```
/*********
  Code by Raphael Beckmann
  Complete project details at https://www.letsbuildthings.ch/
*********/

/*********
 * Configuration
 *********/
// Bibliotheken
#include <Arduino.h>
#include <Wire.h>
#include <WiFi.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#include <Battery18650Stats.h>
#include <Adafruit_BMP280.h>
#include <Adafruit_BME680.h>
#include <PubSubClient.h>
#include <Adafruit_SGP30.h>

// WLAN-Daten
const char *ssid = "carpe_diem!";
const char *password = "QWERT12345!";

// IP Adresse der Sensorphalanx
IPAddress local_IP(192, 168, 0, 199);
// Gateway IP Adresse
IPAddress gateway(192, 168, 0, 1);
IPAddress subnet(255, 255, 255, 0);

// MQTT
IPAddress mqtt_server(192, 168, 0, 200);
// MQTT Cient Name
const char *MQTT_Name = "Sensorphalanx";
const char *MQTT_User = "MQTT-user";
const char *MQTT_Pass = "MQTT";

// Sensorenauswahl
bool HC_SR50X = true;   // Infrarotbewegungsmelder
bool LDR = true;          // LDR Sensor vorhanden
```

```
bool SGP30 = true;         // SGP30 Sensor vorhanden
bool NTC = false;          // NTC Sensor vorhanden
bool BMP280 = false;       // BMP280 Sensor vorhanden
bool BME680 = true;        // BME680 Sensor vorhanden
bool Battery18650 = true; // Battery18650 Sensor vorhanden
bool DHT_11 = false;       // DHT Sensor vorhanden
bool DHT_22 = true;        // DHT Sensor vorhanden
#define DHTTYPE DHT22      // DHT Typ 11 oder 22

// Pindefinition
#define LED_RO_PIN 5  // LED Pin Rot
#define LED_GR_PIN 19 // LED Pin Grün
#define LED_BL_PIN 18 // LED Pin Blau
#define DHT_PIN 26    // DHT_IN Pin
#define LDR_PIN 13    // LDR Pin
#define NTC_PIN 13    // NTC Pin
#define AKKU_PIN 25   // AKKU Pin
#define IFR_BW_PIN 2  // IFR_In Pin / GPIO_NUM_"n" in startDeepSleep() anpassen

// Daten ablage in RTC Speicher
RTC_DATA_ATTR int64_t millis_DeepSleep;
RTC_DATA_ATTR bool BW_ON = 0;
RTC_DATA_ATTR bool BW_ST = 0;

String Backup[60][10];

// Globale Variablen Definition
Adafruit_SGP30 sgp;
int TIME_TO_SLEEP_AND_SGP30 = 60;
int TIME_TO_SLEEP = 30;
struct timeval tv_now;
int64_t time_us;
bool didAction = false;
Battery18650Stats battery(AKKU_PIN, 1.7497, 100);
uint32_t delayMS;
esp_sleep_wakeup_cause_t wakeup_reason;
float Temperature, Pressure, Humidity, Gas, Air_Quality_Score, Volt;
int Charge_LVL, Charge_LVL_TAB, Brightness, WiFiperct, TVOC, eCO2, Raw_H2, Raw_Ethanol;
String Air_Quality_String;

uint32_t getAbsoluteHumidity(float sgptemperature, float sgphumidity)
{
```

```
  // approximation formula from Sensirion SGP30 Driver Integration chapter 3.15
  const float absoluteHumidity = 216.7f * ((sgphumidity / 100.0f) * 6.112f * exp((17.62f * sgptemperature) / (243.12f + sgptemperature)) / (273.15f + sgptemperature)); // [g/m^3]
  const uint32_t absoluteHumidityScaled = static_cast<uint32_t>(1000.0f * absoluteHumidity); // [mg/m^3]
  return absoluteHumidityScaled;
}
int Array_size(String Typ)
{
  int array_size = 0;
  if (LDR == true && Typ == "Brightness")
    array_size++;
  if ((SGP30 == true) && (Typ == "Air_Quality_Score" || Typ == "Gas"))
    array_size++;
  if (NTC == true && Typ == "Temperatur")
    array_size++;
  if ((BMP280 == true) && (Typ == "Temperatur" || Typ == "Pressure"))
