```
/*----- Weekly timer sets with timer plans for the Eigthree ET28 with ESP32-C3 ------
For the Arduino IDE, select board to ESP32C3 dev module.
A brief hard- and software description is found at: https://www.larsenhenneberg.dk
-----
Conditions: The program uses Arduino OTA to flash new firmware verions via wifi.
Make sure to use the correct values for:
Access point (AP) SSID and password to use the simple captive portal.
-----
Work scheme at reset or program restart:
.01: The wifi network ssid and password is read from the EEPROM
.02: The wifi connection is tried for a few sec. with the ssid and password, see .01
.03: Upon RTC error, it waits for wifi connection, otherwise read time from RTC
.04: Upon an active setup button, an Access Point (AP) is started, see .06
.05: The program continues into the loop(), see .11
.06: Use f.ex. a mobile phone to use the AP network.
.07: Browse to the captive portal at http://192.168.4.1
.08: Enter the wifi ssid and password at the captive portal web page and press Save.
.09: The ssid and password from the captive portal are saved into the EEprom.
.10: The ESP32 is restarted, and starts from .01
.11: In the loop(): in case of an active AP, the captive portal web client is handled.
.12: A while(1) loop inside loop() is now the main loop.
.13: The yield() function is called in while(1) to avoid a watch dog program reset.
.14: Upon an active setup button, the wifi is disconnected and the ESP32 is restarted, see .01
.15: If wifi is available, the NTP time is read on the hour, e.g. xx:00 and the RTC is updated
.16: If no wifi is available, the time keeping is made with simple delay loops.
.17: if no wifi at start or end of Daylight Savings Time, the right time is set at next NTP
-----
The time plans are changed by using a http request to the ip address: http://ip_addr:http_port
or http://localURL.local:http_port . LocalURL is defined via a const char* in the code.
The start and end time points are typed in hh:mm format and are saved to the ESP32-C3 EEprom.
.Note1: #define timerSets sets the number of timer sets, one for each output
.Note2: #define timerPlans sets the number of timeplans for ON/OFF scheduling, max 8 timeplans.
.Note3: the max total number timerplans is 16, calculated from timerSets * timerPlans
.Note4: The String htmlTitle defines the HTML page title
.Note5: define the number of output timerset_pins to fit the number of timerSets, see .Note1
-----*/
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <time.h>
#include <EEPROM.h>
#include <WebServer.h>

#define http_Port 8058        // http port to access the timer set/plan webpage
#define TZ_DENMARK "CET-1CEST,M3.5.0,M10.5.0/3" // time zone and daylight def.

#define timerPlans 5      // number of timerPlans

#define setup_Pin 5           // GPIO5 as active low input to invoke the captive portal
#define Blue_LED_pin 20       // GPIO20 as output , active low for blue LED network light
#define Red_LED_pin 21        // GPIO21 as output , active low for red LED relay active light
#define Relay1_pin 6          // GPIO6 as output , active high for Relay1
//#define SEL_PIN 4
//#define CF1_PIN 7
#define CF_PIN 3
#define ledON LOW
#define ledOFF HIGH
#define relayON HIGH
#define relayOFF LOW
```

```
//for power monitoring
#define POWER_MULTIPLIER 1.47 // transforms the measured frequency to power in W
#define CFtimeout 5000  // power calc. timeout after last CF ISR interrupt

#define EEPROM_SIZE 512     // Allocate 512 bytes for wifi ssid, passw., and timeplans
#define EEprom_adr 0        // EEprom start address normally 0

String htmlTitle = "Eigthree week plans";  // web page title
const char* localURL = "week_timer";     // use week_timer.local as LAN URL

// Soft AP ( Access Point) network setup for the captive portal
const char* APssid = "ESP32-C3 week timers";
const char* APpassword = "";              // might add AP password here !

