Brick 'R' knowledge

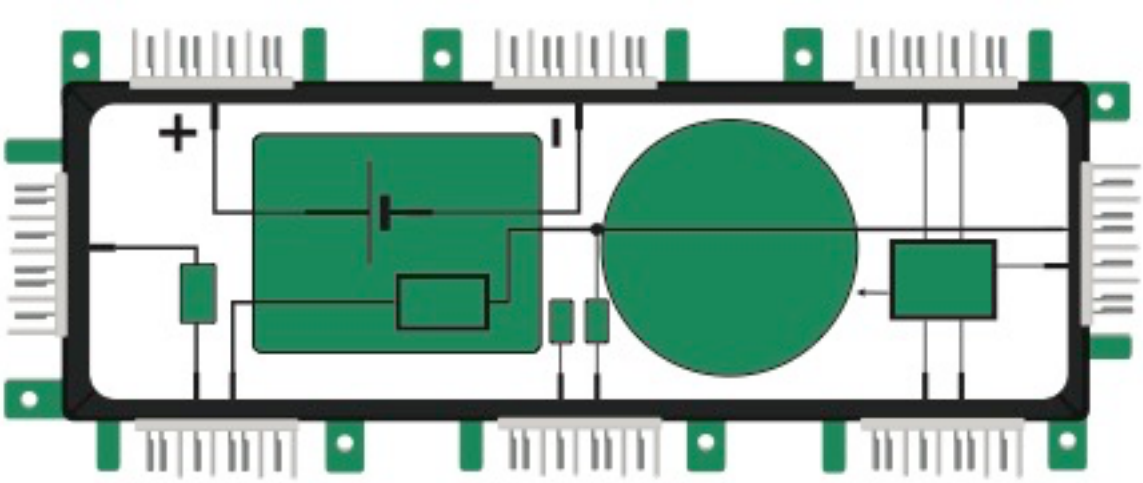
Fuel Cell Set

(Wasserstoff-Brennstoffzelle)

Handbuch

Experimentierkasten von Brick'R'knowledge
Kreativität fördern — Entwicklung stärken

Experimental kit by Brick'R'knowledge
Promote creativity — strengthen development



Impressum

Brick'R'knowledge Brennstoffzelle Anleitung Rev. 1.0

Datum: 09.03.2020

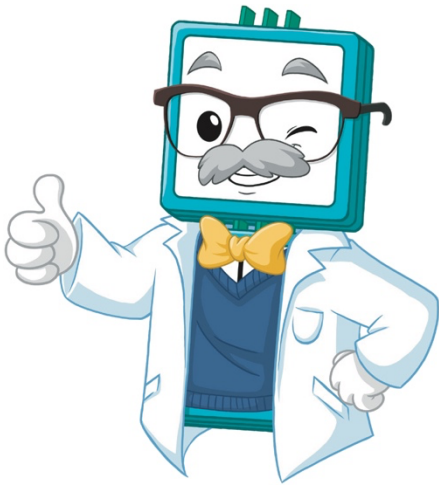
ALLNET® GmbH Computersysteme

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 System wurde zum ersten Mal auf der HAM Radio Ausstellung am 28.06.2014 von DM7RDK (Rufzeichen) vorgestellt. Hier liegt nun der neue Brennstoffzellen-Brick vor.

Das Besondere an unserem Elektronikset ist, dass die einzelnen Bausteine über ein Stecker-System verbunden werden, bei dem die Teile, welche zusammengefügt werden, 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 null Volt (0V, Masse) sind gleich zwei Kontakte vorhanden!

Damit lassen sich kompakte Schaltungen aufbauen, bei der die 0V-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 Brennstoffzellen-MKR-Brick wünscht

Rolf-Dieter Klein

1. Sicherheitshinweise

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

Zur Spannungsversorgung (9V) ausschließlich die original Brick'R'knowledge Netzteile (Batteriebausteine oder Steckernetzteile) verwenden. Die Versorgungsspannung beträgt hier gesundheitsungefährliche 9 Volt bei einem Stromfluss von ca. 1 Ampere. Bitte tragen Sie 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. Schauen Sie niemals direkt in eine Leuchtdiode (LED), da hier die Gefahr besteht, die Netzhaut zu schädigen (Blenden). Die Netzhaut befindet sich im Auge und hat die Aufgabe, die einfallenden Lichtreize durch die auf ihr befindlichen Zapfen (das Farbsehen) und die ebenfalls auf ihr befindlichen Stäbchen (Hell-, und Dunkelsehen) in für das Gehirn verwertbare Reize umzuwandeln.

Es ist unbedingt darauf zu achten, dass das zu verwendende Netzteil (Batteriebaustein oder Steckernetzteil) 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 hinzuziehen!

Gefahrenhinweis: Wasserstoff ist ein brennbares Gas und bildet mit der Luft explosive Mischungen. Die Versuche und auch die Erzeugung von Wasserstoff dürfen daher nur in gut belüfteten Räumen durchgeführt werden.

Je nach genutzter Wasserstoffquelle (Hydrostik Pro oder Wasserstoff-Druck-Gasflasche) sind die entsprechenden Druckminderer zu verwenden. Die Warn- und Gebrauchshinweise der Hersteller für die Wasserstoffquellen, Druckminderer und das Zubehör sind unbedingt zu beachten.

Wasserstoffquellen dürfen nie Temperaturen von über 50°C oder offenen Flammen ausgesetzt werden. Niemals Wasserstoffspeicher demontieren oder gewaltsam aufbrechen. Experimente nie bei Temperaturen von über 50°C oder in der Nähe von offenen Flammen durchführen. Beim Experimentieren niemals rauchen. Nach den Experimenten sind die Wasserstoffquellen immer von der Experimentieranordnung zu entfernen. Alle Zuleitungen, Anschlüsse, Druckminderer usw. sind auf Dichtheit zu prüfen, bevor mit den Experimenten begonnen wird.

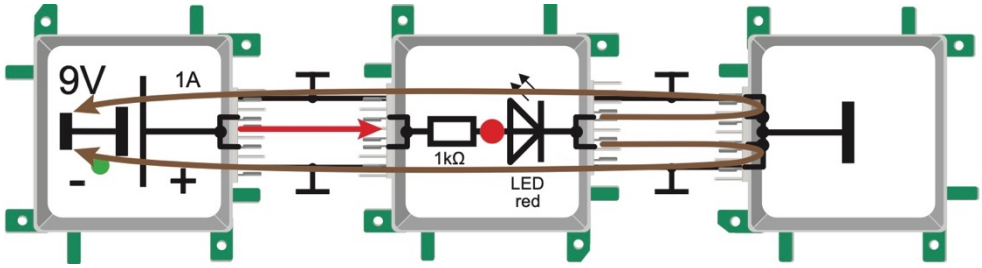
2. Grundlagen des Brick'R'knowledge Systems

2.1 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 0V her.

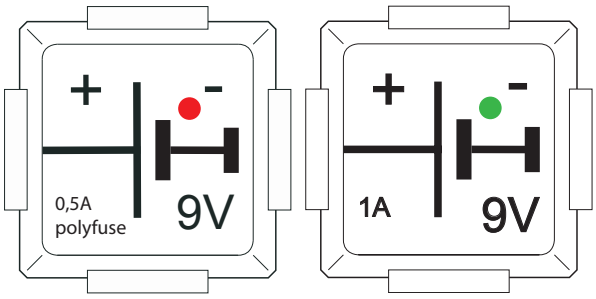
In unserer Schaltung sind das 9Volt gegenüber 0Volt: Man spricht einfach nur „Neun Volt“. Man erstellt in der Elektronik Schaltungen so, dass nachdem alle Bauelemente in ihrer Funktionsweise in die mehr oder weniger komplexen Stromkreise eingebracht sind, diese mit der „Masse“ verbunden werden. Schaltpläne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



Die Masse-Verbindung

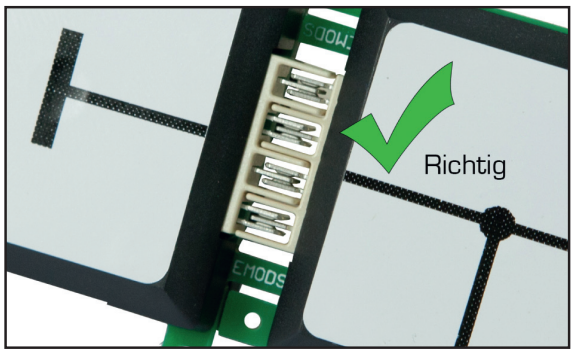
2.2 Die Spannungsversorgung

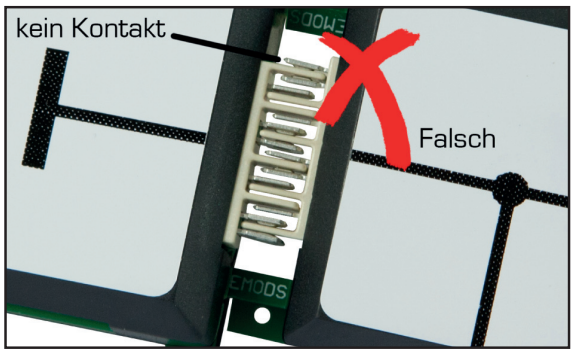


Die Spannungsversorgung des Arduino MKR-Adapter-Bricks erfolgt wahlweise über den Versorgungs-Brick via 9V-Blockbatterie (ALL-BRICK-0002) oder über das 9V-Steckernetzteil (ALL-BRICK-0221). Das Steckernetzteil (rechte Abbildung) liefert eine stabilisierte Gleichspannung von 9V und einen Maximalstrom von 1A. Bei Überlastung schaltet das Netzteil sofort ab, d.h. es ist kurzschlusssicher.

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!

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





Die Steckverbinder

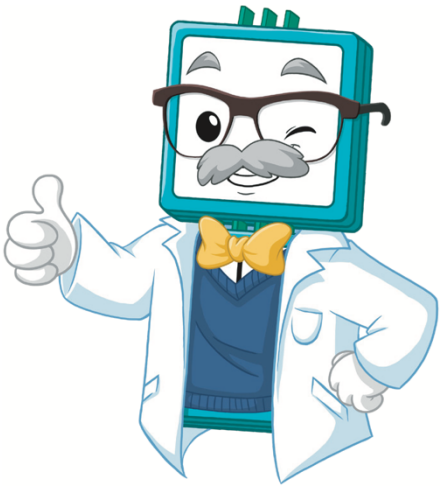
Im linken Bild seht ihr eine richtig gesteckte Verbindung. Die Verbindung besteht jeweils aus kleinen Stiften, die sich mechanisch verklemmen und dabei eine elektrische Verbindung herstellen. Um eine Isolation zwischen den Kontakten zu gewährleisten und einen Kurzschluss zu verhindern sind dazwischen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

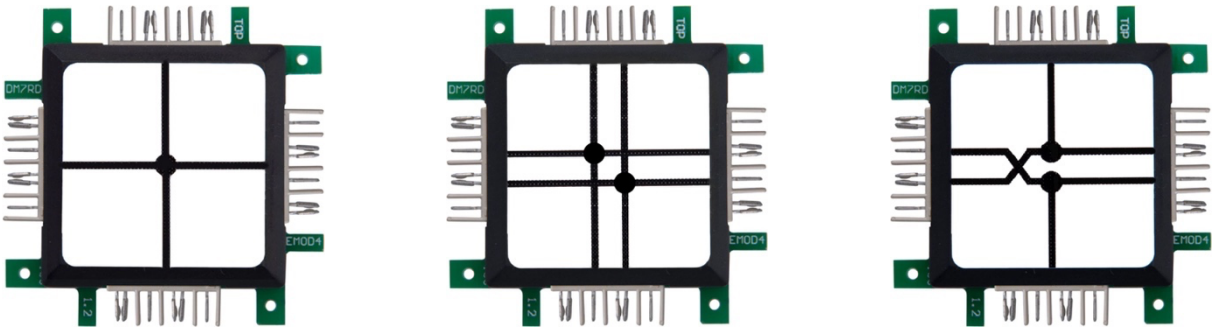


2.3 Signalführung

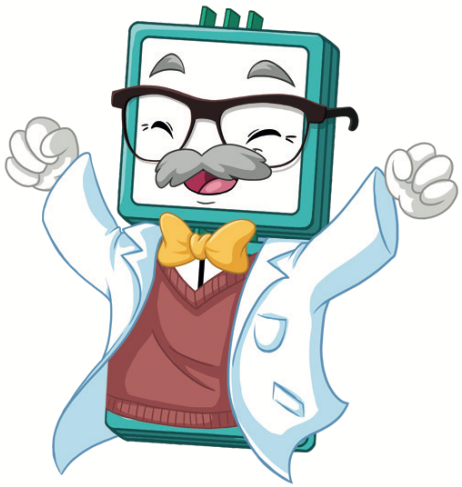
Für die Kennzeichnung der beiden mittleren Signalpins der Brick'R'knowledge Stecker gilt:

Falls nur ein einpoliges Signal über die Anschlüsse des Bricks geführt wird, zeigt das Label nur eine Linie, die mittig zum jeweiligen Stecker führt. In diesem Fall sind die beiden mittleren Signalpins stets verbunden (linkes Beispiel).

Sofern es sich um einen Brick handelt, der zwei getrennte Signale über einen oder mehrere seiner Stecker führt, werden grundsätzlich beide Signalpins gekennzeichnet. Falls an einem Stecker beide Signalpins verbunden sind, wie am oberen und unteren Stecker in der rechten Abbildung zu sehen, wird dies entsprechend dargestellt.



Signalführung einpolig (links), zweipolig (mittig) und ein-/zweipolig gemischt (rechts)



3. Grundlagen allgemein

3.1 Elektrische Größen

Zur Beschreibung der elektrischen Bauteile benötigen wir elektrische Größen, die im Rahmen des Internationalen Einheitensystems (SI = "Système international d'unités") definiert sind. Daher auch die weitverbreitete Bezeichnung SI-Einheiten. Man unterscheidet zwischen dem Namen der Größe, der Einheit und dem Einheitenzeichen (Symbol).

In folgender Tabelle sind die für das Powermeter relevanten Größen zusammengefasst.

Name der Größe	Formelzeichen	Einheit	Einheitszeichen (Symbol)
Elektrische Spannung	U	Volt	V
Elektrische Stromstärke	I	Ampere	A
(Wirk-)Leistung	P	Watt	W
Elektrische Energie	E	Joule (Wattsekunde)	J (Ws)
Elektrischer Widerstand	R	Ohm	Ω (Omega)

Die Symbole sind international einheitlich. Die Namen unterscheiden sich je nach Sprache. Für die Umsetzung sind in der Regel nationale metrologische Institute zuständig. Zum Beispiel:

• Deutschland: Physikalisch-Technische Bundesanstalt (PTB)

• Schweiz: Eidgenössisches Institut für Metrologie (METAS)

• Österreich: Bundesamt für Eich- und Vermessungswesen (BEV)

Übrigens:

Die Metrologie ist die „Wissenschaft vom Messen und ihre Anwendung", nicht zu verwechseln mit der Meteorologie also der Wetterkunde.

Mehr zum Thema internationales Einheitensystem findet Ihr unter:

https://de.wikipedia.org/wiki/Internationales_Einheitensystem

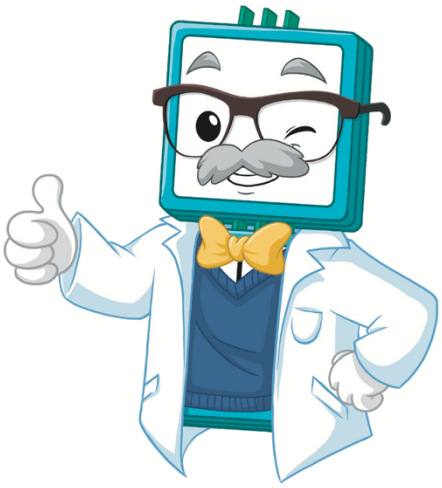
…und eine umfassende Liste physikalischer Größen unter:

https://de.wikipedia.org/wiki/Liste_physikalischer_Größen

3.2 Präfixe

Die sogenannten SI- oder Dezimalpräfixe sind für die Verwendung im Internationalen Einheitensystem (SI) definiert. Sie basieren auf Zehnerpotenzen mit ganzzahligen Exponenten. Man unterscheidet zwischen dem Namen des Präfix und seinem Symbol. Die Symbole sind international einheitlich. Die Namen unterscheiden sich je nach Sprache. Beachte die Groß-/Kleinschreibung der Symbole, das Symbol für Kilo wird aus historischen Gründen kleingeschrieben.

Name des Präfix	Symbol	Wert
Tera	T	10^{12} (Billion)
Giga	G	10^{9} (Milliarde)
Mega	M	10^{6} (Million)
Kilo	k	10^{3} (Tausend)
-	-	10^{0} (Eins)
Milli	m	10^{-3} (Tausendstel)
Mikro	µ	10^{-6} (Millionstel)
Nano	n	10^{-9} (Milliardstel)
Piko	p	10^{-12} (Billionstel)
Femto	f	10^{-15} (Billiardstel)



4. Unser Brennstoffzellen-Set

Eine Brennstoffzelle ist eine galvanische Zelle, die die chemische Reaktionsenergie eines kontinuierlich zugeführten Brennstoffes und eines Oxidationsmittels in elektrische Energie wandelt. Wenn auch mit Brennstoffzelle oft eine Wasserstoff-Sauerstoff-Brennstoffzelle gemeint ist, können je nach Brennstoffzellentyp außer Wasserstoff auch viele andere Brennstoffe genutzt werden, insbesondere Methanol, Butan oder Erdgas. Unsere Brennstoffzelle ist eine Wasserstoff-Sauerstoff-Brennstoffzelle.

Brennstoffzellen alleine sind keine Energiespeicher, sondern Energiewandler, denen Energie in chemisch gebundener Form mit den Brennstoffen zugeführt wird. Ein komplettes Brennstoffzellensystem kann aber einen Brennstoffspeicher enthalten.

Die Gewinnung von elektrischer Energie aus chemischen Energieträgern erfolgt zumeist durch Verbrennung und Nutzung der entstehenden heißen Gase in einer Wärmekraftmaschine mit nachgeschaltetem Generator. So wird erst chemische Energie durch Verbrennung in thermische Energie und dann in mechanische Arbeit umgewandelt. Erst aus dieser wird im Generator Strom erzeugt.

Eine Brennstoffzelle ist jedoch geeignet, die Umformung ohne die Umwandlung in Wärme und Kraft zu erreichen und ist dadurch potenziell effizienter. Im Unterschied zu einer Verbrennungskraftmaschine wandelt sie chemische Energie direkt in elektrische Energie um und unterliegt nicht dem schlechten Wirkungsgrad von Verbrennungskraftmaschinen. Die theoretisch erreichbare Nutzarbeit ist allein durch die freie Enthalpie der chemischen Reaktion beschränkt und kann damit höher sein als bei der Koppelung einer Wärmekraftmaschine mit einem Generator zur Stromerzeugung.

Das Prinzip der Brennstoffzelle wurde 1838 von Christian Friedrich Schönbein gefunden, als er zwei Platindrähte in verdünnter Schwefelsäure mit Wasserstoff bzw. Sauerstoff umspülte und zwischen den Drähten eine elektrische Spannung bemerkte. Sir William Grove erkannte zusammen mit Schönbein die Umkehrung der Elektrolyse und das Erzeugen von Strom durch mehrere Versuche.

Recht bald war man von den Brennstoffzellen regelrecht begeistert. Man hoffte, Kohle und Dampfmaschinen zu ersetzen. 1875 schrieb Jules Verne in seinem Buch „Die geheimnisvolle Insel“ über die Brennstoffzelle: „Das Wasser ist die Kohle der Zukunft. Die Energie von morgen ist Wasser, das durch elektrischen Strom zerlegt worden ist. Die so zerlegten Elemente des Wassers, Wasserstoff und Sauerstoff, werden auf unabsehbare Zeit hinaus die Energieversorgung der Erde sichern.“

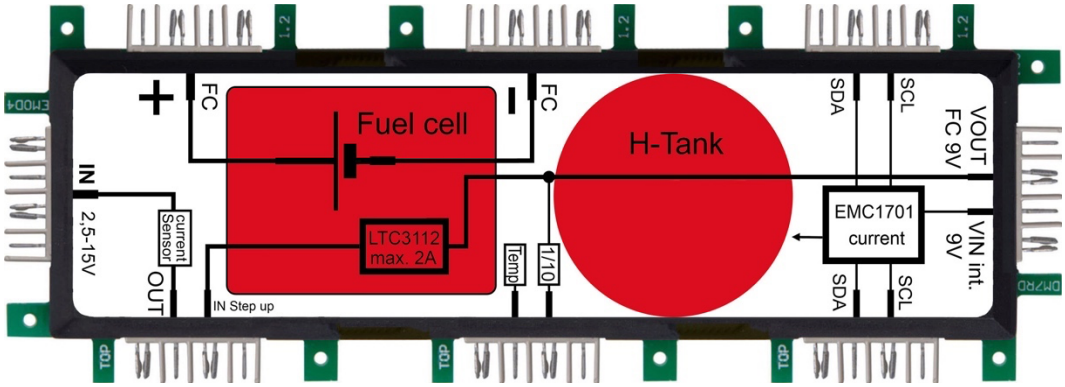
Wegen der Erfindung des elektrischen Generators, damals Dynamomaschine genannt, durch Werner von Siemens geriet die als „galvanische Gasbatterie“ bezeichnete Erfindung zunächst in Vergessenheit. Die Dynamomaschine war in Verbindung mit der Dampfmaschine bezüglich Brennstoff und Materialien relativ unkompliziert und wurde daher zu dieser Zeit der komplexen Brennstoffzelle vorgezogen. Wilhelm Ostwald machte sich um die theoretische Durchdringung der Brennstoffzelle verdient. 1894 erkannte er ihr hohes Potential gegenüber den Wärmekraftmaschinen.

Erst in den 1950er Jahren wurde die Idee wieder aufgegriffen, da in der Raumfahrt und beim Militär kompakte und leistungsfähige Energiequellen benötigt wurden. Die Brennstoffzelle wurde ab 1963 erstmals an Bord eines Satelliten und für die Gemini- und Apollo-Raumkapseln eingesetzt.

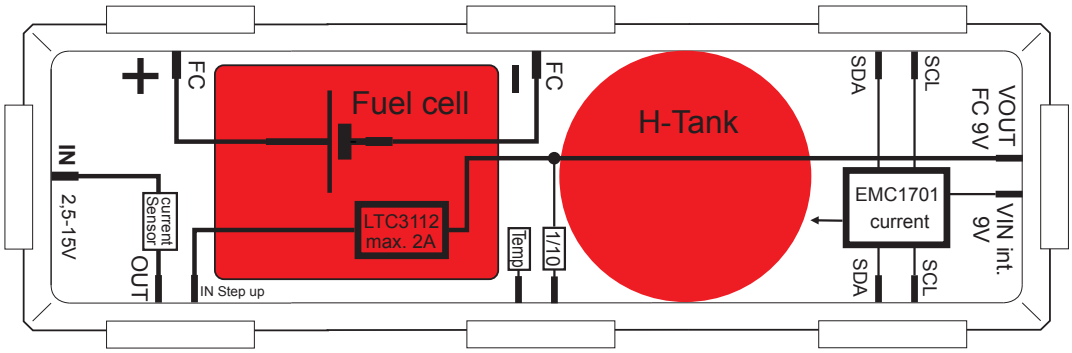
In den 1990er-Jahren führte die kalifornische Gesetzgebung zu neuen Impulsen. Nun wurden Fahrzeuge mit niedrigen Emissionen von jedem Hersteller gefordert. Seither machte die Brennstoffzellenentwicklung international große Fortschritte, sowohl in der Forschung als auch in der Anwendung.

4.1 Brennstoffzellen-Brick

Bricks sind kompakte Bausteine aus der Welt der Elektrotechnik. Sie eignen sich zum Ausprobieren neuen Wissens für junge Forscher, die den Umgang mit komplexen Schaltungen kennenlernen wollen. Unsere Brennstoffzelle wandelt Wasserstoff (aus einem HydroStik Pro Behälter) und Sauerstoff (aus der Luft) in Elektrizität um.



ALLNET Brick'R'knowledge Brennstoffzellen Brick mit Temperatur- und Spannungsmessung
Hersteller-Nummer: ALL-BRICK-0711 • ALLNET-Art.-Nr.: 180230



Der Pluspol der Brennstoffzelle (FC+) ist links oben verfügbar, der Minuspol (FC-) mittig oben. Dieser wird mit einem Masse-Brick verbunden. Der Spannungssensor (Current Sensor, EMC 1701) misst die Spannung zwischen dem linken Anschluß (IN) und dem linken Pin des Anschluß links unten (OUT). Der rechte Pin des Anschluß links unten (IN Step up) ist der Eingang für den 9V Spannungswandler (LTC3112). Der Ausgang des Spannungswandlers (VOUT) ist der obere Pin des Anschluß rechts. Der untere Pin des Anschluß rechts ist die interne Spannungsversorgung (VIN int.) für den i2c-Bus und den Spannungssensor EMC1701.

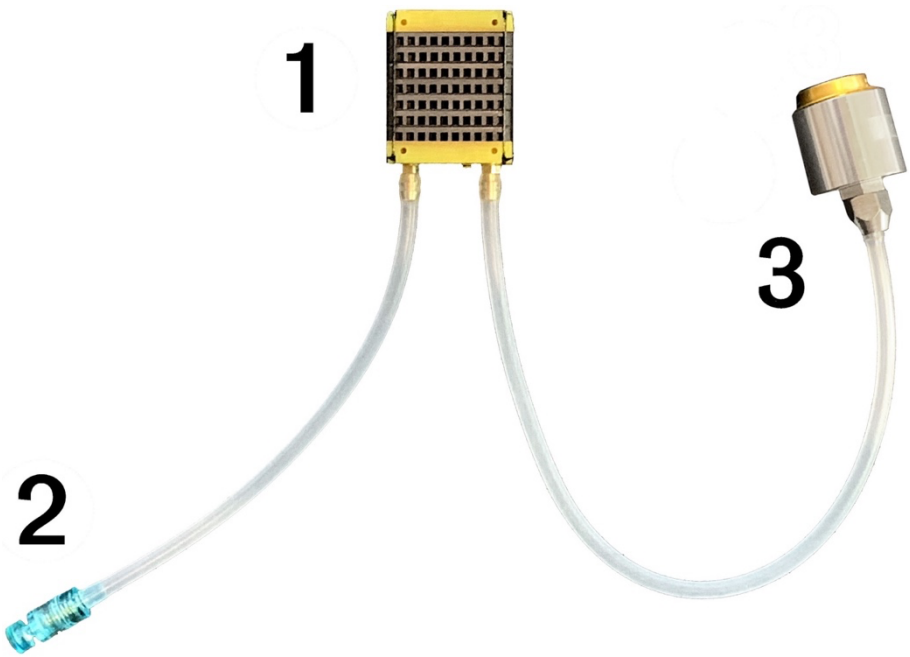
In der Mitte unten ist am linken Pin der NTC-Temperatursensor angeschlossen und am rechten Pin bekommt man 1/10 der Spannung, die hinter dem 9V Spannungswandler (LTC3112) anliegt. Hier kann man einen A/D-Wandler anschließen, um gefahrlos die Spannung hinter dem Spannungswandler zu messen. Der i2c-Bus ist rechts oben und rechts unten am Brennstoffzellen Brick verfügbar.

Die durch die Brennstoffzelle erzeugte Spannung beträgt etwa 3,6 Volt mit rund 700 mW Leistung bei 200 mA Stromstärke über einen Zeitraum von ca. 90 Minuten. Wird mehr Leistung oder höhere Stromstärken verbraucht, reduziert sich die Laufzeit entsprechend.

Gefahrenhinweis: Wasserstoff ist ein brennbares Gas und bildet mit der Luft explosive Mischungen. Die Versuche und auch die Erzeugung von Wasserstoff dürfen daher nur in gut belüfteten Räumen durchgeführt werden.

4.1.1 Brennstoffzelle

Auf unserem Brennstoffzellen-Brick befindet sich die eigentliche (**1**) Brennstoffzelle (Art.-Nr. 183366), welche aus Wasserstoff und Sauerstoff Strom erzeugt. Vom mitgelieferten, 1 Meter langen, Schlauch (Art.-Nr. 130642) sind zwei ca. 15 cm lange Stück abzutrennen und so anzuschließen, wie es in der nachfolgenden Abbildung zu sehen ist. Beide Anschlüsse an der Brennstoffzelle sind identisch, d.h. es ist egal, an welchem Anschluss der (**2**) Entlüftungstaster (Art.-Nr. 183361) oder der (**3**) Druckminderer (Art.-Nr. 183360) angeschlossen wird.



An dem Druckminderer (3) wird zum Schluss der HydroStik Pro Behälter (optional erhältlich) angeschlossen. Mit dem hydraulischen Taster (2) entlüftet man für eine Sekunde das System nach dem Anschluss des HydroStik Pro Behälters, damit sich nur noch Wasserstoff in den Schläuchen befindet.

Die Anschlüsse für die Schläuche haben einen Durchmesser von 3,4 mm. Unsere Brennstoffzelle ist 37,5 mm hoch (ohne Schlauchanschlüsse), 24 mm tief sowie 29 mm breit und wiegt 28 g.

Sobald der HydroStik Pro Behälter mit dem Schlauch an der Brennstoffzelle angeschlossen ist, wird Wasserstoff verbraucht und zwar unabhängig davon, ob ein elektrischer Verbraucher am Brennstoffzellen-Brick angeschlossen ist. Wenn man die Brennstoffzelle nicht benutzt, muss daher der HydroStik Pro Behälter vom Druckminderer getrennt werden. Aus einem nicht angeschlossenen HydroStik Pro Behälter entweicht so gut wie kein Wasserstoff und der Behälter lässt sich tagelang in gefülltem Zustand lagern.

Technische Daten:

Arbeitsspannung: 6 - 10 V • Stromstärke: 200 - 300 mA • Leistung: 1-3 W
Maximaler Wasserstoff-Druck: 1,1146 - 1,2159 bar (111,46 - 121,59 kPa = 16,1655 - 17,6351 PSI)
Notwendiger (regulärer) Wasserstoff-Druck 0,45 bis 0,55 bar (45 - 55 kPa = 6,5 - 8 PSI)
Wasserstoffreinheit (gemäß GOST 3022-80): 99,99 %

4.1.2 Hinweise zur Pflege und Aufbewahrung der Brennstoffzelle

Nach der Benutzung sollte die Brennstoffzelle nicht für längere Zeit offen herumliegen. Stattdessen sollte sie in einem dichten Behälter aufbewahrt werden. Die Brennstoffzelle sollte sich nach der ersten Benutzung so oft wie möglich in Gebrauch befinden. Wenn es notwendig ist, die Brennstoffzelle für einen längeren Zeitraum einzulagern, also mehrere Tage oder Wochen, sollte sie vorher wenn möglich mit Argon oder Stickstoff eine Minute lang unter Druck ausgeblasen werden. Gibt es dazu keine Möglichkeit, sollte die Brennstoffzelle ersatzweise mit Wasserstoff ausgeblasen werden und dann der Wasserstoff in der Brennstoffzelle durch hineingepustete Luft verdrängt werden.

So wird die Brennstoffzelle von Kondenswasser und parasitären Reaktionsprodukten, die sich an der internen Anodenregion der Brennstoffzelle ansammeln, befreit. Beim Betrieb muss die Gasleitung mit dem Drucktaster nach unten zeigen, damit sich das Kondenswasser vor dem Drucktaster sammelt und nicht in direkter Nähe der Brennstoffzelle. Ein Druck von 0,45 - 0,55 bar (45 - 55 kPa) wird während des Betriebs für die Durchströmung der Gasleitungen benötigt. Dadurch wird Kondenswasser aus dem Gasleitungssystem nach draußen gedrückt und kann am Drucktaster entlüftet werden.

Nach der so durchgeführten Reinigung sollen die Gasleitungen mit einem kurzen Stück Schlauch gegeneinander verschlossen werden, um eine weitere Luftzirkulation zu vermeiden. Die Gasmembran in der Brennstoffzelle darf nie komplett austrocknen, da sie dadurch beschädigt würde.

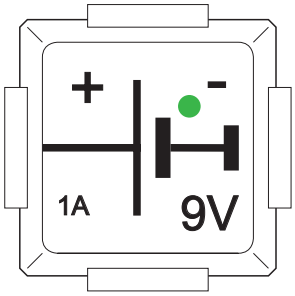
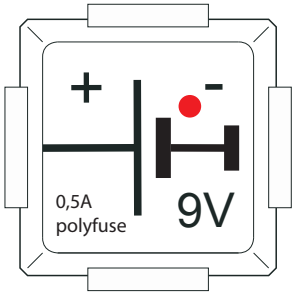
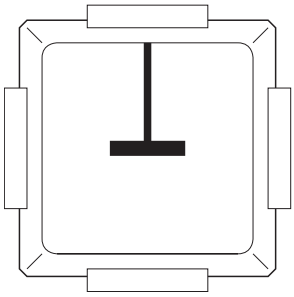
Ein Nutzungszeitraum von mehr als 5.000 Betriebsstunden ist möglich, wenn die Brennstoffzelle ordnungsgemäß gepflegt wird. Eine Lagerung von bis zu einem Jahr ist möglich, aber nicht empfehlenswert. Im Laufe der Zeit erfährt der Katalysator in der Brennstoffzelle einen gewissen Verschleiß durch Sekundäroxidation, während die Brennstoffzelle nicht benutzt wird. Dadurch kommt es zu einer Verminderung bei der Stromerzeugung und die Gesamtlebensdauer der Brennstoffzelle verringert sich ebenfalls. Auch verschlechtert sich die Vorlaufzeit, die notwendig ist, um in den regulären Betriebsmodus zu gelangen.

4.1.3 Hinweise zum Betrieb der Brennstoffzelle

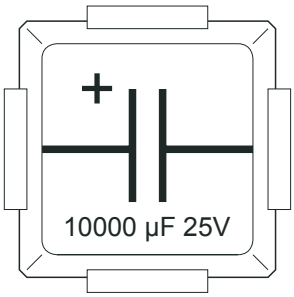
- Es darf nur reiner Wasserstoff verwendet werden und keine verunreinigten Gasgemische.
- Die Brennstoffzelle darf nur dann elektrisch angeschlossen sein, wenn Wasserstoff zugeführt wird.
- Die Brennstoffzelle darf nicht kurz hintereinander ein- und wieder ausgeschaltet werden.
- Die Sauerstoffzufuhr während des Betriebs muss gewährleistet sein.
- Beim Betrieb muss die Gasleitung mit dem Drucktaster nach unten zeigen.
- Die Brennstoffzelle darf nur bei Raumtemperatur (18 - 25 Grad Celsius) betrieben werden.
- Die Brennstoffzelle benötigt für die Wasserstoffzufuhr 0,45 - 0,55 bar (45 - 55 kPa) Gasdruck.
- Der Druckminderer der Brennstoffzelle darf mit maximal 30 bar (3.000 kPa) betrieben werden.
- Die Brennstoffzelle darf nicht mit zu großen Strömen überlastet werden.
- Alkohol und Aceton sowie deren Dämpfe beschädigen die Brennstoffzelle.

Gefahrenhinweis: Wasserstoff ist ein brennbares Gas und bildet mit der Luft explosive Mischungen. Die Versuche und auch die Erzeugung von Wasserstoff dürfen daher nur in gut belüfteten Räumen durchgeführt werden.

4.2 Versorgungs-Bricks

Abbildung	Anzahl	Kurzbeschreibung	Art.-Nr. / Brick-ID
	1	Netzteiladapter 9V 1A Sicherung und Masse	Art.-Nr.: 118627 Brick-ID: ALL-BRICK-0221
		Der Netzadapter liefert eine stabilsierte 9 V Gleichspannung bei maximal 1 A und ist mit einer selbstheilenden Sicherung vom Typ Polyfuse abgesichert. Eine LED zeigt an, sobald der Brick Strom liefert. Der Pluspol ist direkt herausgeführt und der Minuspol mit Masse verbunden, sodass kein weiterer Masse-Brick verwendet werden muss. **Beachte:** Bitte die Schaltung vor jeder Inbetriebnahme kontrollieren, da ansonsten die Gefahr besteht, durch Kurzschluss, Verpolung oder andere Fehler empfindliche Bauteile zu beschädigen! Das Netzteil ist am Ende der Versuchsdurchführung sofort vom Stromnetz zu trennen!	
	1	Batterieadapter 9V 0,5A Sicherung und Masse	Art.-Nr.: 113629 Brick-ID: ALL-BRICK-0002
		Der Batterieadapter verwendet eine 9 V Block-Batterie und versorgt die Schaltung mit einer Gleichspannung von 9 V. Die LED leuchtet rot, sobald die selbstheilende Sicherung vom Typ Polyfuse bei Kurzschluss oder Überlastung (> 0,5 A) den Stromkreis trennt. Der Pluspol ist direkt herausgeführt und der Minuspol mit Masse verbunden, sodass kein weiterer Masse-Brick verwendet werden muss. **Beachte:** Bitte die Schaltung vor jeder Inbetriebnahme kontrollieren, da ansonsten die Gefahr besteht, durch Kurzschluss, Verpolung oder andere Fehler empfindliche Bauteile zu beschädigen!	
	3	Leitung Masse-Brick	Art.-Nr.: 113630 Brick-ID: ALL-BRICK-0003
		Der Masse-Brick stellt den Anschluss zum Bezugspotential „Masse“ her. Mit Hilfe solcher „Masse“-Anschlüsse kann der Stromkreis auf einfache Weise und sehr zuverlässig geschlossen werden ohne eine Vielzahl an Leitungs-Bricks zu benötigen. Zudem werden die Schaltungen dadurch übersichtlicher. Bei allen theoretischen Betrachtungen wird das Potential dieses Leitungsnetzes normalerweise als Bezugspotential 0 Volt definiert. Der Masse-Brick verbindet die beiden mittleren Kontakte des 4-poligen Hermaphrodit-Steckers mit den beiden außen liegenden Massekontakten.	

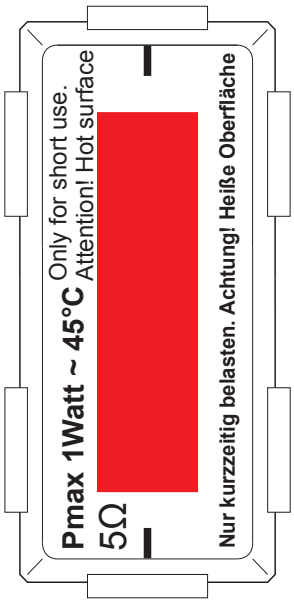
4.3 Kondensator (Kapazität)

Abbildung	Anzahl	Kurzbeschreibung	Art.-Nr. / Brick-ID
	1	Kondensator 10.000 µF	Art.-Nr.: 133745 Brick-ID: ALL-BRICK-0615

Dieser sogenannte Elektrolyt-Kondensator hat eine Kapazität von 10.000 µF, das entspricht zwar 10 mF, aber man gibt die Kapazität von Elektrolyt-Kondensatoren üblicherweise in µF an.

Beachte: Bei diesem Elektrolytkondensator ist auf die richtige Polung zu achten. Der Pluspol (+) darf nur mit der positiven Seite (+) der Spannungsversorgung (9 V) direkt oder indirekt verbunden werden. Dementsprechend darf der Minuspol nur mit der negativen Seite der Spannungsversorgung (Masse) direkt oder indirekt verbunden werden. Er darf nur bis zu einer Spannung von 25 V betrieben werden!

