```
/*----- Web controlled lux smart plug based on the Sonoff S26 with BH1750 ------
Select board to be: Generic ESP8266 Module
-----
A brief hard- and software description is found at: https://www.larsenhenneberg.dk
-----
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 an active setup button, an Access Point (AP) is started, see .05
.04: The program continues into the loop(), see .10
.05: Use f.ex. a mobile phone to use the AP network.
.06: Browse to the captive portal at http://192.168.4.1
.07: Enter the wifi ssid and password at the captive portal web page and press Save.
.08: The ssid and password from the captive portal are saved into the EEprom.
.09: The ESP8266 is restarted, and starts from .01
.10: In the loop(): in case of an active AP, the captive portal web client is handled.
.11: A while(1) loop inside loop() is now the main loop.
.12: The yield() function is called in while(1) to avoid a watch dog program reset.
.13: Upon an active setup button, the wifi is diconnected and the ESP8266 is restarted, see .01
-----
The light levels for turning the output relay ON/OFF are set via the web page
at the http://Ip-addr_of_smart_light_plug:S26_port .
-----*/
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <ArduinoOTA.h>
#include <EEPROM.h>
#include <ESP8266WebServer.h>
#include <Wire.h>
#include <BH1750.h>

BH1750 lightMeter;

#define S26_port 8058       // Port for webpage access

#define setup_Pin    0      // GPIO0 as active low input to invoke the captive portal
#define Blue_LED_pin 13     // GPIO13 as output , active high for blue LED light
#define Relay_pin 12        // GPIO12 as output , active high for relay and red LED on
#define sda_Pin 4           // GPIO4 as output for BH1750 sda
#define scl_Pin 5           // GPIO5 as output for BH1750 scl
#define endRange 240        // end range value for the HTML slider

#define EEPROM_SIZE 512
#define EEprom_adr 0

// Soft AP ( Access Point) network setup for the captive portal
const char* APssid = "Smart lux plug";
const char* APpassword = "@Klamar49";

bool APactive;
ESP8266WebServer APserver(80);  // this webservice is for AP only
WiFiServer server(S26_port);    // webserver for client to set the lux levels

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

String header;  // Variable to store the HTTP request
String outputState; // The current output state
unsigned long currentTime = millis(); // Current time
unsigned long previousTime = 0;       // Previous time
```

```
const long timeoutTime = 2000;        // Connection timeout in msec.
unsigned int intLux, sensorLux;       // current lux value
bool relayActive;

struct luxSettings {
  unsigned int luxOn;
  unsigned int luxOff;
} luxSetting = {};

void setup() {
  pinMode(Blue_LED_pin, OUTPUT);   // Blue LED pin as output
  pinMode(Relay_pin, OUTPUT);      // Relay and red LED pin as output
  digitalWrite(Blue_LED_pin, 1);   // set blue LED inactive
  digitalWrite(Relay_pin, 0);      // set relay inactive
  EEPROM.begin(EEPROM_SIZE);
  EEPROM.get(0, user_wifi);
  EEPROM.get(sizeof(user_wifi), luxSetting);
  EEPROM.end();
  // test lux max limits
  int maxLux = int(pow(endRange,2));
  // the limit test condition only fufill on an empty EEprom with FFs
  if ((luxSetting.luxOff > maxLux) || (luxSetting.luxOn > maxLux)) {
    luxSetting.luxOn = 1;
    luxSetting.luxOff = maxLux;
  }
  // for 5 sec., try to connect wifi with ssid and password from EEprom
  if (WiFi.status() != WL_CONNECTED) {
    int dly5sec = 5;
    WiFi.mode(WIFI_STA);
    WiFi.begin(user_wifi.ssid, user_wifi.password);
    while ((WiFi.status() != WL_CONNECTED) && (dly5sec > 0)) {
      digitalWrite(Blue_LED_pin, 1);
      delay(500);
      digitalWrite(Blue_LED_pin, 0);
      delay(500);
      dly5sec--;
    }
  }
  // go into AP mode on an active setup button
  APactive = false;
  if (setupButton()){
    WiFi.mode(WIFI_AP);
    WiFi.softAP(APssid, APpassword);
    APserver.on("/",  handlePortal);      // start captive portal
    APserver.begin();
    APactive = true;
  }
  initOTA();    // start the "Over The Air" firmware update option
  Wire.begin(sda_Pin,scl_Pin);  // start i2c communication
  lightMeter.begin();
  server.begin(); // start the server to get client start & end switch plug time
}

void loop() {
  if (APactive == true) {
    while (WiFi.status() != WL_CONNECTED) { // waiting for AP client
      digitalWrite(Blue_LED_pin, 1);
      delay(100);                       // short blink to indicate
      digitalWrite(Blue_LED_pin, 0);    // an active captive portal
      delay(500);
      APserver.handleClient();             // to finish portal
    }
    APserver.close();   // Close the APserver
  }
```

```
  while (1) {  // going into the main loop !