    array_size++;
  if ((BME680 == true) && (Typ == "Temperatur" || Typ == "Pressure" || Typ == "Humidity" || Typ == "Air_Quality_Score" || Typ == "Gas"))
    array_size++;
  if ((Battery18650 == true) && (Typ == "Volt" || Typ == "Charge_LVL" || Typ == "Charge_LVL_TAB"))
    array_size++;
  if ((DHT_11 || DHT_22 == true) && (Typ == "Temperatur" || Typ == "Humidity"))
    array_size++;
  if ((SGP30 == true) && (Typ == "TVOC" || Typ == "eCO2" || Typ == "Raw_H2" || Typ == "Raw_Ethanol"))
    array_size++;
  return array_size;
}
void doMoveAction()
{
  Serial.println("--- Interrupt  ---");
  BW_ON = true;
  detachInterrupt(IFR_BW_PIN);
}
void doAction()
{
  attachInterrupt(IFR_BW_PIN, doMoveAction, RISING);
  float arry_Brightness[Array_size("Brightness")];
  for (int n = 0; n != (sizeof(arry_Brightness) / sizeof(int)); n++)
  {
    arry_Brightness[n] = 0.00;
  }
```

```
float arry_Temp[Array_size("Temperatur")];
for (int n = 0; n != (sizeof(arry_Temp) / sizeof(int)); n++)
{
  arry_Temp[n] = 0.00;
}
float arry_Pressure[Array_size("Pressure")];
for (int n = 0; n != (sizeof(arry_Pressure) / sizeof(int)); n++)
{
  arry_Pressure[n] = 0.00;
}
float arry_Humidity[Array_size("Humidity")];
for (int n = 0; n != (sizeof(arry_Humidity) / sizeof(int)); n++)
{
  arry_Humidity[n] = 0.00;
}
float arry_Gas[Array_size("Gas")];
for (int n = 0; n != (sizeof(arry_Gas) / sizeof(int)); n++)
{
  arry_Gas[n] = 0.00;
}
float arry_Air_Quality_Score[Array_size("Air_Quality_Score")];
for (int n = 0; n != (sizeof(arry_Air_Quality_Score) / sizeof(int)); n++)
{
  arry_Air_Quality_Score[n] = 0.00;
}
float arry_Volt[Array_size("Volt")];
for (int n = 0; n != (sizeof(arry_Volt) / sizeof(int)); n++)
{
  arry_Volt[n] = 0.00;
}
int arry_Charge_LVL[Array_size("Charge_LVL")];
for (int n = 0; n != (sizeof(arry_Charge_LVL) / sizeof(int)); n++)
{
  arry_Charge_LVL[n] = 00;
}
int arry_Charge_LVL_TAB[Array_size("Charge_LVL_TAB")];
for (int n = 0; n != (sizeof(arry_Charge_LVL_TAB) / sizeof(int)); n++)
{
  arry_Charge_LVL_TAB[n] = 00;
}
int arry_TVOC[Array_size("TVOC")];
for (int n = 0; n != (sizeof(arry_TVOC) / sizeof(int)); n++)
```

```
{
  arry_TVOC[n] = 0;
}
int arry_eCO2[Array_size("eCO2")];
for (int n = 0; n != (sizeof(arry_eCO2) / sizeof(int)); n++)
{
  arry_eCO2[n] = 0;
}
int arry_Raw_H2[Array_size("Raw_H2")];
for (int n = 0; n != (sizeof(arry_Raw_H2) / sizeof(int)); n++)
{
  arry_Raw_H2[n] = 0;
}
int arry_Raw_Ethanol[Array_size("Raw_Ethanol")];
for (int n = 0; n != (sizeof(arry_Raw_Ethanol) / sizeof(int)); n++)
{
  arry_Raw_Ethanol[n] = 0;
}

// DHT Sensor Enlesen
if (DHT_11 || DHT_22 == true)
{
  DHT_Unified dht(DHT_PIN, DHTTYPE);
  dht.begin();
  sensors_event_t event;
  dht.temperature().getEvent(&event);
  if (isnan(event.temperature))
  {
    Serial.println(F("Fehler beim einlesen der Temperatur!"));
  }
  else
  {
    int i = 0;
    while (arry_Temp[i] != 0.00)
    {
      i++;
    }
    arry_Temp[i] = event.temperature;
  }
  dht.humidity().getEvent(&event);
  if (isnan(event.relative_humidity))
  {
```

```
208       Serial.println(F("Fehler beim einlesen der Luftfeuchtigkeit!"));
209     }
210     else
