

## IT-Technik ITT11

# Einen Sensor-Messkopf konfigurieren



## Aufgabe

Die SmartDevices GmbH soll nach Kundenwunsch Sensor-Messköpfe entwickeln. Diese sollen die erfassten Temperatur-Werte über WLAN mit MQTT bereitstellen. Ihre Aufgabe ist es, mit einer grafischen Entwicklungsumgebung den Steuerungsrechner der Messköpfe zu programmieren und die Temperaturmessung und Datenaufbereitung zu konfigurieren.

## Inhalt

1. Ausgangslage und Sollzustand .....	2
2. Den NodeMCU-SBC kennenlernen.....	3
3. Die Arduino-IDE verwenden .....	5
4. Messkopf in Betrieb nehmen und Sensordaten abfragen .....	7
5. Sensordaten im WLAN mit MQTT bereitstellen .....	11
Anhang .....	15
Schnittstellen und Busse bei Steuerungsrechnern.....	15
6. Fragen.....	18

## Hinweise

- Beachten Sie die **Sicherheitshinweise** mit den Verhaltensregeln bei Netzspannung und Elektrostatischer Entladung (ESD)
- Hier finden Sie Informationen zur Netzwerkkonfiguration im Unterrichtsraum: [hier](#), auf S.17
- Diese und alle weiteren Unterrichts-Unterlagen und die [Folien](#) finden Sie hier:  
<https://webdav-ad-muenchen.musin.de/intk/austausch/lehrer/joachim.wolf/Unterricht/>

### 1. Ausgangslage und Sollzustand

Die SmartDevices GmbH soll nach Kundenwunsch Sensor-Messköpfe entwickeln. Diese sollen die erfassten Temperatur-Werte über WLAN mit MQTT bereitstellen.

Die Messköpfe werden in Serverracks eingesetzt und sollen die Daten von mehreren Temperatursensoren bereitstellen und optional noch weitere Informationen abfragen können, z.B. den Zustand eines Türkontakts.

Sie als Mitarbeiter\*in der SmartDevices GmbH werden zusammen mit Ihrem Team diesen Auftrag übernehmen. Ihre Aufgabe ist es, mit Hilfe der grafischen Entwicklungsumgebung Arduino-IDE den NodeMCU-Steuerungsrechner zu programmieren und die Temperaturmessung und Datenaufbereitung zu konfigurieren.

## Arbeitsschritte

- Den NodeMCU-SBC kennenlernen (S. 3)
- Die Arduino-IDE verwenden (S. 5)
- Messkopf in Betrieb nehmen und Sensordaten lokal abfragen (S. 7)
- Sensordaten im WLAN mit MQTT bereitstellen (S. 11)

## 2. Den NodeMCU-SBC kennenlernen

**Aufgabe:** Beantworten Sie auch mit Hilfe des [Wikipedia-Artikels zu NodeMCU](#) die nachfolgenden Fragen.

**A**

Einigen Sie sich in Ihrem Team jeweils auf eine Antwort. Laden Sie Ihre Antworten zusammen mit den Fragen auf die von der Lehrkraft genannte Dateiablage hoch. Der Senior Analyst der SmartDevices GmbH (Lehrkraft) bespricht anschließend die Antworten mit Ihnen.

1.) Was ist ein [SBC](#)?

Single Board Computer, alle benötigten Teile sind auf einer Platine verbaut

2.) Was ist [NodeMCU](#)?

Ein freies Betriebssystem, außerdem wird das Evaluierungsboard so genannt

3.) Unser NodeMCU-SBC verwendet den 32-Bit-Mikrocontroller [ESP8266](#).  
Was ist ein [Mikrocontroller](#)?

4.) Nennen Sie einige Eigenschaften des [ESP8266](#).

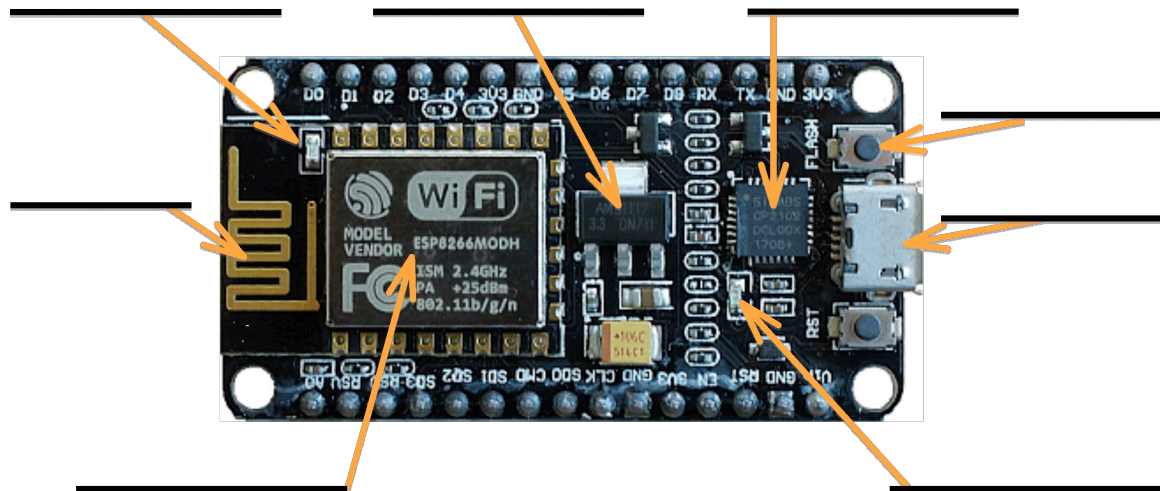
5.) Erklären Sie die Begriffe [UART](#), [I2C](#), [GPIO](#).