WebServer APserver(80);       // this webservice is for AP only
WiFiServer server(http_Port); // webservice for the timeplan setup

// EEprom parameters are read in at start, and saved on changes.
struct wifiSettings {   // the wifiSettings
  char ssid[30];
  char password[30];
} user_wifi = {};

struct turnTime{    // definition of one start time or end time for the time plans
  byte Hour;
  byte Minute;
};

// The timerPlans with start-end_Hour:Minute entries, 7 days a week, saved in EEprom
struct turnTime startTime[7][timerPlans], endTime[7][timerPlans];

// size of one day time plan entry e.g. size of start-time + sizeof end-time
#define dayturnTimeSize 2*sizeof(turnTime)

struct turnMinutes{   // to hold number of minutes from 00:00
  unsigned int startMinutes;
  unsigned int endMinutes;
};
// for the number of minutes from 00:00 for all the Hour:Minute entries the in time plans
struct turnMinutes onoffMinutes[7][timerPlans];

bool outputState;   // boolean for the output state

bool ntpSetupOK = false;   // flag to indicate OK NTP setup
bool getNTPtime;              // flag to request NTP time

// values in msec used by millis() for non-blocking loop() processing
uint32_t previousMillis = 0;
uint32_t OneMinute = 60 * 1000L;
uint32_t adjustMinute;    // to adjust sw time keeping to NTP
bool adjustSWtiming;      // bool to indicate use of adjustMinute

// power metering variables
float powerW;
float pulseFreq;
uint32_t pulseWidth, msNow, msPreviousPulse;
void ICACHE_RAM_ATTR CF_impulse();

// globals for NTP and time keeping
```

```
struct tm timeinfo;
time_t now;
String Hour_str;
String Minute_str;
String Second_str;
String Day_str;
String Month_str;
String DayOfMonth_str;
String Year_str;
String timestr;
// RTC variables
byte Second;      // Second 0..59
byte DayOfMonth;  // Day of month 1..31
byte Month;       // Month 1..12
byte Year;        // Year 0..99
byte Hour;        // Hours 0..23
byte Minute;      // Minutes 0..59
byte Day;         // Day 0..6 , 0 is Monday

void setup() {
  pinMode(setup_Pin, INPUT_PULLUP);
  pinMode(Blue_LED_pin, OUTPUT);   // Blue LED pin as output
  pinMode(Red_LED_pin, OUTPUT);   // Red LED pin as output
  pinMode(Relay1_pin, OUTPUT);     // Relay1 pin as output
  pinMode(CF_PIN, INPUT);               // Set pin to input for capturing CF_PIN pulses
  digitalWrite(Blue_LED_pin, ledOFF);   // set blue LED inactive
  digitalWrite(Red_LED_pin, ledOFF);   // set red LED inactive
  digitalWrite(Relay1_pin, relayOFF);     // set Relay1 output inactive
  interrupts();                                 // Enable interrupts (in case they were previously
disabled)
  attachInterrupt(digitalPinToInterrupt(CF_PIN), CF_impulse, RISING); // power pulse interrupt

  EEPROM.begin(EEPROM_SIZE);
  EEPROM.get(0, user_wifi);
  // read the timePlans from EEprom
  for (int tp=0; tp < timerPlans; tp++) {
    for (int wday=0; wday < 7; wday++) {
      EEPROM.get(EEprom_adr + (wday+7*tp)*dayturnTimeSize+sizeof(struct wifiSettings), startTime[
wday][tp]);
      EEPROM.get(EEprom_adr + ((2*wday+1)+2*7*tp)*sizeof(turnTime)+sizeof(struct wifiSettings),
endTime[wday][tp]);
    }
  }
   // check if Hour:Minute values from the EEprom are out of limit, then Hour:Minute = 00:00
  for (int tp=0; tp < timerPlans; tp++) {
    for (int wday=0; wday < 7; wday++) {
        if ((startTime[wday][tp].Hour < 0) || (startTime[wday][tp].Hour > 23)) { startTime[wday][tp
].Hour = 0; }
        if ((startTime[wday][tp].Minute < 0) || (startTime[wday][tp].Minute > 59)) { startTime[wday
][tp].Minute = 0; }
        if ((endTime[wday][tp].Hour < 0) || (endTime[wday][tp].Hour > 23)) { endTime[wday][tp].Hour
 = 0; }
      if ((endTime[wday][tp].Minute < 0) || (endTime[wday][tp].Minute > 59)) { endTime[wday][tp].
