Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Drahtlose Netzwerke

-

PROJEKTARBEIT

Temperatur abhängige Lüfter Steuerung mit Web Interface

Maximilian Scheibke (552315)

Betreuender Dozent:

Prof. Dr. Alexander Huhn

Inhaltsverzeichnis

1. Aufgabenstellung

Dieses Projekt befasst sich damit eine Aufgabe im Bereich das Internet der Dinge zu lösen. Es handelt sich dabei um eine Lüfter Steuerung die abhängig von einer gemessenen Temperatur, durch einen Temperatur-Sensor, gesteuert wird. Außerdem kann der Lüfter über ein Web Interface an- oder ausgeschaltet werden.

2. Inventar

In diesem Projekt werden folgende Dinge verwendet:

- Arduino Mega 2560
- ESP8266 – ESP01
- DHT11 Sensor
- einen Ventilator
- eine 9V Batterie
- HL-54S v1.0 – 4 Relais Modul

Der Arduino ist das Herzstück, das den Großteil des Codes bereitstellt und mit den übrigen Teilen verdrahtet ist. Das ESP ist ein Modul, das einen kleinen Mikrocontroller mit einem 802.11 Wifi Sender integriert. Dieses Modul enthält den verbleibenden Code und ist das Webinterface für dieses System. Die Temperatur wird über den DHT11 Sensor gemessen. Wenn die Temperatur zu hoch ist, wird das Relais geschaltet, wodurch wiederum der Stromkreis zwischen der Batterie und dem Lüfter geschlossen wird. Das Relais kann auch an mehrere Stromkreise angeschlossen werden. In diesem Projekt wird nur ein Lüfter verwendet, um es einfach zu halten.

3. Module und Werkzeuge

3.1 ESP8266

Das ESP Modul ist ein Low-Power Mikrocontroller einer chinesischen Firma espressif. Das Modul besteht aus einem 32-bit-Prozessor, einen 64 kB RAM Befehlsspeicher, sowie einem 96 kB Datenspeicher und integriertem WLAN. Wegen dem geringen Stromverbrauch, wird dieses Modul im IoT Bereich verwendet. Es werden mehrere Firmware-Varianten unterstützt. Die NodeMCU Firmware unterstützt eine interaktive Programmierung. Das heißt Programme werden im Flash-Speicher abgelegt. Durch NodeMCU wird ein Dateisystem auf dem ESP angelegt. Dadurch können die Lua-Skripts die über die NodeMCU laufen, hoch- und runtergeladen werden. Ist eine init.lua Datei vorhanden, wird diese immer zum Starten verwendet.

3.1.1 ESPTool

Das ESPTool ist ein Python-basiertes, Werkzeug, welches dafür verwendet wird um mit dem ROM bootloader des ESP Chips zu kommunizieren. Es wird dafür verwendet eine neue Firmware auf den ESP zu flashen.

3.1.2 ESPlorer

Der ESPlorer ist eine IDE für das ESP Modul. In der IDE können Skripts geschrieben, hoch- und runtergeladen werden. Zurzeit werden mehrere Firmwares unterstützt: NodeMCU, MicroPython, die AT-Firmware und die Frankenstein Firmware. Für den Schnellzugriff können Befehle auf Snippets gespeichert werden.

3.2 Arduino Mega 2560

Die Arduino Plattform besteht aus einer Hardware und einer Software Komponente. Die Hardware Komponente setzt sich aus einem Mikrocontroller und analogen und digitalen Ein- und Ausgängen zusammen. Die Software besteht aus einer Entwicklungsumgebung, die auch technisch weniger Versierten den Zugang zur Programmierung und zu Mikrocontrollern erleichtern. Die Programmierung erfolgt in einer C bzw. C++ ähnlichen Programmiersprache.

3.3 DHT11 Sensor

Der DHT11 Sensor ist eine Temperatur und Luftfeuchtigkeitssensor. Der Sensor übermittelt die Daten digital. Der Sensor ist auch auf längerer Zeit stabil und ist kostengünstig.

3.4 Relais Modul

Ein Relais ist ein elektrisch betreibender, fernbetätigter Schalter. Das Relais wird über einen Steuerstromkreis aktiviert und kann weitere Stromkreise schalten.

4. Implementation

Das ESP ist der einzige Teil dieses Systems, der programmiert werden muss, aber keine USB-Schnittstelle hat. Um das ESP-Modul zu programmieren, benötigt man ein Betriebssystem. Die NodeMCU ist eine Open-Source-Plattform und enthält eine Firmware, die auf dem ESP-Modul läuft.