6.) In der folgenden Tabelle finden Sie die Technische Daten des NodeMCU-SBC im Vergleich. Ergänzen Sie die fehlenden Angaben

Komponente	NodeMCU-SBC	Arduino Uno	Raspberry Pi 4B	BS FiSi PC
Prozessor (SoC)	ESP8266	ATmega328P	BCM2711	Intel i5-12600
Takt	_____ MHz	16 MHz	1500 MHz	3,3 GHz
Architektur	_____ Bit	8 Bit	64 Bit	64 Bit
Ein-/Ausgänge	_____	14	26 (GPIO)	keine GPIO
A/D-Wandler	_____ x	6x	---	---
Speicher	_____ kB RAM _____ MB Flash	2 kB RAM 32 kB Flash	1-8 GB RAM microSD > 32 GB	16 GB RAM
WLAN-Standards	_____	---	802.11 b/g/n/a/ac	802.11 n/a/ac/ax
Stromversorgung	3,3/5V, 5µA - 0,4A	5V, 60mA	5V, 3 - 8W	230V, 50W

**7.) Ordnen Sie die angegebenen Bestandteile des NodeMCU-Boards zu!**

(WLAN-Antenne, Micro-USB, Leuchtdiode (LED) an D0, Taster an D3, USB-Seriell-Wandler CP210x oder CH340, Spannungsregler 3,3 V, Upload-LED, ESP-12E-Modul)



### 3. Die Arduino-IDE verwenden

#### 1.) Installation der Arduino-IDE

**A**

##### Arbeitsschritte

- Arduino-IDE herunterladen (alte Version 1.8.xx)
- Internetzugang konfigurieren
- Boardverwaltung für NodeMCU vorbereiten
- COM-Schnittstelle des NodeMCU-Boards ermitteln
- **Hinweis:** Arbeiten Sie genau!
- laden Sie auf einem Laborrechner von [www.arduino.cc/en/software](http://www.arduino.cc/en/software) das Zip-File der letzten **alten** Arduino-IDE herunter (**1.8.19**), entpacken Sie das Zip-File nach C:\ und starten Sie im neu angelegten Verzeichnis (z.B. C:\arduino-1.8.19) das Programm **arduino.exe**.
- um der Arduino-IDE den Umgang mit der NodeMCU beizubringen, müssen die benötigten Compiler, Bibliotheken und Einstellungen über das Internet nachgeladen werden:
  - wird der Internetzugang über einen Proxy bereitgestellt, muss in der Arduino-IDE über *Datei --> Voreinstellungen --> Netzwerk* eine manuelle Proxy-Konfiguration vorgenommen werden (im Intranet: Hostname: **proxy** Portnummer: **80**)
  - um die Arduino-IDE für die NodeMCU vorzubereiten, rufen Sie dazu im Menü *Datei* den Menüpunkt *Voreinstellungen* auf und geben dort im Eingabefeld "*Zusätzliche Boardverwalter URLs*" folgende URL ein:  
**`http://arduino.esp8266.com/stable/package_esp8266com_index.json`**
  - wählen Sie dann im Menü *Werkzeuge* den Menüpunkt *Board:...* und danach *Boardverwalter...* aus. **Hinweis:** Zeigt der Boardverwalter einen Fehler, müssen Sie die Boardverwalter-URL und die Proxy-Einstellungen überprüfen
  - klicken Sie im Boardverwalter auf den Eintrag mit *esp8266* und drücken Sie den nun neu erschienenen *Installieren*-Button. Die Installation lädt mehr als 100 MB aus dem Internet und dauert daher etwas. Nun kann die Arduino-IDE mit dem NodeMCU-Board umgehen.
- öffnen Sie den Gerätemanager (z.B. über das Kommando **devmgmt.msc**) und erweitern Sie in der Geräteübersicht, falls vorhanden, den Unterpunkt "*Anschlüsse (COM & LPT)*"
- verbinden Sie jetzt das Node-MCU-Board vorsichtig mit einer USB-Schnittstelle des PCs und überprüfen Sie, ob im Gerätemanager unter "*Anschlüsse (COM & LPT)*" eine weitere COM-Schnittstelle (serielle Schnittstelle) erscheint, das ist dann der USB-Seriell-Wandler Ihrer NodeMCU (meist: *Silicon Labs CP210x bzw. USB SERIAL CH340*)
- **falls beim Anstecken des Node-MCU-Boards keine neue COM-Schnittstelle erscheint**, müssen die USB-Treiber für die USB-Seriell-Wandler CP210x bzw. CH340 geladen werden. Laden Sie diese Treiber von <http://intranet/files/sbc/> herunter, entpacken Sie das Archiv und führen Sie die 64Bit-Version des Installers aus. Es genügt, wenn eine der beiden Komponenten einen grünen Haken hat.
- geben Sie jetzt die COM-Schnittstelle Ihres NodeMCU-Boards an: **COM\_3**
- stellen Sie im Menü *Werkzeuge --> Board:...* --> "*ESP8266 Boards*" das Board "*NodeMCU 1.0 (ESP-12E Modul)*" ein und wählen Sie unter *Werkzeuge --> Port* den COM-Port Ihres NodeMCU-Boards aus

## 2.) NodeMCU-SBC in Betrieb nehmen

Wenn Sie die Arduino-IDE starten, wird automatisch ein neues Programm (Arduino-„Sketch“) angelegt, das bereits über zwei leere Methoden: `setup()` und `loop()` verfügt.