Minute = 0; }
    }
  }
  // for all Hour:Minute entries in the timePlans
  // calculate start-ON and end-OFF points in minutes: time_in_minutes = 60*hours+minutes
  for (int tp=0; tp < timerPlans; tp++) {
    for (int wday=0; wday < 7; wday++) {
```

```
      onoffMinutes[wday][tp].startMinutes = int(startTime[wday][tp].Hour)*60+int(startTime[wday][tp
].Minute);
      onoffMinutes[wday][tp].endMinutes = int(endTime[wday][tp].Hour)*60+int(endTime[wday][tp].
Minute);
    }
  }
  // try wifi connection with ssid,password from EEprom
  WiFi.mode(WIFI_STA);
  WiFi.begin(user_wifi.ssid, user_wifi.password);
  while ((WiFi.status() != WL_CONNECTED) && (!setupButton())) {
    digitalWrite(Blue_LED_pin, 0);
    delay(500);
    digitalWrite(Blue_LED_pin, 1);
    delay(500);
  }
  if ((WiFi.status() != WL_CONNECTED) || (setupButton())){
    WiFi.mode(WIFI_AP);
    WiFi.softAP(APssid, APpassword);
    APserver.on("/",  handlePortal);      // start captive portal
    APserver.begin();
  }

  initOTA();      // start the "Over The Air" firmware update option
  server.begin();

  // set the local URL
  int cnt = 5;
  while (!MDNS.begin(localURL) && (cnt > 0)) {
    delay(1000);
    cnt--;
  }

  getNTPtime = true; // force a NTP request in loop()
  previousMillis = millis();
  msPreviousPulse = millis();
}

void loop() {
  while (WiFi.status() != WL_CONNECTED) { // waiting for AP client
    digitalWrite(Blue_LED_pin, 0);
    delay(100);                       // short blink to indicate
    digitalWrite(Blue_LED_pin, 1);    // an active captive portal
    delay(500);
    APserver.handleClient();          // to finish portal
  }
  APserver.close();   // Close the APserver
  while (1) {             // going into the main loop !
    yield();              // refresh wdt (watch dog timer) to avoid reset
    if (setupButton()) {  // setup button pressed ?
      WiFi.disconnect();  // disconnect wifi
      ESP.restart();      // restart to continue with AP
    }

  if ((millis() - msPreviousPulse) > CFtimeout ) { powerW = 0.0; }
  else { powerW = pulseFreq * POWER_MULTIPLIER + 0.5; } // CF frequency to power in W, rounded up
  // if wifi available then request NTP server regularly
  if ((WiFi.status() == WL_CONNECTED) && ( getNTPtime == true)) {
    digitalWrite(Blue_LED_pin, 0); // set blue LED active
    getNTPtime = false;
    if (ntpSetupOK == false) {  // config NTP ?
```

```
      configTime(0, 0, "3.dk.pool.ntp.org");    // First connect to NTP server, with 0 TZ offset
      if (getLocalTime(&timeinfo)) {
        setenv("TZ",TZ_DENMARK,1);  // Adjust to the TZ and Daylight Savings Time.
