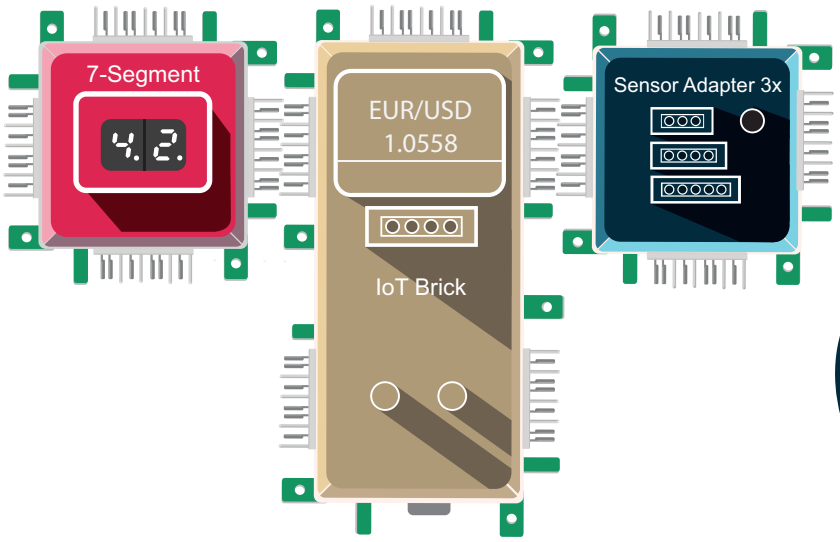
Brick 'R' knowledge

Internet of Things Set

Experimental kit by Brick'R'knowledge
Experimentierkasten von Brick'R'knowledge



Impressum

Brick'R'knowledge Internet of Things Set Anleitung
Rev. 1.0
Datum: 24.03.2017

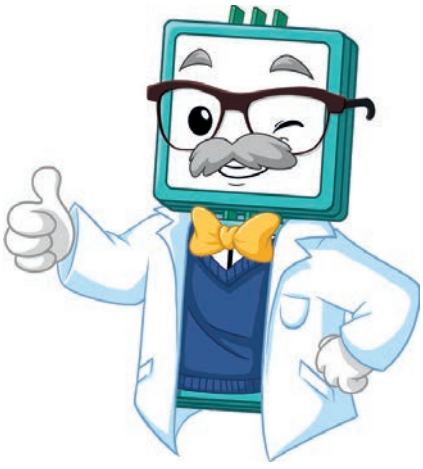
ALLNET® GmbH Computersysteme
Brick'R'knowledge
Maistraße 2
D-82110 Germering

Alle in dieser Anleitung enthaltenen Informationen wurden mit größter Sorgfalt und nach bestem Wissen zusammengestellt. Dennoch sind Fehler nicht ganz auszuschließen. Für die Mitteilung eventueller Fehler sind wir jederzeit dankbar. Bitte sende diese an info@brickrknowledge.de.

Inhaltsverzeichnis

Vorwort

Das Brick'R'knowledge Experimentiersystem wurde zum ersten mal auf der HAM Radio Ausstellung am 28.06.2014 von Rolf-Dieter Klein (Amateurfunkrufzeichen: DM7RDK) vorgestellt. Das Besondere an unseren Elektroniksets ist, dass die einzelnen Bausteine über ein Stecker-System verbunden werden, bei dem die zusammenzufügenden Teile baugleich sind (Hermaphrodite). So können auch knifflige Stromkreise realisiert werden. Auch das Zusammenstecken der einzelnen Bausteine in verschieden Winkeln ist möglich! Für die Rückführung der Masse (0 Volt) sind gleich zwei Kontakte vorhanden! Damit lassen sich kompakte Schaltungen aufbauen, bei der die Masse-Rückführung für eine stabile Spannungsversorgung der Bausteine sorgt. Eine weitere Besonderheit ist, dass man solche Schaltungen sehr leicht erklären und dokumentieren kann.



Viel Spass mit dem Internet of Things Set wünscht

Rolf-Dieter Klein

Quellen- und Literaturverzeichnis:

- Das ESP8266-Praxisbuch, Erik Bartmann, Elektor Verlag GmbH, ISBN 978-3-89576-321-2
- http://www.esp8266.com
- https://en.wikipedia.org/wiki/ESP8266
- Netzwerkbegriffe einfach erklärt: http://www.elektronik-kompendium.de
- Netzwerktechnik-Fibel, Patrick Schnabel, ISBN: 978-3833416811 (3-8334-1681-5)
- HTML-Grundlagen: http://www.selfhtml.org/

Downloads:

- Download Arduino-Entwicklungsumgebung:
  https://www.arduino.cc/en/Main/Software#
- Beispiel-Code und Bibliotheken zu den Übungsbeispielen in dieser Anleitung:
  http://www.brickrknowledge.de/downloads

1. Sicherheitshinweise

Achtung, die Bausteine des Elektroniksets NIE direkt an das Stromnetz (230 V) anschließen, andernfalls besteht Lebensgefahr!

Zur Spannungsversorgung empfehlen wir das mitgelieferte 9 V-Netzteil (oder opt. 9 V-Batterie-Brick). Die Versorgungsspannung beträgt hier gesundheitsungefährliche 9 Volt bei einem Stromfluss von max. 1 Ampere. Bitte trage auch Sorge dafür, dass offen herumliegende Drähte nicht in Berührung oder Kontakt mit Steckdosenleisten (gewöhnliche Zimmerverteiler) kommen bzw. in diese hineinfallen, auch hier besteht andernfalls die Gefahr eines gesundheitsgefährlichen Stromschlags bzw. elektrischen Schocks. Schaue niemals direkt in eine Leuchtdiode (LED), da hier die Gefahr besteht, die Netzhaut zu schädigen (Blendung). Es ist unbedingt darauf zu achten, das mitgelieferte Netzteil nach den Versuchsaufbauten wieder von allen Bausteinen zu trennen, andernfalls besteht die Gefahr eines Elektrobrandes!

Bausteine oder andere Teile des Elektroniksets nicht verschlucken, andernfalls sofort einen Arzt aufsuchen!

2. Was bedeutet eigentlich "Internet der Dinge"?

Wenn man auf Wikipedia nachliest, findet man dort, u. a. folgende, sehr allgemein gehaltene Definition:

"Das Internet der Dinge bezeichnet die Verknüpfung eindeutig identifizierbarer physischer Objekte (things) mit einer virtuellen Repräsentation in einer Internet-ähnlichen Struktur. Es besteht nicht mehr nur aus menschlichen Teilnehmern, sondern auch aus Dingen. Der Begriff geht zurück auf Kevin Ashton, der erstmals 1999 „Internet of Things" verwendet hat."

(Quelle: https://de.wikipedia.org/wiki/Internet_der_Dinge)

Aha, wird sich nun so mancher Leser denken, aber was bedeutet das nun konkret für mich als mehr oder weniger IT-kundiger Zeitgenosse?

Der Begriff "Internet der Dinge" oder englisch "Internet of Things" (abgekürzt: IoT) beschreibt die Vernetzung elektronischer Geräte (sog. Smart Devices) mitunter auch Gadgets (Spielereien) genannt, bis hin zur Vernetzung ganzer Produktionsanlagen (Stichwort: Industrie 4.0). Als Kommunikationsmedium wird in der Regel das standardisierte TCP/IP-Protokoll verwendet, welches sowohl für lokale Netzwerke (LANs) als auch für länderübergreifende Weitbereichsnetzwerke (WANs) eingesetzt wird. Jeder Teilnehmer bzw. jedes Ding benötigt zur Identifizierung eine weltweit eindeutige IP-Adresse. Aufgrund der zunehmenden Adressenknappheit des Internet-Protokolls Version 4 (IPv4) wurde bereits 1998 das Internet-Protokoll Version 6 (IPv6) durch die IETF-Organisation standardisiert. Damit vergrößert sich der Adressraums von IPv4 mit 2^{32} (≈ 4,3 Milliarden) Adressen auf 2^{128} (≈ 340 Sextillionen) Adressen bei IPv6. Um das Prinzip einer Punkt-zu-Punkt-Verbindung im Internet der Dinge zu wahren ist die Vergrößerung des Adressraums unbedingt notwendig, wenngleich es sich bis jetzt nicht wie erhofft durchgesetzt hat.

In Zukunft wird es also nicht mehr nur weltweit vernetzte Rechner geben über welche Menschen miteinander kommunizieren, sondern mehr oder weniger intelligente Geräte mit einer Vielzahl an Sensoren und Aktoren, die Daten direkt miteinander austauschen. Dies führt zu einer noch nie dagewesenen Datenmenge und Datenvielfalt, die quasi in Echtzeit von überall abrufbar ist. Trotz der faszinierenden Möglichkeiten einer vernetzten Welt, muss man sich des potentiellen Sicherheitsrisikos stets bewusst sein.

Einige Beispiele für das Internet der Dinge:

- Microcontroller und Computer mit integriertem Webserver
- Sogenannte Wearables, mit in Kleidungsstücke eingearbeiteten Sensoren
- Gebäudeautomatisierung (Home-Automation)
- Temperaturüberwachung im Serverraum
- Energie-Management, Smart-Metering
- Videoüberwachung

… und nicht zuletzt das Brick'R'knowledge Internet of Things Set mit dem du nun via Internet auf deine Bricks zugreifen kannst.

Zunächst erhältst du einen Überblick über die Grundlagen des Brick'R'knowledge Experimentiersystems. Danach werden alle in diesem Set enthaltenen Bricks und Sensoren vorgestellt. Eine Besonderheit dieses Sets ist die Kombination von Software und Hardware. D. h. wir benötigen eine Software um Programme zu schreiben, welche die Hardware erst zum Leben erweckt. Dazu verwenden wir die Arduino-Entwicklungsumgebung, die kostenlos heruntergeladen werden kann. Nachdem alle Vorbereitungen getroffen sind, steigen wir anhand zahlreicher Übungsbeispiele immer mehr in die Welt des Internet der Dinge ein.

Mit dem zentralen IoT Brick wirst du beispielsweise lernen, deine erste Website zu bauen und I/O-Pins mit deinem Smartphone zu steuern. Außerdem enthält das Set einen Temperatur- und Luftfeuchtigkeits-Sensor, dessen Werte du dir anzeigen lassen kannst. Der erste Schritt zum eigenen Home Automation Projekt! Du kannst aber auch Daten, wie zum Beispiel den Dollar-Kurs aus dem Internet abfragen und anzeigen lassen. Auch ein I^2C-Bus zur Anbindung einer 7-Segmentanzeige oder eines 18 bit A/D-Wandlers ist "on-Brick"!

Das Internet der Dinge wartet darauf, von dir entdeckt zu werden!

3. Grundlagen des Brick'R'knowledge Systems

3.1 Der Masse-Brick

Der Masse-Brick ist ein besonderer Baustein des Brick'R'knowledge Systems. Er spart zusätzliche Verbindungen mit Hilfe anderer Bricks oder Leitungen. Hier wird das Geheimnis unserer vierpoligen Verbinder offenbart. Die mittleren zwei Kontakte sind für die Signalübertragung reserviert, so wie es der Aufdruck verrät. Die äußeren Kontakte werden zum Schließen des Stromkreises, also der Rückführung des Stromflusses zur Spannungsquelle benutzt. Das realisiert der Masse-Brick. Dieser Brick heißt deshalb Masse-Brick, weil in der Elektronik mit der Bezeichnung „Masse“ nicht etwa das Gewicht eines Gegenstandes beschrieben wird, sondern das Bezugspotential, auf das sich alle anderen Potentiale beziehen. Der Masse-Brick stellt in allen Brick'R'knowledge-Sets genau diese Verbindung zu 0 V her.

In unserer Schaltung sind das 9 Volt gegenüber 0 Volt: Man spricht einfach nur „Neun Volt“. Man erstellt in der Elektronik Schaltungen so, dass nachdem alle Bauelemente in ihrer Funktionsweise in die mehr oder weniger komplexen Stromkreise eingebracht sind, diese mit der „Masse“ verbunden werden. Schaltpläne sind nur so zu lesen.

In der Praxis verbindet unser Masse-Brick (rechts) die beiden mittleren Kontakte mit den beiden äußeren. Doch keine Angst, wir verursachen damit keinen Kurzschluss, denn der Strom durchfließt ja über die mittleren Kontakte die Bauelemente in unserem Brick-Stromkreis. Der rote Pfeil in der Abbildung symbolisiert den Pluspol und die braunen Pfeile zeigen die Masserückführung zum Minuspol der Spannungsversorgung.

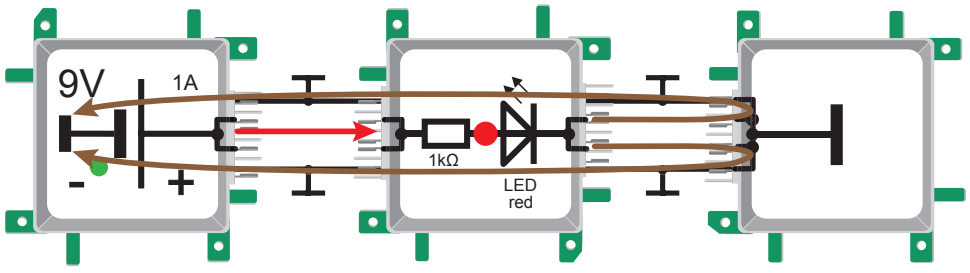


Abb. 1: Die Masse-Verbindung

3.2 Die Spannungsversorgung

Abb. 2: Netzteil-Adapter

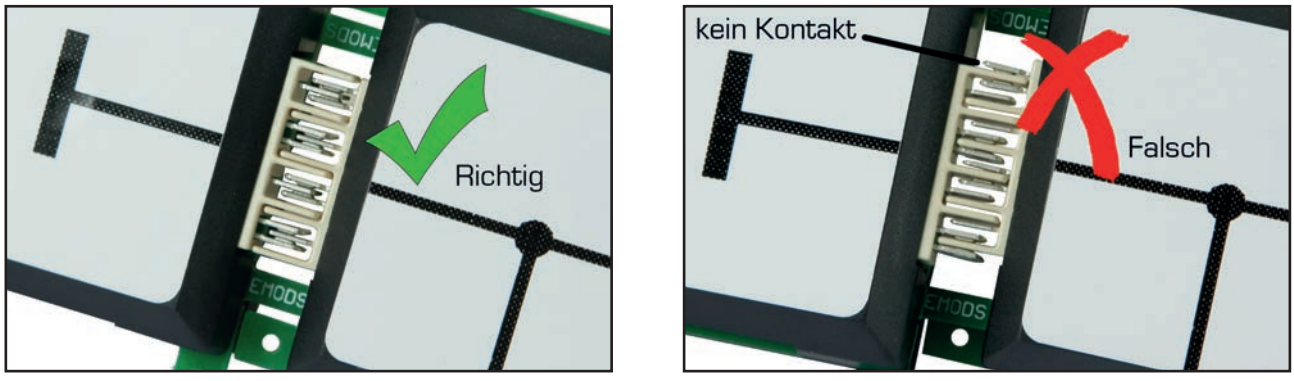
Die Spannungsversorgung des IoT Sets erfolgt über das mitgelieferte 9 V-Steckernetzteil (ALL-BRICK-0221). Es liefert eine stabilisierte Gleichspannung von 9 V und einen Maximalstrom von 1 A. Bei Überlastung schaltet das Netzteil ab, d. h. es ist kurzschlusssicher. Eine LED zeigt an, sobald der Brick Spannung bereitstellt.

Optional steht auch ein Versorgungs-Brick via 9 V-Blockbatterie (ALL-BRICK-0001) zur Verfügung.

Wenn du später die Bricks in den Übungsbeispielen zusammensteckst, achte darauf, den Versorgungs-Brick stets als letzten Brick an deine Schaltung zu stecken, nachdem du diese nochmals kontrolliert hast. **Am Ende der Versuchsdurchführung muss das Netzteil vom Stromnetz getrennt werden!**

3.3 Die Steckverbinder

Beim Zusammenstecken der Bricks muss darauf geachtet werden, dass sich die Kontakte richtig berühren, da sonst die Gefahr von Unterbrechungen oder sogar Kurzschlüssen besteht!



Richtig gesteckt Falsch gesteckt

Abb. 3: Die Steckverbinder

Im linken Bild seht ihr eine richtig gesteckte Verbindung. Die Verbindung besteht jeweils aus kleinen Stiften, die sich mechanisch verklemmen und dabei eine elektrische Verbindung herstellen. Um eine Isolation zwischen den Kontakten zu gewährleisten und einen Kurzschluss zu verhindern sind dazwischen Stege aus Kunststoff eingebracht, die den elektrischen Strom nicht leiten.

Ein Beispiel einer fehlerhaften Verbindung ist im rechten Bild zu sehen. Hier treffen Isolierstege auf Kontakte, sodass kein Strom fließen kann. Der Stromkreis bleibt „offen“ oder ist instabil und die Funktion der Schaltung ist nicht gegeben.

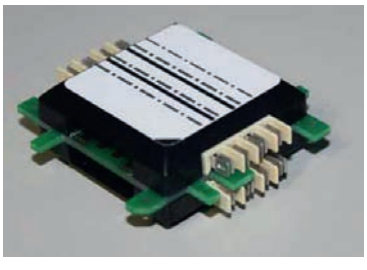
Achtung: Es ist wichtig, grundsätzlich immer den richtigen Sitz der Kontaktstifte zu kontrollieren. Weichen diese zu weit voneinander ab, kann es zu einem Kurzschluss kommen. Dann findet der Stromfluss nicht durch unsere Bauelemente mit der erhofften Wirkung statt, sondern sucht sich den kürzesten Weg zurück zur Spannungsquelle.

Ein Kurzschluss führt zum Maximalstromfluss, da der einzige Widerstand, den der elektrische Strom überwinden muss, der Innen-Widerstand der Spannungsquelle ist. Dieser Widerstand ist anschaulich sehr klein, sodass der Kurzschlussstrom bei längerer Dauer zur Überhitzung führen kann. Es besteht Brandgefahr!

Wichtig: Immer die richtige Stellung der Kontakte überprüfen!

3.4 Spezial-Verbindungsbricks für untere Ebene

Der IoT Brick hat auch einen Signal-Pin (GPIO12) auf der unteren Steckebene. Um an diesen Kontakt heranzukommen gibt es optional erhältliche Spezial-Bricks wie "Leitung Downside Up" (ALL-BRICK-0385) und "Leitung 6-fach Gerade" (ALL-BRICK-0383). Hier muss man beim Zusammenstecken besondere Vorsicht walten lassen. Eine Schutznase verhindert, dass man die Bricks von oben stecken kann, um Kurzschlüsse beim Steckvorgang zu vermeiden.



4. Die Hardware im Überblick

Im Lieferumfang des Internet of Things Set befinden sich folgende Bricks, Sensoren und Zubehör, welche in diesem Kapitel kurz vorgestellt werden.

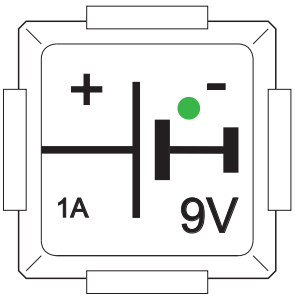
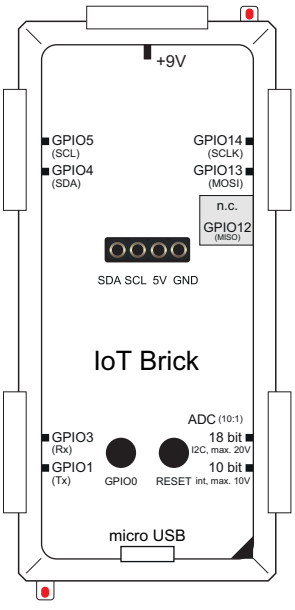
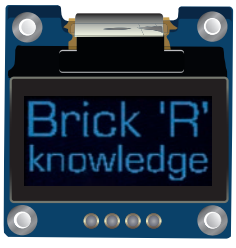
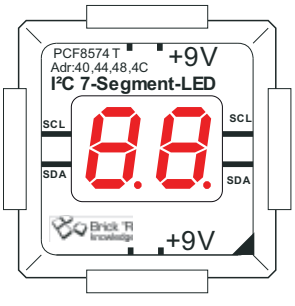
Abbildung	Anzahl	Art.-Nr. / Brick-ID	Kurzbeschreibung
	1	Art.-Nr.: 118627 Brick-ID: ALL-BRICK-0221	**9 V Netzteiladapter** Der 9 V-Netzadapter liefert maximal 1 A Strom für die Bricks. Er ist stabilisiert und vor Kurzschlüssen gesichert. Eine LED zeigt an, sobald er in Betrieb ist. Der Pluspol ist herausgeführt und der Minuspol ist mit Masse verbunden. Setze diesen Brick als letztes ein, nachdem du die Schaltung noch einmal kontrolliert hast.
	1	Art.-Nr.: 136716 Brick-ID: ALL-BRICK-0635	**IoT Brick ESP8266** Das Herzstück des IoT-Sets. ESP8266-Modul mit WLAN-Interface, Versorgungsspannung: +9 V. 7 GPIOs, ein 10 bit A/D-Wandler, ein 18 bit A/D-Wandler, I^2C-Interface, SPI-Interface Weitere Details siehe Kap. 5.1 auf Seite 13.
	1	Art.-Nr.: 139779 Brick-ID: ALL-BRICK-0659	**OLED-Display für IoT Brick** Organisches Leuchtdisplay monochrom. Genaugenommen sind 128 x 64 Leuchtdioden in einer Matrix angeordnet und können einzeln mit Hilfe von Befehlen durch den I^2C-Bus angesteuert werden. Damit lassen sich mehrzeilige Texte darstellen, aber auch einfache monochrome Grafiken. Standard-I2C-Adresse: $78_{(16)}$
	1	Art.-Nr.: 113713 Brick-ID: ALL-BRICK-0086	**I^2C 7-Segmentanzeige** 7-Segmentanzeige bestehend aus 7 Leuchtbalken und einem Punkt. Dahinter verbirgt sich je eine Leuchtdiode. Die LEDs werden mit Hilfe zweier 8574T Bausteine angesteuert (16 Ausgangsleitungen an die LEDs). Auf der Rückseite lässt sich mit kleinen Schaltern die I^2C-Adresse einstellen. Maximal können vier solcher Bausteine an einem I^2C-Bus betrieben werden.

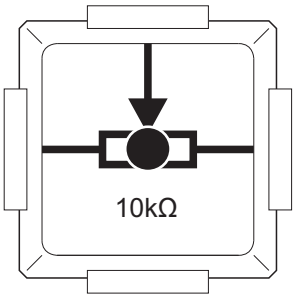
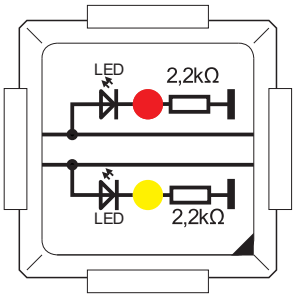
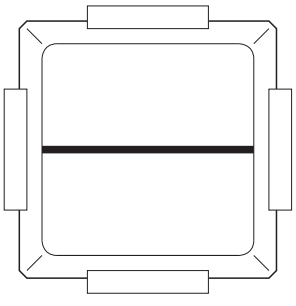
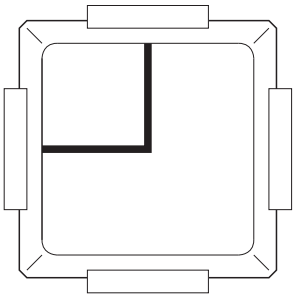
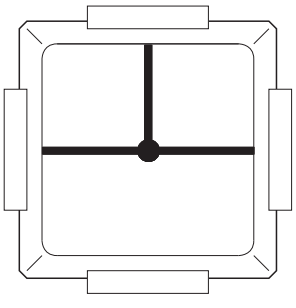
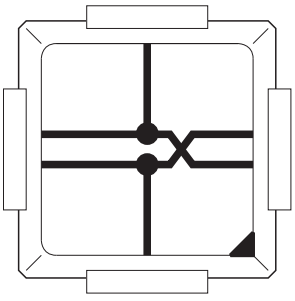
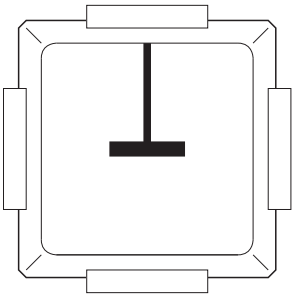
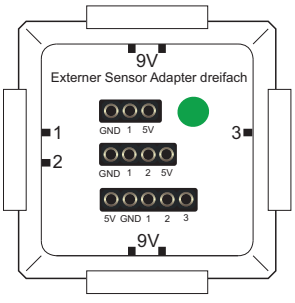
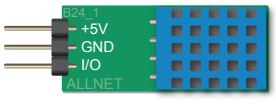
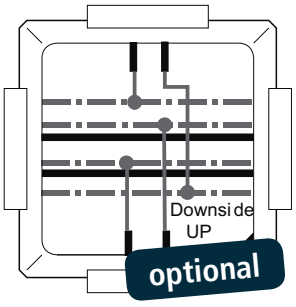
Abbildung	Anzahl	Art.-Nr. / Brick-ID	Kurzbeschreibung
	1	Art.-Nr.: 113654 Brick-ID: ALL-BRICK-0027	**10 kΩ Potentiometer** Das Potentiometer ist ein manuell veränderbarer Widerstand. Hier fährt ein dritter Kontakt (Schleifer) die Länge des Widerstandes ab und ändert so die Höhe des elektrischen Widerstandswertes an seinem Anschluss. Er ist im Bereich 0 bis 10 kΩ einstellbar. Ist der Schleifer oder auch Mittelkontakt genannt oder einer der anderen Kontakte direkt mit der Spannungsversorgung verbunden, so kommt es zu einem Kurzschluss. Dies ist unbedingt zu vermeiden! Das Potentiometer hat eine maximale Leistung von ca. 1/8 W.
	1	Art.-Nr.: 125693 Brick-ID: ALL-BRICK-0410	**Dual-LED auf Masse (rot/gelb)** In dem Baustein sind zwei LEDs (rot/gelb) untergebracht, die intern mit Masse verbunden sind. Die Signalleitungen sind getrennt durchverbunden. Beide LEDs sind über einen 2,2 kΩ Vorwiderstand vor zu hohem Stromfluss geschützt. Sie sind für 2 mA Strom bei 5 V Spannung ausgelegt. Die beiden Widerstände sind intern mit Masse verbunden, sodass man den Baustein direkt anschließen kann.
	1	Art.-Nr.: 113631 Brick-ID: ALL-BRICK-0004	**Leitung Gerade** Die gerade Leitung verbindet zwei gegenüberliegende Anschlüsse miteinander.
	3	Art.-Nr.: 113632 Brick-ID: ALL-BRICK-0005	**Leitung Ecke** Mit dem Eck-Brick werden zwei angrenzende Seiten miteinander verbunden.
	1	Art.-Nr.: 113633 Brick-ID: ALL-BRICK-0006	**Leitung T-Kreuzung** Mit der T-Kreuzung werden Abzweigungen hergestellt. Der Brick kann auch anstelle eines Eck-Bricks verwendet werden.

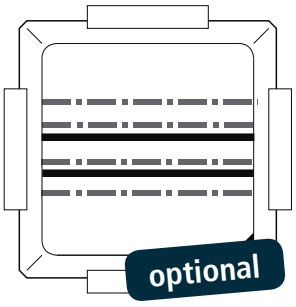
Abbildung	Anzahl	Art.-Nr. / Brick-ID	Kurzbeschreibung
	1	Art.-Nr.: 113675 Brick-ID: ALL-BRICK-0048	**Leitung doppelt überkreuzt** Mit diesem Baustein kann man die mittleren Leitungen getrennt weiterleiten und gleichzeitig kreuzen.
	1	Art.-Nr.: 113630 Brick-ID: ALL-BRICK-0003	**Leitung Masse-Brick** Mit dem Masse-Brick kann man auf einfache Weise an jede Stelle der Schaltung Masse anlegen. Er ist notwendig um den/die Stromkreis(e) zu schließen. Der Masse-Brick verbindet die beiden mittleren Kontakte des Anschlusses mit den beiden außen liegenden Masseleitungen.
	1	Art.-Nr.: 138408 Brick-ID: ALL-BRICK-0649	**Externer Sensor-Adapter dreifach** Versorgungsspannung: +9 V.
	1	Art.-Nr.: 139778 Brick-ID: ALL-BRICK-0658	**Temperatur- & Luftfeuchtigkeitssensor** Temperatur-/Luftfeuchtigkeitssensor auf Platine, Sensor-Typ: DHT11, Temp.: 0..50°C (±2°C), rel. Feuchte: 20..95% (±5%), Spannungsversorgung: 3-5,5V. Eingebauter Pullup Widerstand.
	1		**USB-Kabel** USB-Verbindungskabel (Stecker Typ A auf Stecker Micro-USB Typ B) zwischen Entwicklungsrechner und IoT Brick.

Optional empfohlene Bricks



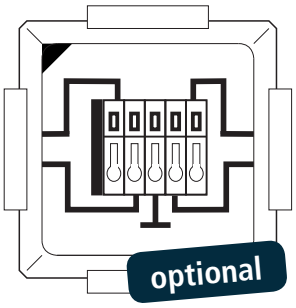
Art.-Nr.: 122448
Brick-ID: ALL-BRICK-0385

Leitung Downside Up
Ein Spezialbaustein, um Zugriff auf das Pin GPIO12 der unteren Steckebene des IoT Bricks zu bekommen. Er besitzt zusätzlich zu den Kontakten auf der oberen Ebene auch noch vier unabhängige Kontakte auf der Unterseite der Platine. Diese werden dann an den Seiten auf die obere Ebene geführt, dort können normale Bricks verwendet werden, um an die Signale heranzukommen.