Die Methode `setup()` wird **genau einmal** beim Start des Boards aufgerufen. Hier werden die Programm-Anweisungen (Befehle) zur Initialisierung des Systems untergebracht.

Die Methode `loop()` wird **dauernd** aufgerufen. Sobald sie einmal durchgelaufen ist, erfolgt der nächste Aufruf ([Endlosschleife](#)). Hier können alle Befehle untergebracht werden, die solange wiederholt werden sollen, wie das Board mit Spannung versorgt ist.


Neben dem USB-Chip befindet sich eine kleine blaue LED, diese kann über den Anschluss-Pin D0 gesteuert werden. Da alle D-Pins als digitaler Eingang oder Ausgang arbeiten können, muss der NodeMCU zunächst mitgeteilt werden, ob an Pin D0 Daten eingelesen oder ausgegeben werden sollen. Um den Pin D0 als Ausgang zu setzen, müssen Sie den folgenden den Befehl zwischen die geschweiften Klammern der Methode `setup()` schreiben:

```
pinMode(D0, OUTPUT);
```

**Hinweis:** In der Arduino-IDE wird die sog. [Arduino Programming Language](#) verwendet, dabei handelt es sich um eine reduzierte Form von C++. Da die Syntax sehr ähnlich zu Java ist, muss auf den Strichpunkt am Ende jeder Programm-Anweisung geachtet werden!

Um die LED an Pin D0 einzuschalten, verwenden Sie den folgenden Befehl:

```
digitalWrite(D0, LOW);
```

Mit dem Knopf  können Sie das Compilieren und das Hochladen auf das Board auslösen. Nach dem Hochladen sollte die blaue LED dauerhaft leuchten.

## 3.) Ihr erstes eigenes Arduino-Programm

Als nächstes soll ein Auto-Blinker nachgebaut werden. Dieser hat nach [§54 StVZO](#) eine Frequenz von 1,5 Hz, d.h er blinkt 1,5 mal in der Sekunde, also 90 mal pro Minute.

Bei jeder Blink-Phase ist der Blinker genauso lang ein- wie ausgeschaltet.

Zur Umsetzung sind zwei weitere Befehle nötig:

```
digitalWrite(D0, HIGH); // schaltet die LED wieder aus
```

```
delay(1000); // wartet 1000 ms (= 1 s) vor der Ausführung des nächsten Befehls
```

Wie müssen jetzt die Befehle in der `loop()`-Methode kombiniert werden, damit die LED wie ein Autoblinder blinkt? Welcher Zahlenwert muss für die Verzögerung verwendet werden?

Tragen Sie das fertige und getestete Programm hier ein:

```
loop() {  
    digitalWrite(D0, LOW);  
    delay(333);  
    digitalWrite(D0, HIGH);  
    delay(333);  
}
```

## 4. Messkopf in Betrieb nehmen und Sensordaten abfragen

### Temperatur mit der NodeMCU messen

Zunächst soll die Funktion des vorhandenen Temperatursensors geprüft werden und der aktuelle Messwert an den Computer übertragen werden.

### Serielle Schnittstelle verwenden

Programme auf der NodeMCU können über die COM-Schnittstelle des Boards Daten an den PC senden. Dazu muss die serielle Schnittstelle vor ihrer Verwendung in *setup()* mit dem Befehl *Serial.begin(speed)*; initialisiert werden. Dann können z.B. mit *Serial.println("Hallo")*; ausgegeben werden.

Um die Daten, die das NodeMCU-Board über die seriellen Schnittstelle an den PC sendet, zu empfangen muss in der Arduino-IDE über *Werkzeuge* der "*Serielle Monitor*" geöffnet werden. Der oben verwendete Wert von *speed* muss dabei mit dem im Seriellen Monitor eingestellten Wert übereinstimmen. Häufig liegt *speed* bei 9600 Bit/s oder 115200 Bit/s. Die seriellen Ausgaben können sehr gut bei der Fehlersuche (debugging) helfen.

### Temperatursensor DS18B20

Der zur Verfügung gestellte Temperatursensor verwendet den Baustein [DS18B20](#) und wird über einen 1-Wire-Bus an die NodeMCU angeschlossen.

Physikalisch ist die Signalleitung des Sensors an der NodeMCU mit dem Anschluss-Pin D1 verbunden. Dazu kommen die Versorgungsspannung mit 3,3 V (Pin 3V3) und der Minuspol GND (Pin GND), siehe Bild auf S. 15.

Damit sowohl der Sensor als auch die NodeMCU Daten senden können, liegt die Signalleitung des 1-Wire-Bus im Ruhezustand über einen *Pullup*-Widerstand an der positiven Versorgungsspannung. Wenn ein Bit übertragen werden soll, wird dies durch eine Absenkung des Spannungspegels der Signalleitung auf 0 V für 50 µs eingeleitet. Folgt dann für 25 µs ein hoher Spannungspegel, wird der Bitwert 1 übertragen, dauert der hohe Pegel 70 µs, wird der Bitwert 0 übertragen.

