins that can be configured as inputs or outputs. These pins are used to connect and control external devices such as sensors, LEDs, relays, and more. GPIO pins are numbered and can be referred to by their BCM (Broadcom SOC channel) or physical numbering schemes.
- **Serial Communication Pins:** The Raspberry Pi has UART (Universal Asynchronous Receiver-Transmitter) pins that allow serial communication with other devices using protocols like UART, SPI (Serial Peripheral Interface), and I2C (Inter-Integrated Circuit).
- **Other Pins:** In addition to the above, the Raspberry Pi may have additional pins specific to certain models or purposes. For example, some models have dedicated pins for camera and display connections, audio output, and other specialized functionalities.

In the following figure, we will see the order of the pins in Raspberry Pi.



2- **DHT11** is a commonly used Temperature and humidity sensor. The sensor comes with a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data. The sensor is also factory calibrated and hence easy to interface with other microcontrollers.



C++ Code

```
#include <wiringPi.h>
#include <iostream>
#include <fstream>
#include <jsoncpp/json/json.h>
#include <thread>
#include <chrono>
#include <string>
using namespace std;
#define DHT_PIN 7             // GPIO pin for DHT11 sensor
#define CoolingS 0           // GPIO pin for relay
#define VentilationS 1
#define JSON_FILE "/var/www/html/data.json" // JSON file path in web
server directory
std::string CoolingS_status, VentilationS_status;
struct SensorData {
    float humidity;
    float temperature;
};
int readDHTSensor(float& humidity, float& temperature) {
    int data[5] = { 0, 0, 0, 0, 0 };

    pinMode(DHT_PIN, OUTPUT);
    digitalWrite(DHT_PIN, LOW);
    delay(18);
    digitalWrite(DHT_PIN, HIGH);
    delayMicroseconds(40);
    pinMode(DHT_PIN, INPUT);

    int pulseCount = 0;
    while (digitalRead(DHT_PIN) == LOW) {
        delayMicroseconds(2);
        pulseCount++;
        if (pulseCount > 10000)
            return -1;
    }

    pulseCount = 0;
    while (digitalRead(DHT_PIN) == HIGH) {
        delayMicroseconds(2);
        pulseCount++;
        if (pulseCount > 10000)
            return -1;
    }
    for (int i = 0; i < 40; i++) {
```

```
        pulseCount = 0;
        while (digitalRead(DHT_PIN) == LOW) {
            delayMicroseconds(2);
            pulseCount++;
            if (pulseCount > 10000)
                return -1;
        }

        pulseCount = 0;
        while (digitalRead(DHT_PIN) == HIGH) {
            delayMicroseconds(2);
            pulseCount++;
            if (pulseCount > 10000)
                return -1;
        }

        data[i / 8] <<= 1;
        if (pulseCount > 30)
            data[i / 8] |= 1;
    }

    if (data[4] == ((data[0] + data[1] + data[2] + data[3]) & 0xFF)) {
        humidity = static_cast<float>((data[0] << 8) + data[1]) * 0.1f;
        temperature = static_cast<float>(((data[2] & 0x7F) << 8) +
data[3]) * 0.1f;
        if (data[2] & 0x80)
            temperature *= -1;
        return 0;
    }

    return -1;
}

void Start_VentilationS() {
    digitalWrite(VentilationS, HIGH);
    std::cout << "Ventilation System turned ON" << std::endl;
    VentilationS_status = "Ventilation System turned ON";
}

void Start_CoolingS() {
    digitalWrite(CoolingS, HIGH);
    std::cout << "Cooling System turned ON" << std::endl;
    CoolingS_status = "Cooling System turned ON";
}

void VentilationS_Timer(int duration = 1) {
    std::this_thread::sleep_for(std::chrono::minutes(duration));
    Start_VentilationS();
```

```
}

void CoolingS_Timer(int duration = 1) {
    std::this_thread::sleep_for(std::chrono::minutes(duration));
    Start_CoolingS();
}

int main() {

    std::thread Ctimer;
    std::thread Vtimer;

    if (wiringPiSetup() == -1) {
        return 1;
    }

    if (wiringPiSetupGpio() == -1) {
        std::cout << "Failed to initialize wiringPi GPIO." <<
std::endl;
        return 1;
    }

    pinMode(CoolingS, OUTPUT);

    while (true) {
        SensorData SD;
        int result = readDHTSensor(SD.humidity, SD.temperature);

        if (result == 0) {
            std::cout << "Temperature: " << SD.temperature << "°C,
Humidity: " << SD.humidity << "%" << std::endl;

            if (SD.temperature > 25) {
                try {
                    Ctimer = std::thread(CoolingS_Timer, 1); // Start
the timer thread with 1 minute duration
                    Ctimer.detach(); // Detach the timer thread,
allowing it to run independently
                }
                catch (const std::exception& e) {
                    std::cout << "Error starting the timer: " <<
e.what() << std::endl;
                }
            }
            else {

                if (Ctimer.joinable()) {
```

```
                    Ctimer.join();  // Stop the timer by waiting for it
to finish
                    std::cout << "Join Secccess" << std::endl;
                    digitalWrite(CoolingS, LOW);
                    std::cout << "Cooling System turned OFF" <<
std::endl;
                    CoolingS_status = "Cooling System turned OFF";
                }
                else {
                    digitalWrite(CoolingS, LOW);
                    std::cout << "Cooling System turned OFF" <<
std::endl;
                    std::cout << "Error stopping the timer: Timer
thread is not joinable." << std::endl;
                    CoolingS_status = "Cooling System turned OFF";
                }

            }

            if (SD.humidity > 60) {
                try {
                    Vtimer = std::thread(VentilationS_Timer, 1); //
Start the timer thread with 1 minute duration
                    Vtimer.detach(); // Detach the timer thread,
allowing it to run independently
                }
                catch (const std::exception& e) {
                    std::cout << "Error starting the timer: " <<
e.what() << std::endl;
                }
            }
            else {
                if (Vtimer.joinable()) {
                    Vtimer.join();  // Stop the timer by waiting for it
to finish
                    std::cout << "Join Secccess" << std::endl;
                    digitalWrite(VentilationS, LOW);
                    std::cout << "Ventilation System turned OFF" <<
std::endl;
                    VentilationS_status = "Ventilation System turned
OFF";

                }
                else {
                    digitalWrite(VentilationS, LOW);
                    std::cout << "Ventilation System turned OFF" <<
std::endl;
                    std::cout << "Error stopping the timer: Timer
thread is not joinable." << std::endl;
```

```
                    VentilationS_status = "Ventilation System turned
OFF";
                }
            }

            // Create a JSON object to store the data
            Json::Value jsonData;
            jsonData["temperature"] = SD.temperature;
            jsonData["humidity"] = SD.humidity;
            jsonData["CoolingS_status"] = CoolingS_status;
            jsonData["VentilationS_status"] = VentilationS_status;

            // Open the JSON file for writing
            std::ofstream jsonFile(JSON_FILE);
            if (!jsonFile.is_open()) {
                std::cout << "Failed to open JSON file for writing." <<
std::endl;
                return 1;
            }

            // Write the JSON data to the file
            jsonFile << jsonData;
            jsonFile.close();
        }
        else {
            std::cout << "Failed to read data from DHT sensor" <<
std::endl;
        }

        delay(2000);
    }

    return 0;
}
//g++ -o main main.cpp -lwiringPi -ljsoncpp
//chmod o+r file_name
```