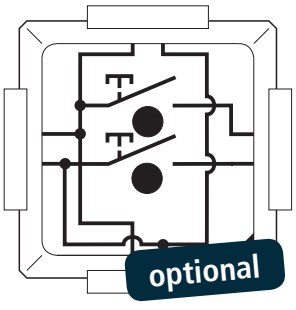
Art.-Nr.: 122446
Brick-ID: ALL-BRICK-0383

Leitung 6-fach Gerade
Die 6-fach Gerade verlängert die vier Kontakte oben (inklusive der Masse an den Außenkontakten) und die vier Signalleitungen auf der unteren Ebene, die unabhängig voneinander sind.



Art.-Nr.: 125674
Brick-ID: ALL-BRICK-0407

Klemme 5-polig Typ 2
Damit können Leitungen oder Bauteile an die Schaltung angeschlossen werden. Mit einem kleinen Schraubendreher drückt man dazu auf den Schlitz oben. Es öffnet sich dann der Kontakt und die Leitung kann seitlich davon eingeführt werden. Beim Loslassen des Schraubendrehers sitzt die Leitung fest. Der mittlere Kontakt ist mit Masse verbunden.



Art.-Nr.: 137824
Brick-ID: ALL-BRICK-0642

Doppeltaster
Dieser Brick enthält 2 einpolige Schließer. Damit können die Eingänge von Gattern oder Flipflops bequem beschaltet werden. Zusätzlich sind die Signalpfade getrennt vom oberen zum unteren Anschluss durchverbunden. Schließe die Versorgung am oberen und/oder unteren Anschluss an und spare so in vielen Situationen zahlreiche Leitungs-Bricks.

5. Der IoT Brick und die Arduino IDE

5.1 Das Herzstück des Internet of Things Set

Das Herzstück des Brick'R'knowledge Internet of Things Set ist der IoT Brick. Dessen zentrale Komponente wiederum ist das ESP-12-F-Modul von AI-THINKER. Es besteht aus einem Microcontroller vom Typ ESP8266 mit integrierter WLAN-Schnittstelle, einem 4 MB Flash-Speicher und einer internen WLAN-Antenne.

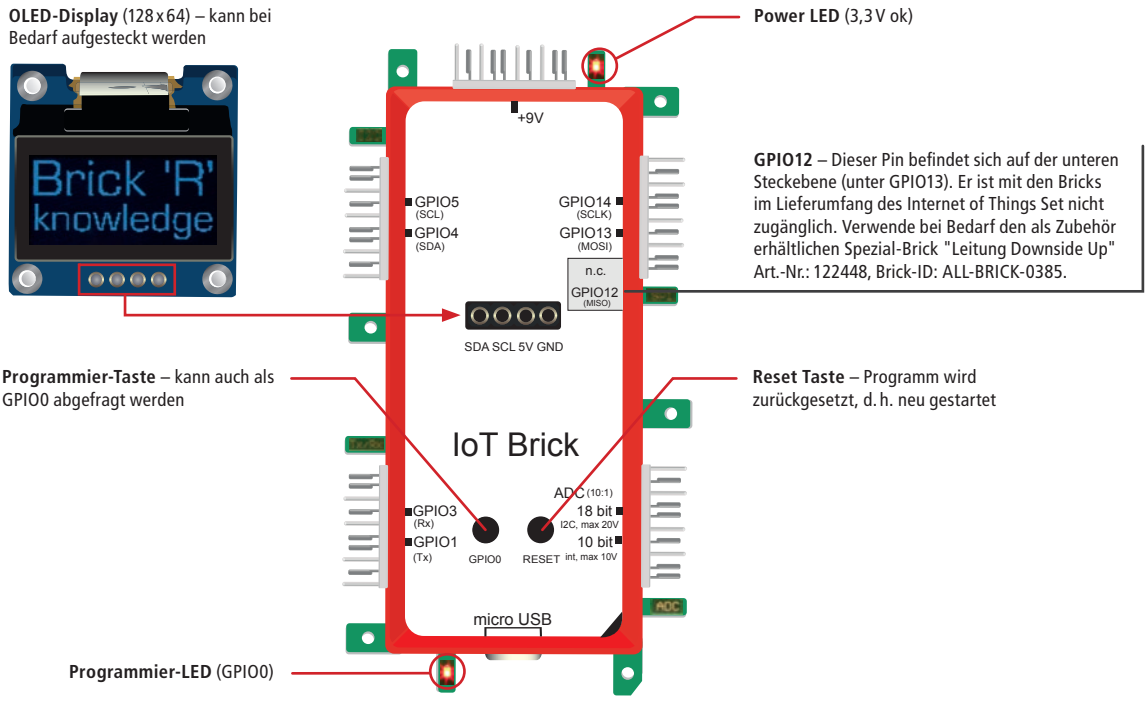


Abb. 4: Das Herzstück des Internet of Things Set

5.1.1 Spezifikation des IoT Bricks

Element	Spezifikation
Microcontroller	80 MHz Tensilica L106 Ultra-Low-Power 32 bit MCU
Flash-Speicher	4 MByte für Programme
WLAN-Interface	IEEE 802.11 b/g/n-Protokoll, IPv4, IP-Adresse: wird per DHCP zugewiesen, integrierter TCP/IP-Stack, ISM-Band (2,4 GHz), unterstützt WPA/WPA2 Sicherheitsmodi, integrierte Antenne
GPIOs	7 GPIOs, Standardfunktion: Digital-Ein-/Ausgang, alternative Funktionen: I^2C- und SPI-Bus GPIO-Ausgangspegel: +5 V; GPIO-Eingangspegel: +5 V (die Eingänge sind nicht 9 V tolerant!)
Analog-Eingänge	**ADC 10 bit**: 10 bit A/D-Wandler intern (Typ: SAR-Wandler), Abtastrate: max. 200 S/s, Eingangsspannungsbereich: max. 10 V **ADC 18 bit**: 18 bit A/D-Wandler MCP3421 über I^2C angebunden (Typ: Delta-Sigma-Wandler), Abtastrate: max. 3,75 S/s, Eingangsspannungsbereich: max. 20 V
Taster	2 Taster (Programmiertaste GPIO0 & Reset): werden für Programmiermodus benötigt (nach Start des ESP8266 ist der Taster GPIO0 als normaler digitaler Eingang nutzbar)
Versorgung	9 V-Versorgung (Power-LED an 3,3 V on-Board-Spannung)
Anschlüsse	1 x Brick-Steckverbinder für 9 V-Versorgung 4 x Brick-Steckverbinder für insgesamt 7 GPIOs 1 x Brick-Steckverbinder für 2 Analogeingänge Micro-USB-Buchse (Typ B) als Programmierschnittstelle
Display	I^2C-OLED-Display monochrom (128 x 64 Pixel) über 4-polige Buchsenleiste aufsteckbar (im Set inklusive)

5.1.2 Die GPIO-Pins des IoT Bricks

Damit ein Microcontroller Daten austauschen kann, sind analoge und digitale Ein- bzw. Ausgänge erforderlich. Die digitalen Ein-/Ausgänge werden meist als GPIOs bezeichnet. GPIO ist die Abkürzung für General Purpose Input/Output und besagt, dass ein derartiger Anschluss – je nach vorheriger Konfiguration – wahlweise als Eingang oder Ausgang genutzt werden kann. In der Programmierung wird diese Richtungsumschaltung als `pinMode` bezeichnet. Der Spannungspegel der GPIOs am IoT Brick entspricht dabei 5 V für High-Pegel und 0 V für Low-Pegel. Manche GPIO-Pins können alternativ auch Sonderfunktionen übernehmen, wie z. B. die Datenkommunikation via I²C- oder SPI-Bus. Um den gewünschten GPIO-Pin in deinem Programm anzusprechen, benötigst du einen Index, den du aus folgender Tabelle entnehmen kannst:

Standard-Funktion	Beschreibung	Alternativ-Funktion	Pullup-Widerstand (siehe Kap. 5.1.3)	Index für Programmierung
GPIO0	Programmier-Taste (Eingang)	-	low: 40 kΩ, high: 4 kΩ	0
GPIO1	Digital-I/O 1	TxD	low: 40 kΩ, high: 4 kΩ	1
GPIO3	Digital-I/O 3	RxD	low: 40 kΩ, high: 4 kΩ	3
GPIO4	Digital-I/O 4	I^2C SDA	low: 40 kΩ, high: 4 kΩ	4
GPIO5	Digital-I/O 5	I^2C SCL	low: 40 kΩ, high: 4 kΩ	5
GPIO12**	Digital-I/O 12	SPI MISO	low: 40 kΩ, high: 4 kΩ	12
GPIO13	Digital-I/O 13	SPI MOSI	low: 40 kΩ, high: 4 kΩ	13
GPIO14	Digital-I/O 14	SPI SCLK	low: 40 kΩ, high: 4 kΩ	14
ADC0 (10 bit)	10 bit Analog-Eingang (intern)	-	-	0
ADC1 (18 bit)	18 bit Analog-Eingang (via I^2C)	-	-	(via I^2C)
Die GPIOs 2, 6, 7, 8, 9, 10, 11, 15 und 16 werden vom ESP-12-F-Modul nicht zur Verfügung gestellt.				

*siehe nächstes Kapitel. **Pin auf unterer Steckebene nur mit optionalem Spezial-Brick zugänglich (Art.-Nr.: 122448, Brick-ID: ALL-BRICK-0385)

Achtung: Lege an die GPIOs niemals direkt 9 V an (z. B. über einen Taster). Zur Versorgung des IoT Bricks und anderer aktiver Bricks werden zwar 9 V benötigt, die GPIOs sind jedoch nur für 5 V-Pegel ausgelegt. **Bei Spannungen über 5 V an den GPIOs kann es zur irreversiblen Beschädigung deiner Bricks kommen!**

5.1.3 Pullup-Widerstände

In digitalen Schaltungen ist es wichtig, dass Eingänge stets einen definierten Pegel haben. Dazu verwendet man einen kleinen Trick, indem man sog. Pullup- oder Pulldown-Widerstände anbringt, damit der Eingang nicht "in der Luft hängt", sondern auf High-Pegel (5 V) oder Low-Pegel (0 V) gezogen wird. In unserem IoT Brick werden Pullup-Widerstände mit automatischer Widerstandsanpassung verwendet. Bei Low-Pegel beträgt der Wert 40 kΩ (es fließt weniger Strom, bei High-Pegel sind es 4 kΩ (damit der High-Pegel sicher erkannt wird). Somit ist der Eingangspegel an den GPIOs immer klar definiert, sodass du den Logikpegel an einem mit `pinMode(`*`GPIOx`*`, INPUT)` konfigurierten Pin immer zuverlässig auswerten kannst. Die Befehle `pinMode(`*`GPIOx`*`, INPUT_`**`PULLUP`**`)` bzw. `pinMode(`*`GPIOx`*`, INPUT_`**`PULLDOWN`**`)` haben in unserem Fall keine Auswirkung auf die GPIO-Pins, da die Pullup-Widerstände immer aktiv sind. Nach dem Einschalten der Versorgung werden als Eingang konfigurierte GPIOs sofort auf High-Pegel gelegt.

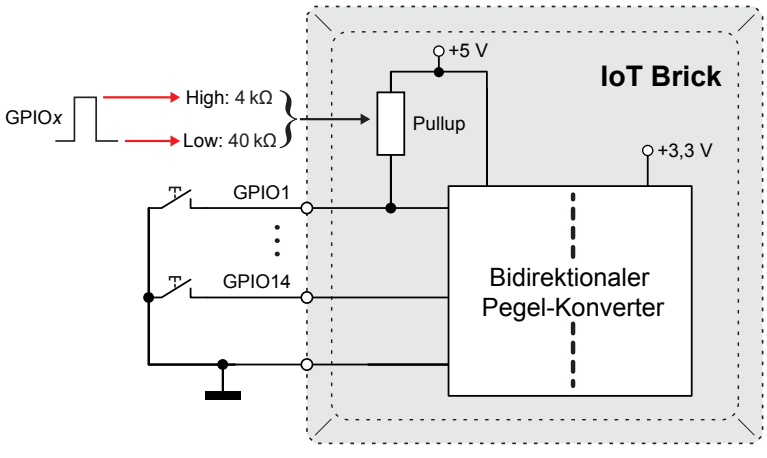


Abb. 5: Pullup-Widerstände

5.2 Die Arduino Entwicklungsumgebung

Um den IoT Brick zu programmieren verwenden wir die Arduino-Entwicklungsumgebung – auch Arduino IDE (IDE steht für Integrierte Entwicklungsumgebung) genannt. Viele kennen die kostenlose Programmier-Software vielleicht schon aus der Arduino-Welt. Es gibt im Internet zahlreiche Arduino-basierende Projekte und eine große Community. In unseren Übungsbeispielen werden wir zahlreiche Open-Source-Bibliotheken einbinden um uns die Programmierung einzelner Hardware-Komponenten zu erleichtern. Es stehen Installer für Windows, Linux, MAC OS X und eine Windows App zur Verfügung. Die Beschreibung und Screenshots dieser Anleitung beziehen sich auf die Windows Version.

- Lade den Installer für die aktuelle Arduino IDE unter https://www.arduino.cc herunter.
- Starte die Installation der Arduino IDE durch Doppelklick auf die heruntergeladene EXE-Datei.

Damit das ESP8266-basierende Microcontroller-Modul mit Hilfe dieser Software programmiert werden kann, muss jetzt noch der entsprechende Core inklusive ein paar benötigter Bibliotheken, sog. Libraries, installiert werden. Dies ist notwendig, da die Arduino-IDE den ESP8266 nicht standardmäßig unterstützt.

- Starte die Arduino IDE, z. B. über das Startmenü unter Windows (je nach Betriebssystem unterschiedlich).
- Öffne das Menü "Datei - Voreinstellungen" und trage unter "Zusätzliche Boardverwalter-URLs" folgende URL ein: http://arduino.esp8266.com/stable/package_esp8266com_index.json

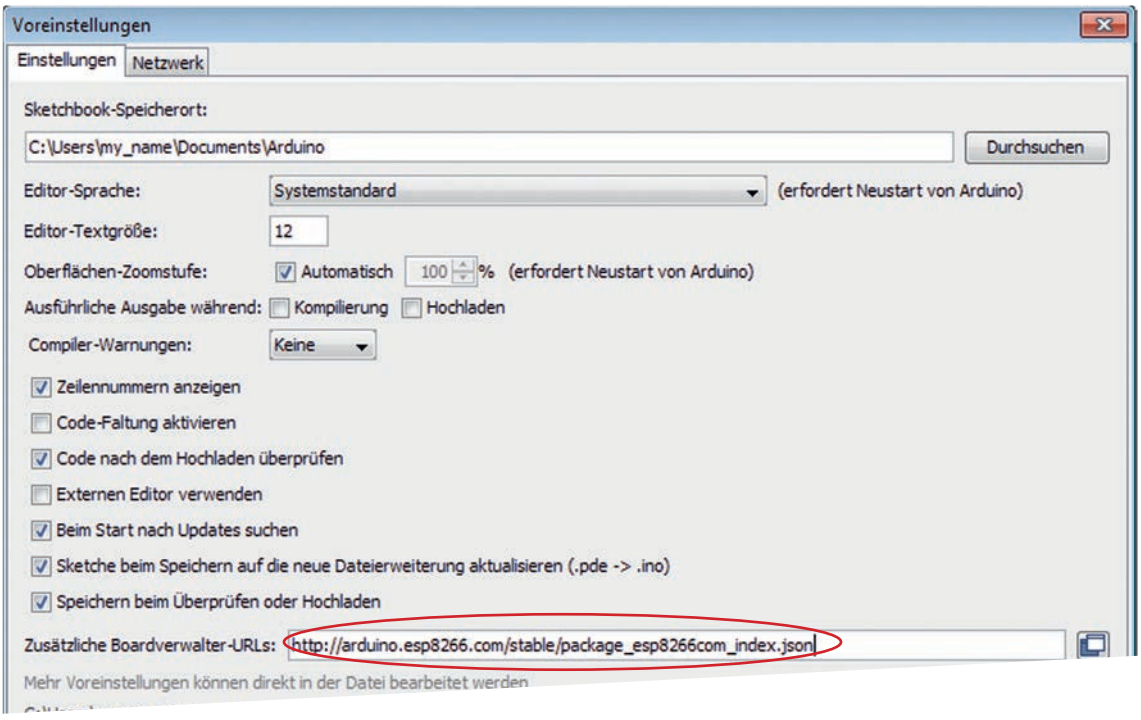


Abb. 6: Eintrag für zusätzliche Boardverwalter-URLs

- Bestätige mit OK.
- Öffne im Menü "Werkzeuge – Board:..." den "Boardverwalter...".

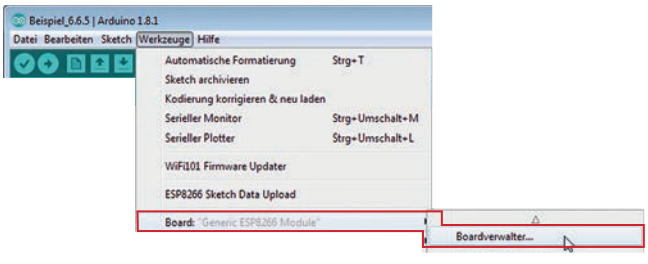


Abb. 7: Boardverwalter

- Gib hier in das Suchfenster esp8266 ein. Es sollte nur der Eintrag "esp8266 by ESP8266 Community" angezeigt werden.

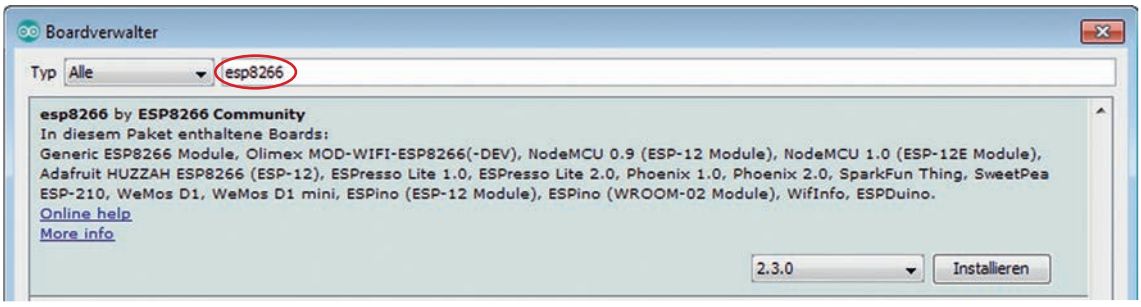


Abb. 8: Installation ESP8266 via Boardverwalter

- Klicke auf den Eintrag und anschließend auf "Installieren". Die Installation kann einige Minuten dauern.
- Beende die Installation mit "Schließen"
- Gehe nun wieder ins Menü "Werkzeuge – Board: … – "Boardverwalter…".
- Wähle den Eintrag "Generic ESP8266 Module" aus. Sobald ausgewählt, ändert sich das Menü unterhalb von "Board: …" und zeigt zahlreiche Einstellungen. Ändere diese gegebenenfalls so, dass sie mit dem Screenshot (rotes Rechteck) übereinstimmen.

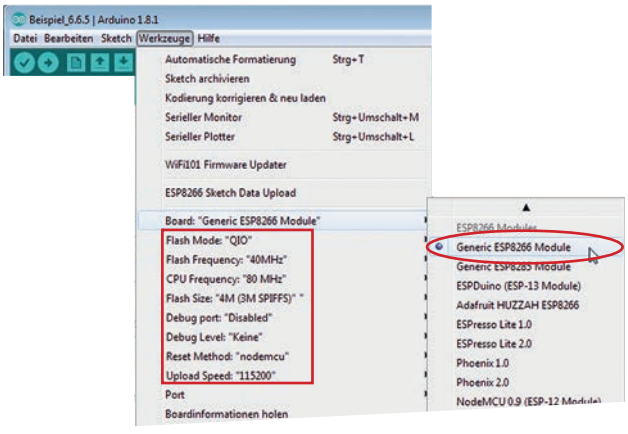


Abb. 9: ESP8266 im Boardverwalter auswählen

- Mache nun mit dem folgenden Kapitel "Bibliotheken installieren" weiter…

5.2.1 Bibliotheken installieren

Zur Vereinfachung der Programmierung verwenden wir in unseren Beispielprogrammen sog. Bibliotheken oder englisch auch "Libraries" genannt. Diese Code-Sammlungen enthalten z. B. umfangreiche Tabellen, mit welchen z. B. Zeichensätze definiert oder die Ansteuerung von 7-Segmentanzeigen codiert werden. Die meisten Bibliotheken werden mit der Arduino IDE mitgeliefert, müssen aber noch explizit installiert werden. Andere Bibliotheken müssen erst aus dem Internet heruntergeladen werden und werden in der Regel als ZIP-Datei eingebunden. Wenn du jetzt alle der hier gelisteten Bibliotheken installierst, bist du bestens gerüstet und musst für die einzelnen Übungen nichts nachinstallieren.

Übrigens findest du die von dir installierten Bibliotheken standardmäßig in folgendem Verzeichnis auf deinem Rechner: C:\Users*my_name*\Documents\Arduino\libraries (ersetze *my_name* mit deinem Benutzernamen).

Hinweise zur Installation zusätzlicher Arduino-Bibliotheken findest du auch unter:

https://www.arduino.cc/en/Guide/Libraries

5.2.1.1 Arduino-Bibliotheken installieren

- Öffne im Menü "Sketch – Bibliothek einbinden – Bibliotheken verwalten …" den Bibliotheksverwalter.

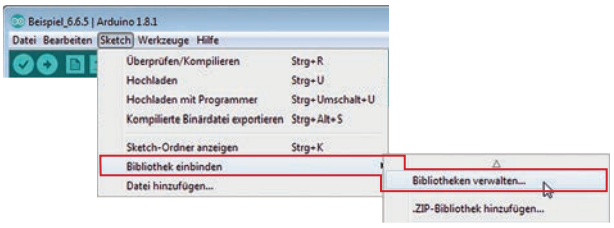


Abb. 10: Bibliotheksverwalter

- Installiere nun die benötigten Bibliotheken anhand der folgenden Kapitel. Grenze die Auswahl durch geeignete Suchbegriffe ein (siehe folgende Screenshots). Sofern eine Versionsauswahl angeboten wird, kann in der Regel die neueste Version installiert werden. Klicke auf den Eintrag und anschließend auf "Installieren".

5.2.1.1.1 Bibliothek "NTPClient"

Die Bibliothek wird benötigt, um Zeit und Datum via Network Time Protocol (NTP) aus dem Internet zu beziehen.

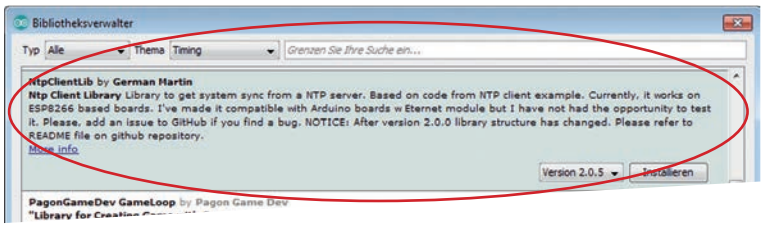


Abb. 11: Installation Bibliothek "NTPClientLib"

- Wähle die Bibliothek mit der neuesten Version aus und klicke auf "Installieren".
- Die Headerdatei wird mit der Anweisung `#include <NTPClientLib.h>` eingebunden.

5.2.1.1.2 Bibliothek "Time"

Die Bibliothek wird benötigt, um Zeit und Datum via Network Time Protocol (NTP) aus dem Internet zu beziehen.

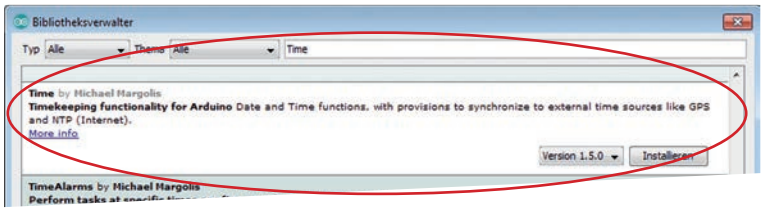


Abb. 12: Installation Bibliothek "Time"

- Wähle die Bibliothek mit der neuesten Version aus und klicke auf "Installieren".
- Die Headerdateien werden mit den Anweisungen `#include <Time.h>` und `#include <TimeLib.h>` eingebunden.

5.2.1.1.3 Bibliothek "Json Streaming Parser"

Die Bibliothek wird benötigt, um den aktuellen Dollarkurs aus dem Internet abzufragen.

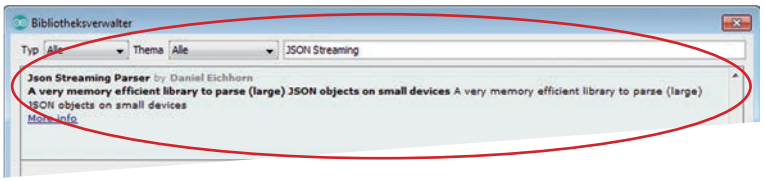


Abb. 13: Installation Bibliothek "Json Streaming Parser"

- Wähle die Bibliothek mit der neuesten Version aus und klicke auf "Installieren".
- Die Headerdatei wird mit der Anweisung `#include <JsonListener.h>` eingebunden.

5.2.1.1.4 Bibliothek "DHT Sensor Library"

Die Bibliothek wird benötigt, um Temperatur und Feuchtigkeit vom Sensor DHT11 einlesen zu können.

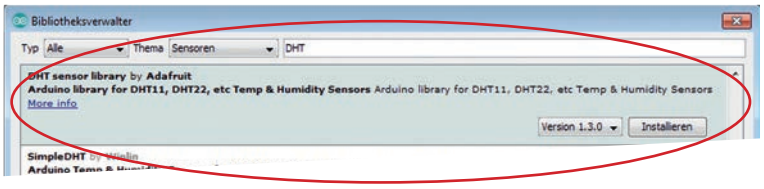


Abb. 14: Installation Bibliothek "DHT Sensor Library"

- Wähle die Bibliothek mit der neuesten Version aus und klicke auf "Installieren".
- Die Headerdateien werden mit den Anweisungen `#include <DHT.h>` und `#include <DHT_U.h>` eingebunden.

5.2.1.1.5 Bibliothek "Adafruit Unified Sensor"

Die Bibliothek wird benötigt, um Temperatur und Feuchtigkeit vom Sensor DHT11 einlesen zu können.

Abb. 15: Installation Bibliothek "Adafruit Unified Sensor"

- Wähle die Bibliothek mit der neuesten Version aus und klicke auf "Installieren".
- Die Headerdatei wird mit der Anweisung `#include <Adafruit_Sensor.h>` eingebunden.

5.2.1.2 Externe Bibliotheken installieren

Es gilt für jede Bibliothek die gleiche Vorgehensweise:

- Zunächst musst du die benötigte Bibliothek aus dem Internet herunterladen. Die jeweiligen Download-Seiten findest du in den folgenden Kapiteln.
- Anschließend öffnest du das Menü "Sketch – Bibliothek einbinden – .ZIP-Bibliothek hinzufügen …".
- Dort wählst du die heruntergeladene ZIP-Datei aus.

5.2.1.2.1 Bibliothek "esp8266-oled-ssd1306-master"

Um die Ansteuerung des OLED-Displays zu vereinfachen, verwenden wir die Bibliothek "esp8266-oled-ssd1306-master". Diese kann von der GitHub-Seite als ZIP-Datei heruntergeladen werden.

- Gehe zu: https://github.com/squix78/esp8266-oled-ssd1306. Wähle dort unter "Clone or download" die Option "Download ZIP" und speichere die ZIP-Datei auf deinem Rechner ab:

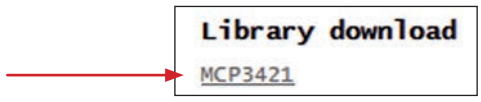
Abb. 16: Installation Bibliothek "esp8266-oled-ssd1306-master"

- Installiere die Bibliothek über das Menü "Sketch – Bibliothek einbinden – .ZIP-Bibliothek hinzufügen …".
- Die Headerdatei wird mit der Anweisung `#include <SSD1306Wire.h>` eingebunden.

5.2.1.2.2 Bibliothek "MCP3421"

Um uns die Kommunikation mit dem 18 bit A/D-Wandler MCP3421 zu vereinfachen, verwenden wir die Bibliothek "MCP3421". Diese kann als ZIP-Datei heruntergeladen werden.

- Folge dem Link: http://interface.khm.de/index.php/lab-log/connect-a-mcp3421-18-bit-analog-to-digital-converter-to-an-arduino-board/. Am Ende dieser Website findest du unter "Library download" den Link zur gesuchten Bibliothek.
- Lade die Datei `MCP3421.zip` herunter und speichere sie auf deinem Rechner.



- Installiere die Bibliothek über das Menü "Sketch – Bibliothek einbinden – .ZIP-Bibliothek hinzufügen …".
- Die Headerdatei wird mit der Anweisung `#include <MCP3421.h>` eingebunden.

5.2.1.2.3 Bibliothek "BrickESP8266"

Die Bibliothek wird unter anderem benötigt um den aktuellen Dollarkurs aus dem Internet abzufragen. Diese kann als ZIP-Datei heruntergeladen werden.