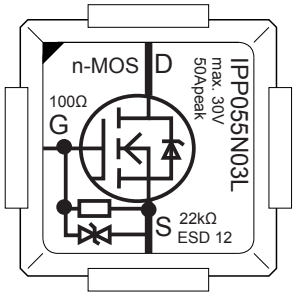
4.4 Widerstand

Abbildung	Anzahl	Kurzbeschreibung	Art.-Nr. / Brick-ID
	1	Lastwiderstand 5 Ω	Art.-Nr.: 142414 Brick-ID: ALL-BRICK-0665

Dieser Brick enthält einen elektrischen Widerstand mit dem Wert 5 Ω. Nach dem ohm'schen Gesetz (R = U / I) entspricht dieser Wert einem Stromfluss von 200 mA bei einer Spannung von 1 V.

Beachte: Die Verlustleistung am Widerstand sollte 1 W nicht übersteigen – sonst droht Überhitzung (~45°C)!

4.5 Transistor

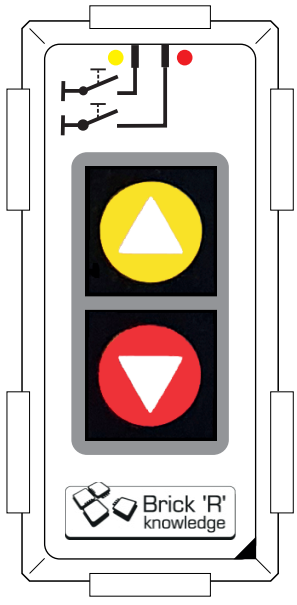
Abbildung	Anzahl	Kurzbeschreibung	Art.-Nr. / Brick-ID
	1	n-MOS Transistor IPP055N03L	Art.-Nr.: 182999 Brick-ID: ALL-BRICK-0717

Dieser Feldeffekt-Transistor steuert den Stromfluss zwischen Drain und Source über die am Gate angelegte Spannung. Das Besondere an diesem Bauelement ist, dass die Verbindung zwischen Gate und Source sehr hochohmig ist. Feldeffekttransistoren werden als „MOSFET“ (Metall-Oxid-Semileitender Feldeffekt-Transistor) bezeichnet oder kurz als „MOS“.

Es gibt unterschiedliche Arten von MOS, dieser ist ein normal sperrender n-Kanal. Dies bedeutet, dass die Schwellspannung am Gate (Tor) anliegen muss, damit ein Stromfluss zwischen Drain (Abfluss) und Source (Quelle) erfahren wird. Die Spannung muss am Gate positiv zur Source sein. Der MOSFET kann bis zu 30V Source-Drain-Spannung vertragen und einen Spitzenstrom von 50A (nicht beides gleichzeitig versteht sich).

Achtung: Die Kontakte des Brickssystems vertragen maximal 6.3A pro Kontakt. Der Gate-Eingang ist mit einer ESD 12V geschützt, sowie einem Widerstand von 22kOhm.

4.6 Taster

Abbildung	Anzahl	Kurzbeschreibung	Art.-Nr. / Brick-ID
	1	Folientaster-Brick zweifach mit Pfeilen up/down	Art.-Nr.: 183342 Brick-ID: ALL-BRICK-0723

Jeder der beiden Taster ist jeweils als einpoliger Schließer ausgeführt. Während des Drückens wird eine elektrische Verbindung zwischen Anschluss und Masse hergestellt. Der gelbe Taster schließt den Kontakt links oben, der rote Taster schließt den Kontakt rechts oben.

Beachte: Ein Taster ist ein elektromechanisches Bauteil und liefert kein entprelltes Signal.

4.7 Leitungs-Bricks

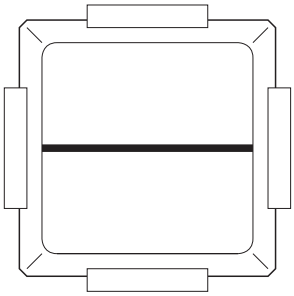
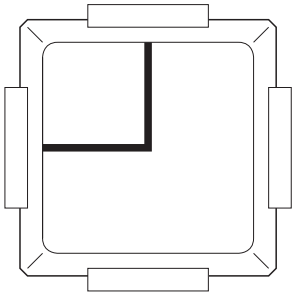
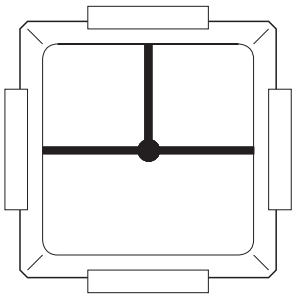
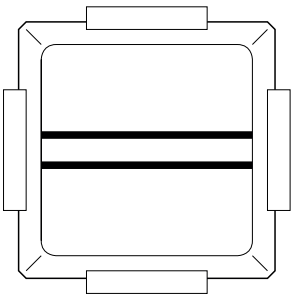
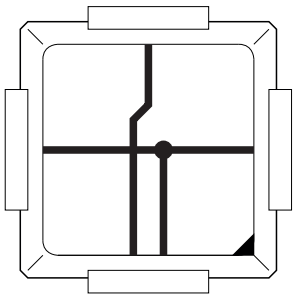
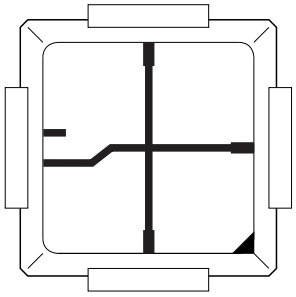
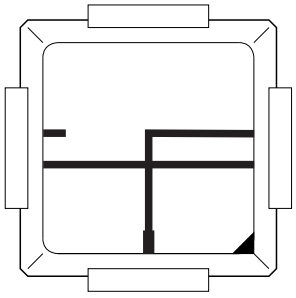
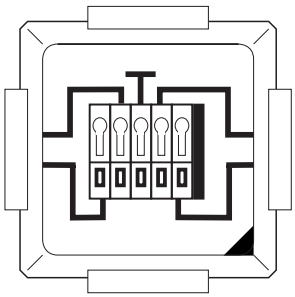
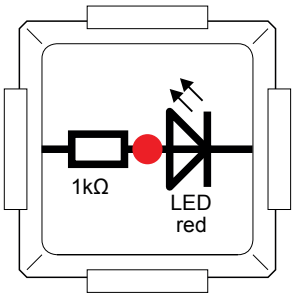
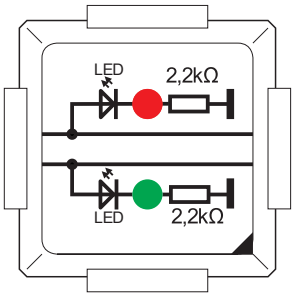
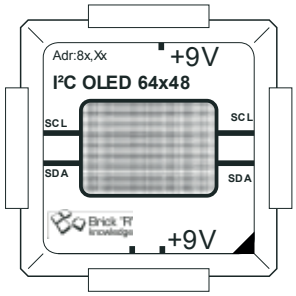
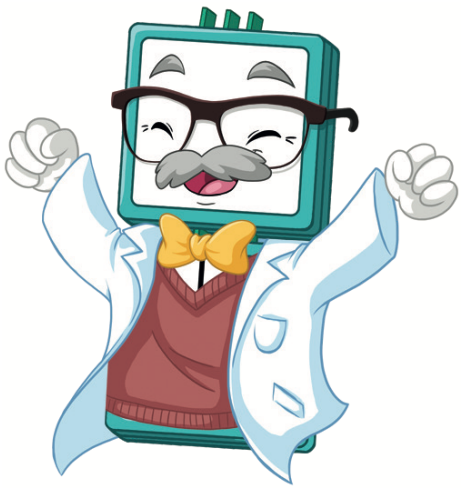
Abbildung	Anzahl	Kurzbeschreibung	Art.-Nr. / Brick-ID
	2	Leitung Gerade	Art.-Nr.: 113631 Brick-ID: ALL-BRICK-0004
Die gerade Leitung verbindet zwei gegenüberliegende Anschlüsse miteinander.			
	4	Leitung Ecke	Art.-Nr.: 113632 Brick-ID: ALL-BRICK-0005
Mit dem Eck-Brick werden zwei angrenzende Seiten miteinander verbunden.			
	3	Leitung T-Kreuzung	Art.-Nr.: 113633 Brick-ID: ALL-BRICK-0006
Mit der T-Kreuzung werden Abzweigungen hergestellt. Dieser Brick kann auch anstelle eines Eck-Bricks verwendet werden.			
	1	Leitung Doppelt	Art.-Nr.: 113676 Brick-ID: ALL-BRICK-0049
Gerade Verbindung für zwei getrennt geführte Signale auf den beiden inneren Kontakten.			
	1	Leitung Doppelspannung für Leistungstreiber	Art.-Nr.: 134017 Brick-ID: ALL-BRICK-0616
Dieser Spezial-Leitungs-Brick erlaubt die Verteilung getrennt geführter Signale und den Übergang von getrennter Signalführung auf Einzelleitungen.			

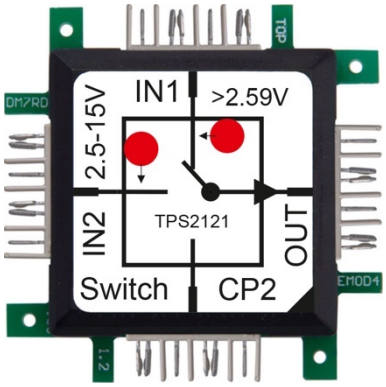
Abbildung	Anzahl	Kurzbeschreibung	Art.-Nr. / Brick-ID
	1	Leitung als Kreuzung mit einfacher Trennung der Leitung	Art.-Nr.: 182906 Brick-ID: ALL-BRICK-0715
		Dieser Spezial-Leitungs-Brick erlaubt die Verteilung getrennt geführter Signale und den Übergang von getrennter Signalführung auf Einzelleitungen.	
	1	Leitung als Winkel und Gerade für die Einzelpins	Art.-Nr.: 182907 Brick-ID: ALL-BRICK-0716
		Dieser Spezial-Leitungs-Brick erlaubt die Verteilung getrennt geführter Signale und den Übergang von getrennter Signalführung auf Einzelleitungen.	
	1	5-polige Klemme Typ 2	Art.-Nr.: 125674 Brick-ID: ALL-BRICK-0407
		Mit der Klemme kann man Leitungen oder Bauteile befestigen und an die Schaltung anschließen. 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.	

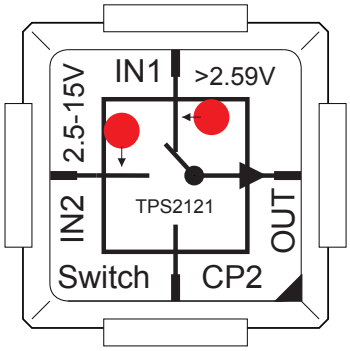
4.8 Anzeige- und LED-Bricks

Abbildung	Anzahl	Kurzbeschreibung	Art.-Nr. / Brick-ID
	1	LED rot	Art.-Nr.: 113636 Brick-ID: ALL-BRICK-0009
Dieser Brick hat eine rote Leuchtdiode mit einem eingebauten Vorwiderstand von 1 kΩ. Sie ist in Durchlassrichtung zu betreiben. Aber auch richtig gepolt (in Durchlassrichtung) kommt erst ein Stromfluss zustande, wenn eine Mindestspannung von ca. 1,5 V erreicht wird. Die Mindestspannung von Leuchtdioden ist von ihrer Farbe abhängig.			
	1	Dual-LED auf Masse (rot/grün)	Art.-Nr.: 162698 Brick-ID: ALL-BRICK-0701
In diesem Baustein sind zwei LEDs (rot und grün) untergebracht, die intern mit Masse verbunden sind. Die Signalleitungen sind getrennt durchverbunden. Beide LEDs sind über einen 2,2 kΩ Widerstand vor zu hohem Stromfluss geschützt. Sie sind für 2 mA Strom bei 5 V Spannung ausgelegt. Beide Widerstände sind intern mit Masse verbunden, wodurch man den Baustein direkt anschließen kann und kein weiterer Masse-Brick verwendet werden darf.			
	1	I²C OLED Display 64 x 48 Pixel	Art.-Nr.: 118430 Brick-ID: ALL-BRICK-0182
Organische Leuchtdiode (OLED), genaugenommen sind 64 x 48 Leuchtdioden in einer Matrix angeordnet und können einzeln mit Hilfe von Befehlen über den I²C angesteuert werden. Damit lassen sich mehrzeilige Texte darstellen, aber auch einfache monochrome Grafiken.			



4.9 Strompfad-Weiche Brick (spannungsgesteuert)





ALLNET Brick'R'knowledge
Strompfad-Weiche Brick
(spannungsgesteuert)

Hersteller-Nummer:
ALL-BRICK-0712

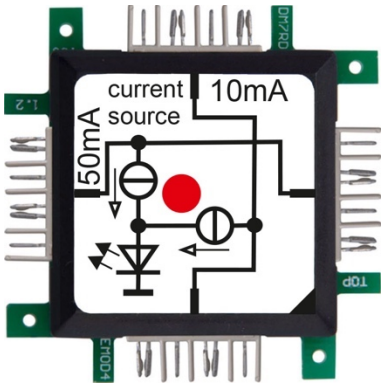
ALLNET-Art.-Nr.:
180232

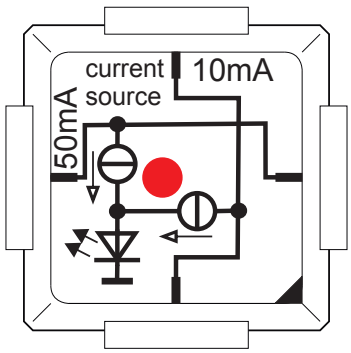
Unser Strompfad-Weiche Brick dient zur Stromversorgung des Brennstoffzellen-Bricks und wird wahlweise von einer festen Stromquelle wie z.B. ein Netzteil-Brick oder vom Brennstoffzellen-Brick selbst mit Strom versorgt. Liefert der Brennstoffzellen-Brick zu wenig Strom, dann erfolgt automatisch eine Umschaltung auf die Netzteil-Stromversorgung. Dadurch ist gewährleistet, dass der Brennstoffzellen-Brick mit seinen zusätzlichen Funktionen wie z.B. die Temperaturmessung durchgängig funktioniert.

Der Brennstoffzellen-Brick wird an in1 (oben) angeschlossen, der Netzteil-Brick wird an in2 (links) angeschlossen. Der Ausgang des USV-Bricks (rechte Seite) wird an die Stromversorgung V_{in} des Brennstoffzellen-Bricks angeschlossen. Sobald die Spannung vom Eingang in1 unter 2,59 Volt fällt, erfolgt eine Umschaltung auf den Eingang i2.

Wenn Cp2 (unten) mit 3,3 Volt verbunden wird, dann erfolgt eine zwangsweise Umschaltung auf in2, auch wenn an in1 eine Spannung von mehr als 2,59 Volt anliegt. Hierdurch ist es möglich mit einem Taster oder über einen Arduino die Umschaltung zu steuern. Die Umschaltung funktioniert nur, wenn auch eine aktive Stromquelle an in2 angeschlossen ist, sonst wird cp2 ignoriert.

4.10 Konstantstrom Dual Last Brick (10 und 50mA)





ALLNET Brick'R'knowledge
Konstantstrom Dual Last Brick
(10 und 50mA)

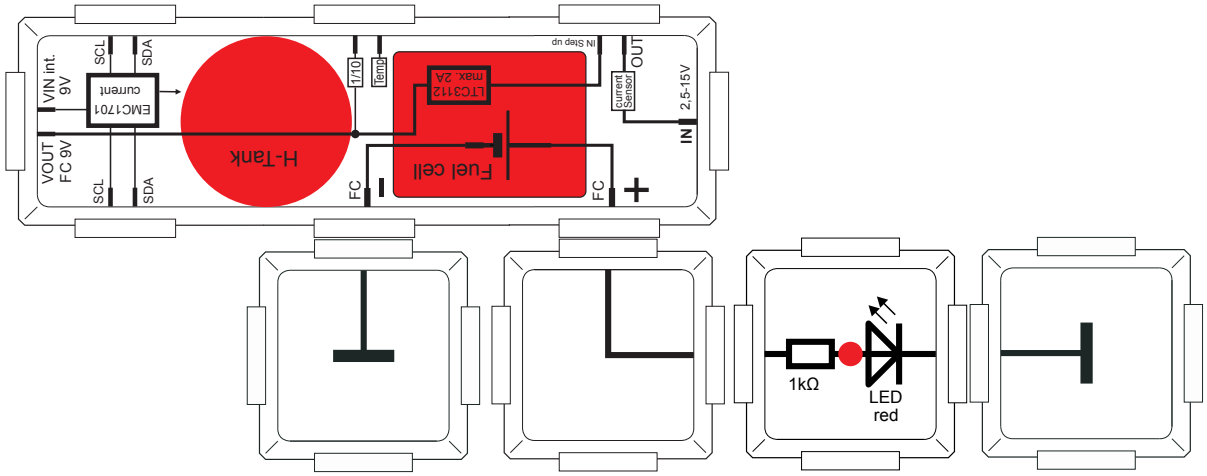
Hersteller-Nummer:
ALL-BRICK-0713

ALLNET-Art.-Nr.:
180233

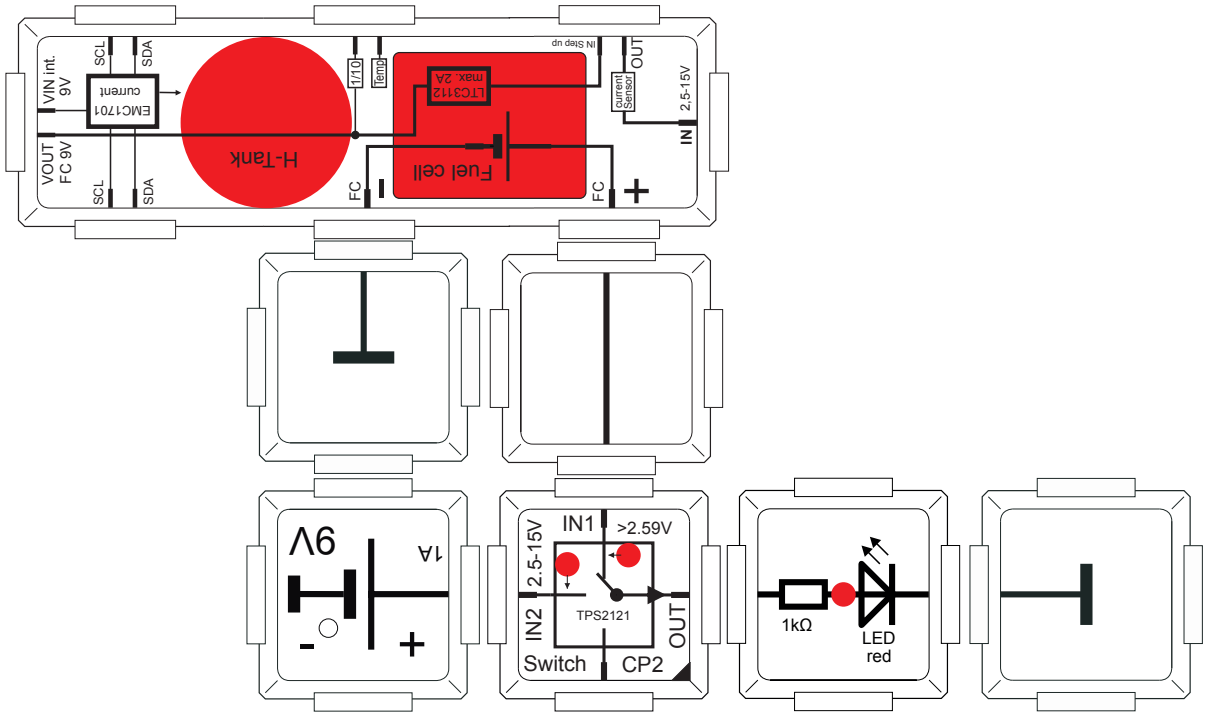
Unser Konstantstrom Dual Last Brick dient zur kontinuierlichen Entnahme einer definierten Menge an Strom. Die Eingänge oben und unten verbrauchen 10 mA, die Eingänge links und rechts verbrauchen 50 mA. Ein separater Masseanschluss ist nicht notwendig, da eine interne Masseverbindung besteht. Eine Leuchtdiode zeigt an, sobald Strom verbraucht wird.

4.11 Inbetriebnahme des Brennstoffzellen-Bricks

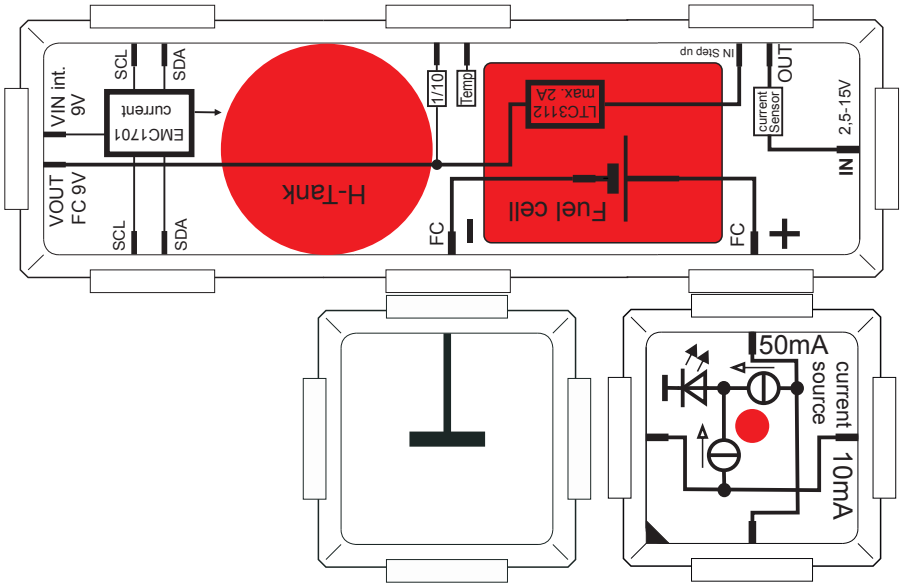
Im Folgenden bauen wir einige Schaltungen auf, um den Brennstoffzellen-Brick zu testen. Dazu benötigen wir außer dem Brennstoffzellen-Brick noch eine Wasserstoffquelle, die mit der Brennstoffzelle verbunden werden muss. Der Sauerstoff wird der Umgebungsluft entnommen und braucht nicht extra zugeführt zu werden. Als Wasserstoffquelle empfehlen wir einen HydroStik Pro Behälter, der über eine HydroStik Pro Ladestation mit Wasserstoff befüllt wird. Unsere erste Schaltung nutzt die Brennstoffzelle als Energiequelle zum Betreiben einer Leuchtdiode. Der Stromverbrauch beträgt ca. 4 mA.



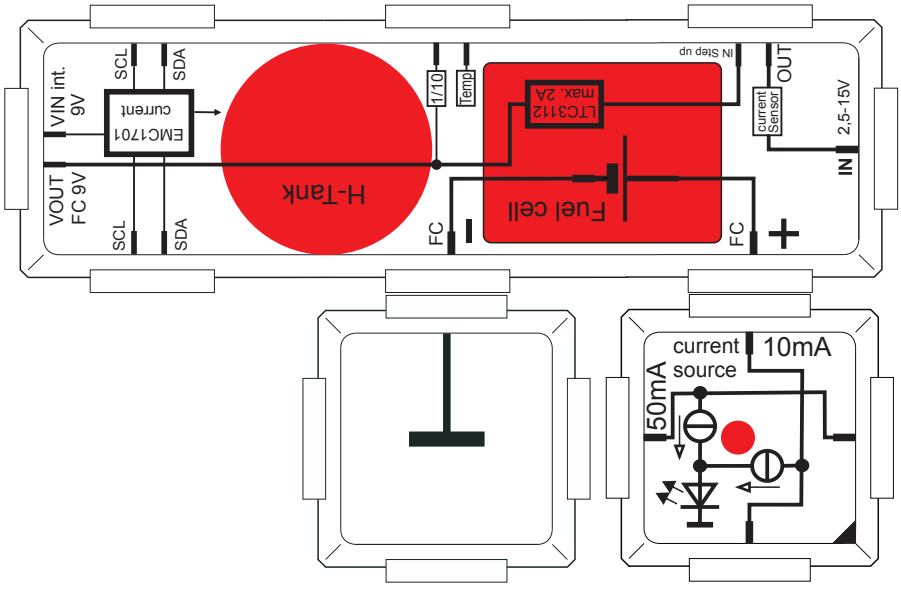
Als Nächstes wird die spannungsgesteuerte Strompfad-Weiche Brick in die Schaltung eingebaut. Wir betreiben die Schaltung zunächst mit angeschlossenen Brennstoffzelle. Die grüne LED auf der rechten Seite leuchtet und zeigt an, dass die Brennstoffzelle mindestens 2,59 Volt Spannung erzeugt. Jetzt entfernen wir die Wasserstoffzufuhr. Es dauert einige Zeit, bis der restliche in der Brennstoffzelle vorhandene Wasserstoff verbraucht ist. Danach schaltet der Strompfad-Weiche Brick auf das Netzteil um, die grüne LED erlischt und die rote LED auf der linken Seite des Strompfad-Weiche Bricks leuchtet. Die Brennstoffzelle liefert nun keine Spannung mehr und der LED-Brick wird über das Netzteil gespeist. Jetzt schließen wir die Wasserstoffzufuhr wieder an. und die grüne LED zeigt uns nach kurzer Zeit an, dass die Brennstoffzelle wieder mindestens 2,59 Volt Spannung erzeugt.



Jetzt wollen wir den Konstantstrom Dual Last Brick anschließen, um zu messen, wie lange es dauert, bis der Inhalt eines HydroStik Pro Behälter bei Entnahme von kontinuierlichen 50 mA leer ist. Wenn man den Stromfluss messen kann, wird man sehen, dass in Wirklichkeit ca. 40 mA verbraucht werden. Das liegt daran, dass die Brennstoffzelle nur eine geringe Spannung von etwa 3,5-5,0 Volt liefert und unser Konstantstrom Dual Last Brick erst bei einer höheren Spannung 50 mA verbraucht.



Ein vollständig gefüllter HydroStik Pro Behälter sollte ausreichen, um diese Schaltung etwa 5 Stunden lang betreiben zu können. Man kann die genaue Zeit messen und danach den Versuchsaufbau durch Drehung des Konstantstrom Dual Last Bricks verändern, dass jetzt kontinuierliche 10 mA verbraucht werden (messtechnisch ca. 8 mA). Wenn man die Möglichkeit dazu hat, kann man mit einem Thermometer messen, welche Temperatur die Brennstoffzelle vor und nach dem Verbrauch hat.



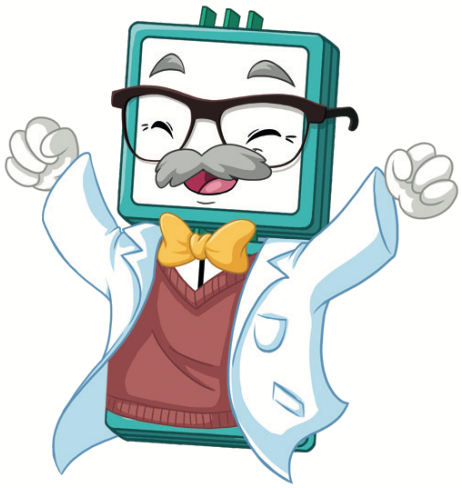
Als Erstes fällt auf, dass die LED im Konstantstrom Dual Last Brick nicht mehr so hell leuchtet, wie im vorigen Versuchsaufbau. Das liegt daran, dass jetzt viel weniger Strom, nämlich nur noch 20% der zuvor entnommenen Menge, verbraucht wird. Auch diese Schaltung sollte etwa 5 Stunden lang betrieben werden können. Wenn man die Brennstoffzelle ohne angeschlossene Verbraucher betreibt, ver-

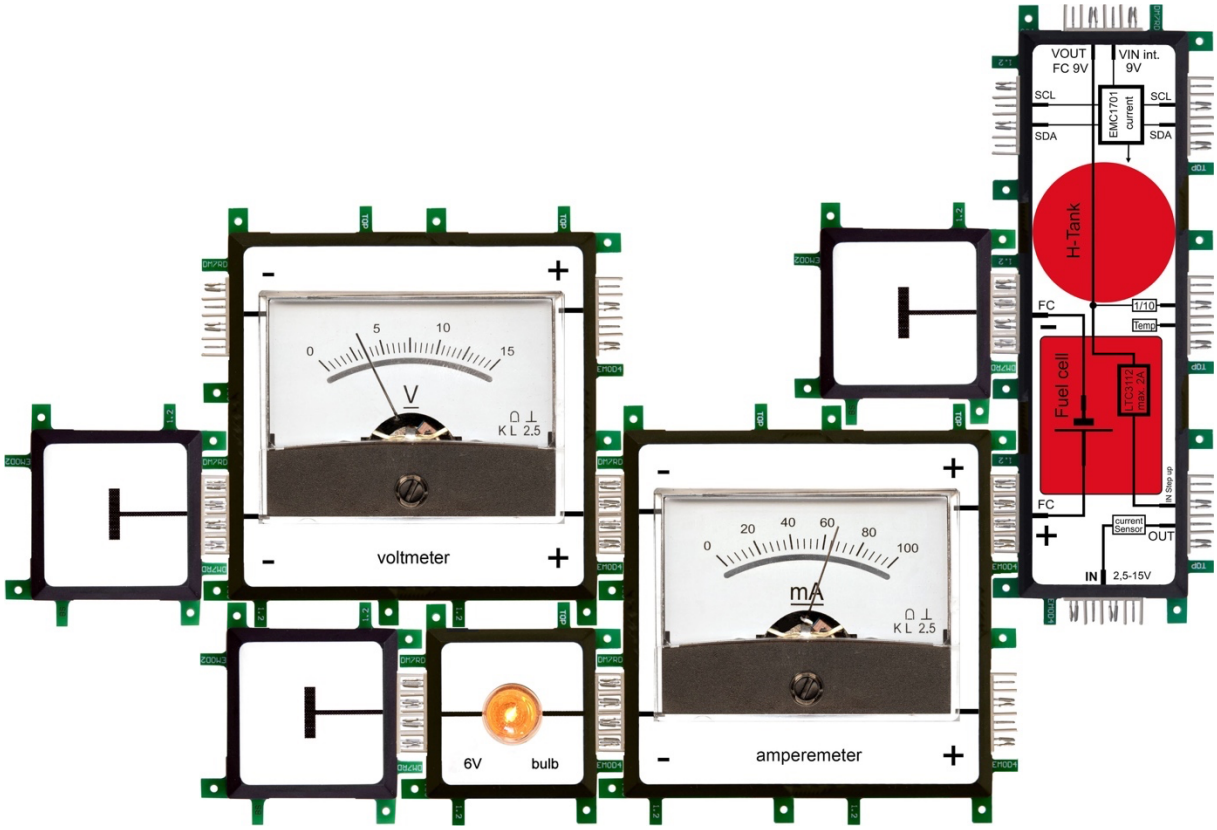
ringert sich die Zeit bis zur Entleerung des Wasserstofftanks ebenfalls nicht signifikant. Das liegt einfach daran, dass immer dieselbe Menge Wasserstoff durch die Zelle strömt und in Energie umgewandelt wird.

Unsere Brennstoffzelle hat keine Elektronik oder sonstige Funktion, mit der eine verbrauchsgerechte Steuerung der Wasserstoffmenge möglich ist. Daher sollte man, sobald ein Versuch beendet ist, die Wasserstoff-Zufuhr zur Brennstoffzelle unterbrechen und zwar durch Entfernen des Druckminderers, damit sich der Wasserstofftank nicht selbstständig und ohne Nutzung entleert. Ein nicht verbundener Wasserstofftank kann hingegen mehrere Wochen lang gelagert werden, ohne dass nennenswerte Mengen an Wasserstoff entweichen.

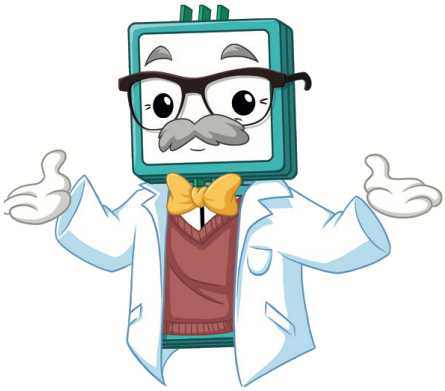
Was passiert in der Brennstoffzelle wenn kein Verbraucher angeschlossen ist, aber trotzdem kontinuierlich Wasserstoff verbraucht wird? Es handelt sich ja um ein geschlossenes System, es kann also kein Wasserstoff entweichen, auch nicht das in der Brennstoffzelle erzeugte Wasser, welches nach einiger Zeit im Schlauch vor dem hydraulischen Taster in kleinen Tröpfchen kondensiert. Die Energie, welche nicht als Strom entnommen wird, gibt die Brennstoffzelle als Wärme ab. Daher ist es interessant zu messen, wie die Temperatur der Brennstoffzelle sich verhält in Abhängigkeit davon, ob und wie viel Strom entnommen wird.

Mit Bricks aus anderen Sets und Messinstrumenten lassen sich noch sehr viel mehr Experimente realisieren, im Prinzip ist fast jedes Experiment möglich, wenn man das ursprünglich benötigte Netzteil bzw. die Batterie durch den Brennstoffzellen-Brick ersetzt. Das Einzige, was man dabei beachten muss, ist die zur Verfügung stehende Spannung von nur etwa 3,5 V im Gegensatz zu den 9 V eines Netzteils bzw. einer Batterie. In Abhängigkeit des angeschlossenen Verbrauchers lassen sich an der Brennstoffzelle maximal knapp über 65 mA Strom bei ca. 3,5 V entnehmen. Das entspricht einer Leistung von knapp 230 mW. Die folgende Schaltung zeigt eine Versuchsanordnung mit Voltmeter und Amperemeter sowie einer Glühlampe für maximal 6 V Spannung als Verbraucher.



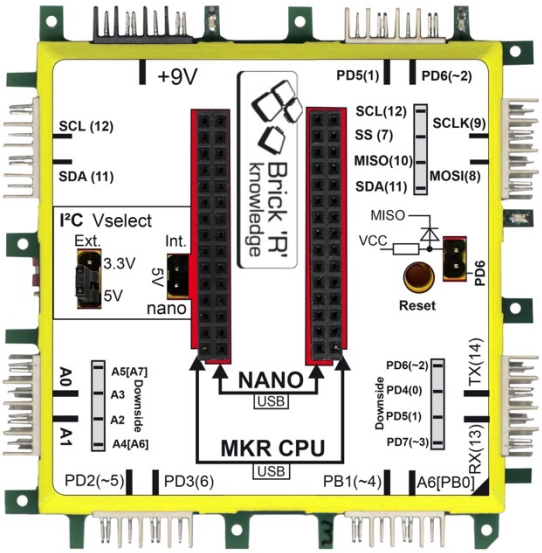


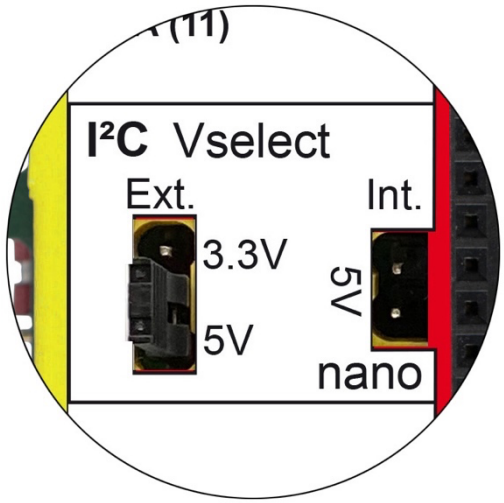
Die auf dieser Seite abgebildeten Meßinstrumente sowie die Glühlampe sind nicht Bestandteil dieses Sets. Um Ströme und Spannungen zu messen, kann alternativ ein handelsübliches Handheld-Meßinstrument (Voltmeter/Amperemeter) verwendet werden. Im Fachhandel gibt es jede Menge Bricks zu kaufen, mit denen man die hier vorgestellten Experimente variieren und erweitern kann. Die oben gezeigte Abbildung soll dabei anregen, eigene Experimente mit optionalen Bricks aufzubauen. Im Anhang dieses Handbuchs sind alle aktuell verfügbaren Brick-Sets aufgelistet.



5. MKR Projekte - Einstieg in die Arduino Welt

5.1 Arduino-MKR-Adapter-Brick

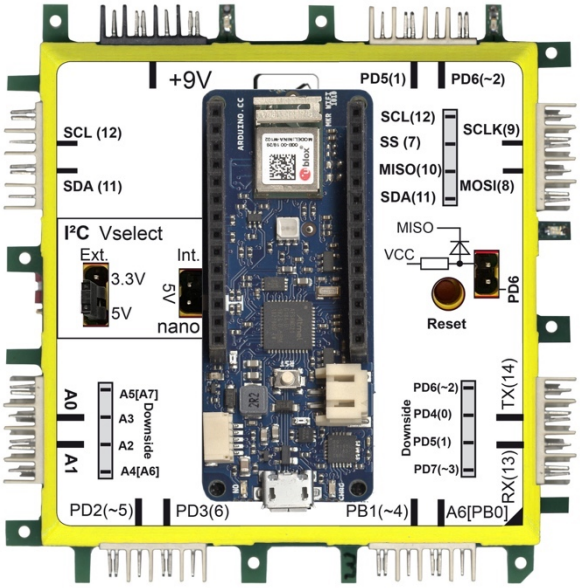




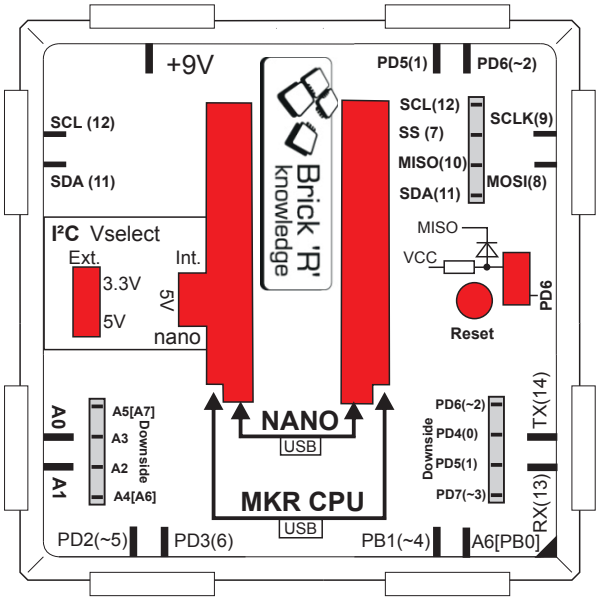
Der Arduino-MKR-Adapter-Brick kann mit allen Arduino-Boards der MKR Serie sowie dem Arduino-Nano betrieben werden. Wenn der Arduino-MKR-Adapter-Brick mit einem 5V kompatiblen externen I2C-Baustein betrieben werden soll, dann muss der Spannungs-Switch für den I2C-Bus (auf der linken Seite des Arduino-MKR-Adapter-Bricks, siehe rechte Abbildung) auf 5V gestellt werden. Da alle originalen I2C-Bausteine von Brick'R'knowledge mit 5V arbeiten, ist das die richtige Jumper-Stellung, wenn man ausschließlich Original-Bricks verwendet. Diese 5V-Kompatibilität gilt aber nur für den i2C-Bus und nicht für die restlichen I/O-Leitungen, die mit der Voltzahl der verwendeten Platine arbeiten, also mit 3,3V bei allen MKR-Platinen und mit 5V bei Verwendung einer Arduino-Nano Platine. Der zweite, rechte Jumper, der mit „nano 5V" beschriftet ist, kann offen oder geschlossen bleiben, da der Arduino-MKR-Brick automatisch erkennt, ob ein Arduino-Nano oder eine Platine aus der Arduino-MKR-Serie verwendet wird. Nur für den speziellen Fall, dass man einen Arduino-Nano-kompatiblen Baustein verwendet, der ausschließlich mit 3,3V arbeitet, muss dieser Jumper entfernt werden. Für die genaue Funktion schau dir bitte dazu das Schaltbild des Arduino-MKR-Adapter-Bricks am Ende dieses Handbuchs an.