Here is a breakdown of the C++ code:

1. The necessary header files are included: wiringPi.h for accessing Raspberry Pi GPIO pins, iostream for input/output operations, fstream for file operations, and json/json.h for JSON handling.
2. Constants are defined for the GPIO pin numbers (DHT_PIN for DHT11 sensor and RELAY_PIN for the relay) and the file path for the JSON file (JSON_FILE).
3. The readDHTSensor function is defined to read data from the DHT11 sensor. It uses the WiringPi library functions to interact with the GPIO pins. The sensor data is stored in the humidity and temperature variables. The function returns 0 on success or -1 on failure.
4. The main function is the entry point of the program.
5. It initializes the WiringPi library using wiringPiSetup and wiringPiSetupGpio functions. If the initialization fails, the program exits with a return value of 1.
6. The GPIO pin mode for the relay pin is set as an output using pinMode.
7. Inside an infinite loop, the temperature and humidity readings are obtained by calling the readDHTSensor function.
8. If the sensor reading is successful (result == 0), the temperature and humidity values are printed to the console.
9. If the temperature is greater than 25 degrees Celsius, the relay is turned on by setting the relay pin to HIGH

(digitalWrite(RELAY_PIN, HIGH)). Otherwise, the relay is turned off (digitalWrite(RELAY_PIN, LOW)).

10. A JSON object jsonData is created to store the temperature and humidity values.
11. The JSON file is opened for writing using an ofstream object. If the file fails to open, an error message is printed, and the program exits with a return value of 1.
12. The JSON data is written to the file using the << operator on the jsonFile object. The file is then closed.
13. If the sensor reading fails (result != 0), an error message is printed to the console.
14. A delay of 2000 milliseconds (2 seconds) is added before the next iteration of the loop.
15. The program returns 0 at the end of the main function.

HTML Page Code

```
<!DOCTYPE html>
<html>
<head>
    <title>Temperature and Humidity</title>
</head>
<body>
    <h1>Temperature and Humidity Data</h1>
    <?php
    // Read the JSON file
    $jsonString = file_get_contents('data.json');

    // Convert the JSON string to an associative array
    $data = json_decode($jsonString, true);

    // Check if the JSON decoding was successful
    if ($data !== null) {
        // Extract temperature and humidity values
        $temperature = $data['temperature'];
        $humidity = $data['humidity'];

        // Print the values on the HTML page
        echo "<p>Temperature: $temperature °C</p>";
        echo "<p>Humidity: $humidity %</p>";
    } else {
        echo "<p>Failed to retrieve data from JSON file.</p>";
    }
    ?>
</body>
</html>
```

Here is a breakdown of the HTML page code:

1. The HTML page structure is defined within the <html>, <head>, and <body> tags.
2. The title of the page is set to "Temperature and Humidity" using the <title> tag.
3. The <h1> tag is used to display the heading "Temperature and Humidity Data" on the page.
4. The PHP code block is started using <?php.
5. The file_get_contents function is used to read the contents of the data.json file and store it in the $jsonString variable.
6. The json_decode function is used to decode the JSON string into an associative array. The true argument is passed to ensure that the result is returned as an array.
7. The code checks if the JSON decoding was successful by comparing the decoded data ($data) to null.
8. If the decoding was successful, the temperature and humidity values are extracted from the $data array.
9. The temperature and humidity values are printed on the HTML page using the echo statement. The values are embedded within the HTML tags to be displayed as paragraphs (<p>).
10. If the decoding fails ($data is null), an error message is printed on the HTML page.
11. The PHP code block is closed using.>?
12. The HTML page is closed with the closing </body> and </html> tags.

What we can add:

- We can also add a safety period before giving the order to turn on or off the cooling system.
- We can make the HTML page more dynamic by making it update the readings taken from the json file every three seconds by adding some Java script code, without the need to refresh the page manually.
- We can also add a ventilation system control approach in order to reduce the degree of humidity.
- And also, we can add more and more features.

Flow Chart For Cooling System Control Project