- Folge dem Link: http://www.brickrknowledge.de/downloads.
- Lade die Datei `BrickESP8266.zip` herunter und speichere sie auf deinem Rechner.
- Installiere die Bibliothek über das Menü "Sketch – Bibliothek einbinden – .ZIP-Bibliothek hinzufügen …".
- Die Headerdatei wird mit der Anweisung `#include <CurrencylayerClient.h>` eingebunden

Falls eine Bibliothek fehlt, wird dies beim Kompilieren durch eine entsprechende Fehlermeldung angezeigt:

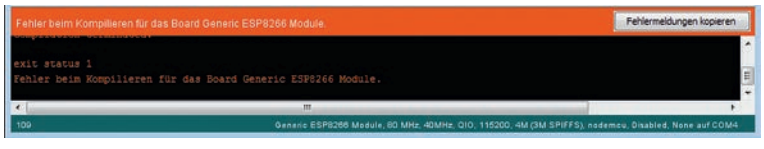


Abb. 17: Fehler beim Kompilieren

Falls du den Treiber für die USB-zu-UART-Bridge noch nicht installiert hast, fahre mit Kap. 5.2.2 fort.

5.2.2 Virtueller COM-Port-Treiber

Damit sich dein Entwicklungsrechner mit dem IoT Brick verständigen kann, muss in der Arduino IDE noch die Schnittstelle deines PCs ausgewählt werden, über welche die Verbindung zum IoT Brick erfolgen soll. Zu diesem Zweck kann man in der Arduino IDE einen seriellen Port (auch COM-Port genannt) auswählen. Klassischerweise sind dies RS-232-Schnittstellen, die jedoch in modernen Rechnern kaum mehr zu finden sind. Stattdessen verwenden wir eine freie USB-Schnittstelle deines Rechners, die wir der Arduino IDE als seriellen Port vorgaukeln. Um dies dem Betriebssystem klar zu machen, müssen wir einen virtuellen COM-Port Treiber (VCP) installieren. Dies ist Voraussetzung für die Kommunikation zwischen Arduino IDE und der USB-Schnittstelle des IoT Bricks.

- Lade den aktuellsten Treiber für dein Betriebssystem (Windows, MAC OS X, Linux) von der Silicon Labs Website herunter:
  https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx

Abb. 18: Screenshot der Download-Seite mit Link für aktuelle Windows-Versionen

- Entpacke die gepackte Datei auf deinem Rechner
- Starte die Installation mit Doppelklick. Je nach Windows-Version musst du den entsprechenden Installer starten: CP210xVCPInstaller_x86.exe für eine 32 Bit Windows-Version oder CP210xVCPInstaller_x64.exe für eine 64 Bit Version.
- Im Windows Gerätemanager muss unter "Anschlüsse (COM & LPT)" ein Eintrag ähnlich dem mit einem roten Pfeil markierten zu finden sein. Die Anzahl der COM-Ports und der Index hängen von deiner Rechner-Konfiguration ab.

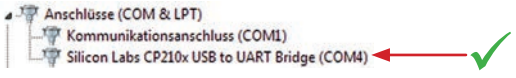


Abb. 19: Virtueller COM-Port im Gerätemanager

5.2.3 Serieller Monitor

Manche Beispielprogramme verwenden den sog. seriellen Monitor zur Anzeige von Werten und Meldungen direkt am Entwicklungsrechner. Um den seriellen Monitor zu öffnen, klickst du in der Arduino IDE einfach oben rechts auf das Lupen-Symbol. Die Baudrate im Monitorfenster (rechts unten) und im Sketch muss übereinstimmen (siehe Abb. 20).

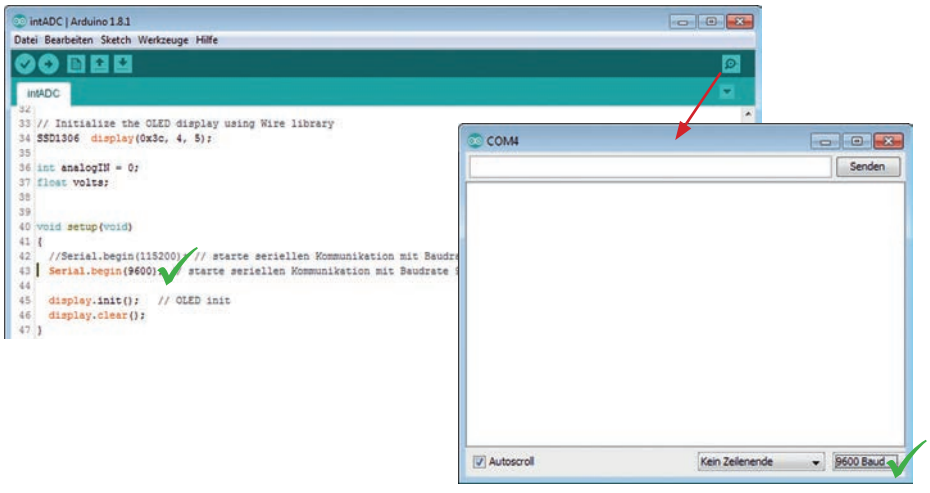


Abb. 20: Daten auf seriellen Monitor ausgeben

5.3 Erste Schritte

Jetzt wird´s spannend! Nachdem du die Arduino-Entwicklungsumgebung inkl. Bibliotheken und den virtuellen COM-Port-Treiber installiert hast, geht es nun an die Inbetriebnahme.

5.3.1 Verbindung herstellen

Verbinde zunächst den IoT Brick mit dem beiliegenden 9 V-Netzteil an dem dafür vorgesehenen, oberen Anschluss (siehe Abb. 21).

Achtung: Das 9 V-Netzteil niemals an die anderen Kontakte des IoT Bricks anschließen, dies könnte den Baustein zerstören!

Verbinde deinen Entwicklungsrechner und den IoT Brick (Micro-USB-Anschluss) mit dem mitgelieferten USB-Kabel.

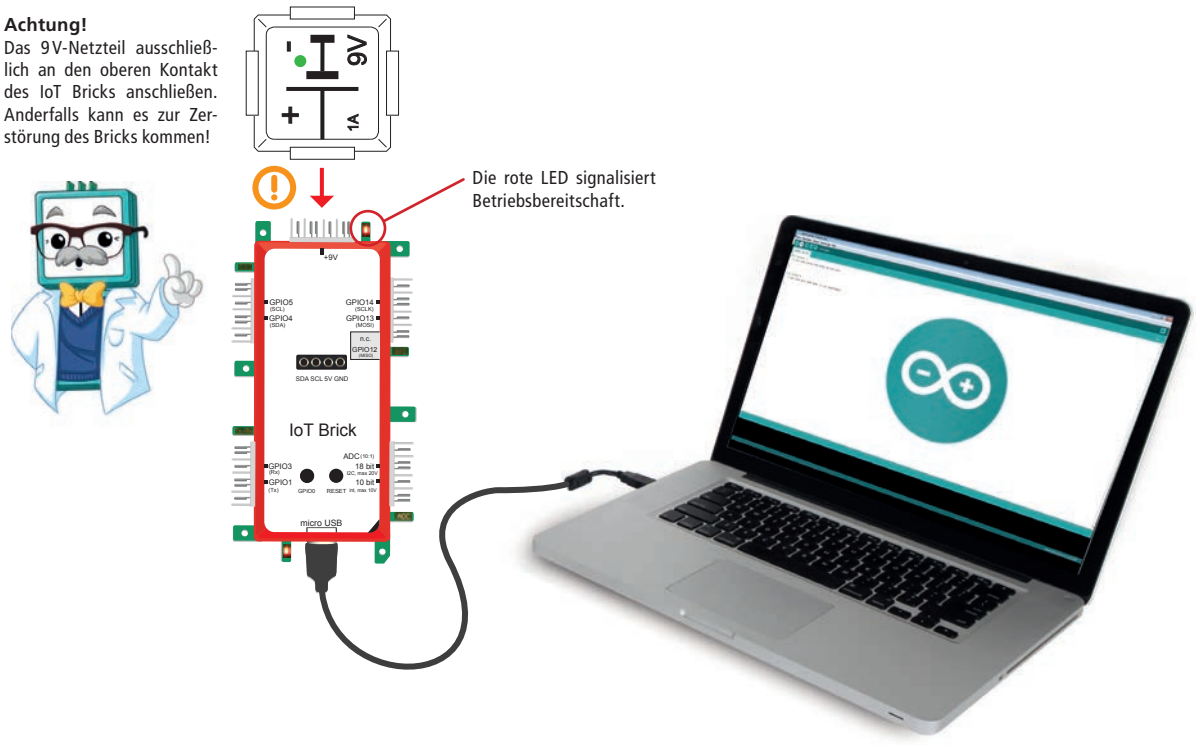


Abb. 21: Grundlegender Anschluss des IoT Bricks

Der Computer sollte jetzt einen neuen, sog. virtuellen COM-Port haben, über den die Arduino IDE die Programme in den IoT Brick lädt. Eine recht einfache Möglichkeit um den Index des COM-Ports herauszufinden, ist über die Arduino Entwicklungsumgebung selbst.

- Trenne dafür zuerst das USB-Kabel vom IoT Brick.
- Starte die Arduino-Software und prüfe welche COM-Schnittstelle(n) unter "Werkzeuge – Port:..." angezeigt werden.
- Verbinde jetzt wieder das USB-Kabel mit dem IoT Brick. Unter "Werkzeuge – Port:..." sollte jetzt eine weitere COM-Schnittstelle angezeigt werden. Wähle diese jetzt aus. Sollte dies nicht möglich sein, so deinstalliere den COM-Port im Geräte-Manager und installiere den "CP210x USB to UART Bridge"-Treiber erneut. Siehe auch Kap. 5.2.2 auf Seite 20.

Abb. 22: Links: USB-Kabel nicht gesteckt, rechts: USB-Kabel gesteckt (virtueller COM-Port ausgewählt)

Im Menü "Werkzeuge" musst du noch weitere Parameter auswählen bzw. überprüfen. Die im folgenden Screenshot dargestellten Einstellungen müssen ausgewählt werden. Unter "Port:" musst du die COM-Schnittstelle wählen, welche zuvor dem virtuellen COM-Port zugewiesen wurde.

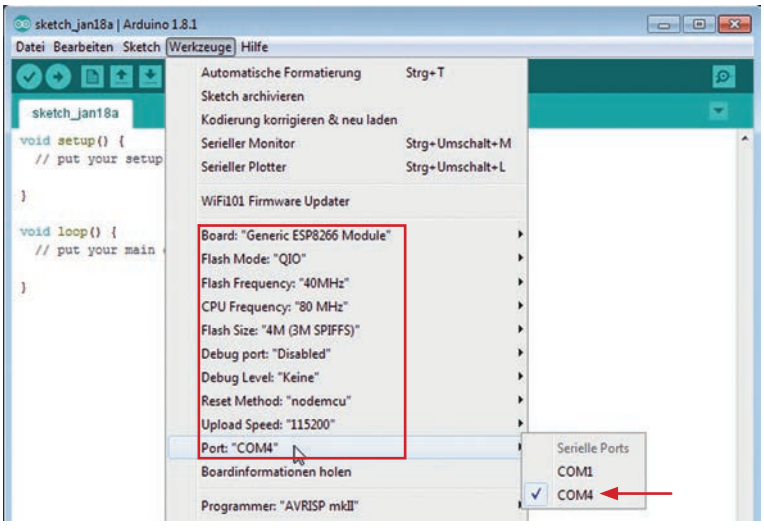


Abb. 23: Einstellungen für IoT Brick

5.3.2 Programm-Code kompilieren und hochladen

Mit einem Klick auf die Schaltfläche mit dem Häkchen wird der Programm-Code – in der Arduino-Welt Sketch genannt – auf Syntax-Fehler überprüft und kompiliert. Syntax-Fehler sind Tippfehler in deinem Sketch oder Verstöße gegen die Regeln der Programmiersprache. Beim Kompilieren wird dein Programmcode in Maschinencode übersetzt, den unser Microcontroller ausführen kann. Dies wird automatisch von einem im Arduino IDE integrierten Compiler (Übersetzer) gemacht.

Nachdem dein Programmcode erfolgreich kompiliert wurde, kannst du mit einem Klick auf den Pfeil-Button das Programm in den IoT Brick laden. Die rote LED auf dem IoT Brick links unten neben dem USB-Port leuchtet, während das Programm in den Flash-Speicher geladen wird. Beginn und Ende des Ladevorgangs wird jeweils durch ein kurzes Aufblitzen signalisiert. Nach Abschluss des Hochladens geht die LED wieder aus.

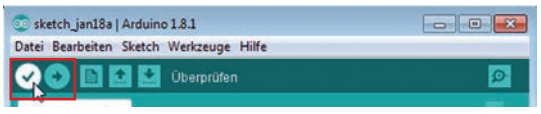


Abb. 24: Sketch überprüfen, kompilieren und hochladen

Wie du den Programmiermodus bei Bedarf manuell starten kannst, erfährst du im nächsten Kapitel.

5.3.3 Programmiermodus

Vorgehensweise, falls du den Programmiermodus von Hand aktivieren möchtest. Dies ist nur nötig, falls die Kommunikation zwischen Arduino IDE und IoT Brick einmal nicht klappen sollte.

1. Programmier-Taste (GPIO0) gedrückt halten (rote LED unten links leuchtet hell)
2. Gleichzeitig Reset-Taste drücken
3. Reset-Taste wieder loslassen
4. Danach die Programmier-Taste loslassen (rote LED unten links leuchtet schwach)

6. Übungsbeispiele

6.1 "Hello World" (Blink-LED)

Öffne in der Arduino IDE den Sketch: `Beispiel_6.1.ino`

Wie so oft in der Programmierwelt geht es mit einem "Hello World"-Beispiel los. In unserem Fall lassen wir zwei LEDs wechselweise blinken.

Ein Programm – in der Arduino-Welt Sketch genannt – besteht immer aus mindestens zwei Teilen.

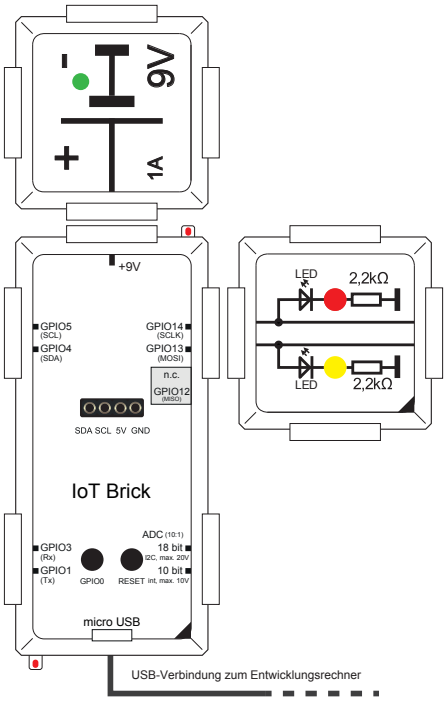
Die Setup-Routine `void setup()`

Zunächst kommt der Abschnitt `void setup() {…}`. Alle Befehle innerhalb dieser geschweiften Klammern werden beim Start des Programmes genau einmal ausgeführt. In unserem Beispiel wird hier die Richtung der GPIO-Pins definiert, also Eingang (Mode: Input) oder wie in unserem Fall: zwei Ausgänge (Mode: Output).

Die Programmschleife `void loop()`

Das eigentliche Programm steht innerhalb der geschweiften Klammern von `void loop() {…}` und wird permanent wiederholt. Die Befehle werden in einer Endlosschleife der Reihe nach ausgeführt bis das Programm, z. B. durch Drücken der Reset-Taste abgebrochen wird.

In unserem ersten Programmierbeispiel (`Beispiel_6.1.ino`) wird der Doppel-LED-Brick an die GPIOs 13 und 14 des IoT Bricks angeschlossen.




```
void setup() {
  pinMode(14, OUTPUT);     // Pin 14 ist Ausgang
  pinMode(13, OUTPUT);     // Pin 13 ist Ausgang
}

void loop() {
  digitalWrite(14, HIGH); // LED an Pin 14 an
  digitalWrite(13, LOW);  // LED an Pin 13 aus

  delay(1000);             // warte für 1000ms

  digitalWrite(14, LOW);  // LED an Pin 14 aus
  digitalWrite(13, HIGH); // LED an Pin 13 an

  delay(1000);             // warte für 1000ms
}
```

Abb. 25: Übungsbeispiel: "Hello World" (Blink-LED)

GPIO 13 und 14 werden in der Setup-Routine mit `pinMode(`*`GPIOx`*`, OUTPUT)` als Ausgang definiert. Somit sind diese Pins in der Lage, einen High-Pegel auszugeben um die beiden LEDs abwechselnd blinken zu lassen.

In der Loop-Schleife wird der Pin mit `digitalWrite(`*`GPIOx`*`, HIGH)` auf High gesetzt. Dies bedeutet, dass am jeweiligen GPIO 5 V ausgegeben werden und somit die LED leuchtet. Mit dem Befehl `digitalWrite(`*`GPIOx`*`, LOW)` wird der Pin auf 0 V gesetzt – die LED geht aus.

Mit dem Befehl `delay(…)` pausiert der Programmablauf für die entsprechende Anzahl an Millisekunden. In diesem Beispiel 1000 Millisekunden, was einer Sekunde entspricht.

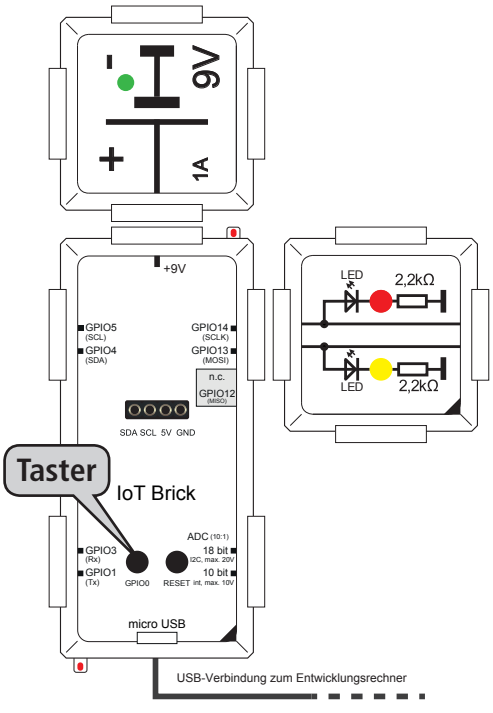
Nachdem du das Code-Beispiel in deinen IoT Brick geladen hast (siehe Kap. 5.3.2 auf Seite 22), blinken die beiden LEDs im Wechsel. Spiele ein bisschen mit dem Code. Ändere die Reihenfolge in der die LEDs an und wieder ausgehen, sowie die Zeitabstände dazwischen.

6.2 Taster und LED

Öffne in der Arduino IDE den Sketch: `Beispiel_6.2.ino`

In diesem Beispiel wollen wir die LEDs abhängig von einem Taster leuchten lassen. Der IoT Brick besitzt neben seinem Reset-Taster, einen weiteren Taster, welcher mit dem GPIO0 verbunden ist. Somit verwenden wir den GPIO0 als Eingang und lesen den Status des Tasters ein. Um zu vermeiden, dass an diesem Eingang ein undefinierter Pegel anliegt solange der Taster nicht gedrückt wird, wurde intern ein Pullup-Widerstand verbaut, der den Eingang auf High-Pegel zieht (siehe auch Kap. 5.1.3 auf Seite 14). Sobald der Taster gedrückt wird, liegt am Eingang 0V an. So lässt sich immer eindeutig ermitteln, ob der Taster gedrückt wird oder nicht.

Lass dich nicht verwirren, dass die Programmier-LED unten links auch leuchtet, solange du die Taste gedrückt hältst. Die LED und der Taster sind fix verbunden und haben nichts mit deinem Programm-Code zu tun.




```
void setup() {
  pinMode(0, INPUT);    // Pin 0 ist Tastereingang

  pinMode(14, OUTPUT);    // Pin 14 ist Ausgang
  pinMode(13, OUTPUT);    // Pin 13 ist Ausgang
}

void loop() {
  if (digitalRead(0)==LOW)
  {              // wenn Taste gedrückt wird

    digitalWrite(14,HIGH);     // LED an Pin 14 an
    digitalWrite(13,LOW);      // LED an Pin 13 aus

  }
  else{         // wenn Taste NICHT gedrückt wird
    digitalWrite(14,LOW);      // LED an Pin 14 aus
    digitalWrite(13,HIGH);     // LED an Pin 13 an
  }

  delay(100);           // warte für 100ms
}
```

Abb. 26: Übungsbeispiel: Taster und LED)

Solange der Taster nicht gedrückt wird, leuchtet die gelbe LED an GPIO13. Sobald die Taste gedrückt wird leuchtet die rote LED an GPIO14.

Der Befehl `delay(100)` am Ende der Loop-Schleife bewirkt, dass das Programm an dieser Stelle 100 Millisekunden lang wartet. Der Microcontroller kann in dieser Zeit "Pause" machen.

Und jetzt bist du wieder gefragt! Ändere den Code ein bisschen ab, um ihn besser zu verstehen. Lass die LED zum Beispiel der Reihe nach aufleuchten, sobald du den Taster drückst.

Achtung: Verwende keine externen Taster, die direkt mit 9V verbunden sind, da die GPIOs nur für 5V-Pegel ausgelegt sind. Wir empfehlen, den eingebauten Taster GPIO0 zu verwenden. **Bei Spannungen über 5V an den GPIOs kann es zur irreversiblen Beschädigung des IoT Bricks kommen!**

6.3 I²C-Bus

Der I²C-Bus ist eine serielle Schnittstelle, die mit zwei Leitungen auskommt. Der Taktleitung SCL (Serial Clock) und der Datenleitung SDA (Serial Data). Die Leitungen arbeiten bidirektional. Bei den Busteilnehmern unterscheidet man zwischen Master und Slave. Der IoT Brick ist in unserem Fall der Master und die anderen Bricks sind Slaves. Die Busteilnehmer werden über I²C-Adressen angesprochen, pro Bus sind 128 möglich. Dabei können einzelne Busteilnehmer auch mehrere Adressen belegen. Manche Busteilnehmer haben auf der Rückseite kleine DIP-Schalter, sodass man die Adressbereiche umschalten kann, wenn mehrere Busteilnehmer am gleichen Bus verwendet werden.

Der Bus kann prinzipiell mit unterschiedlichen Geschwindigkeiten betrieben werden:

Mode	Geschwindigkeit
Standard Mode (Sm)	0,1 Mbit/s
Fast Mode (Fm)	0,4 Mbit/s
High Speed Mode (HS-Mode)	1,0 Mbit/s
Ultra Fast-Mode (UFm)	5,0 Mbit/s

Viele Microcontroller beherrschen nur die ersten beiden Modi (zum Beispiel der Controller des Arduino Nano) und manche noch den dritten Mode. Das gleiche gilt für die Peripheriebausteine. Die Modi müssen natürlich zusammenpassen. Der Master, also meist der Microcontroller, gibt dabei auf der SCL-Leitung den Takt vor. Über die SDA-Leitung werden die eigentlichen Daten übertragen.

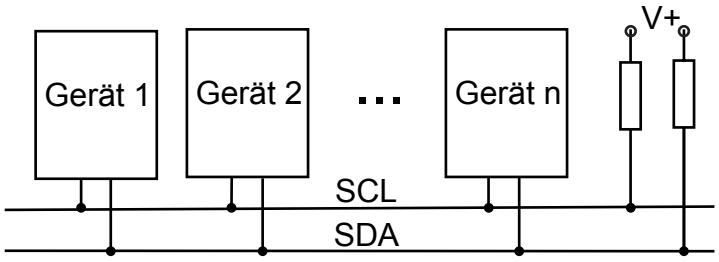


Abb. 27: I²C-Busstruktur

An einen I²C-Bus kann man maximal 128 Geräte anschließen, sofern jedes der Geräte nur eine Adresse belegt, ansonsten entsprechend weniger. Die Geräte sind über zwei Bus-Leitungen miteinander verbunden. Die beiden Pullup-Widerstände (im kΩ-Bereich) zur Versorgungsspannung sind im IoT Brick bereits eingebaut.

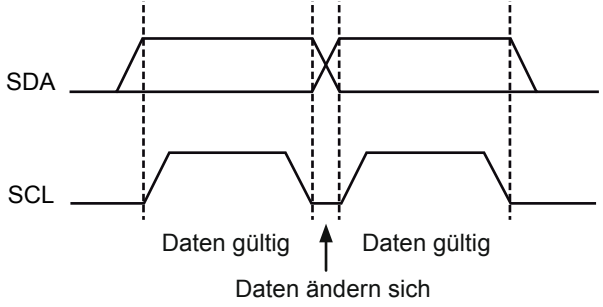


Abb. 28: Gültige Daten am I²C-Bus

Der Takt gibt dabei an, wann gültige Daten anliegen. In Abb. 28 sieht man, dass dies immer beim High-Pegel der SCL-Leitung der Fall ist. Der Empfänger kann jetzt die Daten abtasten und auswerten. Der Master gibt dabei den Takt vor, er legt dann entweder selbst Daten an oder erwartet solche vom entsprechenden Gerät.

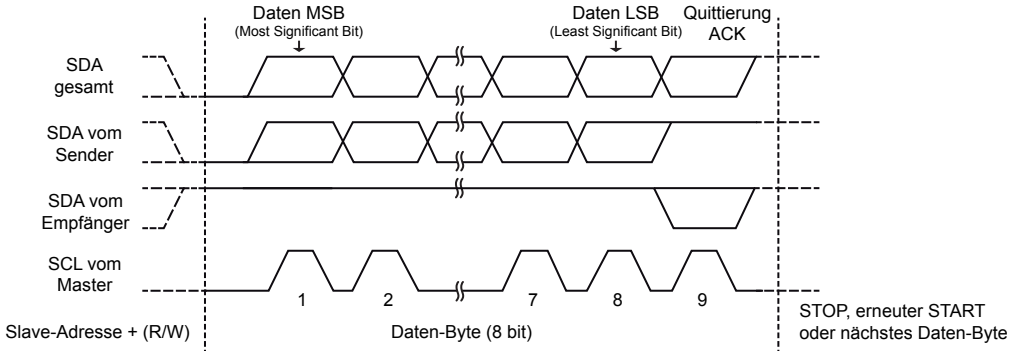


Abb. 29: Datentransfer am I^{2}C-Bus

In Abb. 29 sieht man den zeitlichen Verlauf eines Datentransfers mit folgenden Signalen (von unten nach oben): das Taktsignal SCL (vom Master vorgegeben), die Datenleitung aus Empfängersicht (Receiver) wird nicht aktiv angesteuert, da low-aktiv, die Datenbits wie vom Sender (Transmitter) losgeschickt (low-aktiv) und ganz oben SDA-Signal in der Gesamtschau. Wichtig ist die Synchronisation. Ein Empfänger (egal ob Master oder Slave) sendet am Ende eines jeden Datenpakets ein Quittierungs-Signal (ACK = Acknowlege), indem er die SDA-Leitung auf Low zieht. Da dies einem Wired-OR entspricht, reicht es, wenn ein Slave das ACK-Signal sendet.

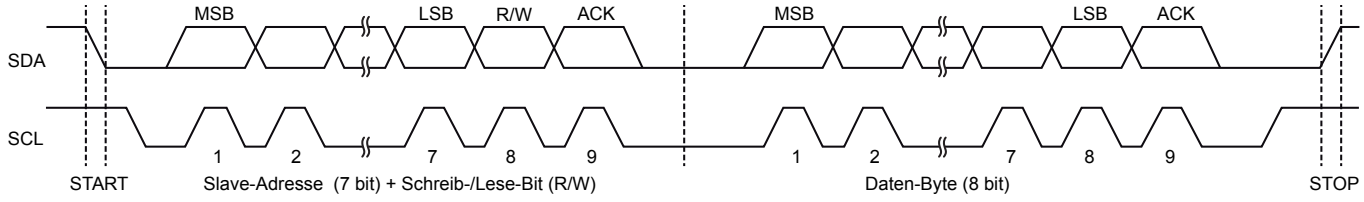


Abb. 30: Übertragungszyklus am I^{2}C-Bus

In Abb. 30 sieht man den gesamten Übertragungszyklus. Zunächst wird ein Paket mit der Adresse gesendet. Die Adresse besteht aus 7 Bits, ergänzt um ein weiteres, R/W (Read/Write) benanntes Bit. Alle Busteilnehmer vergleichen die ausgesendete Adresse mit ihrer eigenen. Bei Übereinstimmung quittiert der betreffende Slave mit einem ACK-Signal, indem er die SDA-Leitung kurzzeitig auf low legt. In Abhängigkeit vom R/W-Bit weiß der adressierte Slave, ob er nun einen Sende- oder Empfangszyklus starten soll. Danach kann der eigentliche Datentransfer beginnen. Als Abschluss wird ein Stop-Zyklus eingeleitet. Dazu wird der Takt auf high gesetzt und dann die SDA-Leitung freigegeben. Die Leitungen SDA und SCL sind nun beide auf High-Pegel, das bedeutet, dass der I^{2}C-Bus frei verwendbar ist. Theoretisch kann nun auch ein anderer Master (falls mehrere am Bus sind), einen neuen Zyklus starten.

Für uns ist der I^{2}C-Bus einfach nutzbar, da die Arduino-Bibliothek mehrere Befehle bereitstellt, um Bricks mit I^{2}C-Interface wie z. B. den 7-Segmentanzeige-Brick über den IoT Brick anzusteuern.

