

Datalogger

Besonderheiten bei den Variablen

Das Programm verwendet Variablen, die zur Kompellierung aufgelöst werden. Diese sind vor dem Setup-Teil mit `#variable wert` gekennzeichnet. Der Vorteil dieser Variablen besteht, darin, dass sie keinen Speicherplatz verbrauchen und es trotzdem ermöglichen das Programm variabel und einfach modifizierbar zu gestalten.

Neben den Variablen findet sich auch folgende Kondition, die auch während der Kompillierung aufgelöst wird. Ist `ECHO_TO_SERIAL` auf `false` gesetzt, wird der eingeschlossene Programmteil also einfach ignoriert und benötigt damit weder Speicher noch Rechenleistung

```
#if ECHO_TO_SERIAL
...
#endif //ECHO_TO_SERIAL
```

Verwendung von interrupts

Das Programm verwendet einen interrupt für den Hallsensor, welcher im Setup-Teil definiert wird.

Mit der Zeile

```
attachInterrupt(digitalPinToInterrupt(hallSensor),
  ↳ hallSensorTriggered, FALLING);
```

wird der Arduino darauf angewiesen, jedes mal, wenn der Wert am Port `hallSensor` von `HIGH` auf `LOW` fällt, die Funktion `hallSensorTriggered()` aufzurufen, welche dann `lastLallSesnorTriggerTime` auf einen neuen Wert setzt, der irgendwann im Loop-Teil engültig in die Log-Datei übertragen wird.

```
// Diese Funktion wird immer aufgerufen, wenn die Ausgabe des Hall
↳ Sesnor von HIGH zu LOW wechselt (siehe setup)
void hallSensorTriggered() {
  // Falls das Programm am laufen ist
  if (_running) {
    // Setze die Zeit des letzten Aufrufs, auf die aktuelle Zeit
    lastLallSesnorTriggerTime = millis();
  }
}
```

Theoretisch ist es möglich, dass der Wert überschrieben wird, bevor er in die Log-Datei übertragen werden konnte. In der Praxis ist dies allerdings sehr unwahrscheinlich, da dieser etwa 15 bis 20 mal pro Sekunde durchlaufen wird.

