Mikrocomputertechnik-Praktikum

Webbasierte Steuerung einer Klimaanlage

WiSe 24/25

Sebastian Pasinski, Benedikt Schnörr

Prüfer: Prof. Dr. Francesco Volpe

Inhalt

Abbildungen

1 Anforderungen

In diesem Praktikum wird eine Steuerungseinrichtung für eine Klimaanlage des Typs Remko MKT 251 realisiert. Der Benutzer kann über eine Weboberfläche Eingaben für die Steuerung der Klimaanlage tätigen. Die Web-GUI und Benutzereingaben sollen über einen Server, der auf einem Mikrocontroller läuft, verwaltet werden. Dazu wird ein ESP32 verwendet, der mit dem Benutzer über die Weboberfläche kommuniziert und durch Infrarot-Licht-Signale nach dem NEC-Protokoll Befehle an die Klimaanlage sendet. Damit kann die Klimaanlage zu festen Uhrzeiten, die der Benutzer frei definieren kann, ein- und ausgeschaltet werden. Die Uhrzeit, an der sich der Mikrocontroller orientiert, soll dabei synchron zur realen mitteleuropäischen Zeit laufen und immer wieder mit dieser synchronisiert werden. Das System muss dabei im ständigen Einsatz und immer bereit sein, einen neuen Zeitplan für die Sendezeiten der Befehle zu erhalten.

2 Blockschaltbild

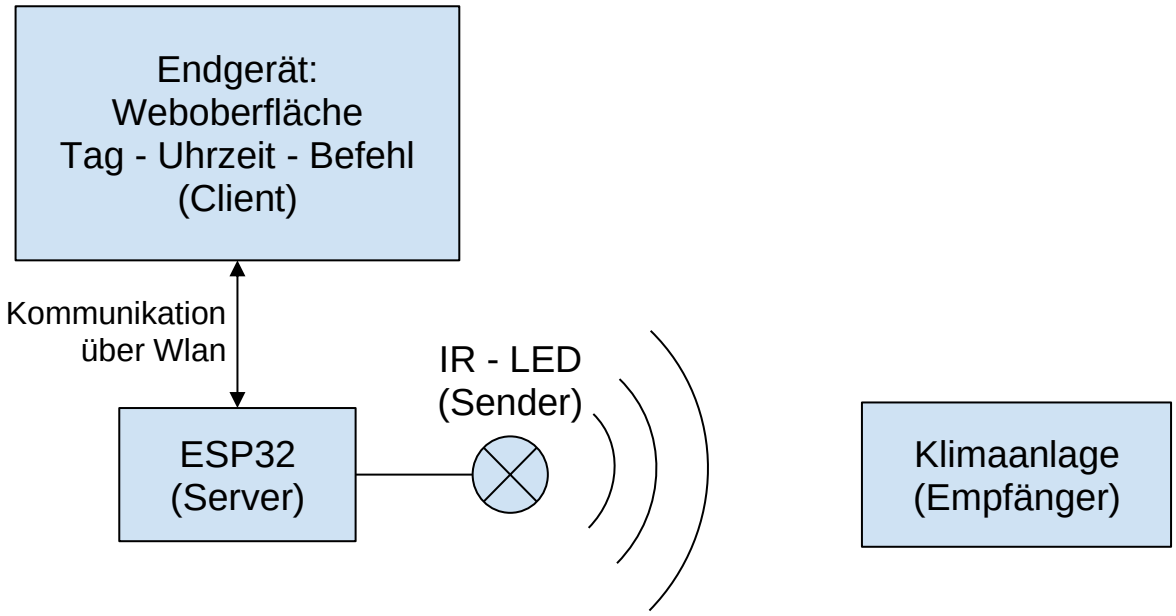

Abbildung 1: Blockschaltbild zur Veranschaulichung des Aufbaus

In *Abbildung 1* wird der Aufbau der Fernbedienung dargestellt. Die Weboberfläche zur Steuerung der Anlage kann von einem beliebigen Endgerät aus aufgerufen werden. Nachdem der Benutzer Eingaben auf der Webseite getätigt hat, werden die eingestellten Daten über Wlan an den Mikrocontroller gesendet. Die übermittelten Parameter enthalten dabei für jeden Befehl die Wochentage, an denen der Befehl gesendet werden soll, und die Uhrzeit für das Senden des Befehls. Außerdem wird auch die aktuelle Uhrzeit übermittelt, um den ESP beim Einstellen von neuen Befehlszeiten mit der tatsächlichen Zeit zu synchronisieren. Der Mikrocontroller steuert dann über einen Ausgangspin zu den eingestellten Zeiten eine Infrarot-LED an, die Signale an die Klimaanlage sendet, um deren Zustand entsprechend zu verändern.

3 Weboberfläche

3.1 Konzept

Zum Ein- und Ausschalten der Anlage bedarf es einer webbasierten grafischen Benutzeroberfläche. Beim Entwurf wird zunächst ein Wireframe erstellt, welches in Abbildung 2 zu sehen ist. Das Wireframe ist ein grobes Modell der Webseite, welches dem Benutzer kommunizieren soll, wie er benötigte Informationen erhält oder Einstellungen tätigen kann. In der Benutzeroberfläche können die Zeiten und Tage zum An- und Abschalten der Klimaanlage ausgewählt werden. Die Tage und Uhrzeiten können dann über einen Button an den ESP gesendet werden.

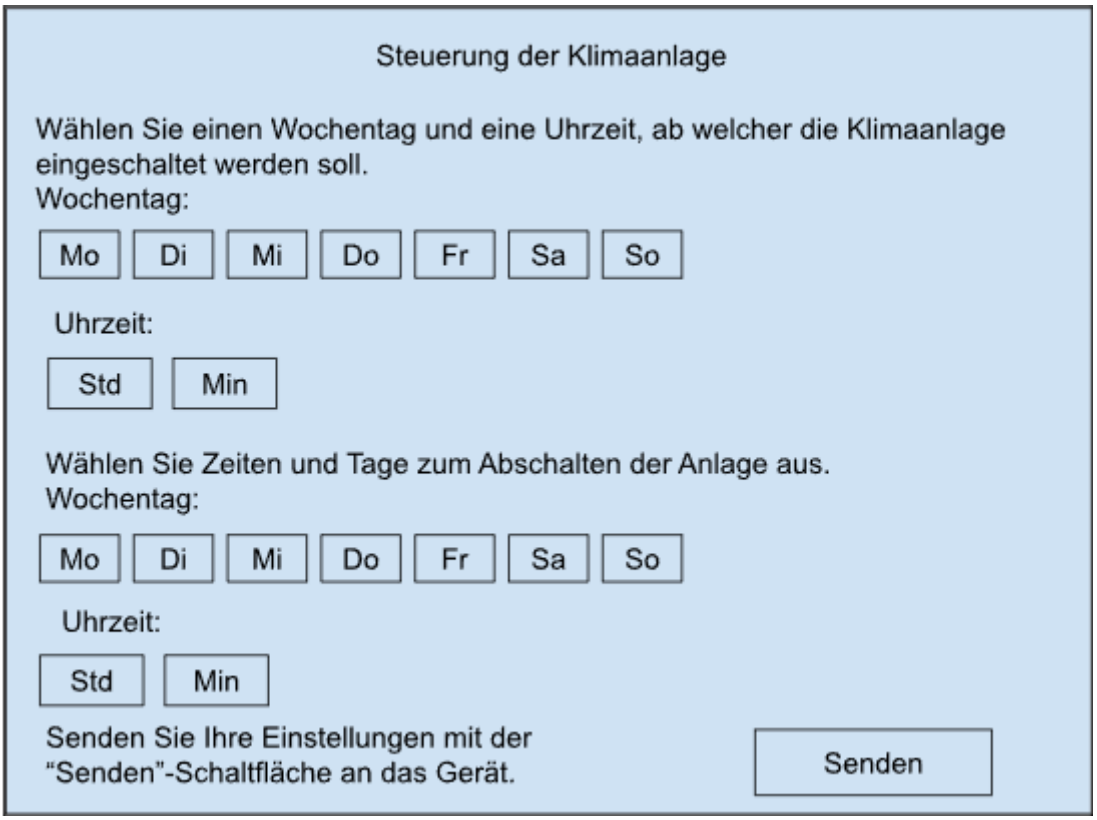

Abbildung 2: Wireframe zur Planung der Weboberfläche

Das Wireframe wurde in HTML implementiert und nach Rücksprache mit dem Labor angepasst. Die fertige Weboberfläche ist in Abbildung 3 zu sehen. Sie enthält, genau wie das Wireframe, Eingabemöglichkeiten für Wochentage und Uhrzeiten zum An- und Abschalten der Klimaanlage. Für jede Funktion steht ein eigener Sende-Button zur Verfügung, mit dem die Einstellungen für jeden Befehl einzeln verändert werden können. Außerdem gibt es einen dritten Abschnitt, in dem in Zukunft ein weiterer Befehl eingefügt werden kann, um die Funktionalität der Fernbedienung zu erweitern.