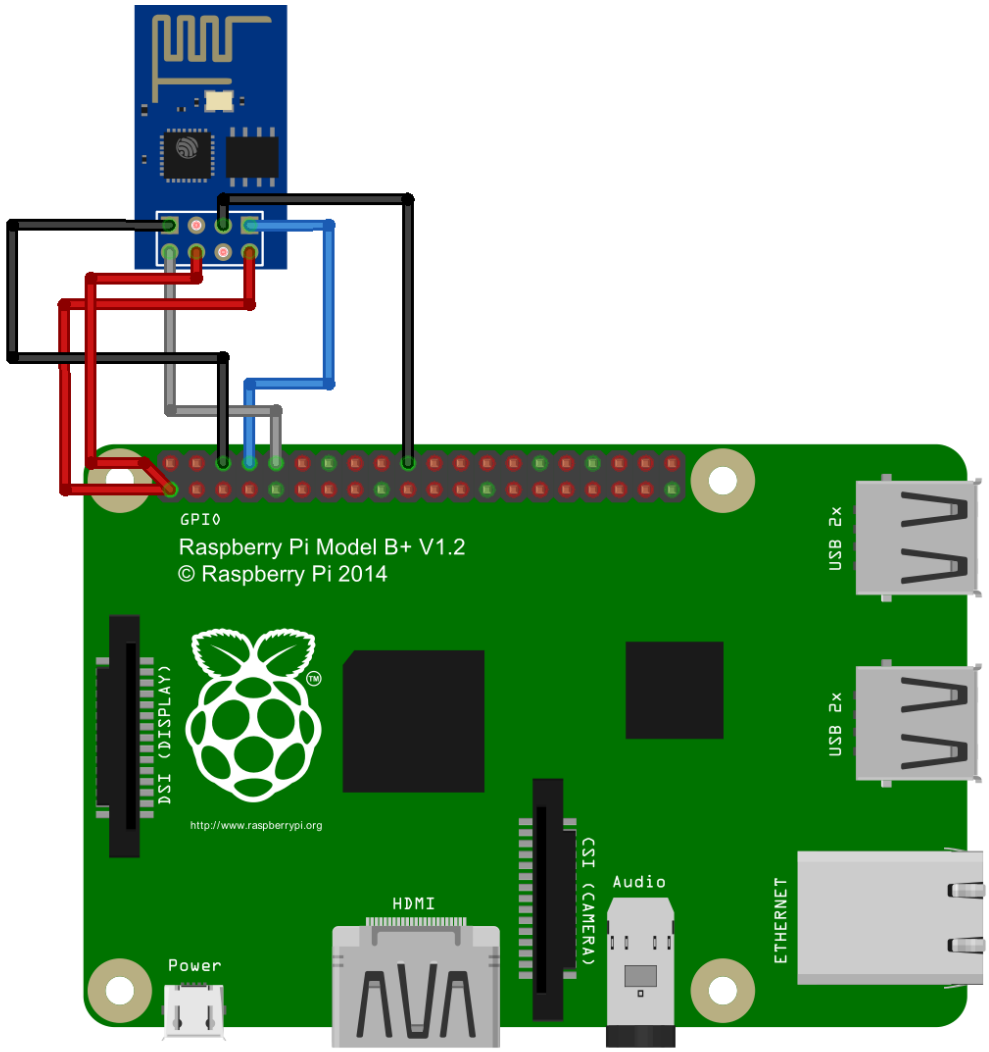
4.1 ESP Modul flashen

Eine Möglichkeit, die NodeMCU-Firmware auf dem Modul zu flashen, ist durch die Verwendung eines Raspberry Pi. Um die Firmware zu aktualisieren, wird ein Tool namens ESPTool benötigt. Das Python-Programm führt das Update basierend auf einer Binärdatei durch. Bevor das ESPTool verwendet werden kann, wird vorher die serielle Python-Bibliothek benötigt:

```
pi@raspberrypi ~ $ sudo apt-get install python-serial
```

Nach dem runterladen und entpacken von ESPTool, wird nun ein Image von NodeMCU benötigt. Die Binärdatei muss sich im Ordner ESPtool-master befinden. Jetzt kommt die Verkabelung zwischen dem ESP-Modul und dem Raspberry Pi.