    yield();   // disable wdt to avoid reset
    if (setupButton()) {  // setup button pressed ?
      WiFi.disconnect();  // disconnect wifi
      ESP.restart();      // restart to continue with AP
    }
  updateOutput();
  // 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
    currentTime = millis();
    previousTime = currentTime;
    int idxOn, idxOff;
    unsigned int oldLuxOn, oldLuxOff;
    char highMSB, highLSB, lowMSB, lowLSB;
    while (client.connected() && ((currentTime - previousTime) <= timeoutTime)) { // loop while
the client's connected
      currentTime = millis();
      if (client.available()) {             // if there's bytes to read from the client,
        char c = client.read();             // read a byte, then
        header += c;
        if (c == '\n') {            // if end of the client HTTP request
          if (currentLine.length() == 0) {  // then send a response:
            client.println("HTTP/1.1 200 OK");    // send HTTP header
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();
            String urlLuxOn = "luxon";
            String urlLuxOff = "luxoff";
            if ((header.indexOf(urlLuxOn) >= 0) && (header.indexOf(urlLuxOff) >= 0)) {
              idxOn = header.indexOf(urlLuxOn);
              idxOff = header.indexOf(urlLuxOff);
              oldLuxOn = luxSetting.luxOn;
              oldLuxOff = luxSetting.luxOff;
              int antalDigits = 1;
              for (int i = idxOn+7; i < idxOn+11; i++) {
                if (header[i] != '&') { antalDigits++; }
                else break;
              }
              highMSB = 0;
              highLSB = 0;
              lowMSB = 0;
              lowLSB = 0;
              if (antalDigits == 1) { lowLSB = header[idxOn+6] - '0';  }
              if (antalDigits == 2) {
                lowMSB = header[idxOn+6] - '0';
                lowLSB = header[idxOn+7] - '0';
              }
              if (antalDigits == 3) {
                highLSB = header[idxOn+6] - '0';
                lowMSB = header[idxOn+7] - '0';
                lowLSB = header[idxOn+8] - '0';
              }
              if (antalDigits == 4) {
                highMSB = header[idxOn+6] - '0';
                highLSB = header[idxOn+7] - '0';
                lowMSB = header[idxOn+8] - '0';
                lowLSB = header[idxOn+9] - '0';
              }
```

```
            if (((highMSB >= 0) && (highMSB <= 9)) && ((highLSB >= 0) && (highLSB <= 9)) &&
                ((lowMSB >= 0) && (lowMSB <= 9)) && ((lowLSB >= 0) && (lowLSB <= 9))) {
              luxSetting.luxOn = highMSB*1000 + highLSB*100 + lowMSB*10 + lowLSB;
              antalDigits = 1;
              for (int i = idxOff+8; i < idxOff+12; i++) {
                if ((header[i] >= '0') && (header[i] <= '9')) { antalDigits++; }
                else break;
              }
              highMSB = 0;
              highLSB = 0;
              lowMSB = 0;
              lowLSB = 0;
              if (antalDigits == 1) { lowLSB = header[idxOff+7] - '0';  }
              if (antalDigits == 2) {
                lowMSB = header[idxOff+7] - '0';
                lowLSB = header[idxOff+8] - '0';
              }
              if (antalDigits == 3) {
                highLSB = header[idxOff+7] - '0';
                lowMSB = header[idxOff+8] - '0';
                lowLSB = header[idxOff+9] - '0';
              }
              if (antalDigits == 4) {
                highMSB = header[idxOff+7] - '0';
                highLSB = header[idxOff+8] - '0';
                lowMSB = header[idxOff+9] - '0';
                lowLSB = header[idxOff+10] - '0';
              }
              if (((highMSB >= 0) && (highMSB <= 9)) && ((highLSB >= 0) && (highLSB <= 9)) &&
                ((lowMSB >= 0) && (lowMSB <= 9)) && ((lowLSB >= 0) && (lowLSB <= 9))) {