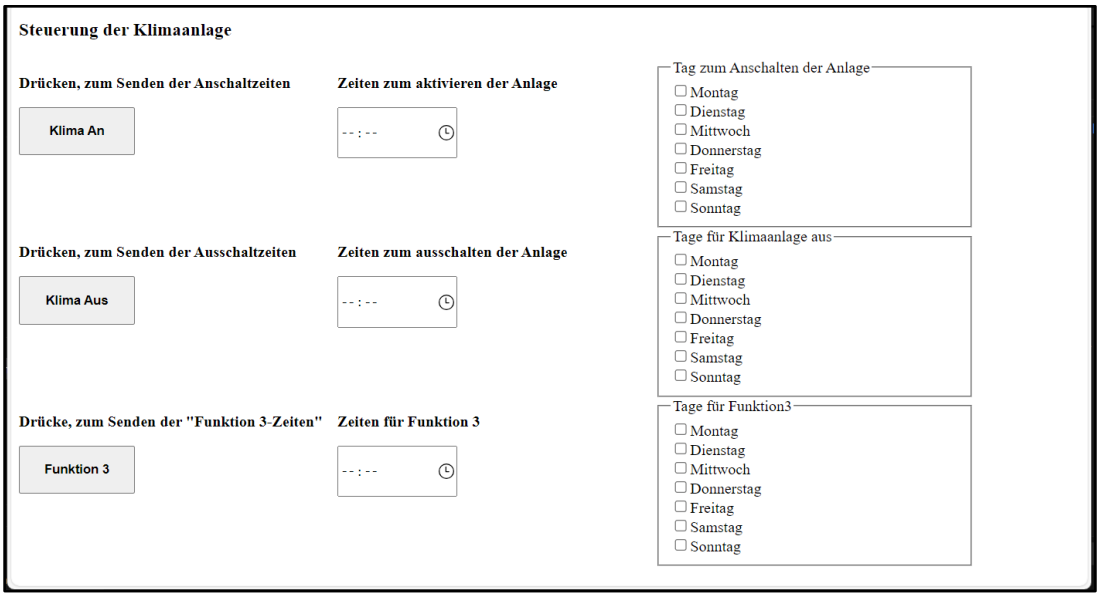

Abbildung 3: Umsetzung des Wireframes als Weboberfläche

3.2 Bedienelemente in html

Der Benutzer gibt die IP-Adresse des ESP32 in dessen erzeugtem Netzwerk in den Browser ein. Dadurch wird vom Browser eine Anfrage an den Mikrocontroller gesendet. Als Antwort wird das HTML-Dokument, welches die Weboberfläche enthält, dem Browser zugesendet und so dem Benutzer zur Verfügung gestellt.
Die Struktur der Weboberfläche ist im HTML-Dokument spezifiziert. Die Oberfläche wird in eine Überschrift und eine Fläche aus drei Spalten und drei Zeilen gleicher Größe unterteilt, in welchen die Komponenten als sections angeordnet werden.

```
<style type="text/css">
    section {
        float: left;
        width: 30%;
    }
    section2 {
        float: left;
        width: 30%;
    }
    section3 {
        float: left;
        width: 30%;
    }
    section4 {
        float: upper;
        height: 50%;
    }
</style>
```

In der linken Spalte befindet sich in jeder Zeile ein Button, der beim Klicken eine Funktion ausführt. Diese Funktion sendet die Daten der Benutzereingabe an den Server auf dem Mikrocontroller. [3]

```
<!-- sections to divide website -->
<section>
      <!-- title for button -->
      <p><b> Drücken, zum Senden der Anschaltzeiten </b></p>
      <!-- create and position button with on click event-->
      <p><button id="ButtonKlimaAn" style="width:120px; height:50px;
      left:NaNpx; top:NaNpx;" onclick=onclickKlimaAn()><b>Klima
      An</b></button></p>
</section>
```

In der mittleren Spalte wird in jeder Zeile ein Zeit-Eingabefeld „`input = time`" platziert, mit dem der Benutzer eine Uhrzeit für den jeweils auszuführenden Befehl angeben kann. [4]

```
<!-- sections to divide website -->
<section>
      <!-- title for input field -->
      <p> <label for="InputZeitKlimaAn"><b>Zeiten zum aktivieren der An-
      lage</b></label> </p>
      <!-- input field for time with hour and minute -->
      <p> <input type="time" id="InputZeitKlimaAn" name="TimeAcOn" re-
      quired style="width:120px;height:50px;"> </p>
</section>
```

In der rechten Spalte ist in jeder Zeile eine Anordnung an Checkboxen, welche es dem User ermöglicht, Wochentage für den jeweiligen Befehl auszuwählen. [5]

```
<!-- sections to divide website -->
<section>
      <!-- group input elements -->
      <fieldset>
            <legend>Tag zum Anschalten der Anlage</legend>
            <!-- create input elements with checkbox for each day of
            the week -->
            <span><div>
                  <input type="checkbox" id="Monday" Name="Mo" on-
                  click="Monday()"><label for="Monday">Montag</label>
            </div>
            <div>
                  <input type="checkbox" id="Tuesday" Name="Tue"><label
                  for="Tuesday">Dienstag</label>
            </div></span>
            <div>
```

```
                <input type="checkbox" id="Wednesday" Name="Wed"><la-
                bel for="Wednesday">Mittwoch</label>
            </div>
            <div>
                <input type="checkbox" id="Thursday" Name="Th"><label
                for="Thursday">Donnerstag</label>
            </div>
            <div>
                <input type="checkbox" id="Friday" Name="Fri"><label
                for="Friday">Freitag</label>
            </div>
            <div>
                <input type="checkbox" id="Saturday" Name="Sat"><label
                for="Saturday">Samstag</label>
            </div>
            <div>
                <input type="checkbox" id="Sunday" Name="Sun"><label
                for="Sunday">Sonntag</label>
            </div>
        </fieldset>
</section>
```

Die Bedienelemente sind hier beispielhaft für die Funktion “Anschalten” der Klimaanlage dargestellt. Im vollständigen Code werden zusätzlich Elemente für die Funktion “Ausschalten” und einen weiteren später einstellbaren Befehl definiert.

3.3 Übertragen der Parameter

Im Folgenden werden die Funktionen innerhalb der script-Tags im HTML-Dokument erklärt, welche die Eingaben des Benutzers versenden.
In der changingBoolToBit-Funktion wird der Zustand der Checkboxen mit den Wochentagen von einem boole’schen Wert in einen binären Wert umgewandelt.

```
function changingBoolToBit(IdElement){
    return document.getElementById(IdElement).checked ? 1 : 0;
}
```

Wenn ein Button geklickt wird, dann wird die jeweils spezifizierte onClick-Funktion ausgeführt, welche eine Instanz der XMLHTTPRequest-Klasse und eine Variable „data” erzeugt. Das XMLHttpRequest-Objekt bietet eine Schnittstelle für die Datenübertragung zwischen Client und Server, somit wird ermöglicht, dass Informationen aus der URL der Webseite entnommen werden. [2] Die data-Variable enthält Informationen über die Art des zu sendenden Befehls, die Wochentage und Uhrzeit zum Senden des Befehls und die aktuelle Uhrzeit zur Synchronisierung des ESP32.

```
function onclickKlimaAn() {
      /* create object for communicating with server */
      var xhttp = new XMLHttpRequest();
      /* store information to append to url including:
         - name of command
         - send time
         - 7 bit to indicate which weekdays are selected with ‘1’ and
           unselected with ‘0’ starting with sunday
         - current time and date for synchronising esp32
      */
      var data = "/setOn?";
      /* add time selected by user*/
      var InputTimeOn = document.getElementById("In-
      putZeitKlimaAn").value;
      data += "time=" + InputTimeOn;
      /* add 7 bits for weekdays */
      data += "&week=" + changingBoolToBit("Sunday") + chang-
      ingBoolToBit("Monday") + changingBoolToBit("Tuesday") + chang-
      ingBoolToBit("Wednesday") + changingBoolToBit("Thursday") + chang-
      ingBoolToBit("Friday") + changingBoolToBit("Saturday");
      /* add current time and date, e.g. “Wed Jan 08 2025 18:43:48
      GMT+0100 (Mitteleuropäische Normalzeit)” */
      var today = new Date();
      data += "&currentTime=" + today;
      /* store data variable for http get protocol and send request to
      server */
      xhttp.open("GET", data);
      xhttp.send();
}
```

4 Der ESP32 als Webserver

Der ESP32 hat zunächst zwei Aufgaben. Er muss die Einstellungen des Benutzers empfangen und verarbeiten und dann, abhängig von den eingestellten Sendezeiten, die IR-LED ansteuern. In den folgenden Abschnitten wird die Initialisierung des ESP32 beschrieben, sodass dieser mit einem Endgerät kommunizieren kann.

4.1 Übertragung der Weboberfläche auf den ESP32

Das Übertragen der ausgearbeiteten Weboberfläche von der Entwicklungsumgebung auf den ESP32 wird vom Tool SPIFFS übernommen. [10] Dabei wird auf dem ESP32 eine Datenstruktur implementiert, die bei einem http-Request des Clients angesprochen werden kann. In diesem Fall besteht die Dateistruktur einfach aus der Datei „index.html", die über die Arduino IDE auf den ESP hochgeladen wird.