4.1.1 Verkabelung zwischen Raspberry Pi und ESP



Raspberry Pi	ESP	Beschreibung
01; 3.3v	VCC	Spannung
01; 3.3v	enable	verantwortlich für die Aktivierung des Senders
06; ground	GND	Grund
08; TX	RX	Sender zu Empfänger
10; RX	TX	Empfänger zu Sender
20; ground	gpio0	muss low sein, um in den Flash-Modus zu gelangen

Nachdem das ESP mit dem Raspberry Pi verkabelt ist, ist alles bereit, die Firmware zu flashen. Dafür ist folgender Befehl nötig:

```
pi@raspberrypi ~ $ sudo python esptool.py --port /dev/ttyAMA0 write_flash 0x00000 nodemcu-master-7-modules-2018-03-27-13-49-36-float.bin
```

Wenn es erfolgreich war, wird das im Terminal auch angezeigt werden. Das Modul kann vom Strom genommen werden indem das Kabel von der Spannung abgezogen wird. Um nicht beim nächsten Start wieder in den Flash Modus zu gelangen, muss das Kabel zwischen ground und gpio0 entfernt werden. Nun kann das Modul wieder mit Spannung versorgt werden.

4.2 ESP über NodeMCU programmieren

Ein weiteres Werkzeug für den ESP ist der ESPlorer. ESPlorer ist eine IDE für die Entwicklung auf dem ESP. Über den ESPlorer können Lua-Dateien geschrieben und auf das ESP hochgeladen werden. Durch das Öffnen des Ports zeigt ESPlorer die Firmware-Spezifikationen an. Wenn das ESP eingeschaltet wird, sucht die Firmeware nach der Datei init.lua und führt diese Datei aus.

init.lua

```
function checkWiFiStatus()
    local s = wifi.sta.status()
    print("WiFi Status: " .. s)
    if s == 5 then
        tmr.stop(0)
        print("Connected on " .. wifi.sta.getip())
        print("Testoutput Check WIFI")
        dofile("runWebServer.lua")
    end
end

wifi.setmode(wifi.STATION)
station_cfg={}
station_cfg.ssid="WiFiSSID"
station_cfg.pwd="password"
wifi.sta.config(station_cfg)
print('Verbindungsversuch')
wifi.sta.connect()
tmr.alarm(0, 1000, 1, checkWiFiStatus)
```

Dieses Skript versucht, eine Verbindung zu einem WLAN mit der angegebenen SSID und dem angegebenen Kennwort herzustellen. War die Verbindung erfolgreich, wird die nächste Datei (runWebServer.lua) ausgeführt.