        tzset();                    // Clock settings are adjusted to show the new local time
        ntpSetupOK = true;
      }
    }
    if (getLocalTime(&timeinfo)) {
      char timeStringBuff[30];      // to hold timeinfo characters
      strftime(timeStringBuff, sizeof(timeStringBuff), "%a %b %d %H:%M:%S %Y", &timeinfo);
      timestr = String(timeStringBuff);  // ex.: Wed Dec 27 17:43:24 2023
      Day_str = timestr.substring(0,3);
      Day = 0;
      if (Day_str == "Mon") { Day = 0; }
      if (Day_str == "Tue") { Day = 1; }
      if (Day_str == "Wed") { Day = 2; }
      if (Day_str == "Thu") { Day = 3; }
      if (Day_str == "Fri") { Day = 4; }
      if (Day_str == "Sat") { Day = 5; }
      if (Day_str == "Sun") { Day = 6; }
      Month_str = timestr.substring(4,7);
      Month = 1;
      if (Month_str == "Jan") { Month = 1; }
      if (Month_str == "Feb") { Month = 2; }
      if (Month_str == "Mar") { Month = 3; }
      if (Month_str == "Apr") { Month = 4; }
      if (Month_str == "May") { Month = 5; }
      if (Month_str == "Jun") { Month = 6; }
      if (Month_str == "Jul") { Month = 7; }
      if (Month_str == "Aug") { Month = 8; }
      if (Month_str == "Sep") { Month = 9; }
      if (Month_str == "Oct") { Month = 10; }
      if (Month_str == "Nov") { Month = 11; }
      if (Month_str == "Dec") { Month = 12; }
      DayOfMonth_str = timestr.substring(8,10);
      DayOfMonth = DayOfMonth_str.toInt();
      Hour_str = timestr.substring(11,13);
      Hour = Hour_str.toInt();          // save hour as byte
      Minute_str = timestr.substring(14,16);
      Minute = Minute_str.toInt();      // save minute as byte
      Second_str = timestr.substring(17,19);
      Second = Second_str.toInt();    // save second as byte
      adjustSWtiming = true;          // use adjustMinute once in SW timer
      adjustMinute = (60 - Second) * 1000L; // set sw time adjust value
      Year_str = timestr.substring(20,24);
      Year = Year_str.toInt() - 2000;  // save year as byte
    }
  }
  updateOutputState();
  // set port value for output
  if (outputState == true) {
    digitalWrite(Relay1_pin, 1);    // set Relay1 output active
    digitalWrite(Red_LED_pin, 0);   // set red LED active
  }
  else {
    digitalWrite(Relay1_pin, 0);    // disable Relay1 output
    digitalWrite(Red_LED_pin, 1);   // set red LED inactive
  }
  // if wifi: look for an OTA update and client request
  if ((WiFi.status() == WL_CONNECTED)) {
```

```
   digitalWrite(Blue_LED_pin, 0); // set blue LED active
   ArduinoOTA.handle();
   WiFiClient client = server.available();
   if (client) {
    String currentLine = "";                // make a String to hold incoming data from the client
    String header = "";                     // Variable to store the HTTP request
    byte oldHour, oldMinute;
    char hourMSB, hourLSB, minMSB, minLSB, wday;
    while (client.connected()) { // loop while the client's connected
      if (client.available()) {             // if there's bytes to read from the client,
        char c = client.read();             // read a byte, then
        header += c;                        // add to the header
        if (c == '\n') {                    // if end of the client HTTP request
          if (currentLine.length() == 0) {  // then send a response:
            client.println("HTTP/1.0 200 OK");    // send HTTP header
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();
            int len = header.length();
            int idx, timeplan;
            idx = header.indexOf("p"); // index of timeplan day
            if (idx > 0) { timeplan = header[idx+1] - '0'; }
            int wday = header[idx+2] - '0';   // day of week
            if ((idx > 0) && (timeplan == 0) && (wday == 0)) {
              while ((idx < len) && (timeplan < timerPlans)) {
                for (wday = 0; wday < 7; wday++) {
                  oldHour = startTime[wday][timeplan].Hour;
                  oldMinute = startTime[wday][timeplan].Minute;
                  hourMSB = header[idx+4] - '0';
                  hourLSB = header[idx+5] - '0';
                  minMSB = header[idx+9] - '0';
                  minLSB = header[idx+10] - '0';
                  if (((hourMSB >= 0) && (hourMSB <= 2)) && ((hourLSB >= 0) && (hourLSB <= 9)) &&