4.2 Bereitstellen der Weboberfläche für den Client

Die Struktur zwischen ESP32 und Benutzer ist unkompliziert aufgebaut. Der ESP32 als Server stellt die Weboberfläche für den Benutzer bereit, während das Endgerät des Benutzers als Client Einstellungen vornehmen kann und diese dann an den Server zurücksendet.
Die Verbindung zwischen Server und Client funktioniert dabei über ein Netzwerk, das der ESP32 selbst erzeugt. Das Endgerät kann sich mit diesem Netzwerk verbinden und so als Client den ESP ansprechen. Der ESP32 wird bei dieser Methode also als Access Point programmiert. [1] Zum Herstellen der Verbindung wird die WiFi-Bibliothek genutzt, die für den Verbindungsaufbau nur den Namen und das Passwort des erzeugten Netzwerks benötigt.

```
// use libraries to provide access point
#include "WiFi.h"
#include "ESPAsyncWebSrv.h"
// set login credentials
const char* ssid = "ESP32-Klimaanlage";
const char* password = "12345";
// setup webserver library
AsyncWebServer server(80);

void setup() {
      Serial.begin(9600);
      // start access point with login credentials
      WiFi.softAP(ssid, password);
      server.begin();
}
```

Der ESP32 kann mit der ESPAsyncWebSrv-Bibliothek eine html-Weboberfläche an den Client senden. Dazu wird zunächst die IP-Adresse abgefragt, über die die Website vom Client beim Server angefragt werden kann.

```
// print ip address to call from mobile device
IPAddress IP = WiFi.softAPIP();
Serial.print("Access Point IP address: ");
Serial.println(IP);
```

Danach kann eine html-Seite festgelegt werden, die vom ESP32 als Server an den Browser als Client zurückgegeben wird. Diese wurde in Abschnitt 4.1 auf dem ESP abgelegt und kommt jetzt zum Einsatz.

```
// handle http request from mobile device by providing html webpage
server.on("/", HTTP_GET, [](AsyncWebServerRequest * request) {
      request->send(SPIFFS, "/index.html", "text/html");
});
```

4.3 Verarbeitung der empfangenen Parameter

Das Programm des ESP32 wird erweitert, um auf weitere URLs zu reagieren, die jeweils für einen Befehl stehen. Beim Aufrufen dieser URLs durch den Client wird wieder ein http-Request gesendet, welches der ESP empfängt. Die in der Weboberfläche eingestellten Parameter für Zeit und Wochentage der verschiedenen Befehle werden mit dem Http-Get-Protokoll an den ESP übermittelt. Dabei liegen die Zeiten und Wochentage zunächst als String-Variablen vor. Die Zeit enthält dabei vier Ziffern, je zwei für die Stunden und zwei für die Minuten, beispielsweise „1545" für die Uhrzeit 15:45 Uhr. Außerdem werden die Wochentage durch je eine Stelle pro Tag dargestellt, die jeweils den Wert „0" oder „1" haben können. Wenn beispielsweise montags, mittwochs und donnerstags ein Befehl an die Klimaanlage gesendet werden soll, hat die entsprechende Variable den Wert „0101100", da das Array mit dem Sonntag an Stelle 0 beginnt. Diese Struktur wird von der verarbeitenden Bibliothek auf dem ESP vorgegeben.
Die Werte werden für die weitere Verarbeitung im Programm in Structs gespeichert. Diese enthalten die Stunden und Minuten als `Integer`-Variablen und die Werte für die einzelnen Tage als `Integer`-Array mit sieben Stellen und jeweils einer „0" oder „1". Die Umwandlung der `String`-Werte in eine Instanz des Structs ist im folgenden Code-Ausschnitt dargestellt, der mit Betätigung der Schaltfläche für den jeweiligen Befehl ausgeführt wird.

```
// define struct for command parameters
struct CommandParam {
  int hours;
  int minutes;
  int weekdays[7]; // with sunday at index 0
};
struct CommandParam paramOn;
// handle request when button for command "on" is pressed
server.on("/setOn", HTTP_GET, [](AsyncWebServerRequest * request) {
      // get value for sending times
      String t = request->getParam("time")->value();
      // convert sending times from string to int
      paramOn.hours = t.substring(0, 2).toInt();
```

```
        paramOn.minutes = t.substring(2, 4).toInt();
        // get value for weekdays on which command should be sent
        String days = request->getParam("week")->value();
        // convert weekdays from strin to int array
        stringToIntArray(days, paramOn.weekdays);
        // serve webpage to setting up other commands
        request->send(SPIFFS, "/index.html", "text/html");
});
```

Neben der Sendezeit wird über das Get-Protokoll auch die reale Zeit an den ESP32 übergeben. So wird eine gleichbleibende Genauigkeit und Synchronisation zwischen der Zeit, die der ESP zum Ermitteln des Sendezeitpunkts und der globalen, realen Zeit sichergestellt. Außerdem werden damit Randfälle, wie beispielsweise die Zeitumstellung zwischen Winter- und Sommerzeit, abgefangen.
Die reale Zeit wird dabei, genau wie die Wochentage und Sendezeit, als String übermittelt. Die resultierende Variable hat beispielsweise den Wert „Tue Dec 17 2024 15:01:22 GMT+0100 (Mitteleuropäische Normalzeit)“ und besitzt damit alle Informationen, um die Zeit des ESP32 zu synchronisieren. Dieser String wird dann in einzelne Variablen zerteilt, um die aktuelle Zeit und das aktuelle Datum mit der setTime-Funktion an die Time-Bibliothek weitergeben zu können. So wird beim Einstellen der Sendezeiten jedes Befehls der ESP32 automatisch mit der realen Zeit synchronisiert. Der Code ist beispielhaft für einen einzelnen Befehl im nächsten Abschnitt dargestellt.

```
// define struct to store current time information
struct CurrentTime {
        int days;
        char months[4]; // month includes 3 letters and null terminator
        int years;
        int hours;
        int minutes;
        int seconds;
};
struct CurrentTime currentTime;
// extract current time from received parameters
String currTime = request->getParam("currentTime")->value();

// convert current time from string to const char ptr for setTime
function
const char* currTimeStr = currTime.c_str();

// split current time into usable parts and store inside struct
sscanf(currTimeStr, "%*s %3s %d %d %d:%d:%d %*s", &currentTime.months,
&currentTime.days, &currentTime.years, &currentTime.hours, &cur-
rentTime.minutes, &currentTime.seconds);

// configure time library with received current time
setTime(currentTime.hours, currentTime.minutes, currentTime.seconds,
currentTime.days, monthToInt(currentTime.months), currentTime.years);
```

5 Der ESP32 als IR-Sender

5.1 Das NEC-Protokoll der Klimaanlage

Bei der Signalanalyse der existierenden Fernbedienung für die Klimaanlage konnte festgestellt werden, dass das NEC-Protokoll verwendet wird. Dazu sind in den Abbildungen 4 und 5 die Struktur des NEC-Protokolls sowie die Signale beim An- und Ausschalten der Fernbedienung dargestellt.
Das Protokoll beginnt mit einem Startsignal, welches aus einem Puls mit 9 ms im HIGH-Zustand und einer LOW-Phase von 4,5 ms besteht. Danach werden die 32 Bits des Protokolls gesendet. Das Ende markiert ein weiterer Puls mit einer Länge von 562,5 µs. Die 32 gesendeten Bits bestehen aus je 8 Bits für die Adresse des Geräts und 8 Bits für das logisch Invertierte der Adresse. Zusätzlich wird der Befehl in 8 Bits und das logisch Invertierte des Befehls in ebenfalls 8 Bit übermittelt. Die Länge der LOW-Zeiten der einzelnen Bit-Pulse unterscheidet eine logische „1” mit 1,6875 ms von einer logischen „0” mit 562,5 µs. Die einzelnen Pulse der Bits werden von einer Carrier-Spannung mit einer Frequenz von 38,22 kHz gebildet. [7]

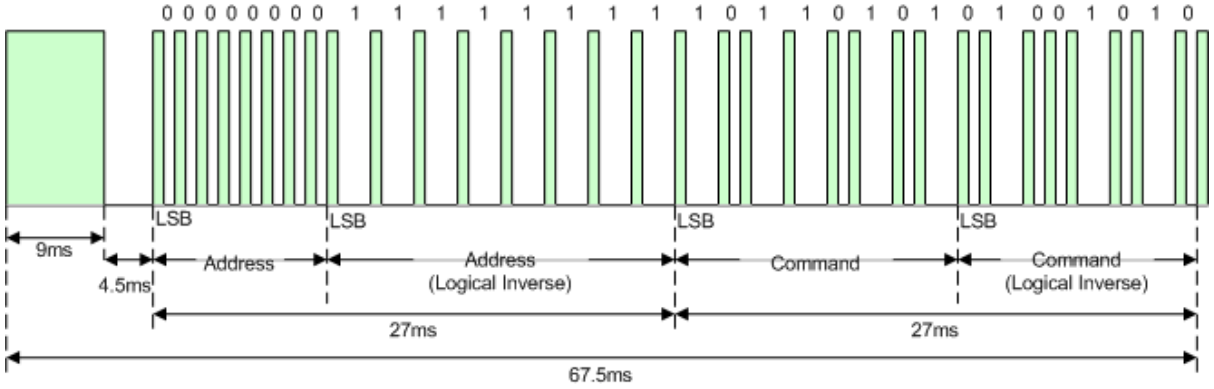

Abbildung 4: Aufbau des NEC-Protokolls zur Nachbildung mit dem ESP32 [7]

Bei der Analyse der Fernbedienung wurde festgestellt, dass das zweite und vierte Byte nicht aus der invertierten Adresse und dem invertierten Befehl bestehen. Es wird vermutet, dass die Fernbedienung ein firmeninternes erweitertes NEC-Protokoll unterstützt, welches dem zweiten und vierten Byte zusätzlichen Informationsgehalt über die Adresse und den Befehl verleiht. Daher ist das Ziel bei der Programmierung des ESP32, die einzelnen Bytes der aufgenommenen Befehle zu replizieren, sodass die Befehle genau so aussehen wie die der Fernbedienung.

