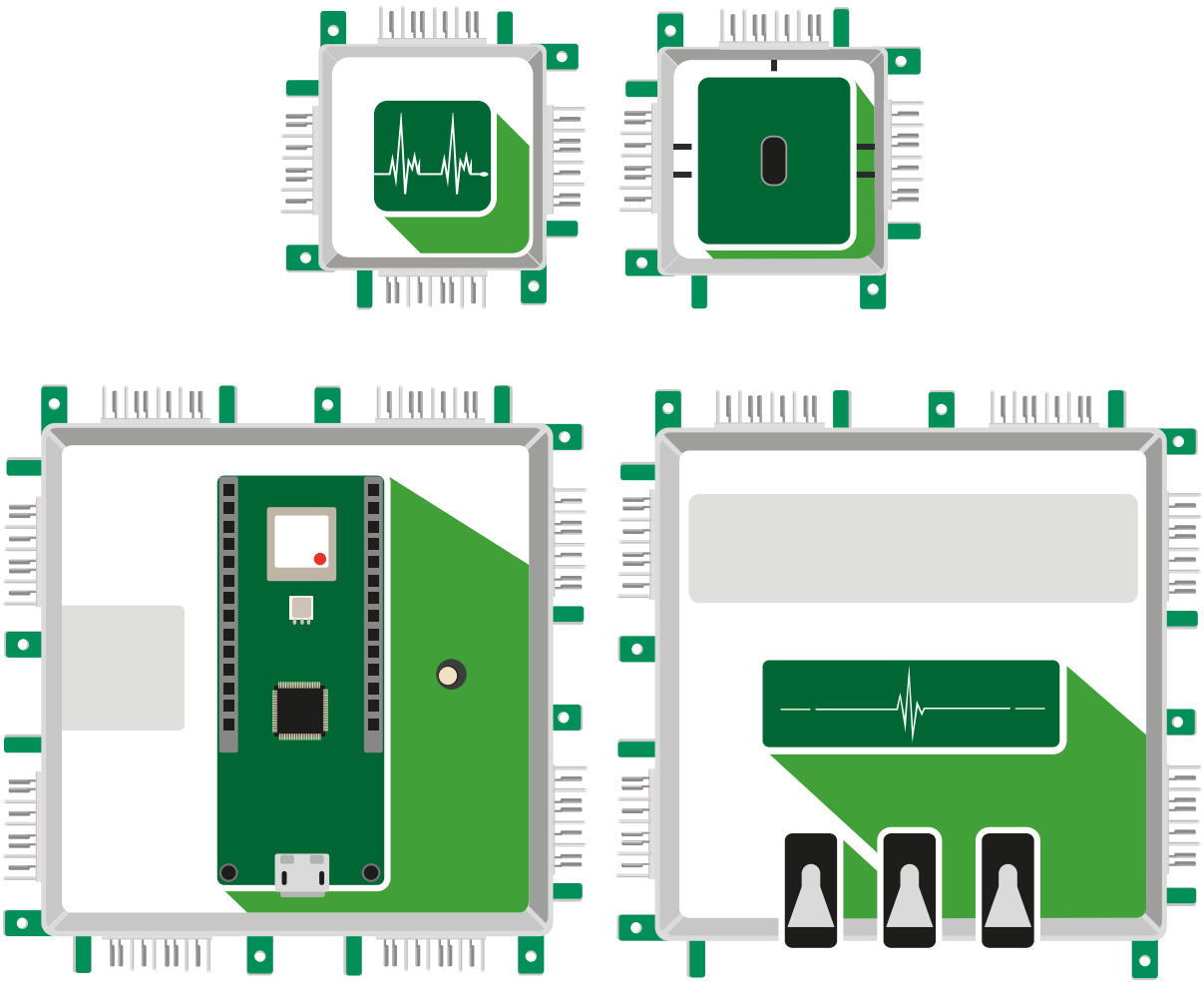
Brick 'R' knowledge

Bio Feedback Set

Handbuch

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

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



Impressum

Brick'R'knowledge Bio Feedback Set Anleitung Rev. 1.0

Datum: 13.06.2019

ALLNET® GmbH Computersysteme

Brick'R'knowledge Maistraße 2 D-82110 Germering

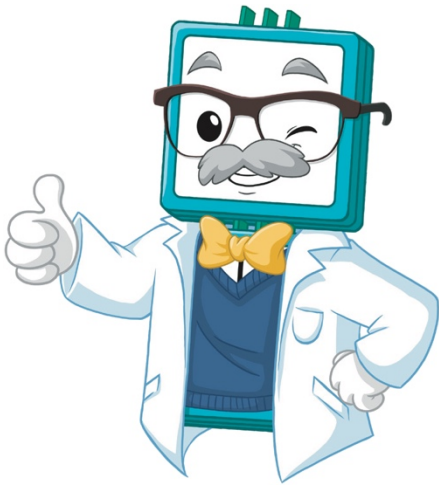
Alle in dieser Anleitung enthaltenen Informationen wurden mit größter Sorgfalt und nach bestem Wissen zusammengestellt. Dennoch sind Fehler nicht ganz auszuschließen. Für die Mitteilung eventueller Fehler sind wir jederzeit dankbar. Bitte sende diese an info@brickrknowledge.de.

Inhaltsverzeichnis

Impressum 2
Inhaltsverzeichnis 3
1. Vorwort 5
1.1 Sicherheitshinweise 6
2. Grundlagen des Brick'R'knowledge Systems 7
2.1 Der Masse-Brick 7
2.2 Die Spannungsversorgung 7
2.3 Signalführung 9
3. Grundlagen allgemein 10
3.1 Elektrische Größen 10
3.2 Präfixe 11
4. Die Bricks im Überblick 12
4.1 Versorgungs-Bricks 12
4.2 Leitungs-Bricks 13
4.3 Anzeige- und LED-Bricks 13
4.4 Arduino MKR WiFi 1010 und Arduino-MKR-Adapter-Brick 14
4.5 Puls-Oxymeter Brick 15
4.6 EEG/EMG/EKG Brick 16
4.7 Verbindung des Arduino-MKR-Adapter-Bricks mit dem EEG/EMG/EKG Brick 18
5. Bio Feedback Projekte - Einstieg in die Arduino Welt 19
5.1 Das Herzstück: Arduino MKR WiFi 1010 Board 19
5.1.1 Spezifikation des MKR WiFi 1010 Boards 19
5.1.2 PullUp-Widerstände 20
5.2 Die Programmierung: Arduino IDE 20
5.2.1 Bibliotheken installieren 21
5.2.2 Virtueller COM-Port-Treiber (optional) 21
5.2.3 Serieller Monitor 22
5.2.4 Serieller Plotter 22
5.3 Das erste Programm: Blink 22
5.3.1 Verbindungen herstellen 22
5.3.2 Der Sketch 23
5.3.3 Die Setup-Routine „void setup()“ 24
5.3.4 Die Programmschleife „void loop()“ 24
5.4 Doppelte LED 25
5.5 I2C-Bus 26
5.6 Analog-Digital Umsetzer 28
5.6.1 A/D Umsetzer - prinzipieller Aufbau 28
5.6.2 I/O Testprogramm - SetPinsTEST 30
5.7. OLED 31
5.7.1. Grafische Anzeige: Das OLED Display 31
5.7.2 OLED Brick Bibliothek 33

5.7.3. OLED Display über I2C 35
5.7.4. OLED Display und Zeichensatz 36
6. Puls Oxymeter Brick 37
6.1 Puls Oxymeter Brick - Erster Test 38
6.2 Puls Oxymeter Brick als Näherungsschalter 40
6.3 Puls Oxymeter Brick als Temperatursensor 42
6.4 Puls Oxymeter Brick - Einfache Pulsausgabe + über Plotterfunktion 44
6.5 Puls Oxymeter Brick - Sauerstoffgehalt 46
6.6 Puls Oxymeter Brick - PULS mit OLED und digitalem Filter 49
7. EKG Messung 54
7.1 Theorie der EKG Messung - Wir hören auf unser Herz! 54
7.1.1 Was ist ein EKG? 54
7.1.2 Was passiert im Körper? 54
7.1.3 Eine kurze Geschichte des EKG 54
7.2 Biologie: Das Herz, ein spannender Muskel 55
7.2.1 Erst einmal hinlegen: das Herz in Ruhe 55
7.2.2 Das Herz im Langzeit Test 55
7.2.3 Wie verhält sich das Herz bei Belastung? 55
7.3 EKG Messung grafisch auf dem PC mit dem seriellen Plotter 56
7.4 EKG Messung grafisch auf dem PC mit dem seriellen Plotter (CH2/CH3) 63
7.5 EKG Messung grafisch auf dem OLED-Display 69
7.6 EKG Messung grafisch auf dem OLED-Display mit Pulsfrequenzanzeige 76
7.7 Theorie der EMG Messung 84
7.7.1 Was ist ein EMG? 84
7.7.2 EMG vs. OEMG 84
7.7.3 Was passiert im Körper? 84
7.7.4 Spontanaktivität und der Grund des EMGs 85
7.8 EMG Messung von Muskel-Strömen (Muskelsteuerung) 86
8. Gehirnströme Messung 93
8.1 EEG Messung grafisch am Terminal ausgeben 93
8.2 EEG Messung grafisch am OLED-Display ausgeben 100
8.3 EEG Messung mit Ausgabe der Frequenzspektren (FFT) 107
8.4 EEG Messung Einteilung in Alpha, Beta, Gamma 114
8.5 EEG Messung - Mind-Control einer LED 123
9. Brick Community 132
10. Brick Sets im Überblick 134

1. 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 das neue Bio Feedback Set 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.

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.

Viel Spaß mit dem Set wünscht

Rolf-Dieter Klein

1.1 Sicherheitshinweise

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

Zur Spannungsversorgung (9V) ausschließlich das mitgelieferte Netzteil (Batteriebaustein) 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 werden LED-Bausteine mitgeliefert: Dual-LED „rot“ (2mA) und „gelb“ (2mA) mit einer Stromaufnahme von 2 Milli-Ampere.

Es ist unbedingt darauf zu achten, dass das mitgelieferte Netzteil (Batteriebaustein) 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!

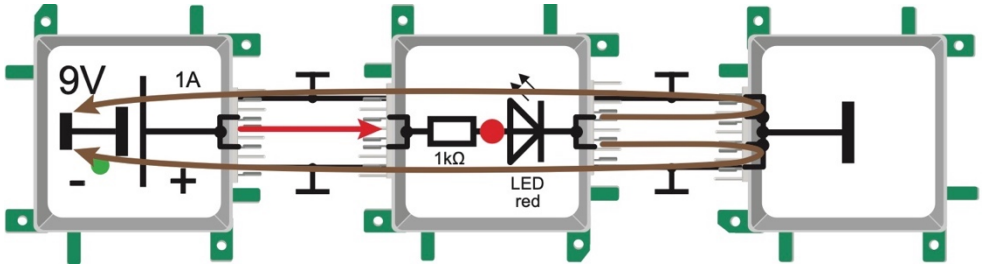
2. Grundlagen des Brick'R'knowledge Systems

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.

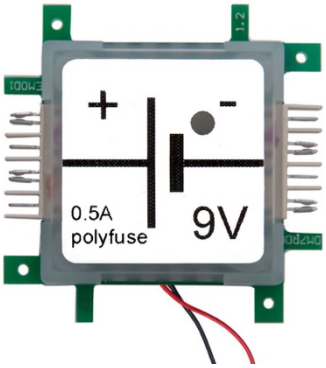
In unserer Schaltung sind das 9Volt gegenüber 0Volt: Man spricht einfach nur „Neun Volt“. Man erstellt in der Elektronik Schaltungen so, dass nachdem alle Bauelemente in ihrer Funktionsweise in die mehr oder weniger komplexen Stromkreise eingebracht sind, diese mit der „Masse“ verbunden werden. Schaltpläne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

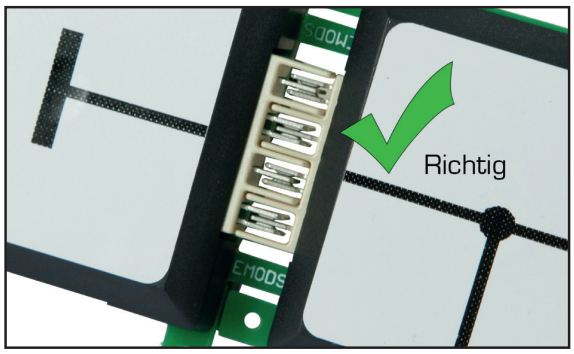
2.2 Die Spannungsversorgung

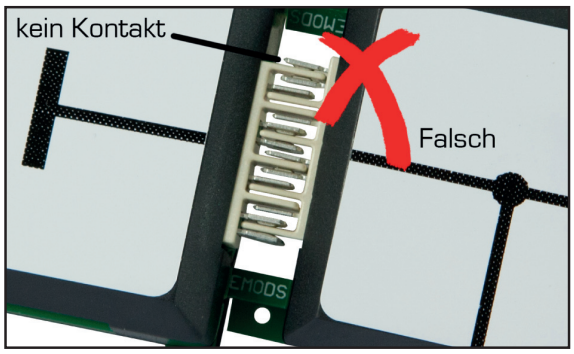


Die Spannungsversorgung des Bio Sets erfolgt über den mitgelieferten Versorgungs-Brick via 9V-Blockbatterie (ALL-BRICK-0001). Alternativ steht auch ein 9V-Steckernetzteil (ALL-BRICK-0221) zur Verfügung. Das Steckernetzteil liefert eine stabilisierte Gleichspannung von 9V und einen Maximalstrom von 1A. Bei Überlastung schaltet das Netzteil 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

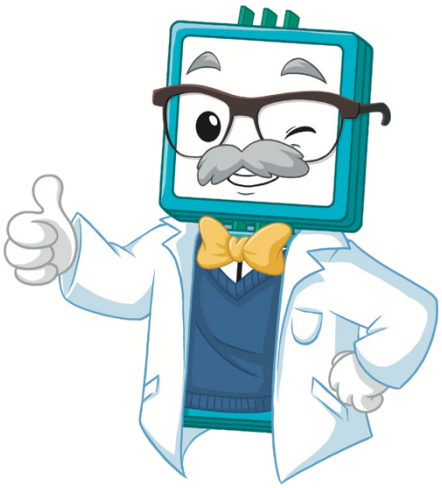
Im linken Bild seht ihr eine richtig gesteckte Verbindung. Die Verbindung besteht jeweils aus kleinen Stiften, die sich mechanisch verklemmen und dabei eine elektrische Verbindung herstellen. Um eine Isolation zwischen den Kontakten zu gewährleisten und einen Kurzschluss zu verhindern sind dazwischen 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 Abb. 4 rechts zu sehen, wird dies entsprechend dargestellt.

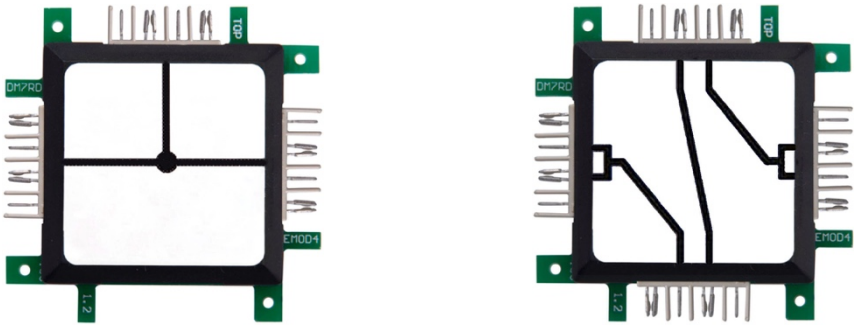
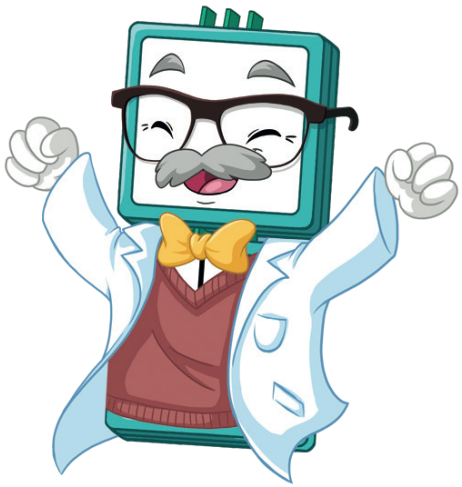


Abb. 4: Signalführung einpolig (links) und zweipolig (rechts)



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	N
(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

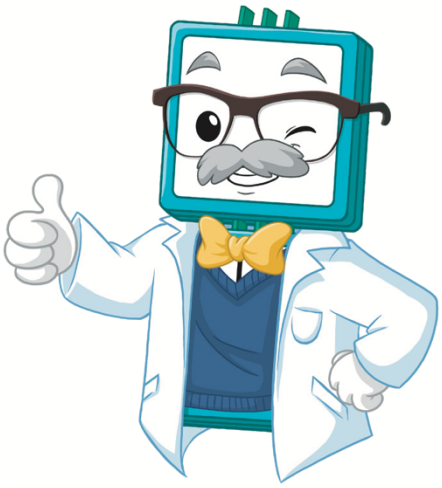
…und eine umfassende Liste physikalischer Größen unter:

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)

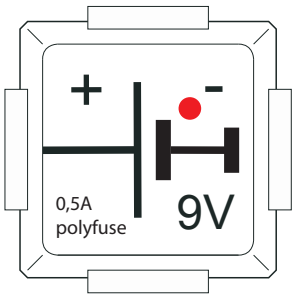
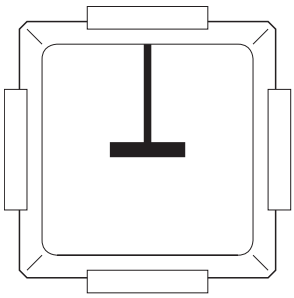


4. Die Bricks im Überblick

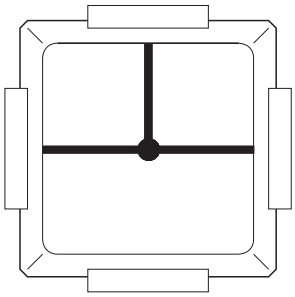
Die Bricks sind kompakte Bausteine aus der Welt der Elektrotechnik. Sie eignen sich zum Ausprobieren neuen Wissens für junge Forscher, die den Umgang mit komplexen Schaltungen kennenlernen wollen.

Das Set beinhaltet die Bricks, welche nachfolgend kurz vorgestellt werden.

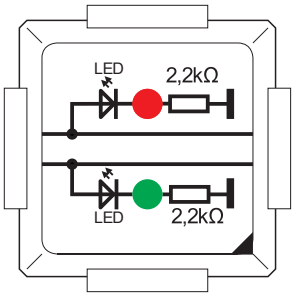
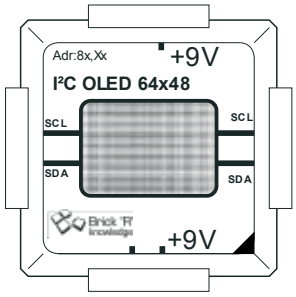
4.1 Versorgungs-Bricks

Abbildung	Anzahl	Kurzbeschreibung	Art.-Nr. / Brick-ID
	1	9V Batterie-Brick	Art.-Nr.: 113629 Brick-ID: ALL-BRICK-0002
		Der Batterie-Brick verwendet eine 9V Block-Batterie und versorgt die Schaltung mit einer Gleichspannung von 9V. Die LED leuchtet rot, sobald die selbstheilende Sicherung vom Typ Polyfuse bei Kurzschluss oder Überlastung (>0,5A) den Stromkreis trennt. Beachte: Es empfiehlt sich die Schaltung vor Inbetriebnahme nochmals zu kontrollieren, da ansonsten die Gefahr besteht, dass durch Kurzschluss, Verpolung oder einen anderen Fehler empfindliche Bauteile beschädigt werden! Die Batterie ist am Ende der Versuchsdurchführung sofort von der Schaltung zu trennen!	
	1	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.	
	1	9V Block-Batterie	Art.-Nr.: 159436
		9 Volt Block-Batterie für den Batterie-Brick. Der A/D-Wandler auf dem EKG-Brick benötigt relativ viel Strom, so dass man für längeres Experimentieren mehrere Batterien bevorraten sollte.	

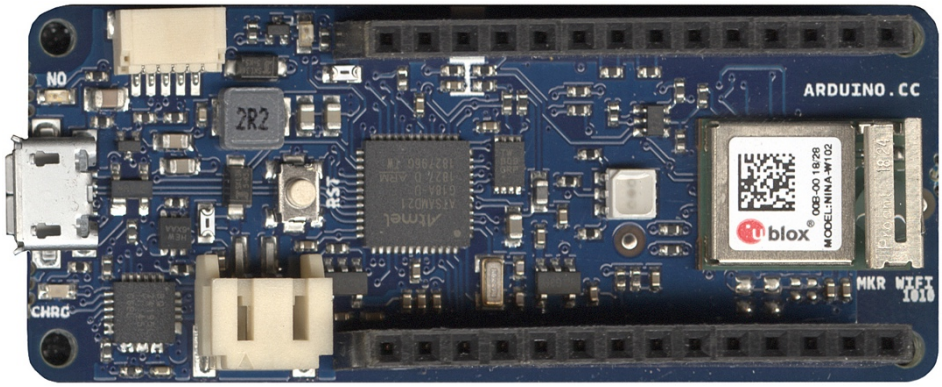
4.2 Leitungs-Bricks

Abbildung	Anzahl	Kurzbeschreibung	Art.-Nr. / Brick-ID
	4	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.	
	3	Duplex-Kabel für Masse/Saugnäpfe	
		Duplex-Kabel für Masse und Saugnäpfe. Drei von diesen Kabeln, mit verschiedenen Farben, sind im Set vorhanden. Dabei sind immer zwei Kabel an einem Stecker befestigt. Das Kabel mit dem schwarzen Anschluss ist das Massekabel, welches ganz rechts eingesteckt werden muss. Es handelt sich um geschirmte Kabel, also bitte vorsichtig damit umgehen. Die Clips am Kabelende sind für handelsübliche Kontaktpads für den Anschluss an die Haut gedacht.	

4.3 Anzeige- und LED-Bricks

Abbildung	Anzahl	Kurzbeschreibung	Art.-Nr. / Brick-ID
	1	Dual-LED auf Masse (rot/grün)	Art.-Nr.: 162698 Brick-ID: ALL-BRICK-0701
		In diesem Baustein sind zwei LEDs (rot/grün) untergebracht, die intern mit Masse verbunden sind. Die Signalleitungen sind getrennt durchverbunden. Beide LEDs sind über einen 2,2 kΩ Vorwiderstand vor zu hohem Stromfluss geschützt. Sie sind für 2 mA Strom bei 5 V Spannung ausgelegt. Die beiden Widerstände sind intern mit Masse verbunden, sodass man den Baustein direkt anschließen kann.	
	1	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.	

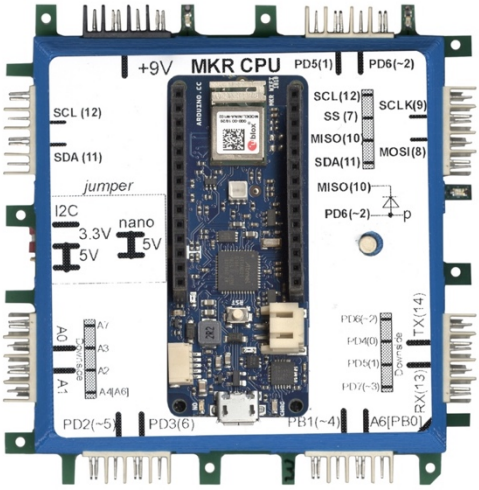
4.4 Arduino MKR WiFi 1010 und Arduino-MKR-Adapter-Brick



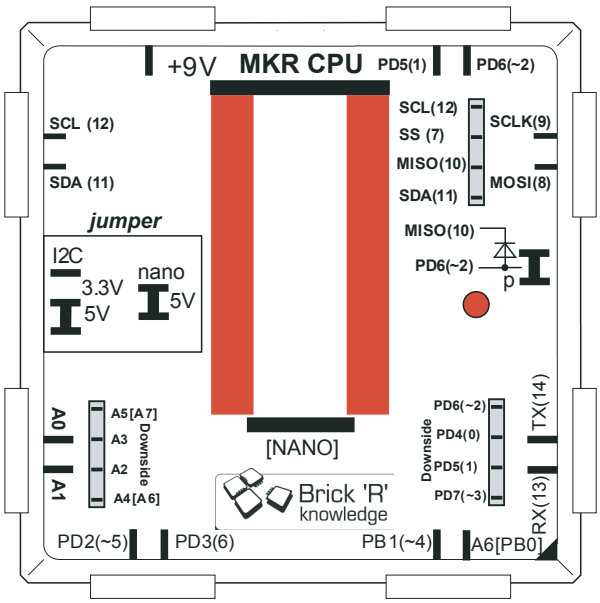
Die Arduino MKR WIFI 1010 Platine ist mit einem ESP32-Modul von U-BLOX ausgestattet, welches die Entwicklung von WiFi-basierten Programmen stark vereinfacht gegenüber den bisherigen Arduino-Platinen. Das WiFi-Modul benötigt dabei nur sehr wenig Strom und unterstützt 2,4 GHz IEEE® 802.11 b/g/n Wi-Fi sowie ECC508 Crypto-Authentifizierung (SHA-256).

Als Mikrocontroller für die Arduino-IDE kommt eine SAMD21 Cortex-M0+ 32 Bit Low Power ARM MCU zum Einsatz, aus der selben Prozessorfamilie stammt, die bereits seit vielen Jahren im Arduino DUE eingesetzt wird. Der USB Port kann zur Stromversorgung sowie zur seriellen Kommunikation und zum Programmieren des Arduino MKR WiFi 1010 genutzt werden. Anders als die meisten anderen Arduino-Platinen, wird die MKR WIFI 1010 Platine mit 3,3 Volt betrieben. Höhere Spannungen als 3,3 Volt können dabei die Arduino MKR WIFI 1010 Platine 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 angechlossen 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 5V ausgelegt sind.



Bitte den Arduino MKR WIFI 1010 wie er auf der Abbildung links zu sehen ist auf den MKR Brick setzen. Der 2x2 MKR Brick enthält dabei nicht nur die Stromversorgung für den Arduino MKR WIFI 1010, 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.



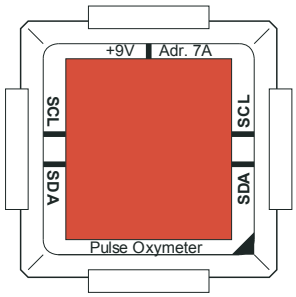
ALLNET Brick'R'knowledge MKR Brick

Hersteller-Nummer: ALL-BRICK-0707

ALLNET-Art.-Nr.: 172781

I/O Zuordnung		**Portbelegung**		**Numerischer Arduino-**
PD0 - 0	A0 - Analog 0	**Kürzel**	**Port**	**Parameter**
PD1 - 1	A1 - Analog 1	SCK	PB5	(9)
PD2 - 2 (5)	A2 - Analog 2	MOSI	PB3	(8)
PD3 - 3 (6)	A3 - Analog 3	SS	PB2	(7)
PD4 - 4 (0)	A4 - Analog 4	MISO	PB4	(10)
PD5 - 5 (1)	A5 - Analog 5			
PD6 - 6 (2)	A6 - Analog 6	SCL	A5	(12)
PD7 - 7 (3)	A7 - Analog 7	SDA	A4	(11)
PB1 - 9 (4)		TX	PD1	(14)
		RX	PD0	(13)

4.5 Puls-Oxymeter Brick



ALLNET Brick'R'knowledge Puls Brick

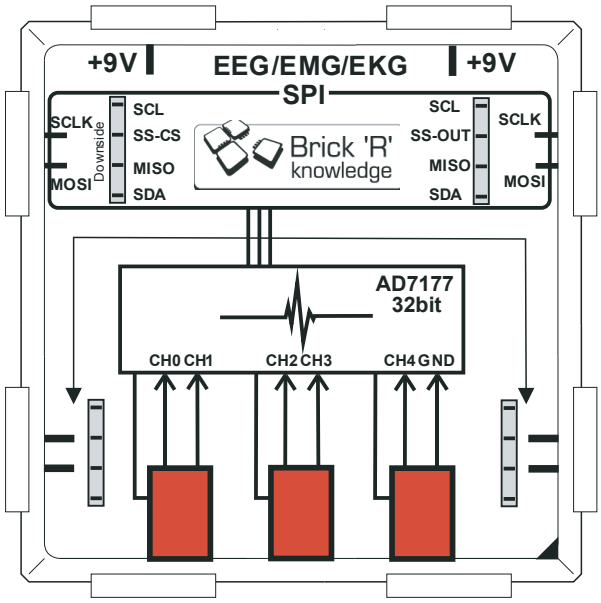
Hersteller-Nummer: ALL-BRICK-0705

ALLNET-Art.-Nr.: 172779

Hochempfindliches Pulsoxymeter MAX30102 (I2C Adressen AE und AF), wie es auch in Industriegeräten verwendet wird. Es verwendet eingebaute LEDs (Rot/IR) zur Anregung und eine Photodiode zur Auswertung. Die Auswertung geschieht intern mit einem 18 Bit A/D-Umsetzer und die Kommunikation erfolgt über den I2C Bus. Im Baustein ist ein Spannungsregler, so dass der Baustein an 9V betrieben werden kann. Ein Pegelumsetzer transformiert den I2C Pegel auf 5V (für Arduino). Beim MKR sollte man die Brücke daher auf 5V stellen.

Wenn man den Finger auflegt muss man ihn mindestens für ca. 10 Pulsschläge also ca. 10 Sekunden ruhig auflegen, da die Abtastung optisch erfolgt werden Bewegungen des Fingers sonst fehlinterpretiert. Kommerzielle Geräte haben meist eine Fingerklemmer, auf die wir der Anschauung wegen aber verzichtet haben. Man kann aber ein Gummiband um den Brick wickeln und damit den Finger besser fixieren, wie auch im Datenblatt empfohlen. Eigenen Experimenten steht hier alles offen.

4.6 EEG/EMG/EKG Brick



ALLNET Brick'R'knowledge EKG Brick

Hersteller-Nummer: ALL-BRICK-0706

ALLNET-Art.-Nr.: 172780

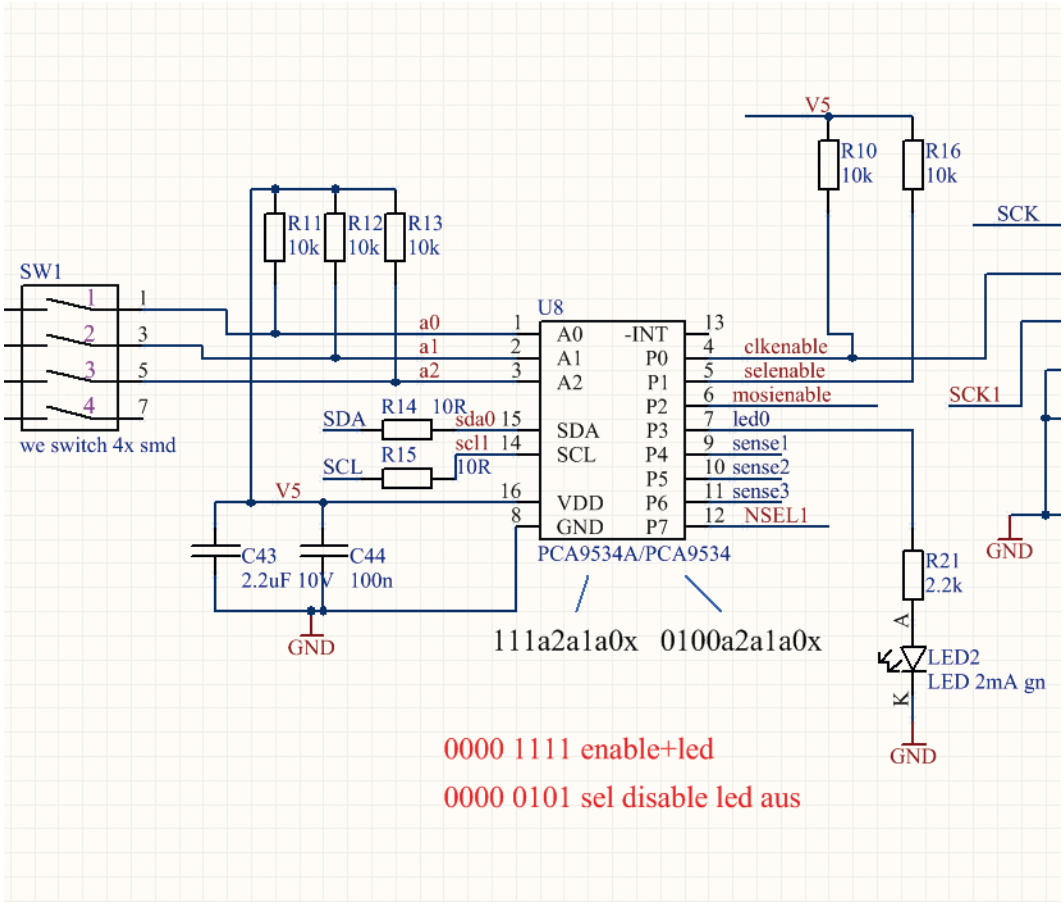
Der Brick verwendet einen hochauflösenden A/D Umsetzer (AD7177) mit 4 gleichzeitig nutzbaren Kanälen, die von mehreren Eingängen umgeschaltet werden können (CH0..CH4 - siehe Datenblatt des Herstellers). Paare können beliebig zu differentiellen Messungen eingestellt werden. Der Eingangsspannungsbereich ist +-2.5V gleichspannungsgekoppelt. Die Wandler können bei 5SPS etwa 24.6 rauschfreie Bits liefern, 28 Bits effektiv oder sogar 32 Bits mit Rauschen), bei 10kSPS noch 19.1 Bits. Eingebaut sind auch 50/60 Hz Filter, die Netzspannungsstörungen unterdrücken können mit bis zu 85dB. Damit eignet sich der Baustein zur Erfassung von EKG-Signalen im mV Bereich aber auch EEG Signalen im µV Bereich ohne zusätzlichen Verstärker. Betrieben wird der Baustein am SPI-Bus des Mikrocontrollers, den I2C Bus verwendet der Brick zur Selektion, so dass man auch mehrere dieser Bricks am Bus betreiben kann. In dem Set wird aber nur ein Brick verwendet und die Programmbeispiele zeigen wie man damit umgeht.

Die Kanäle sind immer paarweise auf Klinkenstecker gelegt, also CH0/CH1 und CH2/CH3 sowie CH4 und dort noch GND auf den freien Kontakt. GND oder Masse ist eine eigene Bezugsmasse, zur Erhöhung des Messgenauigkeit sollte man nicht mit dem 0V GND zusammenschalten, würde aber nichts passieren.

Linke Buchse: CH1 auf dem äußeren Kontakt, CH0 auf dem mittleren und GND auf dem Ring des Klinkensteckers.

Mittlere Buchse: CH3 auf dem äußeren Kontakt, CH2 auf dem mittleren und GND Ring.

Rechte Buchse wird als Bezug verwendet: CH4 auf dem äusseren Kontakt. GND auf dem mittleren über 10kOhm auf Analog Masse und der äußere Ring auf direkt.



Ausschnitt der Ansteuerung. Über den I2C Bus erfolgt die Selektion des A/D Umsetzers. Die Schalter sind normalerweise alle on. Also I/O Adresse 1110000 oder 0100000 je nach Baustein.

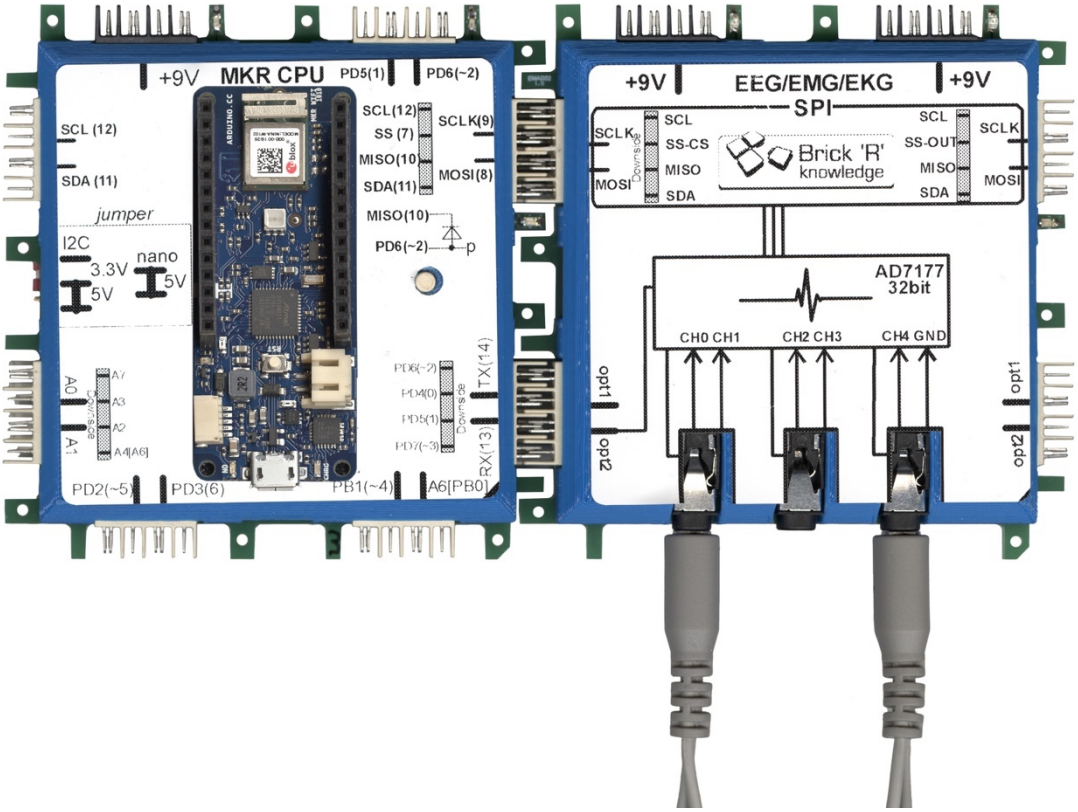
Die drei Sense Eingänge können erkennen, ob eine Klinke eingesteckt wurde. Die LED dient zum Test. Die unteren drei Bits aktivieren den Baustein und schalten ihn zum SPI-Bus durch. NSEL ist eine zusätzliche Selektion für MISO, um die Leitung freizugeben und auf den Bus aufzuschalten.

Drei von diesen Kabeln sind im Set vorhanden. Dabei sind immer zwei Kabel an einem Stecker befestigt. Es handelt sich um geschirmte Kabel, also bitte vorsichtig damit umgehen. Die Clips am Kabelende sind für handelsübliche Kontaktpads für den Anschluss an die Haut gedacht.

Kontakt-Pad (abgebildet ist hier beispielhaft das Panasonic EW0603P).

Im Lieferumfang sind keine Kontakt-Pads enthalten. Es werden insgesamt maximal 6 Pads gleichzeitig benötigt.

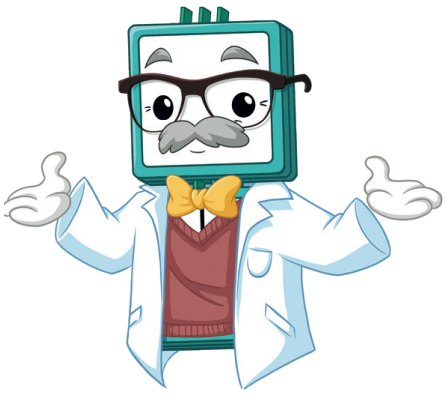
4.7 Verbindung des Arduino-MKR-Adapter-Bricks mit dem EEG/EMG/EKG Brick



Der ALLNET Brick'R'knowledge EKG Brick wird auf der rechten Seite des ALLNET Brick'R'knowledge MKR Brick angeschlossen. Die Elektroden für die Messungen werden am EKG Brick angeschlossen. Je nach durchzuführendem Versuch werden dabei nicht alle Elektroden benötigt.

Oben müssen die 9V Versorgungen zusammengeschaltet werden.

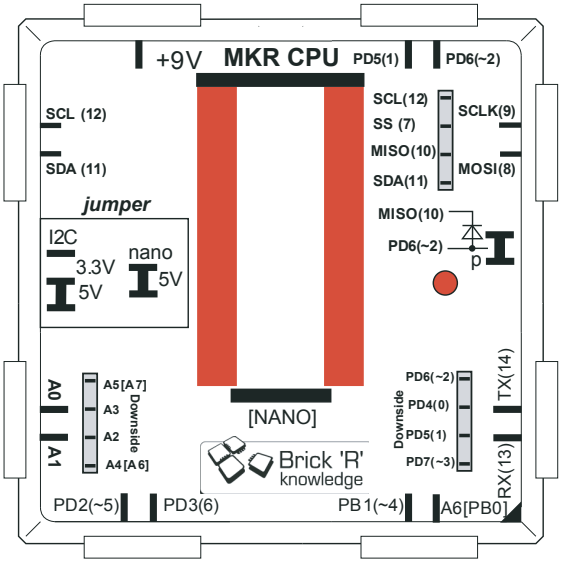
Achtung: PD5,PD6 nicht mit 9V verbinden. Am EEG Brick reicht einer der beiden 9V Eingänge.

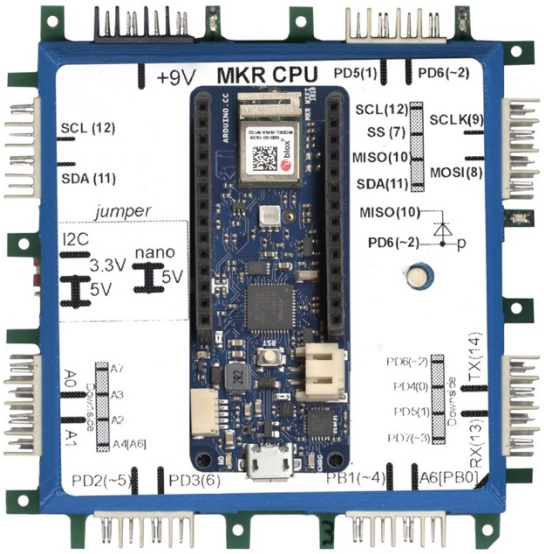


5. Bio Feedback Projekte - Einstieg in die Arduino Welt

5.1 Das Herzstück: Arduino MKR WiFi 1010 Board

Arduino MKR: Dies ist das Herz unseres Sets. 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 Homepage entsprechend runterladen kann. Achtung. Die Eingänge des Arduino sollten nie in direkten Kontakt mit der 9V Quelle kommen, da sie trotz Schutzschaltungen auf der Platine nur für 3,3V bzw. 5V ausgelegt sind.





5.1.1 Spezifikation des MKR WiFi 1010 Boards

Microcontroller	SAMD21 Cortex-M0+ 32bit Low Power ARM MCU
Board Power Supply (USB/VIN)	5V
Circuit Operating Voltage	3.3V
Digital I/O Pins	8
PWM Pins	12 (0, 1, 2, 3, 4, 5, 6, 7, 8, 10, A3 - or 18 -, A4 -or 19)
UART	1
SPI	1
I2C	1
I2S	1
Connectivity	WiFi
Analog Input Pins	7 (ADC 8/10/12 bit)
Analog Output Pins	1 (DAC 10 bit)
External Interrupts	8 (0, 1, 4, 5, 6, 7, 8, A1 -or 16-, A2 - or 17)
DC Current per I/O Pin	7 mA
Flash Memory	256 KB
SRAM	32 KB
Clock Speed	32.768 kHz (RTC), 48 MHz

5.1.2 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 zuverlässig auswerten kannst. Die Befehle pinMode(GPIOx,INPUT_PULLUP) bzw. pinMode (GPIOx,INPUT_PULLDOWN) haben in unserem Fall keine Auswirkung auf die GPIO-Pins, da die Pullup-Widerstände immer aktiv sind. Nach dem Einschalten der Versorgung werden als Eingang konfigurierte GPIOs sofort auf High-Pegel gelegt.