                    ((minMSB >= 0) && (minMSB <= 5)) && ((minLSB >= 0) && (minLSB <= 9))) {
                    startTime[wday][timeplan].Hour = hourMSB*10 + hourLSB;
                    startTime[wday][timeplan].Minute = minMSB*10 + minLSB;
                    if ((oldHour != startTime[wday][timeplan].Hour) || (oldMinute != startTime[wday
][timeplan].Minute)) {
                      EEPROM.put(EEprom_adr+(wday+ 7*timeplan)*dayturnTimeSize+sizeof(struct
wifiSettings), startTime[wday][timeplan]);
                      EEPROM.commit();
                      onoffMinutes[wday][timeplan].startMinutes = startTime[wday][timeplan].Hour*60
+startTime[wday][timeplan].Minute;
                    }
                  }
                  oldHour = endTime[wday][timeplan].Hour;
                  oldMinute = endTime[wday][timeplan].Minute;
                  hourMSB = header[idx+14] - '0';
                  hourLSB = header[idx+15] - '0';
                  minMSB = header[idx+19] - '0';
                  minLSB = header[idx+20] - '0';
                  if (((hourMSB >= 0) && (hourMSB <= 2)) && ((hourLSB >= 0) && (hourLSB <= 9)) &&
                    ((minMSB >= 0) && (minMSB <= 5)) && ((minLSB >= 0) && (minLSB <= 9))) {
                    endTime[wday][timeplan].Hour = hourMSB*10 + hourLSB;
                    endTime[wday][timeplan].Minute = minMSB*10 + minLSB;
                    if ((oldHour != endTime[wday][timeplan].Hour) || (oldMinute != endTime[wday][
timeplan].Minute)) {
                      EEPROM.put(EEprom_adr+(((2*wday+1)+2*7*timeplan))*sizeof(turnTime)+sizeof(
struct wifiSettings), endTime[wday][timeplan]);
                      EEPROM.commit();
```

```
                    onoffMinutes[wday][timeplan].endMinutes = endTime[wday][timeplan].Hour*60+
endTime[wday][timeplan].Minute;
                  }
                }
                idx = idx + 22; // move URL idx to the next day in the timeplan
              } // for wday < 7
              timeplan++; // traverse to next next timeplan
            } // while (len < .. ) loop
          } // if found timeset
          String s, timeValue, strDay, outputStateStr, pwrStr;
          byte timePart;
          pwrStr = "  Power consumption: " + String(int(powerW*10) / 10) + "." + String(int(
powerW*10) % 10) + " W";
          if (Day == 0) { strDay = "Monday"; }
          if (Day == 1) { strDay = "Tuesday"; }
          if (Day == 2) { strDay = "Wednesday"; }
          if (Day == 3) { strDay = "Thursday"; }
          if (Day == 4) { strDay = "Friday"; }
          if (Day == 5) { strDay = "Saturday"; }
          if (Day == 6) { strDay = "Sunday"; }
          strDay += " " + time2String(); // Weekday hour:minute
          updateOutputState();
          outputStateStr = "The state of the output is ";
          if (outputState == true) { outputStateStr += "ON. "; }
          else { outputStateStr += "OFF. "; }
          client.println("<!DOCTYPE html><html lang=\"en\"><head><title>" + htmlTitle +
"</title><style>td { font-size: 14px; text-align: center; }</style></head><body>"); // center
table cell text
          client.println("<h4> " + strDay + " - " + outputStateStr + pwrStr + "</h4>");
          s = "<form><table><tr><td><b>Output
</b></td><td><b>Monday</b></td><td></td><td><b>Tuesday</b></td><td></td><td><b>Wednesday</b></td><td
></td><td><b>Thursday</b></td><td></td><td><b>Friday</b></td><td></td><td><b>Saturday</b></td><td></
td><td><b>Sunday</b></td></tr>";
          for (idx=0; idx < timerPlans; idx++) {
            s += "<tr><td><b>Timeplan " + String(idx+1) + "</b></td>";
            for (wday=0; wday < 7; wday++) {
              timeValue ="";
              timePart = startTime[wday][idx].Hour;
              if (timePart < 10) { timeValue = "0"; }
              timeValue +=  String(timePart) + ":";
              timePart = startTime[wday][idx].Minute;
              if (timePart < 10) { timeValue += "0"; }
              timeValue += String(timePart);
              s += "<td><input type=\"time\" name=\"p" + String(idx) + String(wday) + "\" value="
 + "\"" + timeValue + "\">";
              timeValue = "";
              timePart = endTime[wday][idx].Hour;
              if (timePart < 10) { timeValue = "0"; }
              timeValue += String(timePart) +":";
              timePart = endTime[wday][idx].Minute;