                luxSetting.luxOff = highMSB*1000 + highLSB*100 + lowMSB*10 + lowLSB;
                if ((oldLuxOn != luxSetting.luxOn) || (oldLuxOff != luxSetting.luxOff)) {
                  EEPROM.begin(EEPROM_SIZE);
                  EEPROM.put(sizeof(struct wifiSettings), luxSetting);
                  EEPROM.end();
                }
              }
            }
          }
          updateOutput(); // update output to let the web response reflect any changes
          String relayStr;
          if (relayActive == true) { relayStr = "ON"; } else { relayStr = "OFF"; }
          client.println("<!DOCTYPE html><html><head>");
          client.println("<meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\"><title>Smart lux plug</title>");
          client.println("<style>.slider { width: 70%; height: 25px; background: #d3d3d3;
}</style></head>");
          client.println("<body><h2>Smart plug light levels</h2><h3><div>Current light level:
" + String(sensorLux) + " lx</div><br>");
          client.println("<div>Output relay is " + relayStr + ".</div><br>");
          client.println("<form oninput=\"luxOn.value = this.value\">");
          client.println("<label>Min. light level to turn On</label><br>");
          client.println("<input type=\"range\" name=\"luxon\" min=\"0\" max=" + String(
endRange) + " step=\"1\" class=\"slider\" value=");
          String lightLevel = String(luxSetting.luxOn);
          String sensorLevel = String(int(pow(luxSetting.luxOn,2)));
          client.println(lightLevel + ">" + sensorLevel + " lux");
          client.println("<br><br><label>Max. light level to turn Off</label><br>");
          client.println("<input type=\"range\" name=\"luxoff\" min=\"0\" max=" + String(
endRange) + " step=\"1\" class=\"slider\" value=");
          lightLevel = String(luxSetting.luxOff);
          sensorLevel = String(int(pow(luxSetting.luxOff,2)));
          client.println(lightLevel + ">"  + sensorLevel + " lux<br>");
          client.println("<input type=\"submit\" value=\"Save\" style=\" border-radius:5px;
```

```
padding: 5px; margin:5px 0; background-color:#eafafb;\"></form></h3></body></html>");
            client.println();
            break;
          } else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage return character,
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    header = "";
    client.stop();
   }
  }
  else {  // if no wifi: set blue LED inactive
   digitalWrite(Blue_LED_pin, 1);
  }
  } // while(1)
}

void updateOutput()
{ // read Lux and set the smart plug output accordingly
  float lux = lightMeter.readLightLevel();
  sensorLux = floor(lux);
  intLux = pow(sensorLux,0.5); // HTML slider value = sqr(intLux)
  if (sensorLux == 3) { intLux = 2; } // sensorlux at 3 becomes 2 !!
  if ((intLux >= luxSetting.luxOn) && (intLux <= luxSetting.luxOff)) {
    digitalWrite(Relay_pin, 1); // set relay and red LED active
    relayActive = true;
  }
  else { // is lux level less than ON-level or larger than OFF-level
    if ((intLux < luxSetting.luxOn) || (intLux > luxSetting.luxOff)) {
      digitalWrite(Relay_pin, 0); // disable relay and red LED
      relayActive = false;
    }
    else { // current light level at 0 and max. settings for output OFF ?
      if ((intLux == 0) && (luxSetting.luxOff == 0)) {
        digitalWrite(Relay_pin, 0); // disable relay and red LED
        relayActive = false;
      }
    }
  }
}
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.begin(sizeof(struct wifiSettings));
      EEPROM.put(0, user_wifi);
      EEPROM.end();
    }
    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;}.for
m-control{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
-width: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;}.for
m-control{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
-width: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();
}
```