5.2 Die Programmierung: Arduino IDE

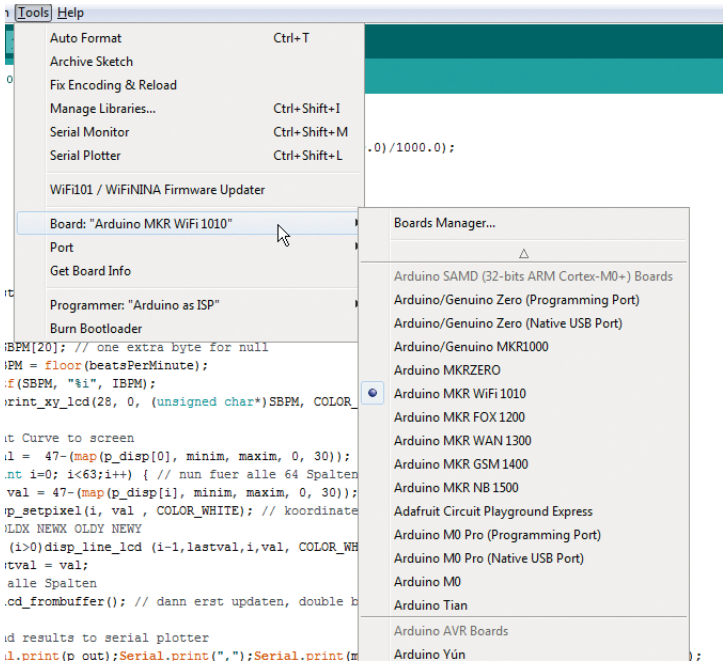
Um den MKR-Brick zu programmieren verwenden wir die Arduino-Entwicklungsumgebung – auch Arduino IDE (IDE steht für Integrierte Entwicklungsumgebung) genannt. Viele kennen die kostenlose Programmier-Software vielleicht schon aus der Arduino-Welt. Es gibt im Internet zahlreiche Arduino-basierende Projekte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



5.2.1 Bibliotheken installieren

Zur Vereinfachung der Programmierung verwenden wir in unseren Beispielprogrammen sog. Bibliotheken oder englisch auch "Libraries" genannt. Diese Code-Sammlungen enthalten z.B. umfangreiche Tabellen, mit welchen z.B. Zeichensätze definiert 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.

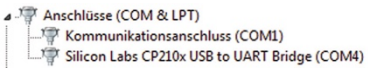
Hinweise zur Installation zusätzlicher Arduino-Bibliotheken findest du auch unter:

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

5.2.2 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.

5.2.3 Serieller Monitor

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

5.2.4 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.

5.3 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.

5.3.1 Verbindungen herstellen

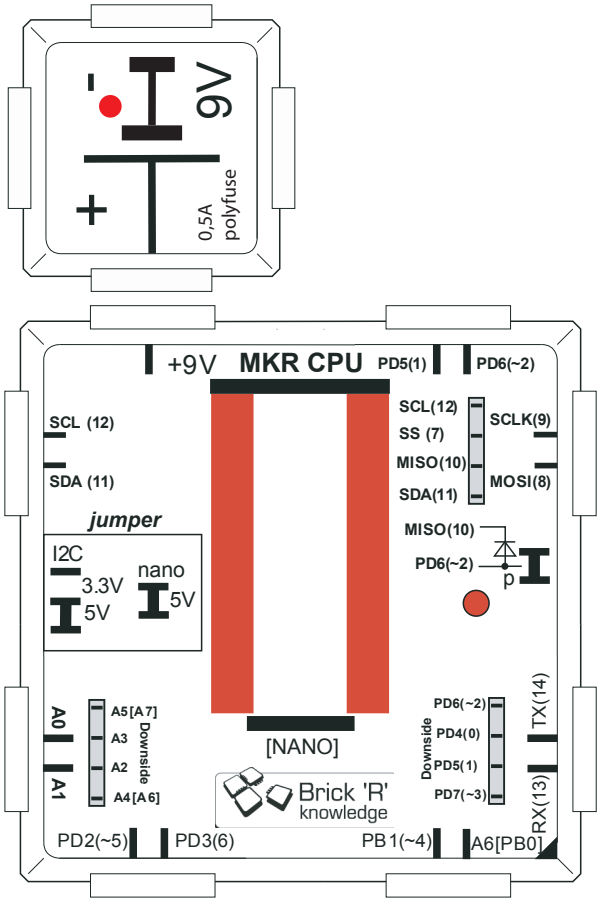
Verbinde zunächst den MKR-Brick mit dem beiliegenden 9V-Netzteil an dem dafür vorgesehenen, oberen Anschluss.

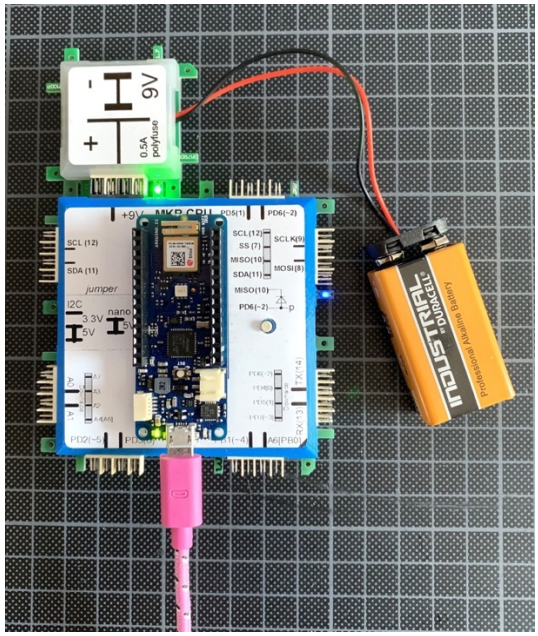
Achtung: Das 9 V-Netzteil niemals an die anderen Kontakte des MKR-Bricks anschließen, dies könnte den Baustein zerstören!

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

5.3.2 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.

5.3.3 Die Setup-Routine „void setup()“

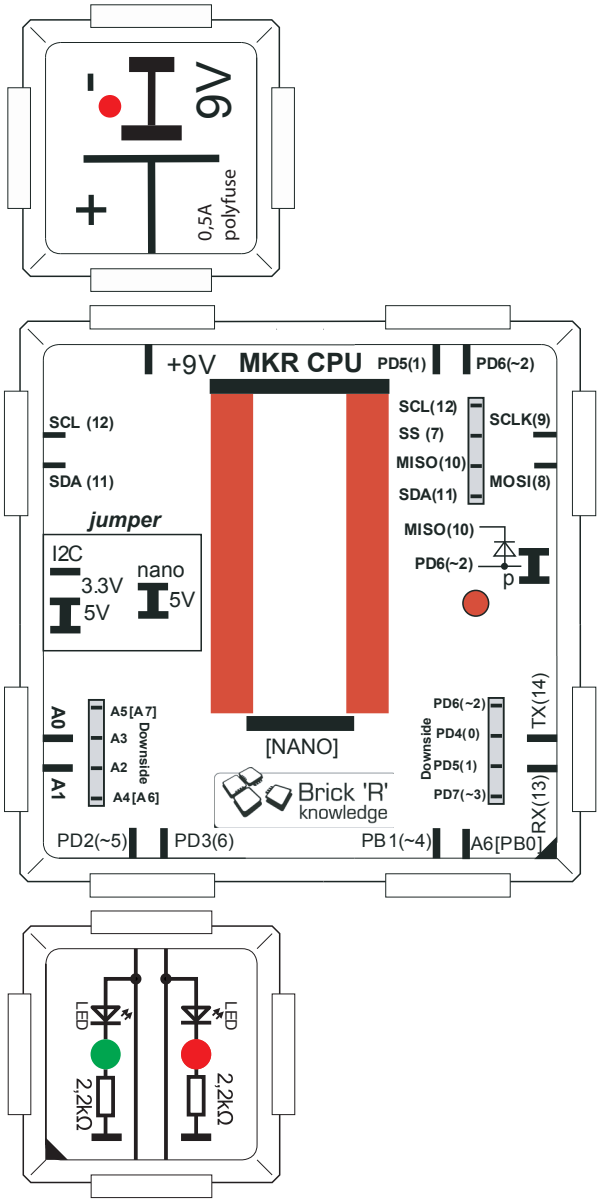
Zunächst kommt der Abschnitt void setup(){...}. Alle Befehle innerhalb dieser geschweiften Klammern werden beim Start des Programmes genau einmal ausgeführt. In unserem Beispiel wird hier die Richtung der GPIO-Pins definiert, also Eingang (Mode: Input) oder wie in unserem Fall: zwei Ausgänge (Mode: Output).

5.3.4 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.

5.4 Doppelte LED



Nun schließen wir einen unserer Bricks direkt an den MKR an. Dazu gibt es im Set 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
}
```

5.5 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.

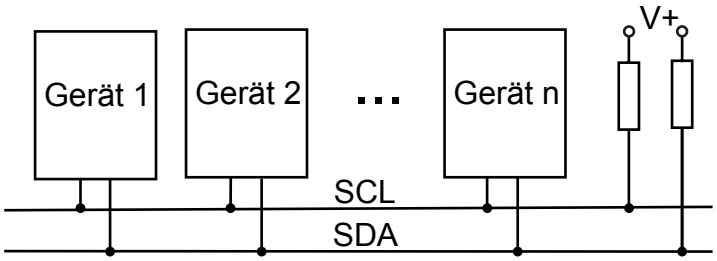


Abbildung: I2C-Busstruktur

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.

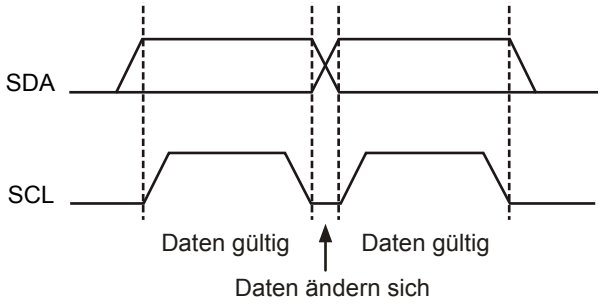
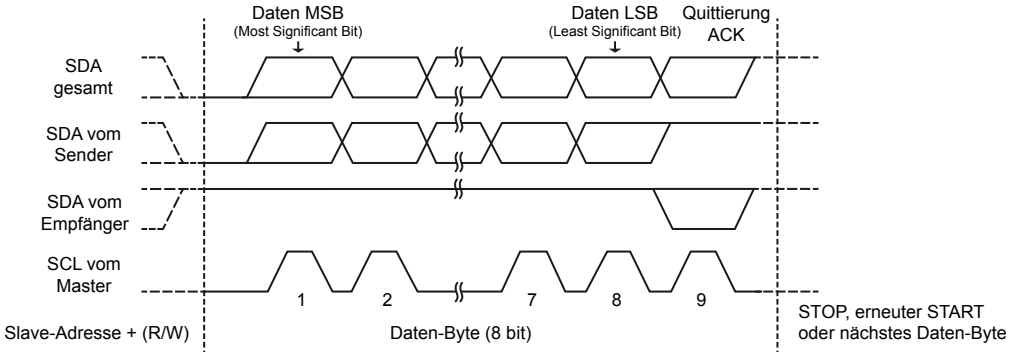


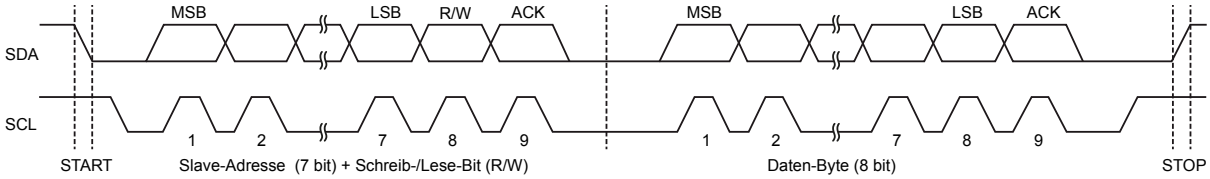
Abbildung: Gültige Daten am I2C-Bus

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

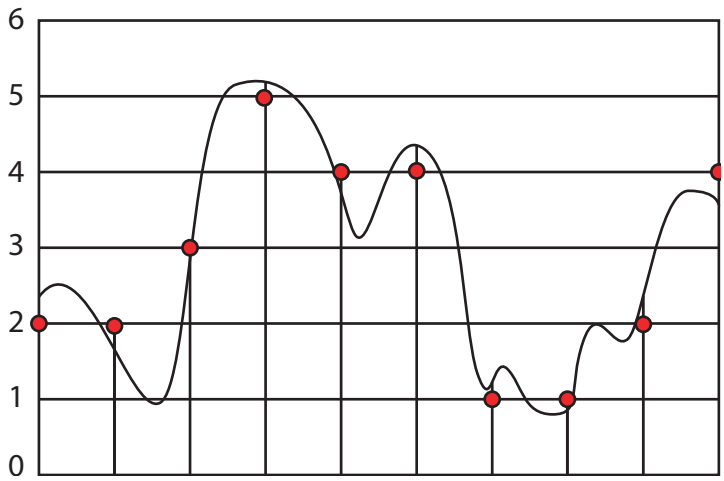
In der oberen Abbildung sieht man den gesamten Übertragungszyklus. Zunächst wird ein Paket mit der Adresse gesendet. Die Adresse besteht aus 7 Bits, ergänzt um ein weiteres, R/W (Read/Write) benanntes Bit. Alle Busteilnehmer vergleichen die ausgesendete Adresse mit ihrer eigenen. Bei Übereinstimmung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

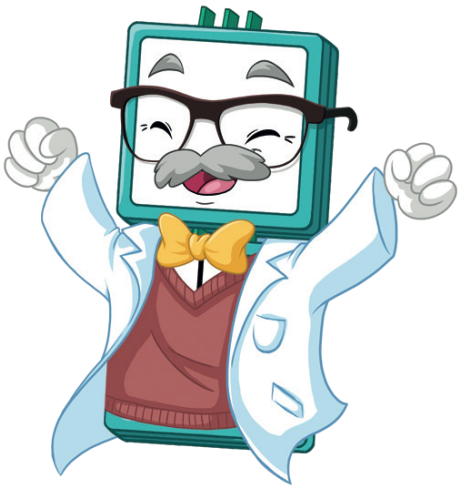
In Abhängigkeit vom R/W-Bit weiß der adressierte Slave, ob er nun einen Sende- oder Empfangszyklus starten soll. Danach kann der eigentliche Datentransfer beginnen. Als Abschluss wird ein Stop-Zyklus eingeleitet. Dazu wird der Takt auf high gesetzt und dann die SDA-Leitung freigegeben. Die Leitungen 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.

5.6 Analog-Digital Umsetzer

5.6.1 A/D Umsetzer - prinzipieller Aufbau





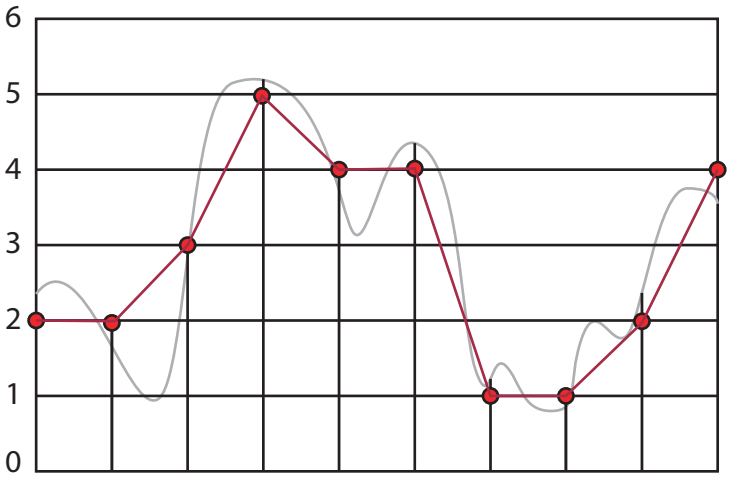
Hier haben wir den zeitliche Verlauf eines Spannungssignals aufgezeichnet (hier kann man sich in der y-Achse die Spannung in V aufgetragen denken und in x-Richtung die Zeit z.B. in Sekunden).

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.

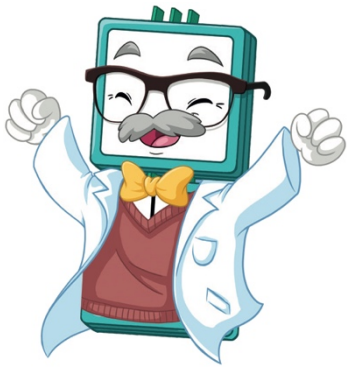
Bei uns = 20*LOG(1024) = 60.2 dB. Dies ist ein ganz guter Wert. Das menschliche Gehört hat aber eine höhere Dynamik, daher muss man Audiosignale für eine HIFI Qualität auch mit mehr Bits digitalisieren, zum Beispiel bei 24 Bit ergeben sich: 20 * LOG (2Hoch24) = 20 * LOG (16777216)=144.5 dB.

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

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

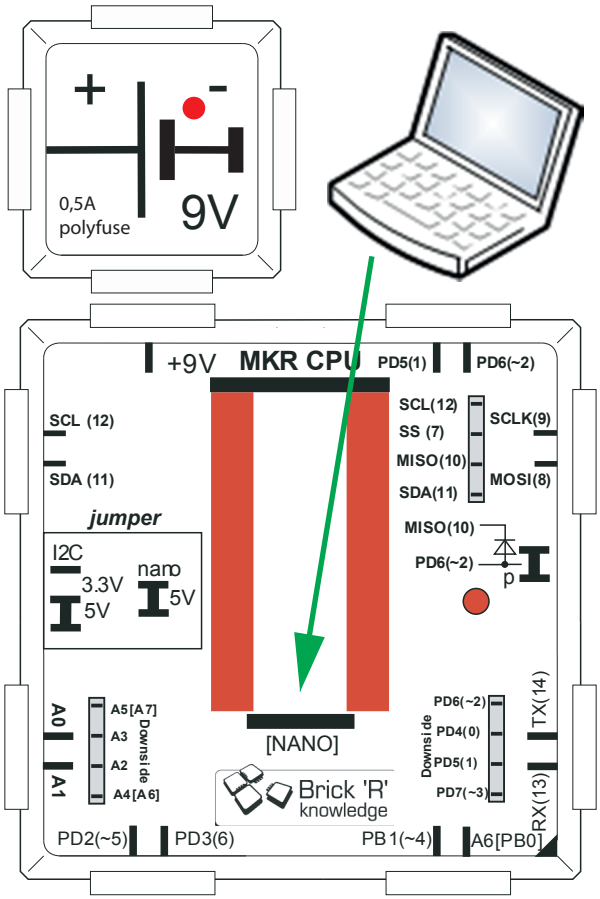
es unterschiedliche Verfahren, die den Rahmen hier schnell sprengen würden. Aber ein paar Stichworte zum Suchen seien hier genannt: Delta-Sigma, Sukzessive Approximation (SAR): Parallelwandler, Sägezahnverfahren, Mehrrampenverfahren, um nur einige zu nennen.

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.



5.6.2 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.

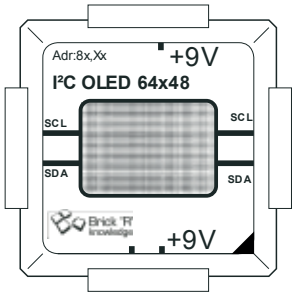
Alternativ dazu kann man auch ein separates Terminal-Programm benutzen.

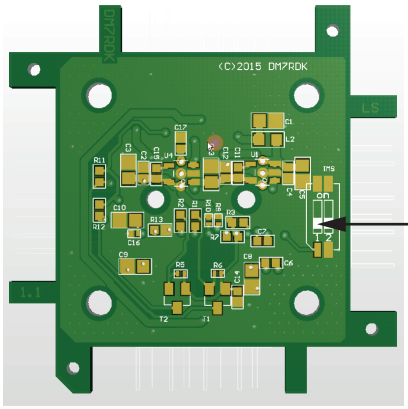
```
// the setup function runs once when you press reset or power the board
void setup() {
 Serial.begin(115200);
 delay(5000); // Wait 5 Seconds until user opens the Terminal
 Serial.println("");
 Serial.println("");
 Serial.println("INIT");
  for (int x = 0; x < 15; x++){
    pinMode(x, OUTPUT);
    }
  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
}
```

5.7. OLED

5.7.1. 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. Unser Set enthält 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 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   */
};
```

5.7.2 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:

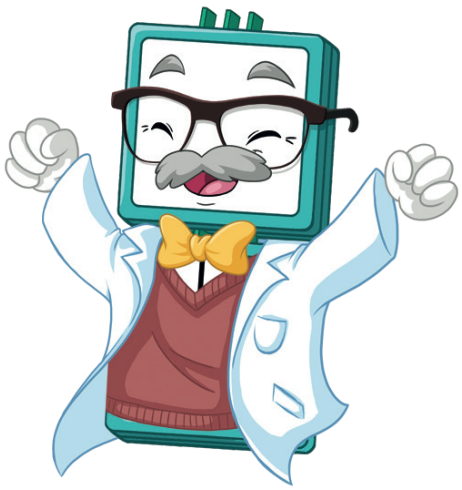
```
#define i2coledssd (0x7A>>1)  // Default
```

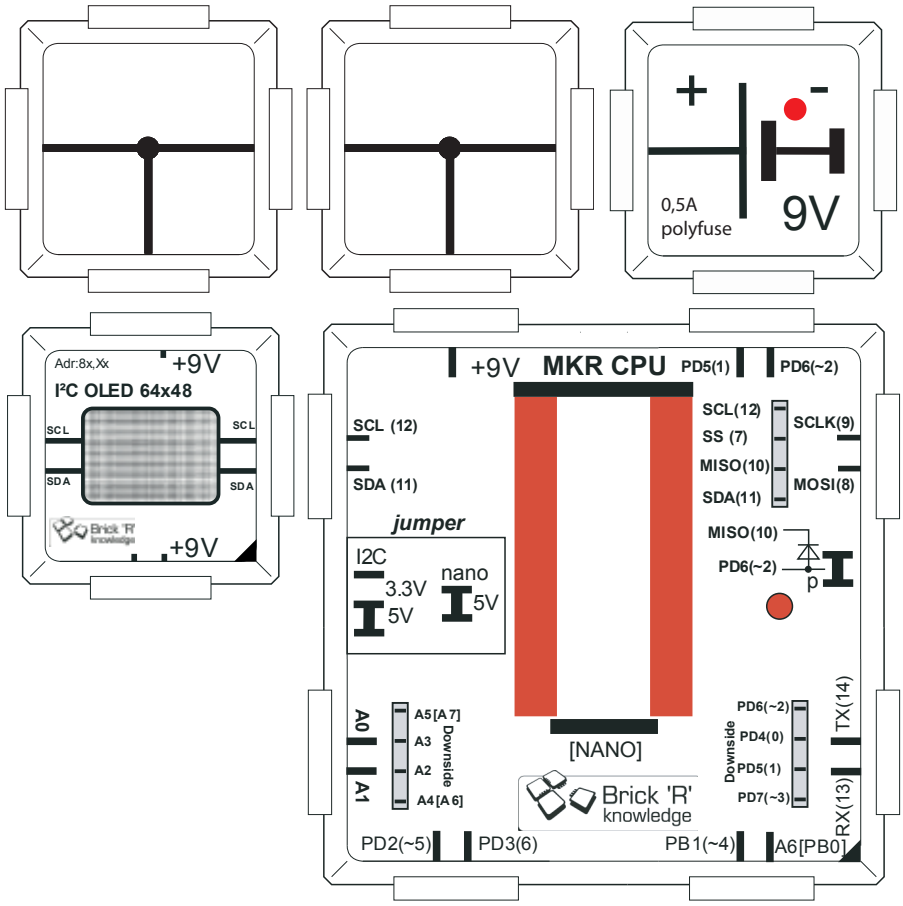
oder

```
#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)
```

5.7.3. 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.

5.7.4. 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).

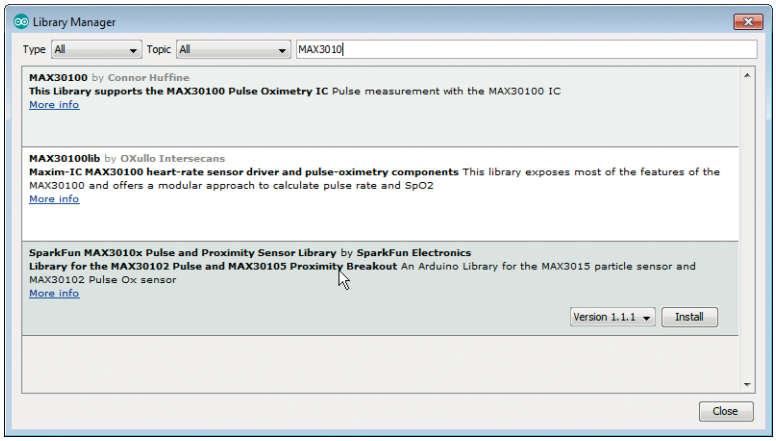
```
// DE_28 OLED Beispiele - Zeichensatz
#include <Wire.h>
#include <avr/pgmspace.h>
// Hier ggf Adresse anpassen 78 oder 7A je nach Schalter
#define i2coledssd (0x7A>>1)  // default ist 7A
// -------------------------OLED ---------------------------------------------
// An dieser Stelle steht der Code 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.

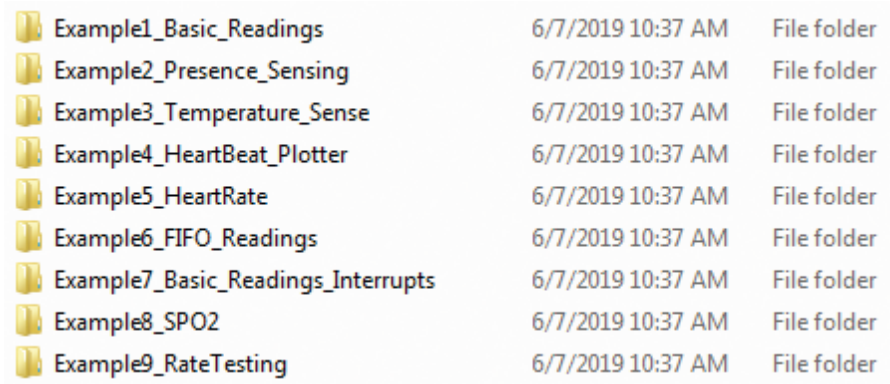
6. Puls Oxymeter Brick

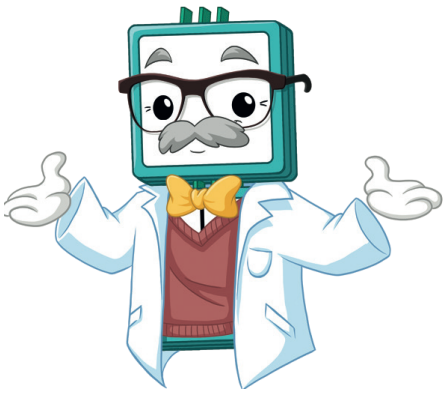
Zur Verwendung der nun folgenden Programme muss man die Bibliothek MAX3010x laden (die ist kompatibel zum von uns verendeten MAX 30102).

Dies geschieht über den Bibliotheksmanager in gewohnter Weise:



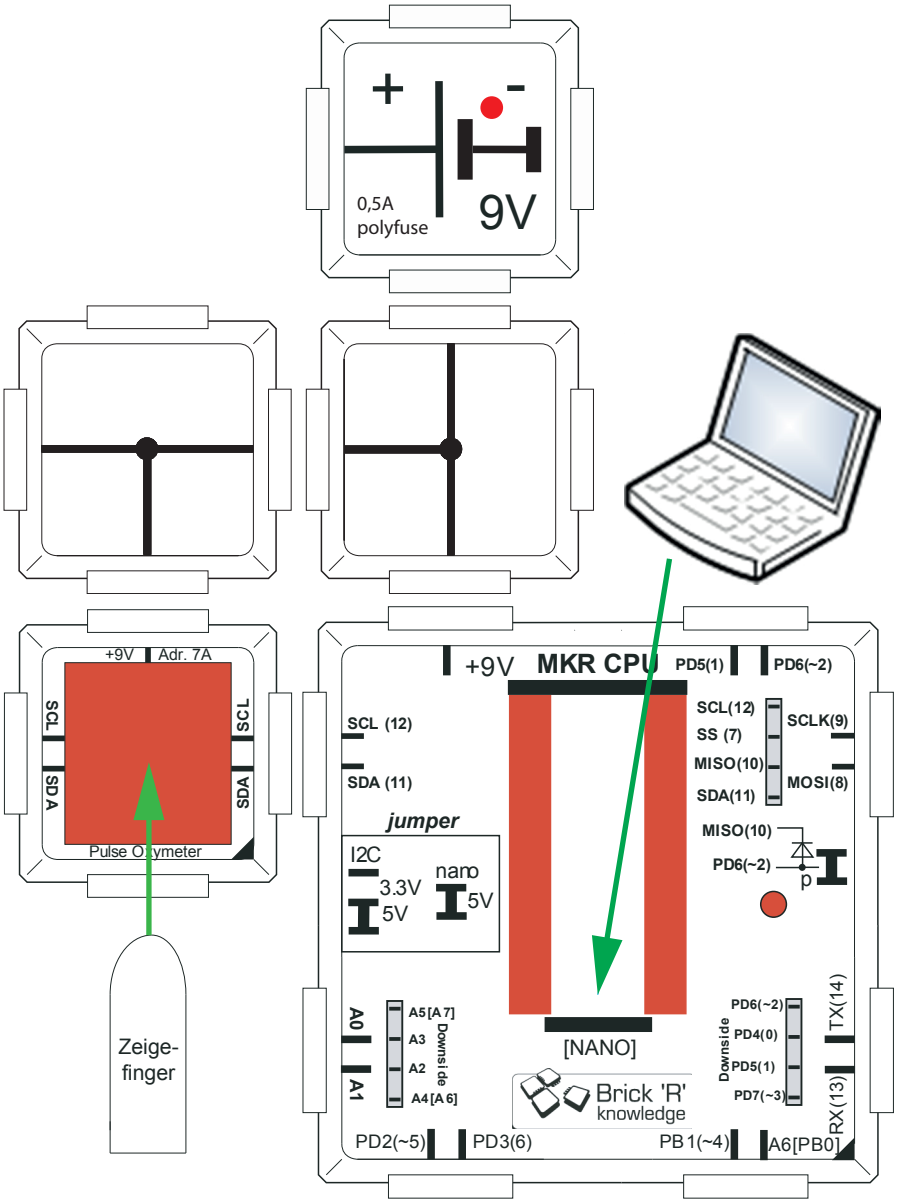
Mit der Bibliothek zum MAX 30102/5 werden auch zahlreiche Beispielprogramme installiert:





6.1 Puls Oxymeter Brick - Erster Test

Das erste prüft einfach ob alle Werte richtig ankommen. Wichtig Baurate auf 115200 einstellen am Terminal Programm. Dann übersetzen und Hochladen, danach Terminalprogramm starten. Wenn man den Finger auflegt kann man die ausgelesenen Werte für Rot und IR sowie Grün (optional) ansehen. Der MAX 30105 hat noch einen Grün-Sensor, der MAX 30102 nicht. Der 30105 kann zusätzlich Rauch erkennen. Bei uns ist aber der 30102 verbaut, daher ist die G Ausgabe nicht von Bedeutung da die entsprechende LED fehlt.

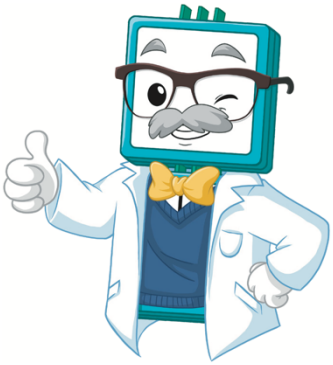



```
/*
  MAX30105 Breakout: Output all the raw Red/IR/Green readings
  By: Nathan Seidle @ SparkFun Electronics
  Date: October 2nd, 2016
  https://github.com/sparkfun/MAX30105_Breakout
  Outputs all Red/IR/Green values.
  Hardware Connections (Breakoutboard to Arduino):
  -5V = 5V (3.3V is allowed)
  -GND = GND
  -SDA = A4 (or SDA)
  -SCL = A5 (or SCL)
```

```
  -INT = Not connected
  The MAX30105 Breakout can handle 5V or 3.3V I2C logic. We recommend powering the
board with 5V
  but it will also run at 3.3V.
  This code is released under the [MIT License](http://opensource.org/licenses/MIT).
*/
#include <Wire.h>
#include "MAX30105.h"
MAX30105 particleSensor;
#define debug Serial //Uncomment this line if you're using an Uno or ESP
//#define debug SerialUSB //Uncomment this line if you're using a SAMD21
void setup()
{
  debug.begin(9600);
  debug.println("MAX30105 Basic Readings Example");
  // Initialize sensor
  if (particleSensor.begin() == false)
  {
    debug.println("MAX30105 was not found. Please check wiring/power. ");
while (1); }
  particleSensor.setup(); //Configure sensor. Use 6.4mA for LED drive
}
void loop() {
  debug.print(" R[");
  debug.print(particleSensor.getRed());
  debug.print("] IR[");
  debug.print(particleSensor.getIR());
  debug.print("] G[");
  debug.print(particleSensor.getGreen());
  debug.print("]");
  debug.println();
}
```

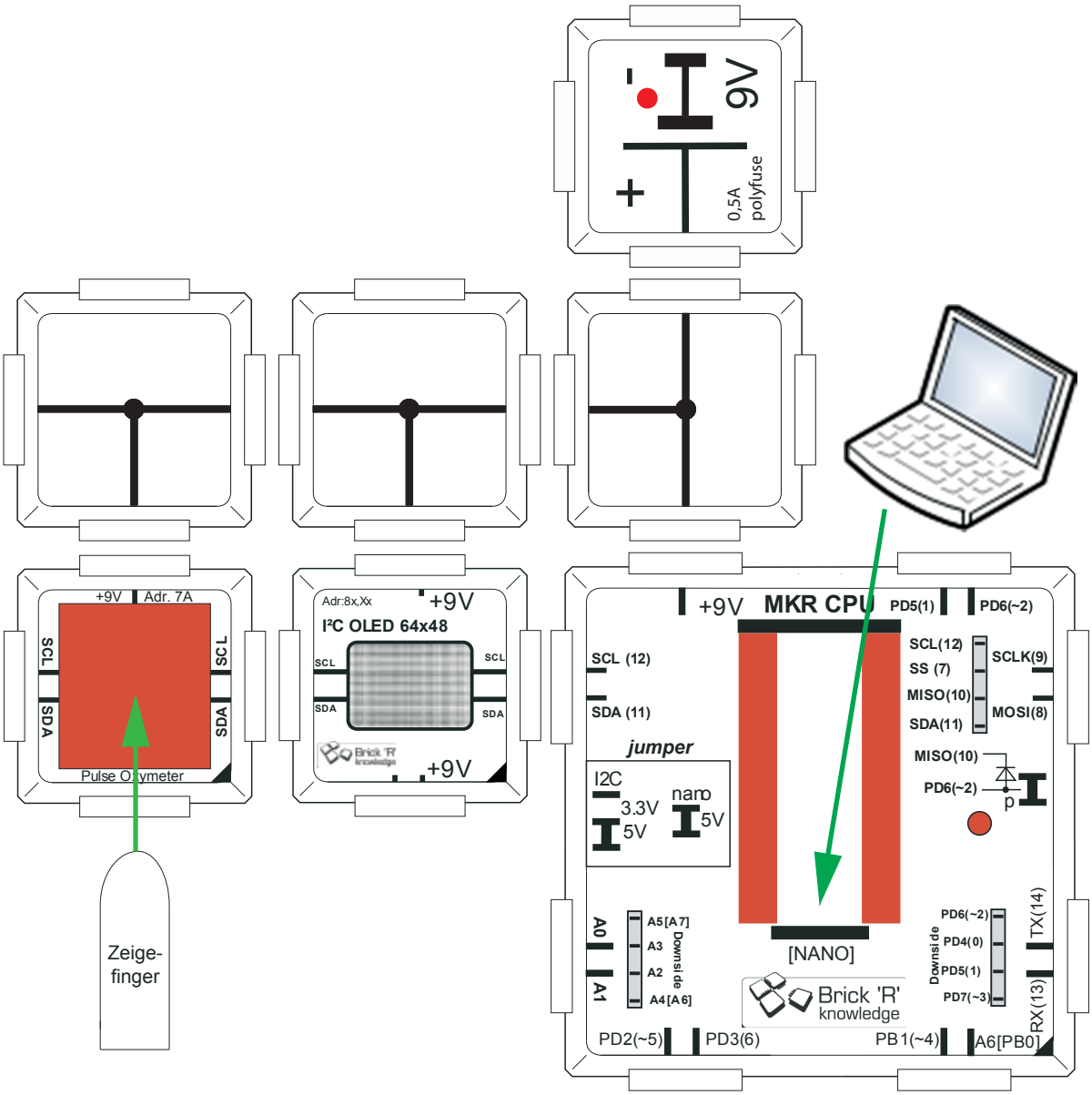
```
12:02:45.829 ->  R[401] IR[316] G[9]
12:02:45.897 ->  R[424] IR[295] G[12]
12:02:45.931 ->  R[434] IR[332] G[8]
12:02:45.999 ->  R[438] IR[348] G[25]
12:02:46.067 ->  R[435] IR[352] G[28]
12:02:46.135 ->  R[432] IR[345] G[41]
12:02:46.169 ->  R[428] IR[352] G[29]
```

Was passiert? Die Messwerte werden auf dem Terminal ausgegeben.



6.2 Puls Oxymeter Brick als Näherungsschalter

Hier wird der Sensor als Näherungsschalter verwendet. Dazu werden die gelesenen werte dauern verglichen und bei einer bestimmten Schwelle wird die Näherung ausgelöst. Dabei reicht eine Annäherung von mehr als 30 cm! Dabei bitte den Finger und die Hand über den Sensor halten.




