CREATIVE - LAB WORKSHOP

SOLDERING THE SOLDERING STATION

3D-Print, SMD soldering and Arduino ALL INCLUSIVE!!

Prototyp 1 (blau) und Prototyp 2 (grün)

Der Workshop dauert 5x2 Stunden.

Während des Workshops wird eine hochwertige aber sehr preiswerte SMD-Lötstation von den Teilnehmern selbst gebaut und programmiert. Hat man einmal mit dieser Station gearbeitet will man nie wieder eine andere :).

Dabei werden Grundkenntnisse im SMD-Löten, in der Arduino Programmierung und dem 3D-Druck vermittelt. Die Teilnehmer können nach dem Workshop die eigene Station natürlich mit nach Hause nehmen (inklusive Gehäuse aus dem 3D-Drucker).

Für den gesamten Bausatz sind 100 Euro zu entrichten. Die Teilnehmerzahl ist auf 6 begrenzt. Bei großem Interesse wird der Workshop wiederholt.

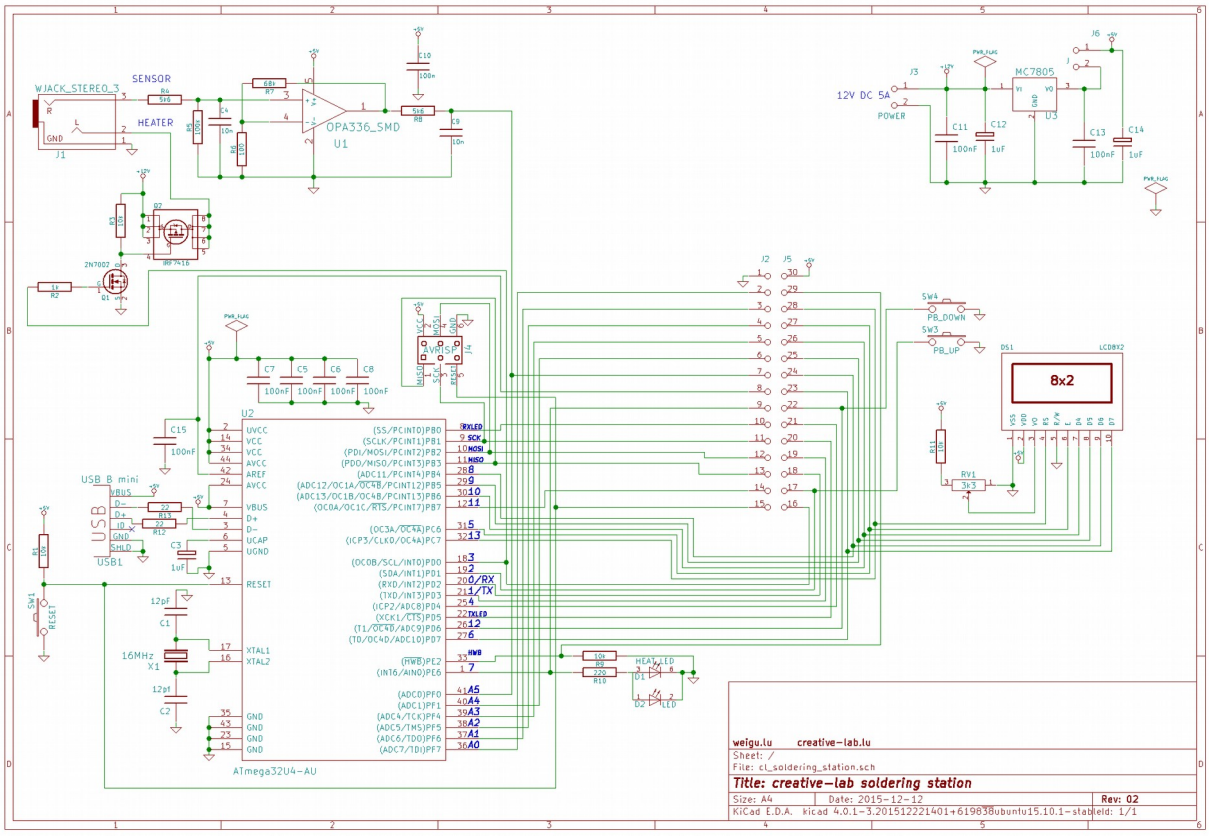
Alle benötigten Dateien und weitere Infos findet man auf www.creative-lab.lu.

Inhaltsverzeichnis

PART 1: SMD

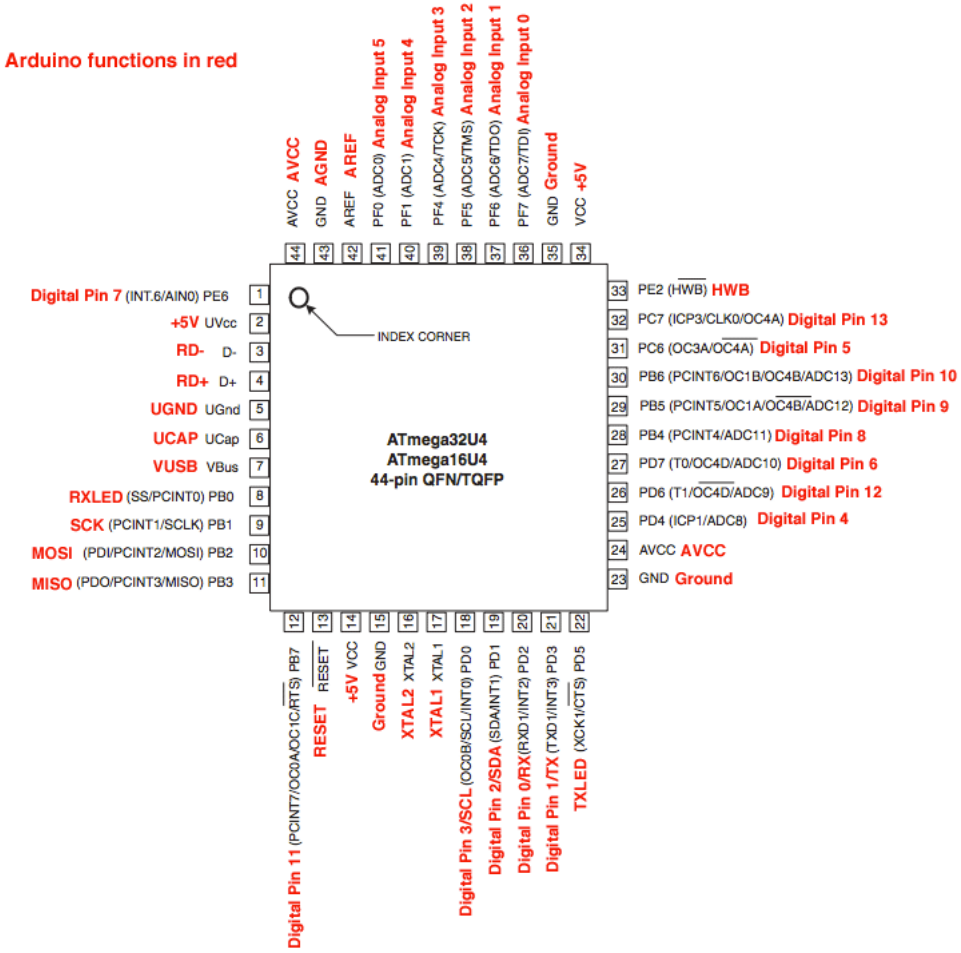
1. Funktionsweise (Schaltung) und Herstellen der Platine

Die Firma Weller verkauft Ersatz-Lötspitzen für Ihre WMRS_Halter. Dies Spitzen vereinen ein Heizelement und ein Thermoelement in einer sehr feinen Lötspitze, die hervorragend zum SMD-Löten geeignet ist. Der Clou ist, dass die Spitzen einen 3,5mm Stereo-Klinkenstecker besitzen, der den Anschluss an eine eigene Station sehr vereinfacht.