Programmablauf

Setup-Teil

Zunächst wird im Setup-Teil des Programmes alles initialisiert, was im späteren Verlauf verwendet wird. Darunter sind unter anderem die Log-Files, der BME, sowie weitere Sensoren und auch die LEDs. Hier wird stets geprüft, ob der entsprechende Prozess erfolgreich verlief, ist das der Fall, so wird am Ende des Programmes gewartet, bis der An-/Ausknopf gedrückt wird. Gab es einen Fehler, so wird dieser über die Status LEDs kommuniziert und zudem am seriellen Monitor ausgegeben.

```
void setup(void)
{
    // Kontrolliert ob ein Fehler innerhalb des setup Teils
    ↪ aufgetreten ist
    bool errorOccuredInSetup = false;

    // Initialisiert den seriellen Monitor
    Serial.begin(9600);

    // Setzt den Pin-Mode der Status-LEDs auf OUTPUT
    pinMode(redLED, OUTPUT);
    pinMode(yellowLED, OUTPUT);
    pinMode(greenLED, OUTPUT);

    // Setzt den Pin-Mode des An- Ausschalters auf INPUT
    pinMode(onOffButton, INPUT);

    // Setzt den Pin-Mode des Hallsesnors auf INPUT
    pinMode(hallSensor, INPUT);

    // Attached die Funktion hallSensorTriggered, zu einem Ändern
    ↪ der Voltage am Port hallSensor von HIGH zu LOW
    attachInterrupt(digitalPinToInterrupt(hallSensor),
    ↪ hallSensorTriggered, FALLING);

    // Die gelbe LED wird für den restlichen Setup-Teil
    ↪ angeschaltet
    digitalWrite(yellowLED, HIGH);

    Serial.println("Initializing SD card...");

    // Setzt den SD-Karten Pin vorsichtshalber bereits auf OUTPUT
    pinMode(10, OUTPUT);
```

```
// Falls die SD-Karte nicht initialisiert werden kann, wird ein
↳ Fehler ausgegeben
if (!SD.begin(chipSelect)) {
    error("Card failed, or not present"); // Ruft die error
↳ Methode auf, die sich dann um den Fehler kümmert
    errorOccuredInSetup = true; // Setzt errorOccuredInSetup auf
↳ wahr, sodass am Ende nicht in den loop Teil übergegangen wird
}
Serial.println("Card initialized.");

// Erstellt ein neues Log-File mit dem Namen LOGGER +
↳ zweistellige noch nicht verwendete Zahl
char filename[] = "LOGGER00.CSV";
// Loopt durch alle möglichen Namen
for (uint8_t i = 0; i < 100; i++) {
    filename[6] = i / 10 + '0';
    filename[7] = i % 10 + '0';
    if (!SD.exists(filename)) {
        // Falls der Name noch nicht belegt ist, wird das Log-File
        ↳ erstellt und der Loop unterbrochen
        logfile = SD.open(filename, FILE_WRITE);
        break;
    }
}

// Erstellt ein neues Log-File mit dem Namen HALLOG00 +
↳ zweistellige noch nicht verwendete Zahl
char hall_trigger_filename[] = "HALLOG00.CSV";
// Loopt durch alle möglichen Namen
for (uint8_t i = 0; i < 100; i++) {
    hall_trigger_filename[6] = i / 10 + '0';
    hall_trigger_filename[7] = i % 10 + '0';
    if (!SD.exists(hall_trigger_filename)) {
        // Falls der Name noch nicht belegt ist, wird das Log-File
        ↳ erstellt und der Loop unterbrochen
        hall_trigger_logfile = SD.open(hall_trigger_filename,
↳ FILE_WRITE);
        break;
    }
}

// Falls eine der beiden Log-Dateien nicht erstellt wurde (weil
↳ z.B. alle Namen belegt waren), wird ein Fehler ausgegeben
if (!logfile) {
    error("Could not create a logfile"); // Ruft die error
↳ Methode auf, die sich dann um den Fehler kümmert
```

```
    errorOccuredInSetup = true; // Setzt errorOccuredInSetup auf
    ↪ wahr, sodass am Ende nicht in den loop Teil übergegangen wird
  } else if (!hall_trigger_logfile) {
    error("Could not create a logfile for the hall sensor
    ↪ triggers"); // Ruft die error Methode auf, die sich dann um
    ↪ den Fehler kümmert
    errorOccuredInSetup = true; // Setzt errorOccuredInSetup auf
    ↪ wahr, sodass am Ende nicht in den loop Teil übergegangen wird
  } else {
    // Andernfalls werden die Namen der erstellten Log-Dateien
    ↪ ausgegeben
    Serial.print("Logging to: ");
    Serial.print(filename);
    Serial.print(" and ");
    Serial.println(hall_trigger_filename);
  }

  // Falls keine Daten vom BME abgefragt werden können, wird ein
  ↪ Fehler ausgegeben
  if (!bme280.init()) {
    error("Could not read data from BME"); // Ruft die error
    ↪ Methode auf, die sich dann um den Fehler kümmert
    errorOccuredInSetup = true; // Setzt errorOccuredInSetup auf
    ↪ wahr, sodass am Ende nicht in den loop Teil übergegangen wird
  }

  // Setzt die Spaltennamen für die CSV-Log-Dateien

  ↪ logfile.println("timestamp,temperature,pressure,humidity,altitude,uv");
  hall_trigger_logfile.println("timestamp");
  // Gibt (falls ECHO_TO_SERIAL wahr ist) eine Warnung aus auf
  ↪ dem seriellen Monitor aus und setzt die Spaltennamen für
  ↪ die Ausgabe
  #if ECHO_TO_SERIAL
    Serial.println("Note: The timestamp starts with the execution
    ↪ of the programm and will overflow after approx. 50 days.");
  ↪ Serial.println("timestamp,temperature,pressure,humidity,altitude,uv");
  #endif //ECHO_TO_SERIAL

  delay(5000); // Wartet kurz

  // Falls noch keine Fehler aufgetreten ist, wird gewartet,
  ↪ bis der Start-Knopf gedrückt wird
  if (errorOccuredInSetup == false) {
    while (analogRead(onOffButton) < 1023) {
```

```
        delay(500);
    }
} else {
    // Andernfalls wird ein Fehler ausgegeben und in einen ewigen
    ↪ Loop übergegangen
    while (1==1) {
        Serial.println("An error occured, see above");
        delay(900000);
    }
}

// Schaltet die grüne LED an
digitalWrite(greenLED, HIGH);

delay(3000); // Wartet kurz

// Die gelbe LED wird wieder ausgeschaltet
digitalWrite(yellowLED, LOW);

// Setzt _running auf wahr
_running = true;

// Die Startzeit wird gespeichert
runningSince = millis();
}
```