```
#include <Wire.h>
#include "MAX30105.h"
MAX30105 particleSensor;
long samplesTaken = 0; //Counter for calculating the Hz or read rate
long unblockedValue; //Average IR at power up
long startTime; //Used to calculate measurement rate
void setup()
{
 Serial.begin(115200);
  Serial.println("MAX30105 Presence Sensing Example");
  // Initialize sensor
  if (particleSensor.begin(Wire, I2C_SPEED_FAST) == false) //Use default I2C port, 400kHz
speed
  {
    Serial.println("MAX30105 was not found. Please check wiring/power. ");
    while (1);
```

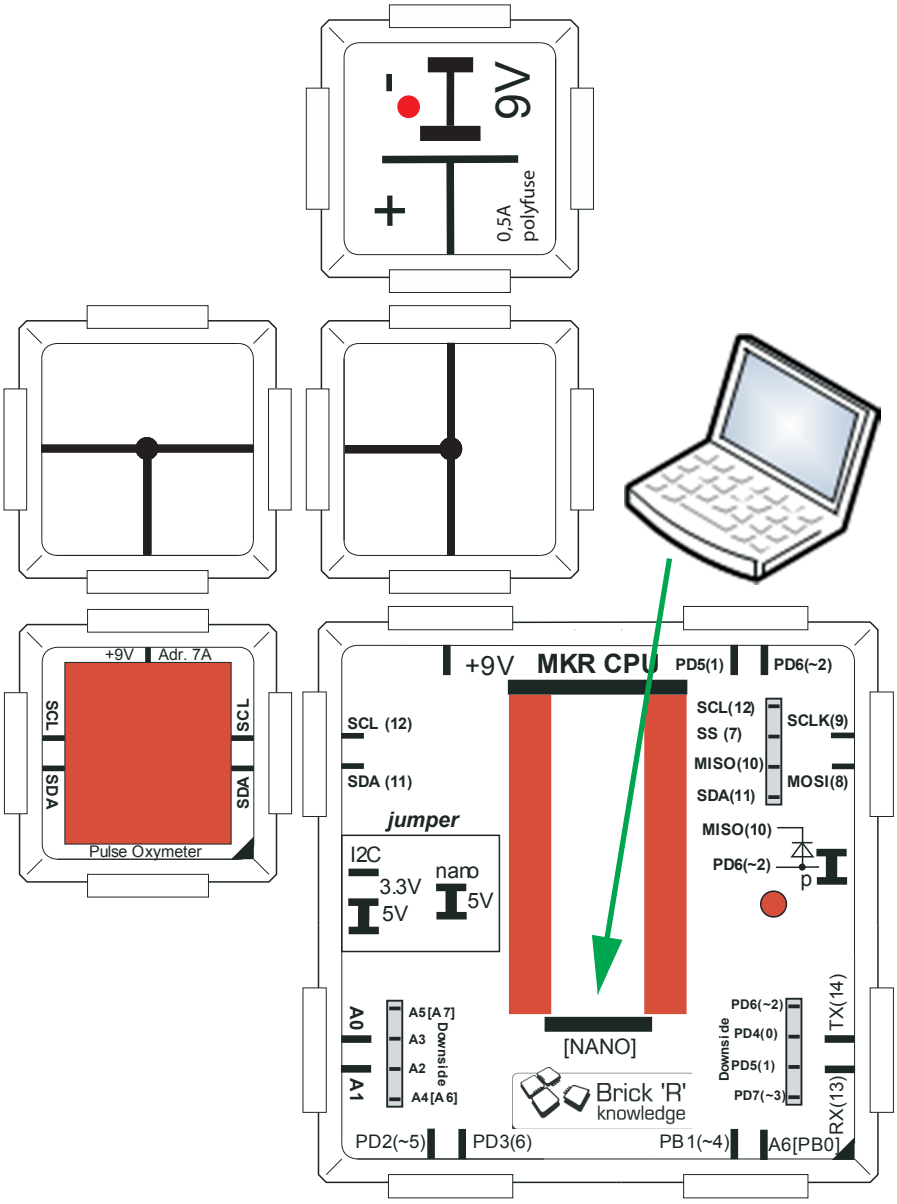
```
}
  //Setup to sense up to 18 inches, max LED brightness
  byte ledBrightness = 0xFF; //Options: 0=Off to 255=50mA
  byte sampleAverage = 4; //Options: 1, 2, 4, 8, 16, 32
  byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green
  int sampleRate = 400; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200
  int pulseWidth = 411; //Options: 69, 118, 215, 411
  int adcRange = 2048; //Options: 2048, 4096, 8192, 16384
  particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulse-
Width,adcRange); //Configure sensor with these settings
  particleSensor.setPulseAmplitudeRed(0); //Turn off Red LED
  particleSensor.setPulseAmplitudeGreen(0); //Turn off Green LED
  //Take an average of IR readings at power up
  unblockedValue = 0;
  for (byte x = 0 ; x < 32 ; x++)
  {
    unblockedValue += particleSensor.getIR(); //Read the IR value
  }
  unblockedValue /= 32;
  startTime = millis();
}
void loop() {
  samplesTaken++;
  Serial.print("IR[");
  Serial.print(particleSensor.getIR());
  Serial.print("] Hz[");
  Serial.print((float)samplesTaken / ((millis() - startTime) / 1000.0), 2);
  Serial.print("]");
  long currentDelta = particleSensor.getIR() - unblockedValue;
  Serial.print(" delta[");
  Serial.print(currentDelta);
  Serial.print("]");
  if (currentDelta > (long)100)
  {
    Serial.print(" Something is there!");
  }
  Serial.println();
  delay(1000);
}
```

```
.2:12:50.658 -> IR[6746] Hz[49.95] delta[95]
.2:12:50.692 -> IR[6750] Hz[49.95] delta[96]
.2:12:50.692 -> IR[6758] Hz[49.95] delta[157] Something is there!
.2:12:50.726 -> IR[6893] Hz[49.95] delta[378] Something is there!
.2:12:50.760 -> IR[7242] Hz[49.95] delta[862] Something is there!
.2:12:50.760 -> IR[7914] Hz[49.95] delta[1766] Something is there!
.2:12:50.794 -> IR[9045] Hz[49.95] delta[3189] Something is there!
.2:12:50.794 -> IR[10778] Hz[49.95] delta[5204] Something is there!
.2:12:50.828 -> IR[13081] Hz[49.95] delta[7727] Something is there!
```

Was passiert? Die Messwerte werden auf dem Terminal ausgegeben und bei einer Näherung der Hand in Richtung Sensor wird „Something is there!“ ausgegeben.

6.3 Puls Oxymeter Brick als Temperatursensor

Der Sensor hat auch einen eingebauten Temperatursensor, den wir in diesem Beispiel auslesen.

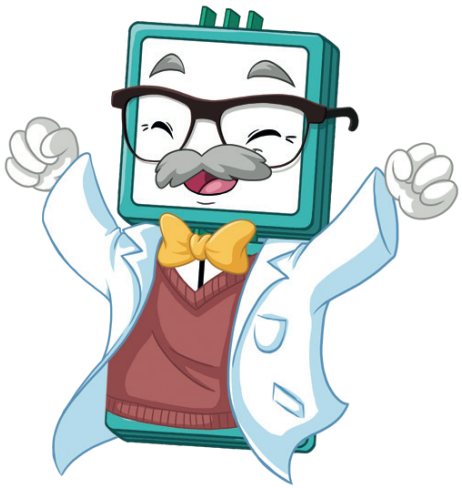



```
#include <Wire.h>
#include "MAX30105.h"  //Get it here: http://librarymanager/All#SparkFun_MAX30105
MAX30105 particleSensor;
void setup()
{
  Serial.begin(11520000);
  Serial.println("Initializing...");
  // Initialize sensor
  if (particleSensor.begin(Wire, I2C_SPEED_FAST) == false) //Use default I2C port, 400kHz
speed
{
      Serial.println("MAX30105 was not found. Please check wiring/power. ");
while (1); }
  //The LEDs are very low power and won't affect the temp reading much but
  //you may want to turn off the LEDs to avoid any local heating
  particleSensor.setup(0); //Configure sensor. Turn off LEDs
  //particleSensor.setup(); //Configure sensor. Use 25mA for LED drive
  particleSensor.enableDIETEMPRDY(); //Enable the temp ready interrupt. This is required.
}
```

```
void loop() {
  float temperature = particleSensor.readTemperature();
  Serial.print("temperatureC=");
  Serial.print(temperature, 4);
  float temperatureF = particleSensor.readTemperatureF(); //Because I am a bad global citizen
  Serial.print(" temperatureF=");
  Serial.print(temperatureF, 4);
  Serial.println();
  delay(1000);

}
```

```
12:17:53.236 -> temperatureC=30.3125 temperatureF=86.4500
12:17:53.304 -> temperatureC=30.3125 temperatureF=86.3375
12:17:53.338 -> temperatureC=30.3125 temperatureF=86.4500
12:17:53.406 -> temperatureC=30.2500 temperatureF=86.5625
12:17:53.474 -> temperatureC=30.3125 temperatureF=86.4500
12:17:53.508 -> temperatureC=30.3750 temperatureF=86.6750
```

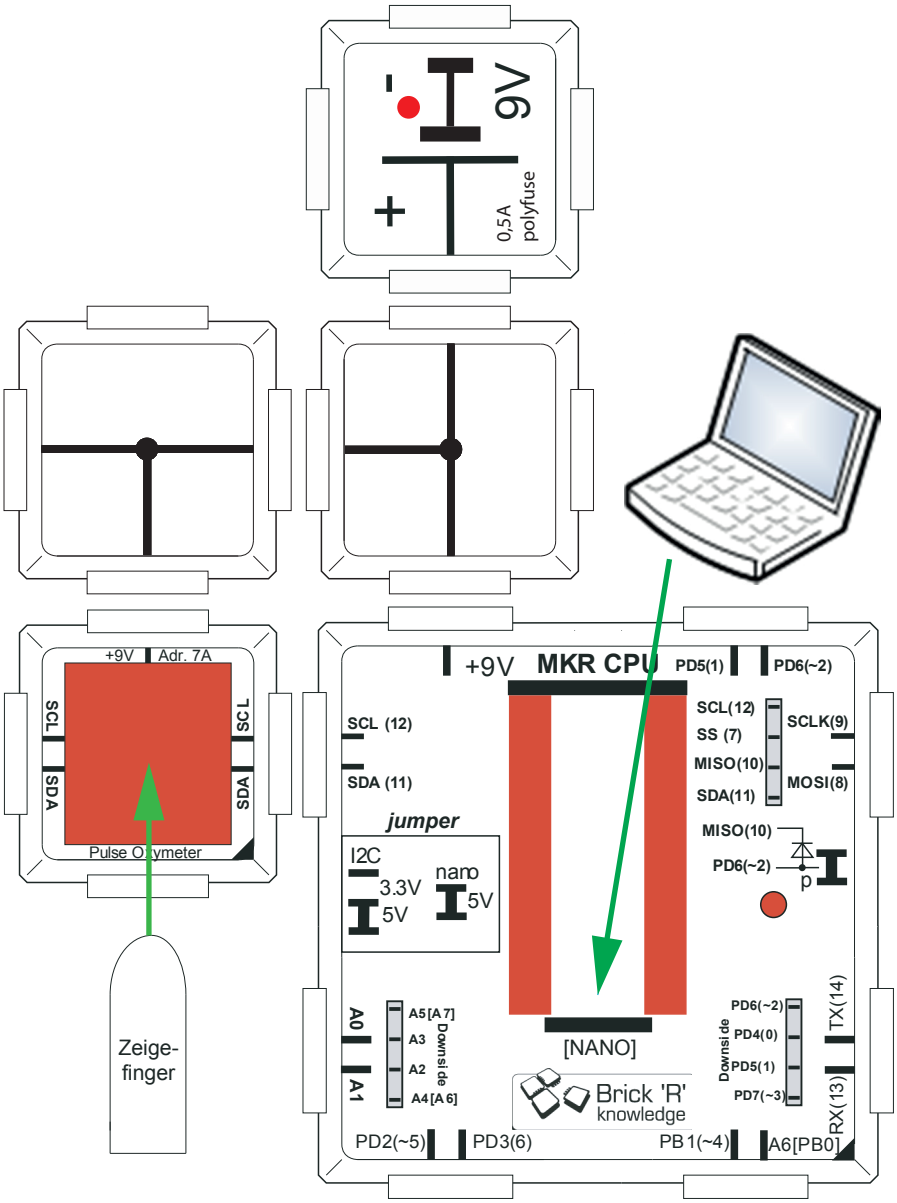
Was passiert? Die Temperatur wird in Grad Celsius und Fahrenheit auf dem Terminal ausgegeben.



6.4 Puls Oxymeter Brick - Einfache Pulsausgabe + über Plotterfunktion

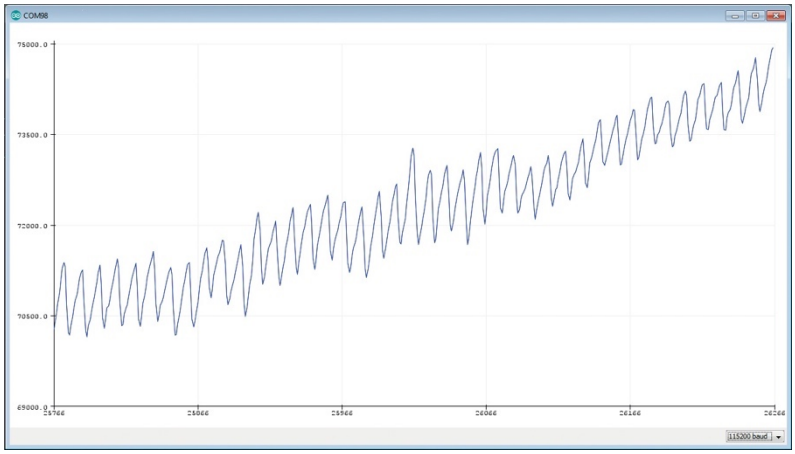
Eine ganz einfache Methode den Herzschlag zu Plotten ist es die IR Messung direkt als Wert auszugeben. Dazu das Terminalprogramm schließen und den seriellen Plotter im Arduino-Programm aktivieren (Baudrate checken 115200). Man sieht schon die Pulskurve. Wichtig man muss dazu ganz lange den Finger extrem ruhig halten.

In einem folgenden Abschnitt zeigen wir wie man mit Hilfe eines sogenannten Digitalfilters (Hochpass) die störende Grundbewegung wegbekommt. Aber wenn man den Finger ruhig hält, kann man den Puls deutlich erkennen. Man muss sich dazu vorstellen, dass die Ausgabe der IR-LED mit der PIN-Dioden Messung extrem empfindlich ist bei 18bit. Reagiert aber auch auf kleinste Bewegungen der Muskeln.



Bitte den Finger so wie hier gezeigt auf den Puls Oxymeter Brick legen, so dass dieser vollständig abgedeckt wird.

```
/*
  MAX30105 Breakout: Output all the raw Red/IR/Green readings
  By: Nathan Seidle @ SparkFun Electronics
  Date: October 2nd, 2016
  https://github.com/sparkfun/MAX30105_Breakout
  Outputs all Red/IR/Green values.
  Hardware Connections (Breakoutboard to Arduino):
  -5V = 5V (3.3V is allowed)
  -GND = GND
  -SDA = A4 (or SDA)
  -SCL = A5 (or SCL)
  -INT = Not connected
  The MAX30105 Breakout can handle 5V or 3.3V I2C logic. We recommend powering the
board with 5V
  but it will also run at 3.3V.
  This code is released under the [MIT License](http://opensource.org/licenses/MIT).
*/
#include <Wire.h>
#include "MAX30105.h"
MAX30105 particleSensor;
#define debug Serial //Uncomment this line if you're using an Uno or ESP
//#define debug SerialUSB //Uncomment this line if you're using a SAMD21
void setup()
{
  debug.begin(9600);
  debug.println("MAX30105 Basic Readings Example");
  // Initialize sensor
  if (particleSensor.begin() == false)
  {
    debug.println("MAX30105 was not found. Please check wiring/power. ");
while (1); }
  particleSensor.setup(); //Configure sensor. Use 6.4mA for LED drive
}
void loop() {
  debug.print(" R[");
  debug.print(particleSensor.getRed());
  debug.print("] IR[");
  debug.print(particleSensor.getIR());
  debug.print("] G[");
  debug.print(particleSensor.getGreen());
  debug.print("]");
  debug.println();
}
```

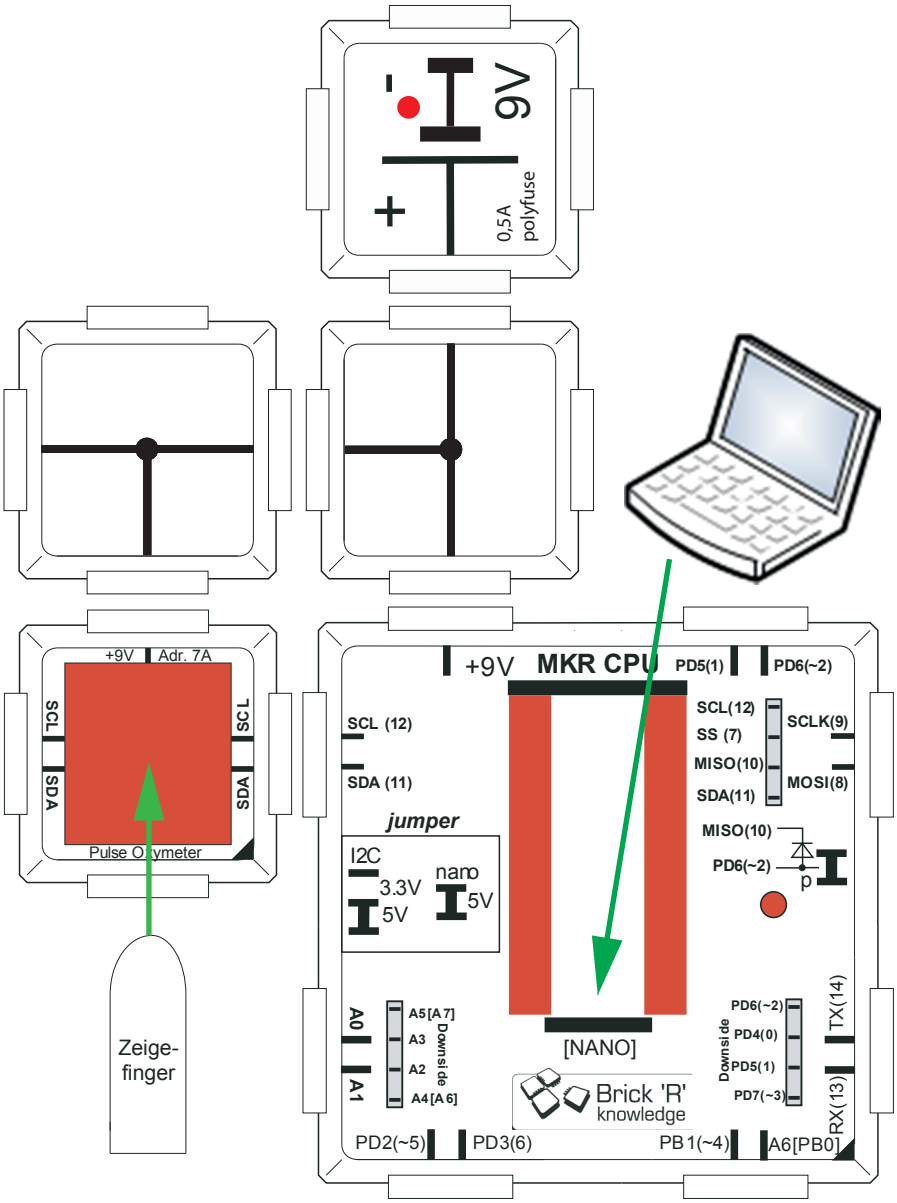


Was passiert? Die Messwerte werden als Grafik auf dem Plotter ausgegeben.

6.5 Puls Oxymeter Brick - Sauerstoffgehalt

Bevor wir aber die Messung des Herzschlags verbessern, noch eine wichtige Anwendung die Messung des Sauerstoffgehalts im Blut. Der Algorithmus ist da von Maxim fertig vorgegeben.

Mit der Bibliothek zum MAX 30102/5 werden zahlreiche Beispielprogramme installiert. Dabei ist Beispiel8 die Sauerstoffmessung auch interessant. In dem Beispiel wird das Ergebnis der Messung auf dem Terminal ausgegeben. Nach dem Starten des Sketches hast du 5 Sekunden Zeit, das Terminal zu öffnen und dann nochmal 5 Sekunden, um den Finger auf den Sensor zu legen.



Bitte den Finger so wie hier gezeigt auf den Puls Oxymeter Brick legen, so dass dieser vollständig abgedeckt wird.

```
#include <Wire.h>
#include "MAX30105.h"
#include "spo2_algorithm.h"

MAX30105 particleSensor;

#define MAX_BRIGHTNESS 255

uint32_t irBuffer[100]; //infrared LED sensor data
uint32_t redBuffer[100];  //red LED sensor data

int32_t bufferLength; //data length
int32_t spo2; //SPO2 value
int8_t validSPO2; //indicator to show if the SPO2 calculation is valid
int32_t heartRate; //heart rate value
int8_t validHeartRate; //indicator to show if the heart rate calculation is valid

byte pulseLED = 11; //Must be on PWM pin
byte readLED = 13; //Blinks with each data read

void setup()
{
  Serial.begin(115200); // initialize serial communication at 115200 bits per second:
  delay(5000);

  pinMode(pulseLED, OUTPUT);
  pinMode(readLED, OUTPUT);

  // Initialize sensor
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) //Use default I2C port, 400kHz speed
  {
    Serial.println("MAX30105 was not found. Please check wiring/power.");
    while (1);
  }

  Serial.println("Attach sensor to finger with rubber band. Waiting 5 secs until start.");
  delay(5000);

  byte ledBrightness = 60; //Options: 0=Off to 255=50mA
  byte sampleAverage = 4; //Options: 1, 2, 4, 8, 16, 32
  byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green
  byte sampleRate = 100; //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

  particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth,
adcRange); //Configure sensor with these settings
}

void loop()
{
  bufferLength = 100; //buffer length of 100 stores 4 seconds of samples running at 25sps

  //read the first 100 samples, and determine the signal range
  for (byte i = 0 ; i < bufferLength ; i++)
  {
    while (particleSensor.available() == false) //do we have new data?
      particleSensor.check(); //Check the sensor for new data

    redBuffer[i] = particleSensor.getRed();
    irBuffer[i] = particleSensor.getIR();
    particleSensor.nextSample(); //We're finished with this sample so move to next sample

    Serial.print(F("red="));
    Serial.print(redBuffer[i], DEC);
    Serial.print(F(", ir="));
    Serial.println(irBuffer[i], DEC);
  }

  //calculate heart rate and SpO2 after first 100 samples (first 4 seconds of samples)
  maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2, &validSPO2,
&heartRate, &validHeartRate);
```

```
  //Continuously taking samples from MAX30102.  Heart rate and SpO2 are calculated every 1
second
  while (1)
  {
    //dumping the first 25 sets of samples in the memory and shift the last 75 sets of samples
to the top
    for (byte i = 25; i < 100; i++)
    {
      redBuffer[i - 25] = redBuffer[i];
      irBuffer[i - 25] = irBuffer[i];
    }

    //take 25 sets of samples before calculating the heart rate.
    for (byte i = 75; i < 100; i++)
    {
      while (particleSensor.available() == false) //do we have new data?
        particleSensor.check(); //Check the sensor for new data

      digitalWrite(readLED, !digitalRead(readLED)); //Blink onboard LED with every data read

      redBuffer[i] = particleSensor.getRed();
      irBuffer[i] = particleSensor.getIR();
      particleSensor.nextSample(); //We're finished with this sample so move to next sample

      //send samples and calculation result to terminal program through UART
      Serial.print(F("red="));
      Serial.print(redBuffer[i], DEC);
      Serial.print(F(", ir="));
      Serial.print(irBuffer[i], DEC);

      Serial.print(F(", HR="));
      Serial.print(heartRate, DEC);

      Serial.print(F(", HRvalid="));
      Serial.print(validHeartRate, DEC);

      Serial.print(F(", SPO2="));
      Serial.print(spo2, DEC);

      Serial.print(F(", SPO2Valid="));
      Serial.println(validSPO2, DEC);
    }

    //After gathering 25 new samples recalculate HR and SP02
    maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2,
&validSPO2, &heartRate, &validHeartRate);
  }
}
```

```
11:54:23.867 -> red=175718, ir=194153, HR=136, HRvalid=1, SPO2=95, SPO2Valid=1
11:54:23.969 -> red=175743, ir=194256, HR=136, HRvalid=1, SPO2=95, SPO2Valid=1
11:54:24.071 -> red=175802, ir=194339, HR=136, HRvalid=1, SPO2=95, SPO2Valid=1
11:54:24.173 -> red=175868, ir=194446, HR=136, HRvalid=1, SPO2=95, SPO2Valid=1
11:54:24.275 -> red=175938, ir=194536, HR=136, HRvalid=1, SPO2=95, SPO2Valid=1
```

```
11:57:06.815 -> red=163340, ir=176056, HR=166, HRvalid=1, SPO2=100, SPO2Valid=1
11:57:06.917 -> red=163316, ir=176177, HR=166, HRvalid=1, SPO2=100, SPO2Valid=1
11:57:07.019 -> red=163287, ir=176321, HR=166, HRvalid=1, SPO2=100, SPO2Valid=1
11:57:07.121 -> red=163300, ir=176558, HR=166, HRvalid=1, SPO2=100, SPO2Valid=1
11:57:07.189 -> red=163376, ir=176736, HR=166, HRvalid=1, SPO2=100, SPO2Valid=1
11:57:07.326 -> red=163481, ir=176918, HR=166, HRvalid=1, SPO2=100, SPO2Valid=1
11:57:07.394 -> red=163354, ir=175686, HR=166, HRvalid=1, SPO2=100, SPO2Valid=1
11:57:07.530 -> red=162962, ir=175243, HR=166, HRvalid=1, SPO2=100, SPO2Valid=1
11:57:07.598 -> red=162983, ir=175538, HR=166, HRvalid=1, SPO2=100, SPO2Valid=1
11:57:07.734 -> red=163093, ir=175805, HR=166, HRvalid=1, SPO2=100, SPO2Valid=1
```

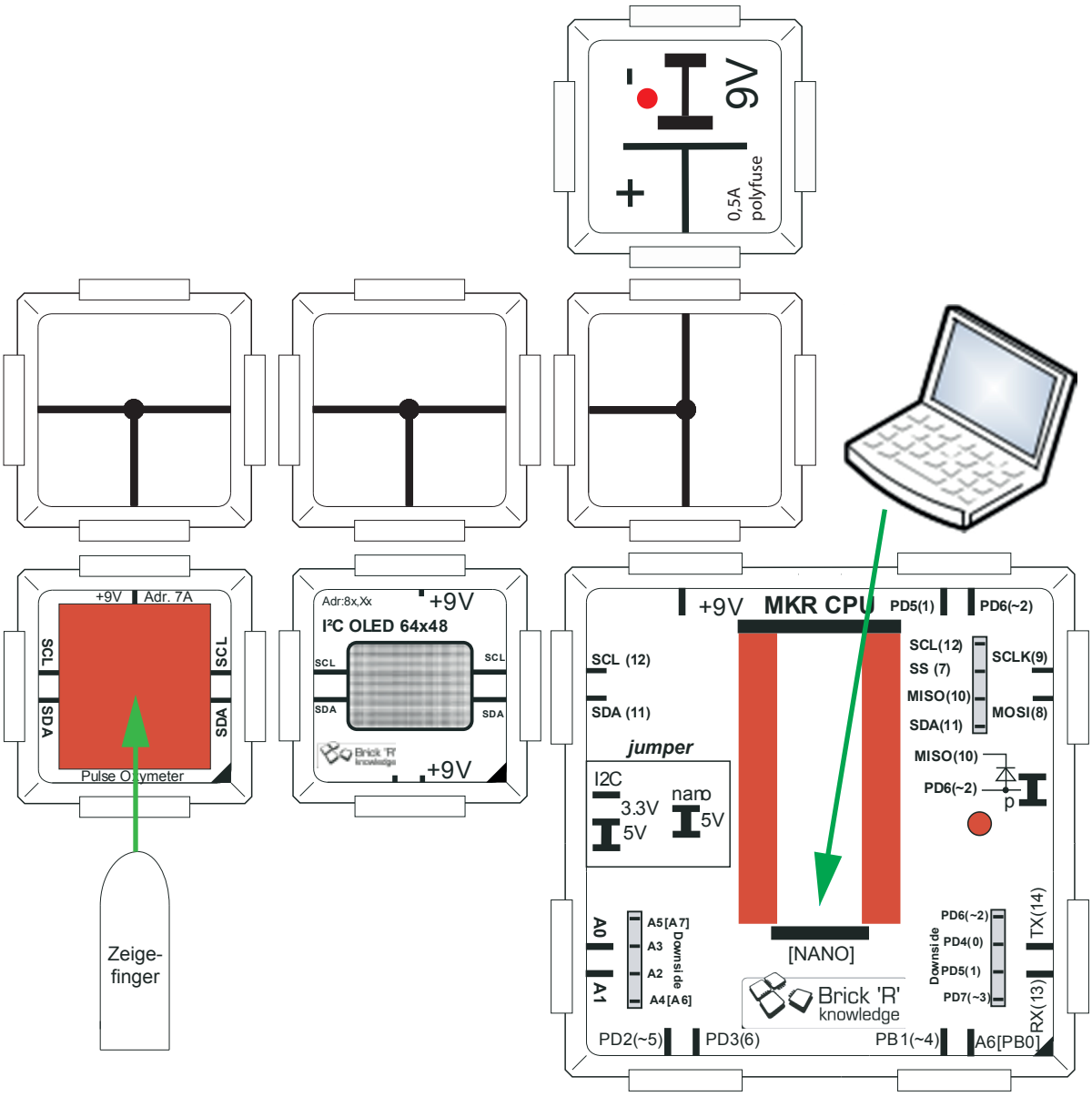
Was passiert? Die Messwerte, die Herzfrequenz und die Sauerstoffsättigung werden auf dem Terminal ausgegeben.

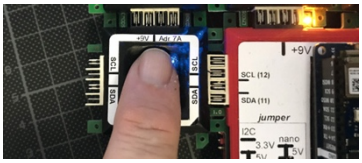
6.6 Puls Oxymeter Brick - PULS mit OLED und digitalem Filter

Der Pulsoxymeter-Baustein misst den Puls am Finger über eine rote und eine infrarote LED, und erfasst die Rückstreuung über eine PIN-Diode. Ein 18 Bit A/D Umsetzer wandelt diese um, und über den I2C (AE und AF) kann man die Daten zum Auswertungs-Programm leiten. Dabei werden die beiden LEDs abwechseln gepulst. Damit lässt sich der Puls am Finger messen, aber auch der Sauerstoffgehalt. Hier wird der integrierte Baustein MAX30102 verwendet.

Achtung: Den Finger ganz ruhig halten und auflegen. Da der Puls optisch gemessen wird, führt Zittern am Finger oder Unruhe zu Fehlmessungen. In der Praxis verwendet man zur Zwangsruhigstellung auch gerne eine Klemmvorrichtung, die hier nicht vorhanden ist. Man kann damit mal experimentieren und z.B. Gummiband etc. verwenden.

Im Programm wird ein sogenanntes Hochpassfilter verwendet, das ermöglicht es langsame Spannungsänderungen zu unterdrücken und bevorzugt den Puls durchzulassen. Die gefundenen Werte werden auch auf dem Terminal mit ausgegeben und auf der OLED angezeigt zusammen mit einer kleinen Grafik.





Bitte den Finger so wie hier gezeigt auf den Puls Oxymeter Brick legen, so dass dieser vollständig abgedeckt wird.

```
/*
  Heart beat plotting!
  By: Nathan Seidle @ SparkFun Electronics
  Date: October 20th, 2016
  https://github.com/sparkfun/MAX30105_Breakout

  Shows the user's heart beat on Arduino's serial plotter

  Instructions:
  1) Load code onto Redboard
  2) Attach sensor to your finger with a rubber band (see below)
  3) Open Tools->'Serial Plotter'
  4) Make sure the drop down is set to 115200 baud
  5) Checkout the blips!
  6) Feel the pulse on your neck and watch it mimic the blips

  It is best to attach the sensor to your finger using a rubber band or other tightening
  device. Humans are generally bad at applying constant pressure to a thing. When you
  press your finger against the sensor it varies enough to cause the blood in your
  finger to flow differently which causes the sensor readings to go wonky.

  Hardware Connections (Breakoutboard to Arduino):
  -5V = 5V (3.3V is allowed)
  -GND = GND
  -SDA = A4 (or SDA)
  -SCL = A5 (or SCL)
  -INT = Not connected

  The MAX30105 Breakout can handle 5V or 3.3V I2C logic. We recommend powering the board with
5V
  but it will also run at 3.3V.
*/

#include <Wire.h>
#include "MAX30105.h"

MAX30105 particleSensor;

double p_coeffs[8]= {-0.005951526098434661, -0.041923825901806856, -0.147699072923821700, -
0.612489327871047262, 0.612489327871047262, 0.147699072923821700, 0.041923825901806856,
0.005951526098434661}; //FIR Coeffisiants
//double p_coeffs[8]= {0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125};
volatile long p_in[9]; // Input Pulse Array for Filter
long p_out; //Pulse Output
long p_disp[65]; //Array for the Display
long minim = 0;
long maxim = 0;
int calampswitch = 64;
long lastval=0;

double rates[4]; //Array of heart rates
byte rateSpot = 0;
double lastBeat = 0;
double BPM;
double beatsPerMinute;
int beatAvg;
long lastPeak;
bool pulse = false;

// 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 ---------------------------------------------
// 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()
{

  disp_buffer_clear(COLOR_BLACK);
  for(int x = 0; x <=7; x++){
        p_in[x] = 0;
        }

     for(int x = 0; x <=63; x++){
        p_disp[x] = 0;
        }
   for(int x = 0; x <=3; x++){
        rates[x] = 0;
        }
  Serial.begin(115200);
  Serial.println("Initializing...");
  Wire.begin(); // I2C Init
  i2c_oled_initall(i2coledssd); // OLED Init
  disp_buffer_clear(COLOR_BLACK);
  disp_lcd_frombuffer();
  // Initialize sensor
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) //Use default I2C port, 400kHz speed
  {
    Serial.println("MAX30105 was not found. Please check wiring/power. ");
    disp_buffer_clear(COLOR_BLACK);
      disp_print_xy_lcd(0, 15, (unsigned char*)"NOT FOUND", COLOR_WHITE, 0);

    disp_lcd_frombuffer();
    while (1);
  }


  //Setup to sense a nice looking saw tooth on the plotter
  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

  particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth,
adcRange); //Configure sensor with these settings

  disp_buffer_clear(COLOR_BLACK);
  disp_print_xy_lcd(0, 0, (unsigned char*)"Freq : ", COLOR_WHITE, 0);
  disp_line_lcd (0,12,63,12, COLOR_WHITE);
}
 //Function to apply FIR Convolution (for Filtering)
void conv ()
{
  int i, j, k;
  double tmp;

  tmp=0;
    for (i = 0; i < 7; i++)  //  position in coefficients array
    {
        tmp += p_coeffs [i] * p_in [i];

    }

    p_out  = tmp;

}
//Find maximum from Array
int findMin(){
  long minimum = p_disp[0];
  for (int c = 0; c <= 63; c++)
    {
```

```
        if (p_disp[c] < minimum)
        {
           minimum = p_disp[c];
        }
    }
  return minimum;

  }

//Find maximum from Array
int findMax(){
  long maximum = p_disp[0];

  for (int c = 0; c <= 63; c++)
    {
        if (p_disp[c] > maximum)
        {
           maximum = p_disp[c];
        }
    }
  return maximum;

  }

void loop()
{

 //Switch Iterator loopvar
 //Shift array elements right
  for (int it = 7; it >= 1; it--)
  {
  p_in[it] = p_in[it-1];
  }
  for (int it = 63; it >= 1; it--)
  {
  p_disp[it] = p_disp[it-1];
  }
  for (int it = 3; it >= 1; it--)
  {
  rates[it] = rates[it-1];
  }
 //Append new value
 int32_t v1= (int32_t)particleSensor.getIR();
 long val = (long)v1;

 //Filter Pulse
 p_in[0] = val;
 conv();
 p_disp[0] = p_out;


 //Find maximum and Minimum for scaling
 maxim = findMax();
 minim = findMin();




 /*
 long rates[15]; //Array of heart rates
 puls min 20 pulse max 220
 */



 //Print to OLED
 //Seld data to Display64x48 Pixel OLED
 disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
 disp_print_xy_lcd(0, 0, (unsigned char*)"Freq : ", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);

  // Get BPM
  if (p_out >= minim + 200){
```

```
    //Get Peak
    long peak =(max(p_out, minim+300)-minim)-300;

    //calculate bpm

    if(peak > lastPeak){
      pulse = false;}

    if (peak < lastPeak)
    {
      if (pulse == false){
        //sensed a beat
        pulse = true;
        Serial.print("!!!PULSE!!!");
        disp_print_xy_lcd(55, 0, (unsigned char*)"O", COLOR_WHITE, 0);
         long lmilli1 = micros();
         long lmilli2 = micros();
         long nowmilis = 0;
         if (lmilli2 > lmilli1){
          nowmilis = lmilli2;
         } else nowmilis = lmilli1;
         Serial.print(nowmilis);
         Serial.print(",");
         long delta = nowmilis - lastBeat;
         lastBeat = nowmilis;
         beatsPerMinute = 60.0 / (((double)delta / 1000.0)/1000.0);
         rates[0] = beatsPerMinute;
        for(int x = 0; x <=3; x++){
           BPM +=rates[x];
           BPM /= 16;
         }
      }
    }
    lastPeak = peak;
  }

 char SBPM[20]; // one extra byte for null
 int IBPM = floor(beatsPerMinute);
 sprintf(SBPM, "%i", IBPM);
 disp_print_xy_lcd(28, 0, (unsigned char*)SBPM, COLOR_WHITE, 0);

 //Print Curve to screen
 lastval =  47-(map(p_disp[0], minim, maxim, 0, 30));
 for (int i=0; i<63;i++) { // nun fuer alle 64 Spalten (x)  des Displays
  long val = 47-(map(p_disp[i], minim, maxim, 0, 30));
    disp_setpixel(i, val , COLOR_WHITE); // koordinate 0,0 ist links oben daher 47-y
    //OLDX NEWX OLDY NEWY
    if (i>0)disp_line_lcd (i-1,lastval,i,val, COLOR_WHITE);
    lastval = val;
  } // alle Spalten
 disp_lcd_frombuffer(); // dann erst updaten, double buffer flimmerfrei

  //Send results to serial plotter

Serial.print(p_out);Serial.print(",");Serial.print(minim);Serial.print(",");Serial.println(max
im);
 //Serial.print(p_disp[0]);
Serial.print(",");Serial.print(minim);Serial.print(",");Serial.print(maxim);Serial.print(",");
Serial.println(beatsPerMinute);
 //Serial.println((max(p_out, minim+400)-minim)-400);

}
```

7. EKG Messung

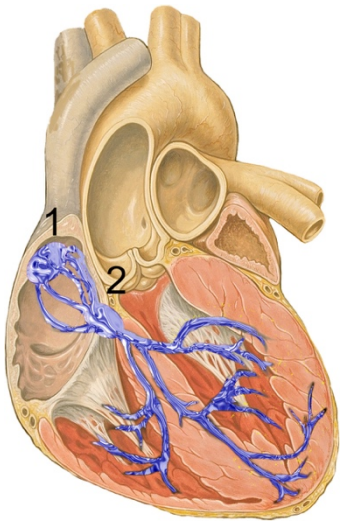
7.1 Theorie der EKG Messung - Wir hören auf unser Herz!

7.1.1 Was ist ein EKG?

Zunächst steht EKG für das Wort „Elektrokardiogramm“. Das Wort setzt sich aus den Griechischen Begriffen „kardia“ für Herz und „grámma“ für Geschriebenes zusammen. Man zeichnet beim EKG die Summe der elektrischen Aktivitäten aller Herzmuskelfasern auf. Im Deutschen nennt man das EGK auch Herzspannungskurve oder Herzschrift.

7.1.2 Was passiert im Körper?

Jede Kontraktion (darunter versteht man die aktive Verkürzung eines Muskels oder Anspannung des Muskels ohne Verkürzung) tritt mit einer elektrischen Erregung, also einem elektrischen Signal zusammen auf. Dieses Signal kommt normalerweise vom sogenannten Sinusknoten. Der Sinusknoten ist somit der „elektrische Taktgeber“ jeder Herzaktion. Er liegt im rechten Vorhof des Herzens (siehe Abbildung „Sinusknoten“). Das erzeugte Signal läuft durch ein Leitungssystem zu den übrigen Herzmuskelzellen. Und genau diese kleine elektrische Spannungsänderung am Herzen kann man dann an der Haut messen.



Schema des Herzens mit Erregungsleitungssystem in Blau:

(1) Sinusknoten

(2) AV-Knoten

7.1.3 Eine kurze Geschichte des EKG

Schon 1843 kam Carlo Matteucci (ein italienischer Physiker und Neurophysiologe) auf den brillanten Schluss, dass die Tätigkeit des Herzens auf elektrischen Vorgängen beruht. Doch erst 1882 schaffte es der englische Physiologe Augustus Desiré Wallter zum ersten Mal ein EKG durchzuführen. Und wer wurde der erste EKG Patient? Sein Hund! Er tauchte dazu die vier Pfoten seines Freunds „Jimmy“ in leitfähige Silberchlorid-Lösung. 5 Jahre später zeichnete er dann erstmals Herzströme mit einem Kapillarelektrometer auf.

Dieses schwer auszusprechende Kapilarelektrometer wurde 1903 durch den niederländischen Arzt Willem Einthoven verbessert, bis es in Klinken richtig genutzt werden konnte und brauchbare Ergebnisse lieferte. Sein Ziel war es, mit einem Gerät und nur einer Ableitung das EKG durchzuführen. Eine Ableitung ist ein Kabel mit einer Elektrode, die an den Körper angebracht wird.

7.2 Biologie: Das Herz, ein spannender Muskel

7.2.1 Erst einmal hinlegen: das Herz in Ruhe

EKG ist nicht gleich EKG, es kommt darauf an, was man genau messen möchte. Wir stellen 3 verschiedene EKG Arten vor: Ruhe-EKG, Langzeit-EKG und Belastungs-EKG

7.2.1.1 Ruhe-EKG Fakten

Dauer: Ca. 5 min
Position: Liegend
Ziel: Schnelldiagnose & allgemeine Untersuchung

7.2.1.2 Ruhe-EKG Durchführung

Beim Ruhe-EKG ist der Patient in liegender Position. Das praktische am Ruhe-EKG ist, dass es sehr schnell geht und es dadurch auch für Notfälle perfekt geeignet ist. Außerdem ist es untern den kardiologischen Standard-Untersuchungen die Aussagekräftigste! Es gibt nur wenige Herz Krankheiten oder Störungen, die vielleicht nicht gefunden werden.

7.2.2 Das Herz im Langzeit Test

7.2.2.1 Langzeit-EKG Fakten

Dauer: Mindestens 24 Stunden
Position: Normal
Ziel: Langzeitdiagnose & Diagnose von temporären Störungen

7.2.2.2. Langzeit-EKG Durchführung

Da 24 Stunden natürlich viel zu lang sind, um die Messung beim Arzt durchzuführen, nimmt der Patient ein kleines Mess- und Aufnahmegerät mit. Heutzutage sind diese sehr klein und passen sogar ohne Probleme in die Hosentasche. Manchmal werden gehen die Aufzeichnungen auch über 48 oder 72 Stunden! Warum misst man nun aber so lange? Manche Herzrhythmusstörungen treten nur temporär auf. Beispiele dafür sind: während des Schlafens oder bei bestimmten Aktivitäten.

7.2.3 Wie verhält sich das Herz bei Belastung?

7.2.3.1 Belastungs-EKG Fakten

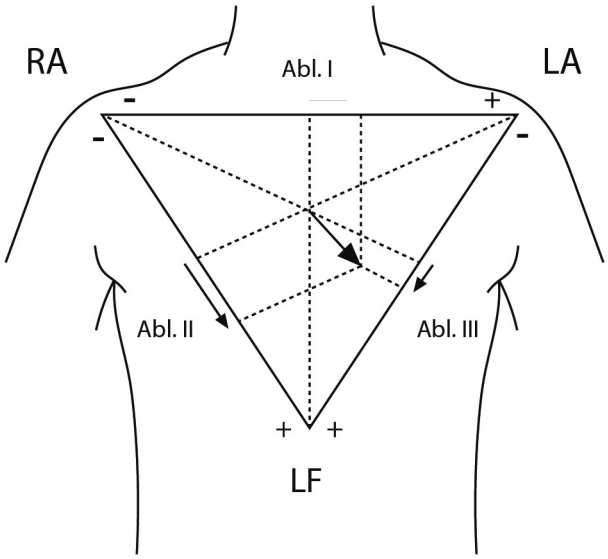
Dauer: Variabel
Position: Unter körperlicher Belastung (laufen)
Ziel: Diagnose von Störungen bei körperlicher Betätigung

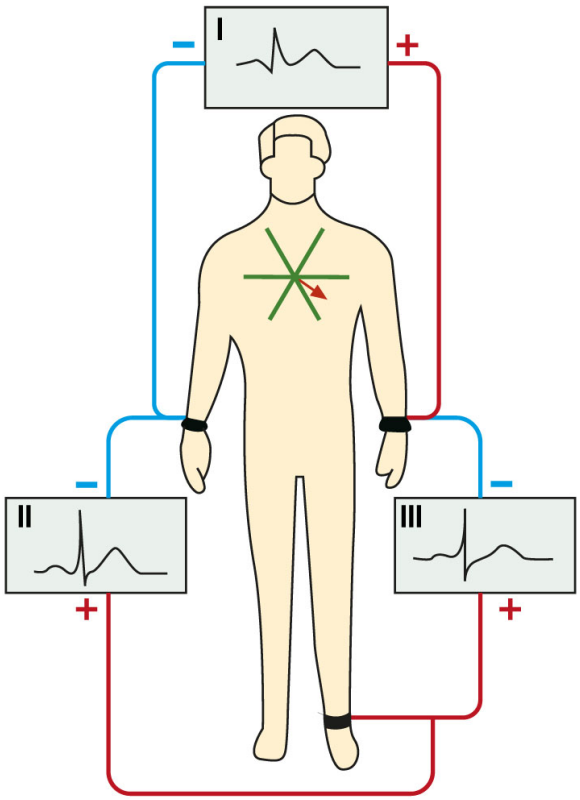
7.2.3.2 Belastungs-EKG Durchführung

Dieses EKG wird vor allem – wie der Name es vermuten lässt - auf dem Laufband oder einem Fahrrad durchgeführt. Wie lange und wie schnell der Patient läuft oder radelt wird durch das sogenannte WHO-Schema bestimmt. Manchmal muss ein Belastungs-EKG auch abgebrochen werden, zum Beispiel, wenn der Blutdruck zu hoch ansteigt oder abfällt und bei Erschöpfung. Beispiele für Erschöpfung sind Schmerzen (meist in den Beinen), Schwindel und Atemnot. Außerdem kann es auch passieren, dass der Puls zu hoch wird. Den maximalen Puls zu errechnen geht ganz einfach: „220 minus Lebensalter“. Wenn man also 25 Jahre alt ist, sollte der Puls nicht über 195 liegen.

7.3 EKG Messung grafisch auf dem PC mit dem seriellen Plotter

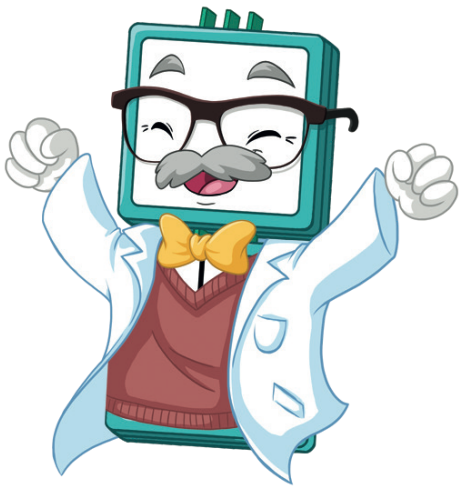
Anschluss nach dem Einthofen-Dreieck an der Brust. Dabei wird immer zwischen zwei Elektroden gemessen, die andere dient als Massebezug. Man kann sie alle versuchsweise durchtauschen, um die charakteristischen Signale zu erhalten.

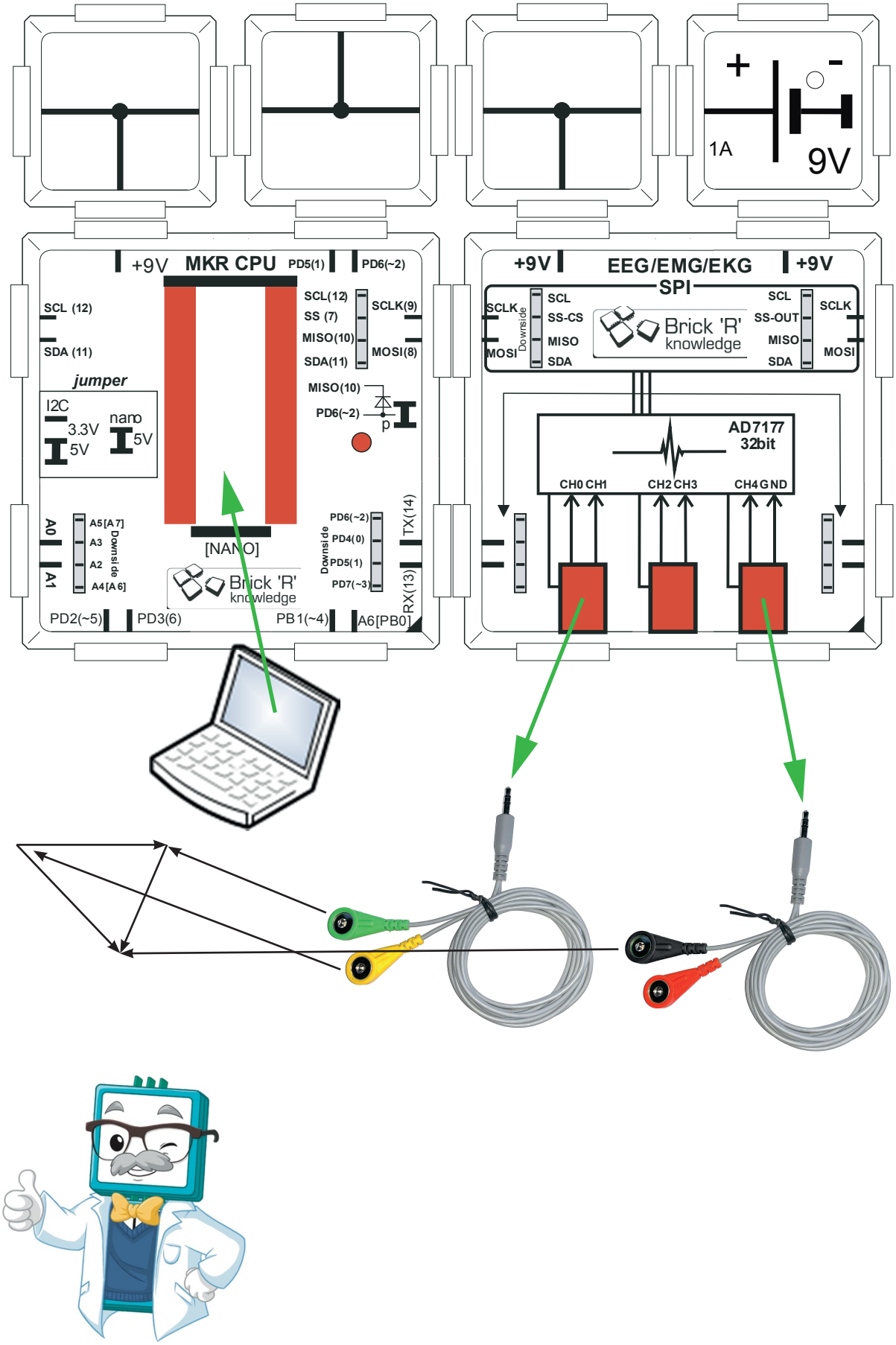




Die Elektroden können direkt bei RA, LA, LF positioniert werden oder wie in der rechten Abildung gezeigt.

I = RA/LA, II = RA/LF, III = LA/LF




+
-
1A
9V
+9V
MKR CPU
PD5(1)
PD6(~2)
SCL (12)
SDA (11)
jumper
I2C
3.3V
5V
nano
5V
SCL(12)
SS (7)
MISO(10)
SDA(11)
SCLK(9)
MOSI(8)
MISO(10)
PD6(~2)
p
A0
A1
A5[A7]
A3
A2
A4[A6]
Downside
[NANO]
Brick 'R' knowledge
PD6(~2)
PD4(0)
PD5(1)
PD7(~3)
Downside
TX(14)
RX(13)
PD2(~5)
PD3(6)
PB1(~4)
A6[PB0]
+9V
EEG/EMG/EKG
+9V
SPI
SCLK
Downside
SCL
SS-CS
MISO
SDA
MOSI
Brick 'R' knowledge
SCL
SS-OUT
MISO
SDA
SCLK
MOSI
AD7177
32bit
CH0 CH1
CH2 CH3
CH4 GND

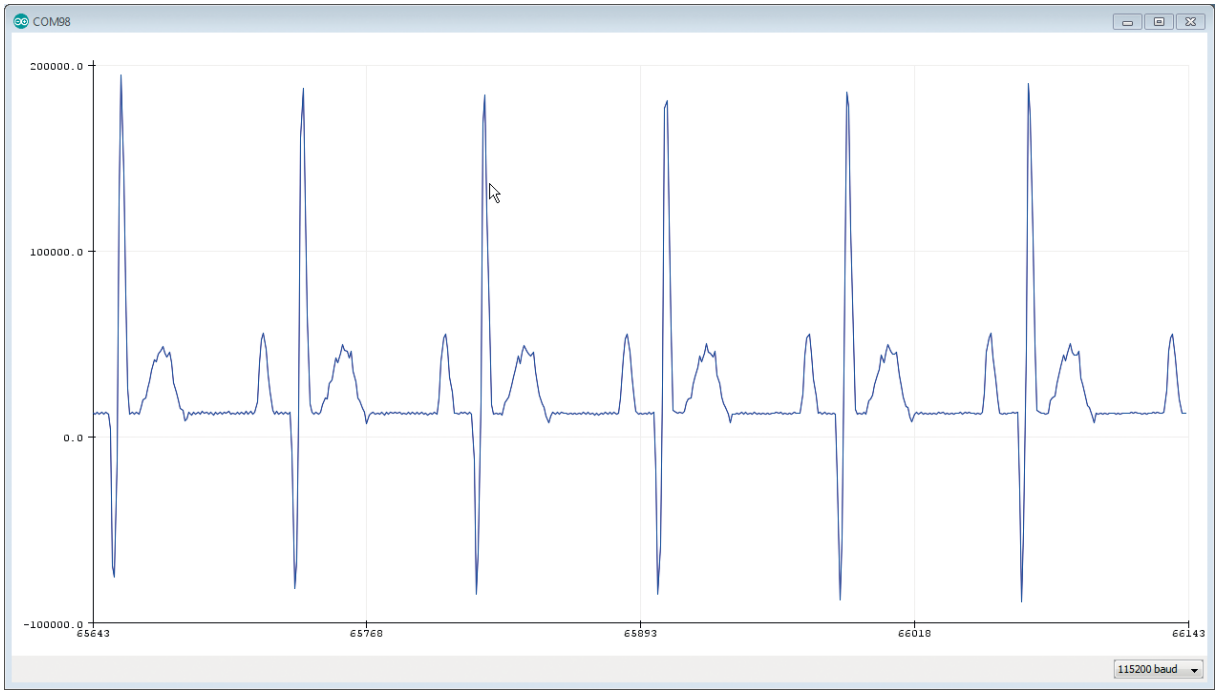
```
// EEG BRICK
//  MKR WIFI 2010 __SAMD21G18A__
// Output to serial Interface !!