Der USB Port kann zur Stromversorgung sowie zur seriellen Kommunikation und zum Programmieren des Arduino genutzt werden. Anders als die meisten anderen Arduino-Platinen, werden die Arduino-MKR-Platinen mit 3,3 Volt betrieben. Höhere Spannungen als 3,3 Volt können dabei die Arduino-MKR Platinen beschädigen. Während die Ausgabe an 5 Volt digitale Geräte möglich ist, kann die bidirektionale Kommunikation mit 5 Volt Geräten nur dann problemlos funktionieren, wenn Level Shifting (damit 5 Volt Bricks, die extern angeschlossen werden, mit den 3,3 Volt der aufgesteckten MKR-CPU zusammen arbeiten können) angewendet wird. Wenn man eine 9V Quelle zur Stromversorgung verwendet, kann das USB-Kabel entfernt werden.

Achtung: Die Eingänge des Arduino sollten nie in direkten Kontakt mit der 9V Quelle kommen, da sie trotz Schutzschaltungen auf der Platine nur für maximal 5V ausgelegt sind!



Bitte das Arduino-MKR-1010-Board (ALLNET-Art.-Nr.: 157817) wie es auf der Abbildung links zu sehen ist, auf den Arduino-MKR-Adapter-Brick setzen. Der 2x2 Arduino-MKR-Adapter-Brick enthält dabei nicht nur die Stromversorgung für den Arduino MKR, sondern auch die Elektronik, die für das Level Shifting notwendig ist. Alle für die Experimente relevanten I/O-Leitungen sind dabei nach außen geführt, um im Brick'R'knowledge System verwendet werden zu können.



ALLNET Brick'R'knowledge MKR Brick

Hersteller-Nummer: ALL-BRICK-0707

ALLNET-Art.-Nr.: 172781

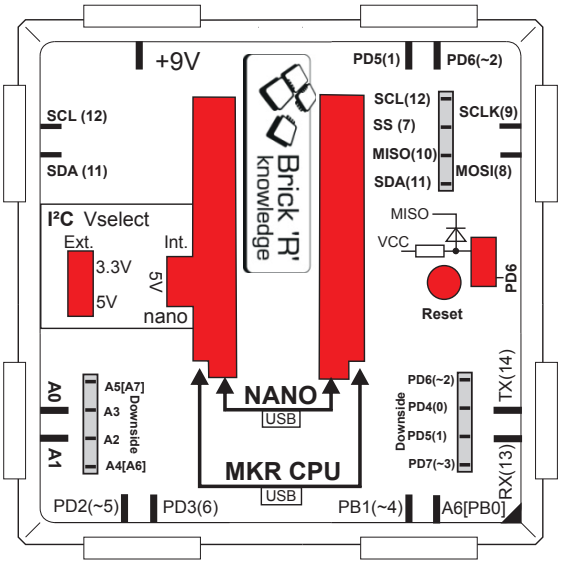
I/O Zuordnung

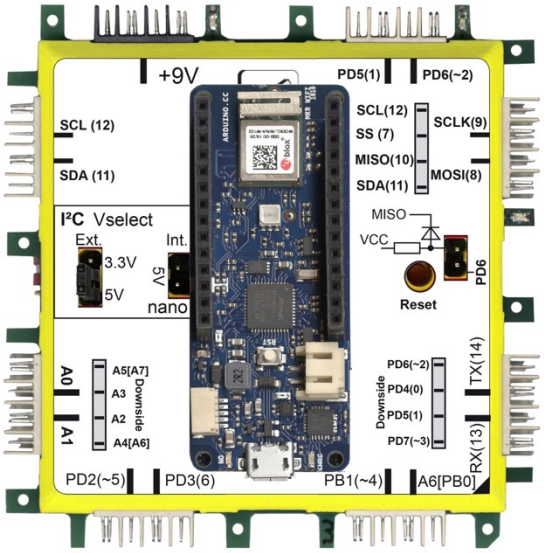
PD0 - 0	A0 - Analog 0
PD1 - 1	A1 - Analog 1
PD2 - 2 (5)	A2 - Analog 2
PD3 - 3 (6)	A3 - Analog 3
PD4 - 4 (0)	A4 - Analog 4
PD5 - 5 (1)	A5 - Analog 5
PD6 - 6 (2)	A6 - Analog 6
PD7 - 7 (3)	A7 - Analog 7
PB1 - 9 (4)	

Portbelegung		**Numerischer (MKR)**
Kürzel	**Port**	**Arduino-Parameter**
SCK	PB5	(9)
MOSI	PB3	(8)
SS	PB2	(7)
MISO	PB4	(10)
SCL	A5	(12)
SDA	A4	(11)
TX	PD1	(14)
RX	PD0	(13)

5.2 Die Arduino MKR Boards

Der Arduino MKR ist das Herz für unseren Arduino-MKR-Adapter-Brick. Der Prozessor wird oben auf die Steckleisten gesetzt. Die Programmierung erfolgt wie bei Arduino üblich, über den PC mit dem Arduino Entwicklungssystem, dass man sich von der Arduino-Homepage entsprechend runterladen kann. Achtung: Die Eingänge des Arduino dürfen nie in direkten Kontakt mit der 9V Quelle kommen, da sie trotz Schutzschaltungen auf der Platine nur für 3,3V (bzw. 5V beim Arduino Nano) ausgelegt sind.





5.2.1 Arduino Nano

Der Arduino Nano ist ein kleines, vollständiges und Steckbrett-freundliches Board, das auf dem ATmega328 (Arduino Nano 3.x) basiert. Es hat mehr oder weniger die gleiche Funktionalität des Arduino Duemilanove, aber in einem anderen Formfaktor. Es benötigt nur einen Stromanschluss und es kann ein Mini-B-USB-Kabel zur Stromversorgung und Programmierung verwendet werden. Achtung: Diese Platine verfügt über I/O-Leitungen, die mit 5V kompatibel sind.

5.2.2 Arduino MKR FOX 1200

MKR FOX 1200 ist ein leistungsstarkes Board, das die Funktionen der Zero- und SigFox-Konnektivität kombiniert. Es ist die ideale Lösung für Anwender, die IoT-Projekte mit minimalen Vorkenntnissen in der Vernetzung von Geräten mit geringem Stromverbrauch entwerfen möchten. Sie erhalten mit dem Kauf des Arduino Boards ein einjähriges kostenloses Abonnement für das Sigfox-Netzwerk mit der Karte (für bis zu 140 Nachrichten pro Tag) sowie kostenlosen Zugriff auf den Geolocation-Service von Spot'it, mit dem Sie das Board ohne GPS oder zusätzliche Hardware verfolgen können. Achtung: Diese Platine verfügt über I/O-Leitungen, die nur mit 3,3V kompatibel sind.

5.2.3 Arduino MKR GSM 1400

Der Arduino MKR GSM 1400 wurde entwickelt, um eine praktische und kostengünstige Lösung für Anwender zu bieten, die ihre Projekte mit globaler GSM-Konnektivität ausstatten möchten. Es basiert auf dem Atmel SAMD21 und einem SARAU201 GSM-Modul. Das Design bietet die Möglichkeit, die Platine mit einem LiPo-Akku oder einer externen Stromquelle mit 5 V zu versorgen. Das Umschalten von einer Quelle zur anderen erfolgt automatisch. Eine schnelle 32-Bit-Rechenleistung, die der Zero-Platine ähnelt, der gewohnt umfangreiche Satz an I/O-Schnittstellen, die globale GSM-Kommunika-

tion und die einfache Verwendung der Arduino Software (IDE) für die Programmierung. All diese Eigenschaften machen dieses Board zur bevorzugten Wahl für die aufkommenden IoT-Projekte mit Batteriebetrieb in einem kompakten Formfaktor. Über den USB-Anschluss kann die Platine mit 5 V versorgt werden. Achtung: Diese Platine verfügt über I/O-Leitungen, die nur mit 3,3V kompatibel sind.

5.2.4 Arduino MKR NB 1500

Mit dem Arduino MKR NB 1500 fügst du deinem Projekt eine Schmalbandkommunikation hinzu. Dies ist die perfekte Wahl für Geräte an entfernten Standorten ohne Internetverbindung oder in Situationen, in denen keine Stromversorgung zur Verfügung steht (z. B. Feldüberwachungssysteme). Dieses Board ist für den weltweiten Einsatz konzipiert und ermöglicht die Nutzung der von Cat M1 / NB1 bereitgestellten Bänder 2, 3, 4, 5, 8, 12, 13, 20, 28 (Vodafone, AT & T, T-Mobile USA, Telstra, Verizon). Achtung: Diese Platine verfügt über I/O-Leitungen, die nur mit 3,3V kompatibel sind.

5.2.6 Arduino MKR Vidor 4000

Den MKR VIDOR 4000 kannst du nach deinen Wünschen konfigurieren. Du kannst damit deine eigenen Baugruppen erstellen. Er ist mit leistungsfähiger Hardware und viel Potenzial ausgestattet: Ein 8-MB-SRAM, ein 2 MB QSPI-Flash-Chip und 1 MB für Benutzeranwendungen. Weiterhin ein Micro-HDMI-Anschluss, ein MIPI-Kameraanschluss und Wifi & BLE aus der U-BLOX NINA W10 Serie. Er enthält auch die klassische MKR-Schnittstelle, an der alle Pins sowohl vom SAMD21 als auch vom FPGA angesteuert werden. Außerdem verfügt er über einen Mini-PCI-Express-Anschluss mit bis zu 25 vom Benutzer programmierbaren Pins. Das FPGA enthält 16K Logic Elements, 504 KB Embedded RAM und 56 18x18-Bit-HW-Multiplikatoren für Hochgeschwindigkeits-DSP-Anwendungen. Jeder Pin kann auf über 150 MHz umgeschaltet und für Funktionen wie UARTs, (Q) SPI, hochauflösendes/hochfrequentes PWM, Quadratur-Encoder, I2C, I2S, Sigma Delta DAC usw. konfiguriert werden Kann auch für High-Speed-DSP-Vorgänge für die Audio- und Videoverarbeitung verwendet werden. Dieses Board verfügt über einen Microchip SAMD21. Die Kommunikation zwischen FPGA und SAMD21 erfolgt nahtlos. Achtung: Diese Platine verfügt über I/O-Leitungen, die nur mit 3,3V kompatibel sind.

5.2.7 Arduino MKR WAN 1300

Der Arduino MKR WAN 1300 wurde entwickelt, um eine praktische und kostengünstige Lösung für Anwender zu bieten, die ihren Projekten Lo-Ra-Konnektivität hinzufügen möchten, ohne dass sie zuvor über minimale Netzwerkerfahrung verfügen müssen. Er basiert auf dem Atmel SAMD21 und einem Murata CMWX1ZZABZ Lo-Ra-Modul. Das Design beinhaltet die Möglichkeit, die Platine mit zwei 1,5 V AA- oder AAA-Batterien oder externen 5 V zu versorgen. Das Umschalten von einer Quelle zur anderen erfolgt automatisch. Eine schnelle 32-Bit-Rechenleistung ähnlich der Zero-Platine, die übliche Fülle an I/O-Schnittstellen, die Lo-Ra-Kommunikation mit geringem Stromverbrauch und die einfache Verwendung der Arduino Software (IDE) für die Programmierung. All diese Eigenschaften machen dieses Board zur bevorzugten Wahl für die aufkommenden IoT-Projekte mit Batteriebetrieb in einem kompakten Formfaktor. Über den USB-Anschluss kann die Platine mit 5 V versorgt werden. Der Arduino MKR WAN 1300 kann mit oder ohne angeschlossene Batterien betrieben werden und hat einen niedrigen Stromverbrauch. Achtung: Diese Platine verfügt über I/O-Leitungen, die nur mit 3,3V kompatibel sind.

5.2.8 Arduino MKR WIFI 1000

Der Arduino MKR1000 wurde entwickelt, um eine praktische und kostengünstige Lösung für Anwender zu bieten, die ihren Projekten WiFi-Konnektivität hinzufügen möchten, ohne zuvor Erfahrung mit

Netzwerken gesammelt zu haben. Es basiert auf dem Atmel ATSAMW25 SoC (System on Chip), das Teil der SmartConnect-Familie von Atmel Wireless-Geräten ist, die speziell für IoT-Projekte entwickelt wurden. Der ATSAMW25 enthält auch eine einzelne 1x1-Stream-PCB-Antenne. Das Design enthält eine Li-Po-Ladeschaltung, durch die der Arduino MKR 1000 mit Batteriestrom oder externen 5 V betrieben werden kann, wobei der Li-Po-Akku bei externer Stromversorgung geladen wird. Das Umschalten von einer Quelle zur anderen erfolgt automatisch. Eine schnelle 32-Bit-Rechenleistung ähnlich der Zero-Platine, die üblichen umfangreichen I/O-Schnittstellen, ein WiFi mit geringem Stromverbrauch und ein Cryptochip für sichere Kommunikation und die Benutzerfreundlichkeit der Arduino Software (IDE) für die Programmierung. All diese Funktionen machen dieses Board zur bevorzugten Wahl für die aufstrebenden IoT-Projekte mit Batteriebetrieb in einem kompakten Formfaktor. Der USB-Anschluss kann zur Stromversorgung (5 V) des Boards verwendet werden. Der Arduino MKR 1000 kann mit oder ohne angeschlossen Li-Po-Akku betrieben werden und hat einen begrenzten Stromverbrauch. Das MKR 1000 Wifi-Modul unterstützt das Zertifikat SHA-256. Achtung: Diese Platine verfügt über I/O-Leitungen, die nur mit 3,3V kompatibel sind.

5.2.9 Arduino MKR WIFI 1010

Der MKR WIFI 1010 ist eine signifikante Verbesserung gegenüber dem MKR 1000 WIFI. Das Board ist mit einem ESP32-Modul von U-BLOX ausgestattet. Dank der Flexibilität des ESP32-Moduls und seines geringen Stromverbrauchs soll dieses Board das Prototyping von WiFi-basierten IoT-Anwendungen beschleunigen und vereinfachen. Das MKR WIFI 1010 bietet schnelle 32-Bit-Rechenleistung, den üblichen umfangreichen Satz an I/O-Schnittstellen und WLAN mit geringem Stromverbrauch und einem Cryptochip für die sichere Kommunikation mithilfe der SHA-256-Verschlüsselung. Außerdem bietet es eine benutzerfreundliche Arduino Software (IDE) für die Programmierung. All diese Eigenschaften machen dieses Board zur bevorzugten Wahl für die aufkommenden IoT-Projekte mit Batteriebetrieb in kompakter Form. Über den USB-Anschluss kann die Platine mit 5 V versorgt werden. Es verfügt über eine Li-Po-Ladeschaltung, mit welcher der Arduino MKR WIFI 1010 mit Batteriestrom oder einer externen 5-Volt-Quelle betrieben werden kann. Auf diese Weise wird der Li-Po-Akku geladen, während er mit externer Energie versorgt wird. Das Umschalten von einer Quelle zur anderen erfolgt automatisch. Achtung: Diese Platine verfügt über I/O-Leitungen, die nur mit 3,3V kompatibel sind.

5.2.10 Arduino MKR Zero

Der MKR ZERO bietet dir die Leistung eines Zero im kleineren Format, das durch den MKR-Formfaktor festgelegt wurde. Das MKR Zero-Board ist ein hervorragendes Lerninstrument, um sich mit der Entwicklung von 32-Bit-Anwendungen vertraut zu machen. Es verfügt über einen integrierten SD-Anschluss mit dedizierten SPI-Schnittstellen (SPI1), mit denen man ohne zusätzliche Hardware Musik-Dateien spielen kann! Das Board wird von der SAMD21-MCU von Atmel mit einem 32-Bit-ARM Cortex® M0+ Kern gespeist. Achtung: Diese Platine verfügt über I/O-Leitungen, die nur mit 3,3V kompatibel sind.

5.3 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 (5V) oder Low-Pegel (0V) gezogen wird. In unserem MKR-Brick werden Pullup-Widerstände mit automatischer Widerstandsanpassung verwendet. Bei Low-Pegel beträgt der Wert 40kΩ (es fließt weniger Strom, bei High-Pegel sind es 4kΩ (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 zu-

verlässig 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.

5.4 Die Programmierung: Arduino IDE

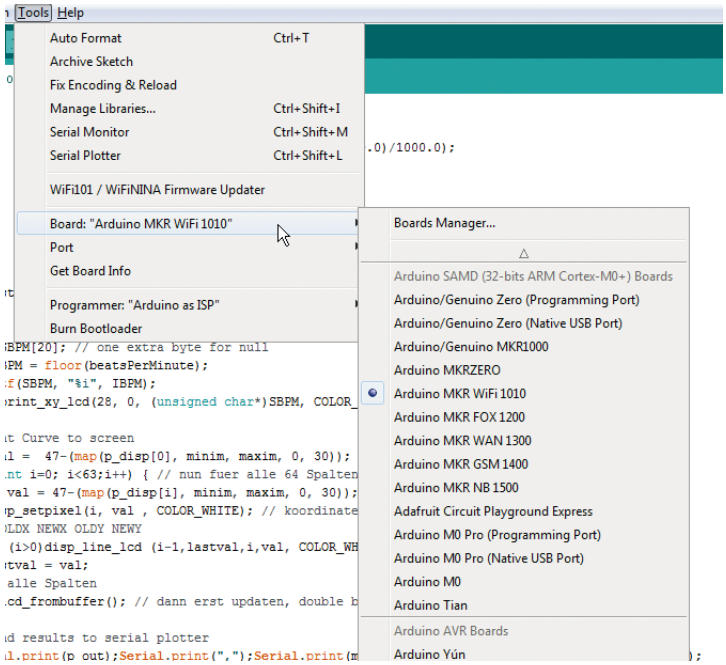
Um den MKR-Brick zu programmieren verwenden wir die Arduino-Entwicklungsumgebung – auch Arduino IDE (IDE steht für Integrierte Entwicklungsumgebung) genannt. Viele kennen die kostenlose Programmier-Software vielleicht schon aus der Arduino-Welt. Es gibt im Internet zahlreiche Arduino-basierende Projekte 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.

• Starte die Arduino IDE, z.B. über das Startmenü unter Windows (je nach Betriebssystem unterschiedlich).

• Stelle unter Werkzeuge als Board „Arduino MKR WiFi 1010" ein. Sollte der MKR WiFi 1010 nicht gelistet sein, dann zuerst den „Boards Manager...“ ganz oben aktivieren und dort nach dem MKR WiFi 1010 suchen und diesen installieren.



5.4.1 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 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). Es ist auch möglich direkt in der IDE unter scetch include add zip library eine Library als zip-Datei zu installieren, oder den direct library loader verwenden, so spart man sich die ganze Sucherei.

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

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

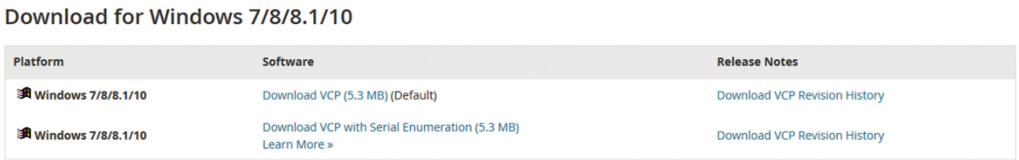
5.4.2 Virtueller COM-Port-Treiber (optional)

Damit sich dein Entwicklungsrechner mit dem MKR-Brick verständigen kann, muss in der Arduino IDE noch die Schnittstelle deines PCs ausgewählt werden, über welche die Verbindung zum MKR-Brick erfolgen soll. Zu diesem Zweck kann man in der Arduino IDE einen seriellen Port (auch COM-Port genannt) auswählen. Klassischerweise sind das 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 ggf. einen virtuellen COM-Port Treiber (VCP) installieren. Dieser ist Voraussetzung für die Kommunikation zwischen Arduino IDE und der USB-Schnittstelle des MKR-Bricks.

Im Windows Gerätemanager muss unter "Anschlüsse (COM & LPT)" ein Eintrag mit dem Präfix „Silicon Labs“ zu finden sein. Die Anzahl der COM-Ports und der Index hängen von deiner Rechner-Konfiguration ab. Wenn Du den Eintrag „Silicon Labs“ sehen kannst, braucht kein zusätzlicher Treiber installiert zu werden. Du kannst dann mit dem nächsten Kapitel „Serieller Monitor“ fortfahren.

Für den Fall, dass kein Eintrag „Silicon Labs“ zu sehen ist, 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



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 32Bit Windows-Version oder CP210xVCPInstaller_x64.exe für eine 64Bit Version.

• Im Windows Gerätemanager muss unter "Anschlüsse (COM & LPT)" ein Eintrag mit dem Präfix „Silicon Labs“ zu finden sein. Die Anzahl der COM-Ports und der Index hängen von deiner Rechner-Konfiguration ab.

5.4.3 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.

5.4.4 Serieller Plotter

Manche Beispielprogramme verwenden den sog. seriellen Plotter zur Anzeige von Graphen. Um den seriellen Plotter zu öffnen, klickst du im Menü „Werkzeuge“ auf den Menüeintrag „Serieller Plotter“. Die Baudrate im Monitorfenster (rechts unten) und im Sketch muss übereinstimmen.

5.5 Das erste Programm: Blink

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.

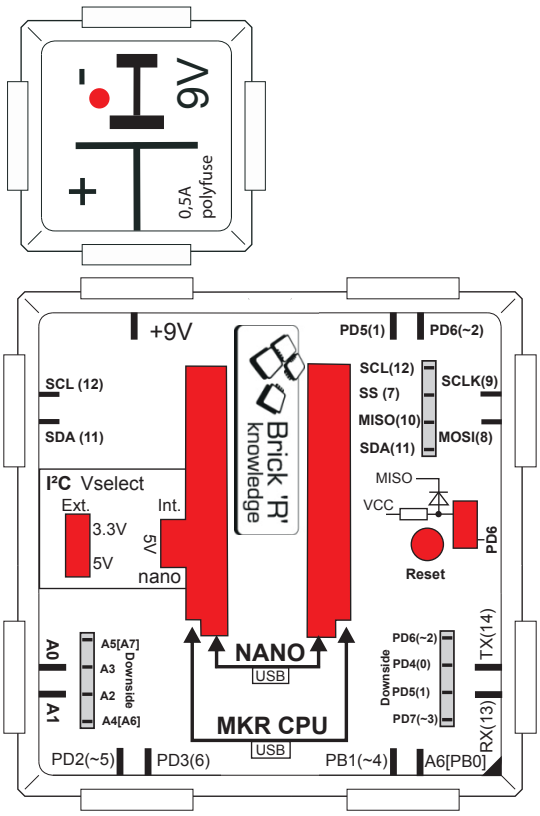
5.5.1 Verbindungen herstellen

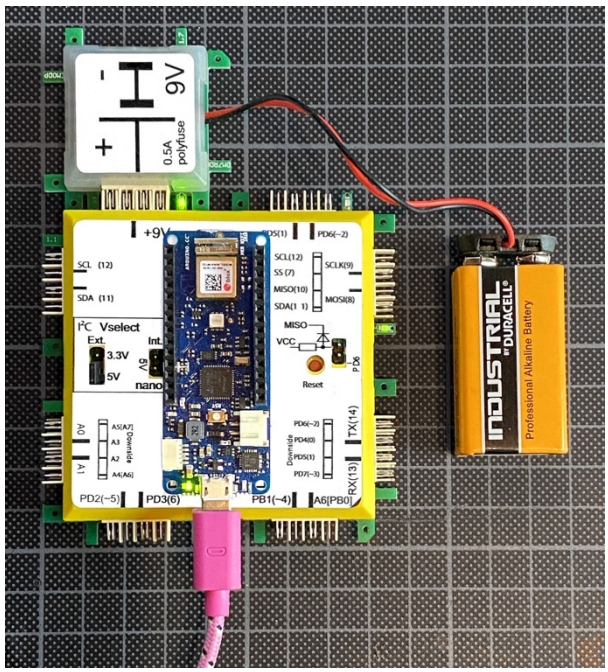
Verbinde zunächst den MKR-Brick mit dem einem 9V-Netzteil an dem dafür vorgesehenen, oberen Anschluss.

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

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

Achtung: Das 9V-Netzteil ausschließlich an den linken oberen Kontakt des MKR-Bricks anschließen. Andernfalls kann es zur Zerstörung des Bricks kommen!





Der Computer sollte jetzt einen neuen, sog. virtuellen COM-Port haben, über den die Arduino IDE die Programme in den MKR-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 MKR-Brick.

• Starte die Arduino-Software und prüfe welche COM-Schnittstelle(n) unter "Werkzeuge – Port:..." angezeigt werden.

• Verbinde jetzt wieder das USB-Kabel mit dem MKR-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.

5.5.2 Der Sketch

Um das Programm zu laden, öffne in der Arduino IDE den Sketch: Beispiele-01.Basics-Blink.

Der Sketch sieht so aus:

```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
```

```
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

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

5.5.3 Die Setup-Routine „void setup()“

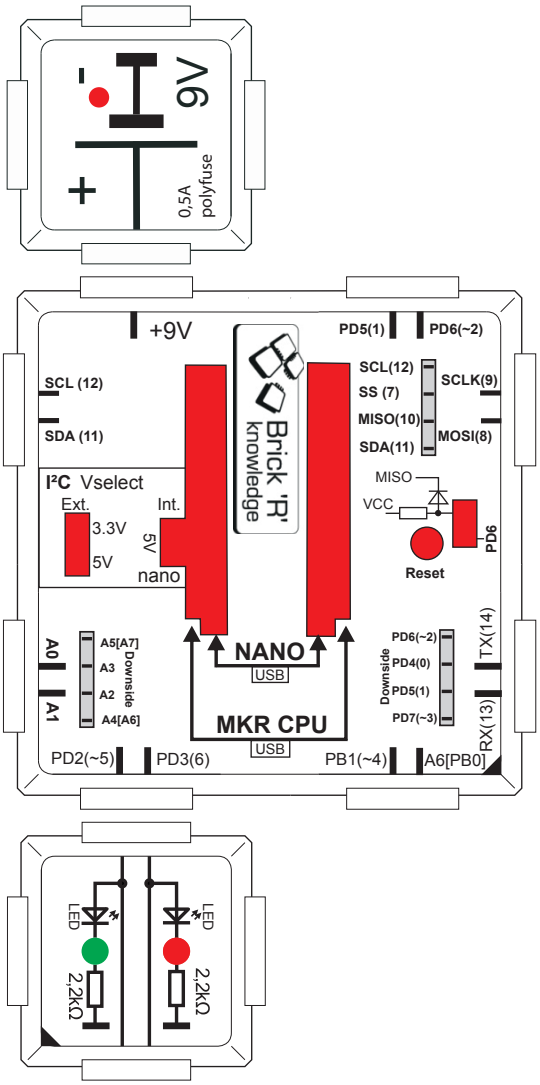
Zunächst kommt der Abschnitt void setup(){...}. Alle Befehle innerhalb dieser geschweiften Klammern werden beim Start des Programmes genau einmal ausgeführt. In unserem Beispiel wird hier die Richtung der GPIO-Pins definiert, also Eingang (Mode: Input) oder wie in unserem Fall: zwei Ausgänge (Mode: Output).

5.5.4 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.

Auf der linken Seite der MKR-Platine blinkt nun im Sekundentakt eine orange LED.

5.6 Doppelte LED



Nun schließen wir einen unserer Bricks direkt an den MKR an. Dazu haben wir einen Baustein mit zwei Leuchtdioden. Der MKR-Brick verwendet neben den üblichen Masseanschlüssen (die äußeren Kontakte des Brick-Bausteins) die beiden inneren Kontakte getrennt. Es gibt beim diesem Brick auch Kontakte die unten liegen, doch dazu später mehr. Die beiden LEDs sind mit den Portausgängen PD2 und PD3 verbunden, dies entspricht auch der Nummerierung 5 und 6 im Sketch. Die LEDs blinken hier abwechseln, da immer paarweise 5 auf high und 6 auf low und danach 5 auf low und 6 auf high gesetzt werden.

```
#define PORTLED2 5 // definiert das Symbol PORTLED2 mit 5
#define PORTLED3 6 // entsprechend fuer PORTLED3 mit 6
// Ausführen am Anfang
void setup() {
  pinMode(PORTLED2,OUTPUT); // Port 2 als Ausgang schalten
  pinMode(PORTLED3,OUTPUT); // Port 3 als Ausgang schalten
  Serial.println(LED_BUILTIN);
}
// Schleife wird wiederholt ausgefuehrt:
void loop() {
  digitalWrite(PORTLED2,HIGH); // Ausgang auf hohen Pegel schalten
  digitalWrite(PORTLED3,LOW); // Ausgang auf niedrigen Pegel schalten
  delay(1000); // Eine weitere Sekunde warten (=1000 ms)
  digitalWrite(PORTLED2,LOW); // Ausgang auf niedrigen Pegel schalten
  digitalWrite(PORTLED3,HIGH); // Ausgang auf hohen Pegel schalten
  delay(1000); // Eine weitere Sekunde warten
}
```

5.7 I2C-Bus

Der I2C-Bus (Inter Integrated Circuits 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 MKR-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 Mikrocontroller, gibt dabei auf der SCL-Leitung den Takt vor. Über die SDA-Leitung werden die eigentlichen Daten übertragen.

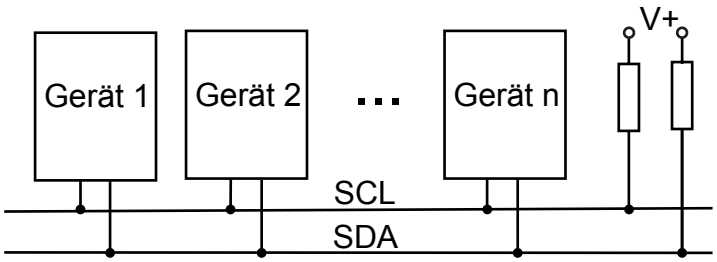


Abbildung: I2C-Busstruktur

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 MKR-Brick bereits eingebaut.

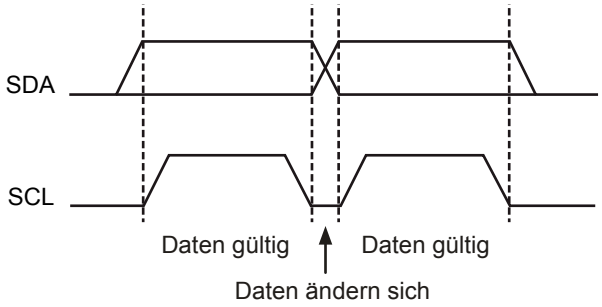
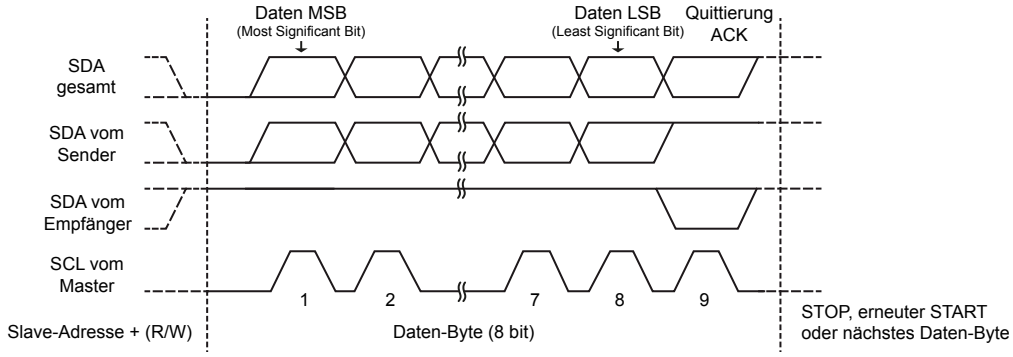


Abbildung: Gültige Daten am I2C-Bus

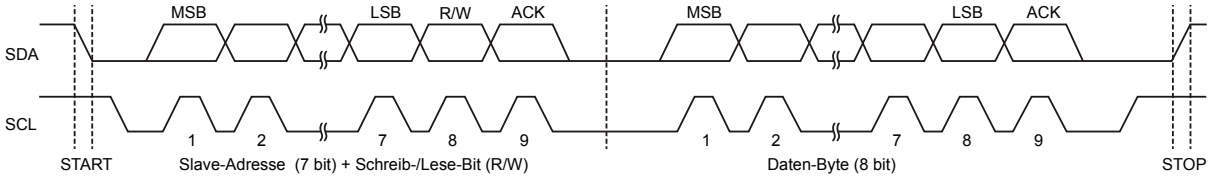
Der Takt gibt dabei an, wann gültige Daten anliegen. In Abb. 28 sieht man, dass dies immer beim High-Pegel der SCL-Leitung der Fall ist. Der Empfänger kann jetzt die Daten abtasten und auswerten.

Der Master gibt dabei den Takt vor, er legt dann entweder selbst Daten an oder erwartet solche vom entsprechenden Gerät.



Datentransfer am I2C-Bus

In der oberen Abbildung 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.



Übertragungszyklus am I2C-Bus

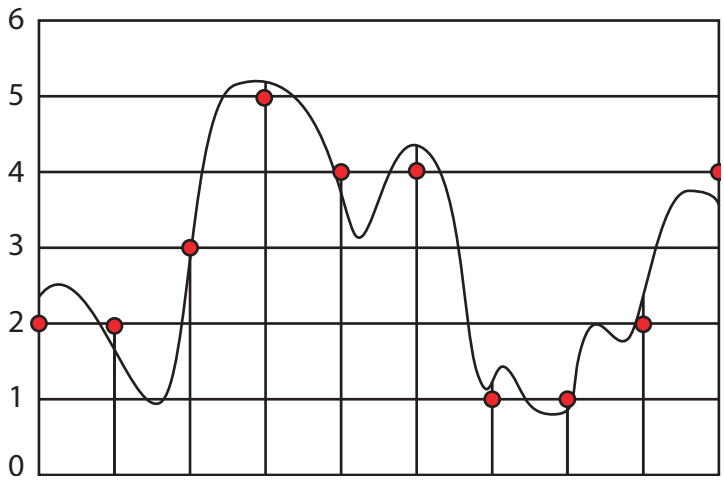
In der oberen Abbildung sieht man den gesamten Übertragungszyklus. Zunächst wird ein Paket mit der Adresse gesendet. Die Adresse besteht aus 7 Bits, ergänzt um ein weiteres, R/W (Read/Write) benanntes Bit. Alle Busteilnehmer vergleichen die ausgesendete Adresse mit ihrer eigenen. Bei Übereinstimmung quittiert der betreffende Slave mit einem ACK-Signal, indem er die SDA-Leitung kurzzeitig auf low legt.

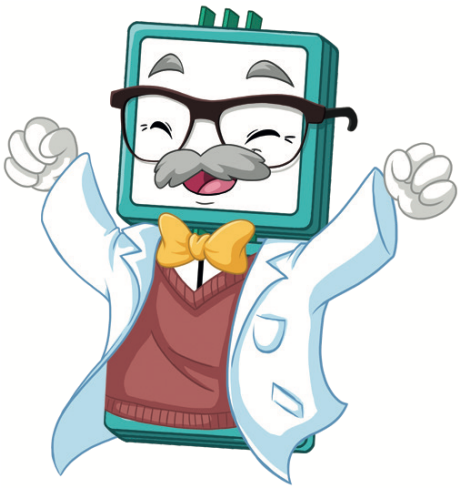
In Abhängigkeit vom R/W-Bit weiß der adressierte Slave, ob er nun einen Sende- oder Empfangszyklus starten soll. Danach kann der eigentliche Datentransfer beginnen. Als Abschluss wird ein Stop-Zyklus eingeleitet. Dazu wird der Takt auf high gesetzt und dann die SDA-Leitung freigegeben. Die Leitungen SDA und SCL sind nun beide auf High-Pegel, das bedeutet, dass der I2C-Bus frei verwendbar ist. Theoretisch kann nun auch ein anderer Master (falls mehrere am Bus sind), einen neuen Zyklus starten.

Für uns ist der I2C-Bus einfach nutzbar, da die Arduino-Bibliothek mehrere Befehle bereitstellt, um Bricks mit I2C-Interface wie z.B. den 7-Segmentanzeige-Brick über den MKR-Brick anzusteuern.

5.8 Analog-Digital Umsetzer

5.8.1 A/D Umsetzer - prinzipieller Aufbau





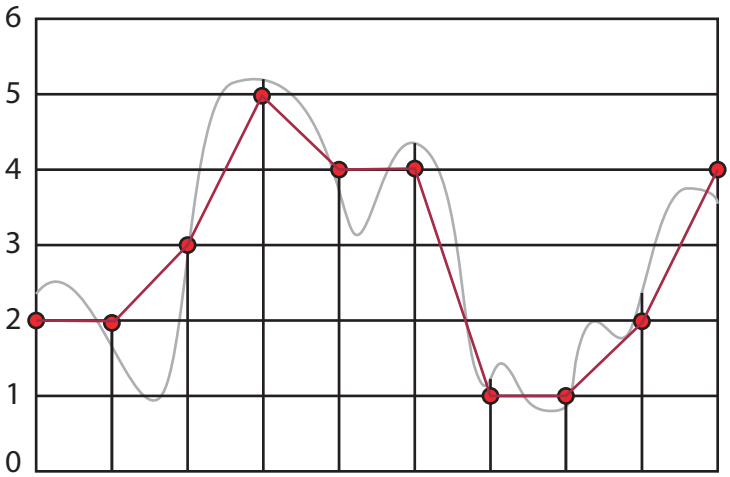
Hier haben wir den zeitliche Verlauf eines Spannungssignals aufgezeichnet (hier kann man sich in der y-Achse die Spannung in V aufgetragen denken und in x-Richtung die Zeit z.B. in Sekunden).

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 dann die folgende Zahlenreihe:

2, 2, 3, 5, 4, 4, 1, 1, 2, 4

Es gibt dabei zwei interessante Effekte. Wir verlieren Informationen in der Amplitude, die beiden ersten Werte sind jedes Mal 2 als Beispiel. Mit einer höheren Auflösung bei der Wandlung, z.B. einer weiteren Nachkommastelle hätte man mehr Information des ursprünglichen Signales erhalten.

Und der zweite Effekt, wir verlieren auch zeitlich relevante Informationen. Die Sequenz 1,1 beinhaltet einen kleinen Schwinger der in der Zahlenreihe nicht mehr sichtbar ist. Die Fachleute sagen dazu dass das sogenannte Nyquist-Kriterium verletzt wird, denn man muss mindestens mit der doppelten Frequenz abtasten, die im Signal enthalten ist. Bei den kleinen Schwingungen ist das Prinzip verletzt. Wenn man nun die roten Punkte verbindet, dann erhält man das für den Computer verwertbare Signal, die ursprüngliche Kurve ist nicht mehr genau rekonstruierbar. Wenn man die Zahl der Abtastpunkte erhöht, wird das Ergebnis besser.



Oben sieht man die rekonstruierte Kurve.

Die Auflösung für die Amplitude wird durch die Anzahl der Bits, die einem Zahlenwert zugeordnet werden, bestimmt. Bei dem Analog-Digital-Umsetzer im Arduino MKR sind dies 10 Bit. Das entspricht 2 hoch 10 Werten, also mathematisch umgerechnet 1024 Stufen. Bei einem Spannungsbereich von 0 bis 5V entspricht die kleinste Stufe einer Spannung von 5V / 1024 = 4.88mV.

Der Techniker interessiert sich auch für den sogenannten Dynamikbereich, den berechnet man aus dem Dynamic Range = 20 * LOG (Anzahl der Stufen) in dB.

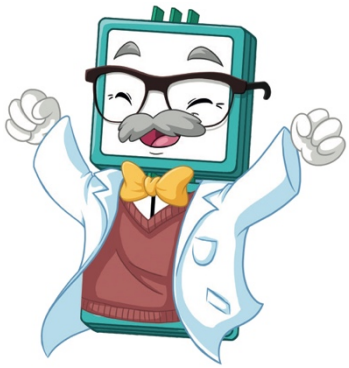
Bei uns = 20*LOG(1024) = 60.2 dB. Dies ist ein ganz guter Wert. Das menschliche Gehört hat aber eine höhere Dynamik, daher muss man Audiosignale für eine HIFI Qualität auch mit mehr Bits digitalisieren, zum Beispiel bei 24 Bit ergeben sich: 20 * LOG (2Hoch24) = 20 * LOG (16777216)=144.5 dB.

Das menschliche Gehör kann in etwa 120 dB in bestimmten Hörbereichen wahrnehmen.

Wie kann man nun solche Signale in digitale umwandeln. Dazu gibt

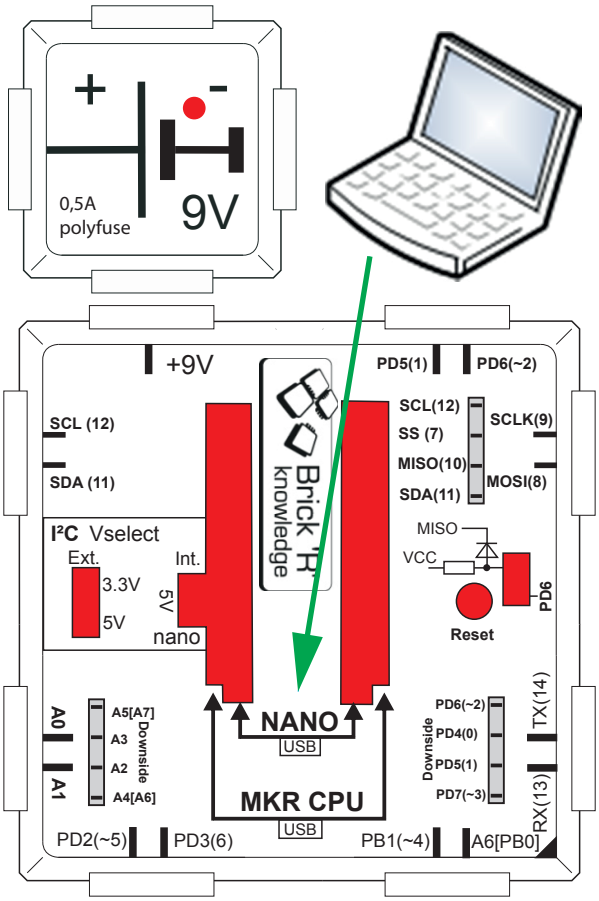
es unterschiedliche Verfahren, die den Rahmen hier schnell sprengen würden. Aber ein paar Stichworte zum Suchen seien hier genannt: Delta-Sigma, Sukzessive Approximation (SAR): Parallelwandler, Sägezahnverfahren, Mehrrampenverfahren, um nur einige zu nennen.

Der A/D-Umsetzer des MKR WiFI 1010 ist im SAMS-Prozessor eingebaut. Der Wandler kann im Prinzip mit 15kSPS digitalisieren (bei maximaler Auflösung von 10 Bit), also 15.000 mal pro Sekunde. Die Eingänge sind gemultiplexed, das heißt nur je einer der 8 analogen Eingänge wird an den internen Wandler geschaltet. Werden mehre Kanäle verwendet, sinkt die maximale Digitalisierungsrate entsprechend, da man den Wandler nur für einen Kanal gleichzeitig verwenden kann. Das Wandelprinzip ist, das der sukzessiven Approximation. Die Arduino Software-Bibliothek macht das Ganze aber einfach, man muss nur den Befehl digitalRead(PINNO...) verwenden und der Wandler wird aktiviert und nach der Umwandlung aktiviert. So bekommt man den Zahlenwert zwischen 0 und 1023, was einer Spannung zwischen 0 und 5V entspricht.



5.8.2 I/O Testprogramm - SetPinsTEST

Das ist ein einfaches Testprogramm für die IO-Ports der CPU. Auf dem Terminal werden die analogen Kanäle ausgegeben. Alle digitalen Kanäle erhalten einen kurzen Puls. Den kann man z.B. mit einem Prüfstift oder Oszilloskop sehen. Achtung: Am Ausgang MISO erscheint bei dieser Programmierung ein Signal mit drei Spannungspegeln, da über die interne Diode das Signal von PD6 mit auf MISO gemischt wird. Diese Diode dient eigentlich dazu, den statischen Zustand von MISO abzufragen, z.B. für den EEG-Brick, da dort der MISO doppelt belegt ist. Man kann dies meist auch direkt am MISO Pin, tun, allerdings abhängig vom Prozessor und der Bibliothek.



Immer wenn der PC hier gezeigt wird, heißt es, dass Ausgaben auf dem Bildschirm zu erwarten sind. Beim Arduino Programm drückt man dazu CTRL-SHIFT-M manchmal werden auch Grafiken ausgeben, dann muss man CTRL-SHIFT-L eingeben.

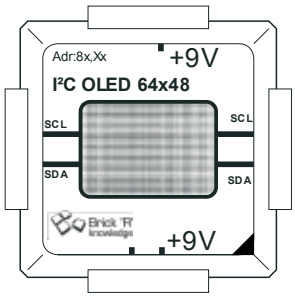
Alternativ dazu kann man auch ein separates Terminal-Programm benutzen.

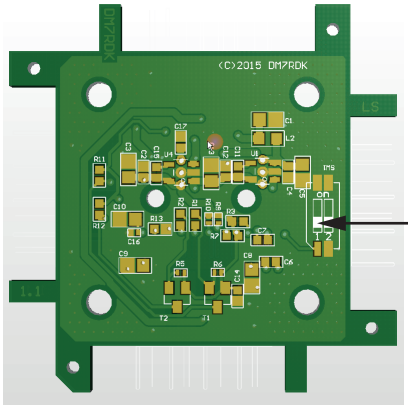
```
// the setup function runs once when you press reset or power the board
void setup() {
 Serial.begin(115200);
 delay(5000); // Wait 5 Seconds until user opens the Terminal
 Serial.println("");
 Serial.println("");
 Serial.println("INIT");
  for (int x = 0; x < 15; x++){
    pinMode(x, OUTPUT);
    }
  pinMode(LED_BUILTIN, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
   for (int x = 0; x < 15; x++){
    digitalWrite(x, HIGH);
    }
    for (int x = 0; x < 15; x++){
    digitalWrite(x, LOW);
    }
   digitalWrite(LED_BUILTIN, LOW);
    Serial.println("Analog Sensors:");
    Serial.print(analogRead(A0));
```