              if (timePart < 10) { timeValue += "0"; }
              timeValue += String(timePart);
              s += "<input type=\"time\" name=\"e\" value=\"" + timeValue + "\"></td>";
              if (wday != 6) { s += "<td>-</td>"; }
            }  // for wday 0..6
            s += "</tr>";
          }  // for 0..timeplans
          s += "</table><input type=\"submit\" value=\"Save output time plans\" style=\"
border-radius:5px; padding: 5px; margin:5px 0; background-color:#eafafb;\"></form><p></p>";
          client.println(s);
```

```
            client.println("</body></html>");
            client.println();
            break;
          } // len == 0
          else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } // if c == '\n'
        else {
          if (c != '\r') {  // if you got anything else but a carriage return character,
            currentLine += c;      // add it to the end of the currentLine
          }
        }
      } // if client available
      yield();
    }  //  while client connected
    header = "";
    client.stop();
   } // if client
  }  // if wifi connected
  else {  // if no wifi: set blue LED inactive
   digitalWrite(Blue_LED_pin, 1);
  }
  // always time keeping
  if (adjustSWtiming == true) { // check NTP sync.:
    OneMinute = adjustMinute;   // using an adjusted, shorter OneMinute !
    adjustSWtiming = false;     // only once per NTP sync. !
  }
  else { OneMinute = 60 * 1000L; } // just a full minute check
  if (millis() - previousMillis >= OneMinute) {
    previousMillis = millis();
    Minute++;
    if (Minute >= 60) {
      getNTPtime = true;
      Minute = 0;
      Hour++;
      if (Hour >= 24) { // passed midnight ?
        Hour = 0;
        Day++;
        if (Day >= 7) { Day = 0; }
      }
    }
  }
  }  //  while(1)
}

// update the boolean output state array
void updateOutputState() {
  unsigned int timeNow_minutes = Hour*60 + Minute; // current no. of minutes since 00:00
  // check the ON period for the output
  outputState = false;
  for (int tp = 0; tp < timerPlans; tp++) {
    if ((timeNow_minutes < onoffMinutes[Day][tp].endMinutes) && (timeNow_minutes >= onoffMinutes[
Day][tp].startMinutes)) {
      outputState = true;
    }
  }
}

String time2String() {
```

```
  String time_2digits = "";
  if (Hour <= 9) { time_2digits += "0"; }
  time_2digits += String(Hour) + ":";
  if (Minute <= 9) { time_2digits += "0"; }
  time_2digits += String(Minute);
  return time_2digits;
}

bool setupButton() {
  bool result = false;
  if (digitalRead(setup_Pin) == 0) {
    delay(20);    // simple anti bounce delay
    if (digitalRead(setup_Pin) == 0) { result = true; }
  }
  return result;
}

void handlePortal() {
  if (APserver.method() == HTTP_POST) {
    struct wifiSettings temp_user_wifi;
    strncpy(temp_user_wifi.ssid,     APserver.arg("ssid").c_str(),     sizeof(user_wifi.ssid));
    strncpy(temp_user_wifi.password, APserver.arg("password").c_str(), sizeof(user_wifi.password));
    temp_user_wifi.ssid[APserver.arg("ssid").length()] = temp_user_wifi.password[APserver.arg(
"password").length()] = '\0';
    if ((temp_user_wifi.ssid != user_wifi.ssid) || (temp_user_wifi.password != user_wifi.password))
 {
      user_wifi = temp_user_wifi;
      EEPROM.put(0, user_wifi);
      EEPROM.commit();
    }
    APserver.send(200, "text/html", "<!doctype html><html lang='en'><head><meta
charset='utf-8'><meta name='viewport' content='width=device-width, initial-scale=1'><title>Wifi
Setup</title><style>*,::after,::before{box-sizing:border-box;}body{margin:0;font-family:'Segoe
UI',Roboto,'Helvetica Neue',Arial,'Noto Sans','Liberation
Sans';font-size:1rem;font-weight:400;line-height:1.5;color:#212529;background-color:#f5f5f5;}.form-c
ontrol{display:block;width:100%;height:calc(1.5em + .75rem + 2px);border:1px solid
#ced4da;}button{border:1px solid
transparent;color:#fff;background-color:#007bff;border-color:#007bff;padding:.5rem
1rem;font-size:1.25rem;line-height:1.5;border-radius:.3rem;width:100%}.form-signin{width:100%;max-wi
dth:400px;padding:15px;margin:auto;}h1,p{text-align: center}</style></head><body><main
class='form-signin'><h1>Wifi Setup</h1><br><br><p>The wifi settings have been saved.<br><br>The
device will restarted.</p></main></body></html>" );
    delay(200);     // delay to allow the response to be send to the client..