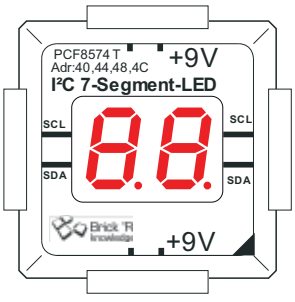
6.3.1 Die 7-Segmentanzeige

In den Anfängen der Computertechnik machte man sich Gedanken, wie man Ziffern darstellen kann. Am einfachsten waren damals 10 Lampen, die von 0 bis 9 beschriftet wurden. Später hat man dann die Lampen zum Beleuchten kleiner Glasplatten mit entsprechenden Bohrungen verwendet. Zur gleichen Zeit kamen die sogenannten Nixi-Röhren auf, die Ziffern waren aus Draht geformt, und bei Anlegen einer höheren Spannung unter Schutzgas-Atmosphäre, begannen die Ziffern zu leuchten. Dann kam man auf die Idee, die Zahlen in Segmente zu zerlegen. Mit sieben Segmenten kann man alle Ziffern zwischen 0 und 9 darstellen. Die ersten Anzeigen verwendeten noch Glühdrähte, aber mit dem Aufkommen der LEDs wurde es einfacher. Hinter jedem Segment verbirgt sich eine LED, die den Balken beleuchtet. Die einzelnen Segmente werden häufig mit a bis g bezeichnet. Zusätzlich gibt es noch eine einzelne LED für den Dezimalpunkt (dp).

Abb. 31: Die 7-Segmentanzeige

Die Ziffern 0 bis 9 kann man so auf einfache Weise darstellen. Mit dem OLED-Display (siehe Kap. 6.4 auf Seite 32) werden wir später noch eine elegantere Form der Anzeige kennenlernen, mit der man auch einfache Grafiken darstellen kann.

In unserem 7-Segmentanzeige-Brick sind zwei 7-Segmentanzeigen untergebracht, die über den I²C-Bus angebunden werden. Die linke Anzeige wird im Programm-Code auch als MSB-Zeichen (von Most Significant Bit) bezeichnet und die rechte Anzeige als LSB-Zeichen (von Least Significant Bit) – siehe auch Kap. 6.5.2.5 auf Seite 38. Die beiden 7-Segmentanzeigen werden über jeweils einen sog. I/O-Extender-Baustein vom Typ 8574T angesteuert. Dieser Baustein übernimmt die Adress-Dekodierung am I²C-Bus und die Dekodierung des Daten-Bytes (hier: Ziffer) zur Ansteuerung der sieben Segmente inkl. LED-Treiberstufe.



6.3.2 7-Segmentanzeige als I²C-Brick – Aufbau und Adressen

Öffne in der Arduino IDE den Sketch: `Beispiel_6.3.2.ino`. Folgende Headerdatei musst du einbinden: `Wire.h`. Falls du die entsprechenden Bibliotheken noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

Wir realisieren nun eine einfache Schaltung mit der 7-Segmentanzeige. Auf der Rückseite lässt sich mit zwei kleinen Schaltern die I²C-Adresse des Bricks einstellen (mögliche Adressen hexadezimal: $40_{(16)}$, $44_{(16)}$, $48_{(16)}$, $4C_{(16)}$). Damit sind maximal vier solcher Bricks an einem I²C-Bus einsetzbar.

Achtung: in der Regel ist die Adresse $40_{(16)}$ (siehe Abb. 32) voreingestellt. Unter Umständen muss die Einstellung kontrolliert (siehe auch Sketch `Beispiel_6.3.2.ino`) und angepasst werden. Die Schalter sind auf der Rückseite der Platine, und sind von außen über ein Loch auf der Unterseite zugänglich. Mit etwas Geschick kann man die kleinen Schiebeschalter z. B. mit einem Zahnstocher einstellen.

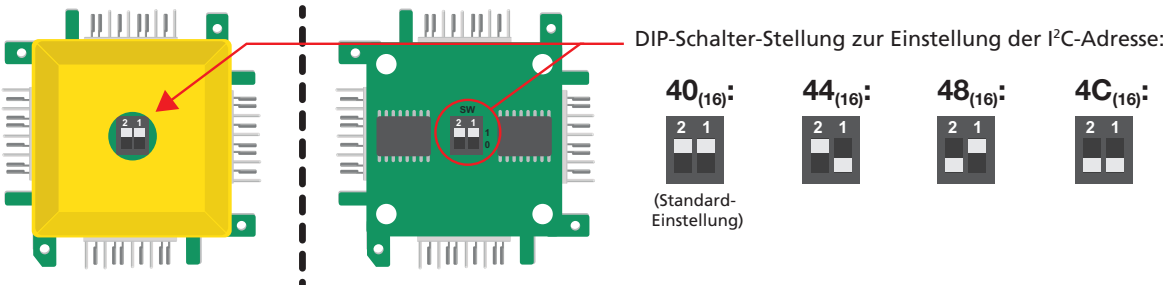


Abb. 32: Adress-Einstellung I²C-7-Segment-Brick

Beachte: Alle Teilnehmer am I²C-Bus müssen eine unterschiedliche Adresse verwenden. Ansonsten kommt es zu Fehlfunktionen!

Wir haben eine kleine Bibliothek für die Arduino-Entwicklungsumgebung vorbereitet, in der alle Segmente kodiert sind. Dies geschieht mit Hilfe einer sogenannten Zeichentabelle. Den Ziffern und sogar allen Buchstaben von A-Z wird mit einem Byte die Kombination der Segmente in einer Tabelle zugewiesen. Zur Vereinfachung haben wir dazu schon ein paar Unterprogramme vorbereitet.

Die Funktion `display_seg1x()` bringt ein einzelnes Segment zur Anzeige. Dazu übergibt man die I²C-Adresse des entsprechenden Treiberbausteins. Die Funktion `get_7seg()` übernimmt dabei die Umrechnung des ASCII-Codes in den Index der Tabelle mit den Segment-Zuordnungen. Mit `display_seg1x-bin()` können die Segmente bei Bedarf auch direkt angesteuert werden. Zwei solcher Treiberbausteine gibt es für die beiden Ziffern, die mit aufsteigender Adresse im Abstand 2 adressiert werden. Der Sketch verwendet standardmäßig für die niederwertige Ziffer (im Sketch auch "LSB Zeichen" genannt) die I²C-Adresse: $40_{(16)}$, und für die höherwertige Ziffer $42_{(16)}$ (im Sketch auch "MSB Zeichen" genannt).

Dazu am besten unser Programmbeispiel genauer studieren und einfach mal damit experimentieren.

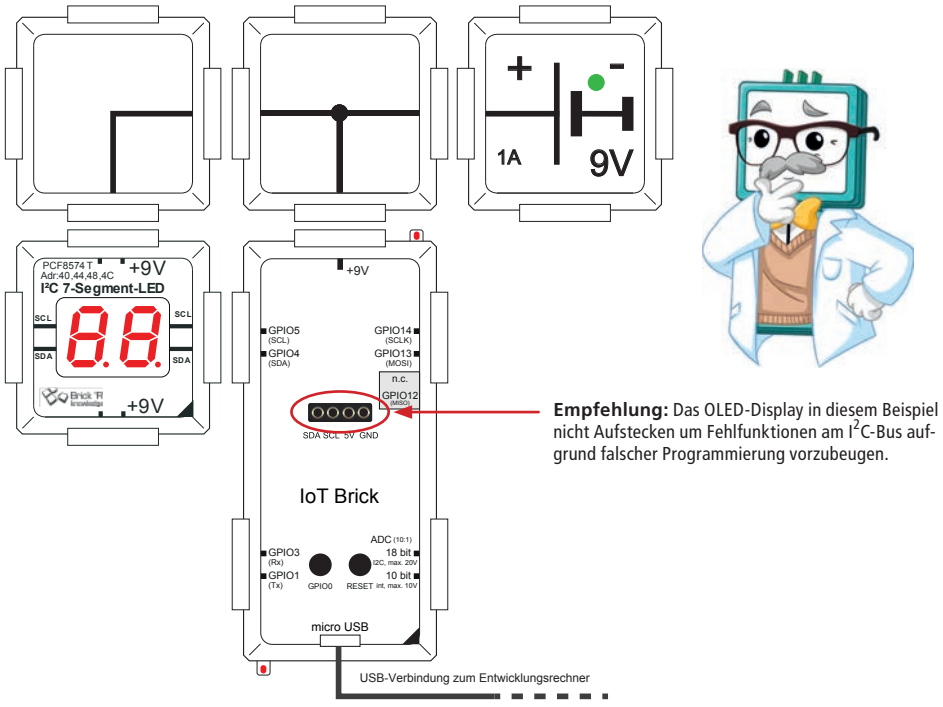


Abb. 33: Übungsbeispiel: I²C-Adresse des 7-Segment-Bricks anzeigen

Programmausschnitt

```
void loop() {
// 7-Segmentanzeige mit Treiberbaustein 8574T
// Alle potentiellen Adressen zur Identifikation ausgeben
// Man kann so sehen welche Adresse eingestellt ist
   display_seg1x(i2cseg7x2amsb1,'4'); // eigene Adresse
   display_seg1x(i2cseg7x2alsb1,'0'); // ausgeben
   display_seg1x(i2cseg7x2bmsb1,'4'); // sind immer PAARE
   display_seg1x(i2cseg7x2blsb1,'4'); // zwei Befehle fuer einen BRICK
   display_seg1x(i2cseg7x2cmsb1,'4');
   display_seg1x(i2cseg7x2clsb1,'8'); // von 0x40 bis 0x4C
   display_seg1x(i2cseg7x2dmsb1,'4');
   display_seg1x(i2cseg7x2dlsb1,'C');
}
```

Was passiert?

Das Beispielprogramm (siehe Sketch `Beispiel_6.3.2.ino`) zeigt die gefundene I²C-Adresse auf dem Display an, indem es die Adresse als Daten-Byte an die korrespondierende Adresse schickt. Dies wird mit allen sinnvollen Adressen ($40_{(16)}$, $44_{(16)}$, $48_{(16)}$, $4C_{(16)}$) wiederholt. Sobald die richtige Adresse gesendet wird, fühlt sich der Brick angesprochen und zeigt die entsprechende Adresse an. So kannst du den Wert leicht herausfinden und notieren.

6.3.3 7-Segmentanzeige als Zähler

 Öffne in der Arduino IDE den Sketch: `Beispiel_6.3.3.ino`. Folgende Headerdatei musst du einbinden: `Wire.h`. Falls du die entsprechenden Bibliotheken noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

In dem Beispiel wollen wir einen einfachen Zähler realisieren. Du kannst die gleiche Schaltung wie in Abb. 33 verwenden. In der Variable `counter` wird der Zählerstand gespeichert. Das Programm erhöht den Wert alle 500 ms um eins. Auf einer zweistelligen 7-Segmentanzeige kann man aber maxmial den Wert 99 darstellen. Deshalb wird in einer `if`-Abfrage der Wert von `counter` auf größer 99 abgefragt. Sobald dies der Fall ist, wird die Anzeige auf 00 zurückgesetzt. Die Variable `counter` zählt also von 0 bis 99 und beginnt dann wieder bei 0.

Programmausschnitt

```
void loop() { // In der Schleife

  char buffer[10];             // Zeichenpuffer definierter Groesse verwenden

  static int counter = 0;     // Zaehlervariable auf 0 setzen

  sprintf(buffer,"%02d",counter++); // Umrechnen Integer auf Zeichen

  if (counter >99) counter = 0;  // Zaehler soll zwischen 0..99 laufen

     // Counter ausgeben als zwei Ziffern, daher Buffer 0 und 1
     display_seg1x(i2cseg7x2alsb1,buffer[1]); // LSB Zeichen auf Adresse 0x40
     display_seg1x(i2cseg7x2amsb1,buffer[0]); // MSB Zeichen auf Adresse 0x42

     delay(500); // ca. alle 500ms hochzaehlen.

} // Ende der Schleife
```

Der Sketch verwendet standardmäßig für die niederwertige Ziffer (im Sketch auch "LSB Zeichen" genannt) die I²C-Adresse: $40_{(16)}$, und für die höherwertige Ziffer $42_{(16)}$ (im Sketch auch "MSB Zeichen" genannt).

Zu beachten ist, dass die Schleife etwas länger als 500 ms braucht, denn zu der Verzögerung durch den Befehl `delay(500)` muss man strenggenommen noch die Ausführungszeiten der anderen Befehle addieren. Will man präziser arbeiten, muss man Timer verwenden und diesen abfragen.

6.3.4 7-Segmentanzeige mit Tastenentprellung

Öffne in der Arduino IDE den Sketch: `Beispiel_6.3.4.ino`. Folgende Headerdatei musst du einbinden: `Wire.h`. Falls du die entsprechenden Bibliotheken noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

Taster und Schalter haben den Nachteil, dass bei Betätigung der mechanische Kontakt (oft ist eine Feder im Spiel) ein mehrfaches Schließen bzw. Öffnen verursacht. Diesen Störeffekt nennt man in der Digitaltechnik auch "Prellen". Dieses Problem kann auf Hardware-Ebene mit einem einfachen RS-Flipflop gelöst werden (mehr zu diesem Thema erfährst du im Brick'R'knowledge Logic Set). In dieser Übung lernst du als Alternative die Tastenentprellung per Software kennen. Grundidee ist, mit sog. Delays eine Wartezeit in der Software einzubauen, die mindestens solange wie ein Prellzyklus dauert.

Zur Demonstration verwenden wir wieder unseren Zähler, den du schon aus Übungsbeispiel 6.3.3 kennst.

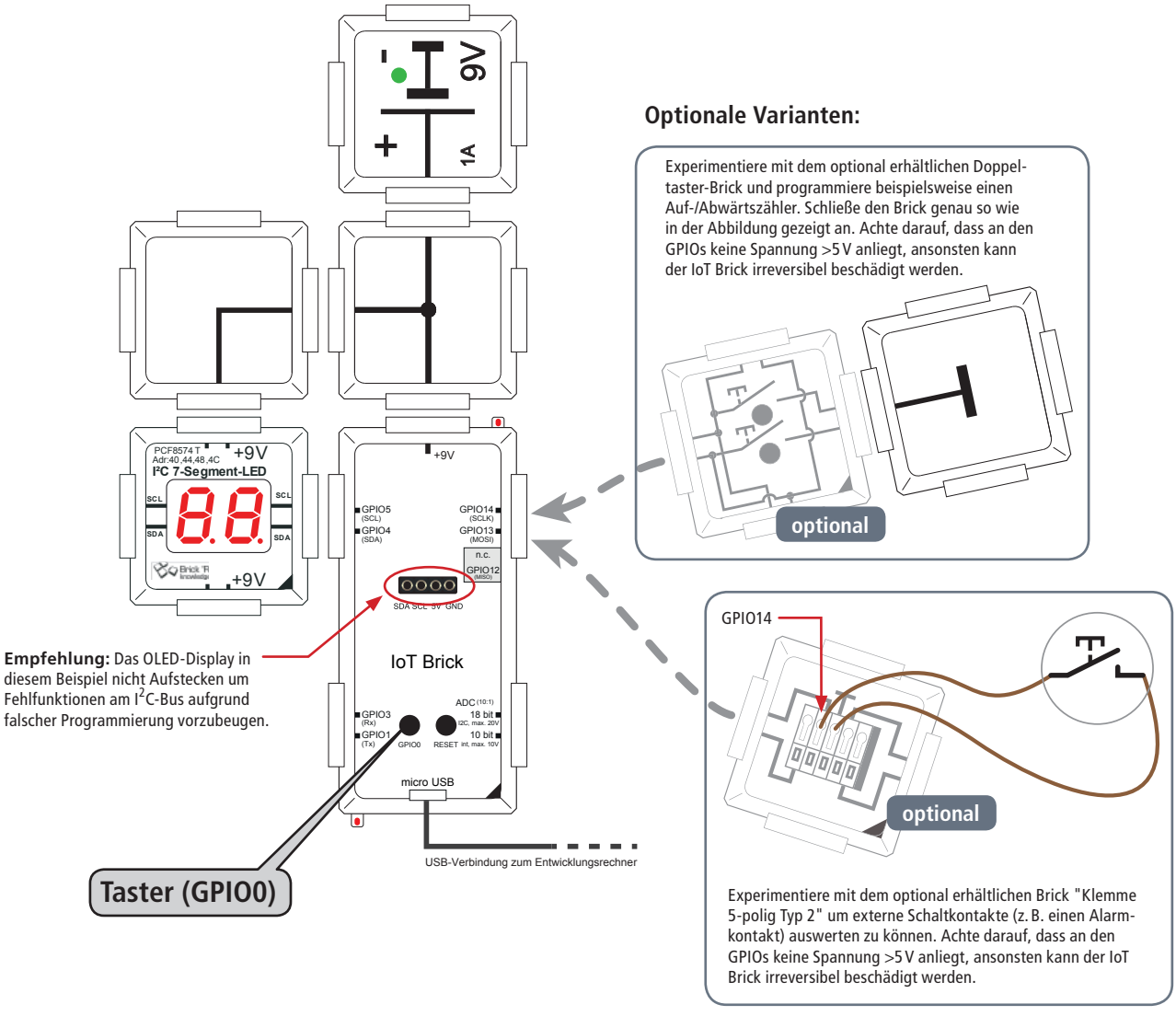


Abb. 34: Zähler mit Tastenentprellung

Achtung: Verwende keine externen Taster, die direkt mit 9 V verbunden sind, da die GPIOs nur für 5 V-Pegel ausgelegt sind. Wir empfehlen den eingebauten Taster GPIO0 zu verwenden. **Bei Spannungen über 5 V an den GPIOs kann es zur irreversiblen Beschädigung des IoT Bricks kommen!**

Teste unterschiedliche Werte im `delay`-Befehl, um eine möglichst kurze aber dennoch zuverlässig entprellende Wartezeit zu ermitteln. Experimentiere auch mit optional erhältlichen Bricks wie den in Abb. 34 gezeigten Bricks "Doppeltaster" und "Klemme 5-polig Typ 2". Achte darauf, die als Eingang konfigurierten GPIOs nur gegen Masse zu schalten. Solange der Taster bzw. Schalter offen ist, liegt am Pin High-Pegel, da intern Pullup-Widerstände bestückt sind (siehe Kap. 5.1.3 auf Seite 14).

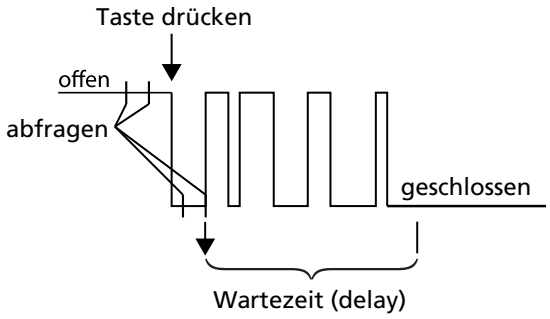


Abb. 35: Tastenentprellung per Software

Algorithmus zum Entprellen

Die Taste verhält sich low-aktiv, d. h. durch Drücken wird ein Low-Pegel an den Eingang (im Beispiel GPIO0) angelegt. Das Signal wird auf den ersten Übergang von high nach low abgefragt (if-Abfrage), danach wird gewartet, bis es auf high geht (die Taste also vermeintlich losgelassen wird) – die `while`-Schleife wird verlassen. Ab jetzt läuft die Verzögerungszeit (hier 40 ms), um schließlich den neuen gültigen Zählerstand auszugeben. Nun wird auf den nächsten Tastendruck gewartet also den High-zu-Low-Übergang. Beachte, dass die Dauer der Verzögerung von der verwendeten Tastenart abhängig ist. Der optimale Wert muss ggf. experimentell ermittelt werden um zu verhindern, dass Prellimpulse mit ausgewertet werden.

Programmausschnitt

```
void loop() { // Schleife

  char buffer[10];                    // Zeichenpuffer definierter Groesse verwenden

  static int counter = 0;            // Zaehlervariable auf 0 setzen

  sprintf(buffer,"%02d",counter);    // Umrechnen Integer auf Zeichen

  // Taste abfragen, kann prellen
  if (digitalRead(0)==LOW) { // Pruefen auf High->low
    // Dann noch den den Low-High Übergang abwarten
    // Koennte man auch nach dem Delay machen, ist aber nicht kritisch.

    while (digitalRead(0)==LOW) {
     // warten bis Taste losgelassen bzw prellt.
    }
    counter++; // den Counter hochzaehlen
    delay(40); // dann warten damit die Abfrage nicht zu schnell erneut erfolgt
  } // Ende der if-Abfrage High->Low

  if (counter > 99) counter = 0; // Zaehler soll zwischen 0..99 laufen

  // Counter ausgeben als zwei Ziffern, daher Buffer 0 und 1
     display_seg1x(i2cseg7x2alsb1,buffer[1]); // LSB Zeichen auf Adresse 0x40
     display_seg1x(i2cseg7x2amsb1,buffer[0]); // MSB Zeichen auf Adresse 0x42

} // Ende der Schleife
```

Was passiert?

Mit jedem Tastendruck sollte die Ausgabe auf der Anzeige um genau "1" hochzählen also 00, 01, 02 .... 99. Danach springt die Anzeige wieder auf 00.

Lass dich nicht verwirren, dass die Programmier-LED unten links auch leuchtet solange du die Taste GPIO0 gedrückt hältst. Die LED und der Taster sind fix verbunden und haben nichts mit deinem Programm-Code zu tun.

6.4 OLED-Display – Grundlagen

7-Segmentanzeigen werden normalerweise nur für Ziffern verwendet und ganz eingeschränkt auch für Buchstaben. Mit 14- und 16-Segmentanzeigen lassen sich auch Buchstaben brauchbar darstellen. Danach gab es dann die ersten Rasteranzeigen, mit einer Matrix von 5x7 Punkten konnte man Schriften schon viel besser darstellen, aber auch erste graphische Symbole waren hiermit möglich. Die Anzeigen basierten zunächst auf LEDs, dann kamen LCDs (Liquid Crystal Displays) auf den Markt und neuerdings auch OLEDs (Organic Light Emitting Diodes). Diese sind wieder den LEDs ähnlicher, da sie selbst leuchten können. Unser Set enthält ein monochromes OLED-Display mit 128x64 Pixel, mit dem sich nicht nur einzelne Zeichen sondern auch mehrzeiliger Text und einfache Grafiken darstellen lassen. Damit man Zeichen in dieser Weise auf dem Display darstellen kann, benötigt man einen Zeichengenerator, bzw. Tabelle, so ähnlich wie die der 7-Segmentanzeige. Der numerische Code wird für die Segmente in den Zustand an oder aus übersetzt. Der Zeichengenerator oder die Zeichensatztabelle benötigt ungleich mehr Speicher. Der Wert hängt von der Zahl der Pixel je Zeichen ab. Bei den kleinsten mit ca. 5x7 Pixel, werden etwa 5 Bytes pro Zeichen benötigt. Wenn man damit den ganzen ASCII-Satz (128 Zeichen inkl. Lücken) darstellen möchte, braucht man 128 x 5 Bytes = 640 Bytes.

Das mitgelieferte OLED-Display steckst du mit dem 4-poligen Stiftstecker, wie in der Graphik zu sehen, auf den IoT Brick.

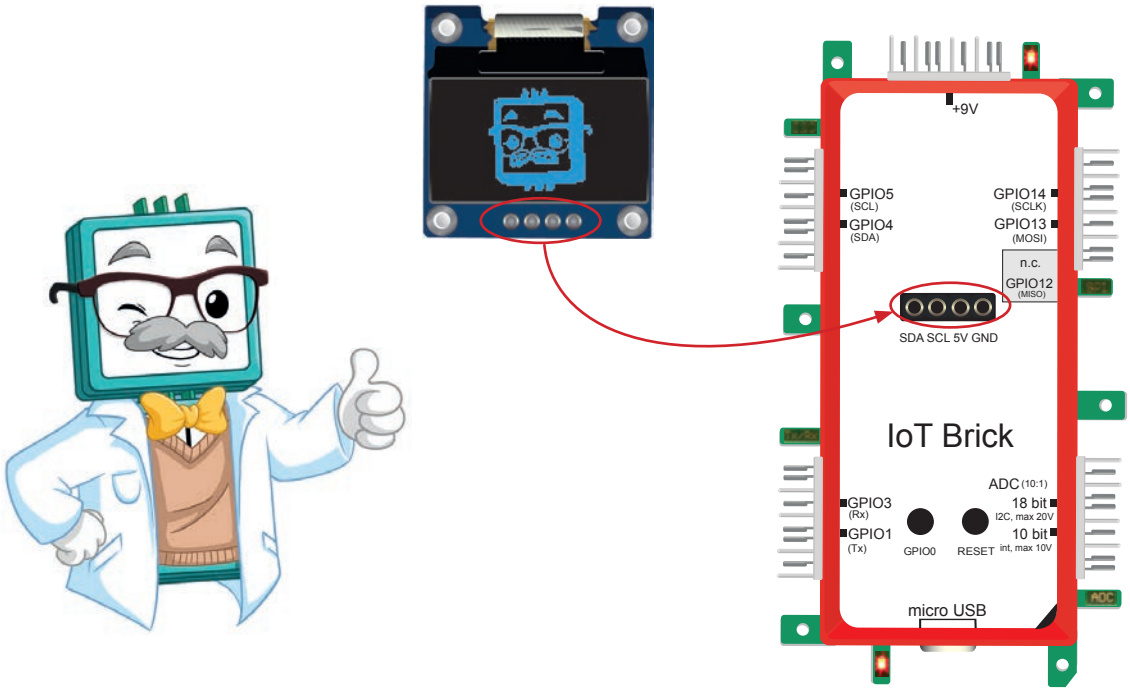


Abb. 36: OLED-Display auf IoT Brick

Öffne in der Arduino IDE den Sketch: `Beispiel_6.4.ino`. Folgende Headerdateien musst du einbinden: `#include <SSD1306Wire.h>` und `#include "images.h"`. Die Datei `images.h` muss sich im Projektverzeichnis dieses Sketches befinden. Falls du die Bibliothek `SSD1306Wire.h` noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

Um die Ansteuerung des OLED-Displays zu vereinfachen, verwenden wir hier eine fertige Bibliothek. Falls du diese noch nicht installiert hast, gehe zu Kap. 5 auf Seite 13 und hole dies jetzt nach.

Die Bibliothek zur Ansteuerung unseres OLED-Displays wird mit `#include "SSD1306Wire.h"` eingebunden und mit `SSD1306Wire display(0x3c, 4, 5);` initialisiert. D.h. um die richtige Adresse für die Funktion `display()` zu erhalten, musst du die Hardware-I^{2}C-Adresse unseres OLED-Displays (Standard: 0x78) um ein Bit nach rechts schieben. Der Programmierer schreibt dafür: `(0x78>>1)` – das Ergebnis ist "0x3c". Die mit den Werten 4 und 5 belegten Parameter geben die GPIOs für den I^{2}C-Bus an.

Um einen ersten Eindruck zu bekommen, kannst du diesen Sketch in deinen IoT Brick laden. Er zeigt was alles möglich ist, von verschiedenen Schriftgrößen über Graphiken bis zum hochlaufenden Fortschrittsbalken.

Schau dir dieses und die anderen Beispiele in Ruhe an. Lass dich nicht verwirren, in den Beispielen ist auch viel Code dabei, der für uns irrelevant ist. Du wirst sehen, im folgenden OLED-Beispiel ist es im Grunde ganz einfach einen Text auf dem Display anzeigen zu lassen.

Weitere Beispiele zum OLED-Display findest du in der Arduino IDE unter:

"Datei – Beispiele – ESP8266 Oled Driver for SSD1306 display – ..."

Damit diese Code-Beispiele mit dem IoT Brick richtig funktionieren, musst du folgende Code-Zeilen überprüfen:

- Die Zuweisung der I²C-Pins für das OLED muss lauten: `SSD1306Wire display(0x3c, 4, 5);` oder: `SSD1306 display(0x3c, 4, 5);` je nach `#include` Anweisung am Beginn des Sketch.
- Auskommentierung der Zeile mit dem Befehl `display.flipScreenVertically();` – es müssen zwei Schrägstriche `//` vorangestellt werden, ansonsten steht die Anzeige auf dem Kopf.

6.4.1 OLED-Display – Text anzeigen

Öffne in der Arduino IDE den Sketch: `Beispiel_6.4.1.ino`. Folgende Header-Datei musst du einbinden: `SSD1306Wire.h`. Falls du die entsprechende Bibliothek noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

In diesem einfachen Beispiel wollen wir unter Verwendung der OLED-Bibliothek `SSD1306Wire.h` einen Text auf dem OLED-Display ausgeben.

Du kannst dabei folgende Parameter festlegen:

- **Koordinaten**
  Ausgehend von der hier definierten Koordinate (*x*,*y*) kannst du mit dem Befehl… `display.drawString(x, y, "Beispieltext");` den Bezugspunkt des Textes und dessen Inhalt bestimmen.
- **Ausrichtung**
  Ausgehend von dem zuvor definierten Bezugspunkt kannst du mit dem Befehl… `display.setTextAlignment(TEXT_ALIGN_x);` bestimmen, ob der Text links (TEXT_ALIGN_ LEFT), zentriert (TEXT_ALIGN_CENTER) oder rechts (TEXT_ALIGN_ RIGHT) ausgerichtet werden soll.
- **Textgröße**
  Die drei Standard-Schriftgrößen, die von dieser Bibliothek nativ unterstützt werden sind 10, 16 und 24 Pixel. Du legst die Größe einfach mit dem Befehl `display.setFont(ArialMT_Plain_X);` fest.

Stell dir das Display als Koordinatensystem vor, in dem sich ganz oben links der Ursprung 0,0 befindet.