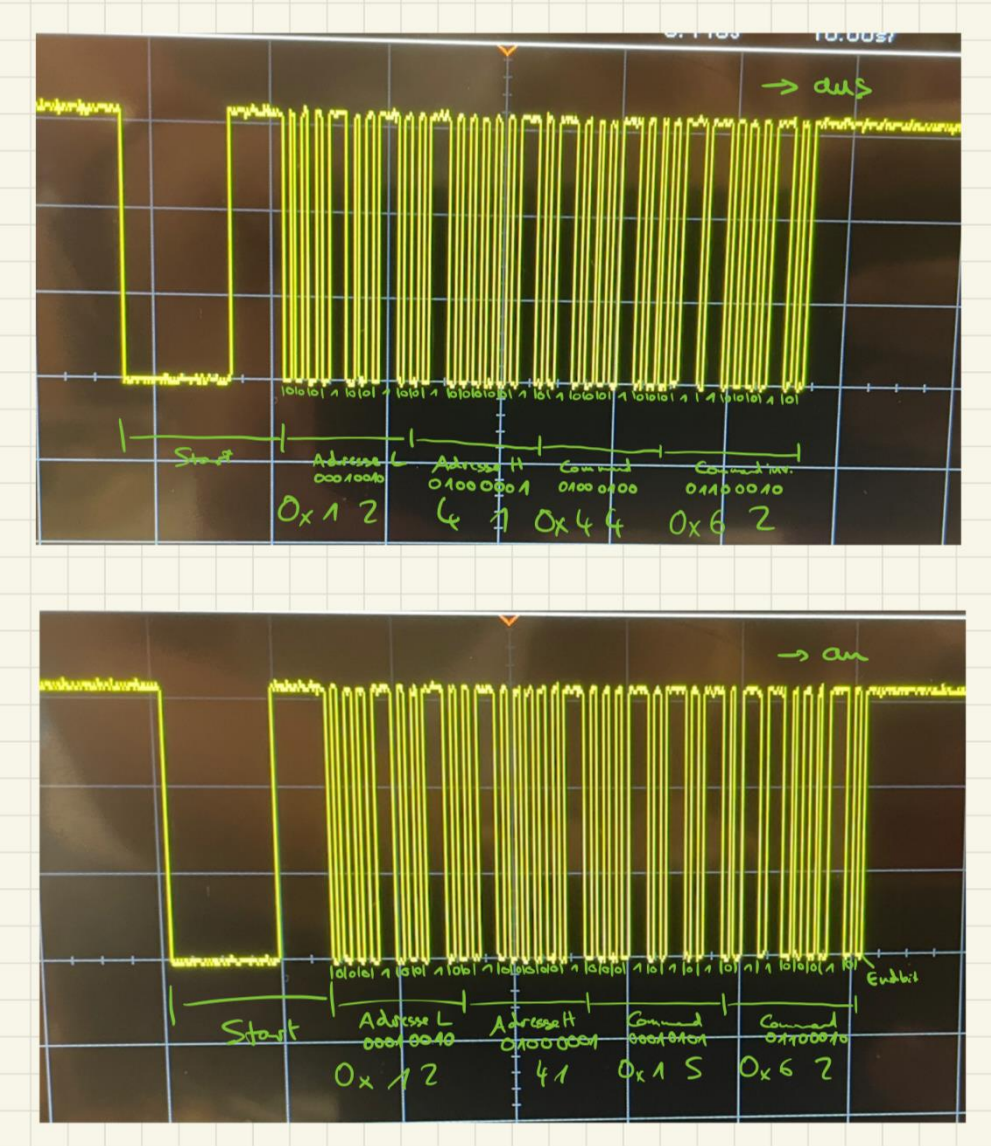

Abbildung 5: Signale der Fernbedienung zur Analyse des verwendeten Protokolls

5.2 Senden von NEC-Befehlen

Zum Senden der Befehle mit dem NEC-Protokoll wird eine Bibliothek benutzt, da dies die einfachste Möglichkeit ist, um die Vorgaben über die Pulslängen am Start und am Ende der Übertragung und über die Längen der HIGH- und LOW-Phasen der Bits umzusetzen. Dazu wird die Arduino-IRremote-Library verwendet. [8] Darin werden Funktionen definiert, mit denen Adressen und Befehle oder einfache Bitfolgen versendet werden können. Da in diesem Fall die Bits, die bei der Fernbedienung gesendet werden, nicht direkt mit dem standardisierten NEC-Protokoll übereinstimmen, wird eine Funktion verwendet, die die Bitfolge der Befehle von der Fernbedienung überträgt. Dabei werden die Bits vom LSB aus gesendet, daher müssen die Bitfolgen der Fernbedienung zunächst umgedreht werden, sodass das erste gesendete Bit der Funktion im Code mit dem der Fernbedienung übereinstimmt. Im folgenden Codeausschnitt ist das Senden eines NEC-Befehls mit der Funktion sendNecRAW aus der `IRremote`-Bibliothek von Armin Joachimsmeyer dargestellt. [8]

```
// use library for sending nec commands
#include <IRremote.hpp>

void setup() {
      // setup output pin for ir led
      pinMode(22, OUTPUT);
      Serial.begin(9600);
      // initialize sender inside irremote library with output pin
      IrSender.begin(22);
}

void loop() {
      // send on command once for every loop
      IrSender.sendNECRaw(0x46A88248, 1);
      // delay must be greater than 5 ms, otherwise the receiver sees
      it as one long signal
      delay(1000);
}
```

Beim Vergleich mit der existierenden Fernbedienung wurde deutlich, dass eine Taste teilweise öfter als einmal gedrückt werden muss, damit die Klimaanlage auf den Befehl reagiert. Das Senden der Befehle mit dem ESP funktionierte in den Tests zwar einwandfrei, im vollständigen Programm wird der Befehl aber trotzdem insgesamt dreimal nacheinander gesendet, um sicherzustellen, dass der Befehl beim Empfänger ankommt.

5.3 Sender-Schaltung mit der IR-LED

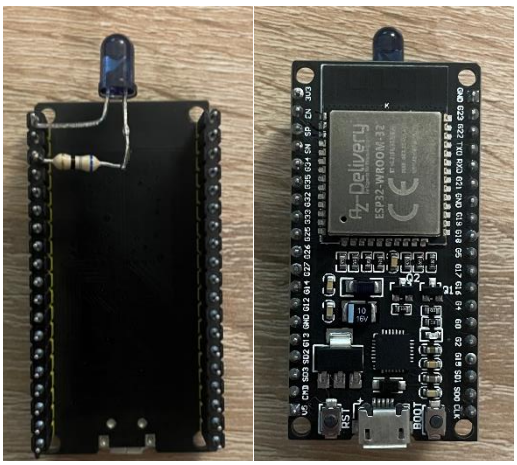

Abbildung 6: Sender-Schaltung des ESP mit IR-LED

Die Schaltung, die zum Senden der IR-Befehle benötigt wird, besteht aus einer einfachen IR-LED, die über einen Vorwiderstand an einen IO-Pin des ESP32 angeschlossen wird. In diesem Projekt wird die LED Vishay TSAL6200 verwendet. [9] Sie sendet Lichtstrahlen mit einer Wellenlänge von 940 nm aus. Die Berechnung des Vorwiderstands ergibt sich aus der Ausgangsspannung von $0{,}8 \cdot 3{,}3\ V = 2{,}64\ V$ und dem maximalen Ausgangsstrom von 40 mA des gewählten Output-Pins GPIO22 des ESP. [1] Mit dem benötigten Spannungsabfall von 1,35 V an der LED fällt am Vorwiderstand in Reihe eine Spannung von 1,29 V ab. Der benötigte Strom des Stromkreises wird hier auf 20 mA geschätzt und liegt somit weit unter dem maximalen Vorwärtsstrom der LED von 100 mA. [9] Der Widerstand berechnet sich somit aus der Spannung, die am Widerstand abfällt, und dem Strom im Stromkreis mit dem Ohm'schen Gesetz zu $\frac{1{,}29\ V}{20\ mA} = 64{,}5\ \Omega \rightarrow 68\ \Omega$.

6 Der ESP32 als Uhr

6.1 Initialisierung der Real Time Clock

In der Weboberfläche soll für das Senden von Befehlen eine vom Benutzer spezifizierte Uhrzeit eingegeben werden können. Dafür muss auf dem Mikrocontroller eine Uhr laufen, die in der Lage ist, die aktuelle Uhrzeit mit der Sendezeit abzugleichen und den Befehl pünktlich zu senden.
Zur Bewältigung dieser Herausforderung wird die Time-Bibliothek von Paul Stoffregen verwendet. Sie benutzt die interne Real Time Clock des ESP32, um eine Uhr zur Verfügung zu stellen, die synchron zur globalen Zeit läuft. Dabei wird zunächst keine Internetverbindung benötigt, was den Einsatz flexibel und unabhängig macht. Allerdings benötigt die Bibliothek eine Startzeit, die mit der in diesem Moment aktuellen globalen Zeit übereinstimmt. So wird sichergestellt, dass die Clock des ESP32 nicht nur synchron läuft, sondern auch die richtige Uhrzeit besitzt. Bei der Initialisierung des Programms wird zunächst die Kompilierzeit als Referenzpunkt für die RTC des ESP32 benutzt. Dieses Verfahren wird im folgenden Codeabschnitt dargestellt.

```
// use library for working with real time clock
#include <TimeLib.h>
// create variables to store compile time and date
int compileHour, compileMinute, compileSecond, compileYear, compileDay;
char compileMonth[4];
// split compile time and date into dedicated variables
sscanf(__TIME__, "%d:%d:%d", &compileHour, &compileMinute, &compileSec-
ond);
sscanf(__DATE__, "%s %d %d", &compileMonth, &compileDay, &compileYear);
// setup reference time for time library
setTime(compileHour, compileMinute, compileSecond, compileDay,
monthToInt(compileMonth), compileYear);
```

Die Real Time Clock des ESP32 wird dann immer synchronisiert, wenn der Client Daten über Benutzereingaben sendet, wie in Abschnitt 4.3 genauer erläutert wurde.