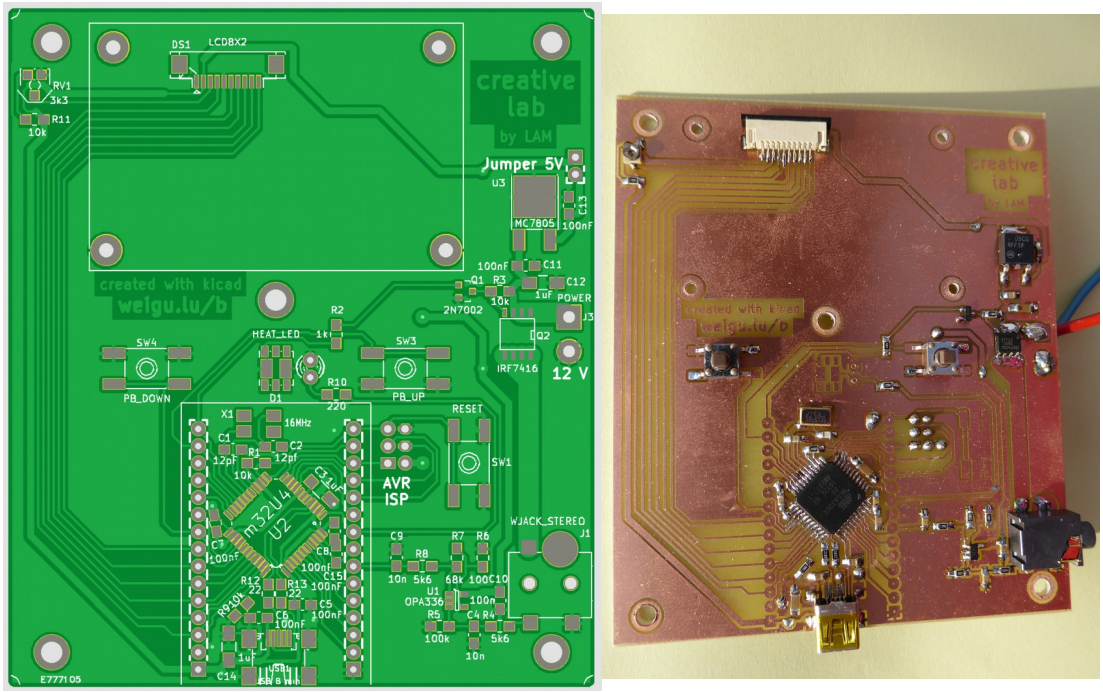


Die Schaltung besteht aus einem selbst gebauten Arduino Leonardo bzw. Micro (ATmega32u4, der Micro ist ein geschrumpfter Leonardo) mit Display, LED und zwei Tastern, einem Spannungsregler (5V aus 12V), einer Verstärkerschaltung zum Messen der Temperatur, und einer Treiberschaltung zum Heizen der Lötspitze.

Damit die Lötspitze nicht zu schnell abnutzt, wird über ein Federkontakt (verbunden mit A0, nicht im Schaltplan eingezeichnet!) ermittelt ob die Spitze in Ruhestellung ist. Die Klinkenbuchse muss aus Metall sein und über das Kabel mit Masse verbunden sein. Ist die Lötspitze eingehängt, so wird die Temperatur auf 50°C begrenzt. Da die Station in wenigen Sekunden aufheizt, muss bei Löten nie gewartet werden.



Leonardo Belegung der ATmega32u4 Pins: https://www.arduino.cc/en/Hacking/PinMapping32u4



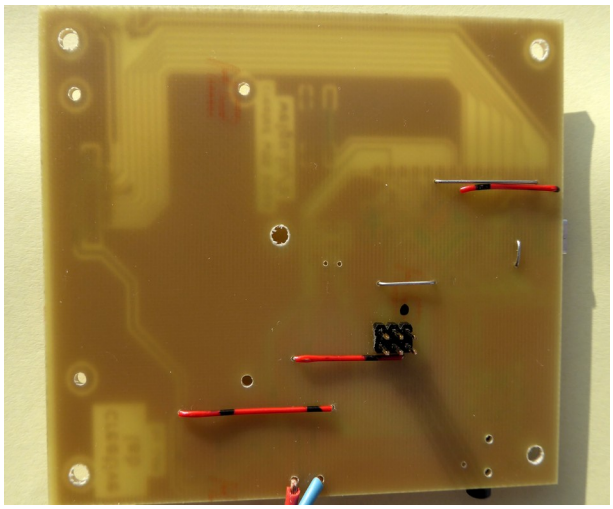
Professionelle hergestellte Platine und teilweise bestückter Prototyp 2

2. Die Materialliste (BOM)

Widerstand SMD 0805 1%	1	R7	68k	mouser.com	667-ERJ-6ENF6802V
Widerstand SMD 0805 1%	1	R6	100Ω	mouser.com	667-ERJ-6ENF1000V
Widerstand SMD 0805 1%	1	R5	100k	mouser.com	667-ERJ-6ENF1003V
Widerstand SMD 0805 1%	2	R4,R8	5k6	mouser.com	667-ERJ-6ENF5601V
Widerstand SMD 0805 1%	1	R2	1k	mouser.com	667-ERJ-6ENF1001V
Widerstand SMD 0805 1%	1	R10	220Ω	mouser.com	667-ERJ-6ENF2200V
Widerstand SMD 0805 1%	4	R1,R3,R9,R11	10k	mouser.com	667-ERJ-6ENF1002V
Widerstand SMD 0805 1%	2	R12,R13	22Ω	mouser.com	667-ERJ-6ENF22R0V
Trimmer SMD 50volt 0.1W	1	RV1	33k 0,1W lin	mouser.com	81-PVZ3A332C01R00
Kondensator MLCC 0805 50volts	1	C1,C2	12pF C0G 5%	mouser.com	80-C0805C120J5G
Kondensator MLCC 0805 50volts	2	C4,C9	10n	mouser.com	80-C0805C103K5R
Kondensator MLCC 0805 50volts	8	C5-C8,C10, C11,C13,C15	100nF X7R 10%	mouser.com	80-C0805C104K5R
Kondensator MLCC 1206 50volts	3	C3,C12,C14	1uF X7R 10%	mouser.com	80-C1206C105K5R
LED SMD Red PLCC2	1	D1	Rot	mouser.com	78-VLMD3100-GS08
MOSFET 2N7002 SOT23	1	Q1	2N7002	mouser.com	512-2N7002
MOSFET IRF7416 SOIC8N	1	Q2	IRF7416	mouser.com	942-IRF7416TRPBF
OPV OPA336 SOT23-5	1	U1	OPA336	mouser.com	595-OPA336NA/3K
ATMEGA32U4-AU TQFP44	1	U2	ATMEGA32U4	mouser.com	556-ATMEGA32U4-AU
Spannungsregler MC7805 TO252	1	U3	MC7805	mouser.com	863-MC7805CDTRKG
Taster 9,5mm	2	SW3,SW4	Taster 9,5mm	mouser.com	688-SKHHAN
2reihige Stiftleiste	1	J4	AVR ISP Stecker	mouser.com	517-30306-6002HB
Klinkenbuchse 3.5mm Stereo SMT	1	J1	Klinkenbuchse	mouser.com	806-STX-35332-4N-TR
Mini USB Typ B Buchse SMT	1	USB1	Mini USB Buchse	mouser.com	538-54819-0572
Netzschalter 1polig	1	Netz	Netzschalter	mouser.com	540-SRB22A2FBBNN
LCD C0802-04	1	(DS1)	8*2	pollin.de	94-120622
Flexprint-Buchse Samtec_ZF1-10	1	DS1	10-polig RM1	pollin.de	94-452069
Quarz 16MHz 6*3,5mm	1	X1	16MHz	pollin.de	94-230128
Netzteil 12V	1	(J3)	12V /5A	pollin.de	94-351534
Lötspitze Weller RT1	1		Weller RT1	reichelt.de	WELLER RT 1
Verlängerung Klinke 1,5m (Metall)	1		Klinke 3,5mm	reichelt.de	HDG AC0110-015
Platine Eurocircuit	1		Platine	eurocircuits	
Gehäuse 3D Printer PLA 200g					
Schrauben, Federkontakt, Kabel					

3. Eine eigene Platine herstellen

Die Platine wurde mit der freien Software Kicad gezeichnet. Die Dateien befinden sich auf der Homepage (www.creative-lab.lu). Da die Platine einseitig ist und nur 6 Drahtbrücken benötigt, kann sie leicht selbst hergestellt werden. Dazu wird die Lötseite auf eine durchsichtige Folie oder Kalkpapier gedruckt (Print-Funktion in Kicad). Dann wird eine lichtempfindliche Platine belichtet und entwickelt. Nach dem Ätzvorgang kann die Platine gebohrt und anschließend bestückt werden.