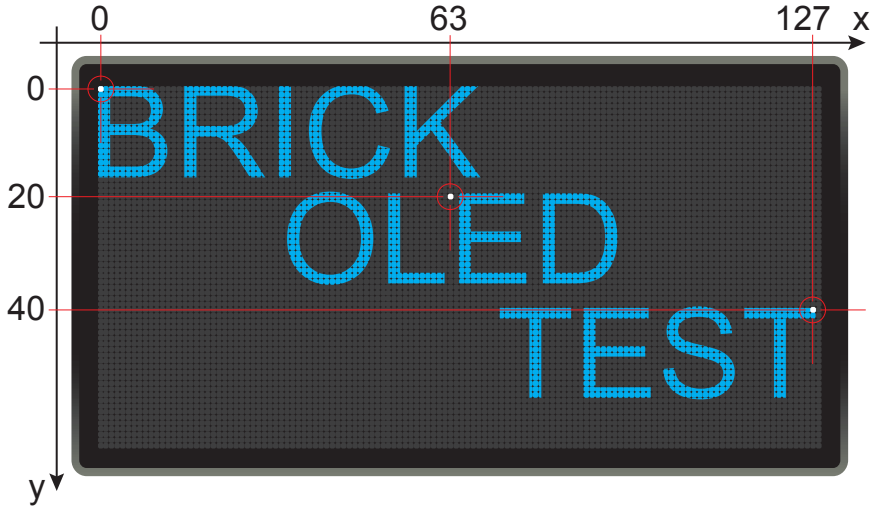


Abb. 37: Das Koordinatensytem der OLED-Matrix

Programmausschnitt

```
void loop() {

  display.clear();                                    // Displayinhalt löschen

  display.setTextAlignment(TEXT_ALIGN_LEFT);          // Text links ausrichten
  display.setFont(ArialMT_Plain_16);                  // Schriftgroeße 16 Pixel
  display.drawString(0, 0, "BRICK");                  // Position 0,0  Text: "BRICK"

  display.setTextAlignment(TEXT_ALIGN_CENTER);        // Text zentriert ausrichten
  display.setFont(ArialMT_Plain_16);                  // Schriftgroeße 16 Pixel
  display.drawString(63, 20, "OLED");                 // Position 63,20  Text: "OLED"

  display.setTextAlignment(TEXT_ALIGN_RIGHT);         // Text rechts ausrichten
  display.setFont(ArialMT_Plain_16);                  // Schriftgroeße 16 Pixel
  display.drawString(127, 40, "TEST");                // Position 127,40  Text: "TEST"

  display.display();                                  // Ausgabe auf Display

  delay(1000);
}
```

Wenn du eine andere Größe oder sogar Schriftart verwenden willst, bietet der Ersteller der OLED-Bibliothek unter: http://oleddisplay.squix.ch/#/home einen Online-Schriftsatz-Editor an.

Nachdem du den Text mit Position, Ausrichtung, Größe und Inhalt festgelegt hast, kannst du ihn mit dem Befehl `display.display();` auf dem OLED anzeigen lassen.

Und jetzt tob´ dich aus! Schreibe verschiedene Texte, in verschiedenen Größen an gewünschte Positionen.

6.5 Analogeingänge

6.5.1 A/D-Wandlung – Grundlagen

A/D-Wandlung steht für Analog-zu-Digital-Wandlung. Ziel ist es, analoge Werte, wie zum Beispiel eine unbekannte Spannung zu messen und in einen digitalen Wert umzusetzen. Diesen Wert kann der Computer dann weiterverarbeiten. Mit analogen Werten kann ein normaler Computer nichts anfangen.

Zwei wichtige Schritte werden dabei durchgeführt, eine Quantisierung der Amplitude z. B. der Spannung und eine Quantisierung der Zeit bei einem zeitlich veränderlichen Verlauf des analogen Wertes.

Was ist damit gemeint?

Die Quantisierung in der Amplitude kann man leicht verstehen. Angenommen eine analoge Spannung kann jeden beliebigen Wert zwischen 0 und 5 V einnehmen. Also 2,3 V oder 2,31 V oder 2,315 V ... usw. Es stellt sich also die Frage, wie genau möchte ich messen und wie fein, d. h. in wievielen Stufen möchte ich mein Signal auflösen? Die Zahlenwerte muss man schließlich für die Verarbeitung in einem digitalen Rechensystem aufbereiten.

Beispiel:

Wir wollen einen Spannungsbereich von 0 bis 5 V in 6 Stufen digital darstellen. Welchen digitalen Wert wird man dann für 2,1 V vergeben? Im Diagramm unten kann man die Zuordnung an den roten Punkten ablesen. Der Wert 2,1 liegt näher bei 2 als bei 3, also wird man den digitalen Wert 2 wählen. Bei einem Wert von 2,5 kann man 3 als digitalen Wert zuordnen, wenn man die Zahl 2,5 kaufmännisch rundet.

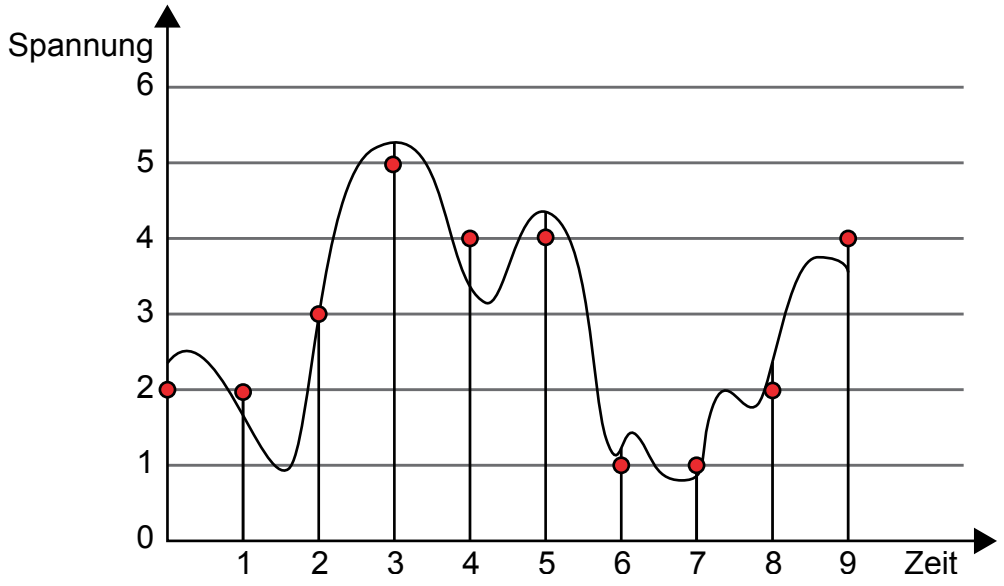


Abb. 38: Abtastung Analogsignal

In obiger Abbildung haben wir den zeitlichen Verlauf eines Spannungssignals aufgezeichnet (in der x-Achse sehen wir die Zeit in Sekunden und in der y-Achse die Spannung in Volt).

Bei der Umwandlung des Signals in eine digitale Zahlenfolge, wird man einmal pro Sekunde einen Messwert abfragen (senkrechte Linien), und dann die Rundung auf eine Stelle durchführen. Damit ergibt sich die folgende Messreihe:

Zeit	0 s	1 s	2 s	3 s	4 s	5 s	6 s	7 s	8 s	9 s
Spannung	2 V	2 V	3 V	5 V	4 V	4 V	1 V	1 V	2 V	4 V

Es gibt dabei zwei interessante Effekte:

1. Wir verlieren Information in der Amplitude, da die kleinste detektierbare Spannungseinheit in unserem Beispiel mit 1 V angenommen wird. Diesen Effekt nennt man auch Quantisierungsfehler, der in unserem Fall bis zu 0,5 V Abweichung vom wahren Wert in positiver oder negativer Richtung betragen kann. Mit einer höheren Auflösung der Wandlung, würden wir auch mehr Information über das ursprüngliche Signal erhalten.
2. Wir verlieren auch zeitlich relevante Informationen. So sehen wir im Bereich zwischen 1 und 2 Sekunden einen Spannungsabfall bis 1 V, der in der Messreihe nicht sichtbar ist. Die Fachleute werden nun anmerken, dass das sogenannte Nyquist-Shannon-Abtasttheorem verletzt wurde. Dieses besagt, dass die Abtastrate für ein periodisches Signal mindestens doppelt so hoch sein muss, wie dessen maximaler Frequenzanteil (auch Oversampling genannt). Dieses Kriterium wird bei den kürzeren Über- und Unterschwingern verletzt.

Wenn man nun die roten Punkte verbindet, erhält man das für den Computer "sichtbare" Signal, das in einem Zeitraster von 1 Sekunde abgetastet wurde. Die ursprüngliche Kurve ist nicht mehr genau rekonstruierbar. Je mehr man jedoch die Zahl der Abtastpunkte erhöht, desto besser kann das Signal rekonstruiert werden (siehe Abtasttheorem).

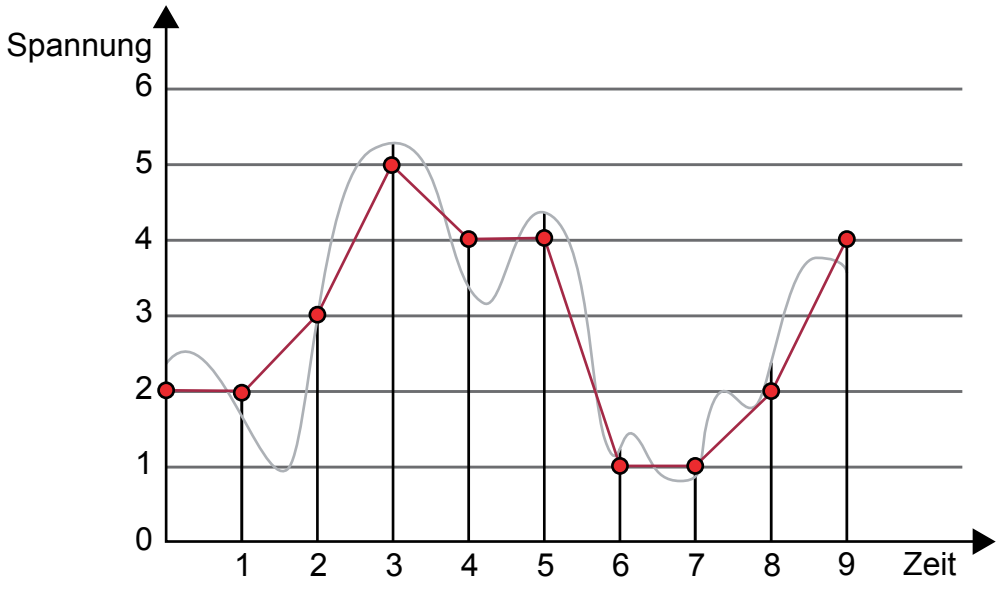


Abb. 39: Rekonstruktion des abgetasteten Signals

Die Auflösung der Amplitude wird durch die Anzahl der diskreten Stufen, die einem Zahlenwert zugeordnet werden, bestimmt. In unserem Beispiel unterscheiden wir 6 Spannungsstufen. Zum Vergleich: ein 10 bit A/D-Wandler stellt bereits 1024 Stufen zur Verfügung, die in der Regel binär codiert werden (siehe auch Kap. 6.5.2.5 auf Seite 38). Im IoT Brick haben wir die Auswahl zwischen einem 10 bit und einem 18 bit A/D-Wandler (im Englischen übrigens "analog digital converter" genannt, abgekürzt: ADC).

10 bit und 18 bit A/D-Wandlung im Vergleich

	10 bit A/D-Wandler	18 bit A/D-Wandler
Anzahl der Stufen (Auflösung)	2^{10} = 1024	2^{18} = 262144
Messbereich und korrespondierender Digitalwert (dezimal)	0…9,99 V $0…1023_{(10)}$	0… 19,999924 V $0…262143_{(10)}$
Kleinste Spannungsstufe (sog. "Least Significant Bit" (LSB))	10 V / 1024 = 0,0098 V (= 9,8 mV)	20 V / 262144 = 0,000076 V (= 76 µV)
Quantisierungsfehler	±4,9 mV	±38 µV
Abtastrate max.	200 Samples/Sekunde (= 200 S/s)	3,75 Samples/Sekunde (= 3,75 S/s)
Wandlertyp	SAR-Wandler	Delta-Sigma-Wandler (ΔΣ-Wandler)

Zur Umwandlung der analogen Signale in digitale, binär codierte Werte, gibt es unterschiedliche Wandlungsverfahren, die den Rahmen dieses Handbuchs sprengen würden. Als Stichwort für die eigene Recherche möchten wir hier nur die wichtigsten Verfahren nennen:

- **Integrierende Wandler (Zählverfahren)**
  Langsamer als Wägeverfahren, störunempfindlich, geringer Hardwareaufwand, Realisierung: Single-, Dual- und Quad-Slope-Wandler, Anwendungsbeispiel: Multimeter
- **Rückgekoppelte Wandler (Wägeverfahren)**
  Guter Kompromiss zwischen Geschwindigkeit und Hardwareaufwand, Realisierung:
  - Delta-Sigma-Wandler (ΔΣ-Wandler), Anwendungsbeispiel: 1-Bit A/D-Wandler in der Audiotechnik
  - SAR (Successive Approximation Register)-Wandler, Anwendungsbeispiel: Messtechnik mit sehr hoher Auflösung.
- **Parallel-Wandler (Flash- und Pipeline-Wandler)**
  Sehr schnell, sehr teuer, Realisierung: Flash- und Pipeline-Wandler, Anwendungsbeispiel: Radartechnik.

6.5.2 Die A/D-Wandler auf dem IoT Brick

6.5.2.1 Der 10 bit A/D-Wandler

Der ESP8266 bietet einen integrierten A/D-Wandler (A/D = Analog-zu-Digital) mit einer Auflösung von 10 bit, was einer Auflösung des Spannungswertes von 2^{10} = 1024 Stufen entspricht. Der Eingangsspannungsbereich des Wandlers selbst geht von 0 V bis 1 V. Zwecks besserer Praxistauglichkeit wurde am Brick-Eingang "ADC 10 bit" ein 10-zu-1 (10 : 1)-Spannungsteiler vorgeschaltet, um so eine Spannung von 0 V bis 10 V direkt messen zu können. Damit kannst du ein einfaches Voltmeter bauen, mit dem du Gleichspannungen von 0 V bis 10 V messen kannst.

6.5.2.2 Der 18 bit A/D-Wandler

Für weitergehende Experimente wurde im IoT Brick zusätzlich ein sehr präziser A/D-Wandler vom Typ MCP3421 verbaut, der mit 18 bit (2^{18} = 262.144 Stufen) sehr hoch auflöst. Dieser Wandler-Baustein ist über den I^2C-Bus angebunden und bietet einen Eingangsspannungsbereich von 0 V bis 2 V mit vorgeschaltetem 10-zu-1-Spannungswandler. Daraus resultiert ein praxistauglicher Eingangsspannungsbereich von 0 V bis 20 V, d. h. am Brick-Eingang "ADC 18 bit" darf man eine Eingangsspannung von max. 20 V angelegen.

6.5.2.3 Der Spannungsteiler

An beiden Spannungswandler-Eingängen "ADC 10 bit" und "ADC 18 bit" ist im IoT Brick ein Spannungsteiler mit einem Teilerverhältnis von 10-zu-1 eingebaut:

Achte darauf, dass am Eingang "ADC 10 bit" nie mehr als 10 V und am Eingang "ADC 18 bit" nie mehr als 20 V anliegen. Ansonsten kann der A/D-Wandler beschädigt werden!

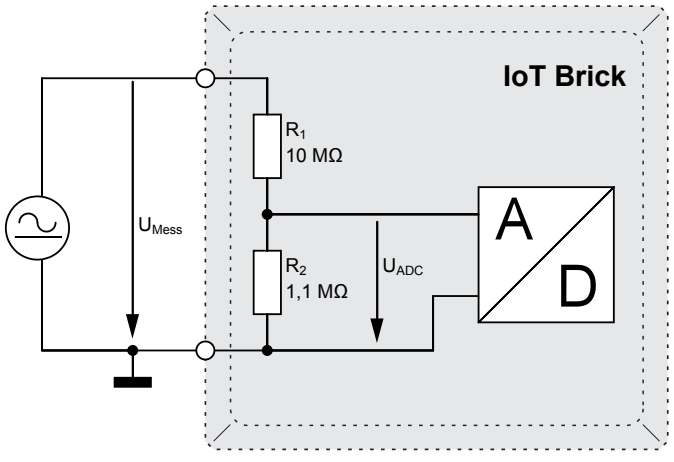


Abb. 40: Spannungsteiler (10 1)

Das Teilerverhältnis berechnet sich wie folgt: $\frac{U_{Mess}}{U_{ADC}} = \frac{R_1 + R_2}{R_1} = \frac{11{,}1\ \text{MΩ}}{1{,}1\ \text{MΩ}} = 10{,}09 \approx 10$

6.5.2.4 Praxistipp: Korrekturfaktor

Bedenke, dass es aufgrund von Toleranzen der internen Widerstandsvorteiler an den Eingängen "ADC 10 bit" und "ADC 18 bit" zu Ungenauigkeiten im Messergebnis kommen kann. Diese Abweichung kann durch einen Korrekturfaktor abgeglichen werden. Zur Bestimmung des Korrekturfaktors kannst du z. B. ein genaues Multimeter verwenden und die Spannung U_{Mess} überprüfen. Danach berechnest du den Korrekturfaktor aus dem Verhältnis der mit dem Multimeter gemessenen Spannung und der am IoT Brick angezeigten Spannung **ohne Korrektur!** Im Beispielprogramm haben wir hierzu entsprechenden Code vorbereitet, den du bei Bedarf nutzen kannst (entferne die beiden Schrägstriche (Kommentar) erst nachdem du die Messung ohne Korrektur gemacht hast). Der Korrekturfaktor wird zwecks besserer Übersichtlichkeit am Anfang des Programms als Konstante definiert (siehe folgender Programmausschnitt für den 10 bit A/D-Wandler).

$$Korrekturfaktor = U_{Mess,\ Multimeter} / U_{Anzeige\ Brick}$$

Programmausschnitt für Korrekturfaktor (siehe Beispiel_6.5.3.ino)

```
//Vorgehensweise gilt fuer 10bit und 18bit ADC gleichermaßen
...
//hier Korrketurfaktor für 10bit ADC definieren
const float Correction_10bit = 1.046;
...
// Korrekturfaktor einrechnen (mit Multimeter-Messung ermittelt)
UMess_10bit = UMess_10bit * Correction_10bit;
...
```

Da jeder Widerstand unterschiedliche Fertigungstoleranzen hat sollte auch der Korrekturfaktor für die beiden A/D-Wandler getrennt bestimmt werden.

6.5.2.5 Binäre Codierung

Ein A/D-Wandler konvertiert einen analogen Messwert am Eingang in einen binär codierten Wert am Ausgang. Man spricht auch von Binärwort dessen Breite, also die Anzahl einzelner Binärzeichen in Bits angegeben wird (als Einheit übrigens klein geschreiben: 1 bit). In unserem Fall haben wir es je nach A/D-Wandler-Auflösung mit einem 10 bit oder 18 bit Binärwort zu tun. Das niederwertigste Bit wird dabei oft auch als "Least Significant Bit" (LSB) und das höchstwertige Bit als "Most Significant Bit" (MSB) bezeichnet.

Höchstwertiges Bit (Most Significant Bit (MSB)) — Niederwertigstes Bit (Least Significant Bit (LSB))

Anzahl der Bits	Anzahl der codierbaren Zustände	Binärwort ≙ Binärzahl (Anzahl der codierbaren Zustände - 1)	Dezimalzahl	Hexadezimalzahl
		2^{17} … Wertigkeit … 2^0		
(0)	$1 = 2^0$	00 0000 0000 0000 0000	0	0
1 bit	$2 = 2^1$	00 0000 0000 0000 0001	1	1
2 bit	$4 = 2^2$	00 0000 0000 0000 0011	3	3
3 bit	$8 = 2^3$	00 0000 0000 0000 0111	7	7
4 bit	$16 = 2^4$	00 0000 0000 0000 1111	15	F
…	…	…		
8 bit	$256 = 2^8$	00 0000 0000 1111 1111	255	FF
10 bit	$1024 = 2^{10}$	00 0000 0011 1111 1111	1023	3FF
12 bit	$4096 = 2^{12}$	00 0000 1111 1111 1111	4095	FFF
14 bit	$16.384 = 2^{14}$	00 0011 1111 1111 1111	16.383	3FFF
16 bit	$65.536 = 2^{16}$	00 1111 1111 1111 1111	65.535	FFFF
18 bit	$262.144 = 2^{18}$	11 1111 1111 1111 1111	262.143	3FFFF

6.5.3 A/D-Wandler 10 bit

Öffne in der Arduino IDE den Sketch: `Beispiel_6.5.3.ino`. Folgende Headerdateien musst du einbinden: `SSD1306Wire.h`. Falls du die entsprechenden Bibliotheken noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

In diesem Beispiel messen wir eine Spannung über einem variablen Widerstand (Potentiometer, oft nur Poti genannt). Der Widerstandswert des Potis variiert je nach Stellung zwischen 0 Ω und 10 kΩ. Die Spannung, die am Poti abfällt, verhält sich linear zum Widerstandswert. Wenn der Poti im Uhrzeigersinn am Anschlag steht, beträgt der Widerstandswert ca. 10 kΩ zwischen Masse und dem Schleifkontakt, welcher am Analogeingang (ADC 10 bit) des IoT Bricks angeschlossen ist. Jetzt liegt also die volle Versorgungsspannung an, welche etwa 9 V betragen sollte. Drehen wir nun den Poti gegen den Uhrzeigersinn auf Anschlag, dann liegt 0 V (Masse) am Eingang des A/D-Wandlers. In der Mittelstellung des Potis wird in etwa die halbe Versorgungsspannung anliegen, d. h. ca. 4,5 V.

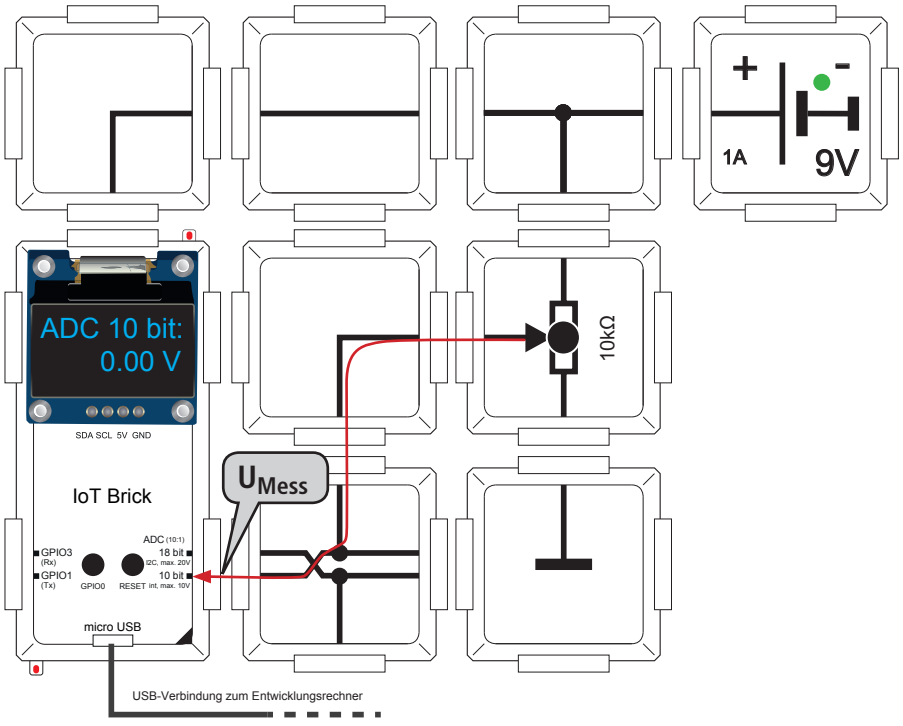


Abb. 41: Brick-Schaltung 10 bit A/D-Wandler

Die gemessene Spannung U_{Mess} wird auf dem OLED-Display ausgegeben und jede Sekunde aktualisiert. Desweiteren wird dir der digitale Rohwert und der daraus berechnete Spannungswert auf dem seriellen Monitor deines Rechners ausgegeben. Um den seriellen Monitor zu öffnen, klickst du in der Arduino IDE einfach oben rechts auf das Lupen-Symbol (siehe Kap. 5.2.3 auf Seite 20).

Mit folgender Formel kannst du den Digitalwert vom A/D-Wandler in den analogen Spannungswert umrechnen (siehe auch Beispielprogramm `Beispiel_6.5.3.ino`).

$$U_{Mess} = (analogIN / 1024) * 10\,V$$

Durch Einsetzen von 1023 für `analogIN` erhältst du den Maximalwert des Eingangsspannungsbereichs. In unserem Fall sind dies 9,99 V.

➔ Programmausschnitt siehe nächste Seite.

Programmausschnitt

```
void loop(void)
{
// Liest Spannung als Rohwert: 0 = 0V bis 1023 = 1V-1LSB
   analogIN = analogRead(adc_esp8266);
// Umrechnen in Volt mit analogIN/Auflösung (*1 Volt), *10 wegen 10:1 Teiler am ADC
   UMess = ((float)analogIN/1024)*10.0;
// Spannungswert in String umwandeln für OLED Ausgabe
   String UMess_str = String(UMess, 2); // 2 Nachkommastellen

   display.clear();       //OLED löschen

   display.setTextAlignment(TEXT_ALIGN_LEFT);
   display.setFont(ArialMT_Plain_24);
   display.drawString(0, 0, "ADC 10 bit:");
   display.setTextAlignment(TEXT_ALIGN_RIGHT);
   display.setFont(ArialMT_Plain_24);
   display.drawString(120, 32, UMess_str + " V");

  display.display();     // Ausgabe auf OLED
  delay(1000);
}
```

Mit dem Befehl `analogIN = analogRead(adc_esp8266);` lesen wir den digitalen Rohwert vom internen A/D-Wandler des ESP8266 ein. Der Wertebereich des 10 bit Wandlers geht von 0 bis 1023 (dezimal). In unserem Beispiel entspricht der Wert 0 der Spannung 0 V und der Rohwert 1023 dem Maximalwert des Eingangsspannungsbereichs.

Aus syntaktischen Gründen müssen wir die Integervariable `analogIN`, welche den Rohwert des A/D-Wandlers zurückliefert zunächst in die Floatvariable `UMess` umwandeln, um die Umrechnung in den richtigen Spannungswert vorzunehmen. Zur Ausgabe benötigen wir am Ende die Stringvariable `UMess_str`.

6.5.4 A/D-Wandler 18 bit

Öffne in der Arduino IDE den Sketch: `Beispiel_6.5.4.ino`. Folgende Header-Dateien musst du einbinden: `SSD1306Wire.h` und `MCP3421.h`. Falls du die entsprechenden Bibliotheken noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

In diesem Beispiel wollen wir den wesentlich präziseren 18 bit A/D-Wandler MCP3421 von Microchip ansteuern, der z. B. zur präzisen Temperaturmessung mit Thermoelementen oder Widerstandstemperatursensoren (RTDs) eingesetzt werden kann und stellen die Messwerte des internen 10 bit Wandlers denen des via I^2C-Bus angesteuerten 18 bit Wandlers gegenüber.

Die Brick-Schaltung für diese Übung unterscheidet sich nur durch den Brick unten in der Mitte gegenüber der vorigen Übung. Somit liegt nun an beiden Analogeingängen die gleiche Spannung U_{Mess} an, die über das Potentiometer variiert werden kann. Dies ermöglicht auf einfache Weise den direkten Vergleich der beiden A/D-Wandler. Die um das 256-fache bessere Auflösung des 18 bit-A/D-Wandlers gegenüber dem 10 bit-A/D-Wandler ist am Messergebnis deutlich zu erkennen.

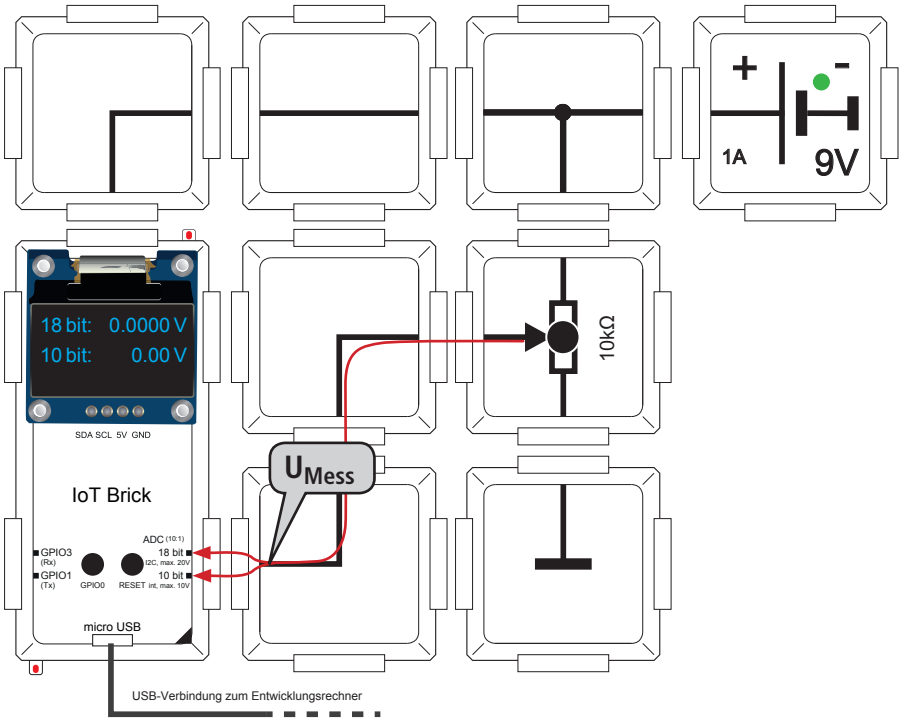


Abb. 42: Brick-Schaltung 18 bit A/D-Wandler

Aus dem Bauch heraus würden wir für diese Schaltung erstmal identische Spannungswerte erwarten. Aber der Spruch "Wer misst, misst Mist" gilt auch hier. Wie bereits im vorigen Kapitel erwähnt, spielen hier u. a. Bauteil-Toleranzen eine Rolle. In erster Linie wirkt sich die Standard-Toleranz der Widerstände von 10% der beiden 10-zu-1-Spannungsteiler an den Eingängen der A/D-Wandler aus (siehe Kap. „6.5.2.3 Der Spannungsteiler“ auf Seite 37).