```
    Serial.print("; ");
    Serial.print(analogRead(A1));
    Serial.print("; ");
    Serial.print(analogRead(A2));
    Serial.print("; ");
    Serial.print(analogRead(A3));
    Serial.print("; ");
    Serial.print(analogRead(A4));
    Serial.print("; ");
    Serial.print(analogRead(A5));
    Serial.print("; ");
    Serial.println(analogRead(A6));
    Serial.println("-------------------------");
    delay(5000); // Wait 5 Seconds until next measurement
}
```

5.9 OLED

5.9.1 Grafische Anzeige: Das OLED Display





Adresse 78 oder 7A:
Schalter auf ON
bedeutet 78

Die 7-Segment-Anzeigen werden normalerweise nur für Ziffern verwendet und ganz eingeschränkt auch für Buchstaben. Mit 14- und 16-Segment-Anzeigen lassen sich auch Buchstaben brauchbar darstellen. Danach gab es dann die ersten Rasteranzeigen, mit einer Matrix von 5x7 Punkten konnte man die 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 Diode), diese sind wieder den LEDs ähnlicher, da sie selbst leuchten können. Für Brick'R'knowledge gibt es ein monochromes OLED Display mit 64x48 Pixeln. Damit lassen sich nicht nur einzelne Zeichen, sondern auch ganzer Text und einfache Grafiken darstellen. Damit man Zeichen in dieser Weise auf dem Display darstellen kann, benötigt man einen Zeichengenerator, bzw. Tabelle, so ähnlich wie die 7-Segmenttabelle. Der numerische Code wird für die Segmente in den Zustand „an" und „aus" gesetzt. Der Zeichengenerator oder die Zeichensatztabelle benötigt ungleich mehr Speicher. Der Wert hängt von der Zahl der Pixel 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. Für den MKR-Brick haben wir eine solche Zeichentabelle vorbereitet, die noch etwas anders aufgebaut ist. Damit wird es möglich auch Unicode Sonderzeichen, die nicht im ASCII-Satz enthalten sind, darzustellen. Um eine bessere Lesbarkeit zu erreichen, ist der Font etwas größer und der Schriftabstand variabel (Proportionalschrift). Zu Erzeugung wurde der BitFontCreator PO v3.0 verwendet, mit dem sich unterschiedliche Schriften in Raster für Mikrocontroller und damit auch den MKR generieren lassen.

Unser OLED-Brick belegt eine I2C Adresse, die entweder 0x78 oder 0x7A ist, je nach Schalterstellung von SW1 auf der Rückseite der Platine. Der innere Schalter bestimmt die Adresse, normalerweise

trägt er die Beschriftung „1“. Die OLED wird über I2C programmiert und hat einen eigenen Controller an Board.

In unserer kleinen Bibliothek gibt es Aufrufe, mit denen man Zeichen, Texte, Linien usw. leicht anzeigen kann, was die folgenden Beispiele noch verdeutlichen.

```
// Auszug aus unserem CODE:

// Einzelne Zeichen codiert

const unsigned char fontArial14h_data_tablep[] PROGMEM =
{

/* character 0x0020 (, ,): [width=3, offset= 0x0000 (0) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x0021 (,!‘): [width=2, offset= 0x000E (14) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x00, 0x80, 0x00, 0x00, 0x00,
...

};

// Tabelle mit offsets in die Zeichentabelle
const unsigned int fontArial14h_offset_tablep[] PROGMEM =
{
/*  offset      offsetHex  -  char    hexcode    decimal */
/*  =======     =========  -  ====    =======    ======= */
  0,          /*         0  -          0020       32      */
  14,         /*         E  -  !       0021       33      */
  28,         /*        1C  -  „       0022       34      */
...
  1876,       /*       754  -  extra address: the end of the last character‘s imagebits data */
};


// Breitentabelle
const unsigned short fontArial14h_width_tablep[] PROGMEM =
{
/*  width    char    hexcode    decimal */
/*  =====    ====    =======    ======= */
  3,        /*          0020        32   */
  2,        /*  !       0021        33   */
  4,        /*  „       0022        34   */
...
  8,        /*  ?       2642      9794   */
  8,        /*  ?       266B      9835   */
};
```

5.9.2 OLED Brick Bibliothek

Wir haben zahlreiche Befehle vorbereitet, mit denen man die OLED einfacher ansprechen kann, als mit reinen I2C Befehlen. Die basieren auf dem internen Controller und man kann damit jedes Pixel ansteuern, Helligkeiten programmieren und vieles mehr. Ausgabe von Zeichen, Schriften, Linien usw. ist nicht im OLED vorgesehen, daher muss es dann extern programmiert werden, dabei hilft unsere eine kleine Bibliothek.

Der wichtigste Befehl ist:

```
i2c_oled_initall(i2coledssd);
```

Damit wird das OLED überhaupt erst in Betrieb genommen. Auch das Timing der OLED wird hier programmiert, so dass es für die interne Brick-Schaltung angepasst ist. Als Parameter wird die I2C Adresse angegeben:

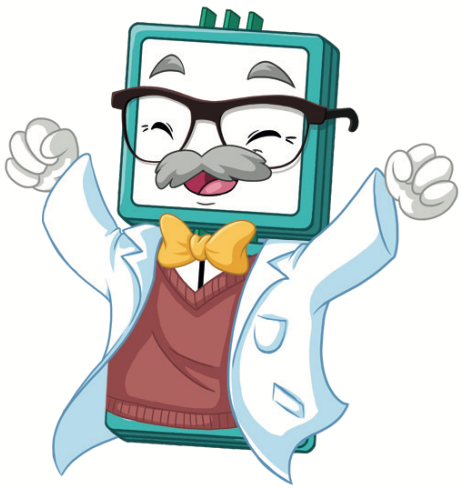
```
#define i2coledssd (0x7A>>1)  // Default
```

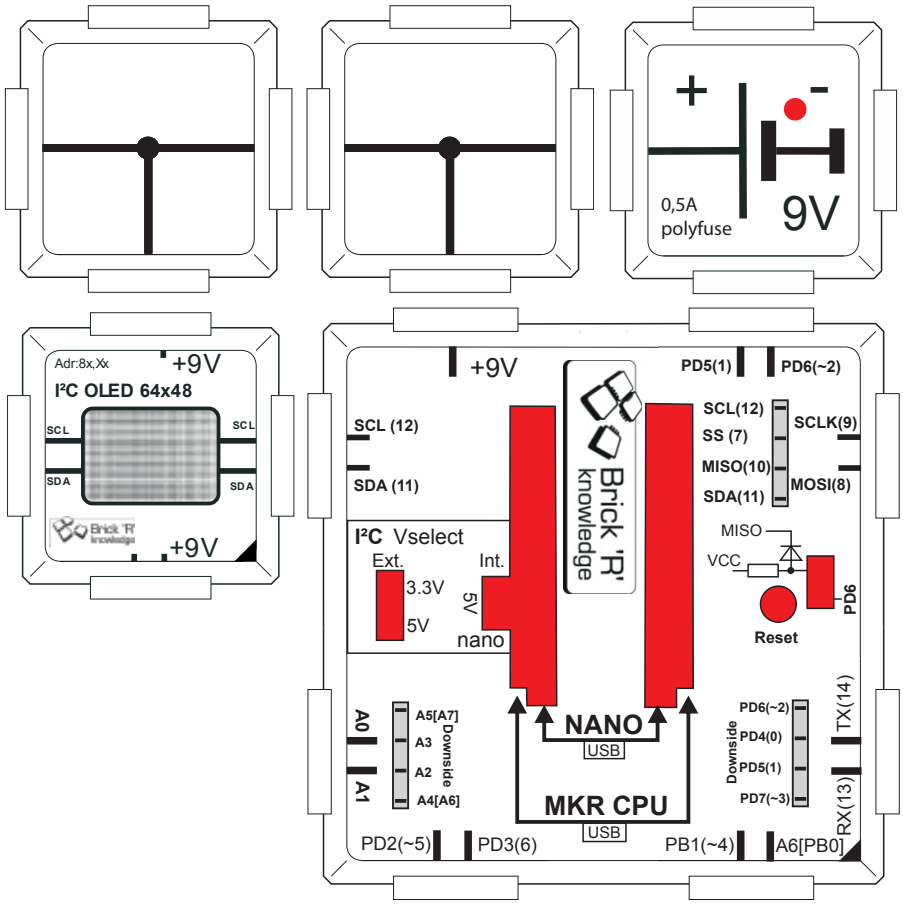
oder

```
#define i2coledssd (0x78>>1)  // optional für zweites OLED
```

Es wird ein interner Buffer (lcdbuffer[]) verwendet, um alle Pixel zu speichern und diese dann auf einmal zu übertragen. Das vermeidet Flimmer-Effekte, was Double-Buffering genannt wird.

```
// Helle & dunkle Pixel, in dieser Ausführung gibt es keine Zwischenstufen.
#define COLOR_BLACK 0 #define COLOR_WHITE 1
// damit vereinfachter Parameter // OLED Pixel = an (wenn nicht invers)
```





```
void i2c_oled_write_command(unsigned char i2cbaseadr, unsigned char cmdvalue)
      Intern verwendet um an den I2C Daten zu schreiben
void i2c_oled_entire_onoff(unsigned char i2cbaseadr, unsigned char onoff)
      Damit kann man die OLED an- oder ausschalten. 1=an 0=aus
void i2c_oled_display_onoff(unsigned char i2cbaseadr, unsigned char onoff)
      Auch zum an- und auschalten des Displays ander Variante
void i2c_oled_setbrightness(unsigned char i2cbaseadr, unsigned char wert)
      Helligkeit des OLEDs setzen 0..255 als Wert 0=dunkel
void i2c_oled_inverse_onoff(unsigned char i2cbaseadr, unsigned char onoff)
      Invertieren des Displays, 1=invers
unsigned char i2c_oled_write_top(unsigned char i2cbaseadr, int zeile,
                  int bytes,  unsigned char barray[],signed int sh1106padding)
      Intern verwendet um eine Zeilegruppe (8) zu schreiben.
void disp_lcd_frombuffer()
      Überträgt den Bufferinhalt (lcdbuffer[]) auf das Display.
void disp_buffer_clear(unsigned short data)
      Löscht den Pixelbuffer (lcdbuffer[]) mit data (=0 oder =1)
void disp_setpixel(int x, int y, unsigned short col1)
      Setzt ein Pixel bei x,y wobei x links und y oben ist.
      col1 definiert die Farbe (0 oder 1)
unsigned short disp_setchar(int x, int y,  unsigned char chidx1, unsigned short color)
      Setzt ein Zeichen bei x,y mit ASCII Coe chidx1 und color (=0 oder =1)
int disp_print_xy_lcd(int x, int y, unsigned char *text, unsigned short color, int
chset)
      Ausgabe eines Textes (ASCII bei char *text) an der Position x,y
      mit dem Zeichensatz chset (bei uns ignoriert), linke obere Ecke.
void  disp_line_lcd(int x0, int y0, int x1, int y1, unsigned short col)
      Linie von x0,y0 nach x1,y1 in Farbe col (0,1)
void disp_rect_lcd(int x1, int y1, int x2, int y2, unsigned short col)
      Rechteck bei x1,y1 aufgespannt nach x2,y2 in Farbe col(0,1)
void disp_filledrect_lcd(int x1, int y1, int x2, int y2, unsigned short col)
      Gefuelltes Rechteck bei x1,y1 aufgespannt nach x2,y2 in Farbe col(0,1)
```

5.9.3 OLED Display über I2C

Hier wird eine Sinus-Funktion auf dem Display dargestellt. Dabei wird nach jedem Bildaufbau die Phase des Sinus leicht geändert, so dass der Sinus durchzulaufen scheint. Versuchsaufbau mit OLED-Display wie zuvor.

Wir verwenden dazu ein paar praktische Elemente aus unserer Bibliothek. Der Aufruf disp_buffer_clear() löscht den Displaybuffer. Man kann eine Löschfarbe (Schwarz oder weiß) als Parameter angeben. Dies geschieht aber nur in einem Buffer auf dem MKR und nicht auf der OLED selbst. Die OLED zeigt zu dem Zeitpunkt noch das aktuelle Bild an. Man nennt das auch Double Buffer Technique. Danach wird eine horizontale Linie gezeichnet. Dazu gibt es den Befehl disp_line_lcd(). Als Parameter wird hier die Startkoordinate x,y gegeben und die Zielkoordinate (der Zielpunkt wird auch als Punkt gezeichnet!). X geht von 0 bis 63 und y von 0 bis 47. Der Ursprung liegt dabei links oben!

Die Funktion sin() muss noch in den Wertebereich angepasst werden, dazu wird y mit dem Range 0..47 berechnet, sin() hat ja bekanntlich den Wertebereich -1.0 .. 1.0.

Mit disp_setpixel() werden nun genau die einzelnen Punkte der Funktion gesetzt, die durchlaufen werden. Als Parameter gibt es x,y und die Farben weiß oder schwarz. Die Koordinate y wird vor der Ausgabe mit 47-y als Parameter an disp_setpixel() übergeben, da der Ursprung bei der OLED für 0,0 oben links liegt.

ACHTUNG: Die Adresse muss hier auf 7A stehen, ggf. den Switch auf der Rückseite auf OFF stellen !

```
// DE_27 OLED Beispiele - Pixelroutinen
#include <Wire.h> // I2C Bibliothek
#include <avr/pgmspace.h> // Zugriff ins ROM
// Hier ggf Adresse anpassen 78 oder 7A je nach Schalter
#define i2coledssd (0x7A>>1)  // default ist 7A
// -------------------------OLED ---------------------------------------------
// An dieser Stelle steht der Code für die OLED-Library.
// In den downloadbaren Beispielen ist der Code hier eingefügt.
// Für das Handbuch verwenden wir nur das eigentliche Hauptprogramm
// -------------------------END OLED -----------------------------------------
void setup() {
  Wire.begin(); // I2C Init
  i2c_oled_initall(i2coledssd); // OLED Init
}
void loop() {  // in der Schleife
  // 64x48 Pixel OLED
  static int phase=0; // Scrolleffekt hier statisch speichern
  disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
  disp_line_lcd (0,24,63,24, COLOR_WHITE); // x0,y0,x1,y1 Linie in Mitte
  for (int i=0; i<63;i++) { // nun fuer alle 64 Spalten (x)  des Displays
    int y=0; // Sinuskurve berechnen und in Radiant umrechnen von Phase
    y = (int)(23.0*sin(((double)i*3.141592*2.0)/64.0+phase/360.0*2.0*3.141592)+24.0);
    disp_setpixel(i, 47-y, COLOR_WHITE); // koordinate 0,0 ist links oben daher 47-y
  } // alle Spalten
  disp_lcd_frombuffer(); // dann erst updaten, double buffer flimmerfrei
  phase++; // beim nächsten Mal mit neuer Phase für den Sinus
  if (phase>=360)phase=0; // Immer 0 bis 359 Grad (als DEG)
}
```

Was passiert? Nach dem Start erscheint eine Sinuskurve auf dem Display. Dabei scrollt die Kurve langsam horizontal durch.

5.9.4 OLED Display und Zeichensatz

Hier wird eine Sinus-Funktion auf dem Display dargestellt. Dabei wird nach jedem Bildaufbau die Phase des Sinus leicht geändert, so dass der Sinus durchzulaufen scheint. Versuchsaufbau mit OLED-Display wie zuvor.

Mit disp_print_xy_lcd() lässt sich ein Text auf dem Display darstellen. Dazu wird als Parameter die Koordinate der linken oberen Ecke des ersten Zeichens angegeben (x,y). 0,0 links links oben. Die Zeichenhöhe kann man mit ca. 10 Pixeln annehmen, wenn man mehrere Zeilen schreibt, so muss man einen entsprechenden Offset eingeben. In dem Programmbeispiel werden mehrere Textzeilen ausgegeben. Der berühmte Satz „the quick brown fox jumps over the lazy dog“ enthält alle vorkommenden Zeichen des Alphabets, daher wird er gerne für solche Beispiele verwendet.

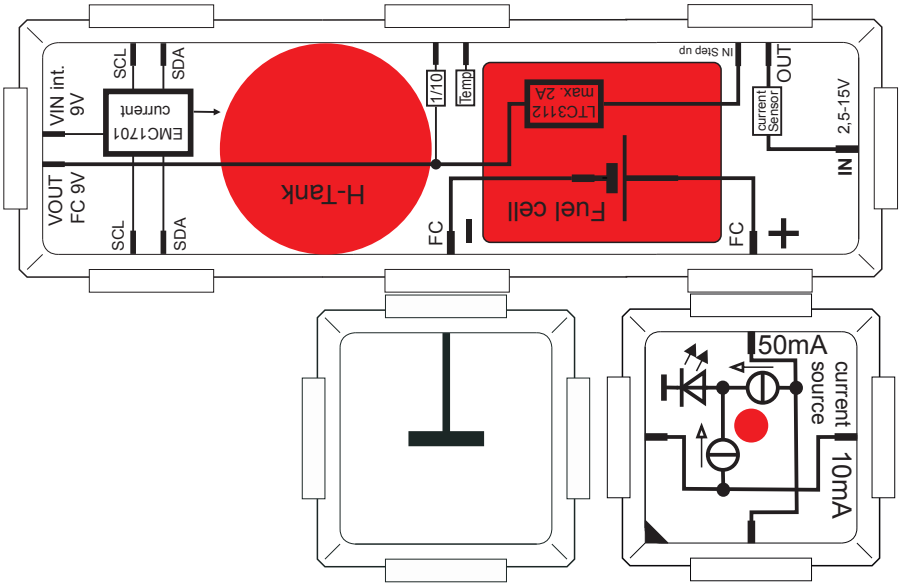
Das Programm verwendet zusätzlich eine Variable yoffset, die bei jedem Durchlauf der Schleife „loop“ um eins verringert wird. Dadurch entsteht ein Scroll-Effekt, der Text läuft nach oben durch. Wenn der Offset von -60 erreicht ist, ist der Bildschirm leer und der Offset wird zurück auf 50 gesetzt, so dass die erste Zeile links unten erscheint (bei einem Offset von 0 würde der Text oben starten).