Zu Bohren sind 5 Löcher (3 mm) zum Befestigen der Platine, 4 Löcher zum Befestigen des Displayhalters (2 mm, dieser wird später mit dem 3D-Drucker gedruckt), 2 Löcher zum Anlöten der 12V Spannungsversorgung (1,5 mm), 2 Löcher für die Klinkenbuchse (1,5 mm), 6 Löcher für den ISP-Stecker (0,9 mm), 12 Löcher für die Drahtbrücken (0,8 mm), 2 Löcher für den Jumper (0,9 mm) und 1 Loch an Pin A0 (PF7) für den Federkontakt (0,9 mm). Falls eine Standard-LED statt einer SMD-LED verwendet werden soll, zusätzlich 2 Löcher (0,8 mm).

Es werden bei der Platine 6 Drahtbrücken benötigt (3 mal 5V und 3 mal GND)

Einfacher ist es natürlich Gerber-Dateien an eine Firma zu senden und die Platine herstellen zu lassen. Das Bohren und die Lötbrücken entfallen und die Platine ist besser Lötbar.

4. SMD-Löten

Mit der richtigen Lötspitze ist SMD-Löten gar nicht so schwierig.

Folgende Werkzeuge werden neben der SMD-Lötstation benötigt:

- Feine Pinzette
- Entlötsaug-Litze
- Feiner bleihaltiger Lötzinn (0,5 mm)
- Flußmittelgel EDSYN FL 22
- Becherlupe min. 8x

Sehr wichtig ist natürlich auch das Ausleuchten des Arbeitsplatzes. Hier haben sich schwenkbare LED Tischleuchten bewährt. Eventuell auch mit Lupe, wenn man unter der Lupe löten möchte.

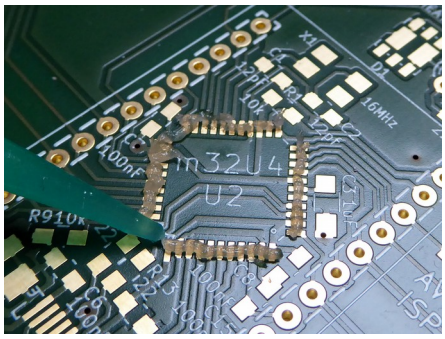
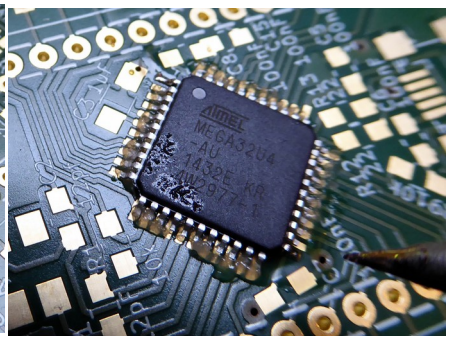
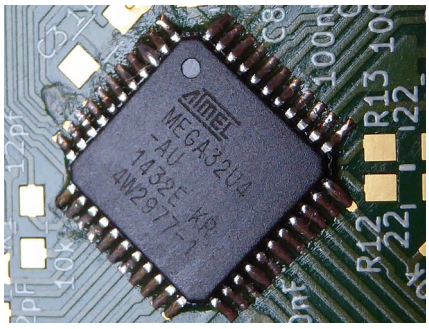
Für einfache Bauteile mit wenigen Anschlüssen wird zuerst eine Lötfläche verzinnt. Dann wird das Bauteil mit der Pinzette aufgehoben und mit einem Bein oder Anschluss fest gelötet (dabei muss kein sauberes Löten erfolgen; das Bauteil muss nur in der richtigen Position haften bleiben).

Sehr wichtig ist jetzt die Ausrichtung des Bauteils. Ist das Bauteil nicht optimal ausgerichtet, so wird die Lötstelle nochmals erhitzt und das Bauteil mit den Spitzen der Pinzette verschoben bis sich alle Anschlüsse mittig auf den Lötflächen befinden (ev. Kontrolle mit der Becherlupe). Hat sich das Bauteil auf einer Seite von der Platine abgehoben, wird die Lötstelle erhitzt und das Bauteil mit der Pinzette nach unten gedrückt.

Das Ausrichten muss manchmal mehrfach wiederholt werden. Nicht die Geduld verlieren! Im Zweifelsfall mal kurz aufstehen, tief durchatmen und noch einmal neu ansetzen.

Danach werden die restlichen Anschlüsse verlötet. Zum Schluss wird der erste Anschluss noch einmal mit ein wenig Zinn erhitzt um eine saubere Lötstelle zu erhalten.

Bei Bauteilen mit mehren Anschlüssen (bei uns nur der Mikrocontroller und ev der 8-polige Mosfet) werden zuerst alle Anschlüsse mit Flussmittelgel versehen.

Dann wird das Bauteil mit einem Bein fest-gelötet. Der nächste Schritt ist wie oben das Ausrichten des Bauteils. Hier ist noch viel mehr Sorgfalt bei der Ausrichtung von Nöten. Alle 44 Pins müssen sich mittig auf den Lötflächen befinden (Kontrolle mit der Becherlupe)! Dann wird ein zweites Bein gegenüber dem ersten fest gelötet, damit das Bauteil nicht mehr verrutschen kann. Jetzt kommen die restlichen Pins dran.

Die kann mit ein wenig Geschick unter Zugabe von wenig! Lötzinn rutschweise, oder aber einzeln verlöten. Es kommt oft vor, dass einzelne Pins durch die Kapillarwirkung zusammen gelötet werden. Dies ist überhaupt nicht schlimm und kaum vermeidbar. Hier kommt jetzt die Entlötsaug-Litze zum Einsatz. Sie wird erhitzt und entlang den verlöteten Pins gezogen. Der überschüssige Zinn verschwindet. Danach sind die Lötstellen einzeln mit der Becherlupe zu inspizieren und gegebenenfalls mit noch ein wenig Lötzinn zu versehen. Ist man sich nicht sicher, so ist eine Kontrolle der Anschlüsse auf Kurzschluss mit einem Durchgangsprüfer (Multimeter) durchzuführen.

Hat man Probleme mit der Litze, so kann man zusätzlich Flussmittelgel zugeben. Das Flussmittelgel setzt die Temperatur herunter und lässt den Zinn sauber verlaufen.

Ist alles schief gelaufen, und das Bauteil sitzt schief oder es sind Kurzschlüsse unter dem Bauteil entstanden, so kann dieses recht leicht mit einer Heißluft-Station entlöten. Diese Stationen kriegt man schon für unter 100 Euro. Das Bauteil wird dabei gleichmäßig erwärmt. Durch kräftiges Schütteln der Platine oder unter Zuhilfenahme der Pinzette wird das Bauteil entlötet.

Die Bauteile sind weniger empfindlich als man annimmt. Mikrocontroller haben nach 3-maligen Entlöten noch immer einwandfrei ihren Dienst verrichtet.

5. Die Bestückung des Creative Lab Leonardo CLEO

Man beginnt mit dem Mikrocontroller. Wie oben beschrieben ist zuerst Flussmittelgel aufzutragen. **Achtung auf die Polung (Ausrichtung) des ATmega32u4**. Der Punkt auf dem Gehäuse (Pin 1) muss sich rechts befinden!! Die Platine liegt dabei so vor uns, dass die Schrift der Kupferpberfläche (creative lab) lesbar ist (USB Stecker unten).

Danach wird der 16MHz Quarz verlötet. **Er ist so auszurichten, dass die Schrift auf dem Quarz lesbar ist**. Es reicht die Pins rechts oben und links unten zu verlöten. Dabei muss der Quarz an den Ecken erhitzt werden.

Als nächstes werden alle Bauteile innerhalb des Arduino-Leonardo Blocks verlötet, so dass dieser getestet werden kann. Im einzeln sind das die Kondensatoren C1 und C2 (12pF), C5, C6, C7, C8, und C15 (100nF), C3 und C14 (1µF), die Widerstände R1 und R9 (10k), R12 und R13 (22Ω) und die USB-Buchse.

Bevor wir weiter Bestücken wird dieser Teil der Platine getestet:

6. CLEO in Betrieb nehmen

Vor der Inbetriebnahme empfiehlt sich eine Sichtkontrolle mit einer Lupe. Im Zweifelsfall sollte unbedingt mit dem Durchgangsprüfer (Ohmmeter) nachgemessen werden.

Dann wird mit dem Ohmmeter überprüft ob kein Kurzschluss zwischen 5V und Masse vorliegt.