211     {
212       int i = 0;
213       while (arry_Humidity[i] != 0.00)
214       {
215         i++;
216       }
217       arry_Humidity[i] = event.relative_humidity;
218     }
219   }
220   // LDR Sensor Enlesen
221   if (LDR == true)
222   {
223     int i = 0;
224     while (arry_Brightness[i] != 0.00)
225     {
226       i++;
227     }
228     arry_Brightness[i] = (analogRead(LDR_PIN));
229   }
230   // NTC Sensor Enlesen
231   if (NTC == true)
232   {
233
234     int Vo = 0;
235     float R1 = 10000;
236     float logR2, R2, T, Tc;
237     float c1 = 1.009249522e-03, c2 = 2.378405444e-04, c3 = 2.019202697e-07;
238     for (int i = 0; i <= 100; i++)
239     { // 100 Messungen zur Stabeliesierung des Messwertes
240       Vo = Vo + analogRead(NTC_PIN);
241     }
242     Vo = Vo / 100; // Mittelwert aus 100 Messungen erzeugen
243     R2 = R1 * (4095.0 / (float)Vo - 1.0);
244     logR2 = log(R2);
245     T = (1.0 / (c1 + c2 * logR2 + c3 * logR2 * logR2 * logR2));
246     Tc = T - 273.15;
247
248     int i = 0;
249     while (arry_Temp[i] != 0.00)
```

```
    {
      i++;
    }
    arry_Temp[i] = Tc;
  }
  // Battery18650 Sensor Enlesen
  if (Battery18650 == true)
  {
    int i = 0;
    while (arry_Volt[i] != 0.00)
    {
      i++;
    }
    arry_Volt[i] = battery.getBatteryVolts();
    i = 0;
    while (arry_Charge_LVL[i] != 0.00)
    {
      i++;
    }
    arry_Charge_LVL[i] = battery.getBatteryChargeLevel();

    i = 0;
    while (arry_Charge_LVL_TAB[i] != 0.00)
    {
      i++;
    }
    arry_Charge_LVL_TAB[i] = battery.getBatteryChargeLevel(true);
  }
  // HC_SR50X Sensor Enlesen
  if (HC_SR50X == true)
  {
    if (BW_ON == true && BW_ST == false)
    {
      BW_ST = true;
    }
    else if (BW_ON == true && BW_ST == true)
    {
      BW_ON = false;
      BW_ST = false;
    }
  }
  // SGP30 Sensor Enlesen
```

```
if (SGP30 == true && arry_Charge_LVL_TAB[0] > 50)
{
  if (!sgp.begin())
  {
    Serial.println("Sensor not found :(");
    while (1)
      ;
  }
  // sgp.setIAQBaseline(0x7FC7, 0x82A5);
  float sgp2temperature, sgp2humidity;
  if (DHT_11 || DHT_22 == true)
  {
    DHT_Unified dht(DHT_PIN, DHTTYPE);
    dht.begin();
    sensors_event_t event;
    // wenn ein DHT Sensor vorhanden ist kann dieser genutzt werden um die Messung zu verbessern
    dht.temperature().getEvent(&event);
    if (isnan(event.temperature))
    {
      sgp2temperature = 22.1; // [°C]
    }
    else
    {
      sgp2temperature = event.temperature;
    }
    dht.humidity().getEvent(&event);
    if (isnan(event.relative_humidity))
    {
      sgp2humidity = 45.2; // [%RH]
    }
    else
    {
      sgp2humidity = event.relative_humidity;
    }
  }
  else
  {
    sgp2temperature = 22.1; // [°C]
    sgp2humidity = 45.2;    // [%RH]
  }

  sgp.setHumidity(getAbsoluteHumidity(sgp2temperature, sgp2humidity));
```

```
  int i = 0;
  while (arry_TVOC[i] != 0)
  {
    i++;
  }
  for (int x = 0; x <= 60; x++)
  {
    if (!sgp.IAQmeasure())
    {
      Serial.println("IAQ Mesung fehlgeschlagen");
      return;
    }
    delay(500);
  }
  arry_TVOC[i] = sgp.TVOC;
  while (arry_eCO2[i] != 0)
  {
    i++;
  }
  arry_eCO2[i] = sgp.eCO2;