6.2 Bestimmung der Sendezeit

Nachdem die aktuelle Uhrzeit und die Zeiten der zu sendenden Befehle empfangen wurden, muss die Dauer der Wartezeit bestimmt werden, welche verstreichen muss, bis der jeweils nächste Befehl gesendet werden kann.
Dafür wird zunächst mit der getNextCommand-Funktion aus dem Zeitplan aller eingestellten Befehle der zeitlich nächste Befehl, relativ zur aktuellen Uhrzeit, bestimmt. Dafür wird zunächst für jeden Befehl die nächste Sendezeit bestimmt.

```
NextCommand getNextCmd(void) {
      // get the next time the on command will be sent
      struct NextCommand nextOnCmd = getNextCommandStruct(paramOn);
      nextOnCmd.command = onCommand;
      // get the next time the off command will be sent
      struct NextCommand nextOffCmd = getNextCommandStruct(paramOff);
      nextOffCmd.command = offCommand;
      // get next time the third command will be sent
      struct NextCommand nextPlaceholderCmd = getNextCommand-
      Struct(paramPlaceholder);
      nextPlaceholderCmd.command = thirdCommand;
      // check which of these commands has the earliest send time
      struct NextCommand earliest = getEarliest(nextOnCmd, nextOffCmd,
      nextPlaceholderCmd);
      return earliest;
}
```

Die genauen Funktionen, wie der jeweils nächste Befehl ermittelt und die drei nächsten Befehle verglichen werden, sind im Anhang zu finden und werden hier nur kurz erläutert. Die getNextCommandStruct-Funktion legt den aktuellen Zeitpunkt mit Zeit und Wochentag als Ausgangspunkt fest. Von diesem Starttag an wird der Index für den Wochentag so lange hochgezählt, bis der jeweilige Wochentag im gespeicherten Array der Wochentage markiert ist. Die Wochentage sind, wie bereits zuvor erwähnt, mit einer „1“ oder „0“ als Sendetage in einem Integer-Array markiert. Wenn der nächste markierte Wochentag gefunden wurde, wird eine Instanz eines Structs erzeugt, in der der Befehl, der Index für den Wochentag und die Uhrzeit eingetragen werden.

```
struct NextCommand {
  uint32_t command;
  int day;
  int hour;
  int minute;
};
```

Dabei können Randfälle entstehen, bei denen beispielsweise eine Sendezeit am aktuellen Wochentag datiert ist, aber die Uhrzeit bereits in der Vergangenheit liegt. Solche Fälle werden ebenfalls berücksichtigt.
Dieses Verfahren wird für alle drei Befehlsarten mit ihren Zeitplänen durchgeführt. Dann können die drei nächsten Befehle in der getEarliest-Funktion miteinander verglichen werden. Dabei wird ein anderes Prinzip genutzt, bei dem der aktuelle Zeitpunkt als Ursprungszeitpunkt genutzt wird und die drei nächsten Befehle relativ zu diesem Zeitpunkt betrachtet werden. Der Befehl, der dann am nächsten zu diesem Ursprungszeitpunkt liegt, ist der Befehl, der zeitlich als nächstes gesendet werden muss.
Mit der Zeit des nächsten Sendezeitpunkts kann die Wartezeit, welche überbrückt werden muss, mit der Funktion getWaitingTime ermittelt werden. Die Wartezeit wird am Ende in einem eigenen Struct gespeichert.

```
typedef struct zuWartendeZeit {
  int Tage;
  int Stunden;
  int minute;
  long inMilliSekunden;
}zuWartendeZeit;
struct zuWartendeZeit WaitingTime;
```

Die Wartezeit wird im Code zunächst durch einfache Subtraktion der aktuellen Zeit von der nächsten Sendezeit ermittelt. Es kann passieren, dass diese Subtraktion negativ ist, weil beispielsweise keine volle Stunde, sondern nur einige Minuten überbrückt werden müssen, die Sendezeit aber trotzdem erst in der nächsten Stunde liegt. Dann werden Carry-Bits gesetzt, die anzeigen, dass die nächsthöhere Einheit, in diesem Beispiel die Anzahl der Stunden um eine Einheit niedriger sein muss, als durch die Subtraktion der Stunden berechnet wurde. Wenn also die aktuelle Uhrzeit 15:30 Uhr ist und der nächste Befehl um 16:15 Uhr gesendet werden muss, muss das Programm nicht eine Stunde und 45 Minuten, sondern nur 45 Minuten warten. Dieses Prinzip ist übertragbar auf die Stunden und Tage. Die Berechnungen werden daher mehrere Male mit den Einheiten Minuten, Stunden und Tage durchgeführt, welche im vollständigen Code im Anhang erkennbar sind.

```
unsigned long getWaitingTime(NextCommand Nachsterbefehl) {
      // define carry bits
      bool carryMinute = false;
      bool carryStunde = false;
      // set carry for minutes in case time interval is less than the
      next higher unit and calculate minutes
      if (Nachsterbefehl.minute - minute() < 0 ) {
            WaitingTime.minute = 60 + (Nachsterbefehl.minute - mi-
            nute());
            carryMinute = true;
      // standard case: calculate minutes
      }else{
            WaitingTime.minute =  Nachsterbefehl.minute - minute();
            carryMinute = false;
      }
      // decrement hours in case time interval is less than the full
      hour, indicated by carry bit
      if (carryMinute == true) {
            // set carry for hours with the same principle as carry
            for minutes and calculate hours
            if ((Nachsterbefehl.hour - 1) - hour() < 0) {
                  WaitingTime.Stunden = 24 + (Nachsterbefehl.hour - 1) -
                  hour();
                  carryStunde = true;
            // standard case: calculate hours
```

```
            }else{
                  WaitingTime.Stunden = (Nachsterbefehl.hour - 1) -
                  hour();
                  carryStunde = false;
            }
      // standard case without carry for minute
      }else { // Keine Stunde weniger weil kein Carry
            // set carry for hours and calculate hours
            if (Nachsterbefehl.hour - hour() < 0) {
                  WaitingTime.Stunden = 24 + (Nachsterbefehl.hour - 1) -
                  hour();
                  carryStunde = true;
            // standard case: calculate hours
            }else{
                  WaitingTime.Stunden = Nachsterbefehl.hour - hour();
                  carryStunde = false;
            }
      }
}
```

Die gesamte Wartezeit aus den ermittelten Minuten, Stunden und Tagen bis zum nächsten Sendezeitpunkt wird in Millisekunden umgerechnet und zurückgegeben.

```
// calculate waiting time in milliseconds
WaitingTime.inMilliSekunden = WaitingTime.minute * 60 * 1000 + Wai-
tingTime.Stunden * 60 * 60 * 1000 + WaitingTime.Tage * 60 * 60 * 24 *
1000;
```

7 Ablauf des Hauptprogramms

In den vorherigen Abschnitten wurden die einzelnen Blöcke des in Abbildung 7 zu sehenden Flussdiagramms entwickelt.