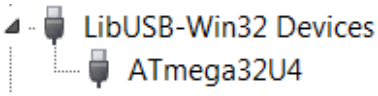
Ist dies nicht der Fall, so können wir die Platine mit einem PC verbinden.

Die Schaltung mit dem ATmega32u4 (16MHz) entspricht der Grundschaltung des Arduino- oder Genuino- Micro-Boards. Von Haus aus besitzt der ATmega32u4 schon einen Bootloader. Dadurch können wir erkennen ob der Quarz und die USB-Buchse richtig angeschlossen sind.

In Linux setzen wir das Kommando "**`lsusb`**" ab. In der List muss der folgende Eintrag zu finden sein:

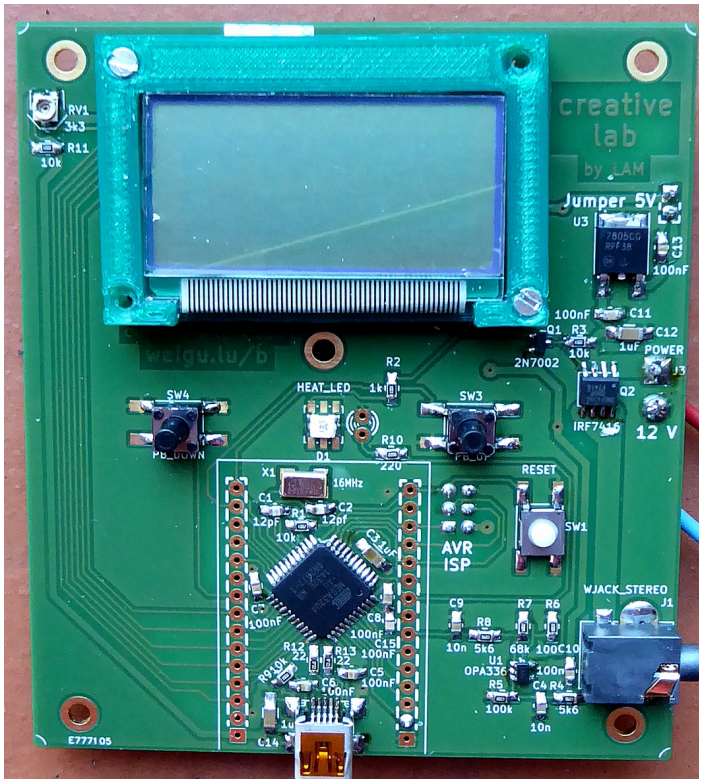
```
Bus 003 Device 004: ID 03eb:2ff4 Atmel Corp. atmega32u4 DFU bootloader
```

In Windows sehen wir im Device Manager (Control Panel) den folgenden Eintrag:



Die ATMEL Software Flip nutzt die LibUSB. Ist sie nicht installiert, so erhalten wir über den Installationsassistenten eine Meldung dass ein Treiber installiert werden muss. Hier erkennt man am Eintrag ATmega32u4DFU, dass alles in Ordnung ist. Der Treiber muss nicht installiert werden, also einfach abbrechen.

7. Die Platine weiter bestücken



Als nächstes werden die restlichen Bauteile bestückt. Zuerst die kleinen Bauteile wie der OPV und die beiden Mosfets (**Achtung auf die richtige Polung des IRF7416**; die Schrift muss verdreht sein Pin 1 oben rechts!). Dann die restlichen Widerstände und Kondensatoren. Bei der **LED ist auf die richtige Polung zu achten** (Anode rechts). Mit einem Multimeter (Diodenfunktion) kann die Ausrichtung vor dem Löten! überprüft werden. Will man eine Standard-LED (2 Löcher) verwenden, so wird die SMD-LED natürlich nicht bestückt.

Jetzt werden die größeren Bauteile bestückt. Wir beginnen mit dem Festspannungsregler (7805). Hier muss die große Fläche zuerst verzinnt werden. Der Regler wird dann zuerst auf diese Fläche aufgelötet. Sie muss ausreichend erhitzt werden!

Der Flachband Stecker (LCD) wird auch seitlich verlötet um die Festigkeit zu erhöhen.

Die beiden Taster müssen präzise zentriert werde. Falls es sich nicht um SMD-Taster handelt, sind die Anschlüsse einfach abzuschneiden. Die Taster werden an den übriggebliebenen Stutzen fest gelötet.

!!Die Bestückung des Reset-Tasters ist optional.

Zum Schluss werden die 12V Kabel angelötet sowie der AVR-ISP-Programierstecker und der 5V Jumper.

ACHTUNG: **Die Bezeichnung des 12V Anschlusses auf der Platine ist nicht eindeutige!! Der Pluspol 12V (rot) befindet sich am oberen viereckigen Lötpunkt!!**

Der AVR-ISP-Programmierstecker und der Jumper befindet sich auf der Rückseite der Platine!!

Für das Pin A0 (PF7) kann ein Lötstift verwendet werden (**Rückseite der Platine!!**) um später den Federkontakt anzuschließen.

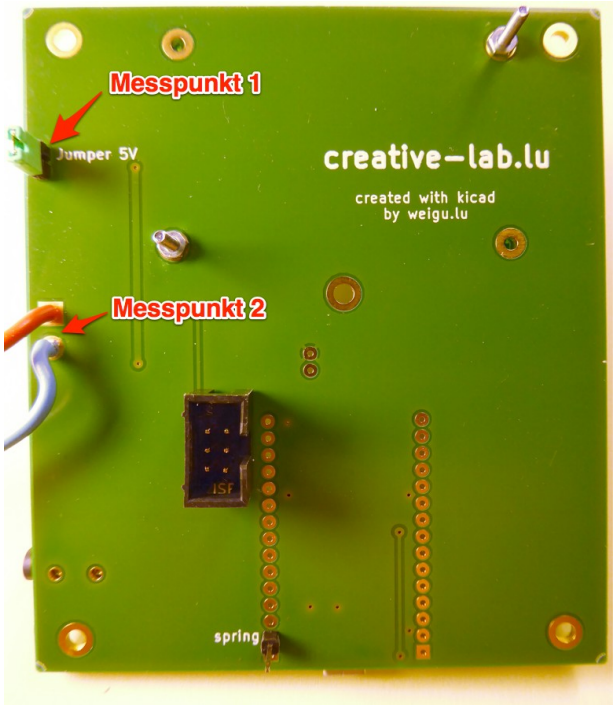
8. Die Platine in Betrieb nehmen

!! Die 12V Spannungsversorgung und die USB-Buchse nur gleichzeitig anschließen wenn die Spannungsregelung überprüft wurde !!

Vor der Inbetriebnahme empfiehlt sich:

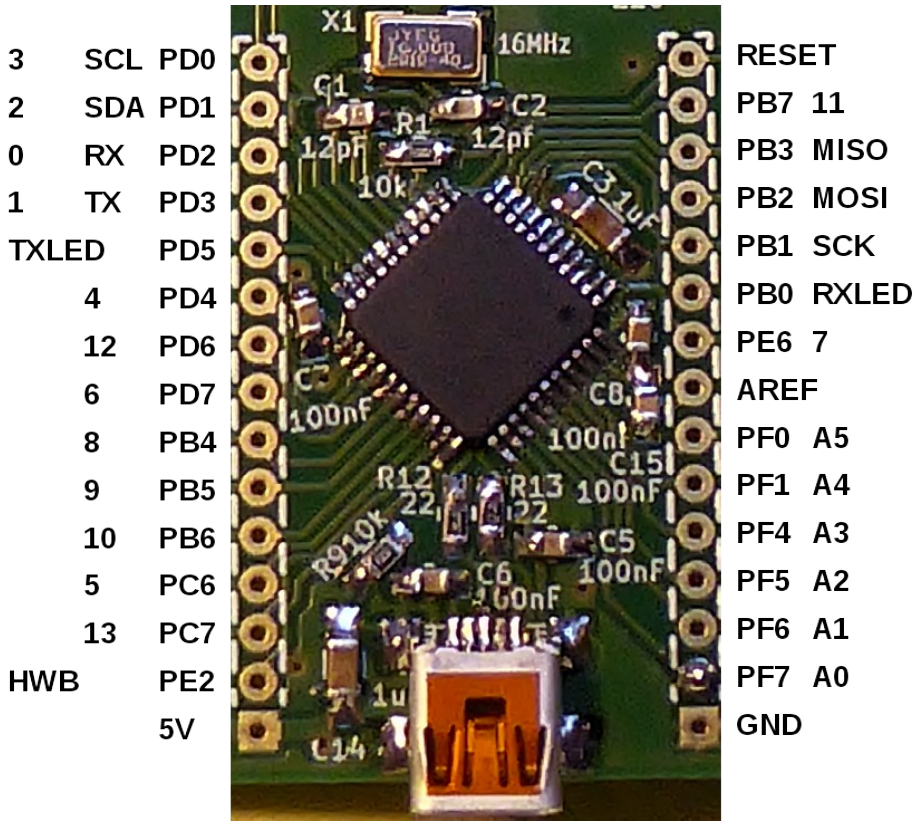
1. Eine Sichtkontrolle mit einer Lupe. Im Zweifelsfall sollte unbedingt mit dem Durchgangsprüfer (Ohmmeter) nachgemessen werden.
2. Eine Überprüfung mit dem Ohmmeter ob kein Kurzschluss zwischen 12V und Masse bzw. zwischen 5V und Masse vorliegt.