```
// DE_28 OLED Beispiele - Zeichensatz
#include <Wire.h>
#include <avr/pgmspace.h>
// Hier ggf Adresse anpassen 78 oder 7A je nach Schalter
#define i2coledssd (0x7A>>1)  // default ist 7A
// -------------------------OLED ---------------------------------------------
// An dieser Stelle steht der Code für die OLED-Library.
// In den downloadbaren Beispielen ist der Code hier eingefügt.
// Für das Handbuch verwenden wir nur das eigentliche Hauptprogramm
// -------------------------END OLED -----------------------------------------
void setup() {
  Wire.begin(); // I2C Initialisieren
  i2c_oled_initall(i2coledssd);  // OLED Initialisieren danach !
}
void loop() {  // Schleife
  // 64x48 Pixel OLED
  char buffer[40]; // Buffer für die Zeichenaushabe
  static int yoffset = 50; // Scrollt von unten rein.
  disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen
  // dann einzelne Zeilen ausgeben, dabei mit einem Offset yoffset.
  disp_print_xy_lcd(0, 0+yoffset, (unsigned char*)"the quick", COLOR_WHITE, 0);
  disp_print_xy_lcd(0, 10+yoffset, (unsigned char*)"brown fox", COLOR_WHITE, 0);
  disp_print_xy_lcd(0, 20+yoffset, (unsigned char*)"jumps over", COLOR_WHITE, 0);
  disp_print_xy_lcd(0, 30+yoffset, (unsigned char*)"the lazy", COLOR_WHITE, 0);
  disp_print_xy_lcd(0, 40+yoffset, (unsigned char*)"dog", COLOR_WHITE, 0);
  disp_print_xy_lcd(0, 50+yoffset, (unsigned char*)"0123456890", COLOR_WHITE, 0);
  disp_lcd_frombuffer();  // ann erste buffer ans OLED uebertragen.
  delay(10); // Eigentlich nicht noetig aber bei schnellen CPUs sicherer
  yoffset = yoffset - 1; // Zeilenweise scrollen 1 Pixel
  if (yoffset < -60) yoffset = 50; // wieder von vorne anfangen
} // Ende der Schleife
```

Was passiert? Der Text „the quick borwn fox jumps over the lazy dog 0123456789“ wird auf dem OLED-DIsplay ausgegeben und scrollt, in mehrere Zeilen zerlegt, vertikal über den Bildschirm. Das Ganze wiederholt sich dann.

5.10 Warmlaufen der Brennstoffzelle

Es ist zu beachten, dass die Brennstoffzelle nicht sofort die volle Stromstärke zur Verfügung stellt. Je nachdem, wie lange die Brennstoffzelle ungenutzt gelagert wurde, kann es bis zu mehrere Minuten dauern, bevor die maximale Nennleistung von der Brennstoffzelle zur Verfügung gestellt wird. Für die Startphase ist es wichtig, einen Stromverbraucher angeschlossen zu haben, damit die Brennstoffzelle in den optimalen Betriebszustand gelangt. Dazu verwenden wir den Verbraucher-Brick mit 50 mA, der direkt an der Brennstoffzelle angeschlossen wird. Die folgende Schaltung betreiben wir für 30 Minuten, um die Brennstoffzelle vorzubereiten. Die LED im Verbraucher-Brick sollte nach kurzer Zeit hell leuchten. Bitte die Brennstoffzelle zu Beginn des Versuchs mit dem hydraulischen Taster entlüften.



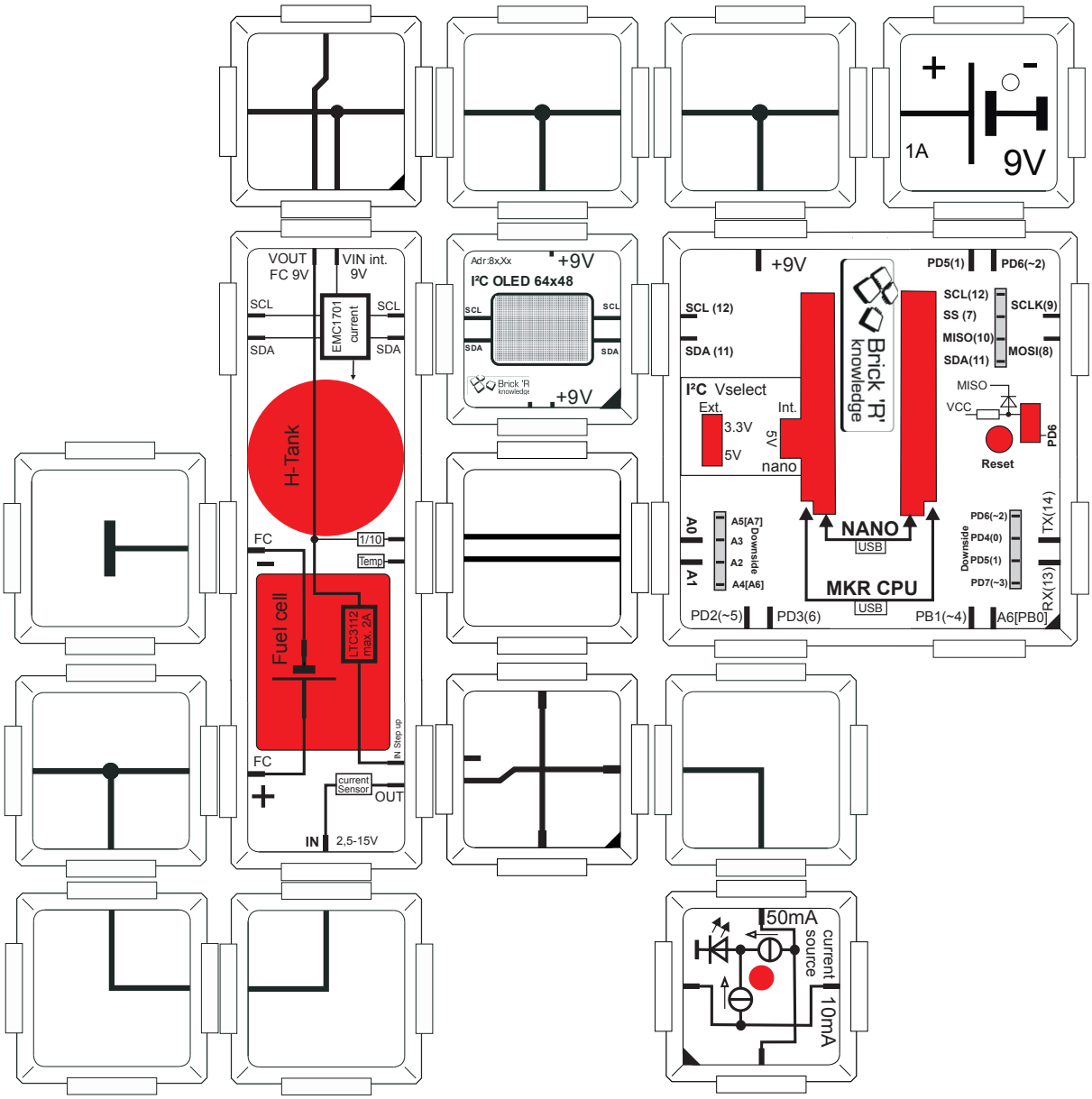
5.11 Versuchsaufbau mit Brennstoffzelle, MKR-Brick, Display und Verbraucher

Um Messungen mit der Brennstoffzelle durchzuführen, benötigen wir den nächsten Versuchsaufbau. Hier wird der Arduino MKR für die Auswertung verwendet. Die Variable für die Versuchsanordnung muß auf „`int versuch = 11;`“ gesetzt werden. Die Ergebnisse werden auf dem OLED-Display dargestellt und werden alternativ auch auf dem seriellen Arduino-Terminal ausgegeben. Als normalisierter Verbraucher dient unser Verbraucher-Brick mit 10 mA oder 50 mA. In der nachfolgenden Versuchsanordnung ist er so angeschlossen, dass 50 mA verbraucht werden.

Der OLED-Brick ist über den i2c-Bus angeschlossen, ebenso die Temperatur- und Spannungsüberwachung des Brennstoffzellen-Bricks. Die Spannungsversorgung der i2c-Elektronik im Brennstoffzellen-Brick wird über das Netzteil realisiert. Je nachdem, wie man den Verbraucher-Brick anschließt, benötigt dieser entweder ca. 10 mA oder ca. 50 mA. Den Verlauf in Bezug auf die Betriebsdauer von Temperaturentwicklung, Stromerzeugung und Gesamtlaufzeit der Brennstoffzelle bei Verwendung eines vollen HydroStik Pro Behälters kann man mit dem MKR Brick messen und protokollieren. Solange die rote LED im Verbraucher-Brick leuchtet, wird Strom durch die Brennstoffzelle erzeugt.

Würde man den folgenden Versuchsaufbau ohne das zuvor beschriebene Warmlaufen der Brennstoffzelle betreiben, so kann es passieren, dass die Brennstoffzelle nicht genügend Strom für den Versuchsaufbau liefert. Die Spannungsmessung hinter dem LTC-Spannunswandler erfolgt über den

Analog-Eingang 0 des MKR-Bricks. Der 1:10 Spannungsteiler im Brennstoffzellen-Brick sorgt dafür,dass auch größere Spannungen als 3,3 gemessen werden können, ohne die MKR-CPU zu beschädigen, da diese nur maximal 3,3 Volt verträgt.

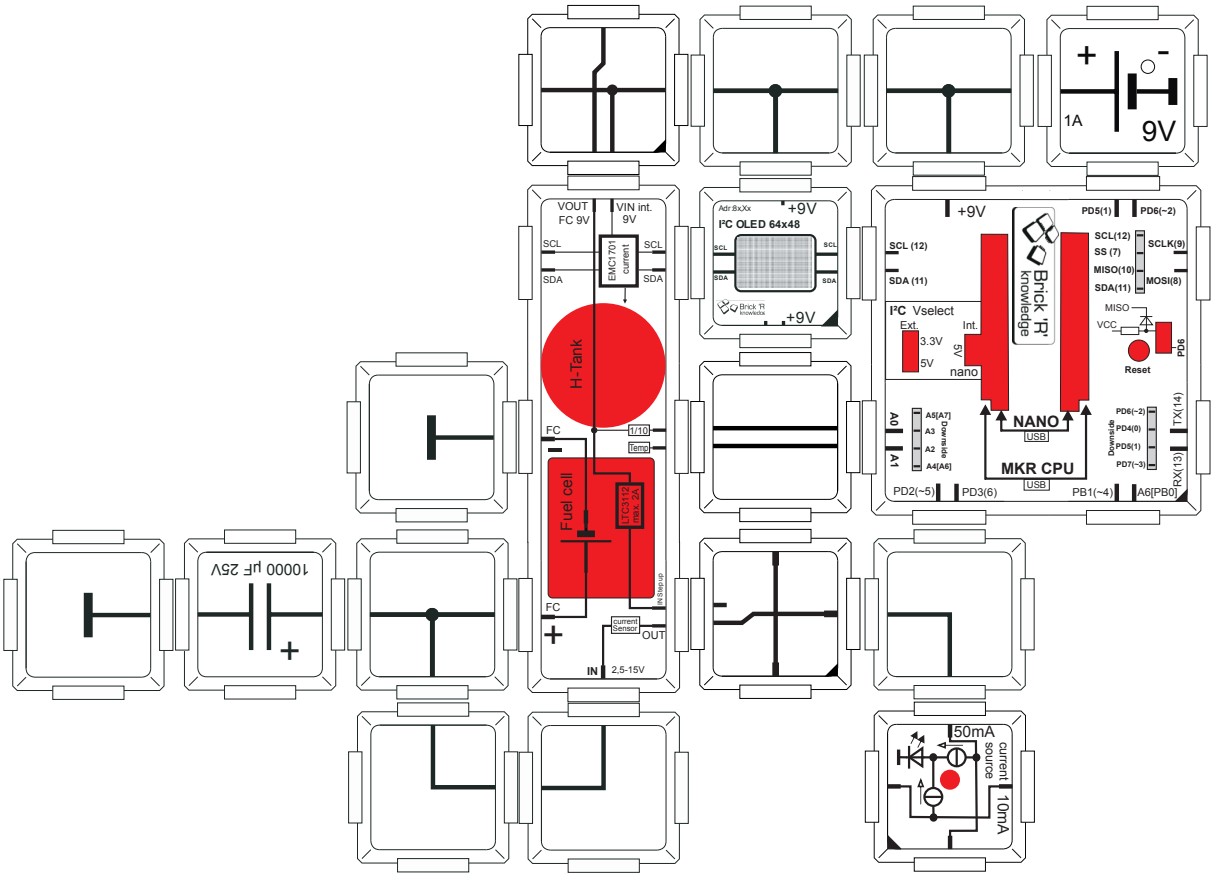


Im OLED-Display werden die Messwerte angezeigt: In der 1. Zeile die Umgebungstemperatur vom EMC, daneben die Spannung hinter dem LTC Spannungswandler. In der 2. Zeile die Temperatur innerhalb der Brennstoffzelle und die Ausgangsspannung der Brennstoffzelle vor dem LTC. In der 3. Zeile der hinter dem Spannungswandler verbrauchte Strom in mA und die daraus resultierende Leistung in mW. Mit einer Spannung von weniger als 2,8 Volt kann der LTC nicht arbeiten und zeigt dann 0mA/0mW an.

Um Ströme und Spannungen zu messen, kann alternativ ein handelsübliches Handheld-Meßinstrument (Voltmeter/Amperemeter) verwendet werden.

5.12 Versuchsaufbau mit Brennstoffzelle, MKR-Brick, Display, Kondensator und Verbraucher

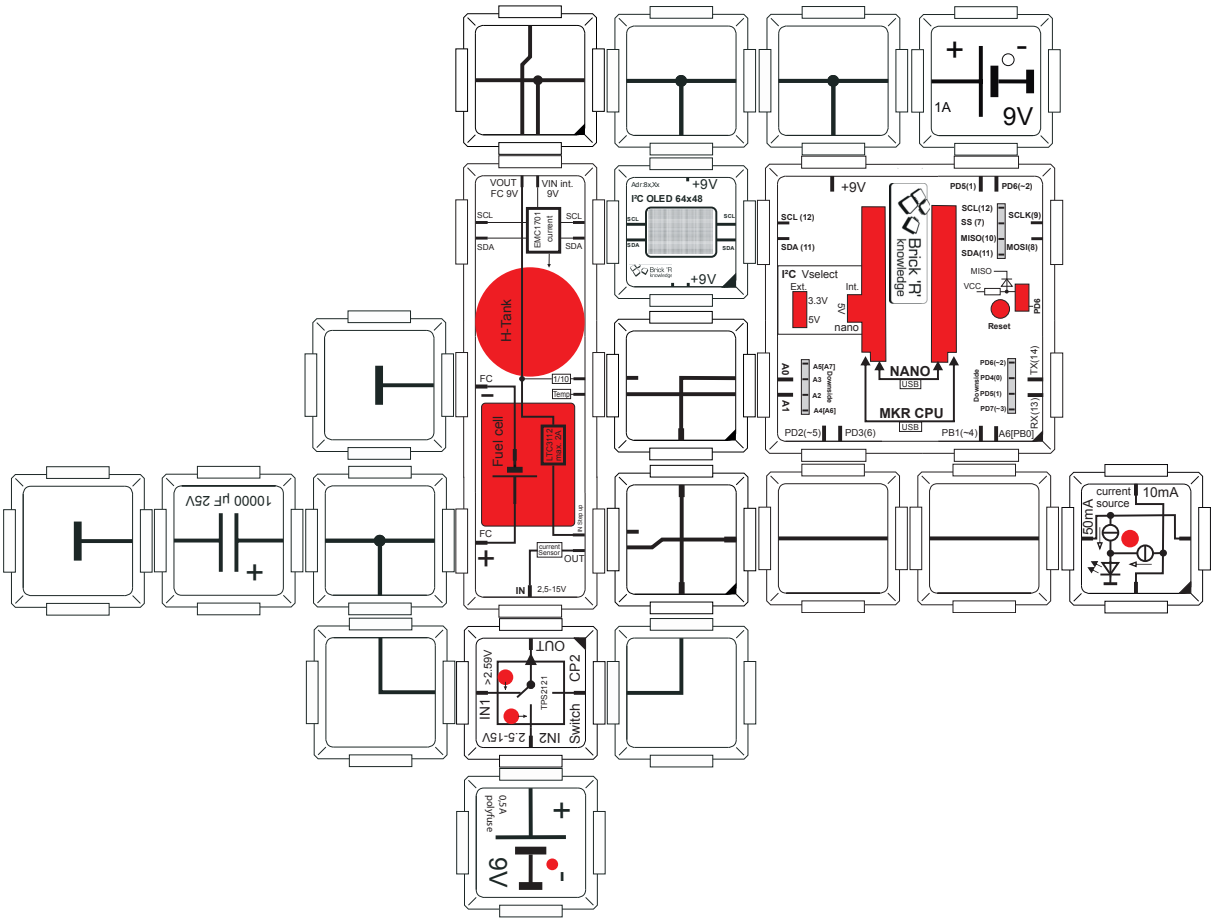
Um die Brennstoffzelle bei der Stromerzeugung zu stabilisieren, bauen wir einen großen Kondensator mit 10.000 µF in die Schaltung ein. Dadurch wird die Brennstoffzelle bei wechselnden Lasten unterstützt. Der Kondensator wirkt hierbei wie ein Stromspeicher, der durch die Brennstoffzelle geladen wird und seine Ladung bei Bedarf sehr schnell abgeben kann. Bei Verwendung eines großen Kondensators kann man die Brennstoffzelle auch bei etwas kürzeren Warmlaufzeiten stabil betreiben. Die Variable für die Versuchsanordnung muß auf „`int versuch = 11;`“ gesetzt werden.

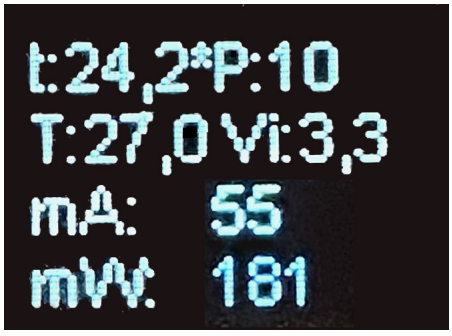


5.13 Versuchsaufbau mit Ausschaltung der Brennstoffzellen-Stromerzeugung

Der nächste Versuchsaufbau benutzt den Schalter unseres Strompfad-Weiche Bricks. Die Variable für die Versuchsanordnung muß auf „`int versuch = 13;`“ gesetzt werden. Sobald der Schalter cp2 auf high (3,3 Volt) gesetzt wird, erfolgt eine zwangsweise Umschaltung der Stromquelle auf in2, auch für den Fall, dass eine Stromquelle an in1 angeschlossen ist und eine genügend hohe Spannung von 2,59 Volt liefert.

Um die Umschaltung beobachten zu könnnen, muss die Variable „const long Laufzeit = ...“ auf „const long Laufzeit = 10“ geändert werden. Nach der angegebenen Anzahl von 10 Sekunden schaltet der Versuchsaufbau aus und als Stromquelle wird die Batterie genutzt anstelle der Brennstoffzelle. Auf dem Display erscheint „Power off“. Um den Versch neu zu starten, muss die Reset-Taste am MKR Brick betätigt werden.



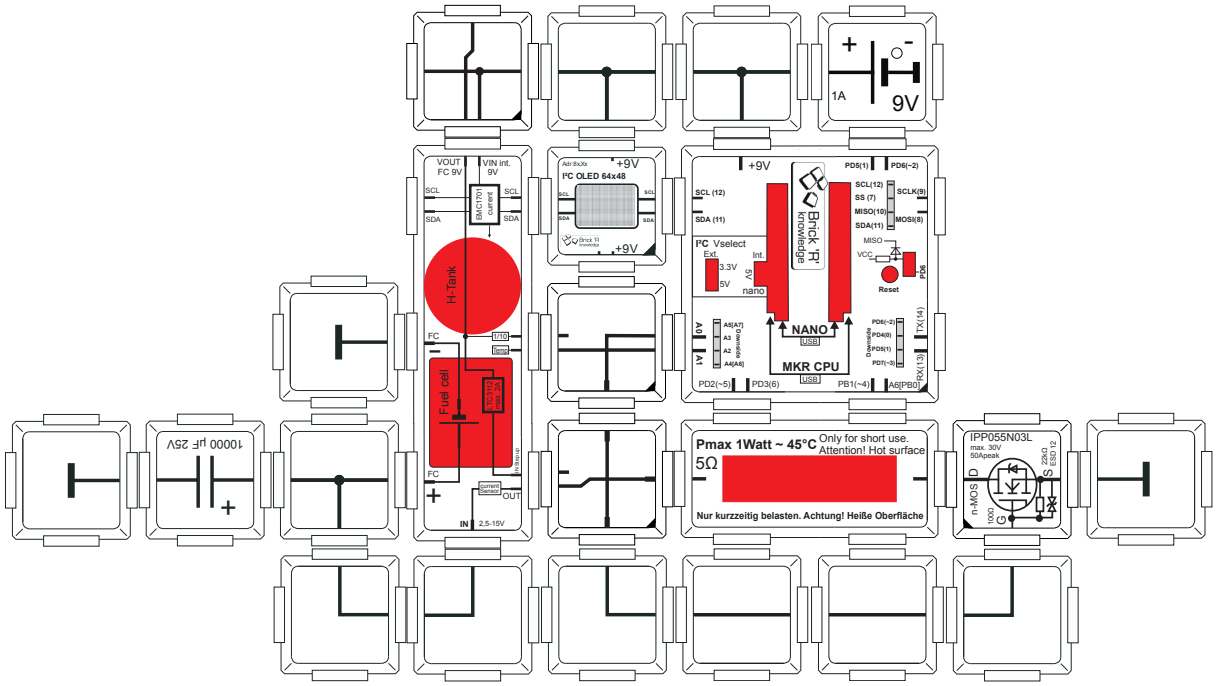


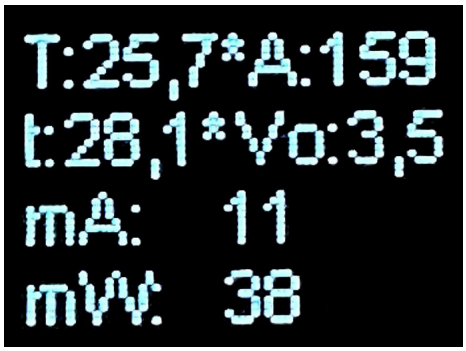
Im OLED-Display werden die Messwerte angezeigt: In der 1. Zeile die Umgebungstemperatur vom EMC, daneben die Anzahl der Sekunden bis zum Ausschalten. In der 2. Zeile die Temperatur innerhalb der Brennstoffzelle und die Ausgangsspannung der Brennstoffzelle vor dem LTC. In der 3. Zeile der hinter dem Spannungswandler verbrauchte Strom in mA und die daraus resultierende Leistung in mW. Mit einer Spannung von weniger als 2,8 Volt kann der LTC nicht arbeiten und zeigt dann 0mA/0mW an.

5.14 Versuchsaufbau mit gesteuerter Strom-Last

Der folgende Versuchsaufbau wird benutzt, um die Strom-Last und davon abhängig auch die Spannung der Brennstoffzelle zu steuern. Diese Steuerung passiert mittels des analogen Ausgangs A0, der unseren n-MOS-Transistor benutzt, um festzulegen, wie viel Strom fließen soll und damit auch welche Spannung die Brennstoffzelle gerade erzeugt. Die Variable für die Versuchsanordnung muß auf „`int versuch = 14;`“ gesetzt werden. Im Display wird oben rechts der Wert des analogen Ausgangs A0 (0-250) gezeigt, der jede Sekunde um 10 hochgezählt wird um nach dem Wert 250 wieder bei 0 zu beginnen. Dabei sieht man, dass die Spannung mit dem Zähler zuerst ansteigt bis ein bestimmter Wert erreicht ist, danach für eine gewisse Zeit auf diesem Spannungswert stehen bleibt um dann im weiteren Verlauf wieder bis auf fast 0 Volt abzufallen. Die Schwellwerte ändern sich in Abhängigkeit von der Temperatur in der Brennstoffzelle. Nach ein paar Minuten Warmlaufzeit werden dabei höhere Spannungen erreicht als zu Beginn des Versuchs. Dieser Versuch sollte nicht über eine längere Zeit

hinweg betrieben werden, da die Brennstoffzelle dadurch einem großen Verschleiß ausgesetzt ist, denn die Stromstärke erreicht bei jedem Durchlauf den Grenzwert der Brennstoffzelle.





Im OLED-Display werden die Messwerte angezeigt: In der 1. Zeile die Umgebungstemperatur vom EMC, daneben der Wert des analogen Ausgangs A0. In der 2. Zeile die Temperatur innerhalb der Brennstoffzelle und die Ausgangsspannung der Brennstoffzelle vor dem LTC. In der 3. Zeile der hinter dem Spannungswandler verbrauchte Strom in mA und die daraus resultierende Leistung in mW.

5.15 Versuchsaufbau mit gesteuerter Strom-Last von 10 bis 12 mA

Nun wollen wir den Sketch so verändern, dass ein konstanter Strom von mindestens 10 mA bis maximal 12 mA fließen soll. Die Variable für die Versuchsanordnung muß auf „`int versuch = 15;`“ gesetzt werden. Die Anzeige im Display sieht analog zum letzten Versuch aus mit dem Unterschied, dass der Wert des analogen Ausgangs A0 konstant gehalten wird, sobald der gewünschte Strom erreicht ist. Sobald mehr als 20 mA Strom fließt, fängt die Kalibration auf 10 mA von vorne an.

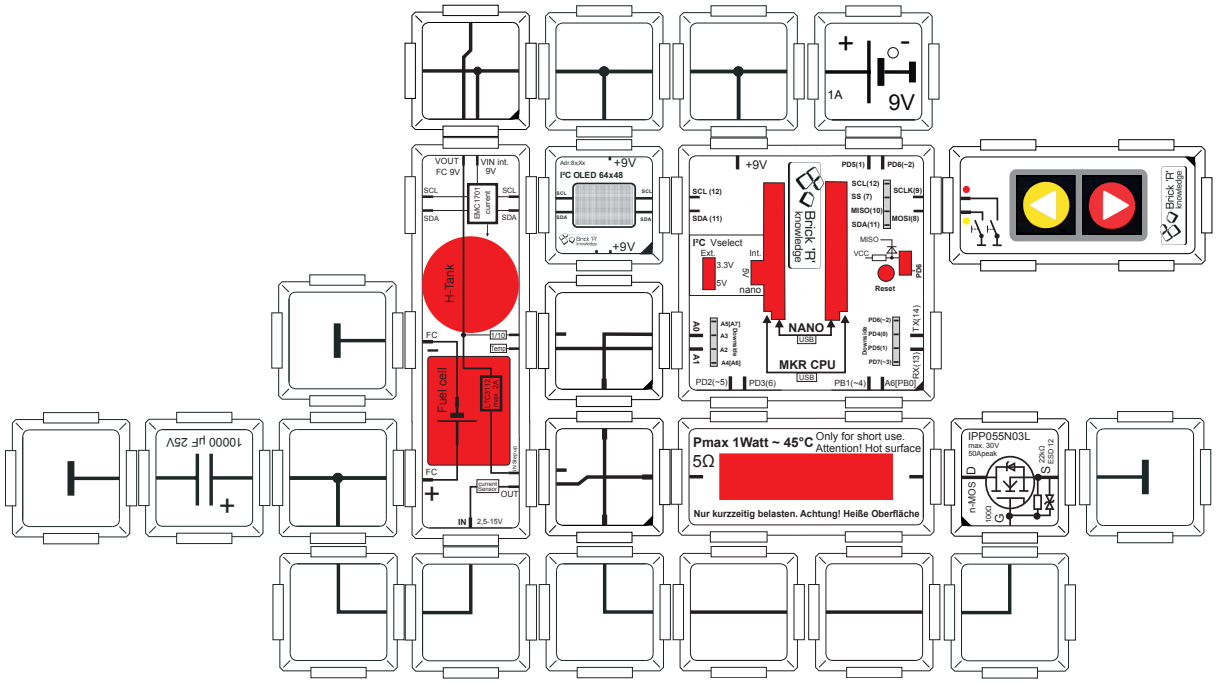
5.16 Versuchsaufbau mit gesteuerter Spannung von 3,0 bis 3,2 V

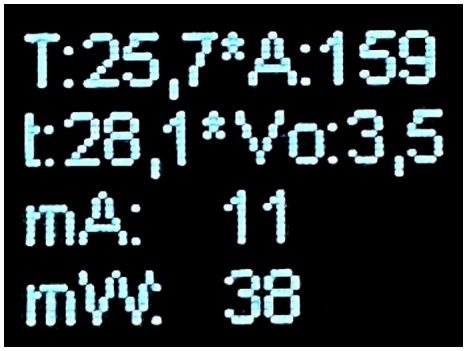
Nun wollen wir den Sketch so verändern, dass eine konstante Spannung von mindestens 3 V bis maximal 3,2 V anliegen soll. Die Variable für die Versuchsanordnung muß auf „`int versuch = 16;`“ gesetzt werden. Die Anzeige im Display sieht analog zum letzten Versuch aus mit dem Unterschied, dass der Wert des analogen Ausgangs A0 konstant gehalten wird, sobald die gewünschte Spannung erreicht ist. Bei mehr als 3,2 V fängt die Kalibration auf 3 V von vorne an. Hierbei ist zu beachten,

dass auch die Stromstärke größer als 0 sein muss, damit sichergestellt wird, dass auch Strom verbraucht wird.

5.17 Versuchsaufbau mit manuell gesteuerter Strom-Last

Der folgende Versuchsaufbau wird benutzt, um die Strom-Last und davon abhängig auch die Spannung der Brennstoffzelle manuell zu steuern. Dafür benutzen wir den Folientaster mit den beiden Pfeilen. Die Variable für die Versuchsanordnung muß auf „`int versuch = 17;`“ gesetzt werden. Im Display wird oben rechts der Wert des analogen Ausgangs A0 (150-180) gezeigt, der mit dem roten Taster um 1 erhöht und mit dem gelben Taster um 1 reduziert werden kann. Der Wertebereich ist in diesem Versuch auf einen sinnvollen Teilbereich von 150 bis 180 reduziert. Die angezeigten Werte für Spannung und Strom ändern sich in Abhängigkeit von der Temperatur in der Brennstoffzelle. Nach ein paar Minuten Warmlaufzeit werden dabei höhere Spannungen erreicht als zu Beginn des Versuchs.





Im OLED-Display werden die Messwerte angezeigt: In der 1. Zeile die Umgebungstemperatur vom EMC, daneben der Wert des analogen Ausgangs A0, den man mit den pfeilen in Einzelschritten hoch und runter schalten kann. In der 2. Zeile die Temperatur innerhalb der Brennstoffzelle und die Ausgangsspannung der Brennstoffzelle vor dem LTC. In der 3. Zeile der hinter dem Spannungswandler verbrauchte Strom in mA und die daraus resultierende Leistung in mW.

```
#include <Wire.h>

// DE_27 OLED Beispiele - Pixelroutinen

#include <avr/pgmspace.h> // Zugriff ins ROM

// Hier ggf Adresse anpassen 78 oder 7A je nach Schalter
#define i2coledssd (0x7A>>1)  // default ist 7A


// -------------------------OLED ---------------------------------------------
// GLO066-D-M2005 -- SSD 1306 driver
// 011110sr  s=sa r=rw bei ssd1306 sa = adressbit optional zu setzen
// 0x78
// 0x78 und 0x7A je nach schalter...

//
// ************************************************************
// RDK 2014 FONT Sets
//

/*******************************************************************************
*                                                                              *
* This file is generated by BitFontCreator Pro v3.0                            *
* by Iseatech Software   http://www.iseasoft.com/bfc.htm                       *
*                        support@iseatech.com                                  *
*                                                                              *
* Font name: Arial                                                             *
* Font width: 0 (proportional font)                                            *
* Font height: 27                                                              *
* Encode: Unicode                                                              *
*                                                                              *
* Data length: 8 bits                                                          *
* Invert bits: No                                                              *
* Data format: Big Endian, Row based, Row preferred, Unpacked                  *
*                                                                              *
* Create time: 13:31 12-01-2011                                                *
*******************************************************************************/
const unsigned char fontArial14h_data_tablep[] PROGMEM =
{

/* character 0x0020 (' '): [width=3, offset= 0x0000 (0) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x0021 ('!'): [width=2, offset= 0x000E (14) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x00, 0x80, 0x00, 0x00, 0x00,

/* character 0x0022 ('"'): [width=4, offset= 0x001C (28) ] */
  0x00, 0x00, 0x00, 0xA0, 0xA0, 0xA0, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x0023 ('#'): [width=6, offset= 0x002A (42) ] */
  0x00, 0x00, 0x00, 0x28, 0x28, 0xF8, 0x50, 0x50,
  0xF8, 0xA0, 0xA0, 0x00, 0x00, 0x00,

/* character 0x0024 ('$'): [width=6, offset= 0x0038 (56) ] */
  0x00, 0x00, 0x00, 0x70, 0xA8, 0xA0, 0x70, 0x28,
  0x28, 0xA8, 0x70, 0x20, 0x00, 0x00,

/* character 0x0025 ('%'): [width=10, offset= 0x0046 (70) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x62, 0x00,
  0x94, 0x00, 0x94, 0x00, 0x68, 0x00, 0x0B, 0x00,
  0x14, 0x80, 0x14, 0x80, 0x23, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00,

/* character 0x0026 ('&'): [width=7, offset= 0x0062 (98) ] */
  0x00, 0x00, 0x00, 0x30, 0x48, 0x48, 0x30, 0x50,
  0x8C, 0x88, 0x74, 0x00, 0x00, 0x00,

/* character 0x0027 (''''): [width=2, offset= 0x0070 (112) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x0028 ('('): [width=4, offset= 0x007E (126) ] */
  0x00, 0x00, 0x00, 0x20, 0x40, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x40, 0x20, 0x00,

/* character 0x0029 (')'): [width=4, offset= 0x008C (140) ] */
```

```
  0x00, 0x00, 0x00, 0x80, 0x40, 0x20, 0x20, 0x20,
  0x20, 0x20, 0x20, 0x40, 0x80, 0x00,

/* character 0x002A ('*'): [width=4, offset= 0x009A (154) ] */
  0x00, 0x00, 0x00, 0x40, 0xE0, 0x40, 0xA0, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x002B ('+'): [width=6, offset= 0x00A8 (168) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x20, 0xF8,
  0x20, 0x20, 0x00, 0x00, 0x00, 0x00,

/* character 0x002C (','): [width=3, offset= 0x00B6 (182) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x40, 0x40, 0x40, 0x00,

/* character 0x002D ('-'): [width=4, offset= 0x00C4 (196) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0xE0, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x002E ('.'): [width=3, offset= 0x00D2 (210) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x40, 0x00, 0x00, 0x00,

/* character 0x002F ('/'): [width=3, offset= 0x00E0 (224) ] */
  0x00, 0x00, 0x00, 0x20, 0x20, 0x40, 0x40, 0x40,
  0x40, 0x80, 0x80, 0x00, 0x00, 0x00,

/* character 0x0030 ('0'): [width=6, offset= 0x00EE (238) ] */
  0x00, 0x00, 0x00, 0x70, 0x88, 0x88, 0x88, 0x88,
  0x88, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x0031 ('1'): [width=6, offset= 0x00FC (252) ] */
  0x00, 0x00, 0x00, 0x20, 0x60, 0xA0, 0x20, 0x20,
  0x20, 0x20, 0x20, 0x00, 0x00, 0x00,

/* character 0x0032 ('2'): [width=6, offset= 0x010A (266) ] */
  0x00, 0x00, 0x00, 0x70, 0x88, 0x08, 0x08, 0x10,
  0x20, 0x40, 0xF8, 0x00, 0x00, 0x00,

/* character 0x0033 ('3'): [width=6, offset= 0x0118 (280) ] */
  0x00, 0x00, 0x00, 0x70, 0x88, 0x08, 0x30, 0x08,
  0x08, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x0034 ('4'): [width=6, offset= 0x0126 (294) ] */
  0x00, 0x00, 0x00, 0x10, 0x30, 0x50, 0x50, 0x90,
  0xF8, 0x10, 0x10, 0x00, 0x00, 0x00,

/* character 0x0035 ('5'): [width=6, offset= 0x0134 (308) ] */
  0x00, 0x00, 0x00, 0x78, 0x40, 0x80, 0xF0, 0x08,
  0x08, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x0036 ('6'): [width=6, offset= 0x0142 (322) ] */
  0x00, 0x00, 0x00, 0x70, 0x88, 0x80, 0xF0, 0x88,
  0x88, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x0037 ('7'): [width=6, offset= 0x0150 (336) ] */
  0x00, 0x00, 0x00, 0xF8, 0x10, 0x10, 0x20, 0x20,
  0x40, 0x40, 0x40, 0x00, 0x00, 0x00,

/* character 0x0038 ('8'): [width=6, offset= 0x015E (350) ] */
  0x00, 0x00, 0x00, 0x70, 0x88, 0x88, 0x70, 0x88,
  0x88, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x0039 ('9'): [width=6, offset= 0x016C (364) ] */
  0x00, 0x00, 0x00, 0x70, 0x88, 0x88, 0x88, 0x78,
  0x08, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x003A (':'): [width=3, offset= 0x017A (378) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00,
  0x00, 0x00, 0x80, 0x00, 0x00, 0x00,

/* character 0x003B (';'): [width=3, offset= 0x0188 (392) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00,
  0x00, 0x00, 0x80, 0x80, 0x80, 0x00,

/* character 0x003C ('<'): [width=6, offset= 0x0196 (406) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x70, 0x80,
  0x70, 0x08, 0x00, 0x00, 0x00, 0x00,

/* character 0x003D ('='): [width=6, offset= 0x01A4 (420) ] */
```

```
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF8, 0x00,
  0xF8, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x003E ('>'): [width=6, offset= 0x01B2 (434) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x70, 0x08,
  0x70, 0x80, 0x00, 0x00, 0x00, 0x00,

/* character 0x003F ('?'): [width=6, offset= 0x01C0 (448) ] */
  0x00, 0x00, 0x00, 0x70, 0x88, 0x08, 0x10, 0x20,
  0x20, 0x00, 0x20, 0x00, 0x00, 0x00,

/* character 0x0040 ('@'): [width=11, offset= 0x01CE (462) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1F, 0x00,
  0x60, 0x80, 0x4D, 0x40, 0x93, 0x40, 0xA2, 0x40,
  0xA2, 0x40, 0xA6, 0x80, 0x9B, 0x00, 0x40, 0x40,
  0x3F, 0x80, 0x00, 0x00,

/* character 0x0041 ('A'): [width=8, offset= 0x01EA (490) ] */
  0x00, 0x00, 0x00, 0x10, 0x28, 0x28, 0x28, 0x44,
  0x7C, 0x82, 0x82, 0x00, 0x00, 0x00,

/* character 0x0042 ('B'): [width=7, offset= 0x01F8 (504) ] */
  0x00, 0x00, 0x00, 0xF8, 0x84, 0x84, 0xFC, 0x84,
  0x84, 0x84, 0xF8, 0x00, 0x00, 0x00,

/* character 0x0043 ('C'): [width=7, offset= 0x0206 (518) ] */
  0x00, 0x00, 0x00, 0x38, 0x44, 0x80, 0x80, 0x80,
  0x80, 0x44, 0x38, 0x00, 0x00, 0x00,

/* character 0x0044 ('D'): [width=7, offset= 0x0214 (532) ] */
  0x00, 0x00, 0x00, 0xF0, 0x88, 0x84, 0x84, 0x84,
  0x84, 0x88, 0xF0, 0x00, 0x00, 0x00,

/* character 0x0045 ('E'): [width=6, offset= 0x0222 (546) ] */
  0x00, 0x00, 0x00, 0xF8, 0x80, 0x80, 0xF8, 0x80,
  0x80, 0x80, 0xF8, 0x00, 0x00, 0x00,

/* character 0x0046 ('F'): [width=6, offset= 0x0230 (560) ] */
  0x00, 0x00, 0x00, 0xF8, 0x80, 0x80, 0xF0, 0x80,
  0x80, 0x80, 0x80, 0x00, 0x00, 0x00,

/* character 0x0047 ('G'): [width=8, offset= 0x023E (574) ] */
  0x00, 0x00, 0x00, 0x38, 0x44, 0x82, 0x80, 0x8E,
  0x82, 0x44, 0x38, 0x00, 0x00, 0x00,

/* character 0x0048 ('H'): [width=7, offset= 0x024C (588) ] */
  0x00, 0x00, 0x00, 0x84, 0x84, 0x84, 0xFC, 0x84,
  0x84, 0x84, 0x84, 0x00, 0x00, 0x00,

/* character 0x0049 ('I'): [width=2, offset= 0x025A (602) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x00, 0x00, 0x00,

/* character 0x004A ('J'): [width=5, offset= 0x0268 (616) ] */
  0x00, 0x00, 0x00, 0x10, 0x10, 0x10, 0x10, 0x10,
  0x90, 0x90, 0x60, 0x00, 0x00, 0x00,

/* character 0x004B ('K'): [width=7, offset= 0x0276 (630) ] */
  0x00, 0x00, 0x00, 0x84, 0x88, 0x90, 0xB0, 0xD0,
  0x88, 0x88, 0x84, 0x00, 0x00, 0x00,

/* character 0x004C ('L'): [width=6, offset= 0x0284 (644) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0xF8, 0x00, 0x00, 0x00,

/* character 0x004D ('M'): [width=8, offset= 0x0292 (658) ] */
  0x00, 0x00, 0x00, 0x82, 0xC6, 0xC6, 0xAA, 0xAA,
  0xAA, 0x92, 0x92, 0x00, 0x00, 0x00,

/* character 0x004E ('N'): [width=7, offset= 0x02A0 (672) ] */
  0x00, 0x00, 0x00, 0x84, 0xC4, 0xA4, 0xA4, 0x94,
  0x94, 0x8C, 0x84, 0x00, 0x00, 0x00,

/* character 0x004F ('O'): [width=8, offset= 0x02AE (686) ] */
  0x00, 0x00, 0x00, 0x38, 0x44, 0x82, 0x82, 0x82,
  0x82, 0x44, 0x38, 0x00, 0x00, 0x00,

/* character 0x0050 ('P'): [width=6, offset= 0x02BC (700) ] */
  0x00, 0x00, 0x00, 0xF0, 0x88, 0x88, 0x88, 0xF0,
  0x80, 0x80, 0x80, 0x00, 0x00, 0x00,
```

```
/* character 0x0051 ('Q'): [width=8, offset= 0x02CA (714) ] */
  0x00, 0x00, 0x00, 0x38, 0x44, 0x82, 0x82, 0x82,
  0x9A, 0x44, 0x3A, 0x00, 0x00, 0x00,

/* character 0x0052 ('R'): [width=7, offset= 0x02D8 (728) ] */
  0x00, 0x00, 0x00, 0xF8, 0x84, 0x84, 0xF8, 0x90,
  0x88, 0x88, 0x84, 0x00, 0x00, 0x00,

/* character 0x0053 ('S'): [width=7, offset= 0x02E6 (742) ] */
  0x00, 0x00, 0x00, 0x78, 0x84, 0x80, 0x60, 0x18,
  0x04, 0x84, 0x78, 0x00, 0x00, 0x00,

/* character 0x0054 ('T'): [width=6, offset= 0x02F4 (756) ] */
  0x00, 0x00, 0x00, 0xF8, 0x20, 0x20, 0x20, 0x20,
  0x20, 0x20, 0x20, 0x00, 0x00, 0x00,

/* character 0x0055 ('U'): [width=7, offset= 0x0302 (770) ] */
  0x00, 0x00, 0x00, 0x84, 0x84, 0x84, 0x84, 0x84,
  0x84, 0x84, 0x78, 0x00, 0x00, 0x00,

/* character 0x0056 ('V'): [width=8, offset= 0x0310 (784) ] */
  0x00, 0x00, 0x00, 0x82, 0x82, 0x44, 0x44, 0x28,
  0x28, 0x10, 0x10, 0x00, 0x00, 0x00,

/* character 0x0057 ('W'): [width=11, offset= 0x031E (798) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x84, 0x20,
  0x8A, 0x20, 0x4A, 0x40, 0x4A, 0x40, 0x51, 0x40,
  0x51, 0x40, 0x20, 0x80, 0x20, 0x80, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00,

/* character 0x0058 ('X'): [width=7, offset= 0x033A (826) ] */
  0x00, 0x00, 0x00, 0x84, 0x48, 0x48, 0x30, 0x30,
  0x48, 0x48, 0x84, 0x00, 0x00, 0x00,

/* character 0x0059 ('Y'): [width=8, offset= 0x0348 (840) ] */
  0x00, 0x00, 0x00, 0x82, 0x44, 0x44, 0x28, 0x10,
  0x10, 0x10, 0x10, 0x00, 0x00, 0x00,

/* character 0x005A ('Z'): [width=7, offset= 0x0356 (854) ] */
  0x00, 0x00, 0x00, 0x7C, 0x08, 0x10, 0x10, 0x20,
  0x20, 0x40, 0xFC, 0x00, 0x00, 0x00,

/* character 0x005B ('['): [width=3, offset= 0x0364 (868) ] */
  0x00, 0x00, 0x00, 0xC0, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0xC0, 0x00,

/* character 0x005C ('\'): [width=3, offset= 0x0372 (882) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0x40, 0x40, 0x40,
  0x40, 0x20, 0x20, 0x00, 0x00, 0x00,

/* character 0x005D (']'): [width=3, offset= 0x0380 (896) ] */
  0x00, 0x00, 0x00, 0xC0, 0x40, 0x40, 0x40, 0x40,
  0x40, 0x40, 0x40, 0x40, 0xC0, 0x00,

/* character 0x005E ('^'): [width=5, offset= 0x038E (910) ] */
  0x00, 0x00, 0x00, 0x20, 0x50, 0x50, 0x88, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x005F ('_'): [width=6, offset= 0x039C (924) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0xFC, 0x00,

/* character 0x0060 ('`'): [width=4, offset= 0x03AA (938) ] */
  0x00, 0x00, 0x00, 0x80, 0x40, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x0061 ('a'): [width=6, offset= 0x03B8 (952) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x70, 0x88, 0x78,
  0x88, 0x98, 0x68, 0x00, 0x00, 0x00,

/* character 0x0062 ('b'): [width=6, offset= 0x03C6 (966) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0xB0, 0xC8, 0x88,
  0x88, 0xC8, 0xB0, 0x00, 0x00, 0x00,

/* character 0x0063 ('c'): [width=6, offset= 0x03D4 (980) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x70, 0x88, 0x80,
  0x80, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x0064 ('d'): [width=6, offset= 0x03E2 (994) ] */
```

```
  0x00, 0x00, 0x00, 0x08, 0x08, 0x68, 0x98, 0x88,
  0x88, 0x98, 0x68, 0x00, 0x00, 0x00,

/* character 0x0065 ('e'): [width=6, offset= 0x03F0 (1008) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x70, 0x88, 0xF8,
  0x80, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x0066 ('f'): [width=4, offset= 0x03FE (1022) ] */
  0x00, 0x00, 0x00, 0x20, 0x40, 0xE0, 0x40, 0x40,
  0x40, 0x40, 0x40, 0x00, 0x00, 0x00,

/* character 0x0067 ('g'): [width=6, offset= 0x040C (1036) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x68, 0x98, 0x88,
  0x88, 0x98, 0x68, 0x08, 0xF0, 0x00,

/* character 0x0068 ('h'): [width=6, offset= 0x041A (1050) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0xB0, 0xC8, 0x88,
  0x88, 0x88, 0x88, 0x00, 0x00, 0x00,

/* character 0x0069 ('i'): [width=2, offset= 0x0428 (1064) ] */
  0x00, 0x00, 0x00, 0x80, 0x00, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x00, 0x00, 0x00,

/* character 0x006A ('j'): [width=2, offset= 0x0436 (1078) ] */
  0x00, 0x00, 0x00, 0x80, 0x00, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x00, 0x00,

/* character 0x006B ('k'): [width=5, offset= 0x0444 (1092) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0x90, 0xA0, 0xC0,
  0xA0, 0xA0, 0x90, 0x00, 0x00, 0x00,

/* character 0x006C ('l'): [width=2, offset= 0x0452 (1106) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x00, 0x00, 0x00,

/* character 0x006D ('m'): [width=8, offset= 0x0460 (1120) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0xBC, 0xD2, 0x92,
  0x92, 0x92, 0x92, 0x00, 0x00, 0x00,

/* character 0x006E ('n'): [width=6, offset= 0x046E (1134) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x88, 0x88,
  0x88, 0x88, 0x88, 0x00, 0x00, 0x00,

/* character 0x006F ('o'): [width=6, offset= 0x047C (1148) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x70, 0x88, 0x88,
  0x88, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x0070 ('p'): [width=6, offset= 0x048A (1162) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0xB0, 0xC8, 0x88,
  0x88, 0xC8, 0xB0, 0x80, 0x80, 0x00,

/* character 0x0071 ('q'): [width=6, offset= 0x0498 (1176) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x68, 0x98, 0x88,
  0x88, 0x98, 0x68, 0x08, 0x08, 0x00,

/* character 0x0072 ('r'): [width=4, offset= 0x04A6 (1190) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0xA0, 0xC0, 0x80,
  0x80, 0x80, 0x80, 0x00, 0x00, 0x00,

/* character 0x0073 ('s'): [width=6, offset= 0x04B4 (1204) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x70, 0x88, 0x60,
  0x10, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x0074 ('t'): [width=3, offset= 0x04C2 (1218) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0xC0, 0x80, 0x80,
  0x80, 0x80, 0xC0, 0x00, 0x00, 0x00,

/* character 0x0075 ('u'): [width=6, offset= 0x04D0 (1232) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x88, 0x88,
  0x88, 0x98, 0x68, 0x00, 0x00, 0x00,

/* character 0x0076 ('v'): [width=6, offset= 0x04DE (1246) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x88, 0x50,
  0x50, 0x20, 0x20, 0x00, 0x00, 0x00,

/* character 0x0077 ('w'): [width=10, offset= 0x04EC (1260) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x88, 0x80, 0x94, 0x80, 0x55, 0x00,
  0x55, 0x00, 0x22, 0x00, 0x22, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00,
```

```
/* character 0x0078 ('x'): [width=6, offset= 0x0508 (1288) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x50, 0x20,
  0x20, 0x50, 0x88, 0x00, 0x00, 0x00,

/* character 0x0079 ('y'): [width=6, offset= 0x0516 (1302) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x88, 0x50,
  0x50, 0x20, 0x20, 0x20, 0x40, 0x00,

/* character 0x007A ('z'): [width=6, offset= 0x0524 (1316) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0xF8, 0x10, 0x20,
  0x20, 0x40, 0xF8, 0x00, 0x00, 0x00,

/* character 0x007B ('{'): [width=4, offset= 0x0532 (1330) ] */
  0x00, 0x00, 0x00, 0x20, 0x40, 0x40, 0x40, 0x80,
  0x40, 0x40, 0x40, 0x40, 0x20, 0x00,

/* character 0x007C ('|'): [width=2, offset= 0x0540 (1344) ] */
  0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x00,

/* character 0x007D ('}'): [width=4, offset= 0x054E (1358) ] */
  0x00, 0x00, 0x00, 0x40, 0x20, 0x20, 0x20, 0x10,
  0x20, 0x20, 0x20, 0x20, 0x40, 0x00,

/* character 0x007E ('~'): [width=6, offset= 0x055C (1372) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE8, 0xB0,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x007F ('□'): [width=8, offset= 0x056A (1386) ] */
  0x00, 0x00, 0x00, 0xE0, 0xA0, 0xA0, 0xA0, 0xA0,
  0xA0, 0xA0, 0xE0, 0x00, 0x00, 0x00,

/* character 0x00A2 ('¢'): [width=6, offset= 0x0578 (1400) ] */
  0x00, 0x00, 0x00, 0x10, 0x10, 0x70, 0xA8, 0xA0,
  0xA0, 0xA8, 0x70, 0x40, 0x40, 0x00,

/* character 0x00A3 ('£'): [width=6, offset= 0x0586 (1414) ] */
  0x00, 0x00, 0x00, 0x30, 0x48, 0x40, 0x40, 0xE0,
  0x40, 0x60, 0x98, 0x00, 0x00, 0x00,

/* character 0x00A5 ('¥'): [width=6, offset= 0x0594 (1428) ] */
  0x00, 0x00, 0x00, 0x88, 0x88, 0x50, 0x50, 0xF8,
  0x20, 0xF8, 0x20, 0x00, 0x00, 0x00,

/* character 0x00A7 ('§'): [width=6, offset= 0x05A2 (1442) ] */
  0x00, 0x00, 0x00, 0x70, 0x88, 0x40, 0xE0, 0x90,
  0x48, 0x28, 0x10, 0x88, 0x70, 0x00,