runWebServer.lua

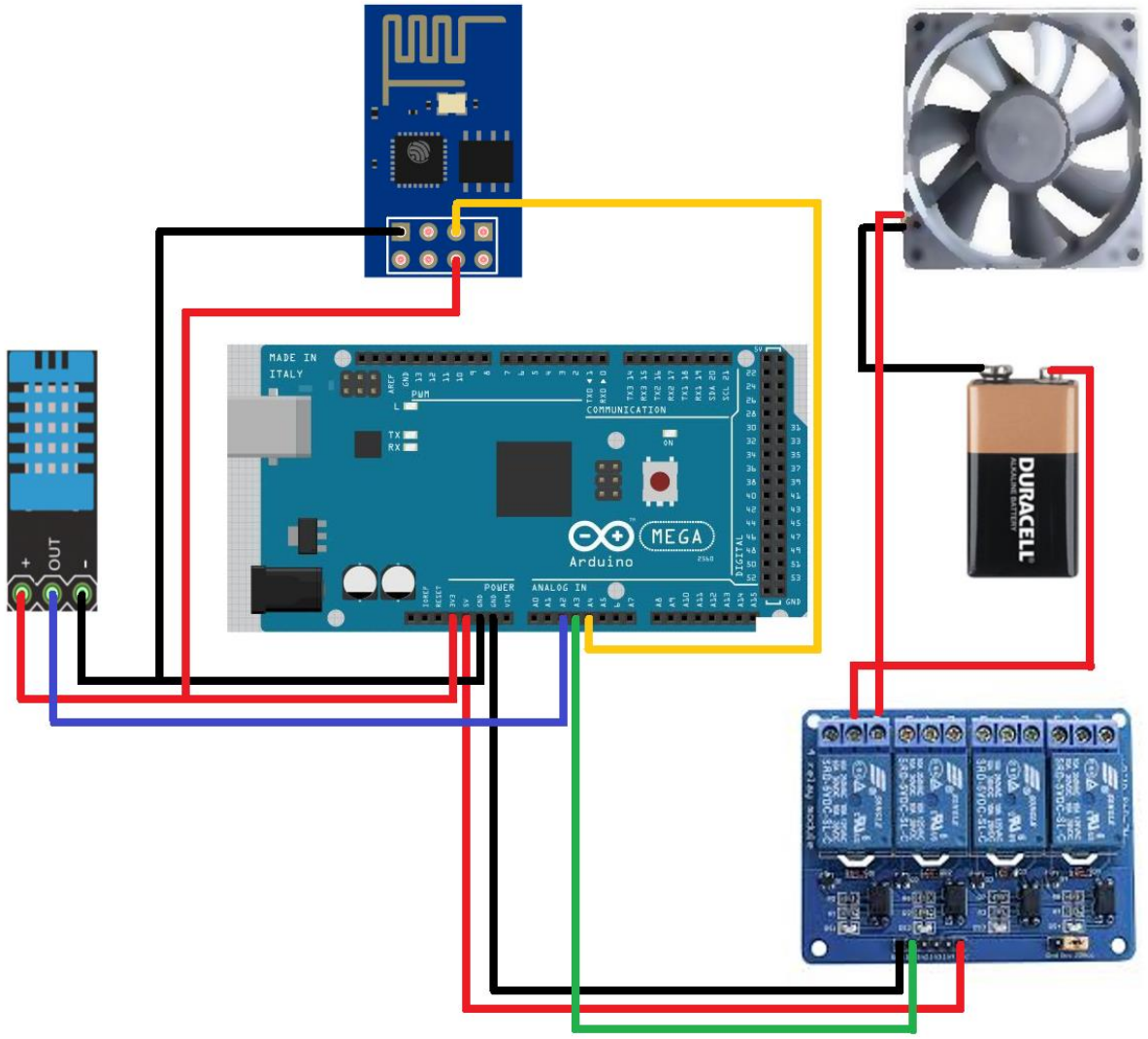
```
print("Webserver starts")
gpio.mode(4, gpio.OUTPUT)
gpio.write(4, gpio.LOW)
print("WIFI Status before Server start, WIFI Status
"..wifi.sta.status())
srv=net.createServer(net.TCP, 3)

print("Server created on " .. wifi.sta.getip())

srv:listen(80,function(conn)
    conn:on("receive",function(conn,request)
        print(request)
        _, j = string.find(request, 'control_switch=')
        if j ~= nil then
            command = string.sub(request, j + 1)
            if command == 'on' then
                gpio.write(4, gpio.HIGH)
                    else
                gpio.write(4, gpio.LOW)
            end
        end
        conn:send('<!DOCTYPE html>')
        conn:send('<html lang="en">')
        conn:send('<head><meta charset="utf-8" />')
        conn:send('<title>ESP Server</title></head>')
        conn:send('<body><h1>Fan Control Web Interface</h1>')
        -- Send the current status of the Fan Control unit
        if gpio.read(4) == gpio.HIGH then
            fan = "ON"
        else
            fan = "OFF"
        end
        conn:send('<p> Status of the fan unit : ' .. fan .. '</p>')
        conn:send('<form method="post">')
        conn:send('<input type="radio" name="control_switch"
value="on">ON</input><br />')
        conn:send('<input type="radio" name="control_switch"
value="off">OFF</input><br />')
        conn:send('<input type="submit" value="Fan Control Switch"
/>')
        conn:send('</form></body></html>')
    end)
end)
```

Im RunWebServer-Skript startet das Modul einen Web-Server. Es enthält eine kleine Schnittstelle, in der die Leistungssteuerung des Lüfters ein- oder ausgeschaltet werden kann. Diese Funktion wird durch einen Pin realisiert. Der Pin gpio0 vom ESP ist mit dem Arduino-Board verbunden. Wenn der Pin high ist und die Temperaturobergrenze von 30°C erreicht ist, wird der Lüfter angeschaltet. Wenn der Pin low ist, wird der Lüfter unabhängig von der Temperatur des DHT11 ausgeschaltet. Nach dem Hochladen beider Skripte auf das ESP kann das ESP vom Raspberry Pi getrennt werden.