➔ Programmausschnitt siehe nächste Seite.

Programmausschnitt

```
void setup(void)
{
…
// Init MCP3421: I2C-Adresse, 18 Bit Modus, keine Verstärkung
   MCP.init(0x68,3,0);
…
}
void loop(void)
{
…
// Liest Spannung als Double-Werte von MCP3421, Eingangsbereich: 0 bis 2,048V
   analogIN_18bit=MCP.getDouble();
   UMess_18bit = analogIN_18bit*10.0; // *10 wegen 10:1 Teiler
// Die nächste Zeile zur Ermittlung des Korrekturfaktors mittels
// Multimeter-Messung bitte auskommentieren!
   UMess_18bit = UMess_18bit*Correction_18bit;  // Korrekturfaktor
// Spannungswert in String umwandeln für OLED Ausgabe
   String UMess_18bit_str = String(UMess_18bit, 4);
…
  display.display();     // Ausgabe beider Spannungswerte auf OLED
  delay(1000);
}
```

Wir verwenden die Bibliothek `MCP3421.h`, um uns die Programmierung des MCP3421 zu vereinfachen. Der A/D-Wandler wird zunächst mit `MCP.init(0x68,3,0);` initialisiert. Mit der Funktion `MCP.getDouble()` wird der Spannungswert bereits als Doublewert – also als Gleitkommazahl – zurückgegeben. Da der MCP3421 einen Eingangsspannungsbereich von 0 bis 2,048 V überstreicht, muss der Wert noch mit Faktor 10 multipliziert werden. Als nächstes führen wir – wie in Kap. 6.5.2.4 auf Seite 38 beschrieben – einen Korrekturfaktor ein, der zu Beginn des Sketches als Konstante vom `float` definiert wird. Zur Ausgabe auf OLED-Display bzw. seriellen Monitor müssen wir die Gleitkommazahlen `UMess_10bit` und `UMess_18bit` in korrespondierende Strings umwandeln. Die Aktualisierung der Werte erfolgt etwa im Sekundentakt.

6.6 IoT-Beispiele

Steuere deinen IoT Brick über dein Smartphone oder andere beliebige WLAN-fähige Endgeräte und entdecke die Möglichkeiten. Programmiere eine kleine Webseite und realisiere deine eigene Schaltzentrale.

Du lernst…

…wie du Uhrzeit und Datum mit dem Internet synchronisieren kannst (siehe Beispiel 6.6.2)

…wie man Temperatur und Feuchtigkeit von einem Sensor einliest (siehe Beispiel 6.6.3)

…wie man den aktuellen Dollar-Kurs aus dem Internet abfragen kann (siehe Beispiel 6.6.4)

Im nächsten Schritt (Beispiel 6.6.5) programmierst du eine kleine Webseite, als Grundlage für deine eigene IP-Messzentrale. Über den universellen Sensor-Adapter-Brick (ALL-BRICK-0649) kannst du übrigens zahlreiche Sensoren leicht anschließen. Als krönenden Abschluss erfährst du in Beispiel 6.6.6, wie du über´s Internet LEDs ein- und ausschalten kannst. Dies ist eine Grundfunktionalität wie sie z. B. in zahlreichen Anwendungen der Gebäudeautomation benötigt wird.

Durch die simple Netzwerkintegration des IoT Bricks ergeben sich eine Vielzahl neuer Möglichkeiten.

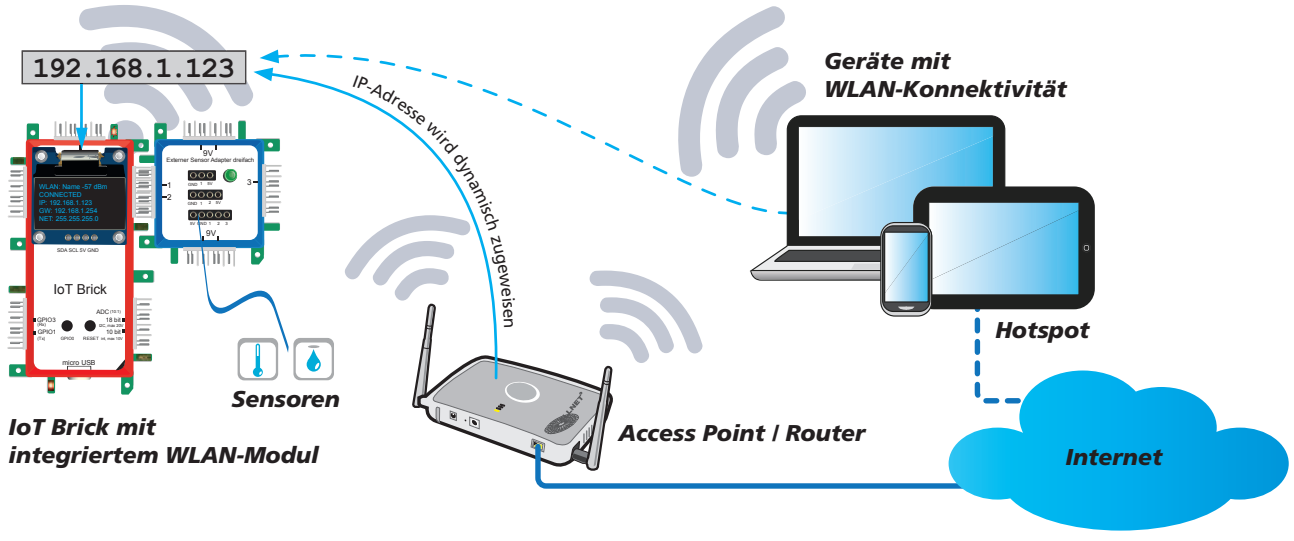


Abb. 43: Der IoT Brick im Netzwerk

Der große Vorteil des im IoT Brick verbauten ESP-12-F-Moduls gegenüber den Arduinos ist die Netzwerkfähigkeit dank integriertem WLAN-Modul. Der ESP8266 Microcontroller lässt sich mit wenigen Zeilen Code in ein bestehendes WLAN-Netzwerk einbinden. Die IP-Adresse wird grundsätzlich dynamisch zugewiesen. Dies kann der Router deines lokalen Netzwerks oder dein Smartphone als Hotspot sein.

Alle IoT-Beispiele benötigen eine Internet-Verbindung. Dies wird in den folgenden Kapiteln auch durch das WLAN-Symbol angedeutet. In den Beispielen 6.6.5 und 6.6.6 kannst du über dein Netzwerk auf den im IoT Brick integrierten Webserver zugreifen. Theoretisch auch via Internet, sofern du dein Netzwerk dafür einrichten kannst. Die Stichworte sind hier Port-Forwarding und Firewall. Handelsübliche Router bieten in der Regel die Möglichkeit einen entsprechenden Zugriff aus dem Internet an einen internen Port weiterzuleiten, hinter dem sich der IoT Brick mit seiner lokalen IP-Adresse verbirgt (sog. Port-Forwarding). Grundvoraussetzung ist, dass deine Firewall solche Zugriffe "von außen" zulässt. Weiterführende Informationen zur entsprechenden Konfiguration deines Routers findest du in der Dokumentation deines Routers oder im Internet.

6.6.1 IoT Brick als WLAN-Client einrichten

 Öffne in der Arduino IDE den Sketch: `Beispiel_6.6.1.ino`. Folgende Header-Dateien musst du einbinden: `ESP8266WiFi.h` und `SSD1306Wire.h`. Falls du die entsprechenden Bibliotheken noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

Um via WLAN mit deinem IoT Brick kommunizieren zu können, musst du ihn zuerst in dein lokales Netzwerk einbinden. Dazu brauchst du den WLAN-Namen (auch SSID genannt) deines Access-Points bzw. Routers und das zugehörige Passwort. Da wir am Brick keine Möglichkeit haben, Zeichen einzugeben, müssen wir die Zugangsdaten in unserem Sketch hinterlegen.

Suche die im folgenden Programmausschnitt grün hinterlegten Zeilen am Anfang der Sketch-Datei. Ersetze `mein_wlan_name` (Anführungszeichen bleiben stehen) durch den Namen (SSID) deines WLANs und anstatt `mein_wlan_passwort` gibst du das zugehörige Passwort ein.

Programmausschnitt

```
//hier eigenen WLAN-Namen (SSID) eintragen:
const char* ssid = "mein_wlan_name";
//hier eigenes WLAN-Passwort eintragen:
const char* password = "mein_wlan_passwort";
...
void setup() {
...
}
```

In der Funktion `void loop()` siehst du wie die folgenden Strings zusammengebaut und zeilenweise mit dem Befehl `display.drawString(x, y, "String")` zur Anzeige vorbereitet werden. Zur Ermittlung der einzelnen Parameterwerte werden die im folgenden genannten Funktionen verwendet.

- `wlan_oled` (WLAN-Name und Feldstärke), Funktion `WiFi.SSID()` und `WiFi.RSSI()`.
- `state` (Verbindungsstatus), Funktion `WiFi.state`
- `ip_oled` (IP-Adresse des IoT Bricks), Funktion `WiFi.localIP()[x]`
- `gw_oled` (Gateway-IP), Funktion `WiFi.gatewayIP()[x]`
- `mask_oled` (Subnetzmaske), Funktion `WiFi.subnetMask()[x]`

Die Ausgabe erfolgt schließlich mit dem Befehl `display.display();`

Parallel dazu werden die Informationen zur Netzwerkverbindung auch auf dem seriellen Monitor deines Rechners ausgegeben. Um den seriellen Monitor zu öffnen, klickst du in der Arduino IDE einfach oben rechts auf das Lupen-Symbol (siehe Kap. 5.2.3 auf Seite 20).

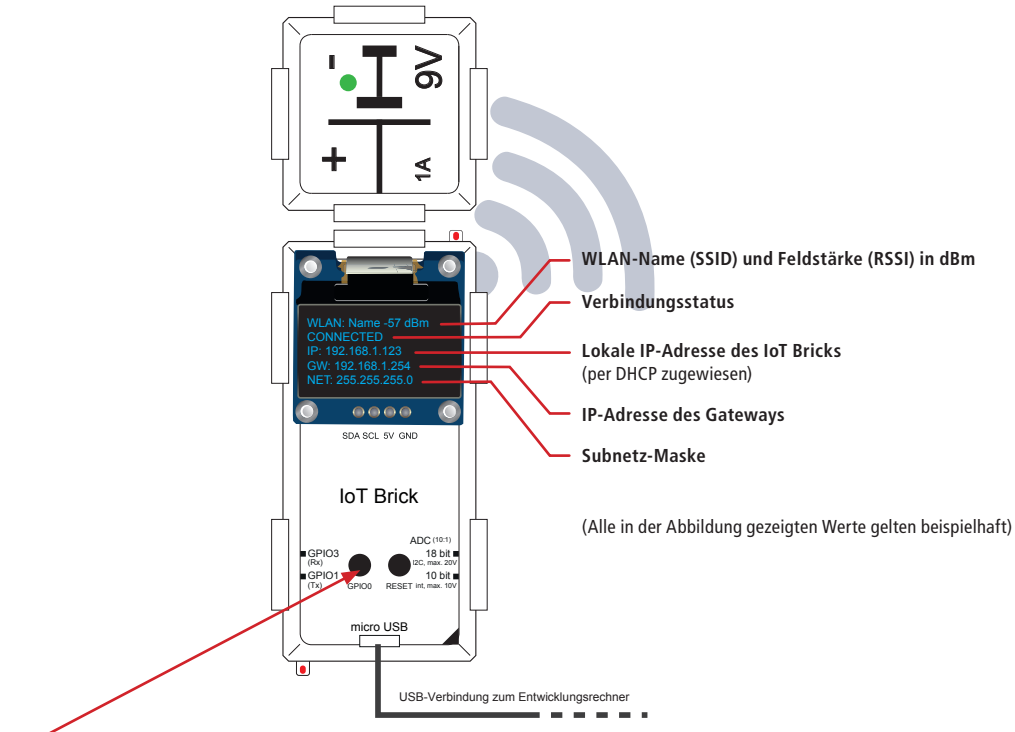
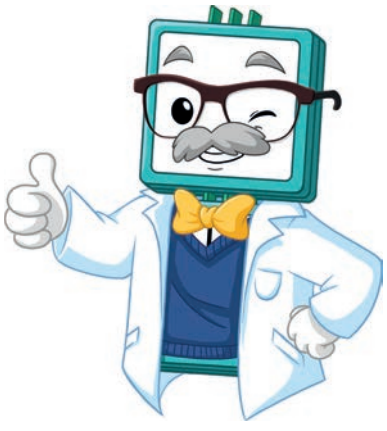


Abb. 44: Brick-Schaltung WLAN-Client einrichten

Die WLAN-Konfiguration des IoT Bricks kann auch in den folgenden Übungen durch Drücken der Taste GPIO0 angezeigt werden (siehe Abb. 44)!



6.6.2 Zeit aus dem Internet

 Öffne in der Arduino IDE den Sketch: `Beispiel_6.6.2.ino`. Folgende Header-Dateien musst du einbinden: `ESP8266WiFi.h`, `SSD1306Wire.h`, `TimeLib.h` und `NtpClientLib.h`. Falls du die entsprechenden Bibliotheken noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

Auch für dieses Beispiel musst du zunächst deinen WLAN-Namen (SSID) und das zugehörige Passwort im Beispiel-Programm eingeben. Gehe dazu wie in Kap. 6.6.1 beschrieben vor.

Im Internet gibt es sogenannte Zeit-Server, auf die man via NTP (Network-Time-Protocol) zugreifen kann, um die aktuelle Uhrzeit und das Datum zu ermitteln. Über die Domain `pool.ntp.org` kann jedermann kostenlos einen Pool aus Zeitservern nutzen. Um die Verarbeitung der vom Zeitserver übertragenen Information zu vereinfachen, benutzen wir in dieser Übung vordefinierte Funktionen in den Bibliotheken `TimeLib.h` und `NtpClientLib.h`. Als Anzeige verwenden wir wieder unser OLED-Display.

Programmausschnitt

```
...
void setup() {
...
//Falls WLAN verbunden, Verbindung zu NTP Zeit-Server herstellen
if(WLAN_connect_status){
   NTP.begin("pool.ntp.org", 1, true);  //Verbindung zu NTP Zeit-Server herstellen
   NTP.setInterval(63);          //Synchronisierung alle 63 Sekunden aktualisieren
   }

NTP.onNTPSyncEvent([](NTPSyncEvent_t event) {
   ntpEvent = event;
   syncEventTriggered = true;     //Verbindung zum Zeit-Server hergestellt
  });
...
}

void loop() {
...
   String time = NTP.getTimeStr();  //Uhrzeit als String in Variable time einlesen
   String date = NTP.getDateStr();  //Datum als String in Variable date einlesen
   display.clear();
   display.setTextAlignment(TEXT_ALIGN_LEFT);
   display.setFont(ArialMT_Plain_24);
   display.drawString(15, 5, time);    //Ausgabe der Uhrzeit vorbereiten
   display.drawString(5, 35, date);    //Ausgabe des Datums vorbereiten
   display.display();                 //OLED aktualisieren
...
}
```

Sobald die WLAN-Verbindung steht, wird mit der Funktion `NTP.begin("pool.ntp.org",1,true);` die Verbindung zum NTP-Server hergestellt. Dies kann einige Sekunden dauern. Der erste Parameter übergibt die URL zum Zeitserver-Pool, der zweite Parameter die Abweichung unserer Zeitzone bezüglich der "Universal Time Coordinated" (UTC) von +1 Stunde und der Wert `true` im dritten Parameter gibt an, dass in unserer Zeitzone zwischen Sommer- und Winterzeit umgestellt wird. Mit `NTP.setInterval(63)` wird definiert, in welchem Intervall in Sekunden die Synchronisation mit dem NTP-Server aktualisiert werden soll und `syncEventTriggered=true` meldet die erfolgreiche Synchronisation mit dem NTP-Server.

In `void loop()` wird mit der Funktion `NTP.getTimeStr()` die Uhrzeit und mit `NTP.getDateStr()` das Datum als String gelesen und zur Ausgabe vorbereitet. Die eigentliche Ausgabe auf das OLED-Display erfolgt mit dem Befehl `display.display();`. Wichtig ist in diesem Beispiel auch die Zeile `display.clear();` damit das Display in jedem Schleifen-Durchlauf gelöscht und nicht nur überschrieben wird.

Parallel zur Anzeige auf dem OLED-Display erfolgt die Ausgabe auch auf dem seriellen Monitor deines Rechners. Um den seriellen Monitor zu öffnen, klickst du in der Arduino IDE einfach oben rechts auf das Lupen-Symbol (siehe Kap. 5.2.3 auf Seite 20). Neben Uhrzeit und Datum werden hier noch weitere Informationen wie Sommer- bzw. Winterzeit sowie die seit der Erstsynchronisierung verstrichene Zeit (Uptime) und der Zeitpunkt der Erstsynchronisierung angezeigt.

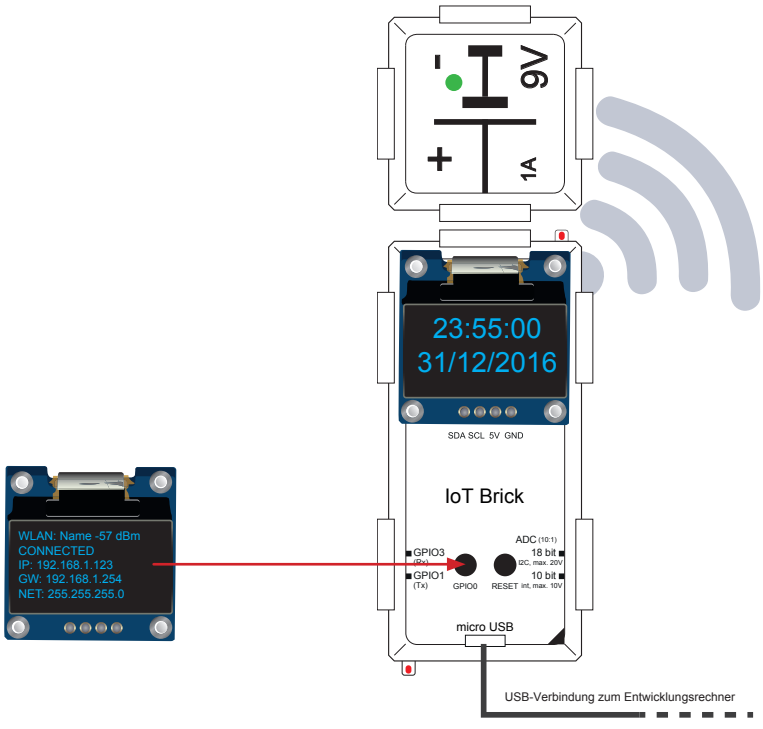
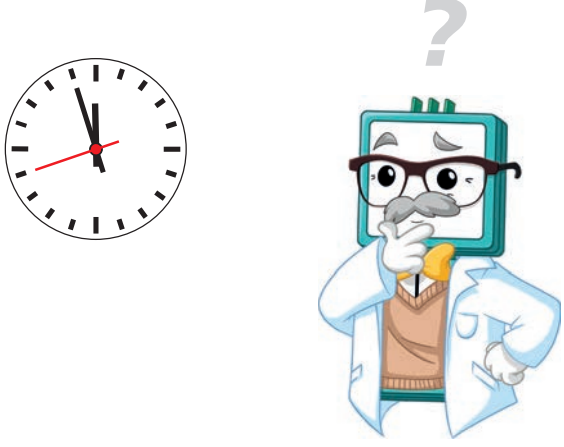


Abb. 45: Brick-Schaltung Zeit aus dem Internet

TIPP: Der Verbindungsstatus des IoT Bricks kann durch Drücken der Taste GPIO0 jederzeit angezeigt werden (siehe auch Abb. 44)!



6.6.3 Temperatur und Luftfeuchtigkeit messen

Öffne in der Arduino IDE den Sketch: `Beispiel_6.6.3.ino`. Folgende Header-Dateien musst du einbinden: `ESP8266WiFi.h`, `SSD1306Wire.h`, `TimeLib.h`, `NtpClientLib.h` und `DHT.h`. Falls du die entsprechenden Bibliotheken noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

Auch für dieses Beispiel musst du zunächst deinen WLAN-Namen (SSID) und das zugehörige Passwort im Beispiel-Programm eingeben. Gehe dazu wie in Kap. 6.6.1 beschrieben vor.

Zur Temperatur- und Feuchtigkeitsmessung verwenden wir den weitverbreiteten Kombi-Sensor vom Typ DHT11, für den es bereits eine Bibliothek (`DHT.h`) und Beispiele gibt. Daneben gibt es z. B. aus der Arduino-Welt viele verschiedene Sensoren wie:

- Temperatursensoren
- Infrarot-Lichtschranke
- Bewegungsmelder (IR-Sensor)
- Lichtsensor (LDR)
- Gas-Sensor
- Hall-Sensor
- Erschütterungssensor
- Sensortaste

Über den universellen Sensor-Adapter-Brick (ALL-BRICK-0649) kannst du zahlreiche handelsübliche Sensoren ganz leicht anschließen. Für viele Sensoren gibt es auch schon Beispiele und Bibliotheken, sodass sich diese auch software-seitig problemlos in eigene Projekte einbinden lassen.

Stecke zunächst die Bricks wie in Abb. 46 gezeigt zusammen. Beachte unbedingt den korrekten Anschluss des mitgelieferten DHT11-Sensors. Stecke den Sensor exakt wie in Abb. 46 gezeigt ganz links in die untere, 5-polige Buchsenleiste des Sensor-Adapter-Bricks. Die Beschriftung "5V" am Sensor muss dabei ganz links auf die Buchse mit der Beschriftung "5V" treffen. Ansonsten besteht die Gefahr, dass Sensor und/oder IoT Brick irreversibel beschädigt werden!

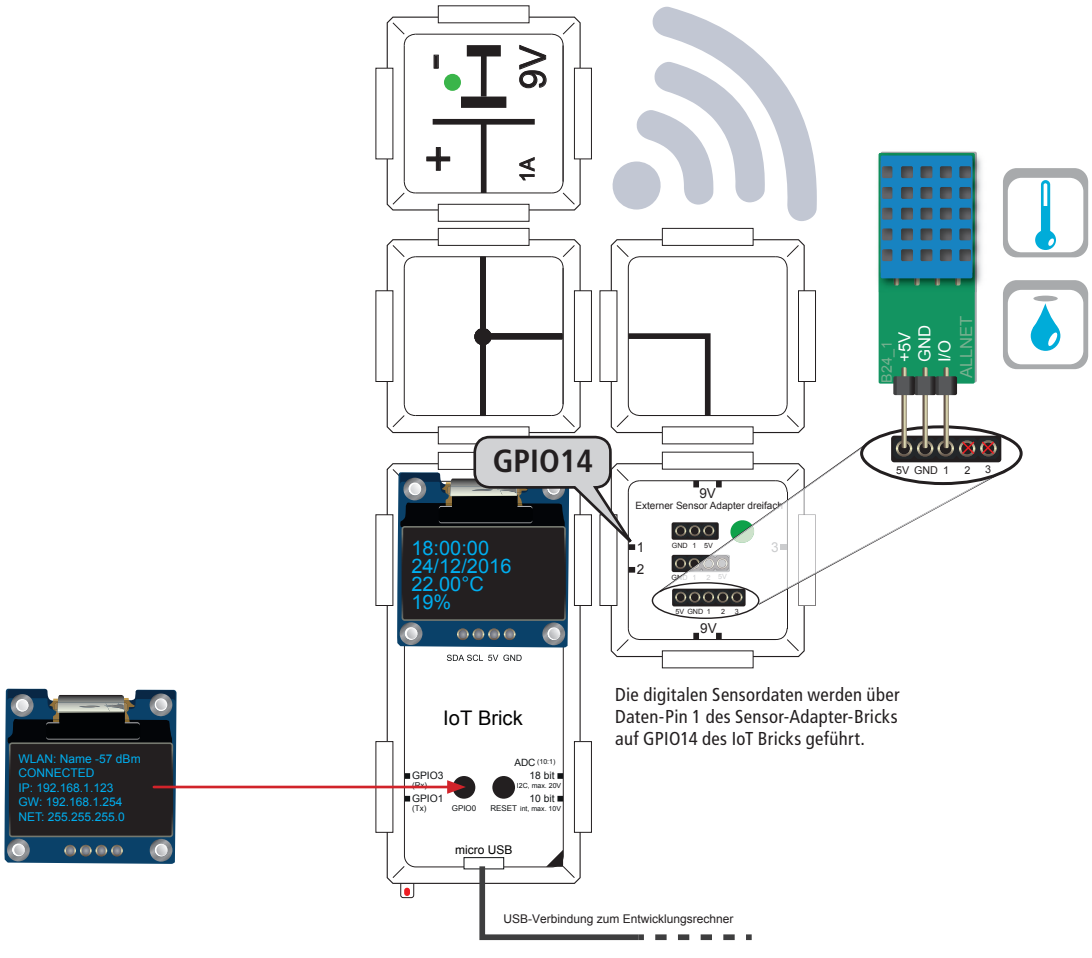


Abb. 46: Brick-Schaltung Temperatur und Luftfeuchtigkeit messen

TIPP: Der Verbindungsstatus des IoT Bricks kann durch Drücken der Taste GPIO0 jederzeit angezeigt werden (siehe auch Abb. 44)!

Programmausschnitt

```
#define DHT_TYPE DHT11     // Sensortyp definieren: DHT11
const int DHT_PIN = 14;  //Datenleitung des Sensors an GPIO14 des IoT Bricks
char temp[20];           //String-Variable für Temperatur definieren
char humi[20];           //String-Variable für Feuchtigkeit definieren

DHT dht(DHT_PIN, DHT_TYPE);    //Variable vom Typ DHT definieren

...
void setup() {
...
   dht.begin();                 //Sensor initialisieren
}

void loop() {
...
 if (counter%1000==0){               //Sensor etwa einmal pro Sekunde abfragen
    float t = dht.readTemperature(); //Temperatur auslesen (Celsius)
    float h = dht.readHumidity();    //Feuchtigkeit auslesen

    sprintDouble(temp,t,2);  //Temp. mit 2 Nachkommastellen in String konvertieren
    sprintDouble(humi,h,0);  //Feuchte ohne Nachkommastelle in String konvertieren
    strcat(temp," °C");      //String mit °C ergänzen
    strcat(humi," %");       //String mit %-Zeichen ergänzen

  }
  display.drawString(5, 30, temp);   //Ausgabe der Temperatur vorbereiten
  display.drawString(5, 45, humi);   //Ausgabe der Feuchtigkeit vorbereiten
  display.display();                 //OLED aktualisieren
...
}
```

Zu Beginn des Sketches wird der Sensortyp DHT11 definiert und an welchen GPIO-Pin (hier GPIO14) die Datenleitung unseres Sensors mit digitaler Schnittstelle angeschlossen wird. Es folgen verschiedene Variablen-Deklarationen. Eine spezielle Variable ist die vom Typ `DHT`. mit deren Hilfe in `void setup()` der Sensor initialisiert wird.

Doch nun zur eigentlichen Messung im Abschnitt `void loop()`. Mit den beiden Funktionsaufrufen aus der Sensor-Bibliothek `dht.readTemperature()` und `dht.readHumidity()` wird die Temperatur bzw. Feuchtigkeit vom Sensor gelesen und in den beiden Fließkommavariablen `t` bzw. `h` abgelegt. Mit der Hilfsfunktion `sprintDouble()` werden die Fließkommazahlen in Strings mit der gewünschten Anzahl an Nachkommastellen umgewandelt und mit der String-Operation `strcat()` wird anschließend noch die jeweilige Einheit ergänzt.

Neben den Werten für Temperatur und Feuchtigkeit wird die Anzeige noch mit Uhrzeit und Datum ergänzt – wie bereits in Übung 6.6.2 gezeigt. Die eigentliche Ausgabe aller Werte auf das OLED-Display erfolgt wie gewohnt mit dem Befehl `display.display();`.

Parallel dazu erfolgt die Ausgabe auch auf dem seriellen Monitor deines Rechners. Um den seriellen Monitor zu öffnen, klickst du in der Arduino IDE einfach oben rechts auf das Lupen-Symbol (siehe Kap. 5.2.3 auf Seite 20). In diesem Beispiel wurde die Ausgabe von Beispiel 6.6.2 um Temperatur und Feuchtigkeit ergänzt.

Durch entsprechende Anpassung von Steckplatz und Sketch können mit dem "Sensor-Adapter-Brick dreifach" zahlreiche auf dem Markt befindlichen Sensoren angeschlossen werden.