// I2C Select: clken selen mosien led0 sense1 sense2 sense3 spez
//             0     1      2      3     4     5      6      7
//             out   out    out   out   in     in     in     in
//

#include <Wire.h> // Definitionen laden fuer I2C
#include <SPI.h>
#include <avr/pgmspace.h> // weitere Definitionen

// USe IOs of MKR
#ifdef ARDUINO_SAMD_MKRWIFI1010
#define __AVR_SAMD21__   1

#endif


// Alle Adressen bei den 8574xx bricks:
#define i2cIO9534_0 (0x40>>1) // Trick um elegant mit Bytes
#define i2cIO9534_1 (0x42>>1) // zu arbeiten statt in 7 Bit
#define i2cIO9534_2 (0x44>>1) // das letzte Bit ist das R/W
#define i2cIO9534_3 (0x46>>1) // dass von der Arduino
#define i2cIO9534_4 (0x48>>1) // Bibliothek dazugefuegt wird
#define i2cIO9534_5 (0x4A>>1) // wir definieren hier alle
#define i2cIO9534_6 (0x4C>>1) // Bereiche die man mit den
#define i2cIO9534_7 (0x4E>>1) // Bricks einstellen kann,.

#define i2cIO9534A_0 (0x70>>1) // Die Serie PCF8474AT
#define i2cIO9534A_1 (0x72>>1) // beginnt bei Adresse
#define i2cIO9534A_2 (0x74>>1) // 0x70 = 70 sedezimal
#define i2cIO9534A_3 (0x76>>1)  // 01110000 binaer
#define i2cIO9534A_4 (0x78>>1) // oder intern 0111000
#define i2cIO9534A_5 (0x7A>>1) // dabei 0111xxx
#define i2cIO9534A_6 (0x7C>>1) // mit x fuer die
#define i2cIO9534A_7 (0x7E>>1) // Dilschalterposition

#define myi2cIOadr i2cIO9534_0 // HIER PASSENDE ADRESSE EINTRAGEN

#define SSPB 10 // Pin 10 = SS  - --- PB2  1<<2 als Maske

// SPI: SCLK MOSI MISO und SS Brick=PB2 D10 fest !!
// MEGA Pin 51=MOSI Pin 52=SCK und Pin50=MISO °!!!
// mega:
// NANO: MOSI:PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// UNO: MOSI: PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// ACHUNG: MISO ist auch STatus DATA READY !!!
//
// PORTB 4  (7 6 5 4 -- 3 2 1 0)
// BEI MKR PD6

#if defined(__AVR_ATmega2560__)  // Arduino MEGA2560
#define MISOPIN 50
#warning ("INFO : compiles for AT MEGA2560")
#elif defined(__AVR_SAMD21__)
#warning ("INFO : compiles for MKR WIFI1010")
#define MISOPIN 2  // READ ONLY MKR ueber Diode...
#else // NANO
#warning ("INFO : compiles rest: also NANO...")
#define MISOPIN 12
#endif

// #define MISOPIN 50

void enabless()
{
  //digitalWrite(SSPB, LOW);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, LOW);
  #else
```

```
   PORTB &= 0xfb; // Default = high 7 6 5 4 3 2low 1 0 PB2 low
  #endif
  delayMicroseconds(10);
}
void disabless()
{
  delayMicroseconds(10);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, HIGH);
  #else
    PORTB |= 1 << 2; // Default = high bit2 PB2
  #endif
  // digitalWrite(SSPB, HIGH);
  delayMicroseconds(10);
  //
}

//AD 7177-x
// Register
// SS:-------_________________----------

//
unsigned short ad71_readid()
{
  unsigned char ch1;
  unsigned short val1=0;
  enabless();
  ch1 = SPI.transfer(0x47); // COmm: -wen r-w ra5..9  r-1=1 read 0=writ
  val1 = SPI.transfer16(0); // ID
  disabless();
  return(val1);
}

//

//
#define NOCHAN 1

void ad71_init()
{
   enabless();
   // Register setzen...
   SPI.transfer(0x01); SPI.transfer16(0x800C);  // ADCMODE ! ref en, mode continouus, clock
Crystal : 100 00 000  0 000 11 00
   SPI.transfer(0x02); SPI.transfer16(0x0442);  // INTERFACE MOD data_stat(6)en w32 en --- 000
0   0 10 0    0 1   0 0  00 1 0  old 0x0042
   SPI.transfer(0x06); SPI.transfer16(0x060d);  // GPIO 000 0  0 11 0   0 0 00 1101
   // Channel 0
   SPI.transfer(0x10); SPI.transfer16(0x8001);  // CH0   enable set0 ain0 ain1 ... 10 00 00 00
000 0 0001
   // Channel 1
   SPI.transfer(0x11); SPI.transfer16(0x0043);  // CH1   enable set0 ain2 ain3 ... 10 00 00 00
010 0 0011
   // Channel 2
   SPI.transfer(0x12); SPI.transfer16(0x0096);  // CH2   enable set0 ain4 ref- ... 10 00 00 00
100 1 0110
   // Channel 3  optional
   SPI.transfer(0x13); SPI.transfer16(0x0232);  // CH3   enable set0 smp+ tmp- ... 10 00 00 10
001 1 0010
   // alle gleiches setup daher setup 0
   // SETup registers.
   SPI.transfer(0x20); SPI.transfer16(0x1300);  // SET0 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x21); SPI.transfer16(0x1300);  // SET1 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x22); SPI.transfer16(0x1300);  // SET2 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x23); SPI.transfer16(0x1300);  // SET3 : 000 1  0 0 1 1  0 0 00 0000
   // Filter Config gehoeren zu setup reg
   SPI.transfer(0x28); SPI.transfer16(0x050e);  // FLT0 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x29); SPI.transfer16(0x050a);  // FLT1 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2A); SPI.transfer16(0x050a);  // FLT2 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2B); SPI.transfer16(0x050a);  // FLT3 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
```

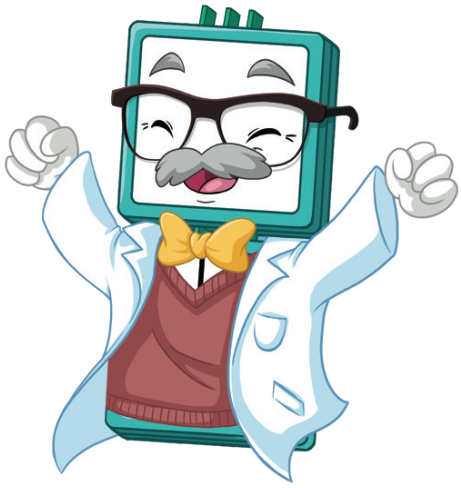
```
   // OFFSET und GAIN auf factory lassen...
   disabless();

}

unsigned long values[4];

// tmperature 470uV/K
//         1181972248 beispiel bei 2.5V Ref+- 0.137mV 292K ->19.7
// 67BCCE0 108776672 -3.5


void ad71_reset()
{
  enabless();
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  disabless();
  delayMicroseconds(550);
}


void ad71_dataread()
{
  // Wenn ready incl status 4 byts 32 bit
  // STATUS
  unsigned char sts;
  unsigned char ch1;
  unsigned long val1=0;
  unsigned long val2=0;
  unsigned int idx;
  static int first = 0;
  int val=0;
  int i=0;
  for (i=0; i<NOCHAN;i++) {
    // SYNC:
    // NICHT RDY Abfragen,,,
  #ifdef XXXXXX
    enabless();
    do {

      sts = SPI.transfer(0x40);
      sts = SPI.transfer(0x00);
       delayMicroseconds(10);
       // Serial.print(sts);
       // Serial.println("S wt:");

    } while (((sts & 0x80)== 0x80) && (first == 1));
    first = 1;
    disabless();
#endif

    // DATA RDY
    delayMicroseconds(10);
    //
    enabless();
    delayMicroseconds(4); // DRDY valid...
    // SINGLE MODE SPI.transfer(0x01); SPI.transfer16(0x801C);  // ADCMODE ! ref en, mode
SINGLE, clock Crystal : 100 00 000  0 000 11 00
    //
    // Sonst abfragen hier immer...
    do {
      val = digitalRead(MISOPIN); // MISOPIN
    } while (val == HIGH);

    //
    ch1 = SPI.transfer(0x44); // COmm:  data read
```

```
    //
    val1 = SPI.transfer16(0); // data <
    val2 = SPI.transfer16(0); // data
    //

     //
    delayMicroseconds(5); // DRDY valid...
    disableSS();
    //
    val1 = val2 | (val1 << 16); // 4 bit status
    idx = val2 & 0x3; // CHANNEL FLG  bit 3210
    values[idx] = val1 & 0xfffffff0;

  }
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("Starting EEG:");

  #if defined(__AVR_SAMD21__)
   pinMode(MISOPIN, INPUT); // MISO !!
   pinMode(7, OUTPUT);
  #else
   pinMode(12, INPUT); // MISO !!
   pinMode(SSPB, OUTPUT);
  #endif
  disableSS();



  Wire.begin(); // I2C aktivieren !
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x03) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0xF0) ;  // 1=input
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0x0f) ;  // led an !!
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  // INIT
   SPI.begin();
  SPI.beginTransaction(SPISettings(7000000, MSBFIRST, SPI_MODE3));
  // NICHT VERWENDEN:
  //SPI.setDataMode(SPI_MODE3);  // sonst immer 1     clk ---____----   rising edhe = spi mode
3 krit mkr´
  //SPI.setClockDivider(8);

  Serial.println("Searching EEG:");
  unsigned short id=ad71_readid(); // 0x4fdx
  while ((id & 0xfff0)!= 0x4fd0) {
     Serial.print("ERROR id=");Serial.print(id);
     Serial.println(" EEG Brick not found");
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x07) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x0f) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     id=ad71_readid(); // 0x4fdx
  }
  Serial.print("id=");Serial.println(id);
  // OK Init dr Register
  ad71_reset();
  ad71_init();


}
```

```
void loop() {


ad71_dataread();
#ifdef ALLL
  Serial.print("ad:");Serial.print(values[0],HEX);
  Serial.print(" ");Serial.print(values[1],HEX);
  Serial.print(" ");Serial.print(values[2],HEX);
  Serial.print(" ");Serial.print(values[3],HEX);
  Serial.println();
#endif
  long v1 = values[0]-0x800000001;
  // long v2 = values[1]-0x800000001;
 //Serial.print(v1);Serial.print(",");Serial.println(v2);
 Serial.println(v1);
}
```

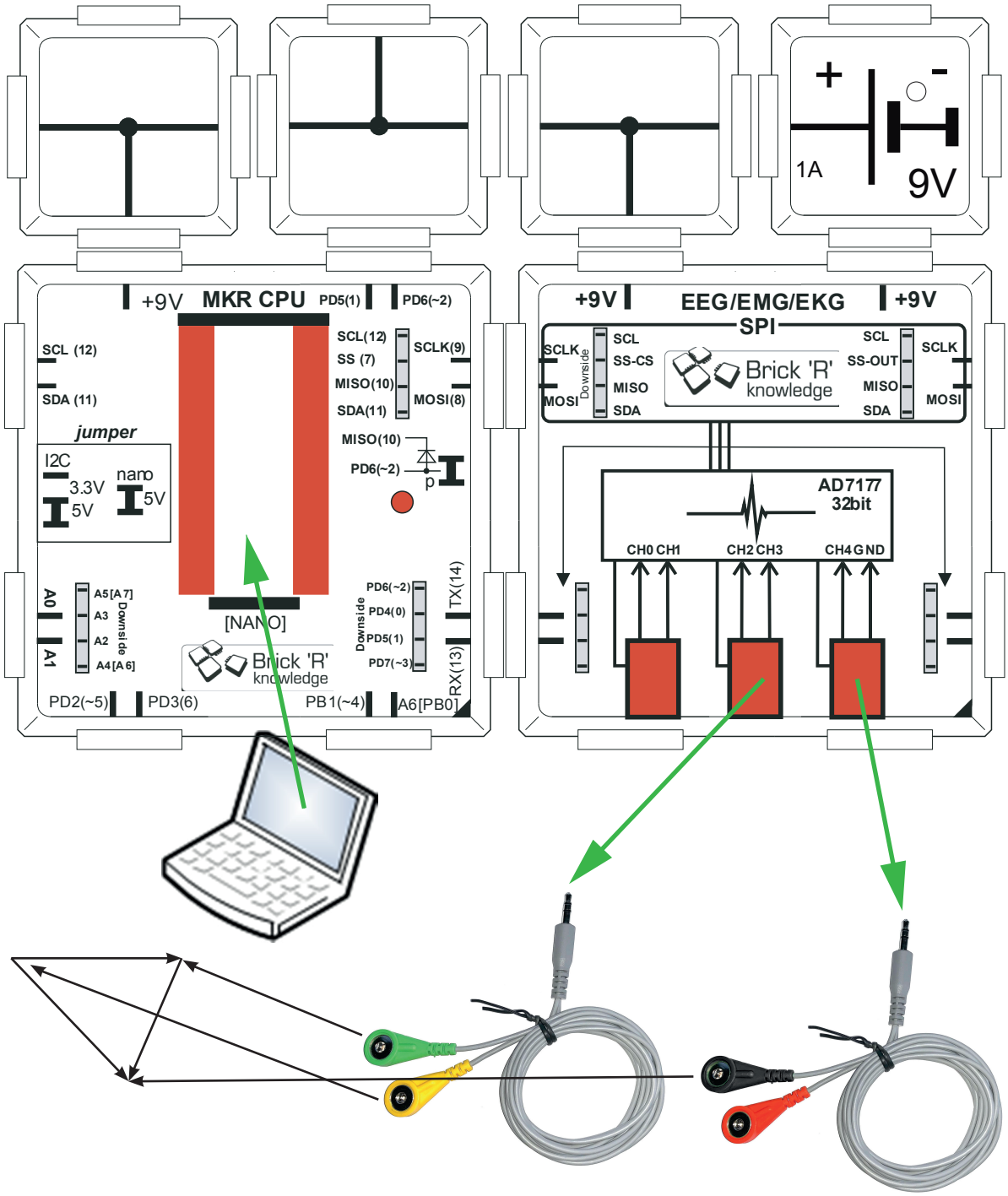


Was passiert? Die Messwerte, die Herzfrequenz wird auf dem seriellen Plotter ausgegeben.



7.4 EKG Messung grafisch auf dem PC mit dem seriellen Plotter (CH2/CH3)

Anschluss nach dem Einthofen-Dreieck an der Brust. Dabei wird immer zwischen zwei Elektroden gemessen, die andere dient als Massebezug. Man kann sie alle versuchsweise durchtauschen, um die charakteristischen Signale zu erhalten. Im Vergleich zum letzten Versuch werden hier die Kanäle CH2 und CH3 benutzt.



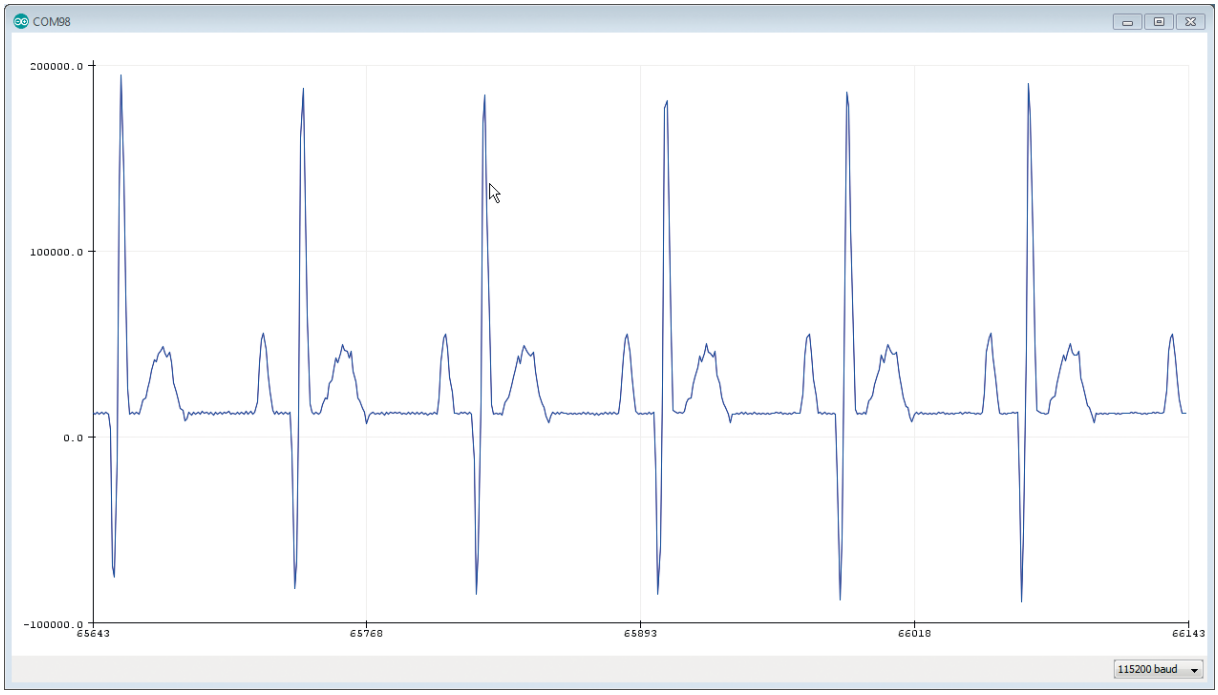
```
// EEG BRICK
//  MKR WIFI 2010 __SAMD21G18A__
// Output to serial Interface !!



// I2C Select: clken selen mosien led0 sense1 sense2 sense3 spez
//             0     1      2      3     4     5      6      7
//             out   out    out   out   in     in     in     in
//

#include <Wire.h> // Definitionen laden fuer I2C
#include <SPI.h>
#include <avr/pgmspace.h> // weitere Definitionen

// USe IOs of MKR
#ifdef ARDUINO_SAMD_MKRWIFI1010
#define __AVR_SAMD21__   1

#endif


// Alle Adressen bei den 8574xx bricks:
#define i2cIO9534_0 (0x40>>1) // Trick um elegant mit Bytes
#define i2cIO9534_1 (0x42>>1) // zu arbeiten statt in 7 Bit
#define i2cIO9534_2 (0x44>>1) // das letzte Bit ist das R/W
#define i2cIO9534_3 (0x46>>1) // dass von der Arduino
#define i2cIO9534_4 (0x48>>1) // Bibliothek dazugefuegt wird
#define i2cIO9534_5 (0x4A>>1) // wir definieren hier alle
#define i2cIO9534_6 (0x4C>>1) // Bereiche die man mit den
#define i2cIO9534_7 (0x4E>>1) // Bricks einstellen kann,.

#define i2cIO9534A_0 (0x70>>1) // Die Serie PCF8474AT
#define i2cIO9534A_1 (0x72>>1) // beginnt bei Adresse
#define i2cIO9534A_2 (0x74>>1) // 0x70 = 70 sedezimal
#define i2cIO9534A_3 (0x76>>1)  // 01110000 binaer
#define i2cIO9534A_4 (0x78>>1) // oder intern 0111000
#define i2cIO9534A_5 (0x7A>>1) // dabei 0111xxx
#define i2cIO9534A_6 (0x7C>>1) // mit x fuer die
#define i2cIO9534A_7 (0x7E>>1) // Dilschalterposition

#define myi2cIOadr i2cIO9534_0 // HIER PASSENDE ADRESSE EINTRAGEN

#define SSPB 10 // Pin 10 = SS  - --- PB2  1<<2 als Maske

// SPI: SCLK MOSI MISO und SS Brick=PB2 D10 fest !!
// MEGA Pin 51=MOSI Pin 52=SCK und Pin50=MISO °!!!
// mega:
// NANO: MOSI:PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// UNO: MOSI: PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// ACHUNG: MISO ist auch STatus DATA READY !!!
//
// PORTB 4  (7 6 5 4 -- 3 2 1 0)
// BEI MKR PD6

#if defined(__AVR_ATmega2560__)  // Arduino MEGA2560
#define MISOPIN 50
#warning ("INFO : compiles for AT MEGA2560")
#elif defined(__AVR_SAMD21__)
#warning ("INFO : compiles for MKR WIFI1010")
#define MISOPIN 2  // READ ONLY MKR ueber Diode...
#else // NANO
#warning ("INFO : compiles rest: also NANO...")
#define MISOPIN 12
#endif

// #define MISOPIN 50

void enabless()
{
  //digitalWrite(SSPB, LOW);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, LOW);
  #else
```

```
   PORTB &= 0xfb; // Default = high 7 6 5 4 3 2low 1 0 PB2 low
  #endif
  delayMicroseconds(10);
}
void disabless()
{
  delayMicroseconds(10);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, HIGH);
  #else
    PORTB |= 1 << 2; // Default = high bit2 PB2
  #endif
  // digitalWrite(SSPB, HIGH);
  delayMicroseconds(10);
  //
}

//AD 7177-x
// Register
// SS:-------_________________----------

//
unsigned short ad71_readid()
{
  unsigned char ch1;
  unsigned short val1=0;
  enabless();
  ch1 = SPI.transfer(0x47); // COmm: -wen r-w ra5..9  r-1=1 read 0=writ
  val1 = SPI.transfer16(0); // ID
  disabless();
  return(val1);
}

//

//
#define NOCHAN 1

void ad71_init()
{
   enabless();
   // Register setzen...
   SPI.transfer(0x01); SPI.transfer16(0x800C);  // ADCMODE ! ref en, mode continouus, clock
Crystal : 100 00 000  0 000 11 00
   SPI.transfer(0x02); SPI.transfer16(0x0442);  // INTERFACE MOD data_stat(6)en w32 en --- 000
0   0 10 0    0 1   0 0  00 1 0  old 0x0042
   SPI.transfer(0x06); SPI.transfer16(0x060d);  // GPIO 000 0  0 11 0   0 0 00 1101
   // Channel 0
   SPI.transfer(0x10); SPI.transfer16(0x8043);  // CH0   enable set0 ain0 ain1 ... 10 00 00 00
000 0 0001  OLD:0 Chenael Tausch!!
   // Channel 1
   SPI.transfer(0x11); SPI.transfer16(0x0043);  // CH1   enable set0 ain2 ain3 ... 10 00 00 00
010 0 0011
   // Channel 2
   SPI.transfer(0x12); SPI.transfer16(0x0096);  // CH2   enable set0 ain4 ref- ... 10 00 00 00
100 1 0110
   // Channel 3  optional
   SPI.transfer(0x13); SPI.transfer16(0x0232);  // CH3   enable set0 smp+ tmp- ... 10 00 00 10
001 1 0010
   // alle gleiches setup daher setup 0
   // SETup registers.
   SPI.transfer(0x20); SPI.transfer16(0x1300);  // SET0 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x21); SPI.transfer16(0x1300);  // SET1 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x22); SPI.transfer16(0x1300);  // SET2 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x23); SPI.transfer16(0x1300);  // SET3 : 000 1  0 0 1 1  0 0 00 0000
   // Filter Config gehoeren zu setup reg
   SPI.transfer(0x28); SPI.transfer16(0x050e);  // FLT0 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x29); SPI.transfer16(0x050a);  // FLT1 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2A); SPI.transfer16(0x050a);  // FLT2 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2B); SPI.transfer16(0x050a);  // FLT3 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
```

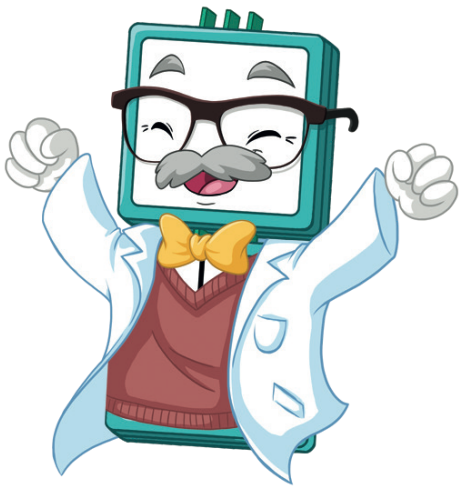
```
   // OFFSET und GAIN auf factory lassen...
   disabless();

}

unsigned long values[4];

// tmperature 470uV/K
//         1181972248 beispiel bei 2.5V Ref+- 0.137mV 292K ->19.7
// 67BCCE0 108776672 -3.5


void ad71_reset()
{
  enabless();
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  disabless();
  delayMicroseconds(550);
}


void ad71_dataread()
{
  // Wenn ready incl status 4 byts 32 bit
  // STATUS
  unsigned char sts;
  unsigned char ch1;
  unsigned long val1=0;
  unsigned long val2=0;
  unsigned int idx;
  static int first = 0;
  int val=0;
  int i=0;
  for (i=0; i<NOCHAN;i++) {
    // SYNC:
    // NICHT RDY Abfragen,,,
  #ifdef XXXXXX
    enabless();
    do {

      sts = SPI.transfer(0x40);
      sts = SPI.transfer(0x00);
       delayMicroseconds(10);
       // Serial.print(sts);
       // Serial.println("S wt:");

    } while (((sts & 0x80)== 0x80) && (first == 1));
    first = 1;
    disabless();
#endif

    // DATA RDY
    delayMicroseconds(10);
    //
    enabless();
    delayMicroseconds(4); // DRDY valid...
    // SINGLE MODE SPI.transfer(0x01); SPI.transfer16(0x801C);  // ADCMODE ! ref en, mode
SINGLE, clock Crystal : 100 00 000  0 000 11 00
    //
    // Sonst abfragen hier immer...
    do {
      val = digitalRead(MISOPIN); // MISOPIN
    } while (val == HIGH);

    //
    ch1 = SPI.transfer(0x44); // COmm:  data read
```

```
    //
    val1 = SPI.transfer16(0); // data <
    val2 = SPI.transfer16(0); // data
    //

     //
    delayMicroseconds(5); // DRDY valid...
    disableSS();
    //
    val1 = val2 | (val1 << 16); // 4 bit status
    idx = val2 & 0x3; // CHANNEL FLG  bit 3210
    values[idx] = val1 & 0xfffffff0;

  }
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("Starting EEG:");

  #if defined(__AVR_SAMD21__)
   pinMode(MISOPIN, INPUT); // MISO !!
   pinMode(7, OUTPUT);
  #else
   pinMode(12, INPUT); // MISO !!
   pinMode(SSPB, OUTPUT);
  #endif
  disableSS();

  Wire.begin(); // I2C aktivieren !
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x03) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0xF0) ;  // 1=input
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0x0f) ;  // led an !!
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  // INIT
   SPI.begin();
  SPI.beginTransaction(SPISettings(7000000, MSBFIRST, SPI_MODE3));
  // NICHT VERWENDEN:
  //SPI.setDataMode(SPI_MODE3);  // sonst immer 1     clk ---____----   rising edhe = spi mode
3 krit mkr´
  //SPI.setClockDivider(8);

  Serial.println("Searching EEG:");
  unsigned short id=ad71_readid(); // 0x4fdx
  while ((id & 0xfff0)!= 0x4fd0) {
     Serial.print("ERROR id=");Serial.print(id);
     Serial.println(" EEG Brick not found");
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x07) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x0f) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     id=ad71_readid(); // 0x4fdx
  }
  Serial.print("id=");Serial.println(id);
  // OK Init dr Register
  ad71_reset();
  ad71_init();
}

void loop() {

ad71_dataread();
#ifdef ALLL
```

```
  Serial.print("ad:");Serial.print(values[0],HEX);
  Serial.print(" ");Serial.print(values[1],HEX);
  Serial.print(" ");Serial.print(values[2],HEX);
  Serial.print(" ");Serial.print(values[3],HEX);
  Serial.println();
#endif
  long v1 = values[0]-0x800000001;
  // long v2 = values[1]-0x800000001;
 //Serial.print(v1);Serial.print(",");Serial.println(v2);
 Serial.println(v1);
}
```

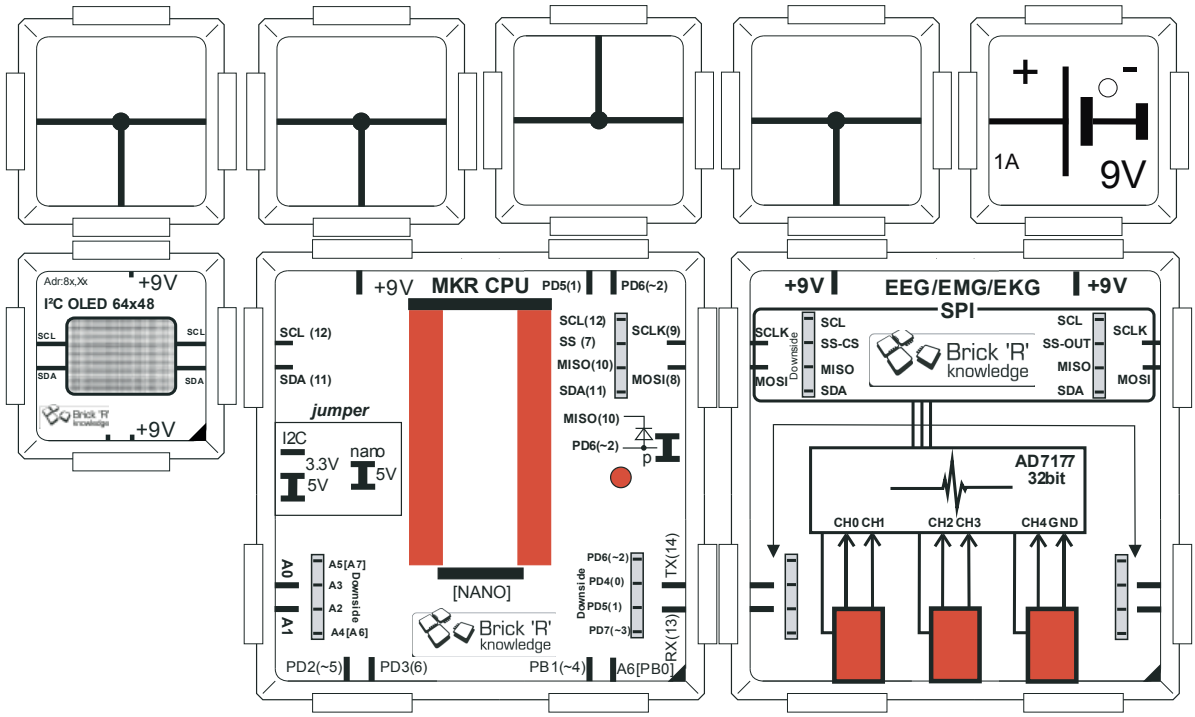


Was passiert? Die Messwerte, die Herzfrequenz wird auf dem seriellen Plotter ausgegeben.



7.5 EKG Messung grafisch auf dem OLED-Display

Anschluss nach dem Einthofen-Dreieck an der Brust. Dabei wird immer zwischen zwei Elektroden gemessen, die andere dient als Massebezug. Man kann sie alle versuchsweise durchtauschen, um die charakteristischen Signale zu erhalten.




```
// EEG BRICK
//  MKR WIFI 2010 __SAMD21G18A__
// Output to serial Interface !!


// I2C Select: clken selen mosien led0 sense1 sense2 sense3 spez
//             0     1      2      3     4      5      6      7
//             out   out    out    out   in     in     in     in
//

#include <Wire.h> // Definitionen laden fuer I2C
#include <SPI.h>
#include <avr/pgmspace.h> // weitere Definitionen

// USe IOs of MKR
#ifdef ARDUINO_SAMD_MKRWIFI1010
#define __AVR_SAMD21__   1

#endif


// Alle Adressen bei den 8574xx bricks:
#define i2cIO9534_0 (0x40>>1) // Trick um elegant mit Bytes
#define i2cIO9534_1 (0x42>>1) // zu arbeiten statt in 7 Bit
#define i2cIO9534_2 (0x44>>1) // das letzte Bit ist das R/W
#define i2cIO9534_3 (0x46>>1) // dass von der Arduino
#define i2cIO9534_4 (0x48>>1) // Bibliothek dazugefuegt wird
#define i2cIO9534_5 (0x4A>>1) // wir definieren hier alle
#define i2cIO9534_6 (0x4C>>1) // Bereiche die man mit den
#define i2cIO9534_7 (0x4E>>1) // Bricks einstellen kann,.