### Programm-Bibliotheken

Um auf der NodeMCU den DS18B20-Temperatursensor abzufragen, müssen umfangreiche Einstellungen und Konfigurationen programmiert werden. Die benötigten Programm-Codes liegen bei populärer Hardware oft bereits in Form einer oder mehrerer sogenannter *Bibliotheken* vor. Programm-Bibliotheken sind also eine Sammlung von Unterprogrammen und vereinfachen den Umgang mit der Hardware erheblich.

Um die Programmierunterstützung für den DS18B20-Temperatursensor zu erhalten, müssen die Bibliotheken *OneWire* und *DallasTemperature* installiert werden. Öffnen Sie dazu im Menu *Sketch* --> "*Bibliothek einbinden*" --> "*Bibliotheken verwalten...*" den *Bibliotheksverwalter* und installieren Sie jeweils die neueste Version dieser Bibliotheken.

**A**



## Das Mess-Programm kennenlernen

```
// Mit einem DS18B20 Temperatursensor die Temperatur messen
// modifiziert, nach: https://makesmart.net/ds18b20-temperaturfuhrer-esp8266-d1-mini/

#include <OneWire.h> (a)
#include <DallasTemperature.h> (a)

#define ONE_WIRE_BUS 5 // Der PIN D1 (GPIO 5) wird als BUS-Pin verwendet

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature DS18B20(&oneWire);

float temperature; // In dieser Variable wird die Temperatur gespeichert

void setup(){
  Serial.begin(115200); // Serielle Schnittstelle initialisieren
  DS18B20.begin(); // DS18B20 initialisieren
}

void loop(){
  DS18B20.requestTemperatures();
  temperature = DS18B20.getTempCByIndex(0); // 0 --> erster Sensor am Bus

  Serial.println(String(temperature) + " °C"); // Ausgabe im seriellen
                                              // Monitor

  delay(3000); // 3 Sekunden warten
}
```

**Aufgabe:** Beantworten Sie folgende Fragen!

A

- 1.) Betrachten Sie den obigen Programmcode und markieren Sie mit den jeweiligen Buchstaben der Teilaufgaben, die Stellen,

- a) **Beispiel:** an denen die benötigten Bibliotheken eingebunden werden,
- b) wo ein Wert vom Temperatursensor gelesen wird und
- c) wo über die serielle Schnittstelle die Temperatur-Daten gesendet werden.

- 2.) Geben Sie den Datentyp von *temperature* an. **Float**

Was machen folgende Funktionen?

**Serial.println()**

Sendet Daten an eine serielle Schnittstelle

**String()**

Stellt sicher, dass ein Datentyp zu einem String konvertiert wird

Schauen Sie dazu in der Arduino-**Sprach-Referenz** nach!

- 3.) Erklären Sie das fett Hervorgehobene von **Serial.println( String(temperature) + " °C" );**

Hier wird die (vorher als float gespeicherte) Temperatur zu einem String konvertiert und mit dem String " °C" konkateniert (zusammengefügt)

- 4.) Was bedeutet *Initialisierung*?

Einrichtung von Anfangswerten etc.



**5.) Was bedeutet *Compilieren*?**

Umwandeln von geschriebenem Code zu Maschinen-Code (und Prüfung dessen)

**6.) Was ist eine Programm-Bibliothek?**

Eine Sammlung von Code um Funktionalitäten bereitzustellen

**7.) Was wird bei der Programmierung unter *Debugging* verstanden?**

Schritt-für-schritt durch den Code gehen um Fehler zu finden

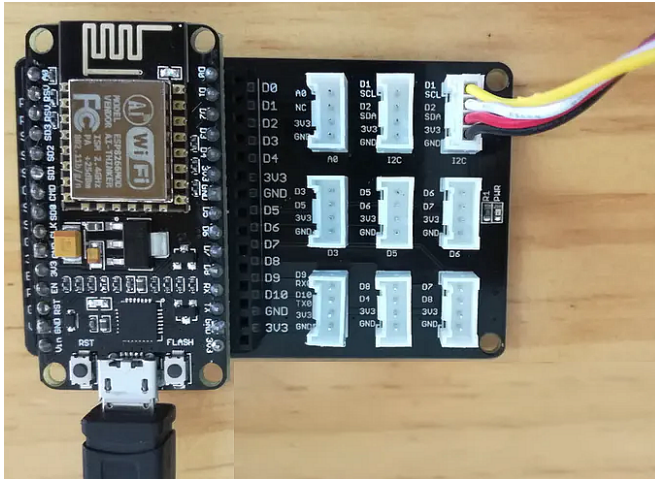
**8.) \*Ein Bit auf dem 1-Wire-Bus wird (wie oben beschrieben) mit dem Zyklus 50/25  $\mu$ s für eine '1' bzw. 50/70  $\mu$ s für eine '0' übertragen.  
Wieviele Bit können so im Idealfall pro Sekunde übertragen werden, vorausgesetzt, es werden genauso viele '0' wie '1' übertragen (z.B. 1010101010101010...) ?**

## Das Messprogramm umsetzen

A

Erstellen Sie in der Arduino-IDE über *Datei --> Neu* einen neuen Sketch. Kopieren Sie mit *Copy-and-Paste* (bsinfo.eu->Service->WebDAV) den oben dargestellten Programmcode aus dieser Unterlage heraus oder schreiben Sie ihn ab. Mit Strg-T können Sie den Code in der Arduino-IDE korrekt einrücken.

- Verbinden Sie den Temperatursensor mit dem Träger-Board der NodeMCU **genau so**, wie es auf dem folgenden Bild dargestellt ist:



Bei uns ist die NodeMCU bereits auf das *Grove Base Shield for NodeMCU* gesteckt!