   Dazu berührt man die zuerst die beiden Lötstellen am Messpunkt 2 mit den Spitzen des Ohmmeters. Ist alles OK, so zeigt dieser an, dass der Widerstand zwischen den Punkten hoch ist und demnach keine Verbindung besteht. Dann misst man am zwischen dem oberen Pin des Jumpers (Messpunkt 1, Jumper nicht gesteckt!) und der Masse (untere Lötstelle am Messpunkt 2 (rundes Lötpad). Auch hier muss der Widerstand hoch sein.
3. Ist dies nicht der Fall oder hat man Zweifel an daran dass man alles richtig angeschlossen hat, so schließt man ein regelbares Netzteil mit Strombegrenzung an die 12V Versorgungsbuchse an. Die Strombegrenzung setzt man auf ungefähr 30-50 mA. Dann fährt man die Spannung langsam hoch bis auf 12V und misst dabei die 5V am Ausgang des 7805 oder am „Jumper 5V“.
4. Ist alles in Ordnung klemmt man die 12V wieder ab und verbindet die Schaltung über die USB-Buchse mit den PC.



9. Die Stiftbelegung des CLEO

Der Creative Lab Leonardo (CLEO) kann auch aus der Platine herausgeschnitten werden und als eigenständiges Arduino-Board genutzt werden. Hier die Stiftbelegung unseres CLEO:



Folgende Pins werden auf der Platine nicht benötigt und stehen für Hacks zur Verfügung:

0, 1, 4, 11, A1, A2,A3 und A4

https://youtu.be/HPI_8dhBebM

PART 2: Arduino

1. Programmierung des Arduino Bootloaders[1]:

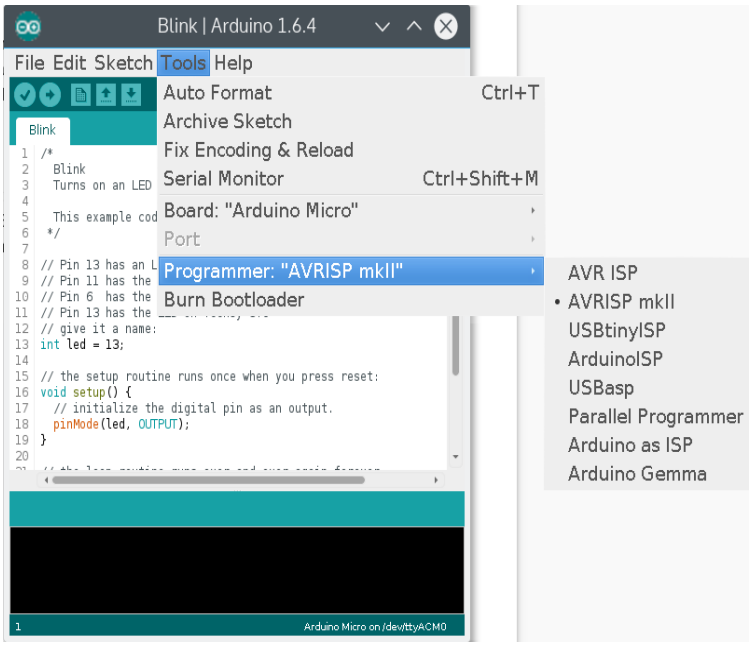
Wie im ersten Teil beschrieben besteht die Schaltung aus einem selbst gebauten Arduino Leonardo bzw: Micro (ATmega32u4, der Micro ist ein geschrumpfter Leonardo) mit Display, LED und zwei Tastern, einem Spannungsregler (5V aus 12V), einer Verstärkerschaltung zum Messen der Temperatur, und einer Treiberschaltung zum Heizen der Lötspitze.

Da wir den Controller über die Arduino-SDK[2] programmieren wollen, müssen wir den ATMEL-Bootloader ersetzen durch einen Arduino_Micro Bootloader. Dazu benötigen wir allerdings ein Programmiergerät (z.B. ATMEL AVRISP mkII). Es kann auch ein anderes Arduino-Board als Programmiergerät genutzt werden (https://www.arduino.cc/en/Tutorial/ArduinoISP).

Das Programmiergerät ist mit dem 6 poligen ISP Stecker auf der Rückseite der Platine zu verbinden. Achte auf die Polung falls kein verpolungsicherer Stecker verwendet wird!

Die Schaltung wird über den USB Stecker mit dem PC verbunden. **Der Jumper zum Spannungsregler muss entfernt werden falls die 12V Spannungsversorgung noch nicht angeschlossen wurde.**

In der Arduino-SDK (runterladen unter https://www.arduino.cc/en/Main/Software) muss unter *Tools* jetzt das Arduino-Micro-Board ausgewählt werden.



Danach wird einfach auf "*Burn Bootloader*" geklickt.

Nach dem Brennen des Bootloaders wird das Programmiergerät wieder entfernt.

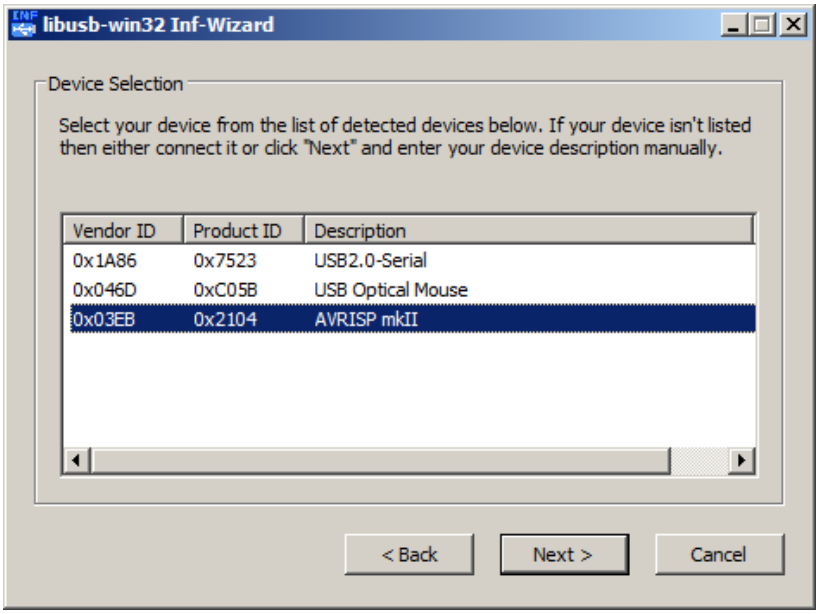
1 Ein Bootloader ist ein Stück Software das sich im Mikrocontroller befindet und ermöglicht diesen ohne spezielles Programmiergerät zu programmieren.

2 Ein **Software Development Kit** (**SDK**) ist eine Sammlung von Werkzeugen und Anwendungen, um eine Software zu erstellen (https://de.wikipedia.org/wiki/Software_Development_Kit).