  if (!sgp.IAQmeasureRaw())
  {
    Serial.println("Raw Mesung fehlgeschlagen");
    return;
  }

  i = 0;
  while (arry_Raw_H2[i] != 0)
  {
    i++;
  }
  arry_Raw_H2[i] = sgp.rawH2;

  i = 0;
  while (arry_Raw_Ethanol[i] != 0)
  {
    i++;
  }
  arry_Raw_Ethanol[i] = sgp.rawEthanol;
}
// BMP280 Sensor Enlesen
```

```
376  if (BMP280 == true)
377  {
378    Adafruit_BMP280 bmp;
379    bmp.begin(0x76, 0x58);
380    // Setup oversampling und filter
381    bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,
382                    Adafruit_BMP280::SAMPLING_X2,
383                    Adafruit_BMP280::SAMPLING_X16,
384                    Adafruit_BMP280::FILTER_X16,
385                    Adafruit_BMP280::STANDBY_MS_500);
386    int i = 0;
387    while (arry_Temp[i] != 0.00)
388    {
389      i++;
390    }
391    arry_Temp[i] = bmp.readTemperature() - 1.5;
392
393    i = 0;
394    while (arry_Pressure[i] != 0.00)
395    {
396      i++;
397    }
398    arry_Pressure[i] = bmp.readPressure() / 100;
399  }
400  // BME680 Sensor Enlesen
401  if (BME680 == true)
402  {
403    Adafruit_BME680 bme;
404    float hum_score, gas_score;
405    float gas_reference = 250000;
406    float hum_reference = 40;
407    int getgasreference_count = 0;
408
409    if (!bme.begin())
410    {
411      Serial.println("kann BME680 nicht finden!");
412    }
413    // Setup oversampling und filter
414    bme.setTemperatureOversampling(BME680_OS_8X);
415    bme.setHumidityOversampling(BME680_OS_2X);
416    bme.setPressureOversampling(BME680_OS_4X);
417    bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
```

```
418     bme.setGasHeater(320, 150); // 320*C für 150 ms
419
420     if (!bme.performReading())
421     {
422       Serial.println("Fehler beim lesen");
423       return;
424     }
425     int i = 0;
426     while (arry_Temp[i] != 0.00)
427     {
428       i++;
429     }
430     arry_Temp[i] = bme.temperature - 3;
431
432     i = 0;
433     while (arry_Pressure[i] != 0.00)
434     {
435       i++;
436     }
437     arry_Pressure[i] = bme.pressure / 100;
438
439     i = 0;
440     while (arry_Humidity[i] != 0.00)
441     {
442       i++;
443     }
444     arry_Humidity[i] = bme.humidity;
445
446     i = 0;
447     while (arry_Gas[i] != 0.00)
448     {
449       i++;
450     }
451     arry_Gas[i] = bme.gas_resistance / 1000.0;
452
453     float current_humidity = bme.readHumidity();
454     if (current_humidity >= 38 && current_humidity <= 42)
455       hum_score = 0.25 * 100;
456     else
457     {
458       if (current_humidity < 38)
459         hum_score = 0.25 / hum_reference * current_humidity * 100;
```

```
      else
      {
        hum_score = ((-0.25 / (100 - hum_reference) * current_humidity) + 0.416666) * 100;
      }
    }

    // Umrechnen der gaskonzentration zum IAQ index
    float gas_lower_limit = 5000;  // Limit schlechte Luftquallität
    float gas_upper_limit = 50000; // Limit gute Luftquallität
    if (gas_reference > gas_upper_limit)
      gas_reference = gas_upper_limit;
    if (gas_reference < gas_lower_limit)
      gas_reference = gas_lower_limit;