- Öffnen Sie über *Werkzeuge* den "Seriellen Monitor" mit 115200 Baud.
- Laden Sie den Sketch auf das NodeMCU-Board hoch.
- Testen Sie nun im Seriellen Monitor, ob Sie dort die gesendeten Temperatur-Werte sehen können.

## Temperatur-Schwellenwert signalisieren

A

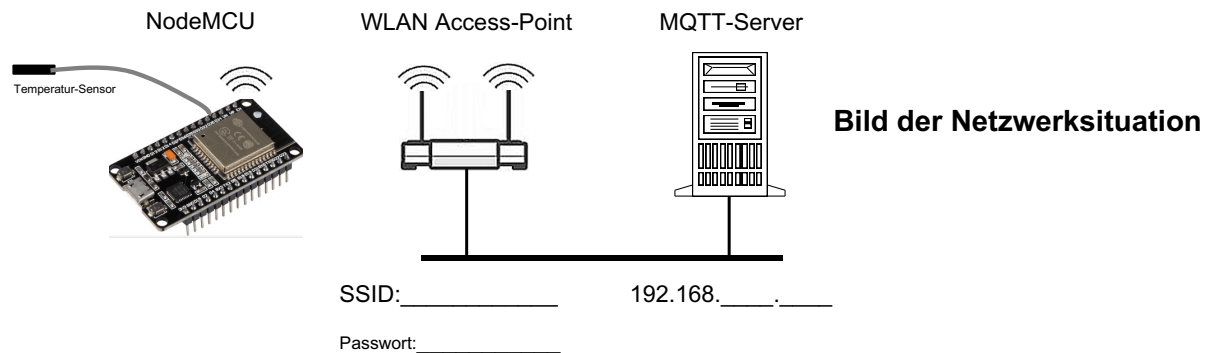
Die blaue LED an Port D0 soll als Signal für eine Klimaanlage dienen. Dazu soll die LED beim Überschreiten eines bestimmten Wertes (Schwellenwert), z.B. 27 °C, eingeschaltet werden und so signalisieren, dass jetzt gekühlt werden muss. Wenn die Temperatur unter den Schwellenwert fällt, soll die LED wieder ausgeschaltet werden.

- Ändern Sie den Sketch nun entsprechend ab und speichern Sie ihn unter dem Namen *Schwellenwert* ab.
- Testen Sie die Schwellenwert-Detektion, indem Sie den Sensor zwischen den Fingern erwärmen und anschließend wieder abkühlen lassen.

## 5. Sensordaten im WLAN mit MQTT bereitstellen

A

Jetzt soll der gemessene Temperaturwert zu Ihrem MQTT-Server übertragen werden.



### MQTT-Server bereitstellen

Zur Umsetzung benötigen Sie einen eigenen MQTT-Server. Die Bereitstellung eines MQTT-Servers haben Sie bereits auf einem RasPi [durchgeführt](#).

Sie können auch die selbstextrahierende VM mit einem laufenden MQTT-Server verwenden: <http://intranet/files/VMs/Windows10-NodeRED-MQTT.vm12.exe>

**Hinweis:** Die VM startet einmal neu. Bitte warten Sie dies ab!

Um Ihren MQTT-Server über WLAN anzusprechen, werden folgende Informationen benötigt:

WLAN-SSID <i>(bitte bei der Lehrkraft erfragen!)</i>	
WLAN-Kennwort <i>(bitte bei der Lehrkraft erfragen!)</i>	
IP-Adresse Ihres MQTT-Servers	192.168.____.____
Topic	<b>raum17/rack05/temperatur</b>

Kopieren Sie den Beispiel-Code von der folgenden Seite in die Arduino-IDE und passen Sie ihn nach den Vorgaben an. Speichern Sie ihn dann unter dem Namen `mqtt_nodemcu` ab.



Erweitern Sie den String `clientId` um das letzte Oktett der VM-IP.

### MQTT-Bibliothek installieren

Um über das MQTT-Protokoll zu kommunizieren, müssen Sie mit dem Arduino Bibliotheks-verwalter die Bibliothek `PubSubClient` installieren.

### Temperaturübertragung testen

Laden Sie das Programm hoch und überprüfen Sie im Seriellen Monitor (115200 Baud), ob eine Verbindung zum WLAN aufgebaut wird und die MQTT-Nachrichten ausgehen.

Prüfen Sie dies auch direkt auf dem MQTT-Server durch Abonnieren des Topics:

```
mosquitto_sub -h 127.0.0.1 -t "raum17/rack05/temperatur"
```

**Hinweis:** Eine portable Arduino-IDE, die bereits alle nötigen Bibliotheken enthält, finden Sie unter: <http://intranet/files/ArduinoIDE-Portable.exe>

## Ein einfaches WLAN/MQTT-Programm

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

#include <OneWire.h>
#include <DallasTemperature.h>

#define ONE_WIRE_BUS 5
#define MSG_BUFFER_SIZE 50

char msg[MSG_BUFFER_SIZE];
float temperature;
const char* ssid = ".....";
const char* password = ".....";
const char* mqtt_server = "192.168.....";
const char* topic = ".....";

WiFiClient espClient;
PubSubClient client(espClient);