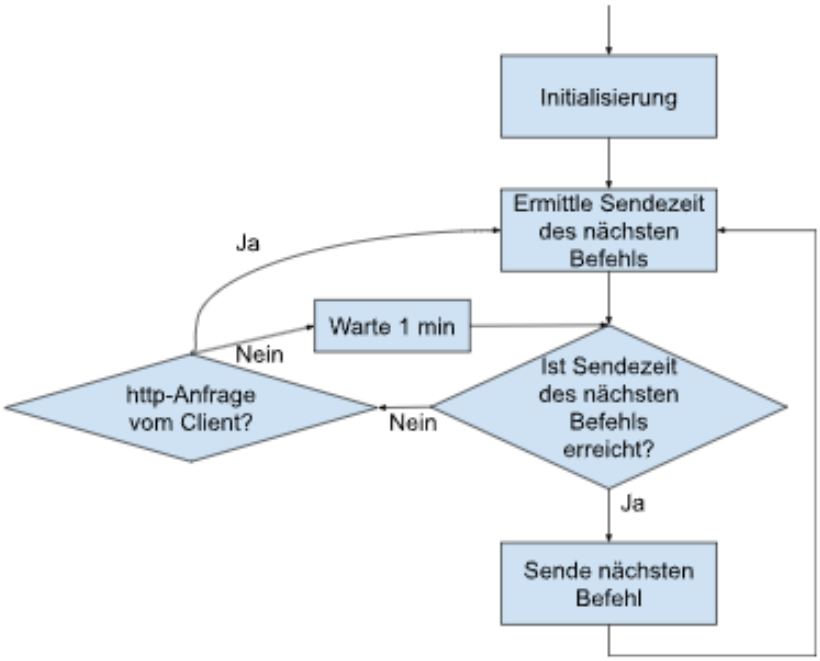

Abbildung 7: Flussdiagramm zur Veranschaulichung des Programmablaufs

Nun sollen diese Blöcke zu einem vollständigen Programm zusammengesetzt werden. Dies passiert in der `loop`-Funktion, die automatisch immer wieder durchlaufen wird. Dabei wird als erstes der Zeitpunkt des nächsten Befehls mit der `getNextCmd`-Funktion ermittelt. An diesen Ausgangspunkt springt der Code immer dann zurück, wenn ein Befehl gesendet wurde oder neue Sendezeiten eingestellt wurden. Dann wird die Zeit berechnet, die bis zum Senden dieses nächsten Befehls verstreichen muss. Dazu wird die `getWaitingTime`-Funktion aus dem vorherigen Abschnitt verwendet. In der darauf folgenden Schleife wird die seit dem Start der `loop`-Funktion verstrichene Zeit mit dem berechneten Zeitintervall bis zur Sendezeit des nächsten Befehls verglichen. Wenn die vergangene Zeit größer als dieses Zeitintervall wird, ist die Sendezeit des nächsten Befehls erreicht und dieser Befehl wird gesendet. In der Zwischenzeit kann es aber passieren, dass neue Einstellungen über die Weboberfläche getätigt werden. In diesem Fall wird ein Flag gesetzt, das zum Beenden der Wartezeit und zum erneuten Ausführen der `loop`-Funktion führt. So wird sichergestellt, dass ein neu eingestellter Befehl, der zeitlich vor dem aktuell nächsten Befehl liegt, tatsächlich als neuer nächster Befehl erkannt und gesendet wird. Die `loop`-Funktion ist im folgenden Code-Ausschnitt dargestellt.

```
void loop() {
      // get next command
      struct NextCommand next = getNextCmd();
      newCommand = false;
      // get waiting time for next command
      unsigned long waitInterval = ErmittelwarteZeit(next);
      // wait until next command should be sent
      previousMillis = millis();
      unsigned long currentMillis = millis();
```

```
        while (true) {
              // check if new parameters are received
              if (newCommand) {
                    break;
              }
              // check if waiting time is reached
              currentMillis = millis();
              float currentWaitTimeTmp = (currentMillis - previousMillis) /
              1000;
              Serial.println("waited for " + String(currentWaitTimeTmp));
              if (currentMillis - previousMillis >= waitInterval) {
                    // send next command
                    sendNec(next);
                    break;
              }

        }
}
```

8 Quellen

[1] https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, letzter Zugriff am 01.12.24.
[2]: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest#specifications, letzter Zugriff am 03.11.24.
[3] https://developer.mozilla.org/en-US/docs/Web/API/Element/click_event, letzter Zugriff am 11.01.25.
[4] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/time, letzter Zugriff am 11.01.25.
[5] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/checkbox, letzter Zugriff am 11.01.25.
[6] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date, letzter Zugriff am 11.01.25.
[7] https://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol, letzter Zugriff am 17.11.2024.
[8] https://github.com/Arduino-IRremote/Arduino-IRremote, letzter Zugriff am 17.11.2024.
[9] https://www.farnell.com/datasheets/2049868.pdf, letzter Zugriff am 17.11.2024.
[10] https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/spiffs.html, letzter Zugriff am 04.11.24.