    gas_score = (0.75 / (gas_upper_limit - gas_lower_limit) * gas_reference - (gas_lower_limit * (0.75 / (gas_upper_limit - gas_lower_limit)))) * 100;
    float airQuality = hum_score + gas_score;

    if ((getgasreference_count++) % 10 == 0)
    {
      int readings = 5;
      for (int i = 1; i <= readings; i++)
      {
        gas_reference += bme.readGas();
      }
      gas_reference = gas_reference / readings;
    }

    i = 0;
    while (arry_Air_Quality_Score[i] != 0.00)
    {
      i++;
    }
    arry_Air_Quality_Score[i] = (100 - airQuality) * 5;

    String IAQ_text = "";
    if (airQuality >= 301)
      IAQ_text += "Gefährlich";
    else if (airQuality >= 201 && airQuality <= 300)
      IAQ_text += "Sehr ungesund";
    else if (airQuality >= 176 && airQuality <= 200)
      IAQ_text += "Ungesund";
```

```
    else if (airQuality >= 151 && airQuality <= 175)
      IAQ_text += "Ungesund für sensible Gruppen";
    else if (airQuality >= 51 && airQuality <= 150)
      IAQ_text += "Mässig";
    else if (airQuality >= 00 && airQuality <= 50)
      IAQ_text += "Gut";
    Serial.print("Luftquallität = ");
    Serial.println(IAQ_text);
  }

  int n;
  if (sizeof(arry_Brightness) != 0)
  {

    for (n = 0; n != (sizeof(arry_Brightness) / sizeof(int)); n++)
    {
      Brightness += arry_Brightness[n];
    }
    Brightness = Brightness / (sizeof(arry_Brightness) / sizeof(int));
  }
  if (sizeof(arry_Temp) != 0)
  {
    for (n = 0; n != (sizeof(arry_Temp) / sizeof(int)); n++)
    {
      Temperature += arry_Temp[n];
    }
    Temperature = Temperature / (sizeof(arry_Temp) / sizeof(int));
  }
  if (sizeof(arry_Pressure) != 0)
  {
    for (n = 0; n != (sizeof(arry_Pressure) / sizeof(int)); n++)
    {
      Pressure += arry_Pressure[n];
    }
    Pressure = Pressure / (sizeof(arry_Pressure) / sizeof(int));
  }
  if (sizeof(arry_Humidity) != 0)
  {
    for (n = 0; n != (sizeof(arry_Humidity) / sizeof(int)); n++)
    {
      Humidity += arry_Humidity[n];
    }
```

```
  Humidity = Humidity / (sizeof(arry_Humidity) / sizeof(int));
}
if (sizeof(arry_Gas) != 0)
{
  for (n = 0; n != (sizeof(arry_Gas) / sizeof(int)); n++)
  {
    Gas += arry_Gas[n];
  }
  Gas = Gas / (sizeof(arry_Gas) / sizeof(int));
}
if (sizeof(arry_Air_Quality_Score) != 0)
{
  for (n = 0; n != (sizeof(arry_Air_Quality_Score) / sizeof(int)); n++)
  {
    Air_Quality_Score += arry_Air_Quality_Score[n];
  }
  Air_Quality_Score = Air_Quality_Score / (sizeof(arry_Air_Quality_Score) / sizeof(int));
}
if (sizeof(arry_Volt) != 0)
{
  for (n = 0; n != (sizeof(arry_Volt) / sizeof(int)); n++)
  {
    Volt += arry_Volt[n];
  }
  Volt = Volt / (sizeof(arry_Volt) / sizeof(int));
}
if (sizeof(arry_Charge_LVL) != 0)
{
  for (n = 0; n != (sizeof(arry_Charge_LVL) / sizeof(int)); n++)
  {
    Charge_LVL += arry_Charge_LVL[n];
  }
  Charge_LVL = Charge_LVL / (sizeof(arry_Charge_LVL) / sizeof(int));
}
if (sizeof(arry_Charge_LVL) != 0)
{
  for (n = 0; n != (sizeof(arry_Charge_LVL_TAB) / sizeof(int)); n++)
  {
    Charge_LVL_TAB += arry_Charge_LVL_TAB[n];
  }
  Charge_LVL_TAB = Charge_LVL_TAB / (sizeof(arry_Charge_LVL_TAB) / sizeof(int));
}
```

```
585  if (sizeof(arry_TVOC) != 0)
586  {