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature DS18B20(&oneWire);

void setup() {
    DS18B20.begin();
    Serial.begin(115200);
    delay(100);
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
    client.setServer(mqtt_server, 1883);
    Serial.print("Attempting MQTT connection...");
    String clientId = "ESP8266Client";
    client.connect(clientId.c_str());
}

void loop() {
    DS18B20.requestTemperatures();
    temperature = DS18B20.getTempCByIndex(0);
    snprintf(msg, MSG_BUFFER_SIZE, "%3.2f", temperature);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish(topic, msg);
    delay(500);
}
```

**Aufgabe:** Beantworten Sie folgende Fragen zum vorstehenden Programm-Code!

- 1.) Betrachten Sie den vorherigen Programmcode und markieren Sie mit den jeweiligen Buchstaben der Teilaufgaben die Stelle,
  - a) an der die für MQTT benötigte Bibliothek eingebunden wird,
  - b) wo der Temperatursensor initialisiert wird,
  - c) an der die Verbindung zum MQTT-Server aufgebaut wird,
  - d) die WLAN-Verbindung initialisiert wird,
  - e) wo der Wert einer *float*-Variable auf 2 Nachkommastellen gewandelt in eine *string*-Variable kopiert wird, und
  - f) wo Daten an den MQTT-Server gesendet werden.

- 2.) Bei den Code-Zeilen, die mit `#include` und `#define` beginnen, handelt es sich um Anweisungen für den *C Precompiler*, z.B.: `#define MSG_BUFFER_SIZE 50`

Mit `#include` werden die benötigten Bibliotheken eingebunden.

Das dargestellte `#define` führt dazu, dass der *C Precompiler* vor dem eigentlichen Compiliervorgang im Quelltext alle Vorkommen von `MSG_BUFFER_SIZE` durch 50 ersetzt.

Im Beispiel führt das dazu, dass aus `char msg[MSG_BUFFER_SIZE];` folgendes wird: `char msg[50];`

Wo gibt es im vorstehenden Programm-Code noch ein weiteres `define`?

- 3.) Um welchen Datentyp handelt es sich hier? `char msg[MSG_BUFFER_SIZE];`

- 4.) Wann wird die nachfolgend dargestellte *while*-Schleife abgebrochen?