Loop-Teil

Im Loop-Teil angekommen, folgt der Ablauf dem Programmablaufplan (siehe nächster Punkt). Zunächst wird überprüft, ob der An-/Ausknopf erneut gedrückt wurde und die Messung gegebenenfalls beendet, was wiederum über die Status LEDs kommuniziert wird. Ist dies nicht der Fall und `_running` ist wahr, so wird zuerst falls nicht 0 die Zeit des letzten Auslösen des Hall-Sensors in die dafür vorgesehene Log-Datei geschrieben. Danach werden, sofern nicht ausschließlich der Hall Sensor gemessen werden soll, die Daten der restlichen Sensoren erfasst und in die Log-Datei geschrieben bzw. zusätzlich auch ausgegeben. Zum Schluss wird falls die letzte Synchronisierung auf die SD-Karte bereits eine bestimmte Zeit her ist, diese wieder synchronisiert.

```
void loop(void)
{
    // Falls der An-/Aus-Knopf gedrückt wird, wird der Messvorgang
    ↪ beendet
    if (analogRead(onOffButton) >= 1023) {
        _running = false; // _running wird auf false gesetzt
    }
}
```

```
// Synchronisiert die gesammelten Daten, während die gelbe
↳ LED einmal blinkt
digitalWrite(yellowLED, HIGH);
logfile.flush();
hall_trigger_logfile.flush();
digitalWrite(yellowLED, LOW);

digitalWrite(greenLED, LOW); // Schaltet die grüne LED aus
digitalWrite(yellowLED, HIGH); // Schaltet die gelbe LED an
digitalWrite(redLED, HIGH); // Schaltet die rote LED an
}

// Falls _running wahr ist, wird gemessen
if (_running == true) {

    // Falls lastLallSesnorTriggerTime nicht 0 ist, wird
    ↳ lastLallSesnorTriggerTime in die Log-Datei geschrieben
    if (lastLallSesnorTriggerTime != 0) {

        // Gibt (falls ECHO_TO_SERIAL wahr ist) "Hall sensor
        ↳ triggered!" auf dem seriellen Monitor aus
        #if ECHO_TO_SERIAL
            Serial.println("Hall sensor triggered!");
        #endif //ECHO_TO_SERIAL

        // Schreibt lastLallSesnorTriggerTime in die Log-Datei
        hall_trigger_logfile.println(lastLallSesnorTriggerTime);

        // Setzt lastLallSesnorTriggerTime wieder auf 0
        lastLallSesnorTriggerTime = 0;
    }

    // Falls nicht nur der Hall Sensor gemessen werden soll,
    ↳ werden auch die Werte der anderen Sensoren gemessen
    if (!onlyMeasureHallSesnor) {

        // Setzt _timestamp auf die aktuelle Zeit minus der
        ↳ Startzeit
        unsigned long _timestamp = millis() - runningSince;

        // Schreibt _timestamp in die Log-Datei
        logfile.print(_timestamp);
        // Gibt (falls ECHO_TO_SERIAL wahr ist) _timestamp am
        ↳ seriellen Monitor aus
        #if ECHO_TO_SERIAL
            Serial.print(_timestamp);
        #endif
    }
}
```

```
#endif //ECHO_TO_SERIAL

// Liest die Sensorwerte des UV-Sensors aus und rechnet sie
↳ in den UVI um
float analogSignal = analogRead(UVSensor);
float voltage = analogSignal/1023*5;
float uvIndex = voltage / 0.1;

// Schreibt die Sensorwerte in die Log-Datei
logfile.print(", ");
logfile.print(bme280.getTemperature());
logfile.print(", ");
logfile.print(bme280.getPressure());
logfile.print(", ");
logfile.print(bme280.getHumidity());
logfile.print(", ");
logfile.print(bme280.calcAltitude(bme280.getPressure()));
logfile.print(", ");
logfile.print(uvIndex);
// Gibt (falls ECHO_TO_SERIAL wahr ist) die Sensorwerte am
↳ seriellen Monitor aus
#if ECHO_TO_SERIAL
    Serial.print(", ");
    Serial.print(bme280.getTemperature());
    Serial.print(" °C");
    Serial.print(", ");
    Serial.print(bme280.getPressure());
    Serial.print(" Pascal");
    Serial.print(", ");
    Serial.print(bme280.getHumidity());
    Serial.print(", ");
    Serial.print(bme280.calcAltitude(bme280.getPressure()));
    Serial.print(" m");
    Serial.print(", ");
    Serial.print(uvIndex);
#endif //ECHO_TO_SERIAL

// Beginnt eine neue Zeile in der Log-Datei
logfile.println();
// Beginnt (falls ECHO_TO_SERIAL wahr ist) eine neue Zeile
↳ im Seriellen Monitor
#if ECHO_TO_SERIAL
    Serial.println();
#endif // ECHO_TO_SERIAL
}
```

```

// Falls die letzte Synchronisierung länger als SYNC_INTERVAL
↪ Millisekunden her ist, wird die SD-Karte synchronisiert
if ((millis() - syncTime) > SYNC_INTERVAL) {
    // Setzt syncTime auf die aktuelle Zeit
    syncTime = millis();

    // Die gesammelten Daten werden synchronisiert, während die
    ↪ gelbe LED einmal blinkt
    digitalWrite(yellowLED, HIGH);
    logfile.flush();
    hall_trigger_logfile.flush();
    digitalWrite(yellowLED, LOW);
}
}
}

```

Erklärung der Status LEDs

grüne LED	gelbe LED	rote LED	Erklärung
aus	an	aus	Programm befindet sich noch im Setup-Teil
aus	aus	an	Programm schlug im Setup-Teil fehl (z.B. volle SD-Karte)
an	blinkt	aus	Programm läuft fehlerfrei und schreibt die Daten auf die SD-Karte (Blinken bedeutet Synchronisierung)
aus	an	an	Programm wurde erfolgreich beendet

Programm Ablauf Plan (PAP)