587    for (n = 0; n != (sizeof(arry_TVOC) / sizeof(int)); n++)
588    {
589      TVOC += arry_TVOC[n];
590    }
591    TVOC = TVOC / (sizeof(arry_TVOC) / sizeof(int));
592  }
593  if (sizeof(arry_eCO2) != 0)
594  {
595    for (n = 0; n != (sizeof(arry_eCO2) / sizeof(int)); n++)
596    {
597      eCO2 += arry_eCO2[n];
598    }
599    eCO2 = eCO2 / (sizeof(arry_eCO2) / sizeof(int));
600  }
601  if (sizeof(arry_Raw_H2) != 0)
602  {
603    for (n = 0; n != (sizeof(arry_Raw_H2) / sizeof(int)); n++)
604    {
605      Raw_H2 += arry_Raw_H2[n];
606    }
607    Raw_H2 = Raw_H2 / (sizeof(arry_Raw_H2) / sizeof(int));
608  }
609  if (sizeof(arry_Raw_Ethanol) != 0)
610  {
611    for (n = 0; n != (sizeof(arry_Raw_Ethanol) / sizeof(int)); n++)
612    {
613      Raw_Ethanol += arry_Raw_Ethanol[n];
614    }
615    Raw_Ethanol = Raw_Ethanol / (sizeof(arry_Raw_Ethanol) / sizeof(int));
616  }
617  Serial.print("IFR_BW_PIN is:  ");
618  Serial.println(BW_ON);
619  Serial.print("Brightness: ");
620  Serial.println(Brightness);
621  Serial.print("Temperature: ");
622  Serial.println(Temperature);
623  Serial.print("Pressure: ");
624  Serial.println(Pressure);
625  Serial.print("Humidity: ");
626  Serial.println(Humidity);
```

```
  Serial.print("Gas: ");
  Serial.println(Gas);
  Serial.print("Air_Quality_Score: ");
  Serial.println(Air_Quality_Score);
  Serial.print("Volt: ");
  Serial.println(Volt);
  Serial.print("Charge_LVL: ");
  Serial.println(Charge_LVL);
  Serial.print("Charge_LVL_TAB: ");
  Serial.println(Charge_LVL_TAB);
  Serial.print("TVOC: ");
  Serial.println(TVOC);
  Serial.print("eCO2: ");
  Serial.println(eCO2);
  Serial.print("Raw_H2: ");
  Serial.println(Raw_H2);
  Serial.print("Raw_Ethanol: ");
  Serial.println(Raw_Ethanol);

  Serial.println("");
  Serial.println("-----ENDE-----");
  Serial.println("");
}
void StartWlan()
{
  digitalWrite(LED_BL_PIN, HIGH);
  digitalWrite(LED_RO_PIN, LOW);
  WiFi.begin(ssid, password);

  if (!WiFi.config(local_IP, gateway, subnet))
  {
    Serial.println("Fehler bei der Konfiguration");
  }
  int n = 0;
  while (WiFi.status() != WL_CONNECTED && n != 5)
  {
    digitalWrite(LED_GR_PIN, LOW);
    delay(400);
    Serial.print("Verbindungsversuch...");
    digitalWrite(LED_GR_PIN, HIGH);
    delay(400);
    n++;
```

```
669   }
670
671   if (WiFi.status() == WL_CONNECTED)
672   {
673     digitalWrite(LED_GR_PIN, LOW);
674     Serial.println("");
675     Serial.println("WiFi Verbunden.");
676
677     // Berechnung der Verbindungsqualität
678     long rssi = WiFi.RSSI();
679     rssi = -rssi;
680
681     if (rssi < 27) // Berechunung W-LAN Qualli
682     {
683       WiFiperct = 100;
684     }
685     else if (rssi >= 27 && rssi < 33)
686     {
687       WiFiperct = 150 - (5 / 2.7) * rssi;
688     }
689     else if (rssi >= 33 && rssi < 36)
690     {
691       WiFiperct = 150 - (5 / 3) * rssi;
692     }
693     else if (rssi >= 36 && rssi < 40)
694     {
695       WiFiperct = 150 - (5 / 3.3) * rssi;
696     }
697     else if (rssi >= 40 && rssi < 80)
698     {
699       WiFiperct = 150 - (5 / 3.5) * rssi;
700     }
701     else if (rssi >= 80 && rssi < 90)
702     {
703       WiFiperct = 150 - (5 / 3.4) * rssi;