```
while (WiFi.status() != WL_CONNECTED) {  
    delay(500);  
    Serial.print(".");  
}
```

- 5.) Welches Schlüsselwort bei der Deklaration der Variablen *topic* verhindert, dass diese während des Programmlaufs geändert werden kann?

- 6.) Laut einem [Arduino Cheat-Sheet](#) deckt der Datentyp *int* die Zahlen von -32768 bis 32767 ab. Wie viele Zahlen sind das insgesamt und wie viele Bits werden für diesen Zahlenbereich benötigt?  
Wie viele Byte belegt demnach eine *int*-Variable im Programm-Speicher?

**Das WLAN/MQTT-Programm weiter anpassen**

Ergänzen Sie das Programm mit den *float*-Variablen *Schwellenwert* und *Alarmwert*.

Unterhalb des Schwellenwerts (z.B. 27 °C) soll die LED an Port D0 mit einer Frequenz von 0,5 Hz blinken (Einschaltzeit 1000 ms, Ausschaltzeit 1000 ms).

Wenn der Schwellenwert überschritten wird, soll die LED mit einer Frequenz von 2 Hz aufblitzen (Einschaltzeit 100 ms, Ausschaltzeit 400 ms).

Wird der Alarmwert überschritten (z.B. 29 °C), soll die LED mit einer Frequenz von 5 Hz aufblitzen (Einschaltzeit 30 ms, Ausschaltzeit 120 ms).

Hinweis: Sie könnten dazu *if/else if/else* verwenden ...

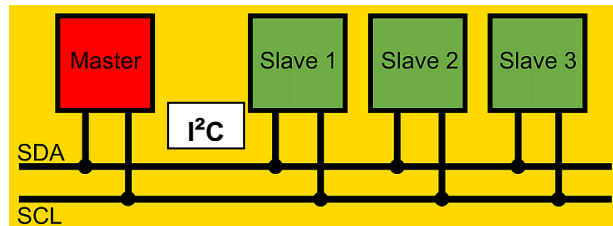
## Anhang

### Schnittstellen und Busse bei Steuerungsrechnern

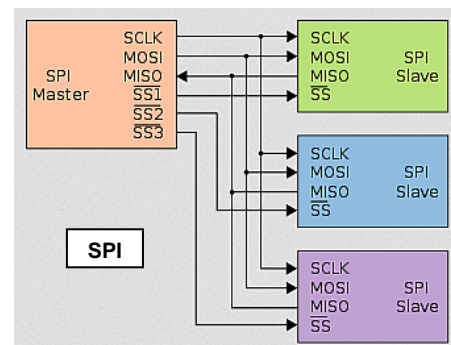
Steuerungssysteme verfügen über eine große Zahl an Ein- und Ausgängen (Schnittstelle, Interface) um beispielsweise Sensoren oder Aktoren anzuschließen. Oft kann an eine Schnittstelle genau ein Gerät angeschlossen werden, z.B: serielle Schnittstelle ([RS-232](#)). Schnittstellen, die den gleichzeitigen Anschluss mehrerer Geräte erlauben, heißen Bus, z.B: [Ethernet](#), [USB](#), [RS-485](#), [CAN](#), [KNX](#), [I<sup>2</sup>C](#), [SPI](#), [1-Wire](#). Die bei Automatisierungs- und Steuerungssystemen eingesetzten Busse werden auch als [Feldbus](#) bezeichnet. **Feldbusse** können Daten in störanfälligen Umgebungen über größere Distanz zuverlässig übertragen.

#### Nahbereichs-Busse: [I<sup>2</sup>C-Bus](#), [SPI](#) und [1-Wire](#)

Der **I<sup>2</sup>C-Bus** (Inter IC-Bus) ist ein Bus, der die Kommunikation zwischen (*inter*) verschiedenen Integrierten Schaltungen (*ICs*) ermöglicht und wurde von Philips entwickelt. Er wird manchmal auch 2-Draht-Bus genannt, da der Bus mit 2 Daten-Leitungen auskommt, dazu kommen noch Versorgungsspannung (**VDD**, Pluspol) und Masse (**GND**, Minuspol). Die SDA-Leitung (serial data) dient zur seriellen Datenübertragung, mit der SCL-Leitung SCL (serial clock) werden die benötigten Takt-Impulse gesendet. Alle Geräte werden parallel an den Bus angeschlossen und verfügen über eine eindeutige Geräteadresse (7 oder 10 Bit). Maximal sind 128 Busteilnehmer (Geräte) erlaubt, davon muss mindestens ein Gerät ein *Master* sein, alle anderen werden *Slave* genannt. Der Bus-Master steuert alle Aktionen auf dem Bus.

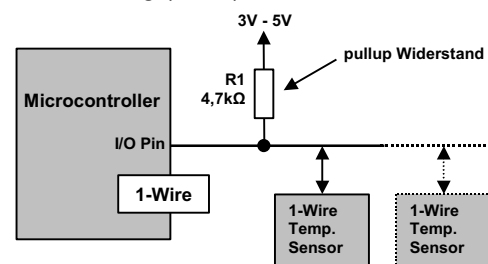


Auch beim **SPI-Bus**-System (Serial Peripheral Interface) werden digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbunden.



**1-Wire** (One-Wire, Eindraht-Bus) beschreibt einen Bus, der mit nur einer Datenleitung auskommt, die sowohl als Stromversorgung für die angeschlossenen Busteilnehmer als auch als Sende- und Empfangsleitung genutzt wird. Zusätzlich ist noch eine Masse-Verbindung (GND) erforderlich. Die Übertragung erfolgt seriell nach dem One-Master/Multi-Slave Prinzip.

Viele der in CPS/IoT-Systemen eingesetzten Sensoren und Aktoren verfügen über einen der 3 genannten Nahbereichs-Busse, beispielsweise verwendet der populäre Temperatursensor [DS18B20](#) den 1-Wire-Bus, der Luftqualitäts-Sensor [CCS811](#) setzt I2C ein, der Umgebungssensor [BME680](#) versteht sowohl SPI als auch I2C. Aus diesen Gründen verfügen Steuerungsrechner über mindestens einen dieser Busse (z.B. Arduino, RasPi, ESP8266).



#### Feldbusse

##### [Ethernet](#)

Ethernet wird zur Kommunikation im Netzwerk zu übergeordneten Steuerungssystemen eingesetzt. Um den Ethernet-Standard auch für die Vernetzung von Geräten in der Automatisierungstechnik zu nutzen, wird [Industrial Ethernet](#) eingesetzt. Industrial Ethernet ist an industrielle Umgebungsbedingungen angepasst und verfügt gegenüber Standard-Ethernet über erhöhte Störsicherheit, verbessertes [Zeitverhalten](#), erweiterter Temperaturbereich und einen besseren Schutz gegen Verschmutzung (z.B. [Profinet](#), [EtherCAT](#)).



**CAN**

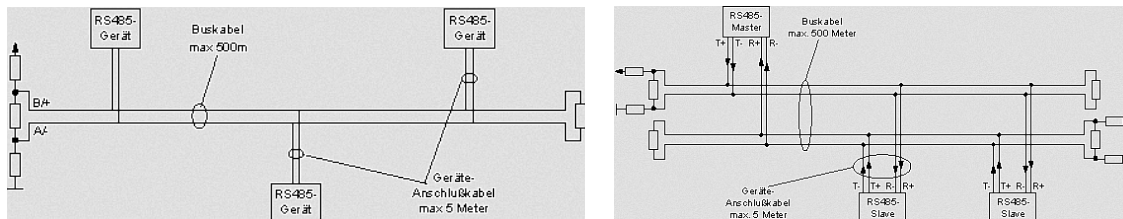