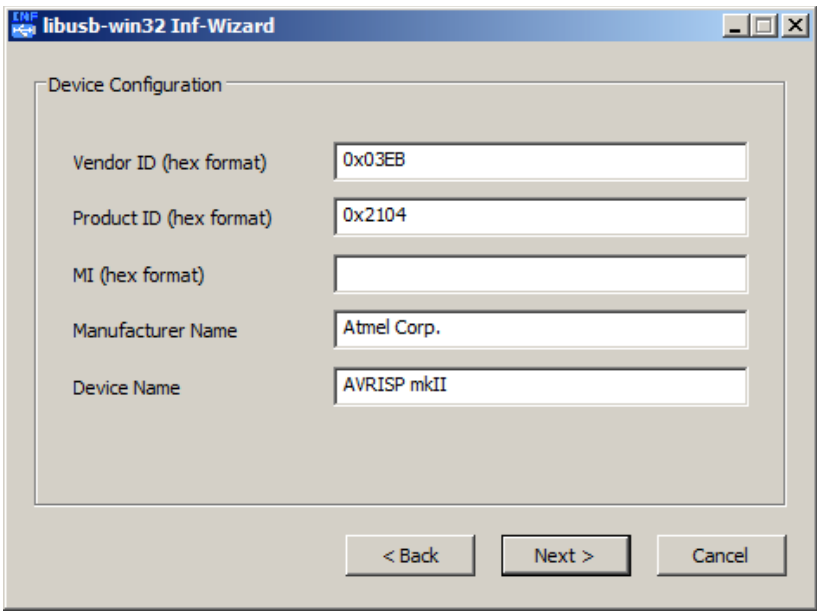
1.1. Installation von LibUSB-win23

Sollte das Programmiergerät nicht erkannt werden oder die Arduino IDE beim Brennen des Bootloaders eine Fehlermeldung ausgeben, so muss die LibUSB neu installiert werden. Dies ist meistens der Fall wenn das AVR-Studio auch auf dem PC installiert ist.

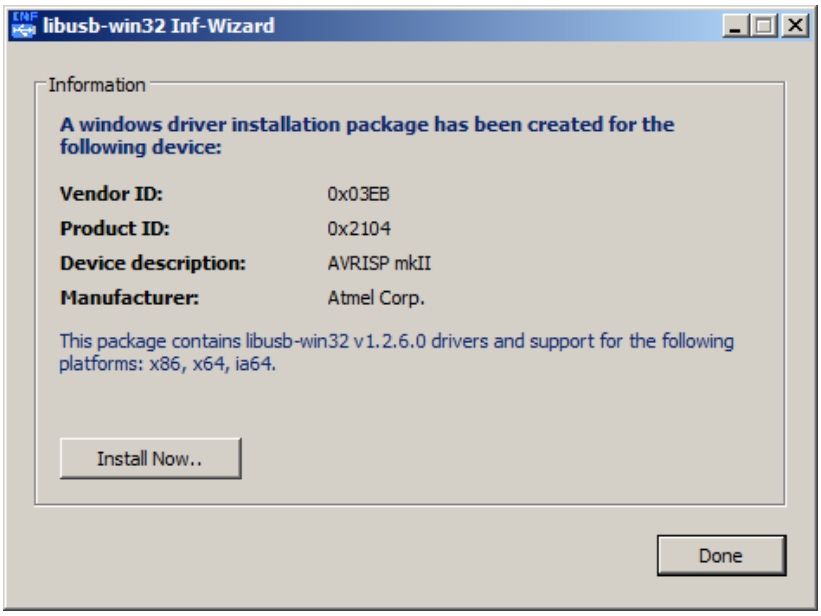
1. Das Programmiergerät anschließen.
2. Die LibUSB herunter laden und das Archive entpacken
   http://sourceforge.net/projects/libusb-win32/files/latest/download
3. Um Verzeichnis „bin“ die Datei „inf-wizard.exe“ starten.
4. Im nächsten Fenster das Programmiergerät auswählen und auf „Next“ klicken.



5. Der Inhalt des nächsten Fensters sollte schon korrekt sein. Demnach einfach mit „Next“ bestätigten.



6. Im nächsten Schritt fraget das Programm nach einem Platz auf der Festplatte wo es seine Dateien abspeichern kann. Wähle ein dir bekanntes Verzeichnis aus. Die erstellten Dateien können nämlich nach Abschluss der Installation wieder gelöscht werden.
7. Im nächsten Fenster kann man mit einem Druck auf die Schaltfläche „Install Now...“ die Installation des Treibers starten.



8. Wurde der Treiber installiert, wird mit „Done“ das Fenster geschlossen. Nun sollte das Programmiergerät korrekt erkannt werden und auch das Brennen des Bootloaders sollte nun in der Arduino IDE funktionieren.

2. Das erste Arduino-Programm (cl_ss_LED_test.ino)

Die Arduino-SDK bietet bereits viele fertige Beispielprogramme. Unter "File►Examples►01.Basics►Blink finden wir ein Programm das unsere LED zum Blinken bringt. Natürlich muss die Pinnummer, an der sich die LED befindet angepasst werden (Pin 7 statt 13). Ändere die Pinnummer (3 Zeilen!) und führe das Programm aus. Das erste Icon in der Icon-Leiste übersetzt das Programm. Das zweite Icon (Pfeil nach rechts) sendet das Programm zum Board.

Hier eine leicht veränderte Version des Programms. Die LED blinkt jetzt schneller.

```
/* cl_ss_LED_test.ino
Blinking LED (1Hz) */

// Pin 7 has an LED connected on the creative-lab soldering board
const byte led = 7;

void setup() {               // the setup routine runs once
  pinMode(LED, OUTPUT);}     // initialize the digital pin as an output.

void loop() {                // the loop routine runs over and over again forever
  digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(500);                // wait for 500ms
  digitalWrite(LED, LOW);    // turn the LED off by making the voltage LOW
  delay(500);                // wait for 500ms
}
```

Ein Arduino-Programm besteht immer aus einer Setup-Funktion in der einmalige Initialisierungen vorgenommen werden und einer Loop-Funktion, die einer Endlosschleife entspricht. Logische Blöcke werden wie in der Programmiersprache C üblich in geschweifte Klammern eingefasst. Jeder Befehl muss mit einem Semikolon (Strichpunkt) abschließen. Funktionen können Parameter erhalten, deshalb die runden Klammern (siehe **`delay`**). Soll eine Funktion selbst einen Wert mit **`return`** zurückgeben, so muss der Datentyp vor der Funktionsdeklaration angegeben werden. "**`void`**" bedeutet, dass kein Datentyp zurückgegeben wird.

Wie im folgenden Schaltplan der Schaltung (Part 1) zu erkennen ist Pin 1 (PE6) des ATmega32u4 mit der LED über einen Vorwiderstand verbunden. Beim Arduino-Micro entspricht diese Pin dem digitalen Arduino Pin 7.

3. Test des LCD-Displays (cl_ss_LCD_test.ino)

Als nächstes soll das LCD-Display getestet werden. Dazu nutzen wir die Arduino-Bibliothek "LiquidCrystal" (https://www.arduino.cc/en/Reference/LiquidCrystal).

Mit

```
#include <LiquidCrystal.h>
```

wird die Bibliothek eingebunden.

Das Objekt **`lcd`** wird dann mit

```
LiquidCrystal lcd(13, 5, 10, 9, 8, 6);
```

erzeugt, wobei die Pins der beiden Steuerleitungen (RS, E) und der 4 Datenleitungen in Arduino-Schreibweise übergeben werden. Mit der Methode **`lcd.begin()`** wird der Bibliothek noch mitgeteilt welches Display verwendet wird. Die weiteren Methoden im Beispiel sind soweit selbsterklärend.