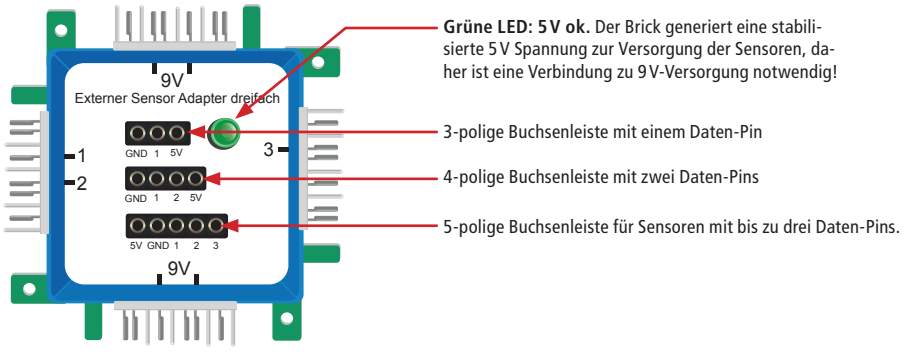


Abb. 47: Anschlussoptionen für Sensor-Adapter-Brick dreifach

Beachte unbedingt den korrekten Anschluss deiner Sensoren am Sensor-Adapter-Brick. Vergewissere dich in der Dokumentation des betreffenden Sensors, falls du dir nicht sicher bist. Ansonsten besteht die Gefahr, dass Sensor und/oder IoT Brick irreversibel beschädigt werden!

6.6.4 Dollarkurs aus dem Internet

Öffne in der Arduino IDE den Sketch: `Beispiel_6.6.4.ino`. Folgende Header-Dateien musst du einbinden: `ESP8266WiFi.h`, `SSD1306Wire.h`, `TimeLib.h`, `NtpClientLib.h` und `CurrencylayerClient.h`. Falls du die entsprechenden Bibliotheken noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

Auch für dieses Beispiel musst du zunächst deinen WLAN-Namen (SSID) und das zugehörige Passwort im Beispiel-Programm eingeben. Gehe dazu wie in Kap. 6.6.1 beschrieben vor.

Um den Dollarkurs (EUR-USD) aus dem Internet abzufragen brauchst du eine spezielle Bibliothek "Brick-ESP8266", welche du unter http://www.brickrknowledge.de/downloads herunterladen kannst. Sofern du in Kap. 5.2.1 schon alle Bibliotheken installiert hast, kannst du gleich weitermachen. Ansonsten musst du zunächst die Bibliothek installieren, wie in Kap. 5.2.1.2.3 auf Seite 19 beschrieben.

Programmausschnitt

```
#include <CurrencylayerClient.h>
...

if(counter%5000==0){      //Umrechnungskurs etwa alle 5 Sekunden aktualisieren
   currencylayer.getLastChannelItem();
   counter = 0;
   }
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.setFont(ArialMT_Plain_16);
display.drawString(0, 45, currencylayer.getFieldValue(0));
display.setTextAlignment(TEXT_ALIGN_RIGHT);
display.drawString(127, 45, " EUR/USD");
display.display();
...
```

In diesem Beispiel holen wir uns den aktuellen Dollarkurs aus dem Internet um auszurechnen, wieviel Euro ich für einen Dollar bezahlen muss (oder umgekehrt).

$$EUR = USD * Umrechnungsfaktor$$

Um den Umrechnungskurs zu bekommen, rufen wir zunächst die Funktion `currencylayer.getLastChannelItem()` auf. Hiermit wird über sog. JSON-Kommandos der aktuelle Dollarkurs von einem vordefinierten Server abgerufen. Das Zeitintervall für die Aktualisierung können wir über den Zähler in der if-Anweisung `if(counter%5000==0)` selbst bestimmen. Die Formatierung der Anzeige erfolgt wie gewohnt – wichtig ist jedoch noch der Aufruf `currencylayer.getFieldValue(0)`. Erst an dieser Stelle steht der auf 4 Nachkommastellen gerundete Dollarkurs als String im Sketch zur Verfügung.

Neben dem aktuellen Umrechnungskurs wird die Anzeige noch mit Uhrzeit und Datum ergänzt – wie bereits in Übung 6.6.2 gezeigt. Die eigentliche Ausgabe aller Werte auf das OLED-Display erfolgt wie gewohnt mit dem Befehl `display.display();`.

Parallel dazu erfolgt die Ausgabe auch auf dem seriellen Monitor deines Rechners. Um den seriellen Monitor zu öffnen, klickst du in der Arduino IDE einfach oben rechts auf das Lupen-Symbol (siehe Kap. 5.2.3 auf Seite 20). Die Ausgabe von Beispiel 6.6.2 wurde um einige Statusmeldungen ergänzt.

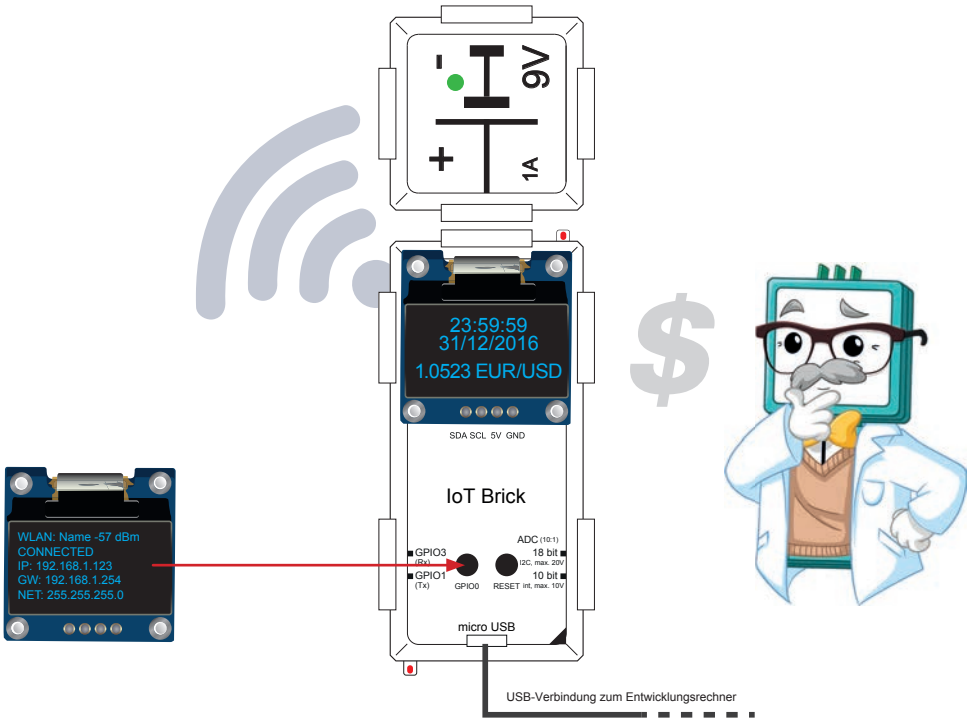


Abb. 48: Brick-Schaltung Dollarkurs aus dem Internet

TIPP: Der Verbindungsstatus des IoT Bricks kann durch Drücken der Taste GPIO0 jederzeit angezeigt werden (siehe auch Abb. 44)!

6.6.5 Meine erste Website

Öffne in der Arduino IDE den Sketch: `Beispiel_6.6.5.ino`. Folgende Header-Dateien musst du einbinden: `ESP8266WiFi.h`, `SSD1306Wire.h`, `TimeLib.h`, `NtpClientLib.h`, `ESP8266WebServer.h` und `DHT.h`. Falls du die entsprechenden Bibliotheken noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

Auch für dieses Beispiel musst du zunächst deinen WLAN-Namen (SSID) und das zugehörige Passwort im Beispiel-Programm eingeben. Gehe dazu wie in Kap. 6.6.1 beschrieben vor.

In diesem Beispiel werden wir eine einfache Website bauen, die uns Uhrzeit, Datum, Temperatur und Feuchtigkeit anzeigt. Grundlage jeder Website ist die Hypertext Markup Language (englisch für Hypertext-Auszeichnungssprache), abgekürzt: HTML. Den HTML-Code für deine erste Website kannst du mit einem WYSIWYG-HTML-Editor schreiben. Du kannst dazu verschiedene kostenfreie HTML-Editoren wie z. B. NVU für Windows, Linux und MAC OS X (siehe: www.nvu.com) oder BlueGriffon (siehe: www.bluegriffon.org) nutzen.

HTML-Grundkenntnisse sind für diese Übung vorteilhaft. Umfassende HTML-Grundlagen findest du im Rahmen des Web-Projekts SELFHTML unter www.selfhtml.org.

Die Brick-Schaltung und der Sketch dieser Übung bauen auf dem Beispiel 6.6.3 auf. Der Verbindungsstatus des IoT Bricks kann durch Drücken der Taste GPIO0 jederzeit angezeigt werden (siehe auch Abb. 44)!

Stecke zunächst die Bricks wie in Abb. 49 gezeigt zusammen, da ansonsten der Sketch nicht korrekt geladen wird. Beachte unbedingt den korrekten Anschluss des mitgelieferten DHT11-Sensors. Stecke den Sensor exakt wie in Abb. 49 gezeigt ganz links in die untere, 5-polige Buchsenleiste des Sensor-Adapter-Bricks. Die Beschriftung "5V" am Sensor muss dabei ganz links auf die Buchse mit der Beschriftung "5V" treffen. Ansonsten besteht die Gefahr, dass Sensor und/oder IoT Brick irreversibel beschädigt werden!

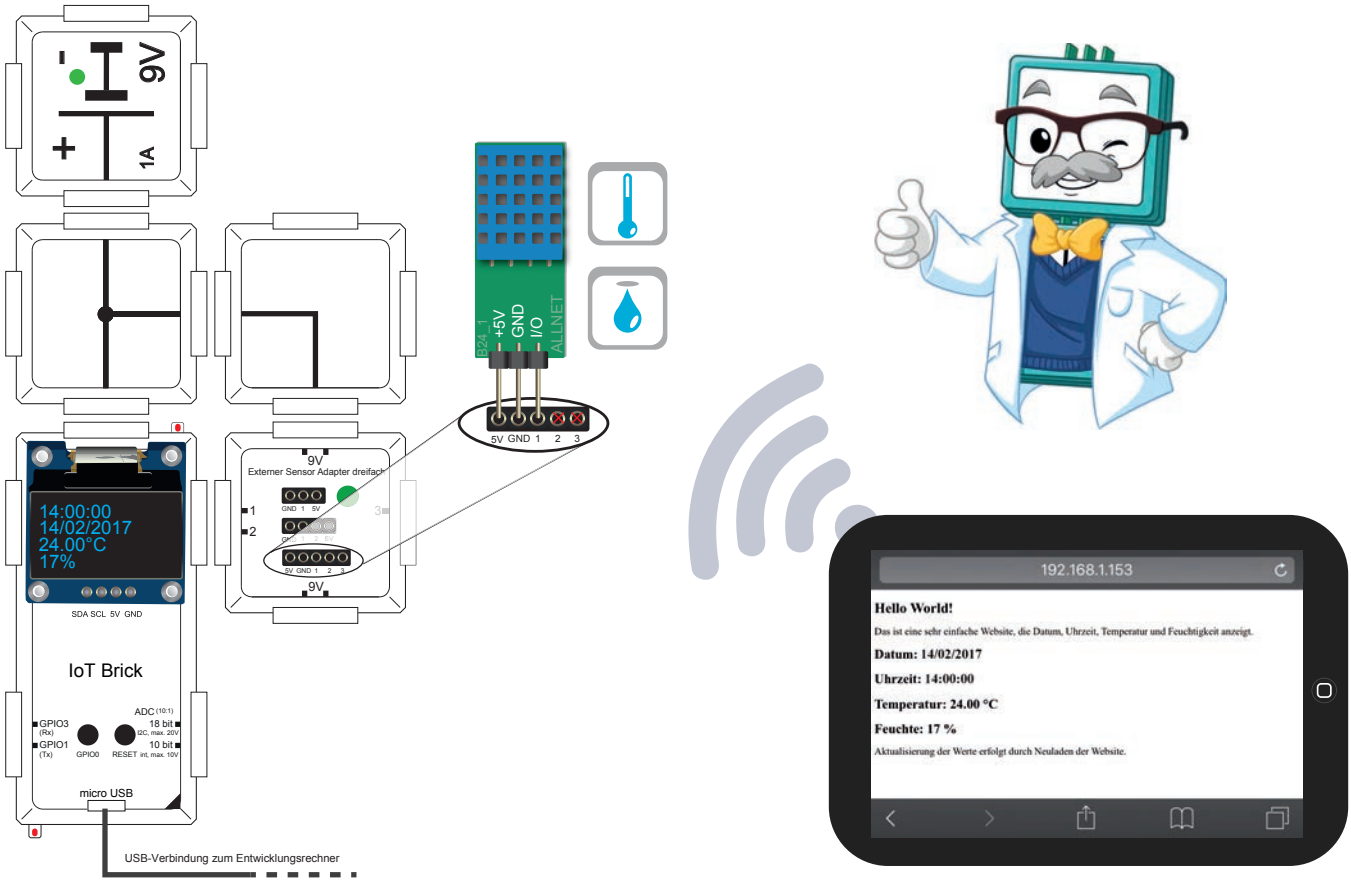


Abb. 49: Brick-Schaltung "Meine erste Website"

Programmausschnitt

```
...
ESP8266WebServer server(80); //Webserver starten auf Port 80
...
void setup() {
...
//Sobald ein Browser direkt auf das Stammverzeichnis zugreift,
//führe die Funktion handleRoot (siehe unten) aus.
   server.on("/", handleRoot);
   server.begin();                  //ab jetzt "hoert" Server auf HTTP-Anfragen
   Serial.println("HTTP server started");
}

void loop() {

   server.handleClient(); //Bediene die HTTP Anfragen
...
}

//Mit der Funktion handleRoot() wird die Website ausgeliefert
//sobald eine Anfrage von einem Browser eintrifft
void handleRoot() {
  String content;
  String time_web = NTP.getTimeStr();
  String date_web = NTP.getDateStr();
  String temp_web = temp;
  String humi_web = humi;
  content = "<!DOCTYPE html>";
  content += "<html>";
  content += "<head>";
 //Naechste Zeile ist wichtig, damit das Grad-Zeichen "°" korrekt dargestellt wird
  content += "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\">";
  content += "<title>Meine erste IoT Brick-Website</title>";
  content += "</head>";
  content += "<body>";
  content += "<h1> Hello World! </h1>";
  content += "<p>Das ist eine sehr einfache Website von deinem IoT Brick, die
              Datum, Uhrzeit, Temperatur und Feuchtigkeit anzeigt.</p>";
  content += "<h2>Datum: "+date_web+"</h2>";
  content += "<h2>Uhrzeit: "+time_web+"</h2>";
  content += "<h2>Temperatur: "+temp_web+"</h2>";
  content += "<h2>Feuchte: "+humi_web+"</h2>";
  content += "<p>Durch Neuladen der Website können die Werte jederzeit aktuali-
              siert werden.</p>";
  content += "</body>";
  content += "</html>";
  server.send(200, "text/html", content);
}
```

Am Anfang des Sketches wird der Webserver gestartet und Standard-Port 80 für Anfragen via HTTP (z. B.: http://www.brickrknowledge.com) zugewiesen. Sobald du nun in deinem Browser die lokale IP-Adresse deines IoT Bricks mit vorangestelltem http:// eingibst, (z. B. http://192.168.1.153), greifst du auf das Stammverzeichnis (engl. Root) deines Webservers zu, sodass die Funktion `handleRoot()` aufgerufen wird. In dieser Funktion wird der Inhalt der Website, in Form von HTML-Code beschrieben. Wie du vielleicht schon unter www.selfhtml.org nachgelesen hast, besteht die Grundstruktur einer jeden Website aus sog. HTML-Tags. Für jedes Element gibt es in der Regel einen Start-Tag und einen End-Tag (zu erkennen an dem vorangestellten Schrägstrich "/"), welche in den charakteristischen spitzen Klammern eingebettet sind. Zum Beispiel

`<p>Normaler Absatz</p>`. In der Funktion `handleRoot()`, wird nun in der String-Variable `content` der Inhalt unserer gesamten Website abgelegt. Um das Ganze übersichtlicher zu gestalten wurde die String-Zuweisung des HTML-Codes auf mehrere Zeilen im Sketch aufgeteilt. Dazu gehören alle Zeilen der Form `content += "...";`

Eine weitere Schwierigkeit ist, dass in der Sketch-Programmierung Anfang und Ende eines Strings mit Anführungszeichen oben (") gekennzeichnet werden. Da aber HTML-Code ebenfalls Anführungszeichen enthalten kann, müssen wir innerhalb des Strings zur korrekten Codierung eine sog. Escape Sequence `\"` für das Anführungszeichen verwenden. Der String selbst wird wiederum mit einem normalen Anführungszeichen begonnen und beendet.

So entspricht beispielsweise die Zeile…

```
content += "<p>\"Dieser Absatz steht in Anführungszeichen\"</p>";
```

…dem HTML-Code (UTF-8 codiert):

```
<p>"Dieser Absatz steht in Anführungszeichen"</p>
```

Ohne den vorangestellten Backslash an den beiden grün markierten Stellen, würde bereits das erste grün markierte Anführungszeichen den String schließen – anstatt das Vierte in der Zeile.

Zum Schluß wird die Website mit dem Kommando `server.send(200, "text/html", content);` ausgeliefert. Der erste Parameter definiert den HTTP-Statuscode, welcher bei erfolgreicher Ausführung zurückgeliefert wird. Der Code 200 bedeutet OK (die Anfrage wurde erfolgreich bearbeitet). Als nächstes wird die Art des ausgelieferten Inhalts definiert nämlich vom Typ "Text" und der dritte Parameter übergibt den `content` String mit dem Inhalt der Website.

6.6.6 Schalten via Website

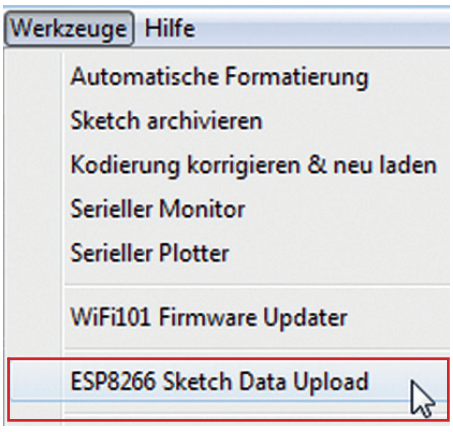
Öffne in der Arduino IDE den Sketch: `Beispiel_6.6.6.ino`. Folgende Header-Dateien musst du einbinden: `ESP8266WiFi.h`, `SSD1306Wire.h`, `TimeLib.h`, `NtpClientLib.h`, `ESP8266WebServer.h` und `FS.h`. Falls du die entsprechenden Bibliotheken noch nicht installiert hast, gehe zu Kap. 5.2.1 auf Seite 16 und hole dies nach.

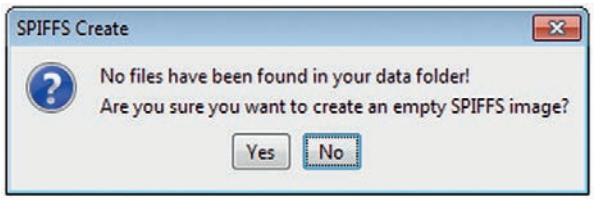
Auch für dieses Beispiel musst du zunächst deinen WLAN-Namen (SSID) und das zugehörige Passwort im Beispiel-Programm eingeben. Gehe dazu wie in Kap. 6.6.1 beschrieben vor.

Als Abschluss der Internet of Things Beispiele lernst du, wie man über eine Website eine Aktion auf dem IoT Brick auslösen bzw. dessen GPIO-Pins ansteuern kann. Als Grundlage für die Programmierung der Website benötigen wir diesmal etwas mehr als nur elementare HTML-Tags. Wir verwenden dazu das kostenlose CSS-Framework namens Bootstrap, mit dem du relativ einfach responsive Webdesigns gestalten kannst, die dann beispielsweise auf mobilen Endgeräten (wie deinem Smartphone oder Tablet) optimiert dargestellt werden. Es stehen Formulare, Buttons, Tabellen, Navigation und ein Grid-System für Layouts sowie verschiedene CSS-Klassen und Javascript-Komponenten zur Verfügung. Weitere Infos zum Thema Bootstrap findest du z. B. unter: https://www.bootstrapworld.de.

Damit du mit der Sketch-Programmierung richtig loslegen kannst, musst du noch etwas Fleißarbeit leisten. Wir müssen uns dazu Zugang zum SPIFFS Dateisystem (ESP8266 File System) des IoT Bricks verschaffen, um einige Dateien dort hinkopieren zu können. Gehe folgendermaßen vor:

1. Lade den Arduino ESP8266 File Uploader von: https://github.com/esp8266/arduino-esp8266fs-plugin/releases herunter. Der Sketch wurde mit der Version 0.2.0 (Direktlink: https://github.com/esp8266/arduino-esp8266fs-plugin/releases/download/0.2.0/ESP8266FS-0.2.0.zip) getestet.
2. Entpacke die ZIP-Datei in ein Verzeichnis auf deinem Rechner. Die darin enthaltene Datei `esp8266fs.jar` wird standardmäßig ins Unterverzeichnis `\ESP8266FS-0.2.0\ESP8266FS\tool` entpackt.
3. Überprüfe, ob auf deinem Rechner im Pfad "Dokumente – Arduino" bereits ein Verzeichnis "tools" vorhanden ist. Falls nicht, lege einen neuen Ordner mit dem Namen "tools" an. Dort werden diverse Werkzeuge installiert die man von der Arduino IDE aus bedienen kann, u. a. auch der von uns benötigte ESP8266 File Uploader.
4. Lege im Verzeichnis "tools" einen weiteren Unterordner mit dem Namen "ESP8266FS" an.
5. Lege im Verzeichnis "ESP8266FS" einen weiteren Unterordner mit dem Namen "tool" (ohne "s" am Ende) an.
6. Kopiere die Datei `esp8266fs.jar` (siehe Punkt 2) in den soeben angelegten Pfad "Dokumente – Arduino – tools – ESP8266FS – tool".
7. Starte die Arduino IDE neu
8. Im Menü "Werkzeuge" erscheint nun die Option "ESP8266 Sketch Data Upload".
9. Wenn man ein Projekt mit SPIFFS File System Unterstützung hat, werden alle Dateien, die im Unterverzeichnis "data" des Projektordners (Standardpfad: "Dokumente – Arduino") liegen, in ein Binär-Format konvertiert und in den SPI-Flash-Speicher des ESP8266-Moduls hochgeladen.
10. Starte den Datei-Upload im Menü "Werkzeuge" mit der Option "ESP8266 Sketch Data Upload". Falls beim ersten Upload die Meldung "SPIFFS Create" erscheint, bestätige mit "Yes".





Das Hochladen der Dateien kann mehrere Minuten dauern!

In diesem Beispiel werden wir den Doppel-LED-Brick wie in Abb. 50 gezeigt, an GPIO14 (rote LED) und GPIO13 (gelbe LED) anschliessen. Über zwei Buttons auf unserer Website können wir dann die LEDs – theoretisch weltweit – ein- und ausschalten. Das OLED-Display zeigt uns ergänzend Uhrzeit und Datum, wie wir es schon aus den vorangegangenen Übungen kennen.

Abb. 50: Brick-Schaltung "Schalten via Website"

TIPP: Der Verbindungsstatus des IoT Bricks kann durch Drücken der Taste GPIO0 jederzeit angezeigt werden (siehe auch Abb. 44)!

Wie bereits erwähnt, verwenden wir in diesem Beispiel auch fortgeschrittene Web-Technologien wie das Bootstrap-Framework sowie verschiedene CSS-Klassen (CSS = Cascaded Style Sheets) und Javascript-Komponenten (Javascript ist eine Scriptsprache, die häufig in Websites verwendet wird). Dies hat zur Folge, dass der Code wesentlich umfangreicher als in den vorangegangenen Übungen ist, baut aber auf diesen auf. Wir werden uns also bei der Besprechung des Sketches auf die wesentlichen Elemente konzentrieren. Bei Fragen zur Verwendung von Bootstrap, CSS und Javascript bitten wir euch im Internet entsprechend zu recherchieren oder ein entsprechendes Fachbuch zu kaufen, da dies den Umfang dieser Brick'R'knowledge Anleitung sprengen würde.

TIPP:

Für alle, die schon gute Kenntnisse in HTML, Bootstrap, CSS und Javascript haben und selbst die Website modifizieren möchten, haben wir den HTML-Code aus der Funktion `handleRoot()` "im Klartext" als Kommentar getarnt, in den Sketch kopiert. Zur Bearbeitung kannst du verschiedene kostenfreie HTML-Editoren wie NVU für Windows, Linux oder MAC OS X (siehe: www.nvu.com) oder BlueGriffon (siehe: www.bluegriffon.org) nutzen.

Damit dieses Beispiel korrekt funktioniert, musst du Javascript in deinem Browser aktivieren. Dies ist in der Regel standardmäßig der Fall.

Programmausschnitt

```
...
ESP8266WebServer server(80); //Webserver starten auf Port 80
...
void setup() {
...
//Wenn der Browser direkt auf das Stammverzeichnis zugreift,
//führe die Funktion handleRoot aus.
   server.on("/", handleRoot);
    // JS (Javascript)
    server.on("/js/jquery", handleJsJquery);
    server.on("/js/bootstrap", handleJsBootstrap);
    server.on("/js/bootstrap-switch", handleJsBootstrapSwitch);
    // CSS (Cascaded Style Sheets)
    server.on("/css/bootstrap", handleCssBootstrap);
    server.on("/css/bootstrap-switch", handleCssBootstrapSwitch);
    // Images (Bilder)
    server.on("/img/bg.png", handleImgBg);
    server.on("/img/brklogo.png", handleImgLogo);

    // WebServer Handles für die GPIO Schaltfunktionen
    server.on("/switch13-on", handleGpio13On);
    server.on("/switch13-off", handleGpio13Off);
    server.on("/switch14-on", handleGpio14On);
    server.on("/switch14-off", handleGpio14Off);

    server.begin();
    Serial.println("HTTP server started");
    setupPins();
}
...
//Handle-Funktion für GPIO13 Einschalten
 void handleGpio13On() {
  digitalWrite(gpio13, dOn);
  server.send(200, "text/html", gpio13Name+" on");
```

Fortsetzung nächste Seite...

In der Setup-Routine unseres Sketches werden mit den zahlreichen `server.on()` Anweisungen zunächst verschiedene Handles definiert, die bestimmen, welche Funktionen ausgeführt werden, sobald bestimmte Links im Browser aufgerufen werden. Falls z. B. der IoT Brick die IP-Adresse 192.168.1.153 vom DHCP-Server erhalten hat, dann würde das Aufrufen der Adresse `http://192.168.1.153/img/brklogo.png` dazu führen, dass die Funktion `handleImgLogo()` ausgeführt wird, die wiederum die Bilddatei `brklogo.png` aus dem internen Dateisystem (SPIFFS) liest und an den Webserver liefert. So wie die Logodatei eben, benötigt jede Javascript-, CSS- oder Bild-Datei, die im internen Dateisystem (SPIFFS) gespeichert ist, einen Datei-Handle damit wir darauf zugreifen können. Außerdem werden Handles definiert, mit denen der Webserver bei Eintreffen einer Browseranfrage zum Ein- oder Ausschalten eines GPIOs reagiert. Danach wird der Webserver mit `server.begin` gestartet. Die Handles selbst werden erst am Ende des Sketches definiert. Im obigen Beispiel-Code haben wir stellvertretend die Handle-Funktion `handleGpio13On()` zum Einschalten der LED an GPIO13 ausgewählt.

Programmausschnitt (Fortsetzung)

```
...
void loop() {

server.handleClient(); //Bediene die HTTP Anfragen

//Mit der Funktion handleRoot() wird die Website ausgeliefert
//sobald eine Anfrage von einem Browser eintrifft
void handleRoot() {
  String content;
  String network(ssid);
  content += "<html>";

  content += "<head>";
  //...diverse Meta-Tags, die nicht relevant für das Verständnis sind
  content += "<title>IoT Brick via WLAN "+network+" schalten</title>";
  content += "<link rel=\"stylesheet\" href=\"/css/bootstrap\">";
  content += "<link rel=\"stylesheet\" href=\"/css/bootstrap-switch\">";
  content += "<style>body{background-image:url(/img/bg.png);margin:0;
               padding:20px;background-size:100% auto;background-repeat:no-repeat;
               font-family:\"Helvetica Neue\",Helvetica,Arial,sans-serif;
               font-size:14px;line-height:1.5;color:#333;background-color:#fff}
               .col-sm-2{margin-top:20px}.img-thumbnail{border:0}</style>";
  content += "</head>";

  content += "<body>";
  content += "<div class=\"container-fluid\">";
  content += "<div class=\"row\">";
  content += "<div class=\"col-sm-2\"><img class=\"img-thumbnail\"
              src=\"/img/brklogo.png\"></div>";
  content += "</div>";
  content += "<div class=\"row\">";
  content += "<div class=\"col-sm-2\"><input type=\"checkbox\" id=\"switch14\"
               data-label-text=\""+gpio14Name+"\" data-label-width=\"120\"></div>";
  content += "<div class=\"col-sm-2\"><input type=\"checkbox\" id=\"switch13\"
               data-label-text=\""+gpio13Name+"\" data-label-width=\"120\"></div>";
  content += "</div>";
  content += "</div>";
  content += "<script src=\"/js/jquery\"></script>";
  content += "<script src=\"/js/bootstrap\"></script>";
  content += "<script src=\"/js/bootstrap-switch\"></script>";
  content += "<script type=\"text/javascript\">";
  content += "$('input[type=\"checkbox\"]').bootstrapSwitch({onSwitchChange:
              function(){$.ajax({url:'/'+$(this).prop('id')+'-'+($(this).prop('checked')?
              'on':'off')});});});";
  content += "var setAdc=function(){$.ajax({url:'/adc'}).done(function(data)
              {data=data||{};if(data.hasOwnProperty('data')){$('#adc')
              .text(data.data);}}).fail(function(jqxhr,textStatus,error){})
              .always(function(){setTimeout(setAdc,1000);});};
               setAdc();";
  content += "</script>";
  content += "</body>";

  content += "</html>";
  server.send(200, "text/html", content);  // HTTP-Statuscode 200 = OK
  }
}
```

In der Funktion `handleRoot()`, wird – wie bereits in Beispiel 6.6.5 praktiziert – in der String-Variable `content` der Inhalt unserer Website gespeichert. Auch hier ist wieder zu beachten, dass in der Sketch-Programmierung Anfang und Ende eines Strings mit Anführungszeichen oben (") gekennzeichnet werden. Da aber HTML-Code ebenfalls Anführungszeichen enthalten kann, müssen wir innerhalb des Strings zur korrekten Codierung eine sog. Escape Sequence `\"` für jedes vorkommende Anführungszeichen verwenden. Zur besseren Verständlichkeit schreiben wir die im folgenden Text zitierten Code-Schnipsel im "Klartext".