Der CAN-Bus arbeitet nach dem Multi-Master-Prinzip, d.h. jede Komponente kann ohne Vermittlung durch einen Master selbstständig auf den Bus zugreifen. So können Sensoren und Aktoren Daten direkt untereinander austauschen. CAN-Busse werden oft in Bereichen verwendet, bei denen es auf hohe Datensicherheit ankommt. Beispiele: Fahrzeugtechnik, Medizintechnik, Agrartechnik.

**KNX**

KNX wird hauptsächlich in der **Gebäudeautomation** eingesetzt und unterstützt große Entfernungen.

**RS-485**

RS485 wurde für schnelle serielle Datenübertragungen über große Entfernungen entwickelt. Ein RS485-Bus verwendet TP-Leitungen und kann entweder als 2-Draht- oder als 4-Draht-System aufgebaut werden. Der Vorteil der 2-Draht-Technik liegt im Wesentlichen in der Multimaster-Fähigkeit (z.B. **PROFIBUS**). Die 4-Draht-Technik kann nur von Master/Slave-Anwendungen verwendet werden.

**Anforderungen an Busse**

möglichst

- viele Geräte
- störungssicher
- große Ausdehnung
- einfacher Aufbau
- hohe Geschwindigkeit
- geringe Kosten

**Aufgabe:** Beantworten Sie folgende Fragen!

1.) Anschluss technik bei Steuerungsrechnern: Was ist ein Bus?

A

2.) Was ist ein Feldbus?

3.) Bus: Worin unterscheidet sich das Multi-Master-Prinzip vom Master-Slave-Prinzip?

4.) Der KNX-Bus wird hauptsächlich in der Gebäudeautomation eingesetzt. Was wird unter *Gebäudeautomation* verstanden?

- 5.) Die folgende Tabelle zeigt wesentliche Kennwerten der wichtigsten Busse. Ergänzen Sie die fehlenden Angaben (grau hinterlegte Felder).

Weitere Infos finden Sie hier: [https://en.wikipedia.org/wiki/List\\_of\\_network\\_buses](https://en.wikipedia.org/wiki/List_of_network_buses)

	<b>Ethernet</b>	<b>USB-2</b>	<b>CAN</b>	<b>KNX</b>	<b>I<sup>2</sup>C</b>	<b>SPI</b>	<b>1-Wire</b>	<b>RS-485</b>
Anzahl Geräte/Bus	<i>unbegrenzt?</i>	<b>127</b>	<b>32-110</b>	<b>256</b>	<b>112/1136?</b>		<b>100(-500)</b>	<i>mindestens 32</i>
max. Länge (in m)		<b>5</b>	<b>40-500</b>	<b>1000</b>	<i>unspezifiziert &lt;5m</i>	<b>&lt;10m ?</b>	<b>150 (max 300)</b>	<i>bis zu 1200</i>
Anzahl der Leitungen	<b>2x2 (TP) oder 4x2 (TP)</b>	<b>2x2 (TP)</b>	<b>1x2 (TP)</b>		<b>2 (TP)</b>	<b>4</b>		<b>2/4 (TP)</b>
max. Geschwindigkeit	<b>10 GBit/s</b>	<b>480 Mbit/s</b>	<b>125 kBit/s bei 500 m 1 Mbit/s bei 40 m</b>	<b>9,6 kBit/s</b>	<b>100/400 kBit/s bis zu 3,4 MBit/s</b>	<b>bis zu 5 MBit/s</b>	<b>16,3 kBit/s (142 kBit/s)</b>	<b>bis zu 10 Mbit/s bei 12 m</b>
verfügbar bei Arduino?	<b>nein</b>	<b>nein</b>	<b>nein</b>	<b>nein</b>	<b>ja</b>	<b>ja</b>	<b>ja</b>	<b>nein</b>
verfügbar bei NodeMCU?		<b>nein (nicht zur Steuerung)</b>	<b>nein</b>	<b>nein</b>		<b>ja</b>	<b>ja</b>	<b>nein</b>
verfügbar bei RasPI?	<b>ja</b>	<b>ja</b>	<b>nein</b>	<b>nein</b>	<b>ja</b>	<b>ja</b>	<b>ja</b>	<b>nein</b>

## 6. Fragen

- die folgenden Fragen stammen aus dem [Fragenpool](#). Sie zeigen den Umfang und die Intensität der Unterrichtsinhalte und dienen zur Vorbereitung auf Leistungskontrollen und Abschlussprüfung
- *manchmal* haben schwierige Fragen ein Sternchen "\*", schwierigere zwei "\*\*\*"
- es werden KEINE Lösungen bereitgestellt, Ziel ist es, dass SIE die Lösungen selbst erstellen!
- Nachfragen, Anmerkungen, Lob und Kritik können Sie an Ihre Lehrkraft richten

- 1.) Was ist ein SBC?
- 2.) Unser NodeMCU-SBC verwendet den 32-Bit-Mikrocontroller ESP8266.  
Was ist ein Mikrocontroller?
- 3.) Anschlusstechnik bei Steuerungsrechnern: Was ist ein Bus?
- 4.) Was ist ein Feldbus?
- 5.) Nennen Sie 2 wesentliche Anforderungen die Feldbusse erfüllen müssen.
- 6.) Worin unterscheidet sich das bei einigen Bussen verwendete Multi-Master-Prinzip vom Master-Slave-Prinzip?
- 7.) Der KNX-Bus wird hauptsächlich in der Gebäudeautomation eingesetzt.  
Was wird unter *Gebäudeautomation* verstanden?
- 8.) Häufig müssen in der Arduino-IDE Programm-Bibliotheken nachinstalliert werden.  
Was sind *Bibliotheken*?
- 9.) Programmierung: Was bedeutet *Initialisierung*?
- 10.) Programmierung: Was bedeutet *Compilieren*?
- 11.) Programmierung: Was bedeutet *Debugging*?
- 12.) Was ist ein *Schwellenwert*?
- 13.) Erklären Sie den folgenden Befehl genau:  
`mosquitto_sub -h 192.168.1.42 -t "raum17/rack05/temperatur"`
- 14.) Erklären Sie den folgenden Befehl genau:  
`mosquitto_pub -h 192.168.1.42 -t "raum17/rack05/temperatur" -m "23.45"`
- 15.) Laut der [Arduino Sprach-Referenz](#) deckt der Datentyp *unsigned long* die Zahlen von 0 bis 4.294.967.295 ab. Wie viele Byte werden von einer *unsigned long*-Variable im Programm-Speicher benötigt?