/* character 0x00A9 ('©'): [width=8, offset= 0x05B0 (1456) ] */
  0x00, 0x00, 0x00, 0x3C, 0x42, 0x9D, 0xA1, 0xA5,
  0x99, 0x42, 0x3C, 0x00, 0x00, 0x00,

/* character 0x00AC ('¬'): [width=6, offset= 0x05BE (1470) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF8, 0x08,
  0x08, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x00AE ('®'): [width=8, offset= 0x05CC (1484) ] */
  0x00, 0x00, 0x00, 0x3C, 0x42, 0xB9, 0xA5, 0xB9,
  0xA5, 0x42, 0x3C, 0x00, 0x00, 0x00,

/* character 0x00B0 ('°'): [width=4, offset= 0x05DA (1498) ] */
  0x00, 0x00, 0x00, 0xE0, 0xA0, 0xE0, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x00B1 ('±'): [width=6, offset= 0x05E8 (1512) ] */
  0x00, 0x00, 0x00, 0x00, 0x20, 0x20, 0xF8, 0x20,
  0x20, 0x00, 0xF8, 0x00, 0x00, 0x00,

/* character 0x00B2 ('²'): [width=4, offset= 0x05F6 (1526) ] */
  0x00, 0x00, 0x00, 0xE0, 0x20, 0x40, 0xE0, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x00B3 ('³'): [width=4, offset= 0x0604 (1540) ] */
  0x00, 0x00, 0x00, 0xE0, 0x40, 0x20, 0xE0, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x00B5 ('µ'): [width=6, offset= 0x0612 (1554) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x88, 0x88,
  0x88, 0x88, 0xF8, 0x80, 0x80, 0x00,
```

```
/* character 0x00B9 ('¹'): [width=4, offset= 0x0620 (1568) ] */
  0x00, 0x00, 0x00, 0x20, 0x60, 0x20, 0x20, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x00BA ('º'): [width=5, offset= 0x062E (1582) ] */
  0x00, 0x00, 0x00, 0x60, 0x90, 0x90, 0x60, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

/* character 0x00C4 ('Ä'): [width=8, offset= 0x063C (1596) ] */
  0x00, 0x28, 0x00, 0x10, 0x28, 0x28, 0x28, 0x44,
  0x7C, 0x82, 0x82, 0x00, 0x00, 0x00,

/* character 0x00D6 ('Ö'): [width=8, offset= 0x064A (1610) ] */
  0x00, 0x28, 0x00, 0x38, 0x44, 0x82, 0x82, 0x82,
  0x82, 0x44, 0x38, 0x00, 0x00, 0x00,

/* character 0x00DC ('Ü'): [width=7, offset= 0x0658 (1624) ] */
  0x00, 0x28, 0x00, 0x84, 0x84, 0x84, 0x84, 0x84,
  0x84, 0x84, 0x78, 0x00, 0x00, 0x00,

/* character 0x00DF ('ß'): [width=7, offset= 0x0666 (1638) ] */
  0x00, 0x00, 0x00, 0x70, 0x88, 0x88, 0x90, 0x98,
  0x84, 0xA4, 0x98, 0x00, 0x00, 0x00,

/* character 0x00E4 ('ä'): [width=6, offset= 0x0674 (1652) ] */
  0x00, 0x00, 0x00, 0x50, 0x00, 0x70, 0x88, 0x78,
  0x88, 0x98, 0x68, 0x00, 0x00, 0x00,

/* character 0x00F6 ('ö'): [width=6, offset= 0x0682 (1666) ] */
  0x00, 0x00, 0x00, 0x50, 0x00, 0x70, 0x88, 0x88,
  0x88, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x00FC ('ü'): [width=6, offset= 0x0690 (1680) ] */
  0x00, 0x00, 0x00, 0x50, 0x00, 0x88, 0x88, 0x88,
  0x88, 0x98, 0x68, 0x00, 0x00, 0x00,

/* character 0x0394 ('?'): [width=8, offset= 0x069E (1694) ] */
  0x00, 0x00, 0x00, 0x10, 0x28, 0x28, 0x44, 0x44,
  0x44, 0x82, 0xFE, 0x00, 0x00, 0x00,

/* character 0x039B ('?'): [width=8, offset= 0x06AC (1708) ] */
  0x00, 0x00, 0x00, 0x10, 0x28, 0x28, 0x44, 0x44,
  0x44, 0x82, 0x82, 0x00, 0x00, 0x00,

/* character 0x03A9 ('O'): [width=8, offset= 0x06BA (1722) ] */
  0x00, 0x00, 0x00, 0x38, 0x44, 0x82, 0x82, 0x82,
  0x82, 0x44, 0xC6, 0x00, 0x00, 0x00,

/* character 0x03B5 ('e'): [width=5, offset= 0x06C8 (1736) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x70, 0x80, 0x60,
  0x80, 0x80, 0x70, 0x00, 0x00, 0x00,

/* character 0x03B8 ('?'): [width=6, offset= 0x06D6 (1750) ] */
  0x00, 0x00, 0x00, 0x70, 0x88, 0x88, 0xF8, 0x88,
  0x88, 0x88, 0x70, 0x00, 0x00, 0x00,

/* character 0x03BC ('μ'): [width=6, offset= 0x06E4 (1764) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x88, 0x88,
  0x88, 0x88, 0xF8, 0x80, 0x80, 0x00,

/* character 0x03C0 ('p'): [width=9, offset= 0x06F2 (1778) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xFF, 0x00, 0x24, 0x00, 0x24, 0x00,
  0x24, 0x00, 0x24, 0x00, 0x24, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00,

/* character 0x263A ('?'): [width=11, offset= 0x070E (1806) ] */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x0F, 0x00, 0x30, 0x80, 0x28, 0x80,
  0x20, 0x80, 0x28, 0x80, 0x17, 0x00, 0x0E, 0x00,
  0x00, 0x00, 0x00, 0x00,

/* character 0x2640 ('?'): [width=8, offset= 0x072A (1834) ] */
  0x00, 0x00, 0x00, 0x3C, 0x42, 0x42, 0x42, 0x44,
  0x38, 0x10, 0x10, 0x3C, 0x00, 0x00,

/* character 0x2642 ('?'): [width=8, offset= 0x0738 (1848) ] */
  0x00, 0x00, 0x02, 0x0E, 0x06, 0x0A, 0x08, 0x78,
  0x44, 0x44, 0x44, 0x78, 0x00, 0x00,
```

```
/* character 0x266B ('?'): [width=8, offset= 0x0746 (1862) ] */
  0x00, 0x00, 0x00, 0x06, 0x1E, 0x22, 0x22, 0x22,
  0x26, 0x6E, 0xE4, 0x60, 0x00, 0x00,

};

/*****************************************************************************************
   Offset table provides the starting offset of each character in the data table.

   If you can calculate the offsets by yourself, undefine USE_OFFSET_TABLE.

   Offset table provides the starting offset of each character in the data table.

   To get the starting offset of character 'A', you can use the following expression:

     const USHORT index = GetIndex('A');
     const ULONG offset = offset_table[index];

*****************************************************************************************/
const unsigned int fontArial14h_offset_tablep[] PROGMEM =
{
/*  offset      offsetHex  -  char    hexcode    decimal */
/*  =======     =========  -  ====    =======    ======= */
  0,         /*         0  -         0020       32      */
  14,        /*         E  -  !      0021       33      */
  28,        /*        1C  -  "      0022       34      */
  42,        /*        2A  -  #      0023       35      */
  56,        /*        38  -  $      0024       36      */
  70,        /*        46  -  %      0025       37      */
  98,        /*        62  -  &      0026       38      */
  112,       /*        70  -  '      0027       39      */
  126,       /*        7E  -  (      0028       40      */
  140,       /*        8C  -  )      0029       41      */
  154,       /*        9A  -  *      002A       42      */
  168,       /*        A8  -  +      002B       43      */
  182,       /*        B6  -  ,      002C       44      */
  196,       /*        C4  -  -      002D       45      */
  210,       /*        D2  -  .      002E       46      */
  224,       /*        E0  -  /      002F       47      */
  238,       /*        EE  -  0      0030       48      */
  252,       /*        FC  -  1      0031       49      */
  266,       /*       10A  -  2      0032       50      */
  280,       /*       118  -  3      0033       51      */
  294,       /*       126  -  4      0034       52      */
  308,       /*       134  -  5      0035       53      */
  322,       /*       142  -  6      0036       54      */
  336,       /*       150  -  7      0037       55      */
  350,       /*       15E  -  8      0038       56      */
  364,       /*       16C  -  9      0039       57      */
  378,       /*       17A  -  :      003A       58      */
  392,       /*       188  -  ;      003B       59      */
  406,       /*       196  -  <      003C       60      */
  420,       /*       1A4  -  =      003D       61      */
  434,       /*       1B2  -  >      003E       62      */
  448,       /*       1C0  -  ?      003F       63      */
  462,       /*       1CE  -  @      0040       64      */
  490,       /*       1EA  -  A      0041       65      */
  504,       /*       1F8  -  B      0042       66      */
  518,       /*       206  -  C      0043       67      */
  532,       /*       214  -  D      0044       68      */
  546,       /*       222  -  E      0045       69      */
  560,       /*       230  -  F      0046       70      */
  574,       /*       23E  -  G      0047       71      */
  588,       /*       24C  -  H      0048       72      */
  602,       /*       25A  -  I      0049       73      */
  616,       /*       268  -  J      004A       74      */
  630,       /*       276  -  K      004B       75      */
  644,       /*       284  -  L      004C       76      */
  658,       /*       292  -  M      004D       77      */
  672,       /*       2A0  -  N      004E       78      */
  686,       /*       2AE  -  O      004F       79      */
  700,       /*       2BC  -  P      0050       80      */
  714,       /*       2CA  -  Q      0051       81      */
  728,       /*       2D8  -  R      0052       82      */
  742,       /*       2E6  -  S      0053       83      */
  756,       /*       2F4  -  T      0054       84      */
  770,       /*       302  -  U      0055       85      */
  784,       /*       310  -  V      0056       86      */
  798,       /*       31E  -  W      0057       87      */
```

```
  826,      /*      33A  -  X       0058       88      */
  840,      /*      348  -  Y       0059       89      */
  854,      /*      356  -  Z       005A       90      */
  868,      /*      364  -  [       005B       91      */
  882,      /*      372  -  \       005C       92      */
  896,      /*      380  -  ]       005D       93      */
  910,      /*      38E  -  ^       005E       94      */
  924,      /*      39C  -  _       005F       95      */
  938,      /*      3AA  -  `       0060       96      */
  952,      /*      3B8  -  a       0061       97      */
  966,      /*      3C6  -  b       0062       98      */
  980,      /*      3D4  -  c       0063       99      */
  994,      /*      3E2  -  d       0064       100     */
 1008,      /*      3F0  -  e       0065       101     */
 1022,      /*      3FE  -  f       0066       102     */
 1036,      /*      40C  -  g       0067       103     */
 1050,      /*      41A  -  h       0068       104     */
 1064,      /*      428  -  i       0069       105     */
 1078,      /*      436  -  j       006A       106     */
 1092,      /*      444  -  k       006B       107     */
 1106,      /*      452  -  l       006C       108     */
 1120,      /*      460  -  m       006D       109     */
 1134,      /*      46E  -  n       006E       110     */
 1148,      /*      47C  -  o       006F       111     */
 1162,      /*      48A  -  p       0070       112     */
 1176,      /*      498  -  q       0071       113     */
 1190,      /*      4A6  -  r       0072       114     */
 1204,      /*      4B4  -  s       0073       115     */
 1218,      /*      4C2  -  t       0074       116     */
 1232,      /*      4D0  -  u       0075       117     */
 1246,      /*      4DE  -  v       0076       118     */
 1260,      /*      4EC  -  w       0077       119     */
 1288,      /*      508  -  x       0078       120     */
 1302,      /*      516  -  y       0079       121     */
 1316,      /*      524  -  z       007A       122     */
 1330,      /*      532  -  {       007B       123     */
 1344,      /*      540  -  |       007C       124     */
 1358,      /*      54E  -  }       007D       125     */
 1372,      /*      55C  -  ~       007E       126     */
 1386,      /*      56A  -  □       007F       127     */
 1400,      /*      578  -  ¢       00A2       162     */
 1414,      /*      586  -  £       00A3       163     */
 1428,      /*      594  -  ¥       00A5       165     */
 1442,      /*      5A2  -  §       00A7       167     */
 1456,      /*      5B0  -  ©       00A9       169     */
 1470,      /*      5BE  -  ¬       00AC       172     */
 1484,      /*      5CC  -  ®       00AE       174     */
 1498,      /*      5DA  -  °       00B0       176     */
 1512,      /*      5E8  -  ±       00B1       177     */
 1526,      /*      5F6  -  ²       00B2       178     */
 1540,      /*      604  -  ³       00B3       179     */
 1554,      /*      612  -  µ       00B5       181     */
 1568,      /*      620  -  ¹       00B9       185     */
 1582,      /*      62E  -  º       00BA       186     */
 1596,      /*      63C  -  Ä       00C4       196     */
 1610,      /*      64A  -  Ö       00D6       214     */
 1624,      /*      658  -  Ü       00DC       220     */
 1638,      /*      666  -  ß       00DF       223     */
 1652,      /*      674  -  ä       00E4       228     */
 1666,      /*      682  -  ö       00F6       246     */
 1680,      /*      690  -  ü       00FC       252     */
 1694,      /*      69E  -  ?       0394       916     */
 1708,      /*      6AC  -  ?       039B       923     */
 1722,      /*      6BA  -  O       03A9       937     */
 1736,      /*      6C8  -  e       03B5       949     */
 1750,      /*      6D6  -  ?       03B8       952     */
 1764,      /*      6E4  -  µ       03BC       956     */
 1778,      /*      6F2  -  p       03C0       960     */
 1806,      /*      70E  -  ?       263A       9786    */
 1834,      /*      72A  -  ?       2640       9792    */
 1848,      /*      738  -  ?       2642       9794    */
 1862,      /*      746  -  ?       266B       9835    */

 1876,      /*      754  -  extra address: the end of the last character's imagebits data */
};

/*****************************************************************************************
   Width table provides the width of each character. It's useful for proportional font.
   For monospaced font, the widths of all character are the same.
```

```
   Generally speaking, the width table is not needed for monospaced font. you can get
   the width from the font header.

   If you do not need the width table, undefine USE_WIDTH_TABLE.

   To get the width of character 'A', you can use the following expression:

     const USHORT index = GetIndex('A');
     const USHORT width = width_table[index];

*****************************************************************************************/
const unsigned short fontArial14h_width_tablep[] PROGMEM =
{
/*  width    char    hexcode    decimal */
/*  =====    ====    =======    ======= */
  3,       /*          0020        32   */
  2,       /*  !       0021        33   */
  4,       /*  "       0022        34   */
  6,       /*  #       0023        35   */
  6,       /*  $       0024        36   */
  10,      /*  %       0025        37   */
  7,       /*  &       0026        38   */
  2,       /*  '       0027        39   */
  4,       /*  (       0028        40   */
  4,       /*  )       0029        41   */
  4,       /*  *       002A        42   */
  6,       /*  +       002B        43   */
  3,       /*  ,       002C        44   */
  4,       /*  -       002D        45   */
  3,       /*  .       002E        46   */
  3,       /*  /       002F        47   */
  6,       /*  0       0030        48   */
  6,       /*  1       0031        49   */
  6,       /*  2       0032        50   */
  6,       /*  3       0033        51   */
  6,       /*  4       0034        52   */
  6,       /*  5       0035        53   */
  6,       /*  6       0036        54   */
  6,       /*  7       0037        55   */
  6,       /*  8       0038        56   */
  6,       /*  9       0039        57   */
  3,       /*  :       003A        58   */
  3,       /*  ;       003B        59   */
  6,       /*  <       003C        60   */
  6,       /*  =       003D        61   */
  6,       /*  >       003E        62   */
  6,       /*  ?       003F        63   */
  11,      /*  @       0040        64   */
  8,       /*  A       0041        65   */
  7,       /*  B       0042        66   */
  7,       /*  C       0043        67   */
  7,       /*  D       0044        68   */
  6,       /*  E       0045        69   */
  6,       /*  F       0046        70   */
  8,       /*  G       0047        71   */
  7,       /*  H       0048        72   */
  2,       /*  I       0049        73   */
  5,       /*  J       004A        74   */
  7,       /*  K       004B        75   */
  6,       /*  L       004C        76   */
  8,       /*  M       004D        77   */
  7,       /*  N       004E        78   */
  8,       /*  O       004F        79   */
  6,       /*  P       0050        80   */
  8,       /*  Q       0051        81   */
  7,       /*  R       0052        82   */
  7,       /*  S       0053        83   */
  6,       /*  T       0054        84   */
  7,       /*  U       0055        85   */
  8,       /*  V       0056        86   */
  11,      /*  W       0057        87   */
  7,       /*  X       0058        88   */
  8,       /*  Y       0059        89   */
  7,       /*  Z       005A        90   */
  3,       /*  [       005B        91   */
  3,       /*  \       005C        92   */
  3,       /*  ]       005D        93   */
  5,       /*  ^       005E        94   */
  6,       /*  _       005F        95   */
  4,       /*  `       0060        96   */
```

```
  6,       /*  a       0061        97   */
  6,       /*  b       0062        98   */
  6,       /*  c       0063        99   */
  6,       /*  d       0064       100   */
  6,       /*  e       0065       101   */
  4,       /*  f       0066       102   */
  6,       /*  g       0067       103   */
  6,       /*  h       0068       104   */
  2,       /*  i       0069       105   */
  2,       /*  j       006A       106   */
  5,       /*  k       006B       107   */
  2,       /*  l       006C       108   */
  8,       /*  m       006D       109   */
  6,       /*  n       006E       110   */
  6,       /*  o       006F       111   */
  6,       /*  p       0070       112   */
  6,       /*  q       0071       113   */
  4,       /*  r       0072       114   */
  6,       /*  s       0073       115   */
  3,       /*  t       0074       116   */
  6,       /*  u       0075       117   */
  6,       /*  v       0076       118   */
  10,      /*  w       0077       119   */
  6,       /*  x       0078       120   */
  6,       /*  y       0079       121   */
  6,       /*  z       007A       122   */
  4,       /*  {       007B       123   */
  2,       /*  |       007C       124   */
  4,       /*  }       007D       125   */
  6,       /*  ~       007E       126   */
  8,       /*  □       007F       127   */
  6,       /*  ¢       00A2       162   */
  6,       /*  £       00A3       163   */
  6,       /*  ¥       00A5       165   */
  6,       /*  §       00A7       167   */
  8,       /*  ©       00A9       169   */
  6,       /*  ¬       00AC       172   */
  8,       /*  ®       00AE       174   */
  4,       /*  °       00B0       176   */
  6,       /*  ±       00B1       177   */
  4,       /*  ²       00B2       178   */
  4,       /*  ³       00B3       179   */
  6,       /*  µ       00B5       181   */
  4,       /*  ¹       00B9       185   */
  5,       /*  º       00BA       186   */
  8,       /*  Ä       00C4       196   */
  8,       /*  Ö       00D6       214   */
  7,       /*  Ü       00DC       220   */
  7,       /*  ß       00DF       223   */
  6,       /*  ä       00E4       228   */
  6,       /*  ö       00F6       246   */
  6,       /*  ü       00FC       252   */
  8,       /*  ?       0394       916   */
  8,       /*  ?       039B       923   */
  8,       /*  O       03A9       937   */
  5,       /*  e       03B5       949   */
  6,       /*  ?       03B8       952   */
  6,       /*  µ       03BC       956   */
  9,       /*  p       03C0       960   */
  11,      /*  ?       263A      9786   */
  8,       /*  ?       2640      9792   */
  8,       /*  ?       2642      9794   */
  8,       /*  ?       266B      9835   */
};


#define LCDWIDTH 64
#define LCDHEIGHT 48
#define LCDSIZE (LCDWIDTH * LCDHEIGHT)  // In PIXEL !!! muss durch 32 teilbar sein fuer longs.
unsigned char lcdbuffer[LCDSIZE/8];


void i2c_oled_write_command(unsigned char i2cbaseadr, unsigned char cmdvalue)
{

  Wire.beginTransmission(i2cbaseadr);
  Wire.write(0x80); // 1000 0000 co=1 DC =0  ist commando oder parameter fuer letztes kommando
  Wire.write(cmdvalue);
  Wire.endTransmission();
}
```

```
void i2c_oled_entire_onoff(unsigned char i2cbaseadr, unsigned char onoff)
{
  if (onoff == 1) { // anschalten alle PIXEL EIN
    i2c_oled_write_command(i2cbaseadr,0xA5);
  } else { // Daten aus dem RAM
    i2c_oled_write_command(i2cbaseadr,0xA4);
  }
}


void i2c_oled_display_onoff(unsigned char i2cbaseadr, unsigned char onoff)
{
  if (onoff == 1) { // anschalten display
    i2c_oled_write_command(i2cbaseadr,0xAF);
  } else { // ausschalten display
    i2c_oled_write_command(i2cbaseadr,0xAE);
  }
}

void i2c_oled_setbrightness(unsigned char i2cbaseadr, unsigned char wert)
{
  i2c_oled_write_command(i2cbaseadr,0x81); // cmd fuer brightness
  i2c_oled_write_command(i2cbaseadr,wert); // 2. wert dann brightness
}


void i2c_oled_inverse_onoff(unsigned char i2cbaseadr, unsigned char onoff)
{
  if (onoff == 1) { // inverse
    i2c_oled_write_command(i2cbaseadr,0xA7);
  } else { // ausschalten display
    i2c_oled_write_command(i2cbaseadr,0xA6);
  }
}

// INIT 64x48 display:


void i2c_oled_initall(unsigned char i2cbaseadr)
{
  // i2c_oled_display_onff(i2cbaseadr,0); // ERST mal aus.
  i2c_oled_write_command(i2cbaseadr,0xAE);   // display off
  //
  i2c_oled_write_command(i2cbaseadr,0x00); /*set lower column address*/
 i2c_oled_write_command(i2cbaseadr,0x12); /*set higher column address*/
 i2c_oled_write_command(i2cbaseadr,0x40); /*set display start line*/
 i2c_oled_write_command(i2cbaseadr,0xB0); /*set page address*/
 i2c_oled_write_command(i2cbaseadr,0x81); /*contract control*/
 i2c_oled_write_command(i2cbaseadr,0xff); /*128*/
 i2c_oled_write_command(i2cbaseadr,0xA1); /*set segment remap*/
 i2c_oled_write_command(i2cbaseadr,0xA6); /*normal / reverse*/
 i2c_oled_write_command(i2cbaseadr,0xA8); /*multiplex ratio*/
 i2c_oled_write_command(i2cbaseadr,0x2F); /*duty = 1/48*/
 i2c_oled_write_command(i2cbaseadr,0xC8); /*Com scan direction*/
 i2c_oled_write_command(i2cbaseadr,0xD3); /*set display offset*/
 i2c_oled_write_command(i2cbaseadr,0x00);
 i2c_oled_write_command(i2cbaseadr,0xD5); /*set osc division*/
 i2c_oled_write_command(i2cbaseadr,0x80);
 i2c_oled_write_command(i2cbaseadr,0xD9); /*set pre-charge period*/
 i2c_oled_write_command(i2cbaseadr,0x21);
 i2c_oled_write_command(i2cbaseadr,0xDA); /*set COM pins*/
 i2c_oled_write_command(i2cbaseadr,0x12);
 i2c_oled_write_command(i2cbaseadr,0xdb); /*set vcomh*/
 i2c_oled_write_command(i2cbaseadr,0x40);
 i2c_oled_write_command(i2cbaseadr,0x8d); /*set charge pump enable*/
 i2c_oled_write_command(i2cbaseadr,0x14);
 i2c_oled_write_command(i2cbaseadr,0xAF);   // enable display
  //
}

void i2c_oled_initalllarge(unsigned char i2cbaseadr)
{
  // i2c_oled_display_onff(i2cbaseadr,0); // ERST mal aus.
  i2c_oled_write_command(i2cbaseadr,0xd5); // divide ratio osc freq
  i2c_oled_write_command(i2cbaseadr,0x80); // f0 flackert werniger als 80
  //
  i2c_oled_write_command(i2cbaseadr,0xa8); // multiplex ratio mode:63
```

```
  i2c_oled_write_command(i2cbaseadr,0x3f);
  //
  i2c_oled_write_command(i2cbaseadr,0xd3); // set display offset
  i2c_oled_write_command(i2cbaseadr,0); // value 0
  //
  i2c_oled_write_command(i2cbaseadr,0x40); // set display startline (D5..D0 = line)
  //
  i2c_oled_write_command(i2cbaseadr,0x8D); // charge pump on + 14 + af
  i2c_oled_write_command(i2cbaseadr,0x14); // Enable charge pump
  //
  i2c_oled_write_command(i2cbaseadr,0xA1); // segment remap hor richtung a1 nach links nach
rechts a0 rechts nach links
  //
  i2c_oled_write_command(i2cbaseadr,0xC8); // c8 von oben nach unten c0 von unten nach oben
  //
  i2c_oled_write_command(i2cbaseadr,0xda); // common pads hardware: alternative
  //
  i2c_oled_write_command(i2cbaseadr,0x12); // 12: OK, 32: dasselbe, 02: Datenmüll
  //
  i2c_oled_write_command(i2cbaseadr,0x81); // set brightness
  i2c_oled_write_command(i2cbaseadr,0xff); // 0..ff
  //
  i2c_oled_write_command(i2cbaseadr,0xD9);  // set precharge period
  i2c_oled_write_command(i2cbaseadr,0xF1);   // F1 flacher staerker, 11//22 weniger stark
  //
  i2c_oled_write_command(i2cbaseadr,0xDB);   // COM Deselect level
  i2c_oled_write_command(i2cbaseadr,0x40);   // 0.83*VCC laut datenblatt von legendary
  //
  i2c_oled_write_command(i2cbaseadr,0xA4);    // Display on alle pixel ein
  //
  i2c_oled_write_command(i2cbaseadr,0xA6);   // set normal display a6=normal a7=invertiert
  //
  i2c_oled_write_command(i2cbaseadr,0xAF);   // enable display
  //
}

// zeilen 0..7
unsigned char i2c_oled_write_top(unsigned char i2cbaseadr, int zeile, int bytes,  unsigned
char barray[],signed int sh1106padding)
{
  int i;

  i2c_oled_write_command(i2cbaseadr,0x20); // page address mode
  i2c_oled_write_command(i2cbaseadr,0x02); // page address mode
  // **
  //
  i2c_oled_write_command(i2cbaseadr,0xb0+(zeile & 7)); // B0..B7
  //
  i2c_oled_write_command(i2cbaseadr,0x00); // $00 lower nibble col + x3x2x1x0
  i2c_oled_write_command(i2cbaseadr,0x10); // $10 high  nibble col + x3x2x1x0
  //
  // DANN Daten senden Umschalten auf Daten Modus
  //

  // ACHTUNG bei Arduino anscheinend max 32 bytes data
  // Wirelib mxa 32 bytes buffer size !!
  // daher zerlegen noetig !!

  // AChtung col 1..64-seg95..seg32  row 1..48 com32..com55

  // zahl bytes / 8 teilbar !!
  int j=0;
  int k =0;

// MAX LIMIT daher zwei schleifen bei Arduino

    Wire.beginTransmission(i2cbaseadr);
    Wire.write(0x40); // 0100 0000 co=0 DC =1  ist data follow no cmd repeat
    for (i=0; i<16; i++) {
       Wire.write(0);
    }
    Wire.endTransmission();

    Wire.beginTransmission(i2cbaseadr);
    Wire.write(0x40); // 0100 0000 co=0 DC =1  ist data follow no cmd repeat
    for (i=0; i<16; i++) {
       Wire.write(0);
    }
    Wire.endTransmission();
```

```
  for (k=0; k<8; k++) {
    Wire.beginTransmission(i2cbaseadr);
    Wire.write(0x40); // 0100 0000 co=0 DC =1  ist data follow no cmd repeat
    for (i=0; i<bytes/8; i++) {
       Wire.write(barray[j++]);
    }
    Wire.endTransmission();
  }
  //
}

// ACHTUNG VERZAHNTes Display...
void disp_lcd_frombuffer() {
    // 132 fix an der Stelle !!
    // 64yx48
    // AChtung col 1..64-seg95..seg32  row 1..48 com32..com55
    // verzahnt com 0..23 nach 48..2 und com 32..55 nach row 47..1
    // offsets im transfer std 0..8
    i2c_oled_write_top(i2coledssd, 0, 64, &lcdbuffer[0], 0);
    i2c_oled_write_top(i2coledssd, 1, 64, &lcdbuffer[64], 0);
    i2c_oled_write_top(i2coledssd, 2, 64, &lcdbuffer[64*2], 0);
    i2c_oled_write_top(i2coledssd, 3, 64, &lcdbuffer[64*3], 0);
    i2c_oled_write_top(i2coledssd, 4, 64, &lcdbuffer[64*4], 0);
    i2c_oled_write_top(i2coledssd, 5, 64, &lcdbuffer[64*5], 0);

}

// Buffer loeschen mit farbe
// 0 oder ff
#define COLOR_BLACK 0
#define COLOR_WHITE 1      // sonderfall

void disp_buffer_clear(unsigned short data) {
  unsigned char  *ptr = (unsigned char*)&lcdbuffer[0];
  unsigned char datal = 0;
  if (data >0) datal = 0xff; // alle an dann beim loeschen
  int i = 0;
  for (i = 0; i<LCDSIZE/8; i++) {
    *ptr++ = datal;
  }
}

// x,y
// dabei row col organisation
// 132 x 64 pixel dabei 8 pixel in y-richtung mit 0 beginnend auf einem byte
// x horizontal y vertikal

// 64 x 48 pixel !!
//
void disp_setpixel(int x, int y, unsigned short col1) {
    // COL =0 dunke =1hell
  // 1 pixel setzen (little endian)
  // col1 =0 hell =1 dunkel
  // Tabelle 8 bytes per zeile
  // eintrag 0 ist rechte pixelgruppe
  // y=0 ist unten links aber physicalisch von oben nach unten (row 7 bit 7 -> y=0)

  unsigned char  *dest;  // zielpointer lcdbuffer
  int yoff = 0;
  int ymod = 0;
  if (x < 0) return;  // CLIP
  if (x >= LCDWIDTH) return;  // CLIP
  if (y < 0) return;
  if (y >= LCDHEIGHT) return;
 #ifdef XXXX
  // SPEZIELL x-achse 0..63 aber y-achse abh von gerade oder ungerader zeile
  if ((y & 1) == 0) {
    // Gerade zeilen
    ymod = y & 0x7; // bitposition 0..7
    yoff = y >> 3; // offset der rows zeilen 0..7 (y>0 !!) bit 0..2 weg


  } else {
    // ungerade zeilen

  }
 #endif

  // berechnen row col und byte pos.
```

```
  // y = 63 - y; // damit y=0 gedreht y=0 link oben !!
  ymod = y & 0x7; // bitposition 0..7
  yoff = y >> 3; // offset der rows zeilen 0..7 (y>0 !!) bit 0..2 weg

  dest = &lcdbuffer[yoff * LCDWIDTH + x]; // DAMIT Byte definiert
  // nun bit 0..7 moeglichs effizienz  austauschen
  if (col1 == 0) { // bit loeschen
   *dest &= ~(1<<ymod);
  } else { // bit setzen
   *dest |= (1<<ymod);
  }
  //
}

unsigned short disp_setchar(int x, int y,  unsigned char chidx1, unsigned short color) { //
breite als ergebnis fuer propschrift
  unsigned char chidx;
  chidx = chidx1;
  if (chidx <= 0x20) chidx = 0x20;
  chidx -= 0x20; // erster index
  unsigned short w = pgm_read_word_near(fontArial14h_width_tablep+chidx);
  unsigned long offset = pgm_read_word_near(fontArial14h_offset_tablep+chidx);
  int maxh = 14; // anzahl der zeilen
  int i = 0, v1 = 0, v2 = 0;
  int b1;
  //

  // breite <= 8 pixel dann ein byte >=8 dann zwei bytes (max width = 16 pixel)
  if (w <= 8) { // 1 byte max
    for (i = 0; i<14; i++) { // alle zeilen abarbeiten
      v1 = pgm_read_byte_near(fontArial14h_data_tablep+offset + i); // 1 byte pro font
durchlaufen
      // alle bits durchlaufen
      for (b1 = 0; b1<w; b1++) {
        if (v1 & (1 << (7 - b1))) { // big endian bits
          disp_setpixel(x + b1, y + i, color);
        }
      } // alle bits durchlaufen
    }
  } else { // 2 byte zeichen
    for (i = 0; i<14; i++) { // alle zeilen abarbeiten
      v1 = pgm_read_byte_near(fontArial14h_data_tablep+offset + i * 2); // 1 byte pro font
durchlaufen
      v2 = pgm_read_byte_near(fontArial14h_data_tablep+offset + i * 2 + 1); // 1 byte pro font
durchlaufen
      // alle bits durchlaufen
      for (b1 = 0; b1<8; b1++) { // erste haelfte first
        if (v1 & (1 << (7 - b1))) { // big endian bits
          disp_setpixel(x + b1, y + i, color);
        }
      } // alle bits durchlaufen
      for (b1 = 0; b1<w; b1++) { // restl bits
        if (v2 & (1 << (7 - b1))) { // big endian bits
          disp_setpixel(x + 8 + b1, y + i, color);
        }
      } // alle bits durchlaufen
    }
  }
  return(w);
}


int disp_print_xy_lcd(int x, int y, unsigned char *text, unsigned short color, int chset) { //
0=14 1=27
  int x1;
  x1 = x;
  if (text == 0) return(x);

    while (*text != 0) {
      x1 = x1 + disp_setchar(x1, y,  *text++, color);
    }

  return(x1);
}

// Linien Zeichnen

void  disp_line_lcd(int x0, int y0, int x1, int y1, unsigned short col) {
  // disp_setpixel(x+b1,y+i,color);
```

```
  int dx, dy, sx, sy, err, e2;
  dx = (x1 - x0);
  if (dx < 0) dx = -dx;
  dy = (y1 - y0);
  if (dy < 0) dy = -dy;
  if (x0 < x1) {
    sx = 1;
  } else {
    sx = -1;
  }
  if (y0 < y1) {
    sy = 1;
  } else {
    sy = -1;
  }
  //
  err = dx - dy;
  do {
    disp_setpixel(x0, y0, col);
    if ((x0 == x1) && (y0 == y1)) return;
    e2 = 2 * err;
    if (e2 > -dy) {
      err = err - dy;
      x0 = x0 + sx;
    }
    if (e2 < dx) {
      err = err + dx;
      y0 = y0 + sy;
    }
  } while (1==1);
}

// Rechteck Zeichnen



void disp_rect_lcd(int x1, int y1, int x2, int y2, unsigned short col) {
  // disp_setpixel(x+b1,y+i,color);
  disp_line_lcd(x1, y1, x2, y1, col);
  disp_line_lcd(x1, y2, x2, y2, col);
  disp_line_lcd(x1, y1, x1, y2, col);
  disp_line_lcd(x2, y1, x2, y2, col);
}

// filled rechteck zeichnen optionaler rahmen
//
void disp_filledrect_lcd(int x1, int y1, int x2, int y2, unsigned short col) {
  // disp_setpixel(x+b1,y+i,color);
  // schnelles fill
  //
  // x1,y1 und x2,y2 swappen ggf.
  unsigned short  *dest;    // zielpointer lcdbuffer mit 8 bytes per row !! (little endian
codiert)

  int x, y;
  if (x1 > x2) {
    x = x1;
    x1 = x2;
    x2 = x;
  }
  if (y1 > y2) {
    y = y1;
    y1 = y2;
    y2 = y;
  }
  if (x1 < 0) x1 = 0;  // CLIP
  if (x1 >= LCDWIDTH) x1 = LCDWIDTH - 1;  // CLIP
  if (y1< 0) y1 = 0;
  if (y1 >= LCDHEIGHT) y1 = LCDHEIGHT - 1;  // CLIP;
  // ACHTUNG rechnen row
  if (x2 < 0) x2 = 0;  // CLIP
  if (x2 >= LCDWIDTH) x2 = LCDWIDTH - 1;  // CLIP
  if (y2< 0) y2 = 0;
  if (y2 >= LCDHEIGHT) y2 = LCDHEIGHT - 1;  // CLIP;

  // noch nicht ganz effizient fuer sonderfaelle
  // koennte man in bytes zusammenfassen wenn eine ganze row betroffen ist
  // also startzeile, n*zeilen 8bit endzeilen z.b.
  // bzw drei masken berechnen dafuer...
  //
```

```
  for (y = y1; y<= y2; y++) {
    for (x = 0; x<=(x2 - x1); x++) {
      disp_setpixel(x, y, col);
    }
  }

}

// -------------------------END OLED ------------------------------------------------

static uint8_t device_code_emc1 = 0x18;
char emcbuffer[100];

String hex (uint64_t v)
{
  String hexStr = "";
  if (v == 0)
  {
    hexStr = "0";
  }
  else
  {
    while (v > 0)
    {
      byte digit = v & 0xF;
      v = v / 16;
      if (digit < 10)
      {
        hexStr = chr(digit + 48) + hexStr;
      }
      else
      {
        hexStr = chr(digit + 55) + hexStr;
      }
    }
  }
  return hexStr;
}

String hexbyte (byte v)
{
  return right("0" + hex(v), 2);
}

String hexword (uint16_t v)
{
  return right("000" + hex(v), 4);
}

String hexlong (uint32_t v)
{
  return right("0000000" + hex(v), 8);
}

String hexint64 (uint64_t v)
{
  return right("00000000000000" + hex(v), 16);
}

String left (String s, int length)
{
  if (length < 1)
  {
    return "";
  }
  else if (length >= s.length())
  {
    return s;
  }
  else
  {
    return s.substring(0, length);
  }
}

String right (String s, int length)
{
  int L = s.length();
  if (L <= length)
  {
```

```
    return s;
  }
  else if (length < 1)
  {
    return "";
  }
  else
  {
    return s.substring(L - length, L);
  }
}

String repeatasc (int a, int n)
{
    String r = "";
    String zchn = chr(a);
    if (n > 0)
    {
        for (int i = 1; i <= n; i++)
        {
            r = r + zchn;
        }
    }
    return (r);
}

double frac (double n)
{
  return n - trunc(n);
}

String chr (int c)
{
  String s = "";
  s += char(c);
  return s;
}

int64_t exponent10 (byte n)
{
  if ((n < 0) || (n > 18))
  {
    return 0;
  }
  else
  {
    long result = 1;
    for (byte i = 1; i <= n; i++)
    {
      result *= 10;
    }
    return result;
  }
}

String indchar (String s, int i)
{
  if ((i >= 1) && (i <= s.length()))
  {
    return chr(s[i - 1]);
  }
  else
  {
    return ("");
  }
}

String strform (uint64_t n, int asc, byte MinVorkomma, bool TausenderPunkt)
{
  String vorzeichen = "";
  if (n < 0)
  {
    vorzeichen = "-";
    //n = abs(n);
  }
  String vorkomma = "";
  if (n == 0)
  {
    vorkomma = "0";
  }
```

```
  else
  {
    while (n !=  0)
    {
      vorkomma = chr(abs(n % 10) + 48) + vorkomma;
      n = n / 10;
    }
  }
  if (TausenderPunkt)
  {
      String s = "";
      byte l = vorkomma.length();
      for (int i = 0; i < l; i++)
      {
          if ((((l - i) % 3) == 0) && (s != ""))
          {
              s = s + chr(46); // "."
          }
          s = s + indchar(vorkomma, i + 1);
      }
      vorkomma = s;
  }
  vorkomma = vorzeichen + vorkomma;
  byte VL = vorkomma.length();
  if (MinVorkomma > VL)
  {
      vorkomma = repeatasc(asc, MinVorkomma - VL) + vorkomma;
  }
  return (vorkomma);
}