9 Anhang

9.1 Vollständiger Code des ESP32

Das gesamte Programm des ESP32 wird in der Datei ir_esp_full.ino definiert. Sie muss auf den ESP32 hochgeladen werden, um das Programm auszuführen.

```
// add libraries
#include <Arduino.h>
#include <IRremote.hpp>
#include <TimeLib.h>
#include "WiFi.h"
#include "ESPAsyncWebSrv.h"
#include "SPIFFS.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <string>

// define login credentials
const char* ssid = "ESP32-Klimasteuerung";
const char* password = "1234567890";

// define command bytes, taken from real remote
const uint32_t onCommand = 0x46A88248;
const uint32_t offCommand = 0x46228248;
const uint32_t thirdCommand = 0x00000000;

// open up server on port 80
AsyncWebServer server(80);
// store compile date and time
int compileHour, compileMinute, compileSecond, compileYear, compileDay;
char compileMonth[4];
bool newCommand = false;
unsigned int previousMillis = 0;
// store parameters from html
struct CommandParam {
  int hours;
  int minutes;
  int weekdays[7]; // with sunday at index 0
};
// store parameters for next command that should be sent
struct NextCommand {
  uint32_t command;
  int day;
  int hour;
  int minute;
};
// predefine send times which will be overwritten by user
struct CommandParam paramOn = {10, 0, {0, 0, 1, 1, 1, 0, 0}};
```

```
struct CommandParam paramOff = {16, 0, {0, 0, 1, 1, 1, 0, 0}};
struct CommandParam paramPlaceholder = {0, 0, {0, 0, 0, 0, 0, 0, 0}};
// store waiting time
typedef struct zuWartendeZeit {
  int Tage;
  int Stunden;
  int minute;
  long inMilliSekunden;
}zuWartendeZeit;
struct zuWartendeZeit WaitingTime;
// store information about current time from html
struct CurrentTime {
  int days;
  char months[4];
  int years;
  int hours;
  int minutes;
  int seconds;
};
struct CurrentTime currentTime;

// convert string with '0' or '1' for each weekday into int array
void stringToIntArray(const String str, int* intArr) {
  for (int i = 0; i < 7; i++) {
    if (str[i] == '0') {
      intArr[i] = 0;
    } else if (str[i] == '1') {
      intArr[i] = 1;
    } else {
      intArr[i] = -1;
    }
  }
}

// convert month identifier into int
int monthToInt(const char* month) {
  const char* months[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec"};
  // loop through array and compare with list
  for (int i = 0; i < 12; ++i) {
    if (strncmp(month, months[i], 3) == 0) {
      return i + 1; // month found, return index
    }
  }
  // return -1 if month was not found
  return -1;
}

// start main program
void setup() {
  Serial.begin(9600);
  while (!Serial)
```

```
    ;
  // setup ir library
  pinMode(22, OUTPUT);
  IrSender.begin(22);

  // setup time library
  sscanf(__TIME__, "%d:%d:%d", &compileHour, &compileMinute, &compileSecond);
  // add 23 seconds to compile time to adjust for time transferring to esp, will be
synchronized with first sent command
  compileSecond += 23;
  if(compileSecond >= 60) {
    compileSecond -= 60;
    compileMinute += 1;
  }
  if(compileMinute >= 60) {
    compileMinute -= 60;
    compileHour += 1;
  }

  sscanf(__DATE__, "%s %d %d", &compileMonth, &compileDay, &compileYear);
  setTime(compileHour, compileMinute, compileSecond, compileDay, monthToInt(com-
pileMonth), compileYear);

  // start spiffs
  if (!SPIFFS.begin(true)) {
    Serial.println("An error has occurred while mounting SPIFFS");
    return;
  }

  // configure esp as access point
  WiFi.softAP(ssid, password);
  IPAddress IP = WiFi.softAPIP();
  Serial.print("Access Point IP address: ");
  Serial.println(IP);

  // setup esp for http requests
  server.on("/", HTTP_GET, [](AsyncWebServerRequest * request) {
    Serial.println("ESP32 Web Server: New request received: /");
    request->send(SPIFFS, "/index.html", "text/html");
  });
  // setup esp for http requests with on command
  server.on("/setOn", HTTP_GET, [](AsyncWebServerRequest * request) {
    Serial.println("on request set: " + String(request->getParam("time")->value()) +
", " + String(request->getParam("week")->value()));
    String t = request->getParam("time")->value();
    paramOn.hours = t.substring(0, 2).toInt();
    paramOn.minutes = t.substring(3, 5).toInt();
    // weekdays
    String days = request->getParam("week")->value();
    stringToIntArray(days, paramOn.weekdays);
    // current time
```

```
    String currTime = request->getParam("currentTime")->value();
    const char* currTimeStr = currTime.c_str();
    sscanf(currTimeStr, "%*s %3s %d %d %d:%d:%d %*s", &currentTime.months, &cur-
rentTime.days, &currentTime.years, &currentTime.hours, &currentTime.minutes, &cur-
rentTime.seconds);
    setTime(currentTime.hours, currentTime.minutes, currentTime.seconds, cur-
rentTime.days, monthToInt(currentTime.months), currentTime.years);

    // html
    request->send(SPIFFS, "/index.html", "text/html");

    newCommand = true;
  });
  // setup esp for http requests with off command
  server.on("/setOff", HTTP_GET, [](AsyncWebServerRequest * request) {
    Serial.print("off request set: " + String(request->getParam("time")->value()) +
", " + String(request->getParam("time")->value()));
    // time
    String t = request->getParam("time")->value();
    paramOff.hours = t.substring(0, 2).toInt();
    paramOff.minutes = t.substring(3, 5).toInt();
    // weekdays
    String days = request->getParam("week")->value();
    stringToIntArray(days, paramOff.weekdays);
    // current time
    String currTime = request->getParam("currentTime")->value();
    const char* currTimeStr = currTime.c_str();
    sscanf(currTimeStr, "%*s %3s %d %d %d:%d:%d %*s", &currentTime.months, &cur-
rentTime.days, &currentTime.years, &currentTime.hours, &currentTime.minutes, &cur-
rentTime.seconds);
    setTime(currentTime.hours, currentTime.minutes, currentTime.seconds, cur-
rentTime.days, monthToInt(currentTime.months), currentTime.years);
    // html
    request->send(SPIFFS, "/index.html", "text/html");

    newCommand = true;
  });
// setup esp for http requests with third command
  server.on("/setPlaceholder", HTTP_GET, [](AsyncWebServerRequest * request) {
    Serial.print("placeholder request set: " + String(request->getParam("time")-
>value()) + ", " + String(request->getParam("time")->value()));
    // time
    String t = request->getParam("time")->value();
    paramPlaceholder.hours = t.substring(0, 2).toInt();
    paramPlaceholder.minutes = t.substring(3, 5).toInt();
    // weekdays
    String days = request->getParam("week")->value();
    stringToIntArray(days, paramPlaceholder.weekdays);
    // current time
    String currTime = request->getParam("currentTime")->value();
    const char* currTimeStr = currTime.c_str();
```

```
    sscanf(currTimeStr, "%*s %3s %d %d %d:%d:%d %*s", &currentTime.months, &cur-
rentTime.days, &currentTime.years, &currentTime.hours, &currentTime.minutes, &cur-
rentTime.seconds);
    Serial.println("set time to: " + String(currentTime.hours) + ":" + String(cur-
rentTime.minutes) + ":" + String(currentTime.seconds));
    setTime(currentTime.hours, currentTime.minutes, currentTime.seconds, cur-
rentTime.days, monthToInt(currentTime.months), currentTime.years);
    // html
    request->send(SPIFFS, "/index.html", "text/html");
    newCommand = true;
  });
  server.begin();
}



// start of main loop
void loop() {
  // get next command
  struct NextCommand next = getNextCmd();
  newCommand = false;

  // get waiting time
  unsigned long waitInterval = getWaitingTime(next);
  // wait one minute less to actually reach correct time
  int waitInterval2 = waitInterval-60000;
  if(waitInterval2 < 0) {
    waitInterval2 = 0;
  }
  Serial.println("wait for " + String(waitInterval2));

  // wait
  previousMillis = millis();
  unsigned long currentMillis = millis();
  while (true) {
    // check if new parameters are received
    if (newCommand) {
      break;
    }
    // check if waiting time is reached
    currentMillis = millis();
    float currentWaitTimeTmp = (currentMillis - previousMillis) / 1000;
    Serial.println("waited for " + String(currentWaitTimeTmp));
    if (currentMillis - previousMillis >= waitInterval2) {
      break;
    }
    delay(10000);
  }

  Serial.println("current time: " + String(hour()) + ":" + String(minute()) + ":" +
String(second()));
```

```
  Serial.println("next time: " + String(next.hour) + ":" + String(next.minute));
  // wait for exact time
  if(newCommand) {
    newCommand = false;
  }else{
    while(true) {;
      if (newCommand) {
        break;
      }
       if(hour() == next.hour && minute() == next.minute && second() >= 0){
        sendNec(next);
        break;
       }
       delay(500);
    }
  }

  Serial.println("finished sending command");
}

// send nec code 3x
void sendNec(const NextCommand& next) {
  Serial.println("send nec command");
  IrSender.sendNECRaw(next.command, 0);
  delay(1000);
  IrSender.sendNECRaw(next.command, 0);
  delay(1000);
  IrSender.sendNECRaw(next.command, 0);
}

// get next send time for one command
NextCommand getNextCommandStruct(const CommandParam paramCmd) {
  struct NextCommand nextCmd = { -1, -1, -1};
  // get current weekday
  int today = weekday() - 1;
  int searchDay = today;
  while (1) {
    // check if command should be sent today with current weekday as index for week-
days array
    if (paramCmd.weekdays[searchDay] == 1) {;
      if (searchDay == today) {
        Serial.println("send next command today");
        // check if current time is later than next command time
        String currentTime = String(hour());
        int minut = minute();
        if (minut < 10) {
          currentTime += "0" + String(minut);
        } else {
          currentTime += String(minut);
        }
        int currentTimeInt = currentTime.toInt();
```

```
        String cmdTime = String(paramCmd.hours);
        if (paramCmd.minutes < 10) {
          cmdTime += "0" + String(paramCmd.minutes);
        } else {
          cmdTime += String(paramCmd.minutes);
        }
        int cmdTimeInt = cmdTime.toInt();
        // send command later today
        if (cmdTimeInt > currentTimeInt) {
          Serial.println("send time is later today");
          break;
        }
      } else {
        Serial.println("send next command at some point");
        break;
      }
    }
    // no command should be sent today, check for the next days
    searchDay += 1;
    if (searchDay > 6) {
      searchDay = 0;
      Serial.println("go to sunday");
    }
    // break when current weekday is reached again (no days found)
    if (searchDay == today) {
      nextCmd.day = 1000000;
      nextCmd.hour = 1000000;
      nextCmd.minute = 1000000;
      return nextCmd;
    }
  }
  nextCmd.day = searchDay + 1;
  nextCmd.hour = paramCmd.hours;
  nextCmd.minute = paramCmd.minutes;
  return nextCmd;
}

// check which of the two commands should be sent earlier
bool isEarlier(const NextCommand& a, const NextCommand& b) {
  if (a.day != b.day) {
    return a.day < b.day;
  }
  if (a.hour != b.hour) {
    return a.hour < b.hour;
  }
  return a.minute < b.minute;
}

// check which of three commands should be sent the earliest
NextCommand getEarliest(NextCommand t1, NextCommand t2, NextCommand t3) {
  transformCommandTimes(t1, t2, t3);
```

```
  NextCommand earliest = t1;
  if (isEarlier(t2, earliest)) {
    earliest = t2;
  }
  if (isEarlier(t3, earliest)) {
    earliest = t3;
  }
  transformTimesBack(earliest);
  return earliest;
}

// calculate commands relative to current time as starting point
void transformCommandTimes(NextCommand &t1, NextCommand &t2, NextCommand &t3) {
  int weekdays = weekday();
  int hours = hour();
  int minutes = minute();
  transformTimes(t1, weekdays, hours, minutes);
  transformTimes(t2, weekdays, hours, minutes);
  transformTimes(t3, weekdays, hours, minutes);
}

// move times around to have current time as starting point and every send time rel-
ative to this starting point
void transformTimes(NextCommand &t, const int weekdays, const int hours, const int
minutes) {
  // subtract current date/time to find difference
  t.day -= weekdays;
  if(t.day < 0) {
    t.day += 7;
  }
  t.hour -= hours;
  if(t.hour < 0) {
    t.hour += 24;
  }
  t.minute -= minutes;
  if(t.minute < 0) {
    t.minute += 60;
  }
}


// move times back after comparing them
void transformTimesBack(NextCommand &t) {
  t.day += weekday();
  if (t.day > 6) {
    t.day -= 7;
  }
  t.hour += hour();
  if (t.hour > 23) {
    t.hour -= 24;
  }
```

```
  t.minute += minute();
  if (t.minute > 59) {
    t.minute -= 60;
  }
}

// return command that should be sent next
NextCommand getNextCmd(void) {
  Serial.println("start getting next cmd");
  struct NextCommand nextOnCmd = getNextCommandStruct(paramOn);
  nextOnCmd.command = onCommand;
  struct NextCommand nextOffCmd = getNextCommandStruct(paramOff);
  nextOffCmd.command = offCommand;
  struct NextCommand nextPlaceholderCmd = getNextCommandStruct(paramPlaceholder);
  nextPlaceholderCmd.command = thirdCommand;

  struct NextCommand earliest = getEarliest(nextOnCmd, nextOffCmd, nextPlaceholder-
Cmd);
  Serial.println("next command: " + String(earliest.command) + " @ " + String(earli-
est.day) + ", " + String(earliest.hour) + ":" + String(earliest.minute));
  return earliest;
}

unsigned long getWaitingTime(NextCommand Nachsterbefehl) {
  bool carryMinute = false;
  bool carryStunde = false;
  // Carry resetten
  if (Nachsterbefehl.minute - minute() < 0 )
  {
    WaitingTime.minute =  60 + (Nachsterbefehl.minute - minute());
    carryMinute = true;
    Serial.println("Minuten bei gesetztem Carry");
    Serial.println(WaitingTime.minute);

  }
  else
  {
    WaitingTime.minute =  Nachsterbefehl.minute - minute();
    carryMinute = false;
    Serial.println("Minuten bei nicht gesetztem Carry");
    Serial.println(WaitingTime.minute);
  }

  if (carryMinute == true)
  {
    // eine Stunde kürzer
    if ((Nachsterbefehl.hour - 1) - hour() < 0)
    {
      WaitingTime.Stunden = 24 + (Nachsterbefehl.hour - 1) - hour();
      carryStunde = true; // Ein Tag weniger
      Serial.println("Stunden bei gesetzem Carry");
```

```
    Serial.println(WaitingTime.Stunden);
  }
  else
  {
    WaitingTime.Stunden = (Nachsterbefehl.hour - 1) - hour();
    carryStunde = false;
    Serial.println("Positive Stunden bei Carry");
    Serial.println(WaitingTime.Stunden);
  }
}

else { // Keine Stunde weniger weil kein Carry
  if (Nachsterbefehl.hour - hour() < 0)
  {
    WaitingTime.Stunden = 24 + (Nachsterbefehl.hour - 1) - hour();
    carryStunde = true; // Ein Tag weniger
    Serial.println("Negative Stunden bei nicht gesetztem Carry");
    Serial.println(WaitingTime.Stunden);
  }
  else
  {
    WaitingTime.Stunden = Nachsterbefehl.hour - hour();
    Serial.println("Postive Stunden bei nicht gesetztem Carry");
    Serial.println(WaitingTime.Stunden);
  }
}


if (carryStunde == true) {
  // Tag weniger
  if ((Nachsterbefehl.day - 1) - weekday() < 0)
  {
    WaitingTime.Tage = 7 + (Nachsterbefehl.day - 1 - weekday());
    Serial.println("Negative Tage wenn carry gesetzt");
    Serial.println(WaitingTime.Tage);
  }
  else
  {
    WaitingTime.Tage = (Nachsterbefehl.day - 1) - weekday();
    Serial.println("Positive Tage, wenn carry gesetzt");
    Serial.println(WaitingTime.Tage);

  }
}

else {
  if (Nachsterbefehl.day - weekday() < 0)
  {
    WaitingTime.Tage = 7 + Nachsterbefehl.day - weekday();
    Serial.println(" negative Tage, wenn carry nicht gesetzt");
    Serial.println(WaitingTime.Tage);
```

```
    }
    else
    {
      WaitingTime.Tage = Nachsterbefehl.day - weekday();
      Serial.println("Positive Tage, wenn carry nicht gesetzt");
      Serial.println(WaitingTime.Tage);
    }
  }

  // Berechnung der Zeit in ms
  WaitingTime.inMilliSekunden = WaitingTime.minute * 60 * 1000 + WaitingTime.Stunden
* 60 * 60 * 1000 + WaitingTime.Tage * 60 * 60 * 24 * 1000 ;
  return WaitingTime.inMilliSekunden;
};
```