704     }
705     else if (rssi >= 90 && rssi < 99)
706     {
707       WiFiperct = 150 - (5 / 3.3) * rssi;
708     }
709     else
710     {
```

```
      WiFiperct = 0;
    }

    Serial.print("WIFI-Qualli: ");
    Serial.println(WiFiperct);
  }
  else // LED 1 SEC ROT
  {
    digitalWrite(LED_GR_PIN, LOW);
    digitalWrite(LED_BL_PIN, HIGH);
    digitalWrite(LED_RO_PIN, HIGH);
    delay(1000);
    digitalWrite(LED_RO_PIN, LOW);
  }
}
void startDeepSleep()
{
  if (SGP30 == true && Charge_LVL_TAB > 50)
  {
    TIME_TO_SLEEP = TIME_TO_SLEEP_AND_SGP30;
  }
  switch (esp_sleep_get_wakeup_cause())
  {
  case ESP_SLEEP_WAKEUP_EXT0:
    gettimeofday(&tv_now, NULL);
    time_us = (int64_t)tv_now.tv_sec * 1000000L + (int64_t)tv_now.tv_usec;
    esp_sleep_enable_timer_wakeup((TIME_TO_SLEEP * 1000000) - (time_us - millis_DeepSleep));
    break;
  case ESP_SLEEP_WAKEUP_TIMER:
    gettimeofday(&tv_now, NULL);
    time_us = (int64_t)tv_now.tv_sec * 1000000L + (int64_t)tv_now.tv_usec;
    esp_sleep_enable_timer_wakeup((TIME_TO_SLEEP * 1000000) - (time_us - millis_DeepSleep));
    if (HC_SR50X == true)
    {
      esp_sleep_enable_ext0_wakeup(GPIO_NUM_2, HIGH);
    }
    break;
  default:
    gettimeofday(&tv_now, NULL);
    time_us = (int64_t)tv_now.tv_sec * 1000000L + (int64_t)tv_now.tv_usec;
    esp_sleep_enable_timer_wakeup((TIME_TO_SLEEP * 1000000) - (time_us - millis_DeepSleep));
    if (HC_SR50X == true)
```

```
    {
      esp_sleep_enable_ext0_wakeup(GPIO_NUM_2, HIGH);
    }
    break;
  }
  digitalWrite(LED_RO_PIN, LOW);
  digitalWrite(LED_GR_PIN, LOW);
  digitalWrite(LED_BL_PIN, LOW);
  Serial.println("  --  DeepSeep  --  ");
  esp_deep_sleep_start();
}
void MQTT()
{
  WiFiClient espClient;
  PubSubClient client(espClient);
  client.setServer(mqtt_server, 1883);
  while (!client.connected())
  {
    Serial.print("Attempting MQTT connection...");
    // verbindungs versuch
    if (client.connect(MQTT_Name, MQTT_User, MQTT_Pass))
    {
      Serial.println("connected");
      char tempString[8];
      if (esp_sleep_get_wakeup_cause() != ESP_SLEEP_WAKEUP_EXT0)
      {
        dtostrf(Temperature, 1, 2, tempString);
        client.publish("Sensorphalanx/temperature", tempString);
        Serial.print("Sensorphalanx/temperature: ");
        Serial.println(tempString);
        dtostrf(Brightness, 1, 2, tempString);
        client.publish("Sensorphalanx/brightness", tempString);
        Serial.print("Sensorphalanx/brightness: ");
        Serial.println(tempString);
        dtostrf(Pressure, 1, 2, tempString);
        client.publish("Sensorphalanx/Pressure", tempString);
        Serial.print("Sensorphalanx/Pressure: ");
        Serial.println(tempString);
        dtostrf(Humidity, 1, 2, tempString);
        client.publish("Sensorphalanx/Humidity", tempString);
        Serial.print("Sensorphalanx/Humidity: ");
        Serial.println(tempString);
```

```
795         dtostrf(Gas, 1, 2, tempString);
796         client.publish("Sensorphalanx/Gas", tempString);
797         Serial.print("Sensorphalanx/Gas: ");
798         Serial.println(tempString);
799         dtostrf(Air_Quality_Score, 1, 2, tempString);