#define i2cIO9534A_0 (0x70>>1) // Die Serie PCF8474AT
#define i2cIO9534A_1 (0x72>>1) // beginnt bei Adresse
#define i2cIO9534A_2 (0x74>>1) // 0x70 = 70 sedezimal
#define i2cIO9534A_3 (0x76>>1)  // 01110000 binaer
#define i2cIO9534A_4 (0x78>>1) // oder intern 0111000
```

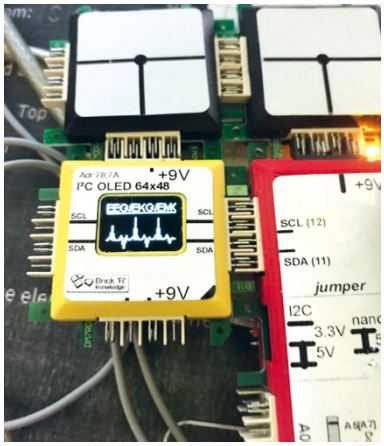
```
#define i2cIO9534A_5 (0x7A>>1) // dabei 0111xxx
#define i2cIO9534A_6 (0x7C>>1) // mit x fuer die
#define i2cIO9534A_7 (0x7E>>1) // Dilschalterposition

#define myi2cIOadr i2cIO9534_0 // HIER PASSENDE ADRESSE EINTRAGEN

#define SSPB 10 // Pin 10 = SS  - --- PB2  1<<2 als Maske

// SPI: SCLK MOSI MISO und SS Brick=PB2 D10 fest !!
// MEGA Pin 51=MOSI Pin 52=SCK und Pin50=MISO °!!!
// mega:
// NANO: MOSI:PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// UNO: MOSI: PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// ACHUNG: MISO ist auch STatus DATA READY !!!
//
// PORTB 4  (7 6 5 4 -- 3 2 1 0)
// BEI MKR PD6

#if defined(__AVR_ATmega2560__)  // Arduino MEGA2560
#define MISOPIN 50
#warning ("INFO : compiles for AT MEGA2560")
#elif defined(__AVR_SAMD21__)
#warning ("INFO : compiles for MKR WIFI1010")
#define MISOPIN 2  // READ ONLY MKR ueber Diode...
#else // NANO
#warning ("INFO : compiles rest: also NANO...")
#define MISOPIN 12
#endif

// #define MISOPIN 50

long p_disp[64];


// 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 ---------------------------------------------
// 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 enabless()
{
  //digitalWrite(SSPB, LOW);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, LOW);
  #else
   PORTB &= 0xfb; // Default = high 7 6 5 4 3 2low 1 0 PB2 low
  #endif
  delayMicroseconds(10);
}
void disabless()
{
  delayMicroseconds(10);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, HIGH);
  #else
    PORTB |= 1 << 2; // Default = high bit2 PB2
  #endif
  // digitalWrite(SSPB, HIGH);
  delayMicroseconds(10);
  //
}

//AD 7177-x
// Register
// SS:-------_________________----------
```

```
//
unsigned short ad71_readid()
{
  unsigned char ch1;
  unsigned short val1=0;
  enabless();
  ch1 = SPI.transfer(0x47); // COmm: -wen r-w ra5..9  r-1=1 read 0=writ
  val1 = SPI.transfer16(0); // ID
  disabless();
  return(val1);
}

//

//
#define NOCHAN 1

void ad71_init()
{
   enabless();
   // Register setzen...
   SPI.transfer(0x01); SPI.transfer16(0x800C);  // ADCMODE ! ref en, mode continouus, clock
Crystal : 100 00 000  0 000 11 00
   SPI.transfer(0x02); SPI.transfer16(0x0442);  // INTERFACE MOD data_stat(6)en w32 en --- 000
0   0 10 0    0 1   0 0  00 1 0  old 0x0042
   SPI.transfer(0x06); SPI.transfer16(0x060d);  // GPIO 000 0  0 11 0   0 0 00 1101
   // Channel 0
   SPI.transfer(0x10); SPI.transfer16(0x8001);  // CH0   enable set0 ain0 ain1 ... 10 00 00 00
000 0 0001
   // Channel 1
   SPI.transfer(0x11); SPI.transfer16(0x0043);  // CH1   enable set0 ain2 ain3 ... 10 00 00 00
010 0 0011
   // Channel 2
   SPI.transfer(0x12); SPI.transfer16(0x0096);  // CH2   enable set0 ain4 ref- ... 10 00 00 00
100 1 0110
   // Channel 3  optional
   SPI.transfer(0x13); SPI.transfer16(0x0232);  // CH3   enable set0 smp+ tmp- ... 10 00 00 10
001 1 0010
   // alle gleiches setup daher setup 0
   // SETup registers.
   SPI.transfer(0x20); SPI.transfer16(0x1300);  // SET0 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x21); SPI.transfer16(0x1300);  // SET1 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x22); SPI.transfer16(0x1300);  // SET2 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x23); SPI.transfer16(0x1300);  // SET3 : 000 1  0 0 1 1  0 0 00 0000
   // Filter Config gehoeren zu setup reg
   SPI.transfer(0x28); SPI.transfer16(0x050e);  // FLT0 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x29); SPI.transfer16(0x050a);  // FLT1 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2A); SPI.transfer16(0x050a);  // FLT2 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2B); SPI.transfer16(0x050a);  // FLT3 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   // OFFSET und GAIN auf factory lassen...
   disabless();

}

unsigned long values[4];

// tmperature 470uV/K
//         1181972248 beispiel bei 2.5V Ref+- 0.137mV 292K ->19.7
// 67BCCE0 108776672 -3.5


void ad71_reset()
{
  enabless();
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
```

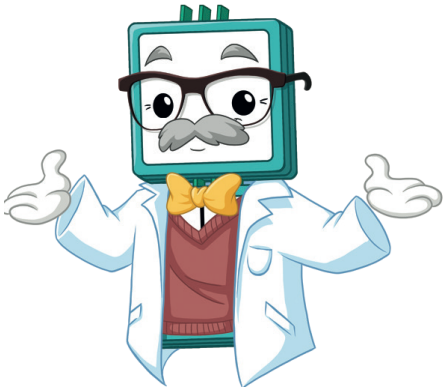
```
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  disabless();
  delayMicroseconds(550);
}


void ad71_dataread()
{
  // Wenn ready incl status 4 byts 32 bit
  // STATUS
  unsigned char sts;
  unsigned char ch1;
  unsigned long val1=0;
  unsigned long val2=0;
  unsigned int idx;
  static int first = 0;
  int val=0;
  int i=0;
  for (i=0; i<NOCHAN;i++) {
    // SYNC:
    // NICHT RDY Abfragen,,,
  #ifdef XXXXXX
    enabless();
    do {

      sts = SPI.transfer(0x40);
      sts = SPI.transfer(0x00);
       delayMicroseconds(10);
       // Serial.print(sts);
       // Serial.println("S wt:");

    } while (((sts & 0x80)== 0x80) && (first == 1));
    first = 1;
    disabless();
#endif

    // DATA RDY
    delayMicroseconds(10);
    //
    enabless();
    delayMicroseconds(4); // DRDY valid...
    // SINGLE MODE SPI.transfer(0x01); SPI.transfer16(0x801C);  // ADCMODE ! ref en, mode
SINGLE, clock Crystal : 100 00 000  0 000 11 00
    //
    // Sonst abfragen hier immer...
    do {
      val = digitalRead(MISOPIN); // MISOPIN
    } while (val == HIGH);

    //
    ch1 = SPI.transfer(0x44); // COmm:  data read

    //
    val1 = SPI.transfer16(0); // data <
    val2 = SPI.transfer16(0); // data
    //

     //
    delayMicroseconds(5); // DRDY valid...
    disabless();
    //
    val1 = val2 | (val1 << 16); // 4 bit status
    idx = val2 & 0x3; // CHANNEL FLG  bit 3210
    values[idx] = val1 & 0xfffffff0;

  }
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("Starting EEG:");
```

```
  #if defined(__AVR_SAMD21__)
   pinMode(MISOPIN, INPUT); // MISO !!
   pinMode(7, OUTPUT);
  #else
   pinMode(12, INPUT); // MISO !!
   pinMode(SSPB, OUTPUT);
  #endif
  disabless();



  Wire.begin(); // I2C aktivieren !
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x03) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0xF0) ;  // 1=input
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0x0f) ;  // led an !!
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  i2c_oled_initall(i2coledssd);
  disp_buffer_clear(COLOR_BLACK);
  // INIT
   SPI.begin();
  SPI.beginTransaction(SPISettings(7000000, MSBFIRST, SPI_MODE3));
  // NICHT VERWENDEN:
  //SPI.setDataMode(SPI_MODE3);  // sonst immer 1     clk ---____----   rising edhe = spi mode
3 krit mkr´
  //SPI.setClockDivider(8);
 disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
 disp_print_xy_lcd(0, 0, (unsigned char*)"EEG/EKG/EMG", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);
  disp_print_xy_lcd(20, 20, (unsigned char*)"Starte...", COLOR_WHITE, 0);
  Serial.println("Searching EEG:");
  disp_lcd_frombuffer();
  unsigned short id=ad71_readid(); // 0x4fdx
  while ((id & 0xfff0)!= 0x4fd0) {
     Serial.print("ERROR id=");Serial.print(id);
     Serial.println(" EEG Brick not found");
     disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
      disp_print_xy_lcd(0, 15, (unsigned char*)"NOT FOUND", COLOR_WHITE, 0);
      disp_lcd_frombuffer();
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x07) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x0f) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     id=ad71_readid(); // 0x4fdx
  }
  Serial.print("id=");Serial.println(id);
  // OK Init dr Register
  ad71_reset();
  ad71_init();

//Init display schlange
   for(int x = 0; x <=63; x++){
        p_disp[x] = 0;
        }


}



int findMin(){
  long minimum = p_disp[0];
  int c;
  for (c = 0; c < 62; c++)
    {
```

```
        if (p_disp[c] < minimum)
        {
           minimum = p_disp[c];
        }
    }
  return minimum;

  }

int findMax(){
  long maximum = p_disp[0];
  int c;
  for (c = 0; c < 62; c++)
    {
        if (p_disp[c] > maximum)
        {
           maximum = p_disp[c];
        }
    }
  return maximum;

  }

long lastval=0;
void loop() {


ad71_dataread();
 #ifdef ALLL
    Serial.print("ad:");Serial.print(values[0],HEX);
    Serial.print(" ");Serial.print(values[1],HEX);
    Serial.print(" ");Serial.print(values[2],HEX);
    Serial.print(" ");Serial.print(values[3],HEX);
    Serial.println();
 #endif

  for (int it = 63; it >= 1; it--)
  {
  p_disp[it] = p_disp[it-1];
  }
  long v1 = values[0]-0x800000001;
   p_disp[0] = v1;

 long  minim = findMin();
 long maxim = findMax();

   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
 disp_print_xy_lcd(0, 0, (unsigned char*)"EEG/EKG/EMG", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);
 for (int i=0; i<63;i++) { // nun fuer alle 64 Spalten (x)  des Displays
  long val = 47-(map(p_disp[i], minim, maxim, 0, 30));
    disp_setpixel(i, val , COLOR_WHITE); // koordinate 0,0 ist links oben daher 47-y
    //OLDX NEWX OLDY NEWY
    disp_line_lcd (i-1,lastval,i,val, COLOR_WHITE);
    lastval = val;
  } // alle Spalten
 disp_lcd_frombuffer(); // dann erst updaten, double buffer flimmerfrei


  // long v2 = values[1]-0x800000001;
 //Serial.print(v1);Serial.print(",");Serial.println(v2);
 Serial.println(v1);


}
```

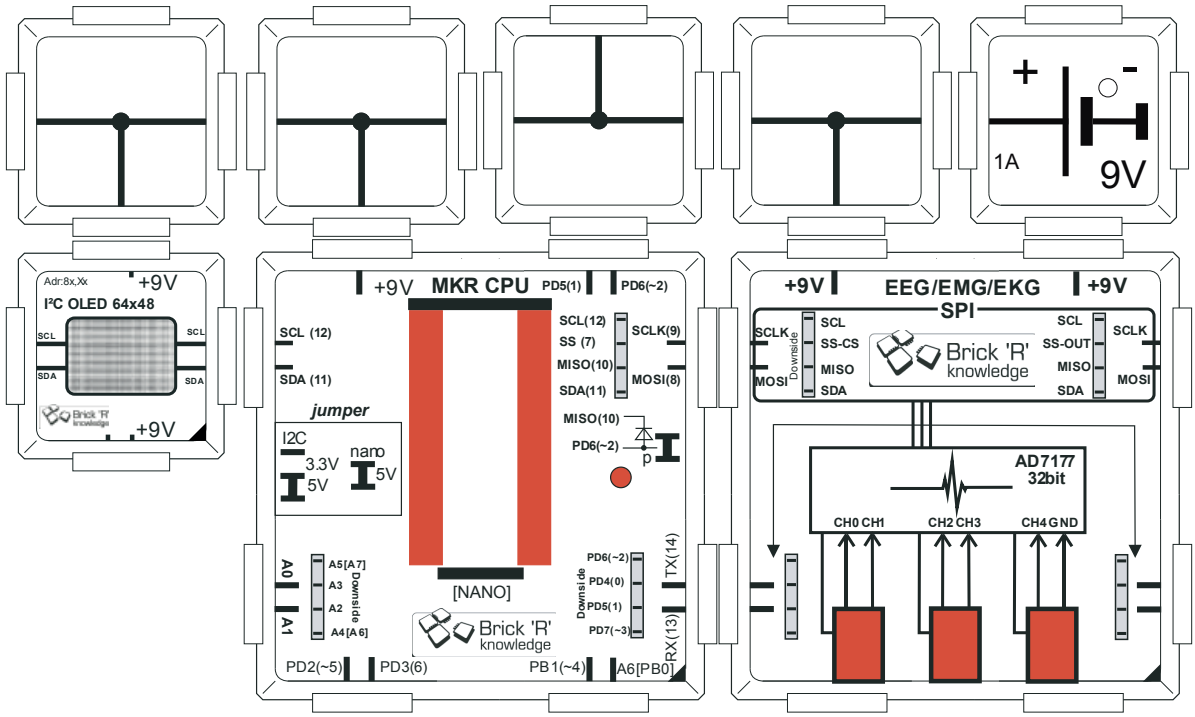


Was passiert? Die Messwerte, die Herzfrequenz wird auf dem OLED-Display ausgegeben.



7.6 EKG Messung grafisch auf dem OLED-Display mit Pulsfrequenzanzeige

Anschluss nach dem Einthofen-Dreieck an der Brust. Dabei wird immer zwischen zwei Elektroden gemessen, die andere dient als Massebezug. Man kann sie alle versuchsweise durchtauschen, um die charakteristischen Signale zu erhalten.




```
// EEG BRICK
//  MKR WIFI 2010 __SAMD21G18A__
// Output to serial Interface !!


// I2C Select: clken selen mosien led0 sense1 sense2 sense3 spez
//             0     1      2      3     4      5      6      7
//             out   out    out    out   in     in     in     in
//

#include <Wire.h> // Definitionen laden fuer I2C
#include <SPI.h>
#include <avr/pgmspace.h> // weitere Definitionen

// USe IOs of MKR
#ifdef ARDUINO_SAMD_MKRWIFI1010
#define __AVR_SAMD21__   1

#endif


// Alle Adressen bei den 8574xx bricks:
#define i2cIO9534_0 (0x40>>1) // Trick um elegant mit Bytes
#define i2cIO9534_1 (0x42>>1) // zu arbeiten statt in 7 Bit
#define i2cIO9534_2 (0x44>>1) // das letzte Bit ist das R/W
#define i2cIO9534_3 (0x46>>1) // dass von der Arduino
#define i2cIO9534_4 (0x48>>1) // Bibliothek dazugefuegt wird
#define i2cIO9534_5 (0x4A>>1) // wir definieren hier alle
#define i2cIO9534_6 (0x4C>>1) // Bereiche die man mit den
#define i2cIO9534_7 (0x4E>>1) // Bricks einstellen kann,.

#define i2cIO9534A_0 (0x70>>1) // Die Serie PCF8474AT
#define i2cIO9534A_1 (0x72>>1) // beginnt bei Adresse
#define i2cIO9534A_2 (0x74>>1) // 0x70 = 70 sedezimal
```

```
#define i2cIO9534A_3 (0x76>>1)  // 01110000 binaer
#define i2cIO9534A_4 (0x78>>1) // oder intern 0111000
#define i2cIO9534A_5 (0x7A>>1) // dabei 0111xxx
#define i2cIO9534A_6 (0x7C>>1) // mit x fuer die
#define i2cIO9534A_7 (0x7E>>1) // Dilschalterposition

#define myi2cIOadr i2cIO9534_0 // HIER PASSENDE ADRESSE EINTRAGEN

#define SSPB 10 // Pin 10 = SS  - --- PB2  1<<2 als Maske

// SPI: SCLK MOSI MISO und SS Brick=PB2 D10 fest !!
// MEGA Pin 51=MOSI Pin 52=SCK und Pin50=MISO °!!!
// mega:
// NANO: MOSI:PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// UNO: MOSI: PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// ACHUNG: MISO ist auch STatus DATA READY !!!
//
// PORTB 4  (7 6 5 4 -- 3 2 1 0)
// BEI MKR PD6

#if defined(__AVR_ATmega2560__)  // Arduino MEGA2560
#define MISOPIN 50
#warning ("INFO : compiles for AT MEGA2560")
#elif defined(__AVR_SAMD21__)
#warning ("INFO : compiles for MKR WIFI1010")
#define MISOPIN 2  // READ ONLY MKR ueber Diode...
#else // NANO
#warning ("INFO : compiles rest: also NANO...")
#define MISOPIN 12
#endif

// #define MISOPIN 50
int numBrks = 0;
int ArdrsBrk[8];

long p_disp[64];

double rates[10]; //Array of heart rates
byte rateSpot = 0;
double lastBeat = 0;
double BPM;
double beatsPerMinute;
int beatAvg;
long lastPeak = 0;
bool pulse = false;
long peak = 0;


// 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 ---------------------------------------------
// 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 enabless()
{
  //digitalWrite(SSPB, LOW);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, LOW);
  #else
   PORTB &= 0xfb; // Default = high 7 6 5 4 3 2low 1 0 PB2 low
  #endif
  delayMicroseconds(10);
}
void disabless()
{
  delayMicroseconds(10);
```

```
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, HIGH);
  #else
    PORTB |= 1 << 2; // Default = high bit2 PB2
  #endif
  // digitalWrite(SSPB, HIGH);
  delayMicroseconds(10);
  //
}

//AD 7177-x
// Register
// SS:-------_________________----------

//
unsigned short ad71_readid()
{
  unsigned char ch1;
  unsigned short val1=0;
  enabless();
  ch1 = SPI.transfer(0x47); // COmm: -wen r-w ra5..9  r-1=1 read 0=writ
  val1 = SPI.transfer16(0); // ID
  disabless();
  return(val1);
}

//

//
#define NOCHAN 1

void ad71_init()
{
   enabless();
   // Register setzen...
   SPI.transfer(0x01); SPI.transfer16(0x800C);  // ADCMODE ! ref en, mode continouus, clock
Crystal : 100 00 000  0 000 11 00
   SPI.transfer(0x02); SPI.transfer16(0x0442);  // INTERFACE MOD data_stat(6)en w32 en --- 000
0   0 10 0    0 1   0 0  00 1 0  old 0x0042
   SPI.transfer(0x06); SPI.transfer16(0x060d);  // GPIO 000 0  0 11 0   0 0 00 1101
   // Channel 0
   SPI.transfer(0x10); SPI.transfer16(0x8001);  // CH0   enable set0 ain0 ain1 ... 10 00 00 00
000 0 0001
   // Channel 1
   SPI.transfer(0x11); SPI.transfer16(0x0043);  // CH1   enable set0 ain2 ain3 ... 10 00 00 00
010 0 0011
   // Channel 2
   SPI.transfer(0x12); SPI.transfer16(0x0096);  // CH2   enable set0 ain4 ref- ... 10 00 00 00
100 1 0110
   // Channel 3  optional
   SPI.transfer(0x13); SPI.transfer16(0x0232);  // CH3   enable set0 smp+ tmp- ... 10 00 00 10
001 1 0010
   // alle gleiches setup daher setup 0
   // SETup registers.
   SPI.transfer(0x20); SPI.transfer16(0x1300);  // SET0 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x21); SPI.transfer16(0x1300);  // SET1 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x22); SPI.transfer16(0x1300);  // SET2 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x23); SPI.transfer16(0x1300);  // SET3 : 000 1  0 0 1 1  0 0 00 0000
   // Filter Config gehoeren zu setup reg
   SPI.transfer(0x28); SPI.transfer16(0x050e);  // FLT0 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x29); SPI.transfer16(0x050a);  // FLT1 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2A); SPI.transfer16(0x050a);  // FLT2 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2B); SPI.transfer16(0x050a);  // FLT3 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   // OFFSET und GAIN auf factory lassen...
   disabless();

}

unsigned long values[4];
```

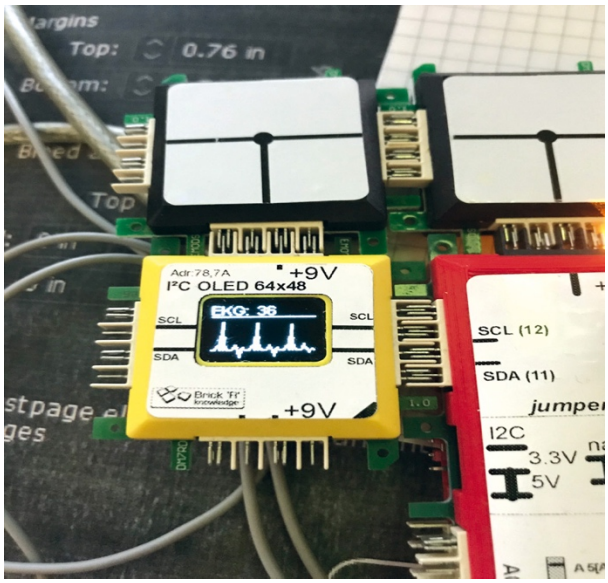
```
// tmperature 470uV/K
//         1181972248 beispiel bei 2.5V Ref+- 0.137mV 292K ->19.7
// 67BCCE0 108776672 -3.5


void ad71_reset()
{
  enabless();
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  disabless();
  delayMicroseconds(550);
}


void ad71_dataread()
{
  // Wenn ready incl status 4 byts 32 bit
  // STATUS
  unsigned char sts;
  unsigned char ch1;
  unsigned long val1=0;
  unsigned long val2=0;
  unsigned int idx;
  static int first = 0;
  int val=0;
  int i=0;
  for (i=0; i<NOCHAN;i++) {
    // SYNC:
    // NICHT RDY Abfragen,,,
  #ifdef XXXXXX
    enabless();
    do {

      sts = SPI.transfer(0x40);
      sts = SPI.transfer(0x00);
       delayMicroseconds(10);
       // Serial.print(sts);
       // Serial.println("S wt:");

    } while (((sts & 0x80)== 0x80) && (first == 1));
    first = 1;
    disabless();
#endif

    // DATA RDY
    delayMicroseconds(10);
    //
    enabless();
    delayMicroseconds(4); // DRDY valid...
    // SINGLE MODE SPI.transfer(0x01); SPI.transfer16(0x801C);  // ADCMODE ! ref en, mode
SINGLE, clock Crystal : 100 00 000  0 000 11 00
    //
    // Sonst abfragen hier immer...
    do {
      val = digitalRead(MISOPIN); // MISOPIN
    } while (val == HIGH);

    //
    ch1 = SPI.transfer(0x44); // COmm:  data read

    //
    val1 = SPI.transfer16(0); // data <
    val2 = SPI.transfer16(0); // data
    //

     //
    delayMicroseconds(5); // DRDY valid...
```

```
    disableSS();
    //
    val1 = val2 | (val1 << 16); // 4 bit status
    idx = val2 & 0x3; // CHANNEL FLG  bit 3210
    values[idx] = val1 & 0xfffffff0;

  }
}

bool CheckForDevice(int Adrs){

  switch (Adrs){   // Startvorgang I2C Adresse
      case 0: Wire.beginTransmission(i2cIO9534_0);break;
      case 1: Wire.beginTransmission(i2cIO9534_1);break;
      case 2: Wire.beginTransmission(i2cIO9534_2);break;
      case 3: Wire.beginTransmission(i2cIO9534_3);break;
      case 4: Wire.beginTransmission(i2cIO9534_4);break;
      case 5: Wire.beginTransmission(i2cIO9534_5);break;
      case 6: Wire.beginTransmission(i2cIO9534_6);break;
      case 7: Wire.beginTransmission(i2cIO9534_7);break;
  }

  Wire.write(0x03) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0xF0) ;  // 1=input
  Wire.endTransmission(); // Stop Kondition setzen bei I2C

  switch (Adrs){     // Startvorgang I2C Adresse
      case 0: Wire.beginTransmission(i2cIO9534_0);break;
      case 1: Wire.beginTransmission(i2cIO9534_1);break;
      case 2: Wire.beginTransmission(i2cIO9534_2);break;
      case 3: Wire.beginTransmission(i2cIO9534_3);break;
      case 4: Wire.beginTransmission(i2cIO9534_4);break;
      case 5: Wire.beginTransmission(i2cIO9534_5);break;
      case 6: Wire.beginTransmission(i2cIO9534_6);break;
      case 7: Wire.beginTransmission(i2cIO9534_7);break;
  }

  Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0x0f) ;  // led an !!
  Wire.endTransmission(); // Stop Kondition setzen bei I2C

  //SPI ID Abfragen
  SPI.begin();
  SPI.beginTransaction(SPISettings(7000000, MSBFIRST, SPI_MODE3));
  Serial.print("Searching EEG in Adrs ");Serial.println(Adrs);
  unsigned short id=ad71_readid(); // 0x4fdx
  if ((id & 0xfff0)!= 0x4fd0) { //Kein EEG Brick hier gefunden
     Serial.print("ERROR id=");Serial.print(id);
     Serial.println(" EEG Brick here not found");
     return false;
  }
  else{ //EEG Hier gefunden -> return True
   Serial.print("id=");Serial.println(id);
   Serial.println(" EEG Brick found");
   // OK Init dr Register
   ad71_reset();
   ad71_init();
   return true;
  }

}

void setup() {

  // put your setup code here, to run once:
  Serial.begin(115200);
  Wire.begin(); // I2C aktivieren !
  Serial.println("Starting EEG:");

  #if defined(__AVR_SAMD21__)
   pinMode(MISOPIN, INPUT); // MISO !!
   pinMode(7, OUTPUT);
  #else
   pinMode(12, INPUT); // MISO !!
```

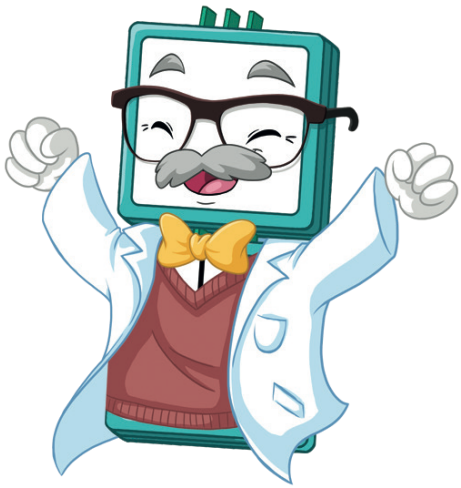
```
   pinMode(SSPB, OUTPUT);
  #endif
  disableSS();

 i2c_oled_initall(i2coledssd);
 disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
 disp_print_xy_lcd(0, 0, (unsigned char*)"EEG/EKG/EMG", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);
 disp_print_xy_lcd(20, 20, (unsigned char*)"Starte...", COLOR_WHITE, 0);
 disp_lcd_frombuffer();


 for (int i = 7; i >= 0; i--){
  if (CheckForDevice(i)){
      ArdrsBrk[numBrks] = i;
      numBrks++;
      Serial.print(numBrks);Serial.println(" Gefunden");
  }
 }

 if (numBrks <= 0){ //Keine Bricks Gefunden
   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
   disp_print_xy_lcd(0, 20, (unsigned char*)"0 Gefunden", COLOR_WHITE, 0);
   disp_lcd_frombuffer();
   while (1){Serial.println("Nichts Gefunden");}
 }
 else{
   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
   disp_print_xy_lcd(0, 20, (unsigned char*)"Gefunden", COLOR_WHITE, 0);
   disp_lcd_frombuffer();
 }

//Init display schlange
   for(int x = 0; x <=63; x++){
        p_disp[x] = 0;
        }

}//END SETUP



int findMin(){
  long minimum = p_disp[0];
  int c;
  for (c = 0; c < 62; c++)
    {
        if (p_disp[c] < minimum)
        {
           minimum = p_disp[c];
        }
    }
  return minimum;

  }

int findMax(){
  long maximum = p_disp[0];
  int c;
  for (c = 0; c < 62; c++)
    {
        if (p_disp[c] > maximum)
        {
           maximum = p_disp[c];
        }
    }
  return maximum;

  }

long lastval=0;
void loop() {


ad71_dataread();
```

```
#ifdef ALLL
   Serial.print("ad:");Serial.print(values[0],HEX);
   Serial.print(" ");Serial.print(values[1],HEX);
   Serial.print(" ");Serial.print(values[2],HEX);
   Serial.print(" ");Serial.print(values[3],HEX);
   Serial.println();
#endif
//Zirkulare schleifen
 for (int it = 63; it >= 1; it--)
 {
 p_disp[it] = p_disp[it-1];
 }

 for (int it = 9; it >= 1; it--)
 {
 rates[it] = rates[it-1];
 }

 //get Values
 long v1 = values[0]-0x800000001;
  p_disp[0] = v1;

//Find Minimum und Maximum von Values
long  minim = findMin();
long maxim = findMax();


disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel

 // Get BPM Pulse Calculieren
 if (v1 >= minim + 150000){
   peak = v1;
        Serial.print("!!!PULSE!!!");
        disp_print_xy_lcd(55, 0, (unsigned char*)"O", COLOR_WHITE, 0);
         long lmilli1 = micros();
         long lmilli2 = micros();
         long nowmilis = 0;
         if (lmilli2 > lmilli1){
          nowmilis = lmilli2;
         } else nowmilis = lmilli1;
         Serial.print(nowmilis);
         Serial.print(",");
         long delta = nowmilis - lastBeat;
         lastBeat = nowmilis;
         beatsPerMinute = 60.0 / (((double)delta / 1000.0)/1000.0);
         rates[0] = beatsPerMinute;
        for(int x = 0; x <=8; x++){
           BPM +=rates[x];
           BPM /= 9;
         }
 }


//Pulse In lesbare form Convertieren
 char SBPM[20]; // one extra byte for null
 int IBPM = floor(beatsPerMinute);
 sprintf(SBPM, "%i", IBPM);
 //
 if (floor(beatsPerMinute) < 150) disp_print_xy_lcd(28, 0, (unsigned char*)SBPM, COLOR_WHITE,
0);
 else{disp_print_xy_lcd(28, 0, (unsigned char*)"0", COLOR_WHITE, 0);}
//Anzeigen
 disp_print_xy_lcd(0, 0, (unsigned char*)"EKG: ", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);
 for (int i=0; i<63;i++) { // nun fuer alle 64 Spalten (x)  des Displays
  long val = 47-(map(p_disp[i], minim, maxim, 0, 30));
    disp_setpixel(i, val , COLOR_WHITE); // koordinate 0,0 ist links oben daher 47-y
    //OLDX NEWX OLDY NEWY
    disp_line_lcd (i-1,lastval,i,val, COLOR_WHITE);
    lastval = val;
  } // alle Spalten
  disp_lcd_frombuffer(); // dann erst updaten, double buffer flimmerfrei
  // long v2 = values[1]-0x800000001;
  // Serial.print(v1);Serial.print(",");Serial.println(v2);
```

```
  Serial.println(v1);
}
```



Was passiert? Die Messwerte, die Herzfrequenz und der Puls werden auf dem OLED-Display ausgegeben.



7.7 Theorie der EMG Messung

7.7.1 Was ist ein EMG?

EMG steht für das Wort Elektromyographie und bezeichnet die Methode zur Messung elektrischer Signale, welche durch die Muskeln erzeugt werden. Kurz gesagt: man misst die Muskelaktivität.

7.7.2 EMG vs. OEMG

Es gibt zwei Möglichkeiten, eine EMG Messung durchzuführen: mit der Nadelelektronen oder Oberflächenelektroden (daher die Abkürzung OEMG = Oberflächen-Elektromyographie). Im Brick'R'knowledge Bio Feedback Set sind Oberflächenelektroden enthalten, da sie für die private Nutzung praktischer sind und natürlich ungefährlicher als richtige Nadeln. Man kann sich den Unterschied der Messergebnisse ganz einfach vorstellen: Mit der Nadel können Ärzte an einem einzigen Muskel eine Messung durchführen, mit der Oberflächenelektrode wird nicht ein einziges Aktionspotenzial, sondern die Summe aller Aktionspotenziale (dazu später mehr) gemessen. Schließlich nehmen die Oberflächenelektroden alle elektrischen Änderungen wahr, die mittels der Haut übertragen werden. Kurz gesagt: mit der Nadel präzise messen, mit den Oberflächenelektroden allgemeiner messen.

7.7.3 Was passiert im Körper?

Wie wir bereits im Kapitel über das EKG gelernt haben, beruht die Tätigkeit des Herzens auf elektrischen Vorgängen. Genauso ist es auch beim EMG, dort werden die elektrischen Impulse der Muskeln gemessen. Die elektrischen Impulse entstehen durch Ionen (elektrisch geladene Atome oder Moleküle), die zwischen den zwei Orten „innerhalb der Zelle" und „außerhalb der Zelle" herumspringen. Ausschlaggebend ist das positive Natriumion und das positive Kalium-Ion. Durch den Ortswechsel der Ionen und somit der Änderung der Polarität fließt Strom, den wir mittels EMG messen können.

Wenn eine Muskelaktivität auftritt, treten drei Phasen ein.

1. Ruhepotential

Das Ruhepotential im Zellinnenraum liegt bei ca. -80mV, somit befinden sich verhältnismäßig mehr negative geladene Ionen im Zellinnenraum, als außerhalb der Zelle. Lapidar gesprochen: viel „Minus" innerhalb der Zelle, weniger „Minus" außerhalb der Zelle. Gegenüber außen ist also der Zellinnenraum negativer.

2. Depolarisation

Nun erfolgt der Reiz: positiv geladene Natriumionen strömen sehr schnell in den Zellinnenraum. Nun ist die Anzahl positiv geladener Ionen im Zellinnenraum höher, als außerhalb. Das verhältnismäßige Zellpotential (also die Differenz von innerhalb und außerhalb der Zelle) wird positiv. Beispielsweise könnte nun das Potential bei +30mV liegen.

3. Repolarisation

Nun strömen positiv geladene Kalium-Ionen aus dem Zellinneren – die Differenz wird wieder kleiner, bis erneut der Ausgangszustand (Ruhepotential) erreicht wird. Das Potential liegt nun wieder bei ca. -80mV.

Vereinfacht ausgedrückt - um einen besseren Überblick zu erhalten - folgt noch einmal eine kurze Zusammenfassung der verhältnismäßigen Zustände innerhalb und außerhalb der Zellen:

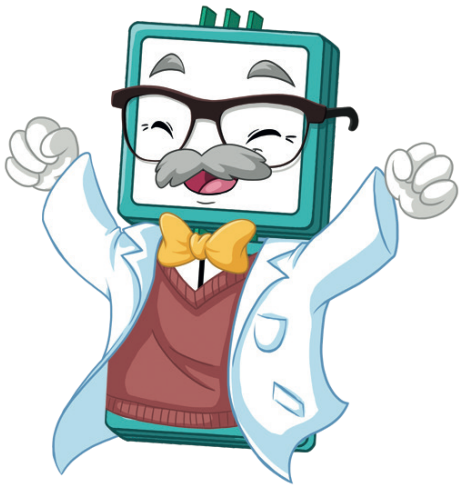
Ruhepotential: Innen Negativ, Außen Positiv

Depolarisation: Innen Positiv, Außen Negativ

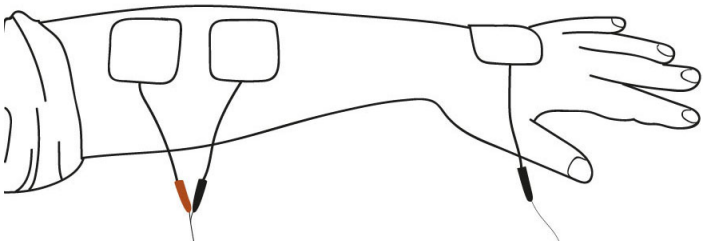
Repolarisation: Innen Negativ, Außen Positiv

7.7.4 Spontanaktivität und der Grund des EMGs

Wie wir im vorherigen Abschnitt gelernt haben, sollte bei einem entspannten Muskel das Ruhepotential vorliegen. Schließlich wird ja kein Muskel angespannt, womit auch keine elektrischen Ströme auftauchen. Und genau dazu ist das EMG da: falls nun im Ruhezustand trotzdem Strom fließt, kann das ein Anzeichen für eine Krankheit sein. Spontanaktivität ist die Eigenaktivität eines Muskels ohne äußere Einwirkung oder Anspannung. Diese Spontanaktivität kann man auf dem Messgerät sehen: plötzlich wird die unausgelenkte, waagerechte Linie zu einer Welle: es passiert etwas, obwohl gar nichts da sein sollte. Diese Spontanaktivität kann viele, verschiedene Gründe haben. Das spannende ist, dass man anhand der Amplitudenwerte und der Frequenzwerte dieser Welle(n) ziemlich genau weiß, was der Grund dafür ist. Beispiele für Erkrankungen sind eine Fehlfunktion der Muskelmembran, Neuropathie, Probleme im Rückenmark und andere Muskelschädigungen.

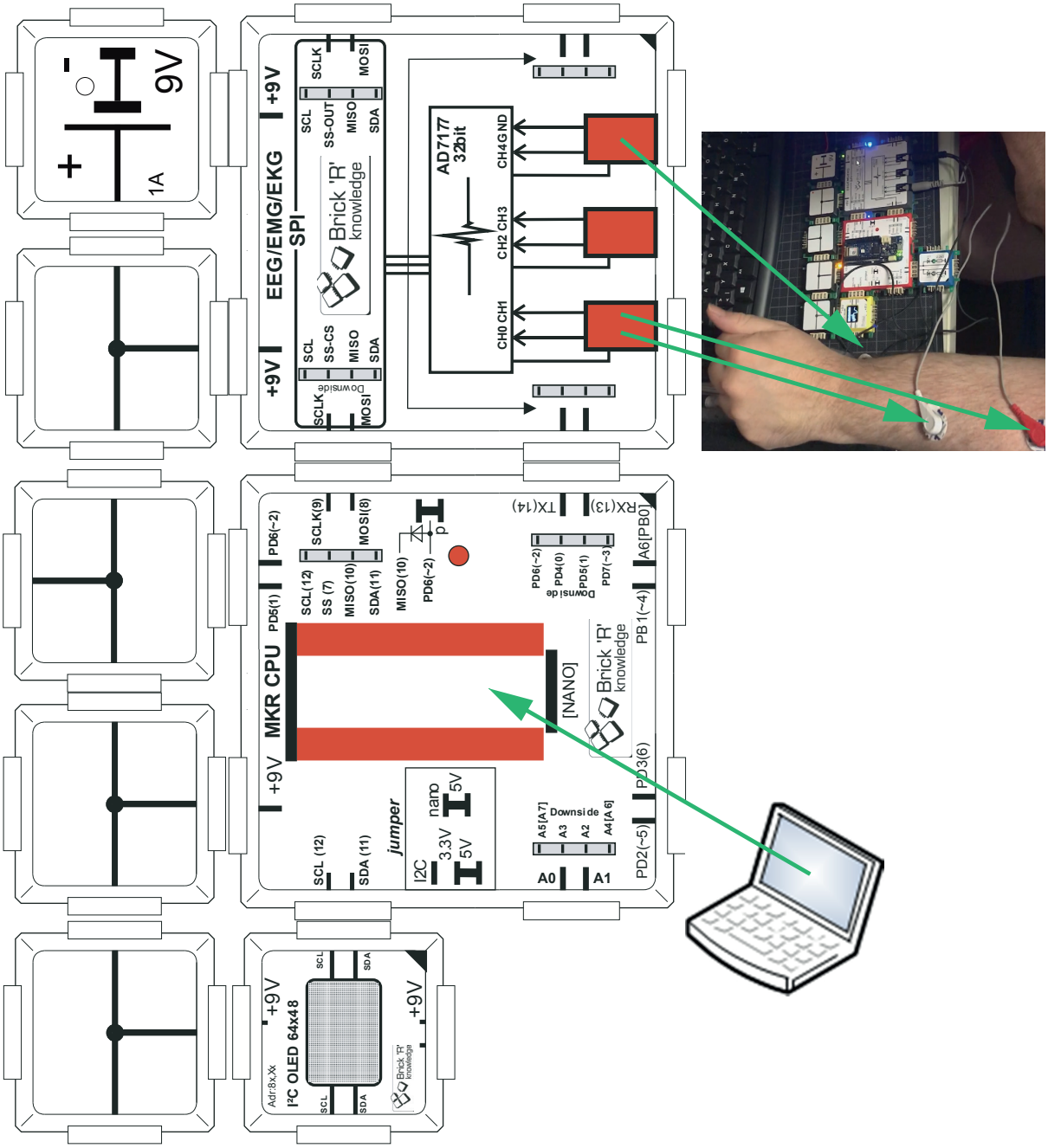


7.8 EMG Messung von Muskel-Strömen (Muskelsteuerung)



Elektroden CH0 und CH1 über dem Muskel platzieren und die Masse an der Oberseite der Hand anbringen.

Nun kann man die Hand auf- und ab-bewegen, dadurch werden kräftige Ausschläge angezeigt. Dem Leser sei es überlassen, die beiden LEDs in Abhängigkeit von Amplitude oder Frequenz auszulösen, um so zum Beispiel eine Steuerung auszulösen.



```
// EEG BRICK
// MKR WIFI 2010 __SAMD21G18A__
// Output to serial Interface !!
// EMG BEISPIEL -- wie EKG ---



// I2C Select: clken selen mosien led0 sense1 sense2 sense3 spez
//             0     1      2      3     4     5      6      7
//             out   out    out   out   in     in     in     in
//

#include <Wire.h> // Definitionen laden fuer I2C
#include <SPI.h>
#include <avr/pgmspace.h> // weitere Definitionen

// USe IOs of MKR
#ifdef ARDUINO_SAMD_MKRWIFI1010
#define __AVR_SAMD21__   1

#endif


// Alle Adressen bei den 8574xx bricks:
#define i2cIO9534_0 (0x40>>1) // Trick um elegant mit Bytes
#define i2cIO9534_1 (0x42>>1) // zu arbeiten statt in 7 Bit
#define i2cIO9534_2 (0x44>>1) // das letzte Bit ist das R/W
#define i2cIO9534_3 (0x46>>1) // dass von der Arduino
#define i2cIO9534_4 (0x48>>1) // Bibliothek dazugefuegt wird
#define i2cIO9534_5 (0x4A>>1) // wir definieren hier alle
#define i2cIO9534_6 (0x4C>>1) // Bereiche die man mit den
#define i2cIO9534_7 (0x4E>>1) // Bricks einstellen kann,.