9.2 Vollständiger Code des Clients

Der Code der html-Weboberfläche, die an den Client gesendet wird, ist auf dem ESP32 lokal mit dem Dateiverwaltungstool Spiffs hinterlegt und beinhaltet die Steuerelemente für den Benutzer, mit denen er Sendezeiten für Befehle einstellen kann.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Steuerung der Klimaanlage</title>
</head>
<body>
    <title>Steuerung der Klimaanlage</title>
    <style type="text/css">
        section {
            float: left;
            width: 30%;
        }

        section2 {
            float: left;
            width: 30%;
        }

        section3 {
            float: left;
            width: 30%;
        }

        section4 {
            float: upper;
            height: 50%;
        }
    </style>
    <h3> Steuerung der Klimaanlage</h3>
    <section>
        <p><b> Drücken, zum Senden der Anschaltzeiten </b></p>
        <p><button id="ButtonKlimaAn"
style="width:120px;height:50px;left:NaNpx;top:NaNpx;"
                onclick=onclickKlimaAn()><b>Klima An</b></button></p>
    </section>
    <section>
        <p> <label for="InputZeitKlimaAn"><b>Zeiten zum aktivieren der An-
lage</b></label> </p>
        <p> <input type="time" id="InputZeitKlimaAn" name="TimeAcOn" required
style="width:120px;height:50px;"> </p>

    </section>
    <section>
        <fieldset>
```

```
            <legend>Tag zum Anschalten der Anlage</legend>
            <span>
                <div>
                    <input type="checkbox" id="Monday" Name="Mo" onclick="Mon-
day()"><label for="Monday">Montag</label>
                </div>
                <div>
                    <input type="checkbox" id="Tuesday" Name="Tue"><label for="Tues-
day">Dienstag</label>
                </div>
            </span>
            <div>
                <input type="checkbox" id="Wednesday" Name="Wed"><label for="Wednes-
day">Mittwoch</label>
            </div>
            <div>
                <input type="checkbox" id="Thursday" Name="Th"><label for="Thurs-
day">Donnerstag</label>
            </div>
            <div>
                <input type="checkbox" id="Friday" Name="Fri"><label for="Fri-
day">Freitag</label>
            </div>
            <div>
                <input type="checkbox" id="Saturday" Name="Sat"><label for="Satur-
day">Samstag</label>
            </div>
            <div>
                <input type="checkbox" id="Sunday" Name="Sun"><label for="Sun-
day">Sonntag</label>
            </div>
        </fieldset>
    </section>
    <section2>
        <p><b>Drücken, zum Senden der Ausschaltzeiten</b></p>
        <p><button id="ButtonKlimaAus"
style="width:120px;height:50px;left:NaNpx;top:NaNpx;"
                onclick=onclickKlimaAus()><b> Klima Aus </b></button></p>
    </section2>
    <section2>
        <p><label for="InputZeitKlimaAus"><b>Zeiten zum ausschalten der An-
lage</b></label><!-- Untereinander--></p>
        <p><input type="time" id="InputZeitKlimaAus" name="TimeAcOn" required
style="width:120px;height:50px;"> </p>
    </section2>
    <section2>
        <fieldset>
            <legend>Tage für Klimaanlage aus</legend>
            <span>
                <div>
```

```
                    <input type="checkbox" id="MondayOff" Name="Mo"><label for="Mon-
dayOff">Montag</label>
                </div>
                <div>
                    <input type="checkbox" id="TuesdayOff" Name="Tue"><label
for="TuesdayOff">Dienstag</label>
                </div>
            </span>
            <div>
                <input type="checkbox" id="WednesdayOff" Name="Wed"><label
for="WednesdayOff">Mittwoch</label>
            </div>
            <div>
                <input type="checkbox" id="ThursdayOff" Name="Th"><label for="Thurs-
dayOff">Donnerstag</label>
            </div>
            <div>
                <input type="checkbox" id="FridayOff" Name="Fri"><label for="Fri-
dayOff">Freitag</label>
            </div>
            <div>
                <input type="checkbox" id="SaturdayOff" Name="Sat"><label for="Sat-
urdayOff">Samstag</label>
            </div>
            <div>
                <input type="checkbox" id="SundayOff" Name="Sun"><label for="Sun-
dayOff">Sonntag</label>
            </div>
        </fieldset>
    </section2>
    <section3>
        <p> <b> Drücke, zum Senden der "Funktion 3-Zeiten"</b> </p>
        <p><button id="Funktion3"
style="width:120px;height:50px;left:NaNpx;top:NaNpx;"
                onclick=onclickKlima3()><b>Funktion 3</b></button></p>

    </section3>
    <section3>
        <p> <label for="InputZeitKlimaAn"> <b>Zeiten für Funktion 3 </b></label>
</p>
        <!-- Untereinander-->
        <p><input type="time" id="InputZeitFunktion3" name="Time3" required
style="width:120px;height:50px;"></p>
    </section3>
    <section3>
        <fieldset>
            <legend>Tage für Funktion3</legend>
            <span>
                <div>
                    <input type="checkbox" id="Monday3" Name="Mo" onclick="Mon-
day()"><label for="Monday3">Montag</label>
```

```
                </div>
                <div>
                    <input type="checkbox" id="Tuesday3" Name="Tue"><label
for="Tuesday3">Dienstag</label>
                </div>
            </span>
            <div>
                <input type="checkbox" id="Wednesday3" Name="Wed"><label
for="Wednesday3">Mittwoch</label>
            </div>
            <div>
                <input type="checkbox" id="Thursday3" Name="Th"><label for="Thurs-
day3">Donnerstag</label>
            </div>
            <div>
                <input type="checkbox" id="Friday3" Name="Fri"><label for="Fri-
day3">Freitag</label>
            </div>
            <div>
                <input type="checkbox" id="Saturday3" Name="Sat"><label for="Satur-
day3">Samstag</label>
            </div>
            <div>
                <input type="checkbox" id="Sunday3" Name="Sun"><label for="Sun-
day3">Sonntag</label>
            </div>
        </fieldset>
    </section3>
</body>


<script>
    function changingBoolToBit(IdElement) {
        return document.getElementById(IdElement).checked ? 1 : 0;
    }
<!-- send parameters for on command -->
    function onclickKlimaAn() {
        var xhttp = new XMLHttpRequest();
        var today = new Date();
        var data = "/setOn?";
        var InputTimeOn = document.getElementById("InputZeitKlimaAn").value;
        data += "time=" + InputTimeOn;
        data += "&week=" + changingBoolToBit("Sunday") + changingBoolToBit("Monday")
+ changingBoolToBit("Tuesday") +
            changingBoolToBit("Wednesday") + changingBoolToBit("Thursday") + chang-
ingBoolToBit("Friday")
            + changingBoolToBit("Saturday");
        data += "&currentTime=" + today;

        xhttp.open("GET", data);
        xhttp.send();
```

```
    }
<!-- send parameters for off command -->
    function onclickKlimaAus() {
        var xhttp = new XMLHttpRequest();
        var data = "/setOff?";
        var today = new Date();
        var InputTimeAus = document.getElementById("InputZeitKlimaAus").value;
        data += "time=" + InputTimeAus;
        data += "&week=" + changingBoolToBit("SundayOff") + changingBoolToBit("Mon-
dayOff") + changingBoolToBit("TuesdayOff") +
            changingBoolToBit("WednesdayOff") + changingBoolToBit("ThursdayOff") +
changingBoolToBit("FridayOff")
            + changingBoolToBit("SaturdayOff");
        data += "&currentTime=" + today;

        xhttp.open("GET", data);
        xhttp.send();
    }
<!-- send parameters for third command -->
    function onclickKlima3() {
        var xhttp = new XMLHttpRequest();
        var data = "/setPlaceholder?";
        var today = new Date();
        var InputZeitFunktion3 = document.getElementById("InputZeitFunk-
tion3").value;
        data += "time=" + InputZeitFunktion3;
        data += "&week=" + changingBoolToBit("Sunday3") + changingBoolToBit("Mon-
day3") + changingBoolToBit("Tuesday3") +
            changingBoolToBit("Wednesday3") + changingBoolToBit("Thursday3") +
changingBoolToBit("Friday3")
            + changingBoolToBit("Saturday3");
        data += "&currentTime=" + today;

        xhttp.open("GET", data);
        xhttp.send();
    }

</script>
</body>
</html>
```