String str (uint64_t n)
{
  return strform(n, 32, 1, true);
}

String strrealform (double n, byte asc, byte MinVorkomma, byte MaxNachkomma, bool
TausenderPunkt, bool CutZero)
{
  String vorzeichen = "";
  if (n < 0) // Negativ
  {
    vorzeichen = "-";
    n = abs(n);
  }
  if (MaxNachkomma > 9) // float ist auf 7 Stellen genau, Double auf 9 Stellen
  {
    MaxNachkomma = 9;
  }
  n = n + (1.0 / exponent10(MaxNachkomma) / 2.0); // Runden
  String result = "";
  long mantissa = n;
  String vorkomma = String(mantissa);
  if (TausenderPunkt)
  {
    String s = "";
    byte l = vorkomma.length();
    for (int i = 0; i < l; i++)
    {
      if ((((l - i) % 3) == 0) && (s != ""))
      {
        s = s + chr(46); // "."
      }
      s = s + indchar(vorkomma, i + 1);
    }
    vorkomma = s;
  }
  vorkomma = vorzeichen + vorkomma;
  byte VL = vorkomma.length();
  if (MinVorkomma > VL)
  {
    vorkomma = repeatasc(asc, MinVorkomma - VL) + vorkomma;
  }
  double nachkomma = frac(n);
  if ((nachkomma != 0) || not(CutZero))
  {
    vorkomma = vorkomma + ",";
    for (byte i = 1; i <= MaxNachkomma; ++i)
    {
      nachkomma = nachkomma * 10;
```

```
      vorkomma = vorkomma + String((long)nachkomma);
      nachkomma = nachkomma - trunc(nachkomma);
    }
    if (CutZero)
    {
      while (right(vorkomma, 1) == "0")
      {
        vorkomma = left(vorkomma, vorkomma.length() - 1);
      }
    }
    if (right(vorkomma, 1) == ",")
    {
      vorkomma = left(vorkomma, vorkomma.length() - 1);
    }
  }
  return vorkomma;
}

uint8_t read_EMC8 (uint8_t registerno)
{
    uint16_t value = 0;

    Wire.beginTransmission(device_code_emc1);
    Wire.write(byte(registerno));
    Wire.endTransmission();
    Wire.requestFrom(device_code_emc1, 1, true);
    if (1 <= Wire.available()) { /* if one byte were received*/
      value = Wire.read();
    }

    //i2c_read(drv_i2cm_1,device_code_emc1,registerno,0,&value);

    return (uint8_t) value;
}


uint16_t read_EMC16 (uint8_t registerno, bool messages)
{
    uint16_t value = 0;

    Wire.beginTransmission(device_code_emc1);
    Wire.write(registerno);
    Wire.endTransmission();

    Wire.requestFrom(device_code_emc1, 2, true);
    if (2 <= Wire.available())
    { /* if two bytes were received*/
      value = Wire.read();  /* receive high byte (overwrites previous reading)*/
      value = value << 8;    /* shift high byte to be high 8 bits*/
      value |= Wire.read(); /* receive low byte as lower 8 bits*/
      if (messages)
      {
        Serial.print("EMC#");
        Serial.print(hexbyte(registerno));
        Serial.print(": ");
        Serial.print(hexword(value));
        Serial.print(chr(9));
      }
    }
    /*i2c_read(drv_i2cm_1,device_code_emc1,registerno,1,&value); */

    return value;
}

uint8_t write_EMC8(uint8_t registerno, uint8_t data)
{
    uint16_t retv;
    Wire.beginTransmission(device_code_emc1);
    Wire.write(byte(registerno));
    retv = Wire.write(byte(data));
    Wire.endTransmission();
    /*retv = i2c_write(drv_i2cm_1,device_code_emc1,registerno,data);*/

    return (uint8_t) retv;
}

float EMC_Ampere = 0;
float EMC_Volt = 0;
float EMC_Watt = 0;
float EMC_Temperature = 0;
```

```
uint8_t enable_emc1701 (bool messages)
{
    uint32_t value1 = 0;
    uint32_t value2 = 0;
    uint32_t value3 = 0;
    uint32_t value4 = 0;
    value1 = read_EMC8(0xfd);
    value2 = read_EMC8(0xfe);
    write_EMC8(0x51,0); //range 10mV
    if (messages)
    {
      sprintf(emcbuffer,"EMC-A: id=%02x  %02x",value1,value2);
    }
    if ((value1 == 0x38) && (value2 == 0x5D)) /*Sensor (id) found*/
    {
        write_EMC8(0x51,0); // range = 10mV
        value1 = read_EMC16(0x54, messages);  // Sensor voltage
        value2 = read_EMC16(0x58, messages);  // SOurce voltage
        value4 = read_EMC16(0x38, messages);  // Diode Temp voltage 38,39
        if ((value4 & 0x8000) != 0)
        {
          value4|= 0xffff0000;   // Temp signed
        }
        value4 = value4 / 32; // not used max +-511
        value4 = value4 * 1279; //  127.875;
        value4 = value4 / 1023; //   umgerechnet mit 1 NStelle
        if (( value1 & 0x8000) != 0)
        {
          if (messages)
          {
            Serial.print("(Negative Value)");
          }
          value1 |= 0xffff0000;   // Sensor kann < 0 sein !!
        }
        value1 = value1 *  1000; // 10mV auf 0.01 OHM = 1A
        value1 = value1 / 32756;

        if (value1 > 100000) // Overflow, es fließt gar kein Strom
        {
          value1 = 0;
        }

        value2 = value2 >> 5; // Bits 0..4=0
        value2 = value2 * 23988;  //Full Scale Voltage
        value2 = value2 / 2047;
        value3 = (value1 * value2) / 1000;

        EMC_Ampere = value1;
        EMC_Volt = value2;
        EMC_Watt = value3;
        EMC_Temperature = value4;

        if (messages)
        {
          Serial.println("");
          sprintf(emcbuffer,"EMC-A: %ld mA  %ld mV  %ld mW  %ld.%0ld C ", value1, value2,
value3, value4 / 10l, value4 % 10l);
        }
        return 1;
    }
    return 0;
}

long    powerUpTime;
String  s;

long    Laufzeit    = 20;       // Laufzeit in Sekunden bis zum PowerOff

int     IL_Start    = 0;        // Startwert                    (zwischen 0 und IL_Ende)
int     IL_Ende     = 255;      // Endwert                      (zwischen IL_Start und 255)
int     IL_Step     = 10;       // Wertzuwachs bei einem Schritt  (zwischen 1 und 255)
int     IL_Current  = IL_Start; // Aktueller Wert von A0
int     IL_Stable   = 0;        // Stabiler Wert von A0         (wird ermitttelt)

long    IL_Steptime = 1000;     // Zeit in Milli-Sekunden bis zum nächsten Schritt
long    IL_Time     = 0;        // Zeitpunkt des letzten Schritts in Milli-Sekunden

void setup()
{
```

```
  Serial.begin(115200);
  Serial.println("Initializing...");
  Wire.begin(); // I2C Init
  i2c_oled_initall(i2coledssd); /* OLED Init */
  byte ledBrightness = 0x1F; /*Options: 0=Off to 255=50mA*/
  byte sampleAverage = 8; /*Options: 1, 2, 4, 8, 16, 32*/
  byte ledMode = 3; /*Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green*/
  int sampleRate = 1600; /*Options: 50, 100, 200, 400, 800, 1000, 1600, 3200*/
  int pulseWidth = 411; /*Options: 69, 118, 215, 411*/
  int adcRange = 4096; /*Options: 2048, 4096, 8192, 16384*/
  disp_buffer_clear(COLOR_BLACK);
  disp_lcd_frombuffer();

  disp_buffer_clear(COLOR_BLACK);
  String s = "Starting...";
  disp_print_xy_lcd(10, 18, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
  disp_lcd_frombuffer();
  delay(5000); // 5 Sekunden auf Terminal-Verbindung warten
  powerUpTime = millis();
  IL_Time = powerUpTime;
}

void loop()
{
  int    NtcRaw = analogRead(A1);                              // Wert vom D/A Wandler A1 für
den NTC
  double VTeiler = NtcRaw * 3.3 / 1023.0;                      // Maximalspannung am Analog-Pin
  double VBatt = 3.3;                                          // Versorgungsspannung
  double RTeiler = 100000.0;                                   // 100 k Teiler
  double RNtc = (VTeiler * RTeiler) / (VBatt - VTeiler);       // Widerstand
  double B = 3800.0;                                           // Je nach NTC unterschiedlich,
muss ermittelt werden
  double RN = 100000.0;                                        // 100 kOhm
  double TN = 298.15;                                          // 25 Grad C in Kelvin
  double T = (B * TN) / (B + log(RNtc / RN) * TN) - 273.15;    // T in Cesius

  int    VoRaw = 0;
  double Vo = 0.0;

  // Variable für die verwendete Versuchsanordnung. Diese muss auf die entsprechende
Kapitelnummer gesetzt werden

//int versuch = 11;                                            // 11, 12 = Versuchsaufbau mit
Brennstoffzelle, MKR-Brick, Display und Verbraucher
//int versuch = 13;                                            // 13 = Versuchsaufbau mit
Ausschaltung der Brennstoffzellen-Stromerzeugung
//int versuch = 14;                                            // 14 = Versuchsaufbau mit
gesteuerter Strom-Last
//int versuch = 15;                                            // 15 = Versuchsaufbau mit
gesteuerter Strom-Last von 10 bis 20 mA
//int versuch = 16;                                            // 15 = Versuchsaufbau mit
gesteuerter Spannung von 3000 bis 3200 mV
  int versuch = 17;                                            // 17 = Versuchsaufbau mit
manuell gesteuerter Strom-Last

  switch (versuch)
  {
    case 11:
      VoRaw = analogRead(A0);                                  // Wert vom D/A Wandler A0 für
den 1/10 Volt-Wert
      Vo = VoRaw * 3.3 / 1023.0 * 9.6;                         // Maximalspannung am Analog-Pin
incl. Korrekturfaktor (9.6 statt 10)
      enable_emc1701(true);
      Serial.println(emcbuffer);
      disp_buffer_clear(COLOR_BLACK);                          // Clear Display
      if (EMC_Volt < 0.1)                                      // Ausgangs-Spannung bricht
zusammen
      {
        EMC_Ampere = 0.0;                                      // Fehlanzeigen vermeiden
        EMC_Watt = 0.0;
      }
      // Ausgaben für das Display vorbereiten
      s = "t:" + left(strrealform(EMC_Temperature / 10.00f, 32, 1, 1, false, false), 4) +
"*Vo:";
      s += left(strrealform(Vo, 32, 1, 1, false, false), 3) + "  ";
      disp_print_xy_lcd(0, 0, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      s = "T:" + left(strrealform(T, 32, 1, 1, false, false), 4) + "*";
      s += "Vi:" + left(strrealform(EMC_Volt / 1000.000f, 32, 1, 1, false, false), 4) + "  ";
      disp_print_xy_lcd(0, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
```

```
      s = left(strrealform(EMC_Ampere, 32, 1, 0, false, true), 4) + "  ";
      disp_print_xy_lcd(0, 24, (unsigned char*)"mA:", COLOR_WHITE, 0);
      disp_print_xy_lcd(30, 24, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      s = left(strrealform(EMC_Watt, 32, 1, 0, false, true), 4) + "  ";
      disp_print_xy_lcd(0, 36, (unsigned char*)"mW:", COLOR_WHITE, 0);
      disp_print_xy_lcd(30, 36, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      disp_lcd_frombuffer(); /*Send to display*/

      break;

    case 13:
      if (((millis() - powerUpTime) / 1000) >= Laufzeit)      // Nach Erreichen der Laufzeit
ausschalten
      {
        analogWrite(A0,255);                                  // Strompfad-Weiche Brick
ausschalten
        disp_buffer_clear(COLOR_BLACK);
        s = "Power Off";
        disp_print_xy_lcd(5, 18, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
        disp_lcd_frombuffer();
      }
      else
      {
        analogWrite(A0,0);                                    // Strompfad-Weiche Brick
einschalten
        enable_emc1701(true);
        Serial.println(emcbuffer);
        disp_buffer_clear(COLOR_BLACK);                       // Clear Display
        if (EMC_Volt < 0.1) // Ausgangs-Spannung bricht zusammen
        {
          EMC_Ampere = 0.0; // Fehlanzeigen vermeiden
          EMC_Watt = 0.0;
        }
        // Ausgaben für das Display vorbereiten
        s = "t:" + left(strrealform(EMC_Temperature / 10.00f, 32, 1, 1, false, false), 4) +
"*P:";
        s += str(Laufzeit - ((millis() - powerUpTime) / 1000)) + "  ";
        disp_print_xy_lcd(0, 0, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
        s = "T:" + left(strrealform(T, 32, 1, 1, false, false), 4) + "  ";
        disp_print_xy_lcd(0, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
        s = "Vi:" + left(strrealform(EMC_Volt / 1000.000f, 32, 1, 1, false, false), 4) + "  ";
        disp_print_xy_lcd(32, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

        s = left(strrealform(EMC_Ampere, 32, 1, 0, false, true), 4) + "  ";
        disp_print_xy_lcd(0, 24, (unsigned char*)"mA:", COLOR_WHITE, 0);
        disp_print_xy_lcd(30, 24, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

        s = left(strrealform(EMC_Watt, 32, 1, 0, false, true), 4) + "  ";
        disp_print_xy_lcd(0, 36, (unsigned char*)"mW:", COLOR_WHITE, 0);
        disp_print_xy_lcd(30, 36, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

        disp_lcd_frombuffer(); /*Send to display*/

      } // "Erreichen der Laufzeit"
      break;

    case 14: // A0-Step-Variante
      analogWrite(A0,IL_Current);                            // AnalogWrite hat auf dem A0-Pin
8 Bit echte Analog-Auflösung
      enable_emc1701(true);
      Serial.println(emcbuffer);
      disp_buffer_clear(COLOR_BLACK);                        // Clear Display
      if (EMC_Volt < 0.1) // Ausgangs-Spannung bricht zusammen
      {
        EMC_Ampere = 0.0; // Fehlanzeigen vermeiden
        EMC_Watt = 0.0;
      }

      Serial.print("Time: ");
      Serial.print(str(millis() - powerUpTime));
      Serial.print(" ms.");
      Serial.print("  NTC Temp.: ");
      Serial.print(strrealform(T, 32, 1, 1, false, false));
      Serial.print("  EMC_Volt: ");
      Serial.print(EMC_Volt);
      Serial.println("");

      Serial.print("IL_Time Dif: ");
```

```
      Serial.print(str(millis() - IL_Time));
      Serial.print(" ms.  ");
      Serial.print("IL_Current: ");
      Serial.print(str(IL_Current));
      Serial.print(" ");
      Serial.println("");

      Serial.println("");

      /*Convert number to Letters*/
      s = "T:" + left(strrealform(EMC_Temperature / 10.00f, 32, 1, 1, false, false), 4) +
"*A:";
      s += String(IL_Current) + "  ";
      disp_print_xy_lcd(0, 0, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      s = "t:" + left(strrealform(T, 32, 1, 1, false, false), 4) + "*";
      disp_print_xy_lcd(0, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      s = "Vo:" + left(strrealform(EMC_Volt / 1000.000f, 32, 1, 1, false, false), 4) + "  ";
      disp_print_xy_lcd(32, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      s = left(strrealform(EMC_Ampere, 32, 1, 0, false, true), 4) + "  ";
      disp_print_xy_lcd(0, 24, (unsigned char*)"mA:", COLOR_WHITE, 0);
      disp_print_xy_lcd(30, 24, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      s = left(strrealform(EMC_Watt, 32, 1, 0, false, true), 4) + "  ";
      disp_print_xy_lcd(0, 36, (unsigned char*)"mW:", COLOR_WHITE, 0);
      disp_print_xy_lcd(30, 36, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      disp_lcd_frombuffer(); /*Send to display*/

      if ((millis() - IL_Time) > IL_Steptime)
      {
        IL_Current += IL_Step;
        if (IL_Current > IL_Ende)
        {
          IL_Current = IL_Start;
        }
        IL_Time = millis();
      }
      break;

    case 15: // A0-Step-Variante mit Suche nach 10 mA bis maximal 20 mA
      if (IL_Current < 150)
      {
        IL_Current = 150;
        IL_Step = 1;
      }

      analogWrite(A0,IL_Current);                           // AnalogWrite hat auf dem A0-Pin
8 Bit echte Analog-Auflösung
      enable_emc1701(false);
      disp_buffer_clear(COLOR_BLACK);                       // Clear Display
      if (EMC_Volt < 0.1) // Ausgangs-Spannung bricht zusammen
      {
        EMC_Ampere = 0.0; // Fehlanzeigen vermeiden
        EMC_Watt = 0.0;
      }

      /*Convert number to Letters*/
      s = "T:" + left(strrealform(EMC_Temperature / 10.00f, 32, 1, 1, false, false), 4) +
"*A:";
      s += String(IL_Current) + "  ";
      disp_print_xy_lcd(0, 0, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      s = "t:" + left(strrealform(T, 32, 1, 1, false, false), 4) + "*";
      disp_print_xy_lcd(0, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      s = "Vo:" + left(strrealform(EMC_Volt / 1000.000f, 32, 1, 1, false, false), 4) + "  ";
      disp_print_xy_lcd(32, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      s = left(strrealform(EMC_Ampere, 32, 1, 0, false, true), 4) + "  ";
      disp_print_xy_lcd(0, 24, (unsigned char*)"mA:", COLOR_WHITE, 0);
      disp_print_xy_lcd(30, 24, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      s = left(strrealform(EMC_Watt, 32, 1, 0, false, true), 4) + "  ";
      disp_print_xy_lcd(0, 36, (unsigned char*)"mW:", COLOR_WHITE, 0);
      disp_print_xy_lcd(30, 36, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      disp_lcd_frombuffer(); /*Send to display*/

      if ((millis() - IL_Time) > IL_Steptime)
      {
        enable_emc1701(true);
```

```
        Serial.println(emcbuffer);                          // Messages ausgeben
        Serial.print("Time: ");
        Serial.print(str(millis() - powerUpTime));
        Serial.print(" ms.");
        Serial.print("  NTC Temp.: ");
        Serial.print(strrealform(T, 32, 1, 1, false, false));
        Serial.print("  EMC_Volt: ");
        Serial.print(EMC_Volt);
        Serial.println("");

        Serial.print("IL_Time Dif: ");
        Serial.print(str(millis() - IL_Time));
        Serial.print(" ms.  ");
        Serial.print("IL_Current: ");
        Serial.print(str(IL_Current));
        Serial.print("  ");
        Serial.print("IL_Stable: ");
        Serial.print(str(IL_Stable));
        Serial.print("  ");

        if (EMC_Ampere < 10)
        {
          Serial.print("A<10  ");
          IL_Current += 1;
        }
        else if ((EMC_Ampere > 14) & (IL_Stable > 0))
        {
          Serial.print("mA>14  ");
          IL_Current = 150;
          IL_Step = 1;
        }
        else if (EMC_Ampere > 12)
        {
          Serial.print("mA>12  ");
          IL_Current -= 10;
        }
        else
        {
          IL_Stable = IL_Current;
        }
        if (IL_Current > 180)
        {
          IL_Current = 150;
          IL_Step = 1;
        }
        IL_Time = millis();
        Serial.println("");
        Serial.println("");
      }
      break;

    case 16: // A0-Step-Variante mit Suche nach 3000 mV bis maximal 3200 mV
      if (IL_Current < 150)
      {
        IL_Current = 150;
        IL_Stable = 0;
        IL_Step = 1;
      }

      analogWrite(A0,IL_Current);                           // AnalogWrite hat auf dem A0-Pin
8 Bit echte Analog-Auflösung
      enable_emc1701(false);
      // Serial.println(emcbuffer);                         // Messages ausgeben
      disp_buffer_clear(COLOR_BLACK);                       // Clear Display
      if (EMC_Volt < 0.1) // Ausgangs-Spannung bricht zusammen
      {
        EMC_Ampere = 0.0; // Fehlanzeigen vermeiden
        EMC_Watt = 0.0;
      }


      /*Convert number to Letters*/
      s = "T:" + left(strrealform(EMC_Temperature / 10.00f, 32, 1, 1, false, false), 4) +
"*A:";
      s += String(IL_Current) + "  ";
      disp_print_xy_lcd(0, 0, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      s = "t:" + left(strrealform(T, 32, 1, 1, false, false), 4) + "*";
      disp_print_xy_lcd(0, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      s = "Vo:" + left(strrealform(EMC_Volt / 1000.000f, 32, 1, 1, false, false), 4) + "  ";
      disp_print_xy_lcd(32, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
```

```
      s = left(strrealform(EMC_Ampere, 32, 1, 0, false, true), 4) + "  ";
      disp_print_xy_lcd(0, 24, (unsigned char*)"mA:", COLOR_WHITE, 0);
      disp_print_xy_lcd(30, 24, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      s = left(strrealform(EMC_Watt, 32, 1, 0, false, true), 4) + "  ";
      disp_print_xy_lcd(0, 36, (unsigned char*)"mW:", COLOR_WHITE, 0);
      disp_print_xy_lcd(30, 36, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      disp_lcd_frombuffer(); /*Send to display*/

      if ((millis() - IL_Time) > IL_Steptime)
      {
        enable_emc1701(true);
        Serial.println(emcbuffer);                          // Messages ausgeben
        Serial.print("Time: ");
        Serial.print(str(millis() - powerUpTime));
        Serial.print(" ms.");
        Serial.print("  NTC Temp.: ");
        Serial.print(strrealform(T, 32, 1, 1, false, false));
        Serial.print("  EMC_Volt: ");
        Serial.print(EMC_Volt);
        Serial.println("");

        Serial.print("IL_Time Dif: ");
        Serial.print(str(millis() - IL_Time));
        Serial.print(" ms.  ");
        Serial.print("IL_Current: ");
        Serial.print(str(IL_Current));
        Serial.print("  ");
        Serial.print("IL_Stable: ");
        Serial.print(str(IL_Stable));
        Serial.print("  ");

        if (EMC_Ampere < 1)
        {
          IL_Current += 1;
          Serial.print("A<1  ");
        }
        else if (EMC_Volt < 3000)
        {
          IL_Current += 1;
          Serial.print("V<3.0  ");
        }
        else if ((EMC_Volt > 3400) & (IL_Stable > 0))
        {
          Serial.print("V>3.4  ");
          IL_Current -= 10;
        }
        else if (EMC_Volt > 3200)
        {
          Serial.print("V>3.2  ");
          IL_Current += 1;
        }
        else
        {
          IL_Stable = IL_Current;
        }

        if (IL_Current > 180)
        {
          IL_Current = 150;
          IL_Step = 1;
        }
        IL_Time = millis();
        Serial.println("");
        Serial.println("");
      }
      break;

      case 17: // A0-Step-Variante mit manuellem Doppel-Taster
      if (IL_Current < 150)
      {
        IL_Current = 150;
        IL_Step = 1;
      }
      analogWrite(A0,IL_Current);                            // AnalogWrite hat auf dem A0-Pin
8 Bit echte Analog-Auflösung
      enable_emc1701(true);
      Serial.println(emcbuffer);
```

```
      disp_buffer_clear(COLOR_BLACK);                        // Clear Display
      if (EMC_Volt < 0.1) // Ausgangs-Spannung bricht zusammen
      {
        EMC_Ampere = 0.0; // Fehlanzeigen vermeiden
        EMC_Watt = 0.0;
      }

      Serial.print("Time: ");
      Serial.print(str(millis() - powerUpTime));
      Serial.print(" ms.");
      Serial.print("  NTC Temp.: ");
      Serial.print(strrealform(T, 32, 1, 1, false, false));
      Serial.print("  EMC_Volt: ");
      Serial.print(EMC_Volt);
      Serial.println("");

      Serial.print("IL_Time Dif: ");
      Serial.print(str(millis() - IL_Time));
      Serial.print(" ms.  ");
      Serial.print("IL_Current: ");
      Serial.print(str(IL_Current));
      Serial.print(" ");
      Serial.println("");

      Serial.println("");

      /*Convert number to Letters*/
      s = "T:" + left(strrealform(EMC_Temperature / 10.00f, 32, 1, 1, false, false), 4) +
"*A:";
      s += String(IL_Current) + "  ";
      disp_print_xy_lcd(0, 0, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      s = "t:" + left(strrealform(T, 32, 1, 1, false, false), 4) + "*";
      disp_print_xy_lcd(0, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      s = "Vo:" + left(strrealform(EMC_Volt / 1000.000f, 32, 1, 1, false, false), 4) + "  ";
      disp_print_xy_lcd(32, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      s = left(strrealform(EMC_Ampere, 32, 1, 0, false, true), 4) + "  ";
      disp_print_xy_lcd(0, 24, (unsigned char*)"mA:", COLOR_WHITE, 0);
      disp_print_xy_lcd(30, 24, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      s = left(strrealform(EMC_Watt, 32, 1, 0, false, true), 4) + "  ";
      disp_print_xy_lcd(0, 36, (unsigned char*)"mW:", COLOR_WHITE, 0);
      disp_print_xy_lcd(30, 36, (unsigned char*)s.c_str(), COLOR_WHITE, 0);

      disp_lcd_frombuffer(); /*Send to display*/

      if (digitalRead(9) == 0)
      {
        IL_Current += IL_Step;
        while (digitalRead(9) == 0)
        {
          delay(100);
        }
      }

      if (digitalRead(8) == 0)
      {
        IL_Current -= IL_Step;
        while (digitalRead(8) == 0)
        {
          delay(100);
        }
      }

      if (IL_Current > 180)
      {
        IL_Current = 150;
        IL_Step = 1;
      }
      break;

    default: // "Bitte im Sketch den Versuch wählen!"
      /*Clear Display*/
      disp_buffer_clear(COLOR_BLACK);
      s = "Bitte im";
      disp_print_xy_lcd(0, 0, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      s = "Sketch den";
      disp_print_xy_lcd(0, 12, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      s = "Versuch";
      disp_print_xy_lcd(0, 24, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
```

```
      s = "waehlen!";
      disp_print_xy_lcd(0, 36, (unsigned char*)s.c_str(), COLOR_WHITE, 0);
      disp_lcd_frombuffer(); /*Send to display*/
      break;