#define i2cIO9534A_0 (0x70>>1) // Die Serie PCF8474AT
#define i2cIO9534A_1 (0x72>>1) // beginnt bei Adresse
#define i2cIO9534A_2 (0x74>>1) // 0x70 = 70 sedezimal
#define i2cIO9534A_3 (0x76>>1)  // 01110000 binaer
#define i2cIO9534A_4 (0x78>>1) // oder intern 0111000
#define i2cIO9534A_5 (0x7A>>1) // dabei 0111xxx
#define i2cIO9534A_6 (0x7C>>1) // mit x fuer die
#define i2cIO9534A_7 (0x7E>>1) // Dilschalterposition

#define myi2cIOadr i2cIO9534_0 // HIER PASSENDE ADRESSE EINTRAGEN

#define SSPB 10 // Pin 10 = SS  - --- PB2  1<<2 als Maske

// SPI: SCLK MOSI MISO und SS Brick=PB2 D10 fest !!
// MEGA Pin 51=MOSI Pin 52=SCK und Pin50=MISO °!!!
// mega:
// NANO: MOSI:PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// UNO: MOSI: PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// ACHUNG: MISO ist auch STatus DATA READY !!!
//
// PORTB 4  (7 6 5 4 -- 3 2 1 0)
// BEI MKR PD6

#if defined(__AVR_ATmega2560__)  // Arduino MEGA2560
#define MISOPIN 50
#warning ("INFO : compiles for AT MEGA2560")
#elif defined(__AVR_SAMD21__)
#warning ("INFO : compiles for MKR WIFI1010")
#define MISOPIN 2  // READ ONLY MKR ueber Diode...
#else // NANO
#warning ("INFO : compiles rest: also NANO...")
#define MISOPIN 12
#endif

// #define MISOPIN 50

long p_disp[64];


// DE_27 OLED Beispiele - Pixelroutinen
```

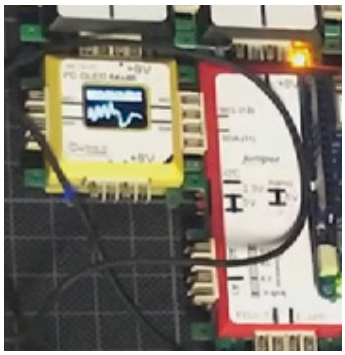
```
#include <avr/pgmspace.h> // Zugriff ins ROM

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

// -------------------------OLED ---------------------------------------------
// 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 enabless()
{
  //digitalWrite(SSPB, LOW);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, LOW);
  #else
   PORTB &= 0xfb; // Default = high 7 6 5 4 3 2low 1 0 PB2 low
  #endif
  delayMicroseconds(10);
}
void disabless()
{
  delayMicroseconds(10);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, HIGH);
  #else
    PORTB |= 1 << 2; // Default = high bit2 PB2
  #endif
  // digitalWrite(SSPB, HIGH);
  delayMicroseconds(10);
  //
}

//AD 7177-x
// Register
// SS:-------_________________----------

//
unsigned short ad71_readid()
{
  unsigned char ch1;
  unsigned short val1=0;
  enabless();
  ch1 = SPI.transfer(0x47); // COmm: -wen r-w ra5..9  r-1=1 read 0=writ
  val1 = SPI.transfer16(0); // ID
  disabless();
  return(val1);
}

//

//
#define NOCHAN 1

void ad71_init()
{
   enabless();
   // Register setzen...
   SPI.transfer(0x01); SPI.transfer16(0x800C);  // ADCMODE ! ref en, mode continouus, clock
Crystal : 100 00 000  0 000 11 00
   SPI.transfer(0x02); SPI.transfer16(0x0442);  // INTERFACE MOD data_stat(6)en w32 en --- 000
0   0 10 0    0 1   0 0  00 1 0  old 0x0042
   SPI.transfer(0x06); SPI.transfer16(0x060d);  // GPIO 000 0  0 11 0   0 0 00 1101
   // Channel 0
   SPI.transfer(0x10); SPI.transfer16(0x8001);  // CH0   enable set0 ain0 ain1 ... 10 00 00 00
000 0 0001
   // Channel 1
   SPI.transfer(0x11); SPI.transfer16(0x0043);  // CH1   enable set0 ain2 ain3 ... 10 00 00 00
010 0 0011
   // Channel 2
   SPI.transfer(0x12); SPI.transfer16(0x0096);  // CH2   enable set0 ain4 ref- ... 10 00 00 00
100 1 0110
   // Channel 3  optional
```

```
   SPI.transfer(0x13); SPI.transfer16(0x0232);  // CH3   enable set0 smp+ tmp- ... 10 00 00 10
001 1 0010
   // alle gleiches setup daher setup 0
   // SETup registers.
   SPI.transfer(0x20); SPI.transfer16(0x1300);  // SET0 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x21); SPI.transfer16(0x1300);  // SET1 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x22); SPI.transfer16(0x1300);  // SET2 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x23); SPI.transfer16(0x1300);  // SET3 : 000 1  0 0 1 1  0 0 00 0000
   // Filter Config gehoeren zu setup reg
   SPI.transfer(0x28); SPI.transfer16(0x050e);  // FLT0 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x29); SPI.transfer16(0x050a);  // FLT1 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2A); SPI.transfer16(0x050a);  // FLT2 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2B); SPI.transfer16(0x050a);  // FLT3 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   // OFFSET und GAIN auf factory lassen...
   disabless();

}

unsigned long values[4];

// tmperature 470uV/K
//         1181972248 beispiel bei 2.5V Ref+- 0.137mV 292K ->19.7
// 67BCCE0 108776672 -3.5


void ad71_reset()
{
  enabless();
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  disabless();
  delayMicroseconds(550);
}


void ad71_dataread()
{
  // Wenn ready incl status 4 byts 32 bit
  // STATUS
  unsigned char sts;
  unsigned char ch1;
  unsigned long val1=0;
  unsigned long val2=0;
  unsigned int idx;
  static int first = 0;
  int val=0;
  int i=0;
  for (i=0; i<NOCHAN;i++) {
    // SYNC:
    // NICHT RDY Abfragen,,,
  #ifdef XXXXXX
    enabless();
    do {

      sts = SPI.transfer(0x40);
      sts = SPI.transfer(0x00);
       delayMicroseconds(10);
       // Serial.print(sts);
       // Serial.println("S wt:");

    } while (((sts & 0x80)== 0x80) && (first == 1));
    first = 1;
    disabless();
#endif
```

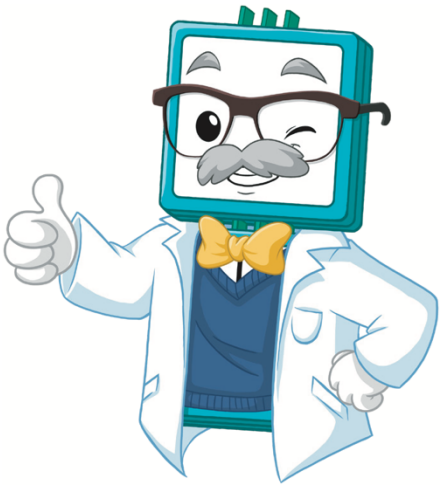
```
    // DATA RDY
    delayMicroseconds(10);
    //
    enabless();
    delayMicroseconds(4); // DRDY valid...
    // SINGLE MODE SPI.transfer(0x01); SPI.transfer16(0x801C);  // ADCMODE ! ref en, mode
SINGLE, clock Crystal : 100 00 000  0 000 11 00
    //
    // Sonst abfragen hier immer...
    do {
      val = digitalRead(MISOPIN); // MISOPIN
    } while (val == HIGH);

    //
    ch1 = SPI.transfer(0x44); // COmm:  data read

    //
    val1 = SPI.transfer16(0); // data <
    val2 = SPI.transfer16(0); // data
    //

     //
    delayMicroseconds(5); // DRDY valid...
    disabless();
    //
    val1 = val2 | (val1 << 16); // 4 bit status
    idx = val2 & 0x3; // CHANNEL FLG  bit 3210
    values[idx] = val1 & 0xfffffff0;

  }
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("Starting EEG:");

  #if defined(__AVR_SAMD21__)
   pinMode(MISOPIN, INPUT); // MISO !!
   pinMode(7, OUTPUT);
  #else
   pinMode(12, INPUT); // MISO !!
   pinMode(SSPB, OUTPUT);
  #endif
  disabless();



  Wire.begin(); // I2C aktivieren !
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x03) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0xF0) ;  // 1=input
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0x0f) ;  // led an !!
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  i2c_oled_initall(i2coledssd);
  disp_buffer_clear(COLOR_BLACK);
  // INIT
   SPI.begin();
  SPI.beginTransaction(SPISettings(7000000, MSBFIRST, SPI_MODE3));
  // NICHT VERWENDEN:
  //SPI.setDataMode(SPI_MODE3);  // sonst immer 1     clk ---____----   rising edhe = spi mode
3 krit mkr´
  //SPI.setClockDivider(8);
 disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
 disp_print_xy_lcd(0, 0, (unsigned char*)"EEG/EKG/EMG", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);
  disp_print_xy_lcd(20, 20, (unsigned char*)"Starte...", COLOR_WHITE, 0);
  Serial.println("Searching EEG:");
  disp_lcd_frombuffer();
  unsigned short id=ad71_readid(); // 0x4fdx
```

```
  while ((id & 0xfff0)!= 0x4fd0) {
     Serial.print("ERROR id=");Serial.print(id);
     Serial.println(" EEG Brick not found");
     disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
      disp_print_xy_lcd(0, 15, (unsigned char*)"NOT FOUND", COLOR_WHITE, 0);
      disp_lcd_frombuffer();
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x07) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x0f) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     id=ad71_readid(); // 0x4fdx
  }
  Serial.print("id=");Serial.println(id);
  // OK Init dr Register
  ad71_reset();
  ad71_init();

//Init display schlange
   for(int x = 0; x <=63; x++){
        p_disp[x] = 0;
        }


}



int findMin(){
  long minimum = p_disp[0];
  int c;
  for (c = 0; c < 62; c++)
    {
        if (p_disp[c] < minimum)
        {
           minimum = p_disp[c];
        }
    }
  return minimum;

  }

int findMax(){
  long maximum = p_disp[0];
  int c;
  for (c = 0; c < 62; c++)
    {
        if (p_disp[c] > maximum)
        {
           maximum = p_disp[c];
        }
    }
  return maximum;

  }

long lastval=0;
void loop() {


ad71_dataread();
 #ifdef ALLL
    Serial.print("ad:");Serial.print(values[0],HEX);
    Serial.print(" ");Serial.print(values[1],HEX);
    Serial.print(" ");Serial.print(values[2],HEX);
    Serial.print(" ");Serial.print(values[3],HEX);
    Serial.println();
 #endif
```

```
  for (int it = 63; it >= 1; it--)
  {
  p_disp[it] = p_disp[it-1];
  }
  long v1 = values[0]-0x800000001;
   p_disp[0] = v1;

 long  minim = findMin();
 long maxim = findMax();

   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
 disp_print_xy_lcd(0, 0, (unsigned char*)"EMG", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);
 for (int i=0; i<63;i++) { // nun fuer alle 64 Spalten (x)  des Displays
  long val = 47-(map(p_disp[i], minim, maxim, 0, 30));
    disp_setpixel(i, val , COLOR_WHITE); // koordinate 0,0 ist links oben daher 47-y
    //OLDX NEWX OLDY NEWY
    disp_line_lcd (i-1,lastval,i,val, COLOR_WHITE);
    lastval = val;
  } // alle Spalten
 disp_lcd_frombuffer(); // dann erst updaten, double buffer flimmerfrei


  // long v2 = values[1]-0x800000001;
 //Serial.print(v1);Serial.print(",");Serial.println(v2);
 Serial.println(v1);
}
```



Was passiert? Die Messwerte bzw. die grafische Ausgabe verändert sich, wenn man die Muskeln anspannt.

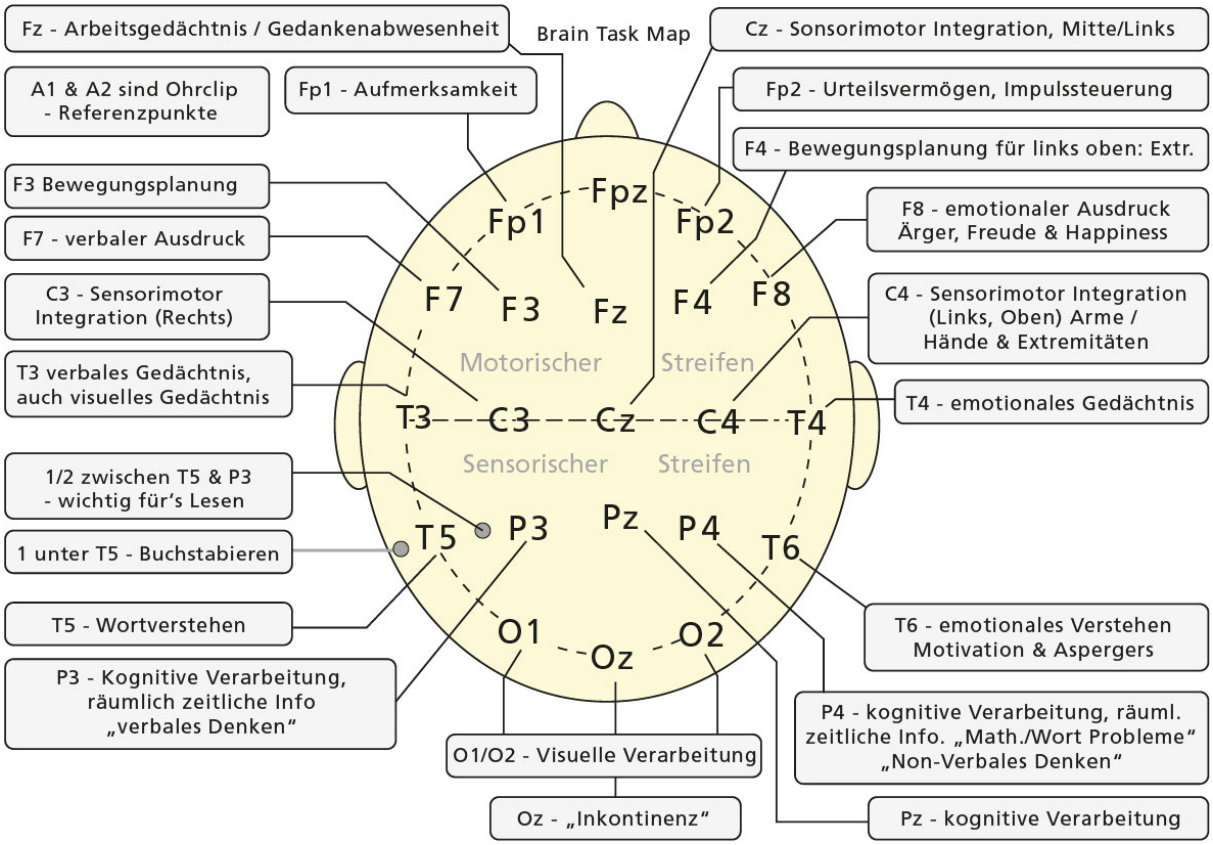


8. Gehirnströme Messung

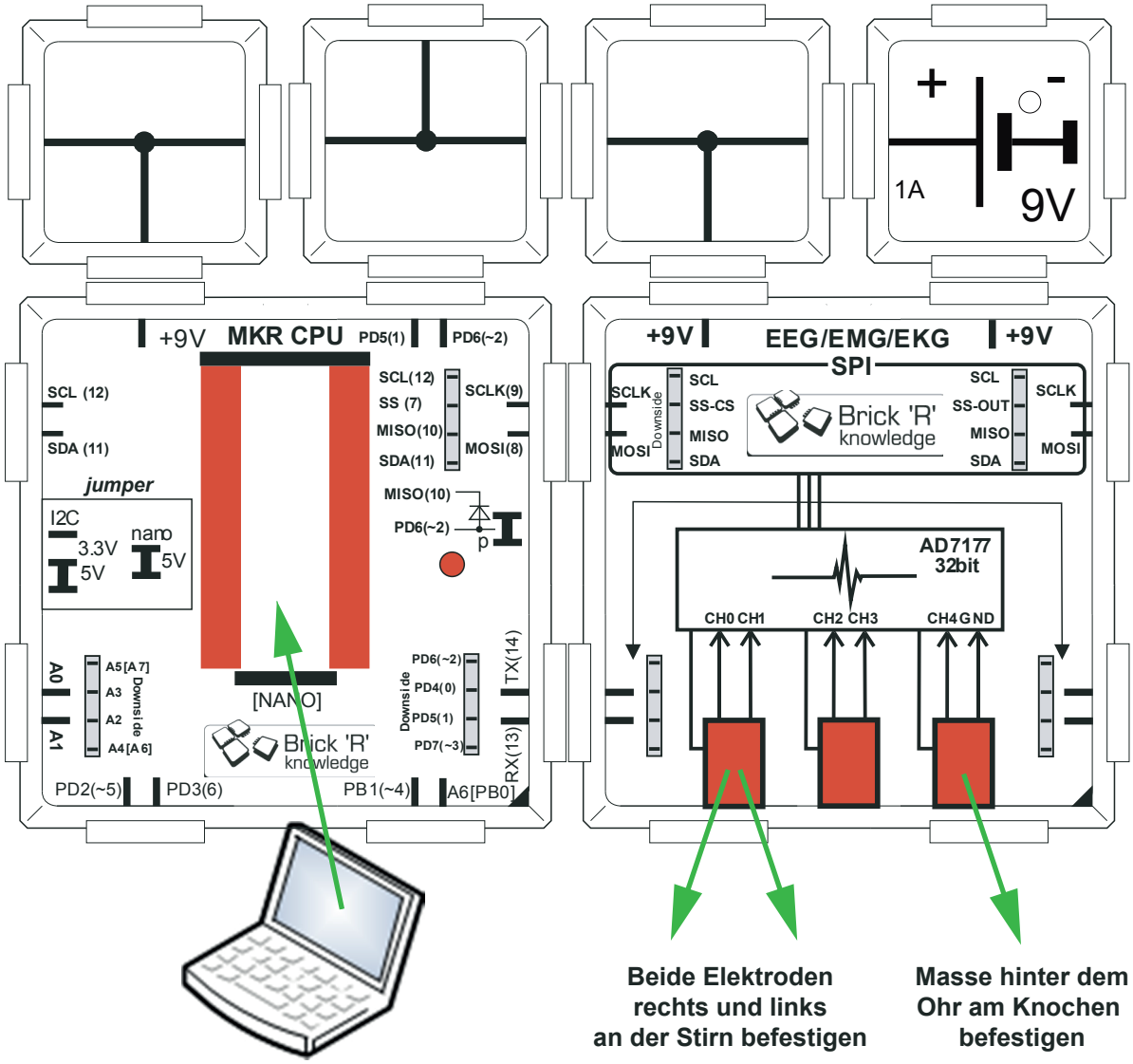
8.1 EEG Messung grafisch am Terminal ausgeben

Die beiden Elektroden von CH0 und CH1 an der Stirn links und rechts befestigen, die Masse bei CH4/GND hinter dem Ohrläppchen. Man kann dort auch beide Elektroden anbringen.

Alternativ besteht auch die Möglichkeit, andere Punkte an Hand des folgenden Diagramms auszusuchen:



Dann ruhige Position einnehmen. Den Versuch sollte man am besten zu zweit durchführen. Ausgabe auf dem grafischen Terminal (serieller Plotter).



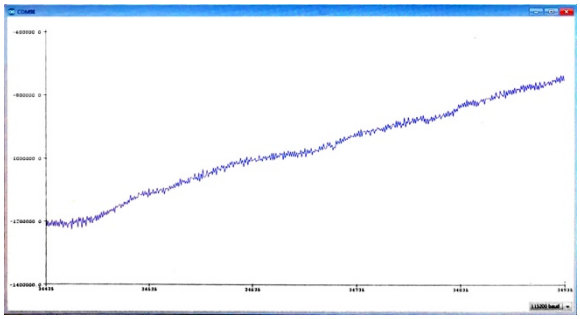

```
// EEG Gehirnstrom-Messung
//  MKR WIFI 2010 __SAMD21G18A__
// Output to serial Interface !!


// I2C Select: clken selen mosien led0 sense1 sense2 sense3 spez
//             0     1      2      3     4      5      6      7
//             out   out    out    out   in     in     in     in
//

#include <Wire.h> // Definitionen laden fuer I2C
#include <SPI.h>
#include <avr/pgmspace.h> // weitere Definitionen

// USe IOs of MKR
#ifdef ARDUINO_SAMD_MKRWIFI1010
#define __AVR_SAMD21__    1

#endif


// Alle Adressen bei den 8574xx bricks:
#define i2cIO9534_0 (0x40>>1) // Trick um elegant mit Bytes
#define i2cIO9534_1 (0x42>>1) // zu arbeiten statt in 7 Bit
#define i2cIO9534_2 (0x44>>1) // das letzte Bit ist das R/W
#define i2cIO9534_3 (0x46>>1) // dass von der Arduino
```

```
#define i2cIO9534_4 (0x48>>1) // Bibliothek dazugefuegt wird
#define i2cIO9534_5 (0x4A>>1) // wir definieren hier alle
#define i2cIO9534_6 (0x4C>>1) // Bereiche die man mit den
#define i2cIO9534_7 (0x4E>>1) // Bricks einstellen kann,.

#define i2cIO9534A_0 (0x70>>1) // Die Serie PCF8474AT
#define i2cIO9534A_1 (0x72>>1) // beginnt bei Adresse
#define i2cIO9534A_2 (0x74>>1) // 0x70 = 70 sedezimal
#define i2cIO9534A_3 (0x76>>1)  // 01110000 binaer
#define i2cIO9534A_4 (0x78>>1) // oder intern 0111000
#define i2cIO9534A_5 (0x7A>>1) // dabei 0111xxx
#define i2cIO9534A_6 (0x7C>>1) // mit x fuer die
#define i2cIO9534A_7 (0x7E>>1) // Dilschalterposition

#define myi2cIOadr i2cIO9534_0 // HIER PASSENDE ADRESSE EINTRAGEN

#define SSPB 10 // Pin 10 = SS  - --- PB2  1<<2 als Maske

// SPI: SCLK MOSI MISO und SS Brick=PB2 D10 fest !!
// MEGA Pin 51=MOSI Pin 52=SCK und Pin50=MISO °!!!
// mega:
// NANO: MOSI:PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// UNO: MOSI: PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// ACHUNG: MISO ist auch STatus DATA READY !!!
//
// PORTB 4  (7 6 5 4 -- 3 2 1 0)
// BEI MKR PD6

#if defined(__AVR_ATmega2560__)  // Arduino MEGA2560
#define MISOPIN 50
#warning ("INFO : compiles for AT MEGA2560")
#elif defined(__AVR_SAMD21__)
#warning ("INFO : compiles for MKR WIFI1010")
#define MISOPIN 2  // READ ONLY MKR ueber Diode...
#else // NANO
#warning ("INFO : compiles rest: also NANO...")
#define MISOPIN 12
#endif

// #define MISOPIN 50

void enabless()
{
  //digitalWrite(SSPB, LOW);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, LOW);
  #else
   PORTB &= 0xfb; // Default = high 7 6 5 4 3 2low 1 0 PB2 low
  #endif
  delayMicroseconds(10);
}
void disabless()
{
  delayMicroseconds(10);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, HIGH);
  #else
    PORTB |= 1 << 2; // Default = high bit2 PB2
  #endif
  // digitalWrite(SSPB, HIGH);
  delayMicroseconds(10);
  //
}

//AD 7177-x
// Register
// SS:-------_________________----------

//
unsigned short ad71_readid()
{
  unsigned char ch1;
  unsigned short val1=0;
  enabless();
```

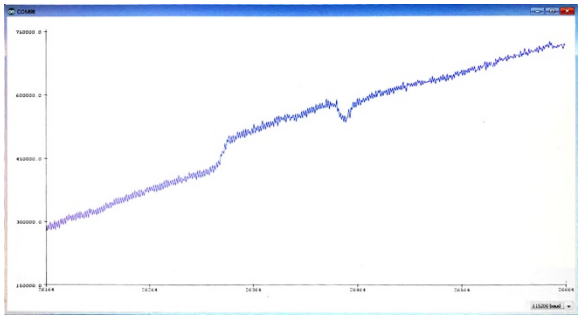
```
  ch1 = SPI.transfer(0x47); // COmm: -wen r-w ra5..9  r-1=1 read 0=writ
  val1 = SPI.transfer16(0); // ID
  disabless();
  return(val1);
}

//

//
#define NOCHAN 1

void ad71_init()
{
   enabless();
   // Register setzen...
   SPI.transfer(0x01); SPI.transfer16(0x800C);  // ADCMODE ! ref en, mode continouus, clock
Crystal : 100 00 000  0 000 11 00
   SPI.transfer(0x02); SPI.transfer16(0x0442);  // INTERFACE MOD data_stat(6)en w32 en --- 000
0   0 10 0    0 1   0 0  00 1 0  old 0x0042
   SPI.transfer(0x06); SPI.transfer16(0x060d);  // GPIO 000 0  0 11 0   0 0 00 1101
   // Channel 0
   SPI.transfer(0x10); SPI.transfer16(0x8001);  // CH0   enable set0 ain0 ain1 ... 10 00 00 00
000 0 0001
   // Channel 1
   SPI.transfer(0x11); SPI.transfer16(0x0043);  // CH1   enable set0 ain2 ain3 ... 10 00 00 00
010 0 0011
   // Channel 2
   SPI.transfer(0x12); SPI.transfer16(0x0096);  // CH2   enable set0 ain4 ref- ... 10 00 00 00
100 1 0110
   // Channel 3  optional
   SPI.transfer(0x13); SPI.transfer16(0x0232);  // CH3   enable set0 smp+ tmp- ... 10 00 00 10
001 1 0010
   // alle gleiches setup daher setup 0
   // SETup registers.
   SPI.transfer(0x20); SPI.transfer16(0x1300);  // SET0 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x21); SPI.transfer16(0x1300);  // SET1 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x22); SPI.transfer16(0x1300);  // SET2 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x23); SPI.transfer16(0x1300);  // SET3 : 000 1  0 0 1 1  0 0 00 0000
   // Filter Config gehoeren zu setup reg
   SPI.transfer(0x28); SPI.transfer16(0x050e);  // FLT0 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x29); SPI.transfer16(0x050a);  // FLT1 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2A); SPI.transfer16(0x050a);  // FLT2 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2B); SPI.transfer16(0x050a);  // FLT3 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   // OFFSET und GAIN auf factory lassen...
   disabless();

}

unsigned long values[4];

// tmperature 470uV/K
//         1181972248 beispiel bei 2.5V Ref+- 0.137mV 292K ->19.7
// 67BCCE0 108776672 -3.5


void ad71_reset()
{
  enabless();
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  disabless();
  delayMicroseconds(550);
}
```

```
void ad71_dataread()
{
  // Wenn ready incl status 4 byts 32 bit
  // STATUS
  unsigned char sts;
  unsigned char ch1;
  unsigned long val1=0;
  unsigned long val2=0;
  unsigned int idx;
  static int first = 0;
  int val=0;
  int i=0;
  for (i=0; i<NOCHAN;i++) {
    // SYNC:
    // NICHT RDY Abfragen,,,
  #ifdef XXXXXX
    enabless();
    do {

      sts = SPI.transfer(0x40);
      sts = SPI.transfer(0x00);
       delayMicroseconds(10);
       // Serial.print(sts);
       // Serial.println("S wt:");

    } while (((sts & 0x80)== 0x80) && (first == 1));
    first = 1;
    disabless();
#endif

    // DATA RDY
    delayMicroseconds(10);
    //
    enabless();
    delayMicroseconds(4); // DRDY valid...
    // SINGLE MODE SPI.transfer(0x01); SPI.transfer16(0x801C);  // ADCMODE ! ref en, mode
SINGLE, clock Crystal : 100 00 000  0 000 11 00
    //
    // Sonst abfragen hier immer...
    do {
      val = digitalRead(MISOPIN); // MISOPIN
    } while (val == HIGH);

    //
    ch1 = SPI.transfer(0x44); // COmm:  data read

    //
    val1 = SPI.transfer16(0); // data <
    val2 = SPI.transfer16(0); // data
    //

     //
    delayMicroseconds(5); // DRDY valid...
    disabless();
    //
    val1 = val2 | (val1 << 16); // 4 bit status
    idx = val2 & 0x3; // CHANNEL FLG  bit 3210
    values[idx] = val1 & 0xfffffff0;

  }
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("Starting EEG:");

  #if defined(__AVR_SAMD21__)
   pinMode(MISOPIN, INPUT); // MISO !!
   pinMode(7, OUTPUT);
  #else
   pinMode(12, INPUT); // MISO !!
   pinMode(SSPB, OUTPUT);
```

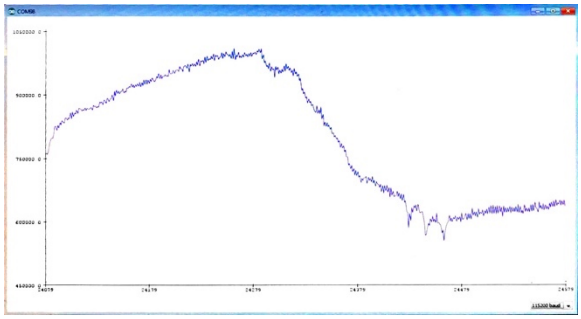
```
  #endif
  disableSS();



  Wire.begin(); // I2C aktivieren !
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x03) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0xF0) ;  // 1=input
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0x0f) ;  // led an !!
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  // INIT
   SPI.begin();
  SPI.beginTransaction(SPISettings(7000000, MSBFIRST, SPI_MODE3));
  // NICHT VERWENDEN:
  //SPI.setDataMode(SPI_MODE3);  // sonst immer 1     clk ---____----   rising edhe = spi mode
3 krit mkr´
  //SPI.setClockDivider(8);

  Serial.println("Searching EEG:");
  unsigned short id=ad71_readid(); // 0x4fdx
  while ((id & 0xfff0)!= 0x4fd0) {
     Serial.print("ERROR id=");Serial.print(id);
     Serial.println(" EEG Brick not found");
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x07) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x0f) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     id=ad71_readid(); // 0x4fdx
  }
  Serial.print("id=");Serial.println(id);
  // OK Init dr Register
  ad71_reset();
  ad71_init();
}

void loop() {


ad71_dataread();
#ifdef ALLL
  Serial.print("ad:");Serial.print(values[0],HEX);
  Serial.print(" ");Serial.print(values[1],HEX);
  Serial.print(" ");Serial.print(values[2],HEX);
  Serial.print(" ");Serial.print(values[3],HEX);
  Serial.println();
#endif
  long v1 = values[0]-0x800000001;
  // long v2 = values[1]-0x800000001;
 //Serial.print(v1);Serial.print(",");Serial.println(v2);
 Serial.println(v1);
}
```



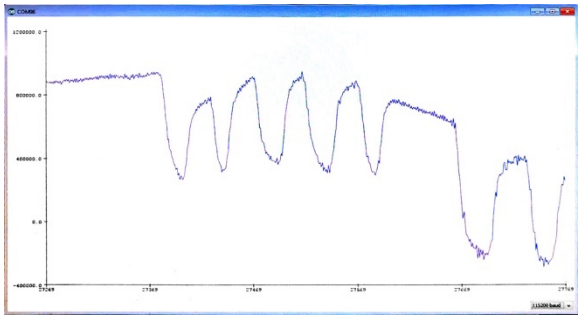
Ruhe-EEG mit geschlossenen Augen



Ruhe-EEG

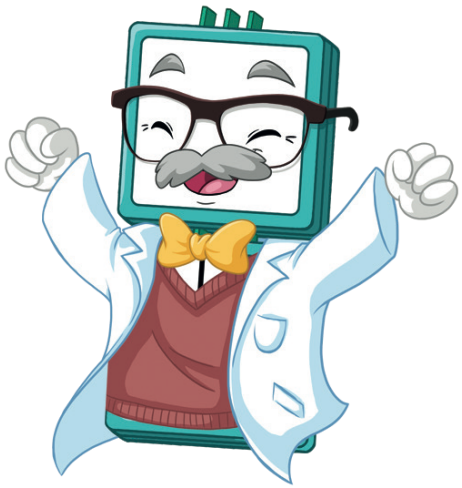


Drei mal mit den Augen blinzeln



Mund auf und Mund zu

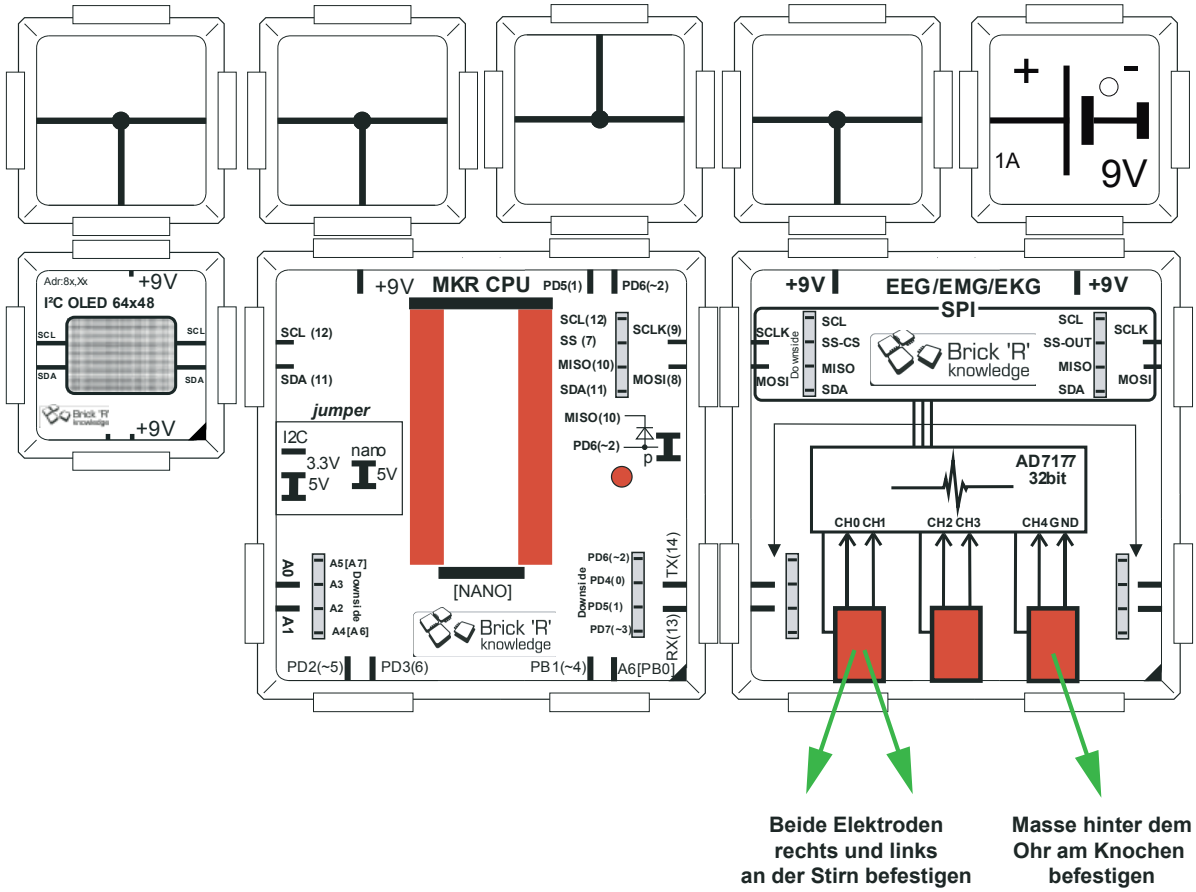
Was passiert? Die Messwerte, die Gehirnströme werden auf dem seriellen Plotter ausgegeben und sind unterschiedlich, je nachdem, was man gerade macht.



8.2 EEG Messung grafisch am OLED-Display ausgeben

Die beiden Elektroden von CH0 und CH1 an der Stirn links und rechts befestigen, die Masse bei CH4/GND hinter dem Ohrläppchen. Man kann dort auch beide Elektroden anbringen.

Dann ruhige Position einnehmen. Den Versuch sollte man am besten zu zweit durchführen. Ausgabe auf dem grafischen OLED-Display.




```
// EEG Gehirnstrom-Messung
//  MKR WIFI 2010 __SAMD21G18A__
// Output to serial Interface !!


// I2C Select: clken selen mosien led0 sense1 sense2 sense3 spez
//             0     1      2      3     4      5      6      7
//             out   out    out    out   in     in     in     in
//

#include <Wire.h> // Definitionen laden fuer I2C
#include <SPI.h>
#include <avr/pgmspace.h> // weitere Definitionen

// USe IOs of MKR
#ifdef ARDUINO_SAMD_MKRWIFI1010
#define __AVR_SAMD21__   1

#endif


// Alle Adressen bei den 8574xx bricks:
#define i2cIO9534_0 (0x40>>1) // Trick um elegant mit Bytes
#define i2cIO9534_1 (0x42>>1) // zu arbeiten statt in 7 Bit
#define i2cIO9534_2 (0x44>>1) // das letzte Bit ist das R/W
#define i2cIO9534_3 (0x46>>1) // dass von der Arduino
#define i2cIO9534_4 (0x48>>1) // Bibliothek dazugefuegt wird
```

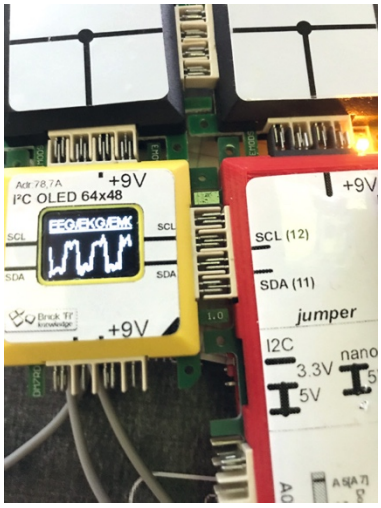
```
#define i2cIO9534_5 (0x4A>>1) // wir definieren hier alle
#define i2cIO9534_6 (0x4C>>1) // Bereiche die man mit den
#define i2cIO9534_7 (0x4E>>1) // Bricks einstellen kann,.

#define i2cIO9534A_0 (0x70>>1) // Die Serie PCF8474AT
#define i2cIO9534A_1 (0x72>>1) // beginnt bei Adresse
#define i2cIO9534A_2 (0x74>>1) // 0x70 = 70 sedezimal
#define i2cIO9534A_3 (0x76>>1)  // 01110000 binaer
#define i2cIO9534A_4 (0x78>>1) // oder intern 0111000
#define i2cIO9534A_5 (0x7A>>1) // dabei 0111xxx
#define i2cIO9534A_6 (0x7C>>1) // mit x fuer die
#define i2cIO9534A_7 (0x7E>>1) // Dilschalterposition

#define myi2cIOadr i2cIO9534_0 // HIER PASSENDE ADRESSE EINTRAGEN

#define SSPB 10 // Pin 10 = SS  - --- PB2  1<<2 als Maske

// SPI: SCLK MOSI MISO und SS Brick=PB2 D10 fest !!
// MEGA Pin 51=MOSI Pin 52=SCK und Pin50=MISO °!!!
// mega:
// NANO: MOSI:PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// UNO: MOSI: PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// ACHUNG: MISO ist auch STatus DATA READY !!!
//
// PORTB 4  (7 6 5 4 -- 3 2 1 0)
// BEI MKR PD6

#if defined(__AVR_ATmega2560__)  // Arduino MEGA2560
#define MISOPIN 50
#warning ("INFO : compiles for AT MEGA2560")
#elif defined(__AVR_SAMD21__)
#warning ("INFO : compiles for MKR WIFI1010")
#define MISOPIN 2  // READ ONLY MKR ueber Diode...
#else // NANO
#warning ("INFO : compiles rest: also NANO...")
#define MISOPIN 12
#endif

// #define MISOPIN 50

long p_disp[64];


// 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 ---------------------------------------------
// 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 enabless()
{
  //digitalWrite(SSPB, LOW);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, LOW);
  #else
   PORTB &= 0xfb; // Default = high 7 6 5 4 3 2low 1 0 PB2 low
  #endif
  delayMicroseconds(10);
}
void disabless()
{
  delayMicroseconds(10);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, HIGH);
  #else
    PORTB |= 1 << 2; // Default = high bit2 PB2
  #endif
```

```
  // digitalWrite(SSPB, HIGH);
  delayMicroseconds(10);
  //
}

//AD 7177-x
// Register
// SS:-------_________________----------

//
unsigned short ad71_readid()
{
  unsigned char ch1;
  unsigned short val1=0;
  enabless();
  ch1 = SPI.transfer(0x47); // COmm: -wen r-w ra5..9  r-1=1 read 0=writ
  val1 = SPI.transfer16(0); // ID
  disabless();
  return(val1);
}

//

//
#define NOCHAN 1

void ad71_init()
{
   enabless();
   // Register setzen...
   SPI.transfer(0x01); SPI.transfer16(0x800C);  // ADCMODE ! ref en, mode continouus, clock
Crystal : 100 00 000  0 000 11 00
   SPI.transfer(0x02); SPI.transfer16(0x0442);  // INTERFACE MOD data_stat(6)en w32 en --- 000
0   0 10 0    0 1   0 0  00 1 0  old 0x0042
   SPI.transfer(0x06); SPI.transfer16(0x060d);  // GPIO 000 0  0 11 0   0 0 00 1101
   // Channel 0
   SPI.transfer(0x10); SPI.transfer16(0x8001);  // CH0   enable set0 ain0 ain1 ... 10 00 00 00
000 0 0001
   // Channel 1
   SPI.transfer(0x11); SPI.transfer16(0x0043);  // CH1   enable set0 ain2 ain3 ... 10 00 00 00
010 0 0011
   // Channel 2
   SPI.transfer(0x12); SPI.transfer16(0x0096);  // CH2   enable set0 ain4 ref- ... 10 00 00 00
100 1 0110
   // Channel 3  optional
   SPI.transfer(0x13); SPI.transfer16(0x0232);  // CH3   enable set0 smp+ tmp- ... 10 00 00 10
001 1 0010
   // alle gleiches setup daher setup 0
   // SETup registers.
   SPI.transfer(0x20); SPI.transfer16(0x1300);  // SET0 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x21); SPI.transfer16(0x1300);  // SET1 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x22); SPI.transfer16(0x1300);  // SET2 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x23); SPI.transfer16(0x1300);  // SET3 : 000 1  0 0 1 1  0 0 00 0000
   // Filter Config gehoeren zu setup reg
   SPI.transfer(0x28); SPI.transfer16(0x050e);  // FLT0 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x29); SPI.transfer16(0x050a);  // FLT1 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2A); SPI.transfer16(0x050a);  // FLT2 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2B); SPI.transfer16(0x050a);  // FLT3 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   // OFFSET und GAIN auf factory lassen...
   disabless();

}

unsigned long values[4];

// tmperature 470uV/K
//         1181972248 beispiel bei 2.5V Ref+- 0.137mV 292K ->19.7
// 67BCCE0 108776672 -3.5
```

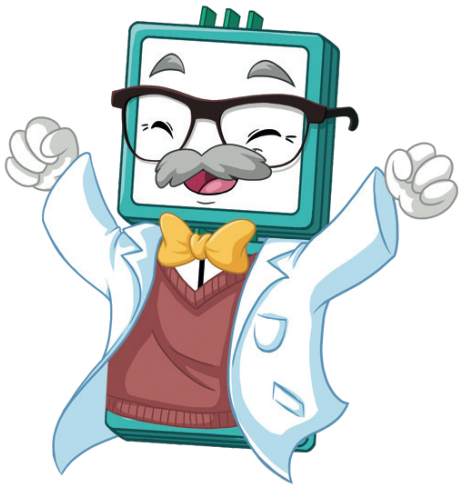
```
void ad71_reset()
{
  enabless();
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  disabless();
  delayMicroseconds(550);
}


void ad71_dataread()
{
  // Wenn ready incl status 4 byts 32 bit
  // STATUS
  unsigned char sts;
  unsigned char ch1;
  unsigned long val1=0;
  unsigned long val2=0;
  unsigned int idx;
  static int first = 0;
  int val=0;
  int i=0;
  for (i=0; i<NOCHAN;i++) {
    // SYNC:
    // NICHT RDY Abfragen,,,
  #ifdef XXXXXX
    enabless();
    do {

      sts = SPI.transfer(0x40);
      sts = SPI.transfer(0x00);
       delayMicroseconds(10);
       // Serial.print(sts);
       // Serial.println("S wt:");

    } while (((sts & 0x80)== 0x80) && (first == 1));
    first = 1;
    disabless();
#endif

    // DATA RDY
    delayMicroseconds(10);
    //
    enabless();
    delayMicroseconds(4); // DRDY valid...
    // SINGLE MODE SPI.transfer(0x01); SPI.transfer16(0x801C);  // ADCMODE ! ref en, mode
SINGLE, clock Crystal : 100 00 000  0 000 11 00
    //
    // Sonst abfragen hier immer...
    do {
      val = digitalRead(MISOPIN); // MISOPIN
    } while (val == HIGH);

    //
    ch1 = SPI.transfer(0x44); // COmm:  data read

    //
    val1 = SPI.transfer16(0); // data <
    val2 = SPI.transfer16(0); // data
    //

     //
    delayMicroseconds(5); // DRDY valid...
    disabless();
    //
    val1 = val2 | (val1 << 16); // 4 bit status
    idx = val2 & 0x3; // CHANNEL FLG  bit 3210
    values[idx] = val1 & 0xfffffff0;
```

```
  }
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("Starting EEG:");

  #if defined(__AVR_SAMD21__)
   pinMode(MISOPIN, INPUT); // MISO !!
   pinMode(7, OUTPUT);
  #else
   pinMode(12, INPUT); // MISO !!
   pinMode(SSPB, OUTPUT);
  #endif
  disabless();



  Wire.begin(); // I2C aktivieren !
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x03) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0xF0) ;  // 1=input
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0x0f) ;  // led an !!
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  i2c_oled_initall(i2coledssd);
  disp_buffer_clear(COLOR_BLACK);
  // INIT
   SPI.begin();
  SPI.beginTransaction(SPISettings(7000000, MSBFIRST, SPI_MODE3));
  // NICHT VERWENDEN:
  //SPI.setDataMode(SPI_MODE3);  // sonst immer 1     clk ---____----   rising edhe = spi mode
3 krit mkr´
  //SPI.setClockDivider(8);
 disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
 disp_print_xy_lcd(0, 0, (unsigned char*)"EEG/EKG/EMG", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);
  disp_print_xy_lcd(20, 20, (unsigned char*)"Starte...", COLOR_WHITE, 0);
  Serial.println("Searching EEG:");
  disp_lcd_frombuffer();
  unsigned short id=ad71_readid(); // 0x4fdx
  while ((id & 0xfff0)!= 0x4fd0) {
     Serial.print("ERROR id=");Serial.print(id);
     Serial.println(" EEG Brick not found");
     disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
      disp_print_xy_lcd(0, 15, (unsigned char*)"NOT FOUND", COLOR_WHITE, 0);
      disp_lcd_frombuffer();
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x07) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x0f) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     id=ad71_readid(); // 0x4fdx
  }
  Serial.print("id=");Serial.println(id);
  // OK Init dr Register
  ad71_reset();
  ad71_init();

//Init display schlange
   for(int x = 0; x <=63; x++){
        p_disp[x] = 0;
        }
```

```
}

int findMin(){
  long minimum = p_disp[0];
  int c;
  for (c = 0; c < 62; c++)
    {
        if (p_disp[c] < minimum)
        {
           minimum = p_disp[c];
        }
    }
  return minimum;

  }

int findMax(){
  long maximum = p_disp[0];
  int c;
  for (c = 0; c < 62; c++)
    {
        if (p_disp[c] > maximum)
        {
           maximum = p_disp[c];
        }
    }
  return maximum;

  }

long lastval=0;
void loop() {


ad71_dataread();
 #ifdef ALLL
    Serial.print("ad:");Serial.print(values[0],HEX);
    Serial.print(" ");Serial.print(values[1],HEX);
    Serial.print(" ");Serial.print(values[2],HEX);
    Serial.print(" ");Serial.print(values[3],HEX);
    Serial.println();
 #endif

  for (int it = 63; it >= 1; it--)
  {
  p_disp[it] = p_disp[it-1];
  }
  long v1 = values[0]-0x800000001;
   p_disp[0] = v1;

 long  minim = findMin();
 long maxim = findMax();

   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
 disp_print_xy_lcd(0, 0, (unsigned char*)"EEG/EKG/EMG", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);
 for (int i=0; i<63;i++) { // nun fuer alle 64 Spalten (x)  des Displays
  long val = 47-(map(p_disp[i], minim, maxim, 0, 30));
    disp_setpixel(i, val , COLOR_WHITE); // koordinate 0,0 ist links oben daher 47-y
    //OLDX NEWX OLDY NEWY
    disp_line_lcd (i-1,lastval,i,val, COLOR_WHITE);
    lastval = val;
  } // alle Spalten
 disp_lcd_frombuffer(); // dann erst updaten, double buffer flimmerfrei


  // long v2 = values[1]-0x800000001;
 //Serial.print(v1);Serial.print(",");Serial.println(v2);
 Serial.println(v1);
}
```



Was passiert? Die Messwerte, die Gehirnströme werden auf dem OLED-Display ausgegeben.

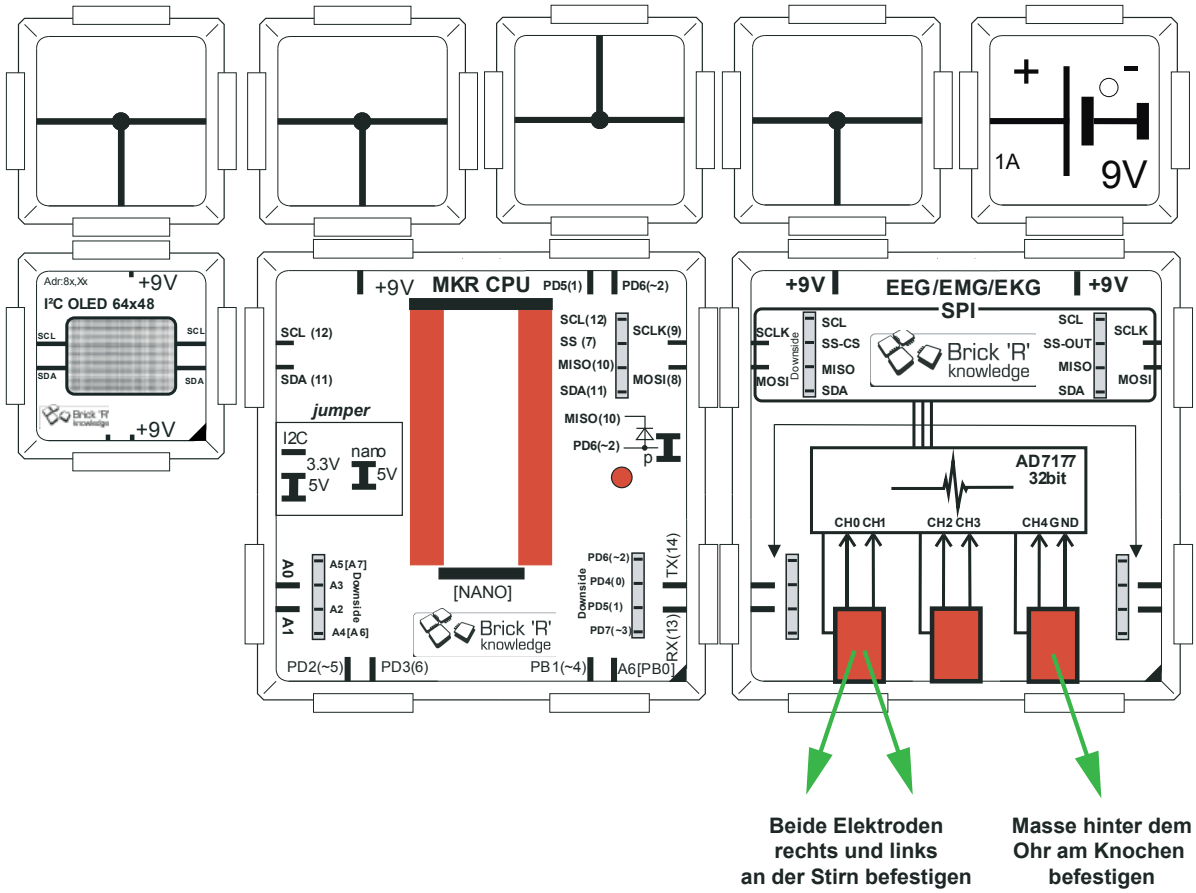


8.3 EEG Messung mit Ausgabe der Frequenzspektren (FFT)

Die beiden Elektroden von CH0 und CH1 an der Stirn links und rechts befestigen, die Masse bei CH4/GND hinter dem Ohrläppchen. Man kann dort auch beide Elektroden anbringen.

FFT bedeutet „Fast Fourier Transformation“. Das Signal wird dabei in seine spektralen Komponenten zerlegt. Damit lässt sich eine bessere und genauere Analyse der Gehirnwellen durchführen

Dann ruhige Position einnehmen. Den Versuch sollte man am besten zu zweit durchführen. Ausgabe auf dem grafischen OLED-Display.