800         client.publish("Sensorphalanx/Air_Quality_Score", tempString);
801         Serial.print("Sensorphalanx/Air_Quality_Score: ");
802         Serial.println(tempString);
803         dtostrf(Volt, 1, 2, tempString);
804         client.publish("Sensorphalanx/Volt", tempString);
805         Serial.print("Sensorphalanx/Volt: ");
806         Serial.println(tempString);
807         dtostrf(Charge_LVL, 1, 0, tempString);
808         client.publish("Sensorphalanx/Charge_LVL", tempString);
809         Serial.print("Sensorphalanx/Charge_LVL: ");
810         Serial.println(tempString);
811         dtostrf(Charge_LVL_TAB, 1, 0, tempString);
812         client.publish("Sensorphalanx/Charge_LVL_TAB", tempString);
813         Serial.print("Sensorphalanx/Charge_LVL_TAB: ");
814         Serial.println(tempString);
815         dtostrf(WiFiperct, 1, 0, tempString);
816         client.publish("Sensorphalanx/WiFiperct", tempString);
817         Serial.print("Sensorphalanx/WiFiperct: ");
818         Serial.println(tempString);
819         dtostrf(TVOC, 1, 0, tempString);
820         client.publish("Sensorphalanx/TVOC", tempString);
821         Serial.print("Sensorphalanx/TVOC: ");
822         Serial.println(tempString);
823         dtostrf(eCO2, 1, 0, tempString);
824         client.publish("Sensorphalanx/eCO2", tempString);
825         Serial.print("Sensorphalanx/eCO2: ");
826         Serial.println(tempString);
827         dtostrf(Raw_H2, 1, 0, tempString);
828         client.publish("Sensorphalanx/Raw_H2", tempString);
829         Serial.print("Sensorphalanx/Raw_H2: ");
830         Serial.println(tempString);
831         dtostrf(Raw_Ethanol, 1, 0, tempString);
832         client.publish("Sensorphalanx/Raw_Ethanol", tempString);
833         Serial.print("Sensorphalanx/Raw_Ethanol: ");
834         Serial.println(tempString);
835       }
836       detachInterrupt(IFR_BW_PIN);
```

```
      dtostrf(BW_ON, 1, 0, tempString);
      client.publish("Sensorphalanx/Bewegungsmelder", tempString);
      Serial.print("Sensorphalanx/Bewegungsmelder: ");
      Serial.println(tempString);
      Serial.print("  --  ENDE MQTT  --  ");
    }
    else
    {
      Serial.print("Fehler, rc=");
      Serial.print(client.state());
      Serial.println(" warte 5 Sec");
      // warte 5 Sec befor ein neuer Versuch unternommen wird
      delay(5000);
    }
  }
}

void setup()
{
  if (esp_sleep_get_wakeup_cause() != ESP_SLEEP_WAKEUP_EXT0)
  {
    gettimeofday(&tv_now, NULL);
    millis_DeepSleep = (int64_t)tv_now.tv_sec * 1000000L + (int64_t)tv_now.tv_usec;
  }
  Serial.begin(115200);
  Serial.println("  --  WakeUP  --  ");
  pinMode(LED_RO_PIN, OUTPUT);
  pinMode(LED_GR_PIN, OUTPUT);
  pinMode(LED_BL_PIN, OUTPUT);
  pinMode(IFR_BW_PIN, INPUT);
  digitalWrite(LED_GR_PIN, HIGH);
}
void loop()
{
  switch (esp_sleep_get_wakeup_cause())
  {
  case ESP_SLEEP_WAKEUP_EXT0:
    doMoveAction();
    if (BW_ST == true)
    {
      BW_ST = false;
      startDeepSleep();
```

```
    }
    break;
  case ESP_SLEEP_WAKEUP_TIMER:
    doAction();
    break;
  default:
    doAction();
    break;
  }
  StartWlan();
  MQTT();
  startDeepSleep();
}
```