   }
}
```

6. Wasserstoff

Wasserstoff ist ein chemisches Element mit dem Symbol H (für lateinisch hydrogenium) und der Ordnungszahl 1. Wasserstoff ist das häufigste chemische Element im Universum, jedoch nicht in der Erdrinde. Er ist Bestandteil des Wassers und beinahe aller organischen Verbindungen. Somit kommt gebundener Wasserstoff in sämtlichen lebenden Organismen vor.

Wasserstoff ist das chemische Element mit der geringsten Atommasse. Sein häufigstes Isotop, das auch als Protium bezeichnet wird, enthält kein Neutron, sondern besteht aus nur einem Proton und einem Elektron. Unter Bedingungen, die normalerweise auf der Erde herrschen, kommt nicht dieser atomare Wasserstoff H vor, sondern der molekulare Wasserstoff H_2, ein farb- und geruchloses Gas.

Entdeckt wurde Wasserstoff vom englischen Chemiker und Physiker Henry Cavendish im Jahre 1766, als er mit Metallen und Säuren experimentierte. Cavendish nannte das dabei entstandene Gas wegen seiner Brennbarkeit „inflammable air“ (brennbare Luft). Er untersuchte das Gas eingehend und veröffentlichte seine Erkenntnisse darüber noch im selben Jahr. Auf ähnliche Weise (Einwirkung von Säuren auf Metalle) erzeugten allerdings schon im 17. Jahrhundert Théodore Turquet de Mayerne (um 1620) und Robert Boyle (um 1670) Knallgas.

Eine genauere Analyse geschah durch Antoine Laurent de Lavoisier, der den Wasserstoff als „Wasser erzeugenden Stoff“ oder „Hydrogen“ bezeichnete und ihm damit seinen heutigen Namen gab. Cavendish hatte inzwischen eine Beobachtung von Joseph Priestley aufgreifend erkannt, dass bei der Verbrennung von Wasserstoff Wasser entsteht (veröffentlicht erst 1784). Lavoisier erfuhr von den Experimenten von Cavendish beim Besuch von dessen Assistenten Charles Blagden 1783.

Lavoisier zeigte in aufsehenerregenden Experimenten, dass es ein eigenständiges Element war und Bestandteil des Wassers, das man damals vielfach selbst noch für elementar gehalten hatte gemäß der alten Vier-Elemente-Lehre. Lavoisier führte seine Experimente quantitativ aus unter Verwendung der von ihm postulierten Massenerhaltung. Er leitete Wasserdampf in einer abgeschlossenen Apparatur über glühende Eisenspäne und ließ ihn an anderer Stelle kondensieren. Dabei stellte er fest, dass die Masse des kondensierten Wassers etwas geringer war als die der ursprünglichen Menge. Dafür entstand ein Gas, dessen Masse zusammen mit dem Gewichtszuwachs des oxidierten Eisens genau der „verloren gegangenen“ Wassermenge entsprach. Sein eigentliches Experiment war also erfolgreich.

Lavoisier untersuchte das entstandene Gas weiter und führte die heute als Knallgasprobe bekannte Untersuchung durch, wobei das Gas verbrannte. Er nannte es daher zunächst wie auch Cavendish als brennbare Luft (im Französischen in umgekehrter Wortstellung „air inflammable“). Als er in weiteren Experimenten zeigte, dass sich aus dem Gas Wasser erzeugen lässt, taufte er es hydro-gène (griechisch: hydro = Wasser; genes = erzeugend). Das Wort bedeutet demnach: „Wassererzeuger“.

Gefahrenhinweis: Wasserstoff ist ein brennbares Gas und bildet mit der Luft explosive Mischungen. Die Versuche und auch die Erzeugung von Wasserstoff dürfen daher nur in gut belüfteten Räumen durchgeführt werden.

6.1 Erzeugung von Wasserstoff mit der HydroFill Pro Station

Der Wasserstoff für unsere Brennstoffzelle wird mit der optional erhältlichen HydroFill Pro Station (Art.-Nr. 183363) erzeugt. Dazu wird ein HydroStik Pro Behälter (Art.-Nr. 183358) in der HydroFill Pro Station mit Wasserstoff befüllt, der elektrolytisch aus destilliertem Wasser erzeugt wird.

Um die Wasserstoff-Erzeugung vorzubereiten, muss die HydroFill Pro Station vom Stromnetz getrennt sein. Danach öffnet man vorsichtig die obere, durchsichtige Abdeckung und füllt destilliertes Wasser in den rechten Tank (Frischwasser-Tank) bis zur inneren Kante ein. Danach wird die Abdeckung geschlossen und ein HydroStik Pro Behälter in die vordere Öffnung eingesetzt und durch Festdrehen im Uhrzeigersinn fixiert.

Auf der linken Seite ist die HydroFill Pro Station ohne eingesetzten HydroStik Pro Behälter gezeigt und auf der rechten Seite mit eingesetztem HydroStik Pro Behälter.

Danach wird das Netzteil angeschlossen und die Betriebsleuchte unten links beginnt rot zu leuchten. Die Befüllung des HydroStik Pro Behälters dauert bis zu 6 Stunden. Dabei sind von Zeit zu Zeit mehr oder weniger laute Betriebsgeräusche hörbar, die durch das Spülen der HydroFill Pro Station entstehen. Jetzt muss man warten, bis die Betriebsleuchte nicht mehr rot, sondern grün leuchtet, bevor die Stromzufuhr unterbrochen werden darf.

Pro Stunde werden etwa 20 ml Wasser elektrolytisch in Wasserstoff (H_2) und Sauerstoff (O_2) zerlegt. Der Reinheitsgrad des dabei erzeugten Wasserstoffs liegt bei 99,99%. Dabei werden pro Stunde bis zu 3 Liter Wasserstoff erzeugt. Der Gewichtsunterschied des Behälters zwischen leer und gefüllt beträgt etwa 0,9 g.

Der Wasserstoff wird im HydroStik Pro Behälter als Feststoff (Hydrid) gespeichert. Dadurch sind keine Druck-Gasflaschen notwendig und der Umgang mit dem Wasserstoff ist weitaus weniger gefährlich als mit Druck-Gasflaschen.

Der bei der Elektrolyse erzeugte Sauerstoff wird an die Umgebungsluft abgegeben und zeigt sich durch Bläschen im Sichtfenster unterhalb der Abdeckung. Im HydroFill Pro wird eine PEM-Elektrolyse (PEM: Protonen-Austausch-Membran, proton exchange membrane) durchgeführt, bei der unter Gleichspannung das Wasser in Wasserstoff und Sauerstoff aufgespaltet wird.

Sobald die Betriebsleuchte dauerhaft grün leuchten, kann man die Stromzufuhr trennen und sollte beide Wassertanks leeren. Wenn die Betriebsleuchte im Sekundentakt grün leuchtet, dann wartet die HydroFill Pro Station darauf, dass ein HydroStik Pro Behälter eingesetzt wird. Ist ein HydroStik Pro Behälter bereits eingesetzt, muss man kontrollieren, ob dieser wirklich fest verschraubt ist.

Während der Befüllung des HydroStik Pro Behälters leuchtet die Betriebsleuchte dauerhaft rot. Wenn die Betriebsleuchte im Sekundentakt rot leuchtet, dann muss man Wasser im rechten Nutzwasser-Tank hinzufügen und sicherstellen, dass der linke Brauchwasser-Tank leer ist.

Wenn die rote Statusanzeige alle 3 Sekunden für 1 Sekunde eingeschaltet wird, dann muss man eine Regenerierung der HydroFill Pro Station durchgeführt werden. Dazu einen Beutel mit Apfelsäure in den Wassertank geben, ohne die HydroFill Pro Station auszuschalten. Danach mindestens eine Stunde lang weiter einen HydroStik Pro Behälter mit Wasserstoff befüllen lassen.

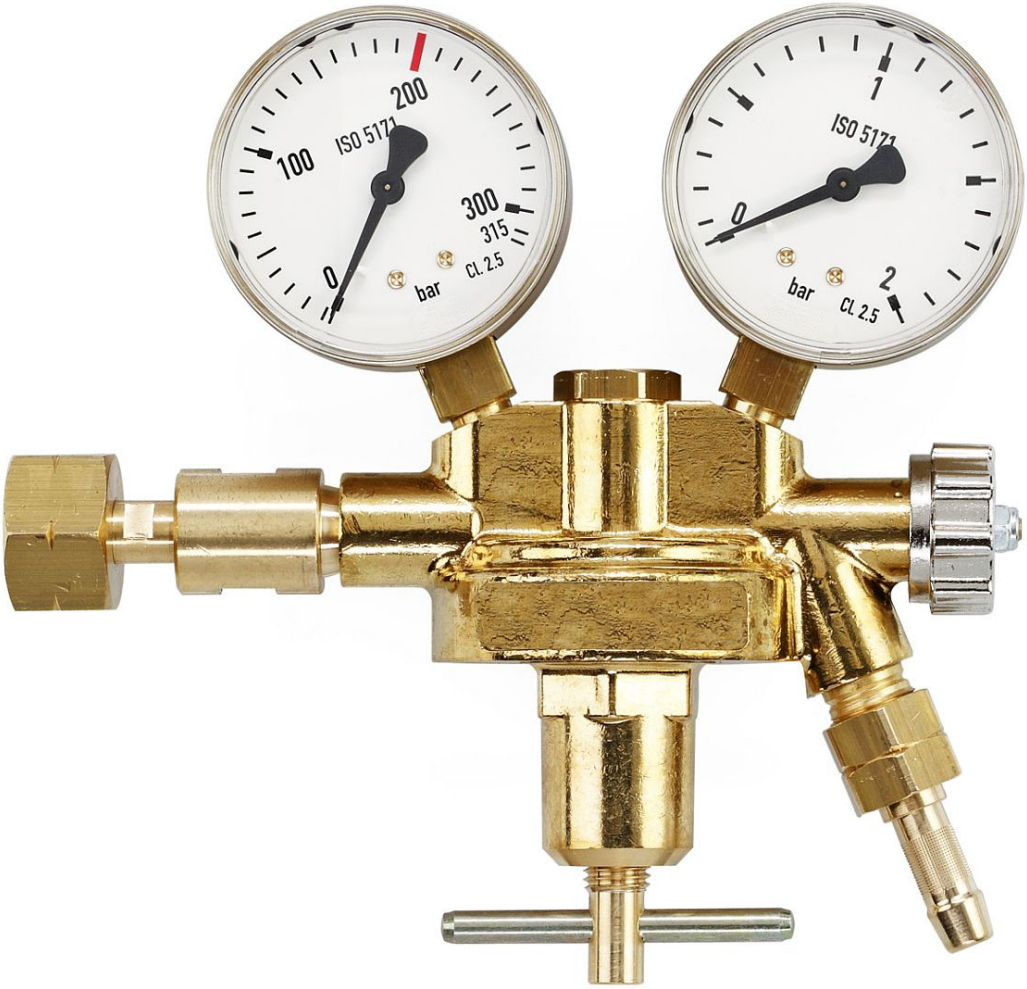
Mit Apfelsäure wird die Elektrolyse-Einheit regeneriert. Dabei werden Metallionen entfernt, die aus der Umgebung oder durch das Wasser in die Elektrolyse-Einheit gelangt sind. Die Elektrolyse-Einheit wird durch Metallionen in ihrer Funktion stark beeinträchtigt und durch die Apfelsäure werden die Metallionen komplex gebunden und ausgewaschen. Um die Wirkung der Apfelsäure zu verstärken, sollte man 40-70° heißes destilliertes Wasser in den Frischwassertank einfüllen. Apfelsäure ist ungiftig, allerdings nicht zum Verzehr geeignet.

Gefahrenhinweis: Wasserstoff ist ein brennbares Gas und bildet mit der Luft explosive Mischungen. Die Versuche und auch die Erzeugung von Wasserstoff dürfen daher nur in gut belüfteten Räumen durchgeführt werden.

6.2 Verwendung von Wasserstoff aus einer Druck-Gasflasche

In vielen Bildungseinrichtungen und Laboren steht Wasserstoff in Form von 150 - 200 bar (15.000 - 20.000 kPa entspricht 2.175 - 2.900 PSI) Gasflaschen zur Verfügung. Dieser Wasserstoff kann zum Betrieb unserer Brennstoffzelle genutzt werden. Die Brennstoffzelle benötigt für die Wasserstoffzufuhr 0,45 - 0,55 bar (45 - 55 kPa entspricht 6,5 - 8 PSI) Gasdruck. Mit einem geeigneten Druckminderer muss der Gasdruck für unsere Brennstoffzelle auf die benötigten 0,45 bis 0,55 bar (45 - 55 kPa entspricht 6,5 - 8 PSI) reduziert werden.

Dafür sollte ein Druckminderer verwendet werden, der für 200 bar Eingangsdruck ausgelegt ist und einen präzise regelbaren Ausgangsdruck von bis zu 1-2 bar zur Verfügung stellt.



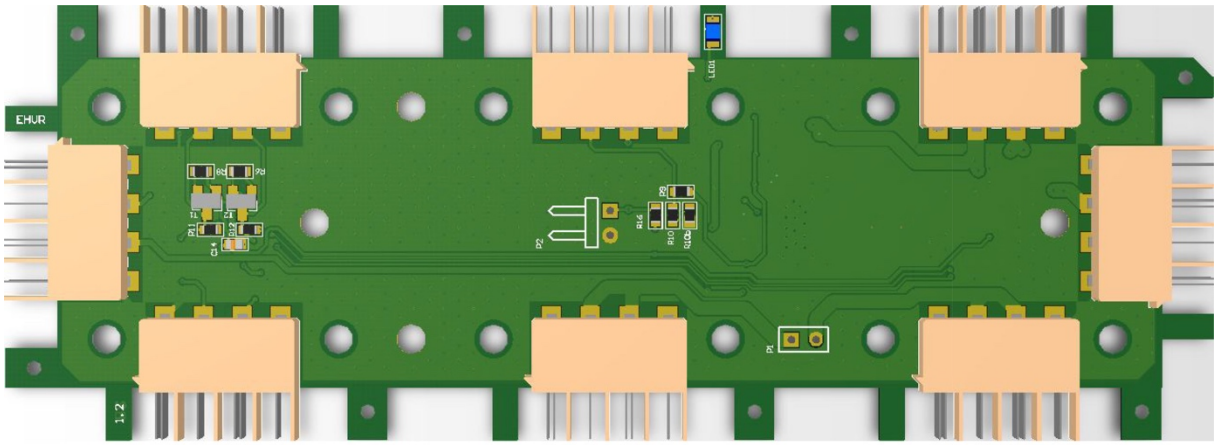
Wenn kein Druckminderer für einen so kleinen Ausgangsdruck zur Verfügung steht, wird der Gasdruck mit einem herkömmlichen Druckminderer im ersten Schritt auf 10 - 40 bar (1.000 - 4.000 kPa entspricht 145 - 580 PSI) reduziert. Danach wird der Wasserstoff in dem kleinen Druckminderer unserer Brennstoffzelle eingespeist, wo er auf die erforderlichen 0,45 bis 0,55 bar (45 - 55 kPa entspricht 6,5 - 8 PSI) reduziert wird.

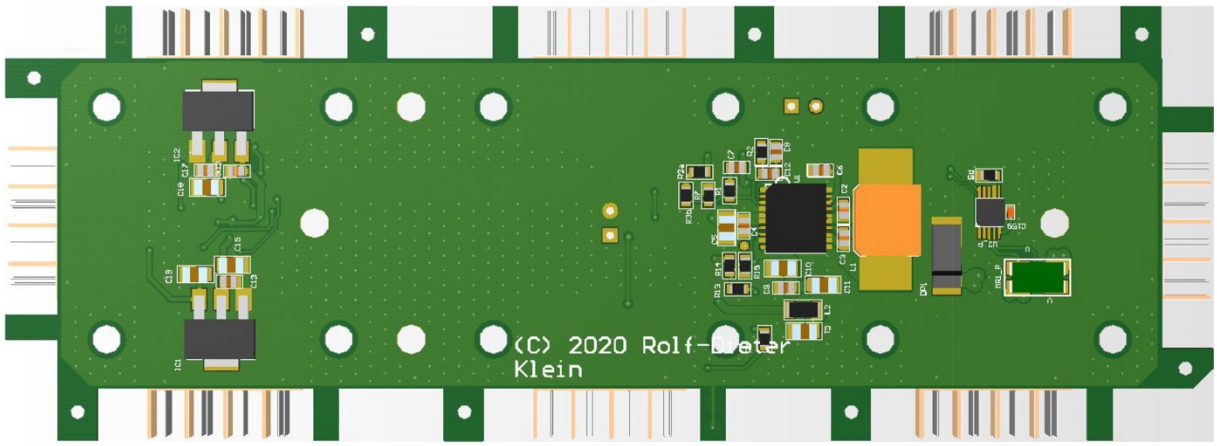
Der Druckminderer, der an der Wasserstoff-Gasflasche angeschlossen ist, sollte in jedem Fall über zwei Druckanzeigen verfügen, eine für den Eingangsdruck und eine für den Ausgangsdruck.

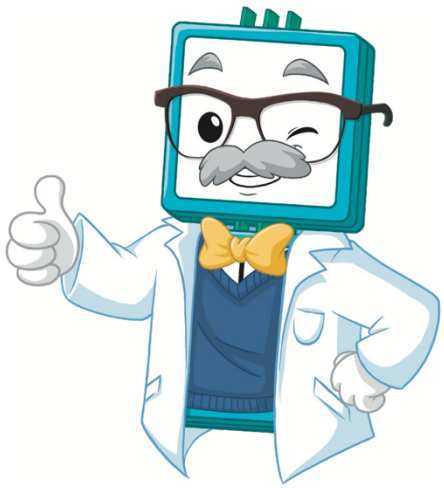
7. Anhang

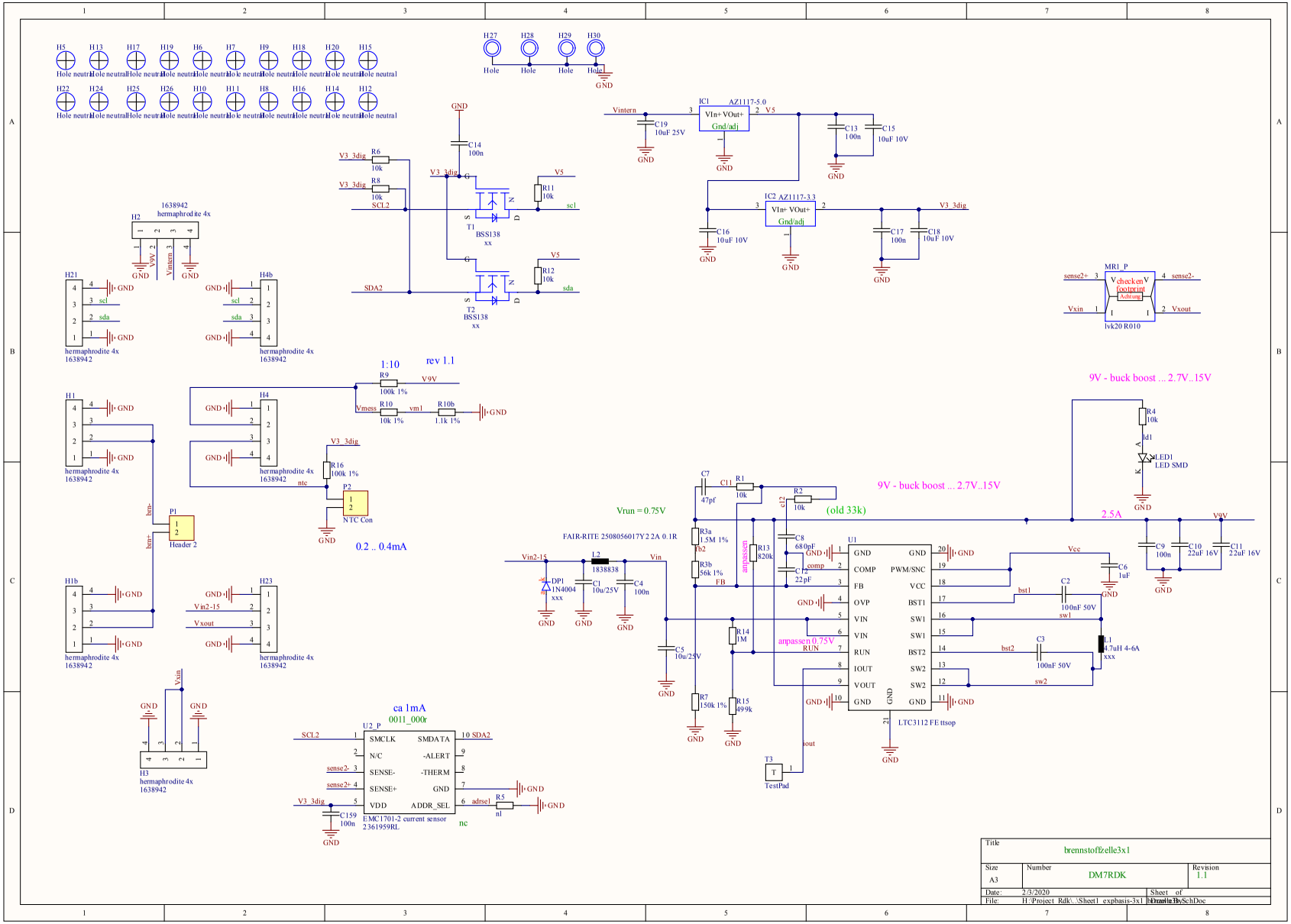
7.1 Schaltplan des Brennstoffzellen-Bricks

Die hier abgebildete schematische Darstellung und der Schaltplan dienen nur zum Verständnis der elektronischen Funktionen des Brennstoffzellen-Bricks. Sie müssen nicht im Detail mit dem Brick identisch sein, der dir vorliegt. Technische Änderungen sind jederzeit möglich.








H5 H13 H17 H19 H6 H7 H9 H18 H20 H15
Hole neutral
H22 H24 H25 H26 H10 H11 H8 H16 H14 H12
H27 H28 H29 H30
Hole
GND
IC1 AZ1117-5.0
VIn+ VOut+
Gnd/adj
Vintern
C19 10uF 25V
C13 100n
C15 10uF 10V
IC2 AZ1117-3.3
C16 10uF 10V
C17 100n
C18 10uF 10V
V3_3dig
V5
V9V
C14 100n
R6 10k
R8 10k
SCL2
SDA2
R11 10k
R12 10k
scl
sda
T1 BSS138 xx
T2 BSS138 xx
1638942 hermaphrodite 4x
H2
H21
H4b
H1
H4
H1b
H23
H3
hermaphrodite 4x 1638942
MR1_P
sense2+
sense2-
Vxin
Vxout
Ivk20 R010
1:10
rev 1.1
R9 100k 1%
R10 10k 1%
R10b 1.1k 1%
Vmess
vm1
R16 100k 1%
ntc
P2
NTC Con
P1
Header 2
bm-
bm+
0.2 .. 0.4mA
9V - buck boost ... 2.7V..15V
Vrun = 0.75V
FAIR-RITE 2508056017Y2 2A 0.1R
Vin2-15
L2 1838838
DP1 1N4004 xxx
C1 10u/25V
C4 100n
Vin
C7 47pf
C11 10k
R1
C12
R2 10k
(old 33k)
R3a 1.5M 1%
R3b 56k 1%
fb2
FB
anpassen
R13 820k
C8 680pF
C72 22pF
comp
U1
GND
COMP
FB
OVP
VIN
VIN
RUN
IOUT
VOUT
GND
PWM/SNC
VCC
BST1
SW1
SW1
BST2
SW2
SW2
GND
C5 10u/25V
R14 1M
anpassen 0.75V
R7 150k 1%
R15 499k
Iout
T3 TestPad
LTC3112 FE ttsop
2.5A
R4 10k
ld1
LED1 LED SMD
C9 100n
C10 22uF 16V
C11 22uF 16V
C6 1uF
Vcc
C2 100nF 50V
bst1
sw1
C3 100nF 50V
bst2
sw2
L1 4.7uH 4-6A xxx
ca 1mA
0011_000r
U2_P
SMCLK
SMDATA
N/C
-ALERT
SENSE-
-THERM
SENSE+
GND
VDD
ADDR_SEL
10 SDA2
adrse1
R5
nl
C159 100n
EMC1701-2 current sensor 2361959RL
nc
Title
brennstoffzelle3x1
Size
A3
Number
DM7RDK
Revision
1.1
Date:
2/3/2020
Sheet of
File:
H:\Project Rdk\..\Sheet1 expbasis-3x1 brennstoffzelle3x1.SchDoc
Drawn By:

7.2 Stückliste für das Brennstoffzellen-Set

An dieser Stelle sind alle Bestandteile des Sets, unabhängig davon ob es sich um Bricks oder Zubehör handelt, mit den jeweiligen Bestellnummern aufgelistet. Alle Bestandteile des Sets sind auch als Ersatzteil einzeln erhältlich.

Stückzahl x Allnet-Art.-Nr. • Hersteller-Nr. • Bezeichnung

1 x 118627 • ALL-BRICK-0221 • Netzteiladapter 9V 1A Sicherung und Masse
1 x 113629 • ALL-BRICK-0002 • Batterieadapter 9V 0,5A Sicherung und Masse
3 x 113630 • ALL-BRICK-0003 • Leitung Masse
1 x 133745 • ALL-BRICK-0615 • Kondensator 10.000 µF
1 x 142414 • ALL-BRICK-0665 • Lastwiderstand 5 Ω
1 x 182999 • ALL-BRICK-0717 • n-MOS Transistor IPP055N03L
1 x 183342 • ALL-BRICK-0723 • Folientaster-Brick zweifach mit Pfeilen up/down
2 x 113631 • ALL-BRICK-0004 • Leitung Gerade
4 x 113632 • ALL-BRICK-0005 • Leitung Ecke
3 x 113633 • ALL-BRICK-0006 • Leitung T-Kreuzung
1 x 113676 • ALL-BRICK-0049 • Leitung doppelt gerade
1 x 134017 • ALL-BRICK-0616 • Leitung Doppelspannung für Leistungstreiber
1 x 182906 • ALL-BRICK-0715 • Leitung als Kreuzung mit einfacher Trennung der Leitung
1 x 182907 • ALL-BRICK-0716 • Leitung als Winkel und Gerade für die Einzelpins
1 x 125674 • ALL-BRICK-0407 • 5-polige Klemme Typ 2
1 x 113636 • ALL-BRICK-0009 • LED rot
1 x 162698 • ALL-BRICK-0701 • LED dual auf Masse rot & grün Signal durchverbunden
1 x 118430 • ALL-BRICK-0182 • I2C MiniOLED Display
1 x 180230 • ALL-BRICK-0711 • Brennstoffzellen Brick mit Temperatur- und Spannungsmessung
1 x 180232 • ALL-BRICK-0712 • Strompfad-Weiche Brick (spannungsgesteuert)
1 x 180233 • ALL-BRICK-0713 • Konstantstrom Dual Last Brick (10 und 50mA)
1 x 172781 • ALL-BRICK-0707 • Arduino MKR Brick Adapter
1 x 157817 • ABX00023 • Arduino® Board MKR WiFi 1010 (WLAN)
1 x 159436 • 083011 • Batterie 9V-Block *Duracell* Industrial
1 x 130642 • Makeblock-Silicone Air Tube 0305 (1 Meter Länge)
1 x 183358 • Hydrostik PRO Kartusche
1 x 183360 • Druckminderer
1 x 183361 • Entlüftungstaster
1 x 183366 • Brennstoffzelle

Optional erhältlich:

Hersteller-Nr. • Bezeichnung

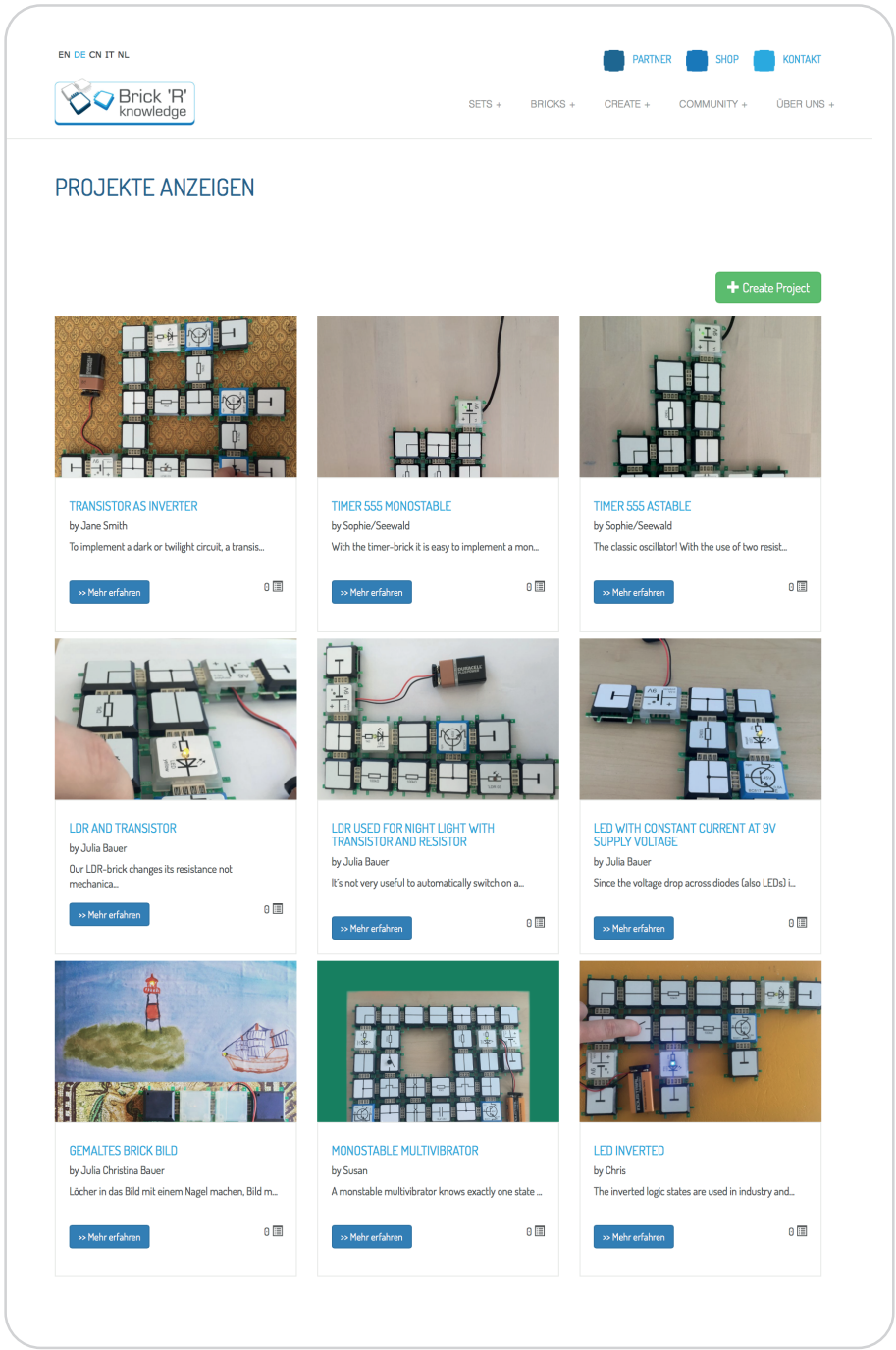
183363 • Hydrofill PRO Ladestation zum Erzeugen von Waserstoff

8. Brick Community

Das Brick Universum dehnt sich aus: Ob auf Messen, auf unserer Website, auf YouTube oder 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.

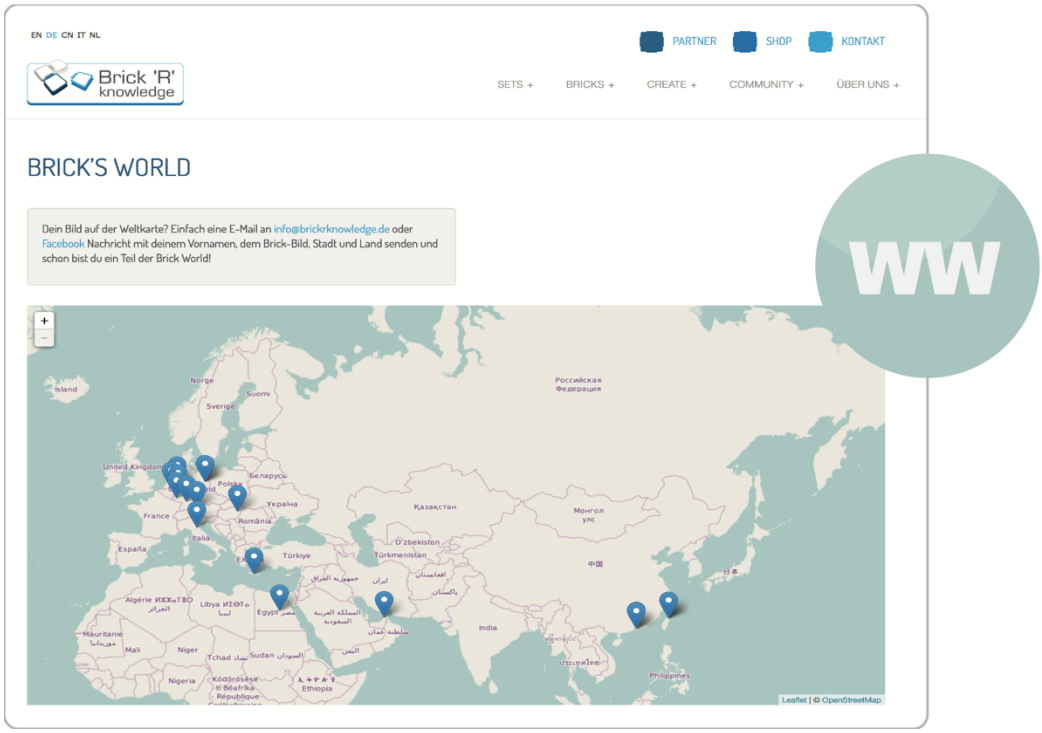


Social Media

Im Reiter „Community“ findest du all unsere Social Media Präsenzen und bleibst 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 wirst es bald auf der Weltkarte finden!



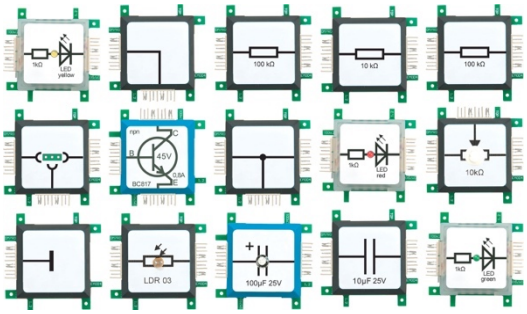
9. Brick Sets im Überblick

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.

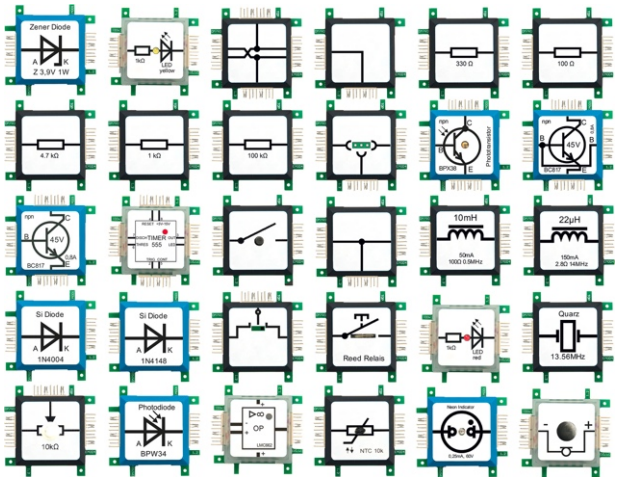


Advanced Set

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.

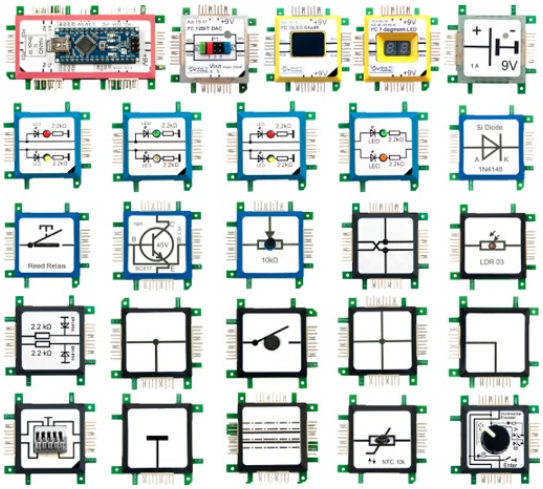


Arduino Coding Set

enthält 44 Bricks

ALL-BRICK-0374

Das Brick´R´knowledge Arduino Coding Set erweitert die Experimente hin zur Digitalelektronik mit der Einführung in die Mikrocontroller-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.

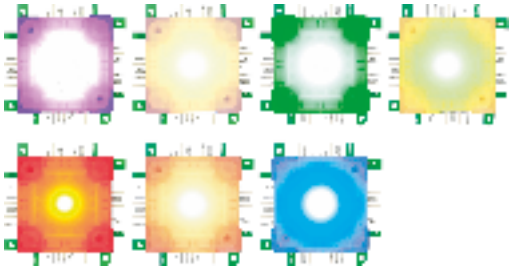


7 Color Light Set

enthält 28 Bricks

ALL-BRICK-0398

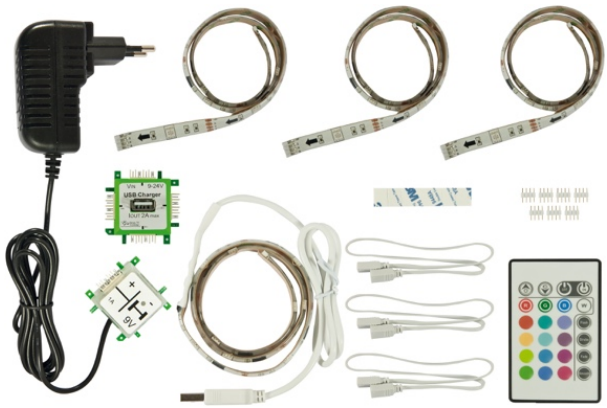
Mit den insgesamt 28 LED-Leucht-Bricks in 7 unterschiedlichen Farben lassen sich beeindruckende Lichtakzente in horizontaler und vertikaler Architektur setzen. Die 1 Watt LEDs in den Farben rot, gelb, blau, orange, violett, grün und warmweiß eignen sich perfekt für individuelle Licht-Figuren oder als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



RGB Color Light Set

ALL-BRICK-0619

Mit den insgesamt 28 LED-Leucht-Bricks in 7 unterschiedlichen Farben lassen sich beeindruckende LichtAk-zente 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-Fi-guren oder als mobile Beleuchtungs-lösung.

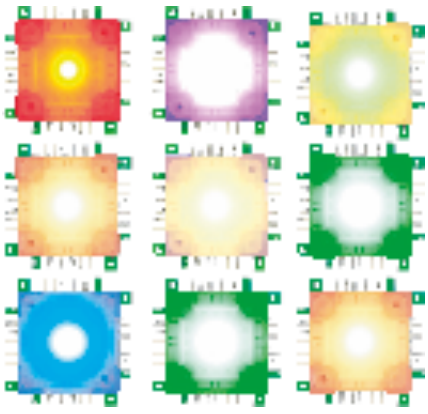


Programmable LED Set

enthält 49 Bricks

ALL-BRICK-0483

Das Set beinhaltet 49 ansteuerbare RGB-LED-Bricks mit zwei oder drei Anschlüssen, sowie einen An-schluss-Brick 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 Bild-animationen zu erstellen und sich spielerisch mit Mi-krocontroller-Programmierung zu befassen. Innova-tive Lichtinstallationen und individuell leuchtende, blinkende und pulsierende Bilder in unterschiedli-chen Farb- und Helligkeitsstufen sind durch das Pro-grammable LED Set wunderbar umsetzbar.

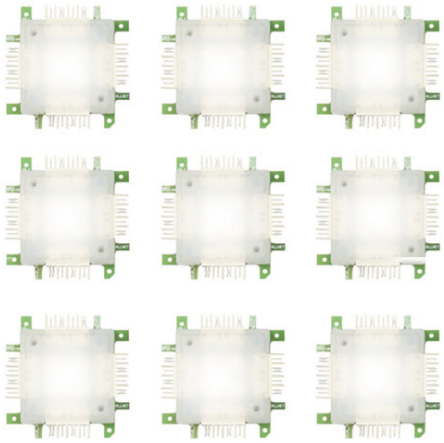


Highpower LED Set

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.

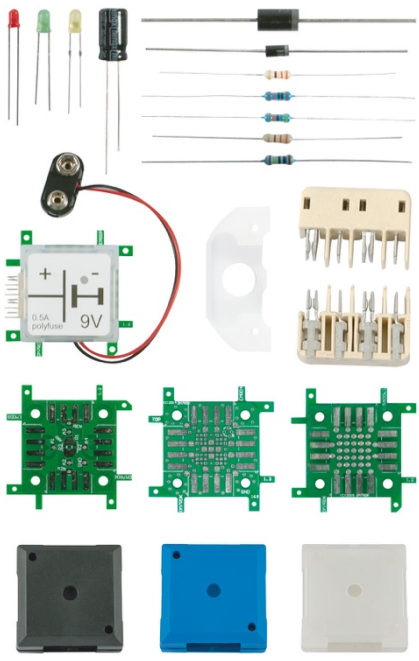


DIY Set

ALL-BRICK-0399

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.

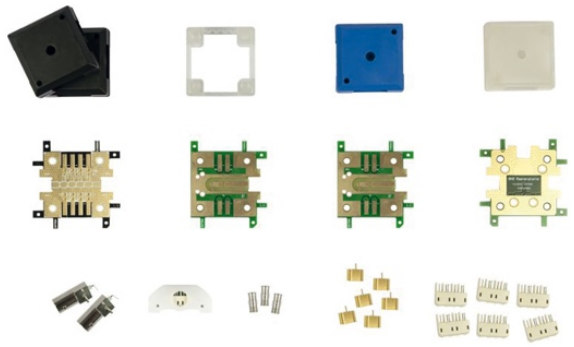
Mit Lötkolben und Lötzinn können die Tüftler die Standard-Bricks nachbauen oder eigene Bricks für individuelle Spezialanwendungen herstellen und somit sogar eigene Sets entwickeln.



MHz DIY Set

ALL-BRICK-0457

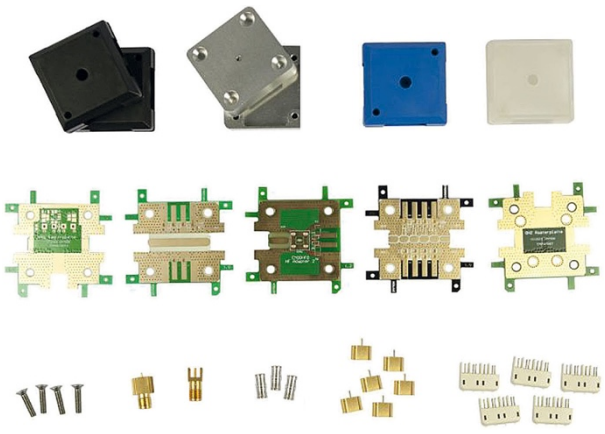
Mit dem MHz DIY Set lassen sich eigene Bausteine für Experimente und Schaltungen im MHz-Bereich realisieren. Das Set enthält drei verschiedene Raster- und Experimentierplatinen, sowie BNC-Buchsen, P-SMP-Stecker und die dazu passenden Verbinder. Außerdem enthält das Set eine Lötlehre für die SMD-Stecker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

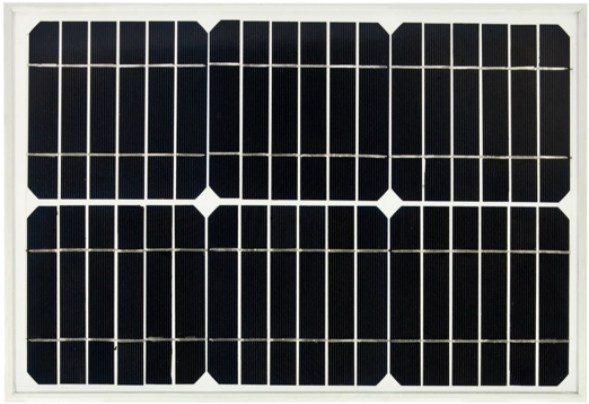
enthält 20 Bricks

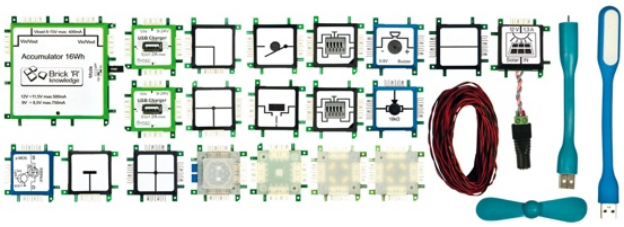
ALL-BRICK-0458

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 sind Sie und Ihre Kinder offizielle Mitglieder der Maker-Generation.





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.

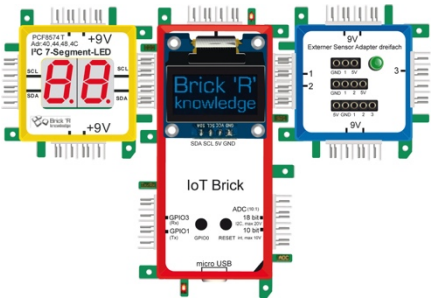
Internet of Things Set

enthält 13 Bricks

ALL-BRICK-0646

Mit dem Internet of Things Set ist es nun möglich, die Bricks via Internet zu kontrollieren. Mit dem zentralen IoT-Brick werden Sie beispielsweise lernen, Ihre erste Website zu bauen und I/O Pins mit Ihrem Smartphone zu steuern. Außerdem enthält das Set einen Temperatur- und Luftfeuchtigkeitssensor, dessen Werte Sie auf einem Display darstellen können: Der erste Schritt zur eigenen Home Automation!

Sie können auch Daten, wie zum Beispiel den Dollar-Kurs aus dem Internet abfragen und sich anzeigen lassen. Um die 7-Segmentanzeige anzusteuern, wird der sogenannte I²C-Bus genutzt, den Sie auch kennenlernen. Das Internet der Dinge wartet darauf, von Ihnen entdeckt zu werden!

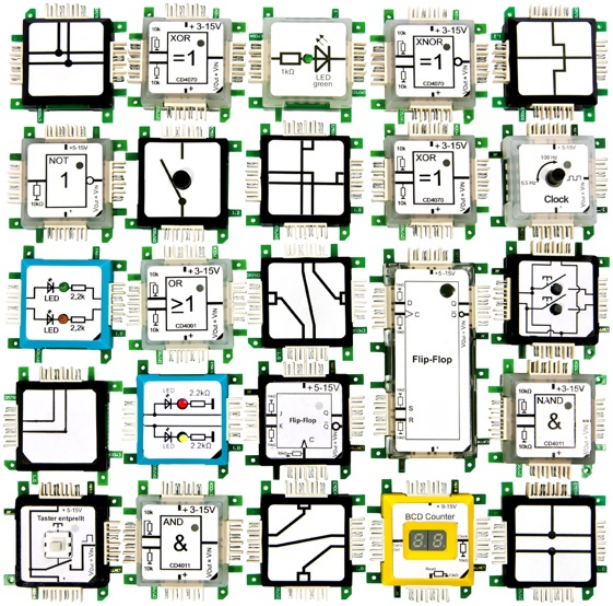


Logic Set

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 Zusammenbauen und Experimentieren mit den Bricks macht Spaß und animiert zum Bau von eigenen Schaltungsvarianten.



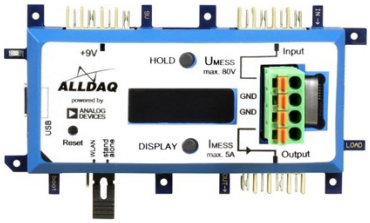
Der Lieferumfang des Logic Sets reicht von einfachen Logik-Bricks (AND, OR, NAND, NOR, XOR, XNOR, NOT) über verschiedene Flipflop-Bricks (D-, RS- und JK-Typ), weiter über einen Taktgeber-Brick (alternativ ein entprellter Taster für Einzelimpulse) bis hin zu einem BCD-Counter-Brick mit integrierter 7-Segment-Anzeige. Eine umfassende Auswahl an LED-, Taster- und Leitungs-Bricks runden das Set ab.

Powermeter Set

enthält 21 Bricks

ALL-BRICK-0696

Messungen, Formeln & Mathematik! Endlich kannst du direkt in deinen Brick Schaltungen Messungen durchführen! Zuerst wirst du durch die Unterschiede der Spannungs- und Stromstärkenmessung geführt und verstehst, wie Messgeräte im Inneren funktionieren. Dann geht es weiter mit Experimenten zum Stromverbrauch: Wie verhält sich dein Handy beim Laden? Teste und erforsche das Ohm'sche Gesetz und Reihen- und Parallelschaltungen. Am Schluss beherrschst du sogar die Brückenschaltungen und die Dreieck-Stern-Transformation. Hört sich kompliziert an? Nicht mehr lange!

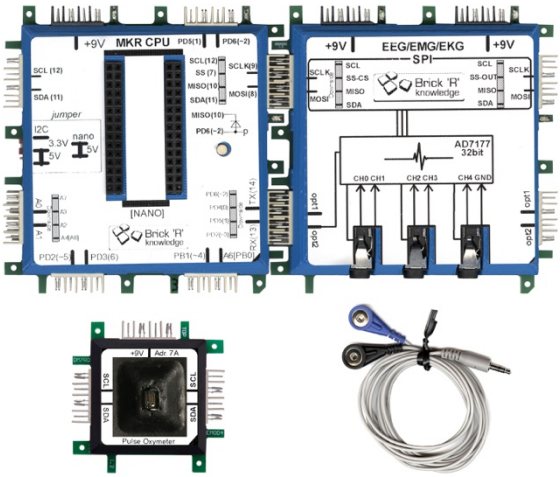


Bio Feedback Set

enthält 11 Bricks

ALL-BRICK-0703

Das Set besteht aus einem Puls-Oxymeter Brick, der optisch den Blutsauerstoffgehalt messen kann, einem Arduino-MKR-Adapter-Brick mit passendem Arduino MKR WiFi 1010 Board sowie dem eigentlichen EEG/EMG/EKG Brick, der einen hochauflösenden A/D Umsetzer enthält, welcher 4 gleichzeitig nutzbaren Kanäle besitzt. Dazu einige Leitungs-Bricks, die Stromversorgung mit 9V Batterie und Anschlusskabel sowie passenden Pads, die auf der Haut befestigt werden müssen.



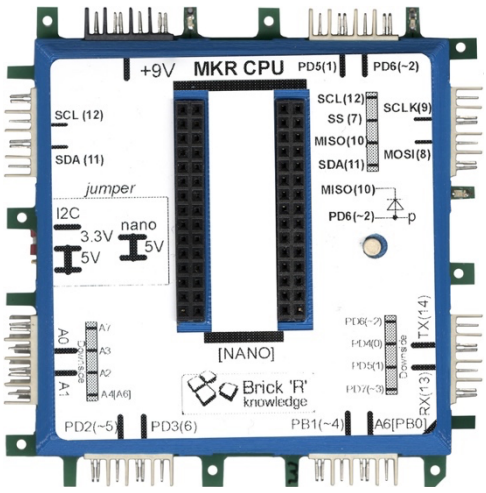
Dieses Set ist kein Medizinprodukt und auch nicht als Ersatz für ein Medizinprodukt gedacht. Es dient lediglich zu Lehrzwecken und zur Veranschaulichung der Theorien, die hinter diesen Dingen stehen.

MKR Brick

enthält 1 Brick

ALL-BRICK-0707

Der Arduino MKR ist das Herz für unseren Arduino-MKR-Adapter-Brick. Der Prozessor wird oben auf die Steckleisten gesetzt. Die Programmierung erfolgt wie bei Arduino üblich, über den PC mit dem Arduino Entwicklungssystem, welches man sich von der Arduino-Homepage entsprechend runterladen kann. Der MKR Adapter Brick ist kompatibel mit allen Arduinos aus der MKR Familie sowie dem Arduino Nano.



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
Prenzlauer Allee 173/Ecke Zelterstraße
D-10409 Berlin
www.maker-store.de

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