```
// EEG BRICK mit Frequenzspektrum
//  MKR WIFI 2010 __SAMD21G18A__
// Output to serial Interface !!


// I2C Select: clken selen mosien led0 sense1 sense2 sense3 spez
//             0     1      2      3     4      5      6      7
//             out   out    out    out   in     in     in     in
//

#include <Wire.h> // Definitionen laden fuer I2C
#include <SPI.h>
#include <avr/pgmspace.h> // weitere Definitionen

#include "Adafruit_ZeroFFT.h"  // LIBRRAY FFT ZERO !!!!

#define DATA_SIZE 128
q15_t datasignal[DATA_SIZE]; // int16_t

//the sample rate
#define FS 100

// USe IOs of MKR
```

```
#ifdef ARDUINO_SAMD_MKRWIFI1010
#define __AVR_SAMD21__   1

#endif


// Alle Adressen bei den 8574xx bricks:
#define i2cIO9534_0 (0x40>>1) // Trick um elegant mit Bytes
#define i2cIO9534_1 (0x42>>1) // zu arbeiten statt in 7 Bit
#define i2cIO9534_2 (0x44>>1) // das letzte Bit ist das R/W
#define i2cIO9534_3 (0x46>>1) // dass von der Arduino
#define i2cIO9534_4 (0x48>>1) // Bibliothek dazugefuegt wird
#define i2cIO9534_5 (0x4A>>1) // wir definieren hier alle
#define i2cIO9534_6 (0x4C>>1) // Bereiche die man mit den
#define i2cIO9534_7 (0x4E>>1) // Bricks einstellen kann,.

#define i2cIO9534A_0 (0x70>>1) // Die Serie PCF8474AT
#define i2cIO9534A_1 (0x72>>1) // beginnt bei Adresse
#define i2cIO9534A_2 (0x74>>1) // 0x70 = 70 sedezimal
#define i2cIO9534A_3 (0x76>>1)  // 01110000 binaer
#define i2cIO9534A_4 (0x78>>1) // oder intern 0111000
#define i2cIO9534A_5 (0x7A>>1) // dabei 0111xxx
#define i2cIO9534A_6 (0x7C>>1) // mit x fuer die
#define i2cIO9534A_7 (0x7E>>1) // Dilschalterposition

#define myi2cIOadr i2cIO9534_0 // HIER PASSENDE ADRESSE EINTRAGEN

#define SSPB 10 // Pin 10 = SS  - --- PB2  1<<2 als Maske

// SPI: SCLK MOSI MISO und SS Brick=PB2 D10 fest !!
// MEGA Pin 51=MOSI Pin 52=SCK und Pin50=MISO °!!!
// mega:
// NANO: MOSI:PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// UNO: MOSI: PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// ACHUNG: MISO ist auch STatus DATA READY !!!
//
// PORTB 4  (7 6 5 4 -- 3 2 1 0)
// BEI MKR PD6

#if defined(__AVR_ATmega2560__)  // Arduino MEGA2560
#define MISOPIN 50
#warning ("INFO : compiles for AT MEGA2560")
#elif defined(__AVR_SAMD21__)
#warning ("INFO : compiles for MKR WIFI1010")
#define MISOPIN 2  // READ ONLY MKR ueber Diode...
#else // NANO
#warning ("INFO : compiles rest: also NANO...")
#define MISOPIN 12
#endif

// #define MISOPIN 50

long p_disp[DATA_SIZE];


// 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 ---------------------------------------------
// 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 enabless()
{
  //digitalWrite(SSPB, LOW);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, LOW);
  #else
```

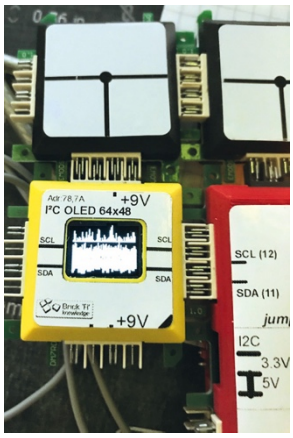
```
   PORTB &= 0xfb; // Default = high 7 6 5 4 3 2low 1 0 PB2 low
  #endif
  delayMicroseconds(10);
}
void disabless()
{
  delayMicroseconds(10);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, HIGH);
  #else
    PORTB |= 1 << 2; // Default = high bit2 PB2
  #endif
  // digitalWrite(SSPB, HIGH);
  delayMicroseconds(10);
  //
}

//AD 7177-x
// Register
// SS:-------_________________----------

//
unsigned short ad71_readid()
{
  unsigned char ch1;
  unsigned short val1=0;
  enabless();
  ch1 = SPI.transfer(0x47); // COmm: -wen r-w ra5..9  r-1=1 read 0=writ
  val1 = SPI.transfer16(0); // ID
  disabless();
  return(val1);
}

//

//
#define NOCHAN 1

void ad71_init()
{
   enabless();
   // Register setzen...
   SPI.transfer(0x01); SPI.transfer16(0x800C);  // ADCMODE ! ref en, mode continouus, clock
Crystal : 100 00 000  0 000 11 00
   SPI.transfer(0x02); SPI.transfer16(0x0442);  // INTERFACE MOD data_stat(6)en w32 en --- 000
0   0 10 0    0 1   0 0  00 1 0  old 0x0042
   SPI.transfer(0x06); SPI.transfer16(0x060d);  // GPIO 000 0  0 11 0   0 0 00 1101
   // Channel 0
   SPI.transfer(0x10); SPI.transfer16(0x8001);  // CH0   enable set0 ain0 ain1 ... 10 00 00 00
000 0 0001
   // Channel 1
   SPI.transfer(0x11); SPI.transfer16(0x0043);  // CH1   enable set0 ain2 ain3 ... 10 00 00 00
010 0 0011
   // Channel 2
   SPI.transfer(0x12); SPI.transfer16(0x0096);  // CH2   enable set0 ain4 ref- ... 10 00 00 00
100 1 0110
   // Channel 3  optional
   SPI.transfer(0x13); SPI.transfer16(0x0232);  // CH3   enable set0 smp+ tmp- ... 10 00 00 10
001 1 0010
   // alle gleiches setup daher setup 0
   // SETup registers.
   SPI.transfer(0x20); SPI.transfer16(0x1300);  // SET0 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x21); SPI.transfer16(0x1300);  // SET1 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x22); SPI.transfer16(0x1300);  // SET2 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x23); SPI.transfer16(0x1300);  // SET3 : 000 1  0 0 1 1  0 0 00 0000
   // Filter Config gehoeren zu setup reg
   SPI.transfer(0x28); SPI.transfer16(0x050e);  // FLT0 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x29); SPI.transfer16(0x050a);  // FLT1 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2A); SPI.transfer16(0x050a);  // FLT2 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2B); SPI.transfer16(0x050a);  // FLT3 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
```

```
   // OFFSET und GAIN auf factory lassen...
   disabless();

}

unsigned long values[4];

// tmperature 470uV/K
//         1181972248 beispiel bei 2.5V Ref+- 0.137mV 292K ->19.7
// 67BCCE0 108776672 -3.5


void ad71_reset()
{
  enabless();
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  disabless();
  delayMicroseconds(550);
}


void ad71_dataread()
{
  // Wenn ready incl status 4 byts 32 bit
  // STATUS
  unsigned char sts;
  unsigned char ch1;
  unsigned long val1=0;
  unsigned long val2=0;
  unsigned int idx;
  static int first = 0;
  int val=0;
  int i=0;
  for (i=0; i<NOCHAN;i++) {
    // SYNC:
    // NICHT RDY Abfragen,,,
  #ifdef XXXXXX
    enabless();
    do {

      sts = SPI.transfer(0x40);
      sts = SPI.transfer(0x00);
       delayMicroseconds(10);
       // Serial.print(sts);
       // Serial.println("S wt:");

    } while (((sts & 0x80)== 0x80) && (first == 1));
    first = 1;
    disabless();
#endif

    // DATA RDY
    delayMicroseconds(10);
    //
    enabless();
    delayMicroseconds(4); // DRDY valid...
    // SINGLE MODE SPI.transfer(0x01); SPI.transfer16(0x801C);  // ADCMODE ! ref en, mode
SINGLE, clock Crystal : 100 00 000  0 000 11 00
    //
    // Sonst abfragen hier immer...
    do {
      val = digitalRead(MISOPIN); // MISOPIN
    } while (val == HIGH);

    //
    ch1 = SPI.transfer(0x44); // COmm:  data read
```

```
    //
    val1 = SPI.transfer16(0); // data <
    val2 = SPI.transfer16(0); // data
    //

     //
    delayMicroseconds(5); // DRDY valid...
    disableSS();
    //
    val1 = val2 | (val1 << 16); // 4 bit status
    idx = val2 & 0x3; // CHANNEL FLG  bit 3210
    values[idx] = val1 & 0xfffffff0;

  }
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("Starting EEG:");

  #if defined(__AVR_SAMD21__)
   pinMode(MISOPIN, INPUT); // MISO !!
   pinMode(7, OUTPUT);
  #else
   pinMode(12, INPUT); // MISO !!
   pinMode(SSPB, OUTPUT);
  #endif
  disableSS();



  Wire.begin(); // I2C aktivieren !
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x03) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0xF0) ;  // 1=input
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
  Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0x0f) ;  // led an !!
  Wire.endTransmission(); // Stop Kondition setzen bei I2C
  i2c_oled_initall(i2coledssd);
  disp_buffer_clear(COLOR_BLACK);
  // INIT
   SPI.begin();
  SPI.beginTransaction(SPISettings(7000000, MSBFIRST, SPI_MODE3));
  // NICHT VERWENDEN:
  //SPI.setDataMode(SPI_MODE3);  // sonst immer 1     clk ---____----   rising edhe = spi mode
3 krit mkr´
  //SPI.setClockDivider(8);
 disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
 disp_print_xy_lcd(0, 0, (unsigned char*)"EEG/EKG/EMG", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);
  disp_print_xy_lcd(20, 20, (unsigned char*)"Starte...", COLOR_WHITE, 0);
  Serial.println("Searching EEG:");
  disp_lcd_frombuffer();
  unsigned short id=ad71_readid(); // 0x4fdx
  while ((id & 0xfff0)!= 0x4fd0) {
     Serial.print("ERROR id=");Serial.print(id);
     Serial.println(" EEG Brick not found");
     disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
      disp_print_xy_lcd(0, 15, (unsigned char*)"NOT FOUND", COLOR_WHITE, 0);
      disp_lcd_frombuffer();
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x07) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     Wire.beginTransmission(myi2cIOadr);// Startvorgang I2C Adresse
     Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
     Wire.write(0x0f) ;  // led an !!
     Wire.endTransmission(); // Stop Kondition setzen bei I2C
     delay(100);
     id=ad71_readid(); // 0x4fdx
```

```
  }
  Serial.print("id=");Serial.println(id);
  // OK Init dr Register
  ad71_reset();
  ad71_init();

//Init display schlange
   for(int x = 0; x < DATA_SIZE; x++){
        p_disp[x] = 0;
        datasignal[x] = 0;
   }


}



int findMin(){
  long minimum = p_disp[0];
  int c;
  for (c = 0; c < DATA_SIZE; c++)
    {
        if (p_disp[c] < minimum)
        {
           minimum = p_disp[c];
        }
    }
  return minimum;

  }

int findMax(){
  long maximum = p_disp[0];
  int c;
  for (c = 0; c < DATA_SIZE; c++)
    {
        if (p_disp[c] > maximum)
        {
           maximum = p_disp[c];
        }
    }
  return maximum;

  }

long lastval=0;
void loop() {


ad71_dataread();
 #ifdef ALLL
    Serial.print("ad:");Serial.print(values[0],HEX);
    Serial.print(" ");Serial.print(values[1],HEX);
    Serial.print(" ");Serial.print(values[2],HEX);
    Serial.print(" ");Serial.print(values[3],HEX);
    Serial.println();
 #endif

  for (int it = DATA_SIZE-1; it >= 1; it--) // ALLE DURCHSCHIEBEN !!!
  {
    p_disp[it] = p_disp[it-1];
  }
  long v1 = values[0]-0x800000001;
   p_disp[0] = v1;

 long  minim = findMin();
 long maxim = findMax();
 // IN FFT Kopieren:
 // und wandlen auf 16 bit range
 static int cnt=0;
 cnt++;
 if (cnt == 256) {
   for (int j=0; j<DATA_SIZE-1;j++){
     datasignal[j] = map(p_disp[j], minim, maxim, -8000,8000);
```

```
   }
   // DANN FFT ::::  alle 256 mal
   ZeroFFT(datasignal, DATA_SIZE);
   // DORT ERGEBNIS min und max FFT
   int minfft,maxfft;
   minfft=datasignal[2];
   maxfft=datasignal[2];
   for(int i=0; i<DATA_SIZE/2; i++){
      if ((i>2) && (i<DATA_SIZE/2-1)) {
       if (minfft>datasignal[i]) minfft=datasignal[i];
       if (maxfft<datasignal[i]) maxfft=datasignal[i];
      }
      //print the frequency
      Serial.print(FFT_BIN(i, FS, DATA_SIZE));
      Serial.print(" Hz: ");

      //print the corresponding FFT output
      Serial.println(datasignal[i]);
    }
   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
   // disp_print_xy_lcd(0, 0, (unsigned char*)"EEG", COLOR_WHITE, 0);
   // disp_line_lcd (0,12,63,12, COLOR_WHITE);
   // 64x48
   //
   for (int i=0;i<63;i++) {
       long val;
       val = 16-(map(datasignal[i], minfft, maxfft, 0, 16));
       disp_line_lcd (i,16,i,val, COLOR_WHITE);
   }
   lastval= 47-(map(p_disp[0], minim, maxim, 0, 30));
   for (int i=0; i<128;i++) { // nun fuer alle 64 Spalten (x)  des Displays  128 DATEN
ausnahmweise direkt
       long val;
       val = 47-(map(p_disp[i], minim, maxim, 0, 30));
       disp_line_lcd (i/2-1,lastval,i/2,val, COLOR_WHITE);
       lastval = val;
    } // alle Spalten
    disp_lcd_frombuffer(); // dann erst updaten, double buffer flimmerfrei
    cnt = 0;
 }

  // long v2 = values[1]-0x800000001;
  //Serial.print(v1);Serial.print(",");Serial.println(v2);
  Serial.println(v1);
}
```



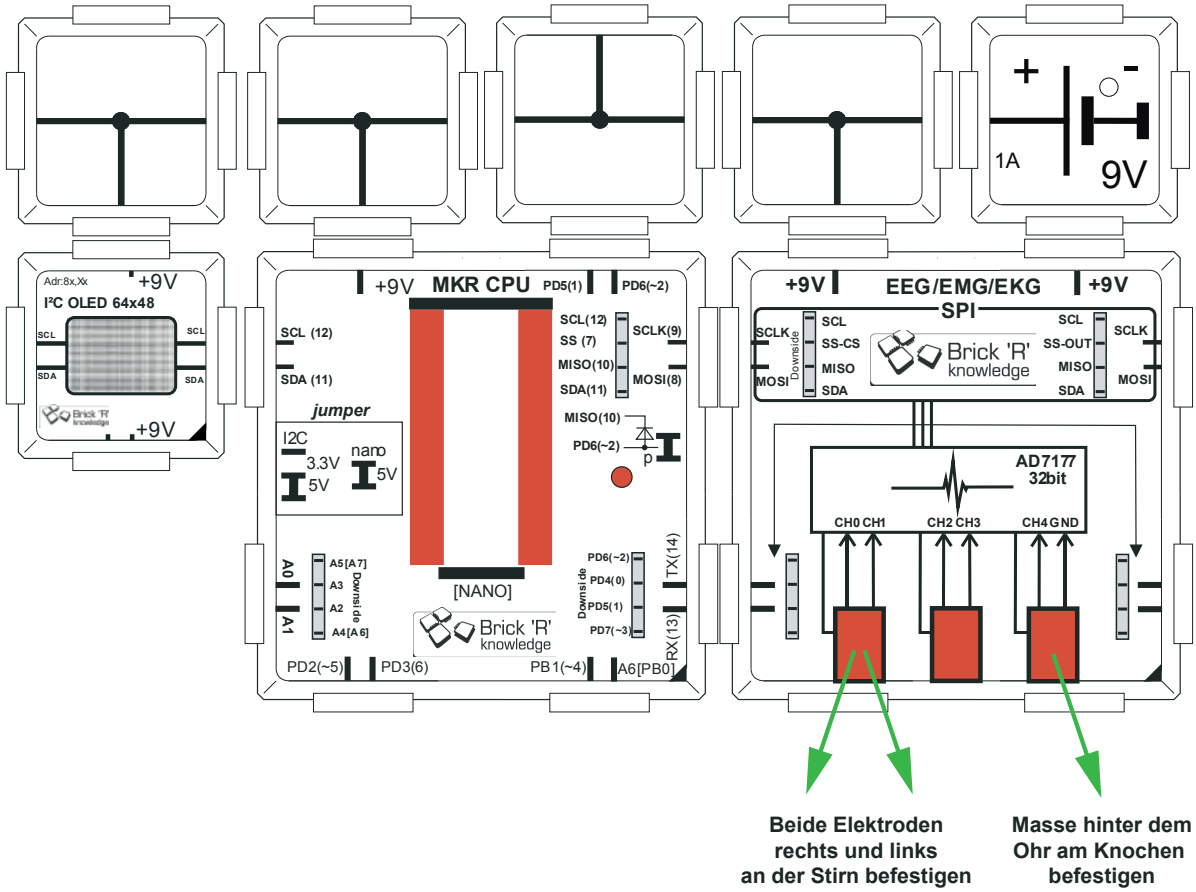
Was passiert? Die Messwerte, die Gehirnströme werden als Frequenzanalyse auf dem OLED-Display ausgegeben.

8.4 EEG Messung Einteilung in Alpha, Beta, Gamma...

Die beiden Elektroden von CH0 und CH1 an der Stirn links und rechts befestigen, die Masse bei CH4/GND hinter dem Ohrläppchen. Man kann dort auch beide Elektroden anbringen.

Nach der FFT folgt in dem Programm eine Klassifizierung des Frequenzspektrums. Man unterscheidet Alpha-, Beta-, Gamma- usw. Wellen. Diese Wellen sind bestimmten Gehirnzuständen wie z.B. Aktivität, Schlaf, Meditation usw. zuzuordnen.

Dann ruhige Position einnehmen. Den Versuch sollte man am besten zu zweit durchführen. Ausgabe auf dem grafischen OLED-Display.




```
// EEG BRICK  Auswertung
//  MKR WIFI 2010 __SAMD21G18A__
// Output to serial Interface !!


// I2C Select: clken selen mosien led0 sense1 sense2 sense3 spez
//             0     1      2      3     4      5      6      7
//             out   out    out    out   in     in     in     in
//

#include <Wire.h> // Definitionen laden fuer I2C
#include <SPI.h>
#include <avr/pgmspace.h> // weitere Definitionen

#include "Adafruit_ZeroFFT.h"  // LIBRRAY FFT ZERO !!!!

#define DATA_SIZE 128
q15_t datasignal[DATA_SIZE]; // int16_t

//the sample rate
#define FS 100
```

```
// USe IOs of MKR
#ifdef ARDUINO_SAMD_MKRWIFI1010
#define __AVR_SAMD21__   1

#endif


// Alle Adressen bei den 8574xx bricks:
#define i2cIO9534_0 (0x40>>1) // Trick um elegant mit Bytes
#define i2cIO9534_1 (0x42>>1) // zu arbeiten statt in 7 Bit
#define i2cIO9534_2 (0x44>>1) // das letzte Bit ist das R/W
#define i2cIO9534_3 (0x46>>1) // dass von der Arduino
#define i2cIO9534_4 (0x48>>1) // Bibliothek dazugefuegt wird
#define i2cIO9534_5 (0x4A>>1) // wir definieren hier alle
#define i2cIO9534_6 (0x4C>>1) // Bereiche die man mit den
#define i2cIO9534_7 (0x4E>>1) // Bricks einstellen kann,.

#define i2cIO9534A_0 (0x70>>1) // Die Serie PCF8474AT
#define i2cIO9534A_1 (0x72>>1) // beginnt bei Adresse
#define i2cIO9534A_2 (0x74>>1) // 0x70 = 70 sedezimal
#define i2cIO9534A_3 (0x76>>1)  // 01110000 binaer
#define i2cIO9534A_4 (0x78>>1) // oder intern 0111000
#define i2cIO9534A_5 (0x7A>>1) // dabei 0111xxx
#define i2cIO9534A_6 (0x7C>>1) // mit x fuer die
#define i2cIO9534A_7 (0x7E>>1) // Dilschalterposition

#define myi2cIOadr i2cIO9534_0 // HIER PASSENDE ADRESSE EINTRAGEN

#define SSPB 10 // Pin 10 = SS  - --- PB2  1<<2 als Maske

// SPI: SCLK MOSI MISO und SS Brick=PB2 D10 fest !!
// MEGA Pin 51=MOSI Pin 52=SCK und Pin50=MISO °!!!
// mega:
// NANO: MOSI:PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// UNO: MOSI: PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// ACHUNG: MISO ist auch STatus DATA READY !!!
//
// PORTB 4  (7 6 5 4 -- 3 2 1 0)
// BEI MKR PD6

#if defined(__AVR_ATmega2560__)  // Arduino MEGA2560
#define MISOPIN 50
#warning ("INFO : compiles for AT MEGA2560")
#elif defined(__AVR_SAMD21__)
#warning ("INFO : compiles for MKR WIFI1010")
#define MISOPIN 2  // READ ONLY MKR ueber Diode...
#else // NANO
#warning ("INFO : compiles rest: also NANO...")
#define MISOPIN 12
#endif

// #define MISOPIN 50

long p_disp[DATA_SIZE];

int numBrks = 0;
int ArdrsBrk[8];


// 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 ---------------------------------------------
// 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 enabless()
```

```
{
  //digitalWrite(SSPB, LOW);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, LOW);
  #else
   PORTB &= 0xfb; // Default = high 7 6 5 4 3 2low 1 0 PB2 low
  #endif
  delayMicroseconds(10);
}
void disabless()
{
  delayMicroseconds(10);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, HIGH);
  #else
    PORTB |= 1 << 2; // Default = high bit2 PB2
  #endif
  // digitalWrite(SSPB, HIGH);
  delayMicroseconds(10);
  //
}

//AD 7177-x
// Register
// SS:-------_________________----------

//
unsigned short ad71_readid()
{
  unsigned char ch1;
  unsigned short val1=0;
  enabless();
  ch1 = SPI.transfer(0x47); // COmm: -wen r-w ra5..9  r-1=1 read 0=writ
  val1 = SPI.transfer16(0); // ID
  disabless();
  return(val1);
}

//

//
#define NOCHAN 1

void ad71_init()
{
   enabless();
   // Register setzen...
   SPI.transfer(0x01); SPI.transfer16(0x800C);  // ADCMODE ! ref en, mode continouus, clock
Crystal : 100 00 000  0 000 11 00
   SPI.transfer(0x02); SPI.transfer16(0x0442);  // INTERFACE MOD data_stat(6)en w32 en --- 000
0   0 10 0    0 1   0 0  00 1 0  old 0x0042
   SPI.transfer(0x06); SPI.transfer16(0x060d);  // GPIO 000 0  0 11 0   0 0 00 1101
   // Channel 0
   SPI.transfer(0x10); SPI.transfer16(0x8001);  // CH0   enable set0 ain0 ain1 ... 10 00 00 00
000 0 0001
   // Channel 1
   SPI.transfer(0x11); SPI.transfer16(0x0043);  // CH1   enable set0 ain2 ain3 ... 10 00 00 00
010 0 0011
   // Channel 2
   SPI.transfer(0x12); SPI.transfer16(0x0096);  // CH2   enable set0 ain4 ref- ... 10 00 00 00
100 1 0110
   // Channel 3  optional
   SPI.transfer(0x13); SPI.transfer16(0x0232);  // CH3   enable set0 smp+ tmp- ... 10 00 00 10
001 1 0010
   // alle gleiches setup daher setup 0
   // SETup registers.
   SPI.transfer(0x20); SPI.transfer16(0x1300);  // SET0 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x21); SPI.transfer16(0x1300);  // SET1 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x22); SPI.transfer16(0x1300);  // SET2 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x23); SPI.transfer16(0x1300);  // SET3 : 000 1  0 0 1 1  0 0 00 0000
   // Filter Config gehoeren zu setup reg
   SPI.transfer(0x28); SPI.transfer16(0x050e);  // FLT0 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
```

```
   SPI.transfer(0x29); SPI.transfer16(0x050a);  // FLT1 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2A); SPI.transfer16(0x050a);  // FLT2 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2B); SPI.transfer16(0x050a);  // FLT3 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   // OFFSET und GAIN auf factory lassen...
   disabless();

}

unsigned long values[4];

// tmperature 470uV/K
//         1181972248 beispiel bei 2.5V Ref+- 0.137mV 292K ->19.7
// 67BCCE0 108776672 -3.5


void ad71_reset()
{
  enabless();
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  disabless();
  delayMicroseconds(550);
}


void ad71_dataread()
{
  // Wenn ready incl status 4 byts 32 bit
  // STATUS
  unsigned char sts;
  unsigned char ch1;
  unsigned long val1=0;
  unsigned long val2=0;
  unsigned int idx;
  static int first = 0;
  int val=0;
  int i=0;
  for (i=0; i<NOCHAN;i++) {
    // SYNC:
    // NICHT RDY Abfragen,,,
  #ifdef XXXXXX
    enabless();
    do {

      sts = SPI.transfer(0x40);
      sts = SPI.transfer(0x00);
       delayMicroseconds(10);
       // Serial.print(sts);
       // Serial.println("S wt:");

    } while (((sts & 0x80)== 0x80) && (first == 1));
    first = 1;
    disabless();
#endif

    // DATA RDY
    delayMicroseconds(10);
    //
    enabless();
    delayMicroseconds(4); // DRDY valid...
    // SINGLE MODE SPI.transfer(0x01); SPI.transfer16(0x801C);  // ADCMODE ! ref en, mode
SINGLE, clock Crystal : 100 00 000  0 000 11 00
    //
    // Sonst abfragen hier immer...
    do {
```

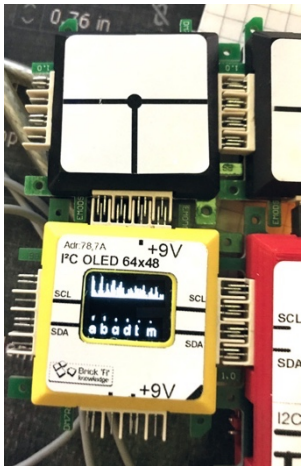
```
      val = digitalRead(MISOPIN); // MISOPIN
    } while (val == HIGH);

    //
    ch1 = SPI.transfer(0x44); // COmm:  data read

    //
    val1 = SPI.transfer16(0); // data <
    val2 = SPI.transfer16(0); // data
    //

     //
    delayMicroseconds(5); // DRDY valid...
    disableSS();
    //
    val1 = val2 | (val1 << 16); // 4 bit status
    idx = val2 & 0x3; // CHANNEL FLG  bit 3210
    values[idx] = val1 & 0xfffffff0;

  }
}


bool CheckForDevice(int Adrs){

  switch (Adrs){   // Startvorgang I2C Adresse
      case 0: Wire.beginTransmission(i2cIO9534_0);break;
      case 1: Wire.beginTransmission(i2cIO9534_1);break;
      case 2: Wire.beginTransmission(i2cIO9534_2);break;
      case 3: Wire.beginTransmission(i2cIO9534_3);break;
      case 4: Wire.beginTransmission(i2cIO9534_4);break;
      case 5: Wire.beginTransmission(i2cIO9534_5);break;
      case 6: Wire.beginTransmission(i2cIO9534_6);break;
      case 7: Wire.beginTransmission(i2cIO9534_7);break;
  }

  Wire.write(0x03) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0xF0) ;  // 1=input
  Wire.endTransmission(); // Stop Kondition setzen bei I2C

  switch (Adrs){     // Startvorgang I2C Adresse
      case 0: Wire.beginTransmission(i2cIO9534_0);break;
      case 1: Wire.beginTransmission(i2cIO9534_1);break;
      case 2: Wire.beginTransmission(i2cIO9534_2);break;
      case 3: Wire.beginTransmission(i2cIO9534_3);break;
      case 4: Wire.beginTransmission(i2cIO9534_4);break;
      case 5: Wire.beginTransmission(i2cIO9534_5);break;
      case 6: Wire.beginTransmission(i2cIO9534_6);break;
      case 7: Wire.beginTransmission(i2cIO9534_7);break;
  }

  Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0x0f) ;  // led an !!
  Wire.endTransmission(); // Stop Kondition setzen bei I2C

  //SPI ID Abfragen
  SPI.begin();
  SPI.beginTransaction(SPISettings(7000000, MSBFIRST, SPI_MODE3));
  Serial.print("Searching EEG in Adrs ");Serial.println(Adrs);
  unsigned short id=ad71_readid(); // 0x4fdx
  if ((id & 0xfff0)!= 0x4fd0) { //Kein EEG Brick hier gefunden
     Serial.print("ERROR id=");Serial.print(id);
     Serial.println(" EEG Brick here not found");
     return false;
  }
  else{ //EEG Hier gefunden -> return True
   Serial.print("id=");Serial.println(id);
   Serial.println(" EEG Brick found");
   // OK Init dr Register
   ad71_reset();
   ad71_init();
   return true;
  }
```

```
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Wire.begin(); // I2C aktivieren !
  Serial.println("Starting EEG:");

  #if defined(__AVR_SAMD21__)
   pinMode(MISOPIN, INPUT); // MISO !!
   pinMode(7, OUTPUT);
  #else
   pinMode(12, INPUT); // MISO !!
   pinMode(SSPB, OUTPUT);
  #endif
  disableSS();

   pinMode(LED_BUILTIN, OUTPUT);

 i2c_oled_initall(i2coledssd);
 disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
 disp_print_xy_lcd(0, 0, (unsigned char*)"EEG/EKG/EMG", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);
 disp_print_xy_lcd(20, 20, (unsigned char*)"Starte...", COLOR_WHITE, 0);
 disp_lcd_frombuffer();


 for (int i = 7; i >= 0; i--){
  if (CheckForDevice(i)){
      ArdrsBrk[numBrks] = i;
      numBrks++;
      Serial.print(numBrks);Serial.println(" Gefunden");
  }
 }



 if (numBrks <= 0){ //Keine Bricks Gefunden
   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
   disp_print_xy_lcd(0, 20, (unsigned char*)"0 Gefunden", COLOR_WHITE, 0);
   disp_lcd_frombuffer();
   while (1){Serial.println("Nichts Gefunden");}
 }
 else{
   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
   disp_print_xy_lcd(0, 20, (unsigned char*)"Gefunden", COLOR_WHITE, 0);
   disp_lcd_frombuffer();
 }

//Init display schlange
   for(int x = 0; x < DATA_SIZE; x++){
        p_disp[x] = 0;
        datasignal[x] = 0;
   }


}



int findMin(){
  long minimum = p_disp[0];
  int c;
  for (c = 0; c < DATA_SIZE; c++)
    {
        if (p_disp[c] < minimum)
        {
           minimum = p_disp[c];
        }
    }
  return minimum;

  }
```

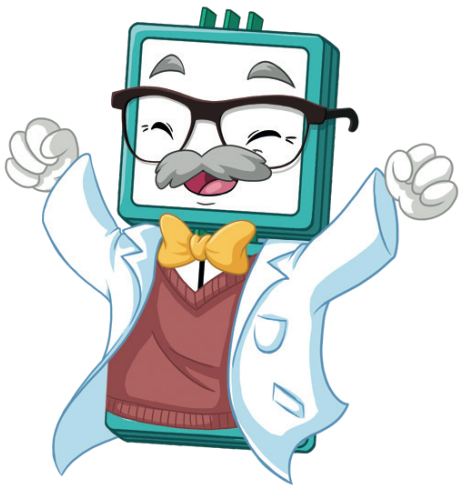
```
int findMax(){
  long maximum = p_disp[0];
  int c;
  for (c = 0; c < DATA_SIZE; c++)
    {
        if (p_disp[c] > maximum)
        {
           maximum = p_disp[c];
        }
    }
  return maximum;

  }

long lastval=0;
void loop() {


ad71_dataread();
 #ifdef ALLL
    Serial.print("ad:");Serial.print(values[0],HEX);
    Serial.print(" ");Serial.print(values[1],HEX);
    Serial.print(" ");Serial.print(values[2],HEX);
    Serial.print(" ");Serial.print(values[3],HEX);
    Serial.println();
 #endif

  for (int it = DATA_SIZE-1; it >= 1; it--) // ALLE DURCHSCHIEBEN !!!
  {
    p_disp[it] = p_disp[it-1];
  }
  long v1 = values[0]-0x800000001;
   p_disp[0] = v1;

 long  minim = findMin();
 long maxim = findMax();

 // IN FFT Kopieren:
 // und wandlen auf 16 bit range
 static int cnt=0;
 cnt++;
 if (cnt == 256) { //Wenn 256 Werte gesnammelt wurden...
   for (int j=0; j<DATA_SIZE-1;j++){
     datasignal[j] = map(p_disp[j], minim, maxim, -8000,8000);
   }
   // DANN FFT ::::  alle 256 mal
   ZeroFFT(datasignal, DATA_SIZE);
   // DORT ERGEBNIS min und max FFT
   int minfft,maxfft;
   minfft=datasignal[2];
   maxfft=datasignal[2];
   for(int i=0; i<DATA_SIZE/2; i++){
      if ((i>2) && (i<DATA_SIZE/2-1)) {
       if (minfft>datasignal[i]) minfft=datasignal[i];
       if (maxfft<datasignal[i]) maxfft=datasignal[i];
      }
      //print the frequency
      Serial.print(FFT_BIN(i, FS, DATA_SIZE));
      Serial.print(" Hz: ");

      //print the corresponding FFT output
      Serial.println(datasignal[i]);
    }
   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
   // disp_print_xy_lcd(0, 0, (unsigned char*)"EEG", COLOR_WHITE, 0);
   // disp_line_lcd (0,12,63,12, COLOR_WHITE);
   // 64x48



   for (int i=0;i<63;i++) {
       long val;
       val = 16-(map(datasignal[i], minfft, maxfft, 0, 16));
```

```
       disp_line_lcd (i,16,i,val, COLOR_WHITE);
   }
   /*
   lastval= 47-(map(p_disp[0], minim, maxim, 0, 30));
   for (int i=0; i<128;i++) { // nun fuer alle 64 Spalten (x)  des Displays  128 DATEN
ausnahmweise direkt
       long val;
       val = 47-(map(p_disp[i], minim, maxim, 0, 30));
       disp_line_lcd (i/2-1,lastval,i/2,val, COLOR_WHITE);
       lastval = val;
    } // alle Spalten*/

     //EEG Frequenzen:
     // ALPHA: 5-18hz BETA: 16-31hz GAMMA: >32hz DELTA: <4 THETA: 4-7hz MU: 8-12
     long Alpha, Beta, Gamma, Delta, Theta, Mu;

     Alpha = datasignal[5];
     for (int i=5;i<=18;i++) {
      if (datasignal[i] > Alpha) Alpha = datasignal[i];
     }



     Delta = datasignal[4];



     Theta = datasignal[4];
     for (int i=4;i<=7;i++) {
     if (datasignal[i] > Theta) Theta = datasignal[i];
     }

     Mu = datasignal[8];
     for (int i=8;i<=12;i++) {
     if (datasignal[i] > Mu) Mu = datasignal[i];
     }


     Beta = datasignal[16];
     for (int i=16;i<=30;i++) {
     if (datasignal[i] > Beta) Beta = datasignal[i];
     }


     Gamma = datasignal[32];
     for (int i=32;i<=40;i++) {
      if (datasignal[i] > Gamma) Gamma = datasignal[i];
     }

     //Buchstaben reihe
    disp_print_xy_lcd(0, 37, (unsigned char*)"a", COLOR_WHITE, 0);  disp_print_xy_lcd(10, 37,
(unsigned char*)"b", COLOR_WHITE, 0);
    disp_print_xy_lcd(20, 37, (unsigned char*)"g", COLOR_WHITE, 0);  disp_print_xy_lcd(30, 37,
(unsigned char*)"d", COLOR_WHITE, 0);
    disp_print_xy_lcd(40, 37, (unsigned char*)"t", COLOR_WHITE, 0); disp_print_xy_lcd(50, 37,
(unsigned char*)"m", COLOR_WHITE, 0);

    //Linien   2,36,2,22

    disp_line_lcd (2,36,2,(map(floor(Alpha), 0, 300, 36, 22)), COLOR_WHITE);
    disp_line_lcd (12,36,12,(map(floor(Beta), 0, 300, 36, 22)), COLOR_WHITE);
    disp_line_lcd (22,36,22,(map(floor(Gamma), 0, 300, 36, 22)), COLOR_WHITE);
    disp_line_lcd (52,36,52,(map(floor(Delta), 0, 300, 36, 22)), COLOR_WHITE);
    disp_line_lcd (32,36,32,(map(floor(Theta), 0, 300, 36, 22)), COLOR_WHITE);
    disp_line_lcd (42,36,42,(map(floor(Mu), 0, 3200, 36, 22)), COLOR_WHITE);
    //disp_line_lcd (0,33,0,(map(Alpha, 0, 100, 33, 20)), COLOR_WHITE);

    disp_lcd_frombuffer(); // dann erst updaten, double buffer flimmerfrei
    cnt = 0; //Reset Count

    //MINDCONTROL AUSWERTUNG

    if (Alpha > Gamma) {
      digitalWrite(LED_BUILTIN, HIGH);
```

```
    }
    else {
      digitalWrite(LED_BUILTIN, LOW);
    }
 } //ENDIF Werte
  // long v2 = values[1]-0x800000001;
  //Serial.print(v1);Serial.print(",");Serial.println(v2);
  Serial.println(v1);
}
```



Was passiert? Die Messwerte, die Gehirnströme werden als Frequenzanalyse mit Zuordnung zu Alpha-Wellen, Beta-Wellen usw. auf dem OLED-Display ausgegeben.

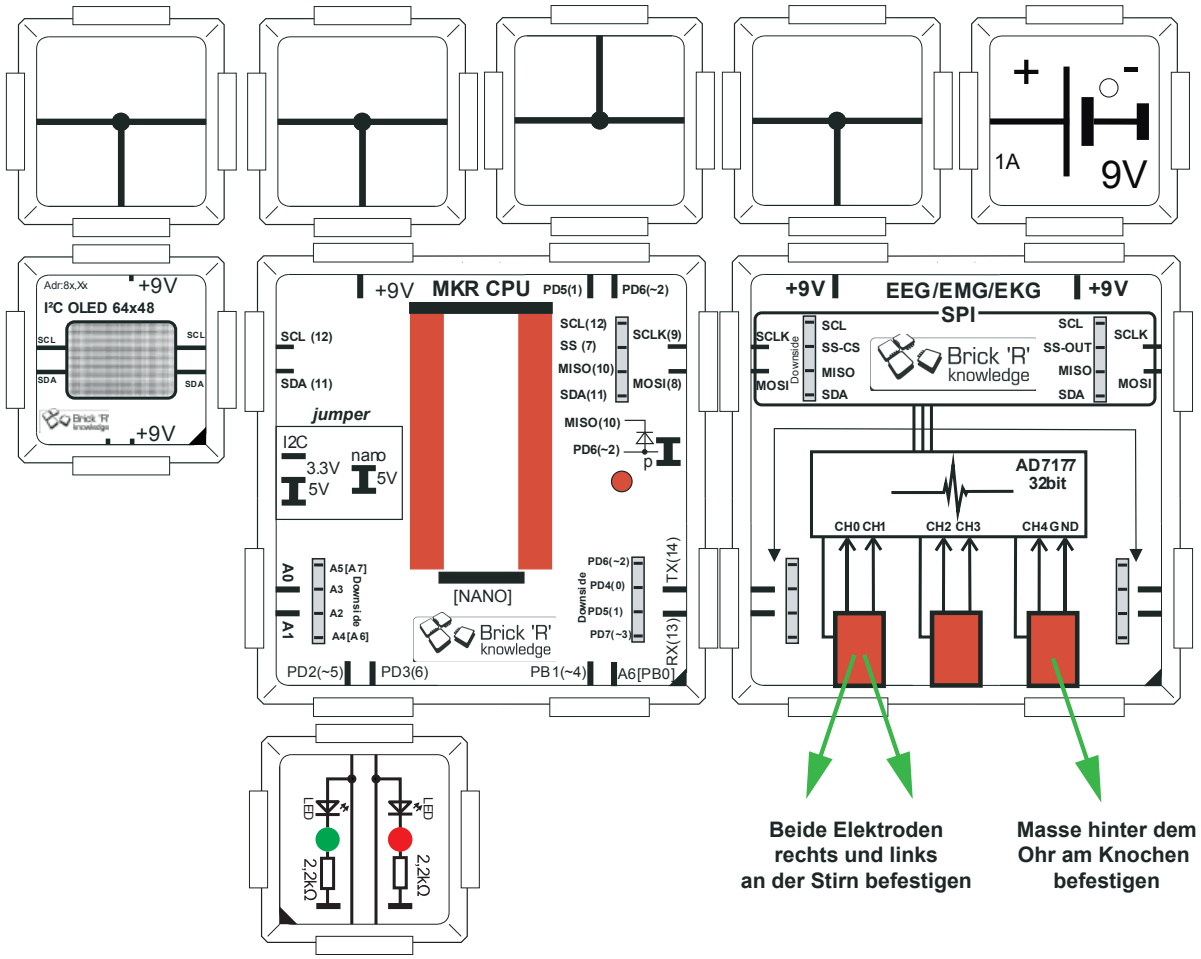


8.5 EEG Messung - Mind-Control einer LED

Die beiden Elektroden von CH0 und CH1 an der Stirn links und rechts befestigen, die Masse bei CH4/GND hinter dem Ohrläppchen. Man kann dort auch beide Elektroden anbringen.

Hier wird den Wellen, wenn Gamma grösser als Alpha ist, eine LED (grüne LED) zugeordnet. Wenn Alpha grösser als Gamma ist, leuchtet die rote LED. Beim Meditieren kann man so eine Funktion auslösen. Das Ganze erfordert natürlich etwas Geduld und Übung. Zur Kontrolle sei z.B. Hemisync empfohlen, eine Methode zur Erlangung bestimmter Meditativer Zustände.

Dann ruhige Position einnehmen. Den Versuch sollte man am besten zu zweit durchführen. Ausgabe auf dem grafischen OLED-Display.



Hier kann ein Programmcode ganz nach Belieben eingebaut werden, um die beiden LEDs zu aktivieren. Im Beispiel ist die interne LED (orange) auf dem MKR WIFI 1010 verwendet, man kann den Duo-LED-Brick dazu nehmen (die rechte, rote LED, von den Beiden entspricht der internen LED) und auch mit anderen Schwellwerten experimentieren.

Um die grüne LED auszulösen, müssen vorwiegend Alpha-Wellen vorhanden sein, also Entspannung und Konzentration ist angesagt. Dann kann man nach einiger Zeit die LED auch kontrollieren. Kann aber unterschiedlich sein, je nachdem, wie geübt man da ist. Augen dabei vorzugsweise schließen und zusammen mit einem Controller durchführen.