Damit das Display den Text lesbar darstellt muss mit Hilfe des Potentiometers (links neben dem Display) der Kontrast richtig eingestellt werden!

```
/* cl_ss_LCD_test.ino
Testing the Pollin 8*2 Display on the creative-lab soldering board. */

#include <LiquidCrystal.h>

const byte led = 7;         // PE6 LED Arduino-Micro Pin 7

// initialize the library with the numbers of the LCD pins
// LiquidCrystal(rs, enable, d4, d5, d6, d7)
// PD6,PD7,PF4,PF5,PF6,PF7:  6,8,A3,A2,A1,A0
LiquidCrystal lcd(13, 5, 10, 9, 8, 6);

void setup() {
  pinMode(led, OUTPUT);     // initialize the digital pin 7 as an output
  lcd.begin(8,2);           // set up the LCD's number of columns and rows
  lcd.print("CREATIVE");    // print a message to the LCD.
  lcd.setCursor(0,1);       // cursor position to second row
  lcd.print("  LAB");
  delay(2000);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(" LOET-");
  lcd.setCursor(0,1);
  lcd.print("STATION");
}

void loop() {
  digitalWrite(led, HIGH);   // turn the LED on
  delay(500);                // wait for 500ms
  digitalWrite(led, LOW);    // turn the LED off
  delay(500);                // wait for 500ms
}
```

4. Mit Tastern die Temperatur verstellen (cl_ss_button_test.ino)

Das nächste Programm fällt etwas umfangreicher aus. Zuerst verbannen wir den Willkommen-Text in eine Funktion. Dies ermöglicht uns das Hauptprogramm später übersichtlicher zu gestalten. Nach dem Willkommen-Text wird dann noch der Text, der später nie ändert (hier "Soll:" und "Ist: " für Sollwert der Temperatur und Ist-Wert der Temperatur) initialisiert. Eine Funktion in Arduino sieht dann folgendermaßen aus:

```
// function with welcome text and initialization of the constant text
void LCD_ini() {
  lcd.begin(8,2);               // set up the LCD's number of columns and rows
  lcd.print("CREATIVE");        // print a message to the LCD.
  lcd.setCursor(0,1);           // cursor position to second row
  lcd.print("  LAB");           // ...
  delay(2000);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(" LOET-");
  lcd.setCursor(0,1);
  lcd.print("STATION");
  delay(2000);
  lcd.clear();
  lcd.print("Soll:");
  lcd.setCursor(0,1);
  lcd.print("Ist: ");
}
```

Beim Aufruf in der Setup-Funktion müssen die Klammern mit angegeben werden:

```
  LCD_ini();                    // function to initalize the display (see below)
```

Damit wir die Taster bei ihrer Funktion beobachten können soll mit Ihnen der Temperaturwert auf dem Display geändert werden. Es gibt einen Taster zum Erhöhen des Wertes (**pbUp**) und einen zum Erniedrigen des Wertes (**pbDown**). Ein Federkontakt am Lötkolben funktioniert genau wie ein Taster und soll hier gleich mit eingebunden werden. Er reduziert die Temperatur auf einen Standby-Wert um die Lötspitze zu schonen (**pinStandby**).

Als Erstes müssen die entsprechenden Pins einmalig in der Setup-Funktion als Ausgang gesetzt werden:

```
const byte pbUp       = 2;      // PD1 UP   Arduino-Micro Pin 2
const byte pbDown     = 12;     // PD6 DOWN Arduino-Micro Pin 12
const byte pinStandby = A0;     // PF7 STANDBY Arduino-Micro Pin A0
...
void setup() {
  pinMode(pbUp,       INPUT_PULLUP);  // 3*PB with internal pullup
  pinMode(pbDown,     INPUT_PULLUP);
  pinMode(pinStandby, INPUT_PULLUP);
  ...
```

Damit unser Programm funktioniert brauchen wir einige Variablen und Konstanten:

```
int targetTemp;
int actualTemp;
int targetTempPb;
const int targetTempIni = 350;
const int standbyTemp   = 50;
const int maxTemp       = 420;
const int mainLoopDelay = 10;   // respond time of pushbuttons and heating!
```

Die Konstante **mainLoopDelay** ist ein wichtiger Wert, da sie entscheidet wie schnell Änderungen an den Tastern später umgesetzt werden.

Neben der Variablen für den Sollwert brauchen wir eine Hilfsvariable (**targetTempPb**; Pb steht nicht für Blei :) für den mit den Tasten eingestellten Sollwert sondern für pushbutton). Dies ist nötig, da der richtige Sollwert auch auf die Standby-Temperatur gesetzt werden kann.

Nach der Initialisierung des Sollwertes und des Ist-Wertes programmieren wir in einer neuen Funktion mit dem Namen **pushbutton_control()** drei **If**- bzw. **If-Else**-Abfragen. Wurde **pbUp** gedrückt (negative Logik da Pull-Up-Widerstand!) so wird der Taster-Sollwert (**targetTempPb**) um Eins erhöht. Beim Erreichen des Maximalwertes wird nicht weiter erhöht.

Ähnliches passiert beim Drücken von **pbDown**, nur dass hier der Wert erniedrigt wird und nicht unter 25 Grad Celsius fallen kann.

Liegt der Lötkolben auf dem Federkontakt auf, so wird er auf Standby-Temperatur gesetzt. Falls nicht wird der Zielwert auf den mit den Tastern eingestellten Wert zurückgesetzt.

```
void setup() {
  ...
  LCD_ini();                    // function to initalize the display (see below)
  targetTemp = targetTempIni;
  targetTempPb = targetTempIni;
}

void loop() {
  actualTemp = 25;
  LCD_update();                 // function to update the display (see below)
```

```
  pushbutton_control();         // funct. to increment and decr. the temperature
  delay(mainLoopDelay);
}

// function to increment and decrement the temperature
void pushbutton_control() {
  if (digitalRead(pbUp)==LOW) { // change temp if one of the pushbuttons
    targetTempPb++;             // is pressed and control max and min values
    if (targetTempPb > maxTemp) targetTempPb = maxTemp;
  }

  if (digitalRead(pbDown) == LOW) {
    targetTempPb--;
    if (targetTempPb < 25) targetTempPb = 25;
  }
  if (digitalRead(pinStandby) == LOW) targetTemp = standbyTemp;
  else targetTemp = targetTempPb;
}
```

Wie ersichtlich wird noch eine Funktion zum Erneuern der Display-Anzeige aufgerufen. Hier werden die Variablen **`targetTemp`** und **`actualTemp`** (Sollwert und Ist-Wert) neu auf das Display gebracht. Da die Werte zwei oder dreistellig sein können, muss das Display entsprechend angepasst werden. Hier das gesamte Programm:

```
/* cl_ss_button_test.ino
Testing the buttons on the creative-lab soldering board */

#include <LiquidCrystal.h>

const byte led        = 7;     // PE6 LED  Arduino-Micro Pin 7
const byte pbUp       = 2;     // PD1 UP   Arduino-Micro Pin 2
const byte pbDown     = 12;    // PD6 DOWN Arduino-Micro Pin 12
const byte pinStandby = A0;    // PF7 STANDBY Arduino-Micro Pin A0

// initialize the library with the numbers of the LCD pins
// LiquidCrystal(rs, enable, d4, d5, d6, d7)
// PC7, PC6, PB6, PB5, PB4, PD7: 13, 5, 10, 9, 8, 6
LiquidCrystal lcd(13, 5, 10, 9, 8, 6);

int targetTemp;
int actualTemp;
int targetTempPb;
const int targetTempIni = 350;
const int standbyTemp   = 50;
const int maxTemp       = 420;
const int mainLoopDelay = 10;   // respond time of pushbuttons and heating!

void setup() {
  pinMode(led,        OUTPUT);        // initialize the digital pin 7 as an output
  pinMode(pbUp,       INPUT_PULLUP);  // 3*PB with internal pullup
  pinMode(pbDown,     INPUT_PULLUP);
  pinMode(pinStandby, INPUT_PULLUP);
  LCD_ini();                    // function to initalize the display (see below)
  targetTemp = targetTempIni;
  targetTempPb = targetTempIni;
}

void loop() {
  actualTemp = 25;
  LCD_update();                 // function to update the display (see below)
  pushbutton_control();         // funct. to increment and decr. the temperature
  delay(mainLoopDelay);
}

// function to increment and decrement the temperature
void pushbutton_control() {
```

```
  if (digitalRead(pbUp)==LOW) { // change temp if one of the pushbuttons
    targetTempPb++;             // is pressed and control max and min values
    if (targetTempPb > maxTemp) targetTempPb = maxTemp;
  }
  if (digitalRead(pbDown) == LOW) {
    targetTempPb--;
    if (targetTempPb < 25) targetTempPb = 25;
  }
  if (digitalRead(pinStandby) == LOW) targetTemp = standbyTemp;
  else targetTemp = targetTempPb;
}
// function to update the changing values on the display
void LCD_update() {
  lcd.setCursor(5,0);
  lcd.print("000");
  if (targetTemp <100) lcd.setCursor(6,0);
  else lcd.setCursor(5,0);
  lcd.print(targetTemp);
  lcd.setCursor(5,1);
  lcd.print("000");
  if (actualTemp <100) lcd.setCursor(6,1);
  else lcd.setCursor(5,1);
  lcd.print(actualTemp);
}

// function with welcome text and initialization of the constant text
void LCD_ini() {
  lcd.begin(8,2);               // set up the LCD's number of columns and rows
  lcd.print("CREATIVE");        // print a message to the LCD.
  lcd.setCursor(0,1);           // cursor position to second row
  lcd.print("  LAB");           // ...
  delay(2000);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(" LOET-");
  lcd.setCursor(0,1);
  lcd.print("STATION");
  delay(2000);
  lcd.clear();
  lcd.print("Soll:");
  lcd.setCursor(0,1);
  lcd.print("Ist: ");
}
```

5. Die Temperatur mit dem ADC ermitteln (cl_ss_ADC_test.ino)

Um die Temperatur zu messen wird eigentlich nur die Funktion **`analogRead()`** benötigt, die eine AD-Wandlung mit der am entsprechenden Pin (hier A5, PF0) anliegenden Spannung durchführt. Als Referenz dient die Versorgungsspannung.