4.3 Verkabelung und Programmierung des Arduinos



Dies ist die Gesamtverdrahtung. Alles ist mit dem Arduino verbunden, außer dem Lüfter und der Batterie. Der DHT11-Sensor sendet seine Temperaturdaten an das Arduino-Board. Liegt die Temperatur über 30°C, wird das Relais geschaltet und der 2. Stromkreislauf mit der Batterie und dem Lüfter ist geschlossen. Wenn die Temperatur wieder unter 25°C sinkt, schaltet das Relais zurück und der Stromkreislauf ist nicht mehr geschlossen. Der DHT11-Sensor und das ESP-Modul benötigen eine Spannung von 3,3 V, während das Relais eine Spannung von 5 V benötigt.

TempSense.ino

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>

#define DHTPIN 2     // Pin which is connected to the DHT sensor.
#define RELAYPIN 3
#define ESPPIN 4

#define DHTTYPE DHT11     // DHT 11

DHT_Unified dht(DHTPIN, DHTTYPE);

uint32_t delayMS;

uint32_t upperLimit = 30;
uint32_t lowerlimit = 25;
bool powerStatus = false;

void setup() {
  Serial.begin(9600);
  // Initialize device.
  dht.begin();
  // Print temperature sensor details.
  sensor_t sensor;
  dht.temperature().getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.println("Temperature");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
  Serial.print  ("Unique ID:    ");
Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value);
Serial.println(" *C");
  Serial.print  ("Min Value:    "); Serial.print(sensor.min_value);
Serial.println(" *C");
  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution);
Serial.println(" *C");
  Serial.println("------------------------------------");
  // Set delay between sensor readings based on sensor details.
  delayMS = sensor.min_delay / 1000;

  // Initilize relay Output
  pinMode(RELAYPIN, OUTPUT);
  digitalWrite(RELAYPIN, HIGH);

  // Initilize ESP INPUT
  pinMode(ESPPIN, INPUT);
}
```

```
void loop() {
  // Delay between measurements.
  delay(delayMS);
  // Get temperature event and print its value.
  sensors_event_t event;
  dht.temperature().getEvent(&event);
  if (isnan(event.temperature)) {
    Serial.println("Error reading temperature!");
  }
  else {
    uint32_t temperature = event.temperature;
    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.println(" *C");

    if(temperature < lowerlimit){
      powerStatus = false;
    } else if(temperature > upperLimit){
      powerStatus = true;
    }
  }

  // turn Relay on when dht11 Temperature is higher then 25
  if(powerStatus && digitalRead(ESPPIN) == HIGH){
    // relay Pin has to be low if relay has to switch electrical
ports
    digitalWrite(RELAYPIN, LOW);
  } else {
    digitalWrite(RELAYPIN, HIGH);
  }
}
```

Der Arduino wird über eine USB Schnittstelle an einem Computer verbunden. Über die Arduino-IDE können die Sketche geschrieben und auf die Hardware hochgeladen werden.

5. Abschluss/Ausblick

Das Ergebnis dieses Projekts ist ein funktionierendes Kühlsystem mit einem Web Interface. Das System ist eher klein und kann daher noch weiter ausgebaut werden. Die eigentliche Kühlung besteht aus einem Lüfter der von einer 9V Batterie angetrieben wird. Ein anderes Kühlsystem könnte effektiver sein als ein Lüfter.
In diesem Projekt wurde das Ein- und Ausschalten über das Web Interface über einem Pin vom ESP Modul zum Arduino realisiert. Es kann durchaus auch die UART Schnittstelle wie sie bei dem Raspberry Pi verwendet wurde, benutzt werden und es könnte so auch die Temperatur auf der Web Seite zu sehen sein.

6. Quellenverzeichnis

[1] ESP8266: https://www.espressif.com/en/products/hardware/esp8266ex/overview

[2] ESPlorer: https://esp8266.ru/esplorer/

[3] ESPTool: https://github.com/espressif/esptool

[4] NodeMCU: http://www.nodemcu.com/index_en.html

[5] NodeMCU Dokumentation: https://nodemcu.readthedocs.io/en/latest/

[6] Relais: https://de.wikipedia.org/wiki/Relais

[7] ESP8266 Tutorial: https://www.hackster.io/waltercoan/esp8266-nodemcu-integrated-with-mobile-services-iot-azure-4bf52a

[8] UART: http://kampis-elektroecke.de/?page_id=1682

[9] DHT11: https://funduino.de/anleitung-dht11-dht22

[10] Arduino Mega 2560: https://store.arduino.cc/arduino-mega-2560-rev3