```
// EEG BRICK  Auswertung
//  MKR WIFI 2010 __SAMD21G18A__
// Output to serial Interface !!
```

```
// I2C Select: clken selen mosien led0 sense1 sense2 sense3 spez
//             0     1      2      3     4     5      6      7
//             out   out    out   out   in     in     in     in
//

#include <Wire.h> // Definitionen laden fuer I2C
#include <SPI.h>
#include <avr/pgmspace.h> // weitere Definitionen

#include "Adafruit_ZeroFFT.h"  // LIBRRAY FFT ZERO !!!!

#define DATA_SIZE 128
q15_t datasignal[DATA_SIZE]; // int16_t

//the sample rate
#define FS 100

// USe IOs of MKR
#ifdef ARDUINO_SAMD_MKRWIFI1010
#define __AVR_SAMD21__   1

#endif


// Alle Adressen bei den 8574xx bricks:
#define i2cIO9534_0 (0x40>>1) // Trick um elegant mit Bytes
#define i2cIO9534_1 (0x42>>1) // zu arbeiten statt in 7 Bit
#define i2cIO9534_2 (0x44>>1) // das letzte Bit ist das R/W
#define i2cIO9534_3 (0x46>>1) // dass von der Arduino
#define i2cIO9534_4 (0x48>>1) // Bibliothek dazugefuegt wird
#define i2cIO9534_5 (0x4A>>1) // wir definieren hier alle
#define i2cIO9534_6 (0x4C>>1) // Bereiche die man mit den
#define i2cIO9534_7 (0x4E>>1) // Bricks einstellen kann,.

#define i2cIO9534A_0 (0x70>>1) // Die Serie PCF8474AT
#define i2cIO9534A_1 (0x72>>1) // beginnt bei Adresse
#define i2cIO9534A_2 (0x74>>1) // 0x70 = 70 sedezimal
#define i2cIO9534A_3 (0x76>>1)  // 01110000 binaer
#define i2cIO9534A_4 (0x78>>1) // oder intern 0111000
#define i2cIO9534A_5 (0x7A>>1) // dabei 0111xxx
#define i2cIO9534A_6 (0x7C>>1) // mit x fuer die
#define i2cIO9534A_7 (0x7E>>1) // Dilschalterposition

#define myi2cIOadr i2cIO9534_0 // HIER PASSENDE ADRESSE EINTRAGEN

#define SSPB 10 // Pin 10 = SS  - --- PB2  1<<2 als Maske

// SPI: SCLK MOSI MISO und SS Brick=PB2 D10 fest !!
// MEGA Pin 51=MOSI Pin 52=SCK und Pin50=MISO °!!!
// mega:
// NANO: MOSI:PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// UNO: MOSI: PB3 D11, MISO: PB4 D12, -SS:PB2:D10
// ACHUNG: MISO ist auch STatus DATA READY !!!
//
// PORTB 4  (7 6 5 4 -- 3 2 1 0)
// BEI MKR PD6

#if defined(__AVR_ATmega2560__)  // Arduino MEGA2560
#define MISOPIN 50
#warning ("INFO : compiles for AT MEGA2560")
#elif defined(__AVR_SAMD21__)
#warning ("INFO : compiles for MKR WIFI1010")
#define MISOPIN 2  // READ ONLY MKR ueber Diode...
#else // NANO
#warning ("INFO : compiles rest: also NANO...")
#define MISOPIN 12
#endif

// #define MISOPIN 50

long p_disp[DATA_SIZE];
```

```
int numBrks = 0;
int ArdrsBrk[8];


// 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 ---------------------------------------------
// 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 enabless()
{
  //digitalWrite(SSPB, LOW);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, LOW);
  #else
   PORTB &= 0xfb; // Default = high 7 6 5 4 3 2low 1 0 PB2 low
  #endif
  delayMicroseconds(10);
}
void disabless()
{
  delayMicroseconds(10);
  #if defined(__AVR_SAMD21__)
    digitalWrite(7, HIGH);
  #else
    PORTB |= 1 << 2; // Default = high bit2 PB2
  #endif
  // digitalWrite(SSPB, HIGH);
  delayMicroseconds(10);
  //
}

//AD 7177-x
// Register
// SS:-------_________________----------

//
unsigned short ad71_readid()
{
  unsigned char ch1;
  unsigned short val1=0;
  enabless();
  ch1 = SPI.transfer(0x47); // COmm: -wen r-w ra5..9  r-1=1 read 0=writ
  val1 = SPI.transfer16(0); // ID
  disabless();
  return(val1);
}

//

//
#define NOCHAN 1

void ad71_init()
{
   enabless();
   // Register setzen...
   SPI.transfer(0x01); SPI.transfer16(0x800C);  // ADCMODE ! ref en, mode continouus, clock
Crystal : 100 00 000  0 000 11 00
   SPI.transfer(0x02); SPI.transfer16(0x0442);  // INTERFACE MOD data_stat(6)en w32 en --- 000
0   0 10 0    0 1   0 0  00 1 0  old 0x0042
   SPI.transfer(0x06); SPI.transfer16(0x060d);  // GPIO 000 0  0 11 0   0 0 00 1101
   // Channel 0
   SPI.transfer(0x10); SPI.transfer16(0x8001);  // CH0   enable set0 ain0 ain1 ... 10 00 00 00
000 0 0001
   // Channel 1
```

```
   SPI.transfer(0x11); SPI.transfer16(0x0043);  // CH1   enable set0 ain2 ain3 ... 10 00 00 00
010 0 0011
   // Channel 2
   SPI.transfer(0x12); SPI.transfer16(0x0096);  // CH2   enable set0 ain4 ref- ... 10 00 00 00
100 1 0110
   // Channel 3  optional
   SPI.transfer(0x13); SPI.transfer16(0x0232);  // CH3   enable set0 smp+ tmp- ... 10 00 00 10
001 1 0010
   // alle gleiches setup daher setup 0
   // SETup registers.
   SPI.transfer(0x20); SPI.transfer16(0x1300);  // SET0 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x21); SPI.transfer16(0x1300);  // SET1 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x22); SPI.transfer16(0x1300);  // SET2 : 000 1  0 0 1 1  0 0 00 0000
   SPI.transfer(0x23); SPI.transfer16(0x1300);  // SET3 : 000 1  0 0 1 1  0 0 00 0000
   // Filter Config gehoeren zu setup reg
   SPI.transfer(0x28); SPI.transfer16(0x050e);  // FLT0 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x29); SPI.transfer16(0x050a);  // FLT1 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2A); SPI.transfer16(0x050a);  // FLT2 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   SPI.transfer(0x2B); SPI.transfer16(0x050a);  // FLT3 : 0 000 0 101   x 00 0 1010 sps (00111
10k 01000 5k...)
   // OFFSET und GAIN auf factory lassen...
   disabless();

}

unsigned long values[4];

// tmperature 470uV/K
//         1181972248 beispiel bei 2.5V Ref+- 0.137mV 292K ->19.7
// 67BCCE0 108776672 -3.5


void ad71_reset()
{
  enabless();
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  disabless();
  delayMicroseconds(550);
}


void ad71_dataread()
{
  // Wenn ready incl status 4 byts 32 bit
  // STATUS
  unsigned char sts;
  unsigned char ch1;
  unsigned long val1=0;
  unsigned long val2=0;
  unsigned int idx;
  static int first = 0;
  int val=0;
  int i=0;
  for (i=0; i<NOCHAN;i++) {
    // SYNC:
    // NICHT RDY Abfragen,,,
  #ifdef XXXXXX
    enabless();
    do {

      sts = SPI.transfer(0x40);
      sts = SPI.transfer(0x00);
       delayMicroseconds(10);
       // Serial.print(sts);
```

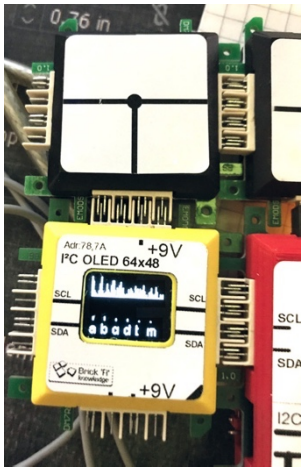
```
      // Serial.println("S wt:");

    } while (((sts & 0x80)== 0x80) && (first == 1));
    first = 1;
    disableSS();
#endif

    // DATA RDY
    delayMicroseconds(10);
    //
    enableSS();
    delayMicroseconds(4); // DRDY valid...
    // SINGLE MODE SPI.transfer(0x01); SPI.transfer16(0x801C);  // ADCMODE ! ref en, mode
SINGLE, clock Crystal : 100 00 000  0 000 11 00
    //
    // Sonst abfragen hier immer...
    do {
      val = digitalRead(MISOPIN); // MISOPIN
    } while (val == HIGH);

    //
    ch1 = SPI.transfer(0x44); // COmm:  data read

    //
    val1 = SPI.transfer16(0); // data <
    val2 = SPI.transfer16(0); // data
    //

     //
    delayMicroseconds(5); // DRDY valid...
    disableSS();
    //
    val1 = val2 | (val1 << 16); // 4 bit status
    idx = val2 & 0x3; // CHANNEL FLG  bit 3210
    values[idx] = val1 & 0xfffffff0;

  }
}


bool CheckForDevice(int Adrs){

  switch (Adrs){   // Startvorgang I2C Adresse
      case 0: Wire.beginTransmission(i2cIO9534_0);break;
      case 1: Wire.beginTransmission(i2cIO9534_1);break;
      case 2: Wire.beginTransmission(i2cIO9534_2);break;
      case 3: Wire.beginTransmission(i2cIO9534_3);break;
      case 4: Wire.beginTransmission(i2cIO9534_4);break;
      case 5: Wire.beginTransmission(i2cIO9534_5);break;
      case 6: Wire.beginTransmission(i2cIO9534_6);break;
      case 7: Wire.beginTransmission(i2cIO9534_7);break;
  }

  Wire.write(0x03) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0xF0) ;  // 1=input
  Wire.endTransmission(); // Stop Kondition setzen bei I2C

  switch (Adrs){     // Startvorgang I2C Adresse
      case 0: Wire.beginTransmission(i2cIO9534_0);break;
      case 1: Wire.beginTransmission(i2cIO9534_1);break;
      case 2: Wire.beginTransmission(i2cIO9534_2);break;
      case 3: Wire.beginTransmission(i2cIO9534_3);break;
      case 4: Wire.beginTransmission(i2cIO9534_4);break;
      case 5: Wire.beginTransmission(i2cIO9534_5);break;
      case 6: Wire.beginTransmission(i2cIO9534_6);break;
      case 7: Wire.beginTransmission(i2cIO9534_7);break;
  }

  Wire.write(0x01) ; // IO Ports auf 01010101 abwechseln
  Wire.write(0x0f) ;  // led an !!
  Wire.endTransmission(); // Stop Kondition setzen bei I2C

  //SPI ID Abfragen
  SPI.begin();
```

```
  SPI.beginTransaction(SPISettings(7000000, MSBFIRST, SPI_MODE3));
  Serial.print("Searching EEG in Adrs ");Serial.println(Adrs);
  unsigned short id=ad71_readid(); // 0x4fdx
  if ((id & 0xfff0)!= 0x4fd0) { //Kein EEG Brick hier gefunden
     Serial.print("ERROR id=");Serial.print(id);
     Serial.println(" EEG Brick here not found");
     return false;
  }
  else{ //EEG Hier gefunden -> return True
   Serial.print("id=");Serial.println(id);
   Serial.println(" EEG Brick found");
   // OK Init dr Register
   ad71_reset();
   ad71_init();
   return true;
  }
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Wire.begin(); // I2C aktivieren !
  Serial.println("Starting EEG:");

  #if defined(__AVR_SAMD21__)
   pinMode(MISOPIN, INPUT); // MISO !!
   pinMode(7, OUTPUT);
  #else
   pinMode(12, INPUT); // MISO !!
   pinMode(SSPB, OUTPUT);
  #endif
  disabless();

   pinMode(LED_BUILTIN, OUTPUT);
   pinMode(5, OUTPUT);

 i2c_oled_initall(i2coledssd);
 disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
 disp_print_xy_lcd(0, 0, (unsigned char*)"EEG/EKG/EMG", COLOR_WHITE, 0);
 disp_line_lcd (0,12,63,12, COLOR_WHITE);
 disp_print_xy_lcd(20, 20, (unsigned char*)"Starte...", COLOR_WHITE, 0);
 disp_lcd_frombuffer();


 for (int i = 7; i >= 0; i--){
  if (CheckForDevice(i)){
      ArdrsBrk[numBrks] = i;
      numBrks++;
      Serial.print(numBrks);Serial.println(" Gefunden");
  }
 }



 if (numBrks <= 0){ //Keine Bricks Gefunden
   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
   disp_print_xy_lcd(0, 20, (unsigned char*)"0 Gefunden", COLOR_WHITE, 0);
   disp_lcd_frombuffer();
   while (1){Serial.println("Nichts Gefunden");}
 }
 else{
   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
   disp_print_xy_lcd(0, 20, (unsigned char*)"Gefunden", COLOR_WHITE, 0);
   disp_lcd_frombuffer();
 }

//Init display schlange
   for(int x = 0; x < DATA_SIZE; x++){
        p_disp[x] = 0;
        datasignal[x] = 0;
   }


}
```

```
int findMin(){
  long minimum = p_disp[0];
  int c;
  for (c = 0; c < DATA_SIZE; c++)
    {
        if (p_disp[c] < minimum)
        {
           minimum = p_disp[c];
        }
    }
  return minimum;

  }

int findMax(){
  long maximum = p_disp[0];
  int c;
  for (c = 0; c < DATA_SIZE; c++)
    {
        if (p_disp[c] > maximum)
        {
           maximum = p_disp[c];
        }
    }
  return maximum;

  }

long lastval=0;
void loop() {

ad71_dataread();
 #ifdef ALLL
    Serial.print("ad:");Serial.print(values[0],HEX);
    Serial.print(" ");Serial.print(values[1],HEX);
    Serial.print(" ");Serial.print(values[2],HEX);
    Serial.print(" ");Serial.print(values[3],HEX);
    Serial.println();
 #endif

  for (int it = DATA_SIZE-1; it >= 1; it--) // ALLE DURCHSCHIEBEN !!!
  {
    p_disp[it] = p_disp[it-1];
  }
  long v1 = values[0]-0x800000001;
   p_disp[0] = v1;

 long  minim = findMin();
 long maxim = findMax();

 // IN FFT Kopieren:
 // und wandlen auf 16 bit range
 static int cnt=0;
 cnt++;
 if (cnt == 256) { //Wenn 256 Werte gesnammelt wurden...
   for (int j=0; j<DATA_SIZE-1;j++){
     datasignal[j] = map(p_disp[j], minim, maxim, -8000,8000);
   }
   // DANN FFT ::::  alle 256 mal
   ZeroFFT(datasignal, DATA_SIZE);
   // DORT ERGEBNIS min und max FFT
   int minfft,maxfft;
   minfft=datasignal[2];
   maxfft=datasignal[2];
   for(int i=0; i<DATA_SIZE/2; i++){
      if ((i>2) && (i<DATA_SIZE/2-1)) {
       if (minfft>datasignal[i]) minfft=datasignal[i];
       if (maxfft<datasignal[i]) maxfft=datasignal[i];
      }
      //print the frequency
      Serial.print(FFT_BIN(i, FS, DATA_SIZE));
      Serial.print(" Hz: ");
```

```
      //print the corresponding FFT output
      Serial.println(datasignal[i]);
    }
   disp_buffer_clear(COLOR_BLACK); // Virtuellen Buffer loeschen BLACK = dunkel
   // disp_print_xy_lcd(0, 0, (unsigned char*)"EEG", COLOR_WHITE, 0);
   // disp_line_lcd (0,12,63,12, COLOR_WHITE);
   // 64x48

   for (int i=0;i<63;i++) {
       long val;
       val = 16-(map(datasignal[i], minfft, maxfft, 0, 16));
       disp_line_lcd (i,16,i,val, COLOR_WHITE);
   }
   /*
   lastval= 47-(map(p_disp[0], minim, maxim, 0, 30));
   for (int i=0; i<128;i++) { // nun fuer alle 64 Spalten (x)  des Displays  128 DATEN
ausnahmweise direkt
       long val;
       val = 47-(map(p_disp[i], minim, maxim, 0, 30));
       disp_line_lcd (i/2-1,lastval,i/2,val, COLOR_WHITE);
       lastval = val;
    } // alle Spalten*/

     //EEG Frequenzen:
     // ALPHA: 5-18hz BETA: 16-31hz GAMMA: >32hz DELTA: <4 THETA: 4-7hz MU: 8-12
     long Alpha, Beta, Gamma, Delta, Theta, Mu;

     Alpha = datasignal[5];
     for (int i=5;i<=18;i++) {
      if (datasignal[i] > Alpha) Alpha = datasignal[i];
     }



     Delta = datasignal[4];


     Theta = datasignal[4];
     for (int i=4;i<=7;i++) {
     if (datasignal[i] > Theta) Theta = datasignal[i];
     }

     Mu = datasignal[8];
     for (int i=8;i<=12;i++) {
     if (datasignal[i] > Mu) Mu = datasignal[i];
     }


     Beta = datasignal[16];
     for (int i=16;i<=30;i++) {
     if (datasignal[i] > Beta) Beta = datasignal[i];
     }


     Gamma = datasignal[32];
     for (int i=32;i<=40;i++) {
      if (datasignal[i] > Gamma) Gamma = datasignal[i];
     }

     //Buchstaben reihe
    disp_print_xy_lcd(0, 37, (unsigned char*)"a", COLOR_WHITE, 0);  disp_print_xy_lcd(10, 37,
(unsigned char*)"b", COLOR_WHITE, 0);
    disp_print_xy_lcd(20, 37, (unsigned char*)"g", COLOR_WHITE, 0);  disp_print_xy_lcd(30, 37,
(unsigned char*)"d", COLOR_WHITE, 0);
    disp_print_xy_lcd(40, 37, (unsigned char*)"t", COLOR_WHITE, 0); disp_print_xy_lcd(50, 37,
(unsigned char*)"m", COLOR_WHITE, 0);

    //Linien   2,36,2,22

    disp_line_lcd (2,36,2,(map(floor(Alpha), 0, 300, 36, 22)), COLOR_WHITE);
    disp_line_lcd (12,36,12,(map(floor(Beta), 0, 300, 36, 22)), COLOR_WHITE);
    disp_line_lcd (22,36,22,(map(floor(Gamma), 0, 300, 36, 22)), COLOR_WHITE);
    disp_line_lcd (52,36,52,(map(floor(Delta), 0, 300, 36, 22)), COLOR_WHITE);
    disp_line_lcd (32,36,32,(map(floor(Theta), 0, 300, 36, 22)), COLOR_WHITE);
```

```
    disp_line_lcd (42,36,42,(map(floor(Mu), 0, 3200, 36, 22)), COLOR_WHITE);
    //disp_line_lcd (0,33,0,(map(Alpha, 0, 100, 33, 20)), COLOR_WHITE);

    disp_lcd_frombuffer(); // dann erst updaten, double buffer flimmerfrei
    cnt = 0; //Reset Count

    //MINDCONTROL AUSWERTUNG

    if (Alpha > Gamma) {
      digitalWrite(LED_BUILTIN, HIGH);
      digitalWrite(5, LOW);
    }
    else {
      digitalWrite(5, HIGH);
      digitalWrite(LED_BUILTIN, LOW);
    }
 } //ENDIF Werte
  // long v2 = values[1]-0x800000001;
  //Serial.print(v1);Serial.print(",");Serial.println(v2);
  Serial.println(v1);
}
```



Was passiert? Die Messwerte, die Gehirnströme werden als Frequenzanalyse mit Zuordnung zu Alpha, Beta usw. auf dem OLED-Display ausgegeben. Wenn mehr Alpha-Wellen als Gamma-Wellen existieren, dann wird die rechte (rote) LED eingeschaltet (und die orange, interne LED). Wenn mehr Gamma-Wellen als Alpha-Wellen existieren, dann wird die linke (grüne) LED eingeschaltet. Ziel ist es, durch Konzentration die linke (grüne) LED zum Leuchten zu bringen.

Eigene Trigger können in den Sketch eingebaut werden, wenn man den folgenden Auswertungs-Block modifiziert:

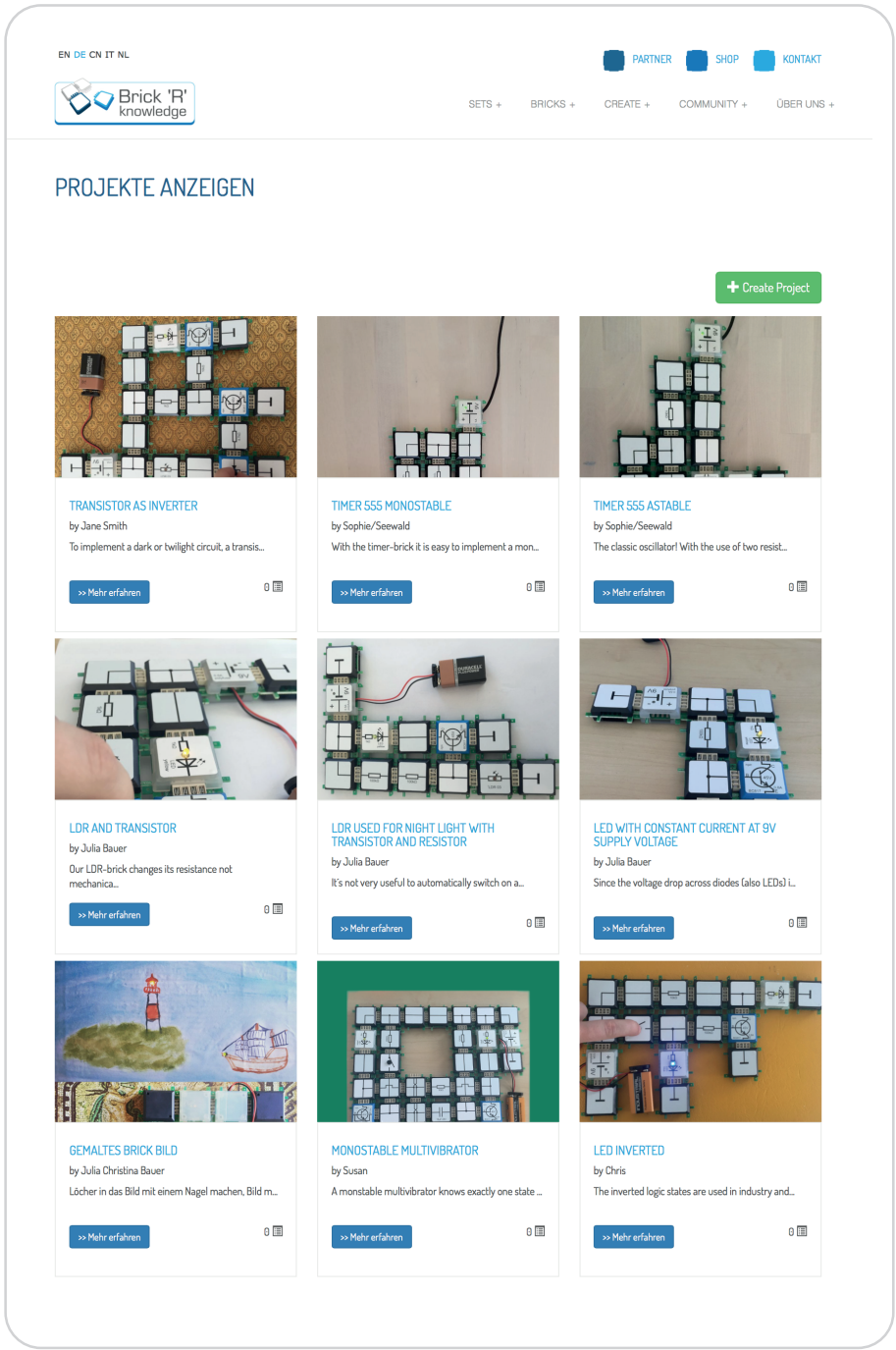
```
//MINDCONTROL AUSWERTUNG

if (Alpha > Gamma) {
  digitalWrite(LED_BUILTIN, HIGH);
  digitalWrite(5, LOW);
}
else {
  digitalWrite(5, HIGH);
  digitalWrite(LED_BUILTIN, LOW);
}
```

9. 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.

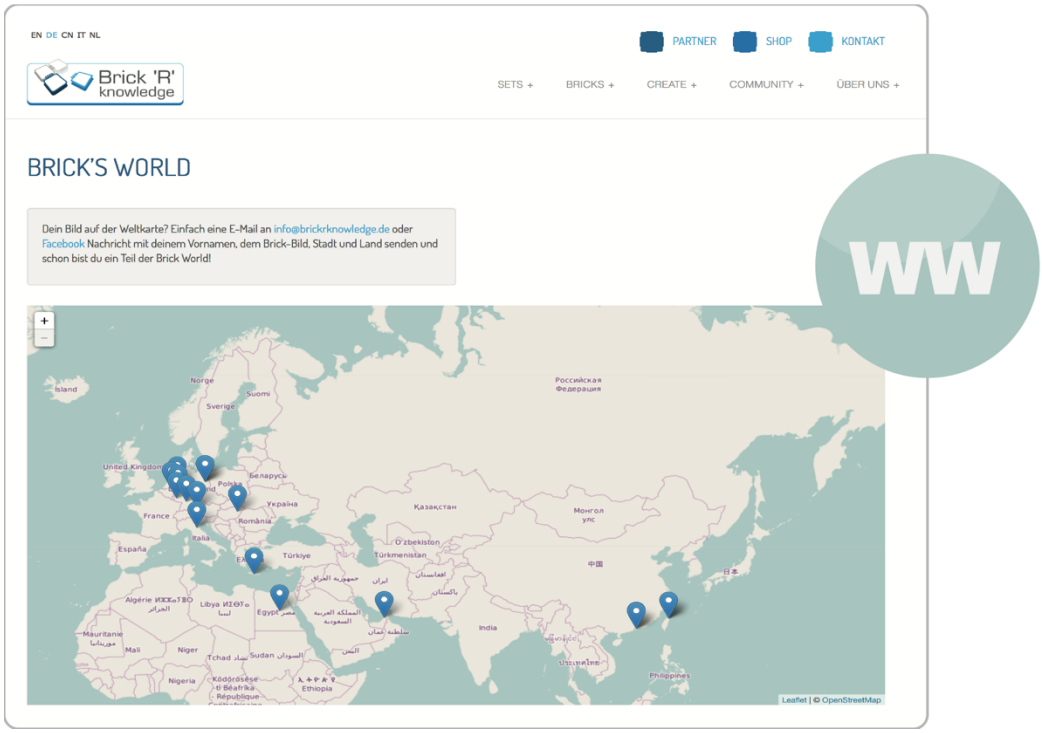


Social Media

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!



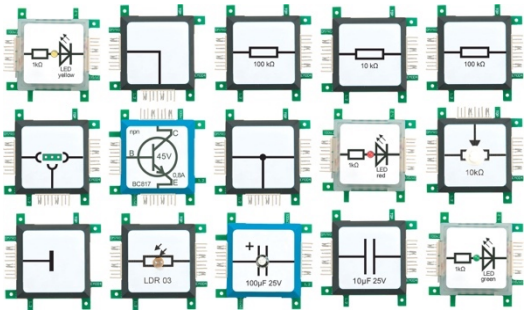
10. Brick Sets im Überblick

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.

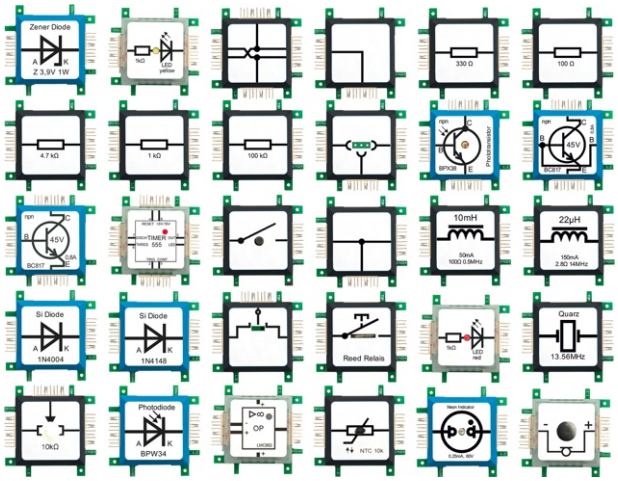


Advanced Set

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.

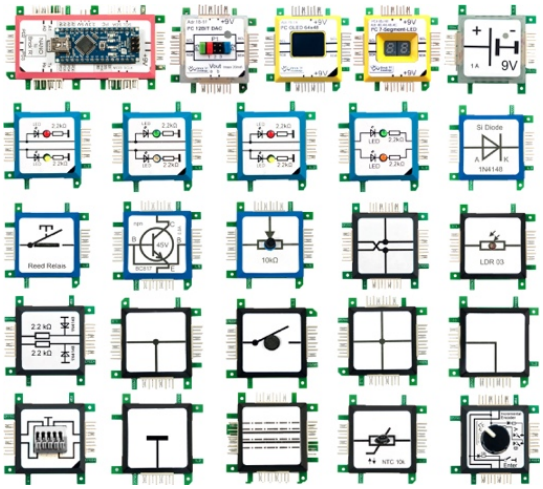


Arduino Coding Set

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.

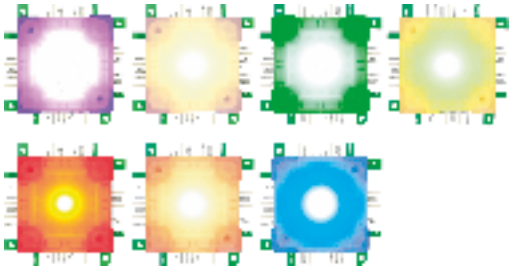


7 Color Light Set

enthält 28 Bricks

ALL-BRICK-0398

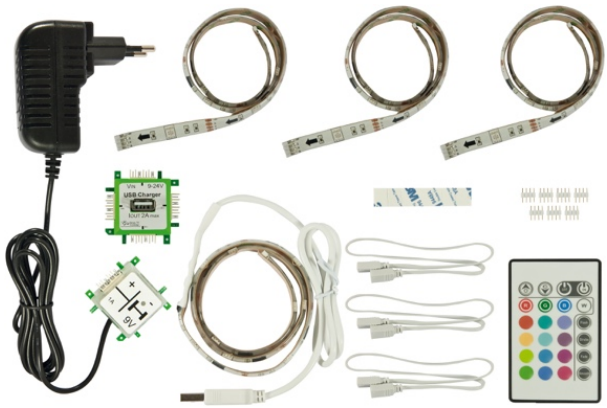
Mit den insgesamt 28 LED-Leucht-Bricks in 7 unterschiedlichen Farben lassen sich beeindruckende Lichtakzente in horizontaler und vertikaler Architektur setzen. Die 1 Watt LEDs in den Farben rot, gelb, blau, orange, violett, grün und warmweiß eignen sich perfekt für individuelle Licht-Figuren oder als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

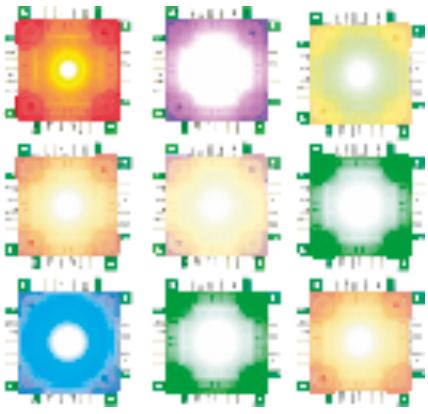


Programmable LED Set

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.

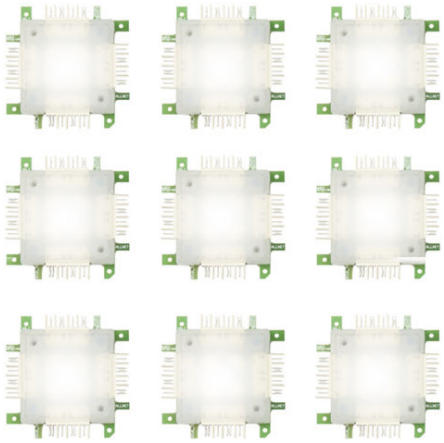


Highpower LED Set

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.

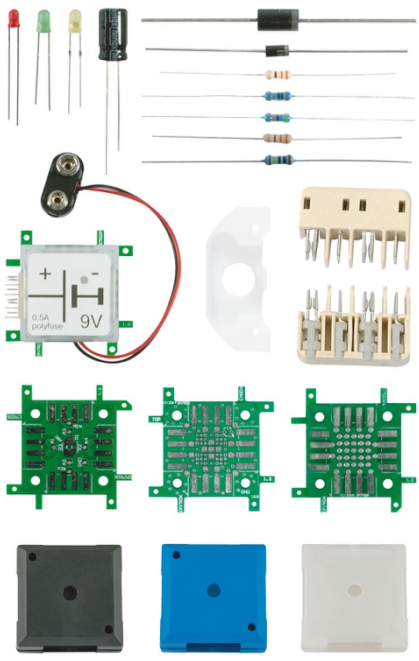


DIY Set

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.

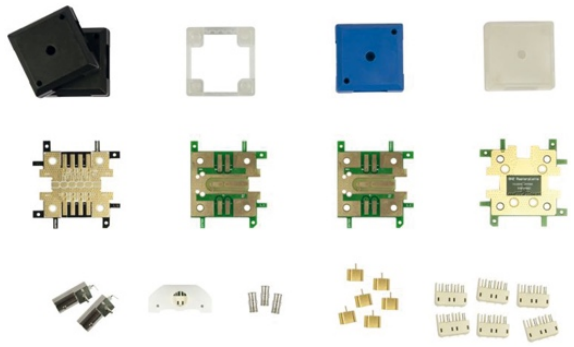
Mit Lötkolben und Lötzinn können die Tüftler die Standard-Bricks nachbauen oder eigene Bricks für individuelle Spezialanwendungen herstellen und somit sogar eigene Sets entwickeln.



MHz DIY Set

ALL-BRICK-0457

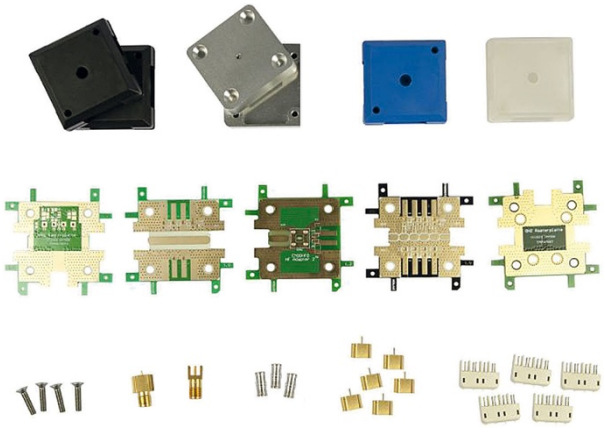
Mit dem MHz DIY Set lassen sich eigene Bausteine für Experimente und Schaltungen im MHz-Bereich realisieren. Das Set enthält drei verschiedene Raster- und Experimentierplatinen, sowie BNC-Buchsen, P-SMP-Stecker und die dazu passenden Verbinder. Außerdem enthält das Set eine Lötlehre für die SMD-Stecker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

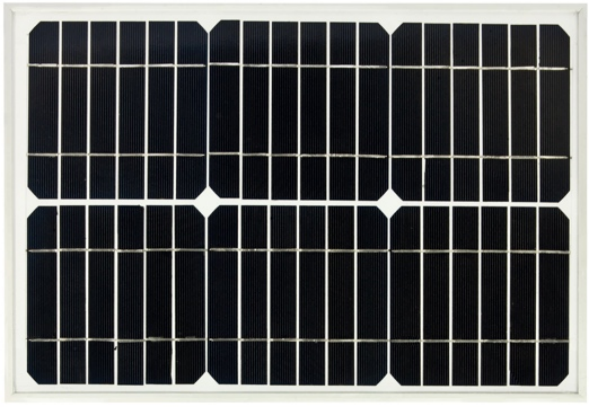
enthält 20 Bricks

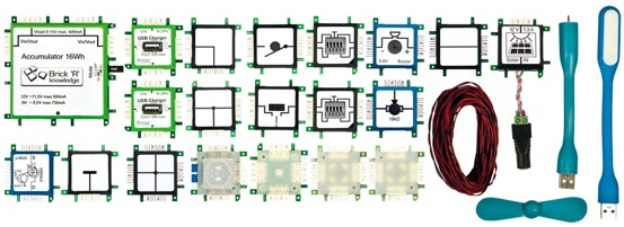
ALL-BRICK-0458

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.

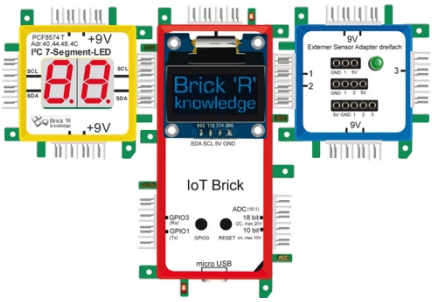
Internet of Things Set

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!

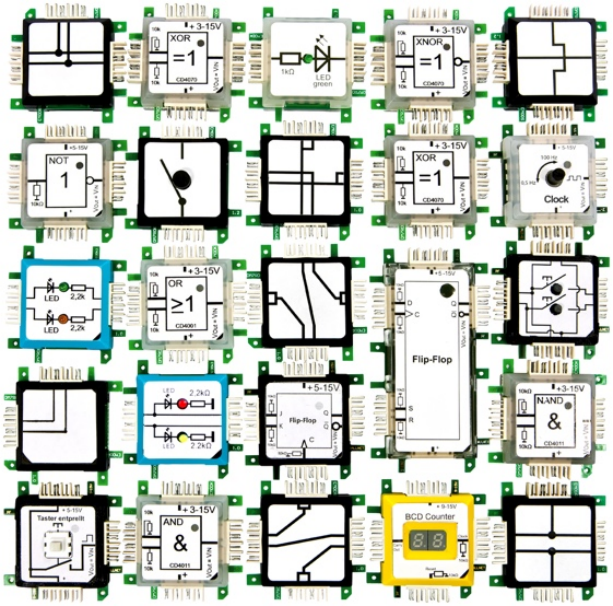


Logic Set

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.



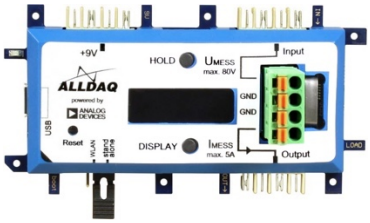
Der Lieferumfang des Logic Sets reicht von einfachen Logik-Bricks (AND, OR, NAND, NOR, XOR, XNOR, NOT) über verschiedene Flipflop-Bricks (D-, RS- und JK-Typ), weiter über einen Taktgeber-Brick (alternativ ein entprellter Taster für Einzelimpulse) bis hin zu einem BCD-Counter-Brick mit integrierter 7-Segment-Anzeige.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



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