Mit dem Style-Element `<style>body{background-image:url(/img/bg.png);...` wird das Hintergrundbild der Website eingebunden sowie weitere Parameter definiert, welche das Erscheinungsbild der Website beeinflussen. Im Body-Bereich siehst du verschachtelte Div-Elemente, die der Strukturierung der Website dienen. In einem Div-Element können wiederum verschiedene Elemente wie Text, Bild oder Formulare zusammengefasst und deren Eigenschaften gesteuert werden. Beispielsweise wird mit dem Div-Element `<div class="col-sm-2"><img class="img-thumbnail" src="/img/brklogo.png">` das Logo eingebunden.

Die Buttons zum Ein- und Ausschalten der LEDs werden mit einem Input-Element vom Typ `"checkbox"` realisiert, die Formatierung erfolgt durch Stilelemente aus dem Bootstrap-Framework. Diverse Script-Elemente gegen Ende des Sketches geben den relativen Pfad zu Javascript-Dateien an, auf welche die Website zugreifen muss. Das Script-Element `<script type="text/javascript">` leitet schließlich einen längeren Abschnitt mit JavaScript-Code ein, in dem mit `onSwitchChange` das Ereignis der Betätigung eines Buttons abgefragt wird und die Handle-Funktion zum Ein- bzw. Ausschalten des LEDs aufgerufen wird.

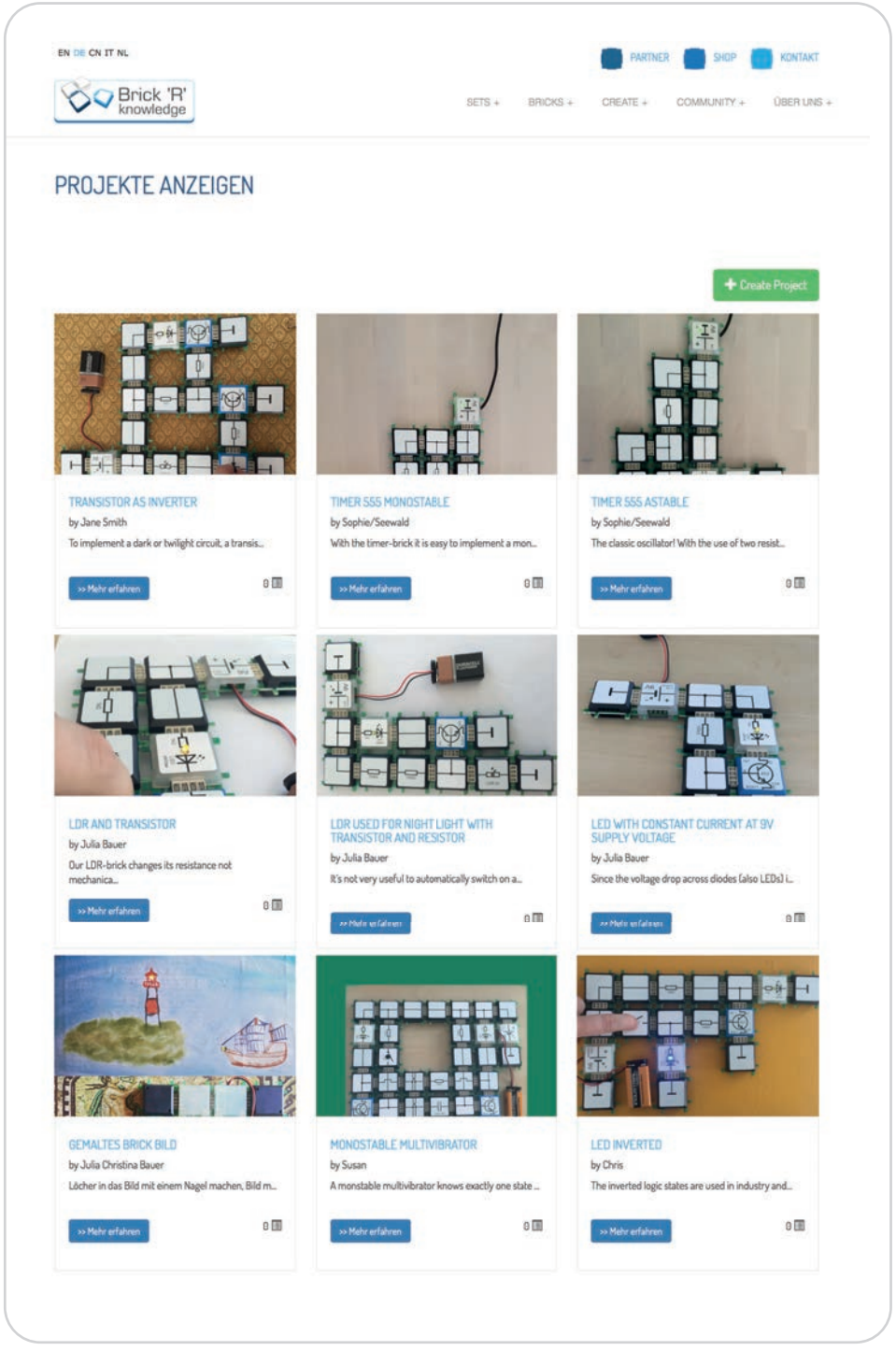
Mit dem Kommando `server.send(200, "text/html", content);` wird die Website dynamisch an den Browser ausgeliefert.

7. Brick Community

Das Brick-Universum dehnt sich aus: Ob auf Messen, auf unserer Website, auf YouTube oder in den sozialen Medien, überall findest du weitere Anregungen, Experimente und neue Bricks, mit denen du deiner Kreativität freien Lauf lassen kannst!

Mehr Projekte

Im Reiter „Create" kannst du Projekte und Schaltungen von anderen Community Mitgliedern ausprobieren, nachbauen und verbessern. Natürlich kannst du der Welt auch deine eigenen Experimente zeigen.

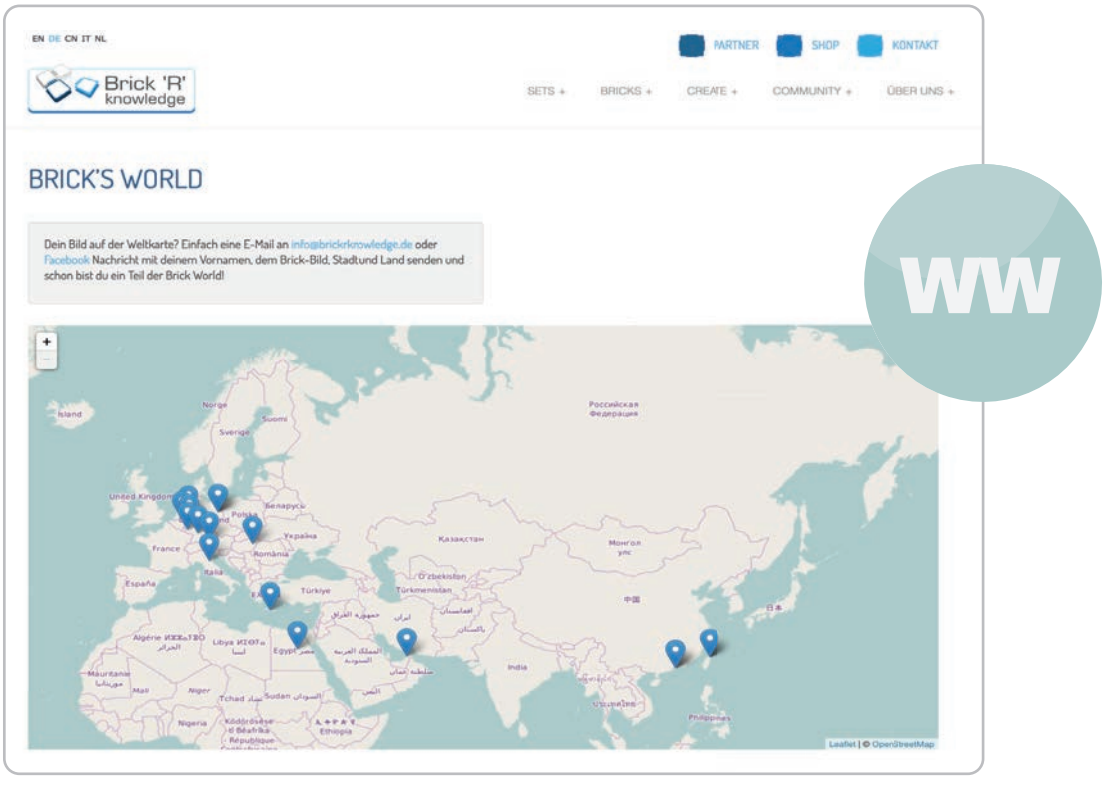


Social Media

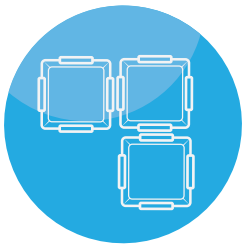
Im Reiter „Community“ findest du unsere Social-Media-Präsenzen und bleibst so immer up-to-date!

Weltweit

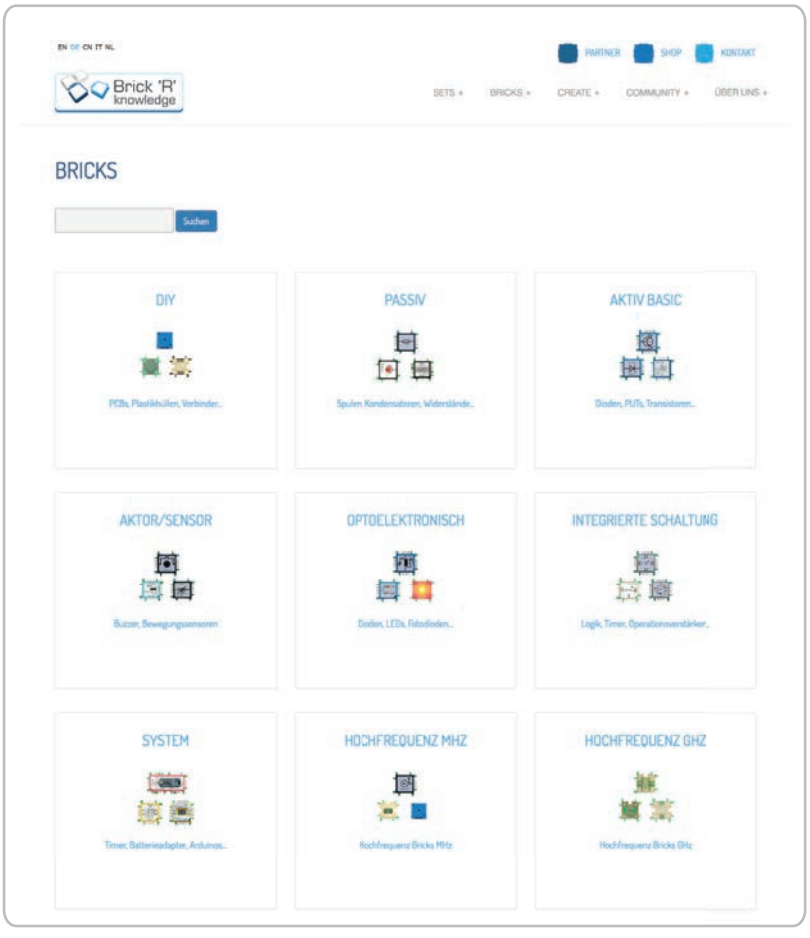
Ebenfalls im Reiter „Community“ kannst du sehen, wo es überall schon Brick-Mitglieder gibt, wo wir gerade sind oder mit welchem Wahrzeichen die Bricks schon fotografiert wurden. Hier kannst du uns auch dein Brick-Bild zusenden und du wirst es bald auf der Weltkarte finden!



Noch mehr Bricks!



Im Reiter „Bricks“ findest du alle verfügbaren Bausteine zum Erweitern deiner Schaltungen und Experimente.



Der Brick Blog

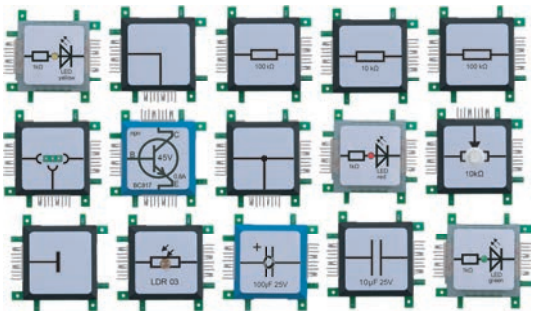
Jede Woche gibt es unter „Community“ einen neuen Blog-Post. Du findest hier Messeberichte, neue Experimente, witzige Geschichten und Informationen über neue Sets und Bricks.

8. Brick Sets im Überblick

Basic Set enthält 19 Bricks

ALL-BRICK-0374

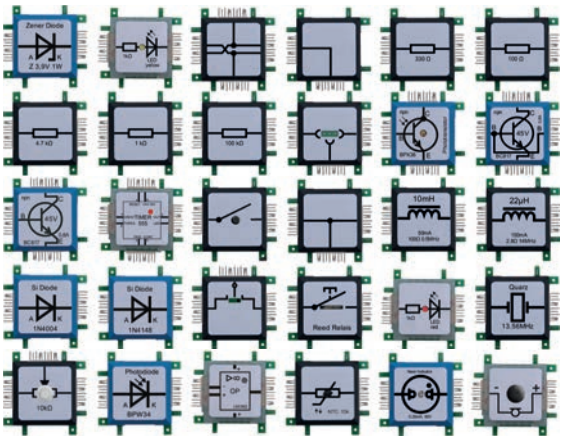
Das Basic Set bietet mit den 19 enthaltenen Bricks einen schnellen Einstieg in die Brick 'R' knowledge Welt und ermöglicht bereits eine Vielzahl von Experimenten. Mit der Basic-Variante können schon junge Entwickler eigene Schaltungen bauen und so ihre ersten physikalischen und technischen Experimente durchführen.



Advanced Set enthält 111 Bricks

ALL-BRICK-0223

Mit 111 Teilen bietet das Advanced Set alles, was zur Veranschaulichung komplexer elektronischer Schaltungen benötigt wird. Unter den über 100 Beispielschaltungen finden sich auch zahlreiche Anwendungen, die wir aus dem Alltag kennen. Das Set wurde so zusammengestellt, dass es auch von Ingenieurbüros zur kostengünstigen Visualisierung im Rahmen von Rapid-Prototyping genutzt werden kann.

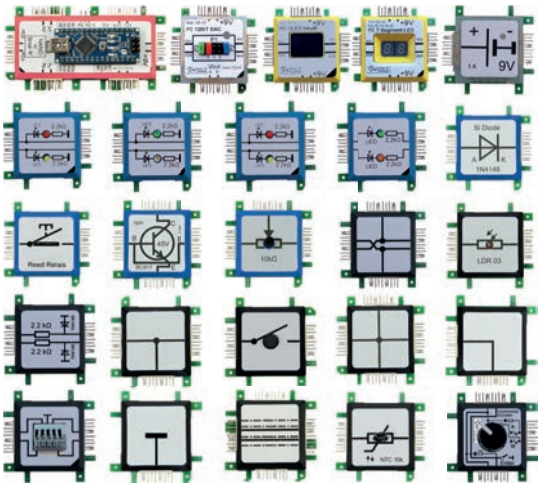


Arduino Coding Set

enthält 44 Bricks

ALL-BRICK-0414

Das Brick´R´knowledge Arduino Coding Set erweitert die Experimente hin zur Digitalelektronik mit der Einführung in die Microcontroller-Programmierung am Beispiel des Arduino Nanos. Neben Bricks für analoge Schaltungen enthält das Set auch Bricks für digitale Anwendungen wie eine 7-Segmentanzeige, ein OLED-Display, einen D/A-Wandler, einen I²C-Brick zur Pin-Erweiterung des Arduino Nanos, einen Arduino Nano Adapter-Brick und natürlich auch den Arduino Nano. Neben der Beschreibung der Experimente werden auch alle Programmierbeispiele zur Verfügung gestellt, um in die Welt der Arduino-Progammierung einsteigen zu können.

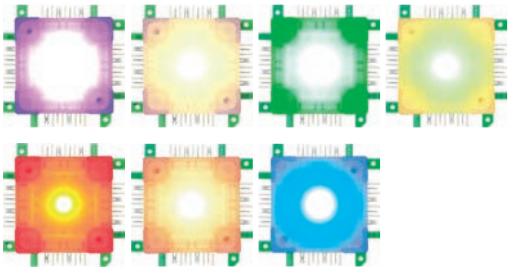


7 Color Light Set

enthält 28 Bricks

ALL-BRICK-0398

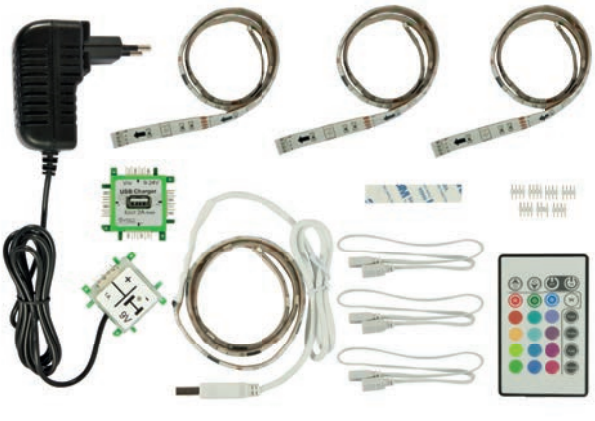
Mit den insgesamt 28 LED-Leucht-Bricks in 7 unterschiedlichen Farben lassen sich beeindruckende Lichtakzente in horizontaler und vertikaler Architektur setzen. Die 1 Watt LEDs in den Farben rot, gelb, blau, orange, violett, grün und warmweiß eignen sich perfekt für individuelle Licht-Figuren oder als mobile Beleuchtungslösung.



RGB Color Light Set

ALL-BRICK-0619

Erschaffe deine eigene Licht-Show! Das RGB Color Light Set enthält vier flexible LED-Streifen mit insgesamt 36 LEDs, die mit einer Infrarot-Fernbedienung angesteuert werden können. Die LED-Streifen können so geklebt, zugeschnitten und verbunden werden, wie du es wünschst. Die Infrarot-Fernbedienung hat 16 verschiedene Farbknöpfe und 4 Licht-Programme.

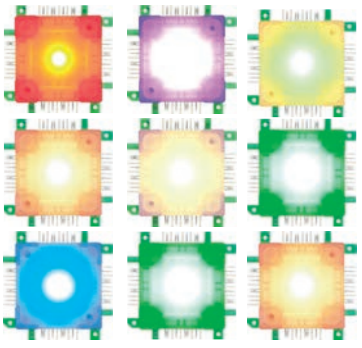


Programmable LED Set

enthält 49 Bricks

ALL-BRICK-0483

Das Set beinhaltet 49 ansteuerbare RGB-LED-Bricks mit zwei oder drei Anschlüssen, sowie einen Anschlussbrick für die Arduino-Steuerung und die Stromversorgung, einen Arduino Adapter-Brick und einen Arduino Nano. Das Set ermöglicht es, eigene LED-Animationen als Farb- oder auch bewegte Bildanimationen zu erstellen und sich spielerisch mit Microcontroller-Programmierung zu befassen. Innovative Lichtinstallationen und individuell leuchtende, blinkende und pulsierende Bilder in unterschiedlichen Farb- und Helligkeitsstufen sind durch das Programmable LED Set wunderbar umsetzbar.

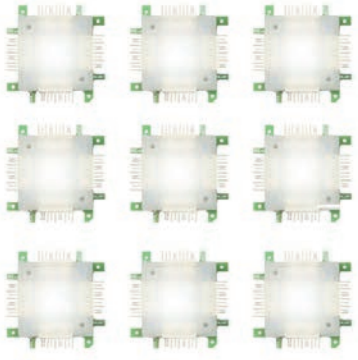


Highpower LED Set

enthält 50 Bricks

ALL-BRICK-0399

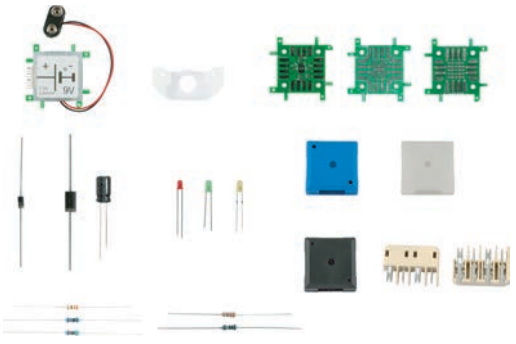
Das strahlende High Power LED Set enthält fünfzig 1 Watt High-Power-Bricks und dazu noch ein 12 Volt, 8 Ampere Netzteil. Die Bricks lassen sich ganz einfach zu individuellen Lösungen zusammenstecken. Zum Beispiel lassen sich aus den Bricks verschiedene Tischlampen bauen, die dann erweiterbar sind. Durch die starke Leuchtkraft bietet dieses ein stilvolles Ambiente und eignet sich perfekt als Nachtlicht. Das High Power LED Set ermöglicht es, sich spielerisch mit Lichtdesign auseinander zu setzen.



DIY Set

ALL-BRICK-0397

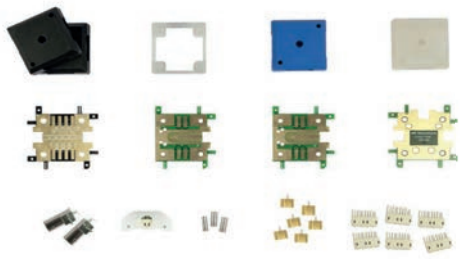
Das „Do-it-yourself“ Set ermöglicht es Tüftlern und Entwicklern, ihre eigenen Bricks in Ergänzung zu den bereits vorhandenen zu bauen. Die hier enthaltenen Komponenten bieten einen tiefen Einblick in Aufbau und Architektur der elektronischen Bauelemente. Mit Lötkolben und Lötzinn können die Tüftler die Standard-Bricks nachbauen oder eigene Bricks für individuelle Spezialanwendungen herstellen und somit sogar eigene Sets entwickeln.



MHz DIY Set

ALL-BRICK-0457

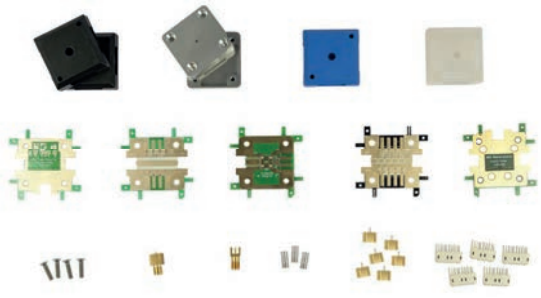
Mit dem MHz DIY Set lassen sich eigene Bausteine für Experimente und Schaltungen im MHz-Bereich realisieren. Das Set enthält drei verschiedene Raster- und Experimentierplatinen, sowie BNC-Buchsen, P-SMP-Stecker und die dazu passenden Verbinder. Außerdem enthält das Set eine Lötlehre für die SMD-Stecker und hermaphrodite Steckverbinder, um Eigenentwicklungen an das Brick-System anzupassen.



GHz DIY Set

ALL-BRICK-0458

Das GHz DIY Set bietet spannende Möglichkeiten zur Entwicklung im Hochfrequenzbereich bis hin zu Gigahertz-Frequenzen. Neben vier verschiedenen Platinen kann das GHz DIY Set auch mit verschiedensten Komponenten, wie liegenden und stehenden SMA- und P-SMP-Koaxialverbindern und den zum Brick-System gehörenden Steckverbindern dienen. So eignet sich das Set besonders für Messtechnik-Fans und Funkamateure.



Solar Set

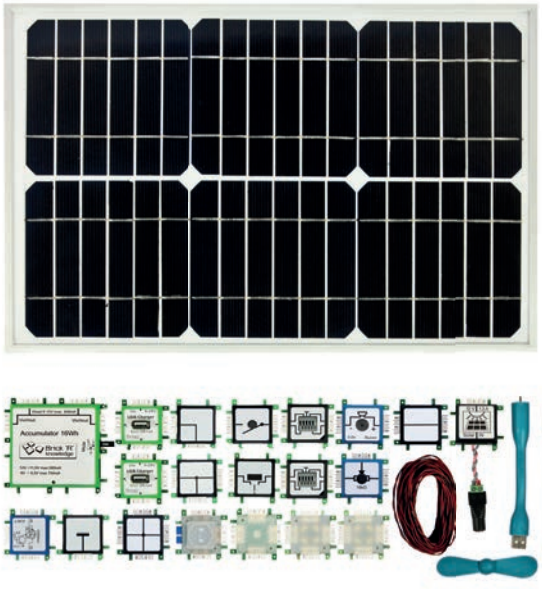
enthält 20 Bricks

ALL-BRICK-0484

Das Solar Set von Brick'R'knowledge garantiert Experimentierspaß für die ganze Familie und bringt Kindern erneuerbare Energien auf spielerische Art und Weise näher.

- Wie funktioniert eine Solarzelle?
- Wie speichert ein Akku Strom?
- Wie baut man ein Nachtlicht mit Bewegungsmelder?

Auf diese und weitere Fragen gibt das Solar Set Antworten. Mit diesem Set bist du offizielles Mitglied der Maker-Bewegung.



Measurement Set One

enthält 4 Bricks

ALL-BRICK-0637

Das Set ermöglicht es, mit Standardmessgeräten in Brick'R'knowledge Schaltungen Spannung, Stromstärke und andere Messgrößen einfach zu ermitteln. Das Messadapter-Set besteht aus folgenden Bricks: einem Messadapter mit 3 x 2 mm Buchse, einem Messadapter mit 4 mm Closed End GND in schwarz mit zusätzlicher Kabelklemme, einem Messadapter mit 4 mm Endpoint in gelb und einem Messadapter mit 4 mm Inline in rot.

Measurement Set Two

enthält 6 Bricks

ALL-BRICK-0638

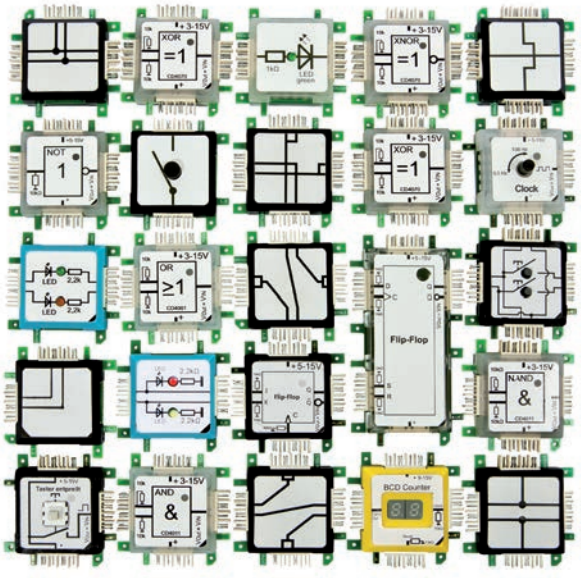
Das Set ermöglicht es, mit Standardmessgeräten in Brick'R'knowledge Schaltungen Spannung, Stromstärke und andere Messgrößen einfach zu ermitteln. Das Messadapter-Set besteht aus folgenden Bricks: zwei Messadapter mit 4 mm Closed End GND in schwarz, zwei Messadapter mit 4 mm Inline in rot und zwei Messadapter mit 4 mm Open End GND in schwarz.

Logic Set

enthält 93 Bricks

ALL-BRICK-0630

Das Logic Set eignet sich ideal für den schnellen Einstieg in die digitale Schaltungstechnik. Anhand des Begleithefts mit didaktisch aufeinander aufbauenden Schaltungsbeispielen können sich Lernende die wichtigsten Digitalschaltungen wie Addierer, Schieberegister und Zähler schnell erarbeiten. Aber auch Lehrende erhalten mit dem umfassend ausgestatteten Set eine praxisorientierte Basis für den täglichen Lehrbetrieb. Das Zusammenstecken und Experimentieren mit den Bricks macht Spaß und animiert zu eigenen Schaltungsvarianten. Der Lieferumfang des Logic Sets reicht von einfachen Logik-Bricks (AND, OR, NAND, NOR, XOR, XNOR, NOT) über verschiedene Flipflop-Bricks (D-, RS- und JK-Typ), weiter über einen Taktgeber-Brick (alternativ ein entprellter Taster für Einzelimpulse) bis hin zu einem BCD-Counter-Brick mit integrierter 7-Segment-Anzeige. Eine umfassende Auswahl an LED-, Taster- und Leitungs-Bricks runden das Set ab.



Brick 'R' knowledge

ALLNET© GmbH Computersysteme
Maistrasse 2
D-82110 Germering
www.brickrknowledge.com

Telefon: +49 (0)89 894 222 921
Fax: +49 (0)89 894 222 33
info@brickrknowledge.com

Maker Store & Maker Space
Danziger Straße 22
D-10435 Berlin
www.maker-store.de

Telefon: +49 (0)30 473 756 80
service@allknow.de