    ESP.restart();  // ..before resetting for setup() to try the ssid, password
  } else {
    APserver.send(200, "text/html", "<!doctype html><html lang='en'><head><meta
charset='utf-8'><meta name='viewport' content='width=device-width, initial-scale=1'><title>Wifi
Setup</title> <style>*,::after,::before{box-sizing:border-box;}body{margin:0;font-family:'Segoe
UI',Roboto,'Helvetica Neue',Arial,'Noto Sans','Liberation
Sans';font-size:1rem;font-weight:400;line-height:1.5;color:#212529;background-color:#f5f5f5;}.form-c
ontrol{display:block;width:100%;height:calc(1.5em + .75rem + 2px);border:1px solid
#ced4da;}button{cursor: pointer;border:1px solid
transparent;color:#fff;background-color:#007bff;border-color:#007bff;padding:.5rem
1rem;font-size:1.25rem;line-height:1.5;border-radius:.3rem;width:100%}.form-signin{width:100%;max-wi
dth:400px;padding:15px;margin:auto;}h1{text-align: center}</style></head><body><main
class='form-signin'><form action='/' method='post'><h1 class=''>Wifi Setup</h1><br><div
class='form-floating'><label>SSID</label><input type='text' class='form-control'
name='ssid'></div><div class='form-floating'><br><label>Password</label><input type='password'
class='form-control' name='password'></div><br><br><button
type='submit'>Save</button></form></main> </body></html>" );
```

```
  }
}
void initOTA() {
    // Port defaults to 8266
    // ArduinoOTA.setPort(8266);

    // Hostname defaults to esp8266-[ChipID]
    // ArduinoOTA.setHostname("smart_plug");

    // No authentication by default
    // ArduinoOTA.setPassword("admin");

    // Password can be set with it's md5 value as well
    // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
    // ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");

    ArduinoOTA.onStart([]() {
            String type;
            if (ArduinoOTA.getCommand() == U_FLASH) {
                    type = "sketch";
            } else { // U_FS
                    type = "filesystem";
            }
    });

    ArduinoOTA.onEnd([]() {     });

    ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
//          Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
    });

    ArduinoOTA.onError([](ota_error_t error) { //   Serial.printf("Error[%u]: ", error);
            if (error == OTA_AUTH_ERROR) { //       Serial.println("Auth Failed");
            } else if (error == OTA_BEGIN_ERROR) { //   Serial.println("Begin Failed");
            } else if (error == OTA_CONNECT_ERROR) { // Serial.println("Connect Failed");
            } else if (error == OTA_RECEIVE_ERROR) { // Serial.println("Receive Failed");
            } else if (error == OTA_END_ERROR) {// Serial.println("End Failed");
            }
    });
    ArduinoOTA.begin();
}

void CF_impulse() { // Captures count of pulses from CF_PIN
  msNow = millis();
  pulseWidth = msNow - msPreviousPulse;
  msPreviousPulse = msNow;
  pulseFreq = float(1000 / pulseWidth);
}
```