Laut Martin Krumm (Funkamateur 7/14 S732) beträgt der Temperaturkoeffizient des Thermoelements rund 16µV/°C. Mit einer Verstärkung des OPV von 680 (68k/100Ω) ergibt sich ein Multiplikationswert **`adcGain`** von $5V/1023/680/16*10^{-6} = 0{,}45$

Das Programm kann um folgende Zeilen erweitert werden. Fasst man die Lötspitze an, so müsste sich die Temperatur um einige Grad erhöhen.

```
const byte pinTemp = A5;        // PF0 ADC Arduino-Micro Pin A5
...
const float adcGain = 0.45;     // R1=100 R2=68K adcGain = 5V/1023/680/16*10E-6
const float adcOffset = 25;
```

```
void loop() {
  actualTemp = getTemp();
  ...

int getTemp() {
  unsigned int adcValue = analogRead(pinTemp);        // read the input on analog pin
  return round(((float) adcValue)*adcGain+adcOffset); // calculate temperature
}
```

6. Die Lötspitze mit der PWM heizen (cl_ss_firmware.ino)

ACHTUNG: **Beim Programmieren dieser Aufgabe muss die Lötspitze abgezogen werden!!**

Ist nur der USB-Stecker angeschlossen bezieht die Spitze die Energie zum Aufheizen sonst über den USB-Port (nicht so gut). Die Spitze darf nur dran bleiben, falls die 12V auch schon angeschlossen wurden.

Es fehlt nur noch ein kleiner Teil um die Software für unsere Lötstation fertigzustellen. Mit einer Pulsweitenmodulation wird die Lötspitze erhitzt. Das erledigt praktischerweise die Funktion **`analogWrite()`**. Die Regelung besteht aus der Berechnung der Differenz aus Sollwert und Ist-Wert und einer zusätzlichen Multiplikation mit einem Verstärkungsfaktor.

Wir benötigen eine Variable für den PWM-Wert. Damit die Lötstation nicht gleich aufheizt ist dieser auf Null zu setzen. Der PWM-Ausgang muss natürlich initialisiert werden und eine zusätzliche Variable hilft die Regelung zu beschleunigen.

```
const byte pwmOut = 3;          // PD0 pwmOut Arduino-Micro Pin 3
...
int pwm;
...
const int   ctrlGain = 10;      // for pwm to heat faster

void setup() {
  ...
  pinMode(pwmOut, OUTPUT);      // PWM output
  pwm=0;                        // dont heat up immediatly
}
```

In der Hauptschleife wird die Funktion, welche die Regelung übernimmt aufgerufen. In der Funktion **`automatic_control()`** werden Sollwert und Ist-Wert werden verglichen und mit dem Verstärkungsfaktor multipliziert. Die PWM-Variable wird auf 255 reduziert, da die PWM mit 8 Bit arbeitet. Ist die aktuelle Temperatur höher als die Soll-Temperatur wird die PWM abgeschaltet. Zusätzlich zeigt unsere LED jetzt das Aufheizen an.

```
void loop() {
  actualTemp = getTemp();
  automatic_control();
  LCD_update();                 // function to update the display (see below)
  pushbutton_control();         // funct. to increment and decr. the temperature
  delay(mainLoopDelay);
}

// function for automatic control (P controller)
void automatic_control() {
  int diff = targetTemp-actualTemp;     // get difference
  pwm = diff*ctrlGain;                  // heat up faster
  if(pwm > 255) pwm = 255;              //limit pwm value to 0...255
  if (actualTemp > targetTemp) pwm = 0;
  analogWrite(pwmOut, pwm);
```

```
  if(pwm > 0) digitalWrite(led, HIGH);  //set heat LED
  else digitalWrite(led, LOW);
}
```

Eine kleine Änderung ist noch in der Funktion zum Einlesen der Temperatur nötig. Da die 12V PWM den Temperaturfühler beeinflusst, muss diese während des Messens abgeschaltet werden. Vor der Messung muss der vorgeschaltete Tiefpass genügend Zeit zum Einschwingen haben.

```
  ...

int getTemp() {
  analogWrite(pwmOut, 0);                                 // switch off heater
  delay(adcDelay);                                        // wait for low pass filter (steady state)
  unsigned int adcValue = analogRead(pinTemp);            // read the input on analog pin
  analogWrite(pwmOut, pwm);
  return round(((float) adcValue)*adcGain+adcOffset); // calculate temperature
}
```

7. Vollständige Firmware für die Lötstation:

```
/* cl_ss_firmware.ino
firmware for the creative-lab soldering board */

#include <LiquidCrystal.h>

const byte led        = 7;     // PE6 LED  Arduino-Micro Pin 7
const byte pbUp       = 2;     // PD1 UP   Arduino-Micro Pin 2
const byte pbDown     = 12;    // PD6 DOWN Arduino-Micro Pin 12
const byte pinStandby = A0;    // PF7 STANDBY Arduino-Micro Pin A0
const byte pinTemp    = A5;    // PF0 ADC Arduino-Micro Pin A5
const byte pwmOut     = 3;     // PD0 pwmOut Arduino-Micro Pin 3

// initialize the library with the numbers of the LCD pins
// LiquidCrystal(rs, enable, d4, d5, d6, d7)
// PC7, PC6, PB6, PB5, PB4, PD7: 13, 5, 10, 9, 8, 6
LiquidCrystal lcd(13, 5, 10, 9, 8, 6);

int targetTemp;
int actualTemp;
int targetTempPb;
int pwm;

const int   targetTempIni = 350;
const int   standbyTemp   = 50;
const int   maxTemp       = 420;
const int   mainLoopDelay = 5;      // respond time of pushbuttons and heating!
const int   adcDelay      = 10;     // delay before measure in ms (low pass filter)
const float adcGain       = 0.45;   // R1=100 R2=68K adcGain = 5V/1023/680/16*10E-6
const float adcOffset     = 25;
const int   ctrlGain      = 10;     // for pwm to heat faster

void setup() {
  pinMode(led,        OUTPUT);        // initialize the pin for the led as an output
  pinMode(pbUp,       INPUT_PULLUP);  // 3*PB with internal pullup
  pinMode(pbDown,     INPUT_PULLUP);
  pinMode(pinStandby, INPUT_PULLUP);
  pinMode(pwmOut,     OUTPUT);  // PWM output
  LCD_ini();                    // function to initalize the display (see below)
  targetTemp   = targetTempIni;
  targetTempPb = targetTempIni;
  pwm=0;                        // dont heat up immediatly
}

void loop() {
```

```
  actualTemp = getTemp();
  automatic_control();
  LCD_update();                 // function to update the display (see below)
  pushbutton_control();         // funct. to increment and decr. the temperature
  delay(mainLoopDelay);
}

// function for automatic control (P controller)
void automatic_control() {
  int diff = targetTemp-actualTemp;      // get difference
  pwm = diff*ctrlGain;                   // heat up faster
  if(pwm > 255) pwm = 255;               //limit pwm value to 0...255
  if (actualTemp > targetTemp) pwm = 0;
  analogWrite(pwmOut, pwm);
  if(pwm > 0) digitalWrite(led, HIGH);   //set heat LED
  else digitalWrite(led, LOW);
}

// function read the temperature (ADC)
int getTemp() {
  analogWrite(pwmOut, 0);                              // switch off heater
  delay(adcDelay);                                     // wait for low pass filter (steady state)
  unsigned int adcValue = analogRead(pinTemp);         // read the input on analog pin
  analogWrite(pwmOut, pwm);
  return round(((float) adcValue)*adcGain+adcOffset); // calculate temperature
}

// function to increment and decrement the temperature
void pushbutton_control() {
  if (digitalRead(pbUp)==LOW) { // change temp if one of the pushbuttons
    targetTempPb++;             // is pressed and control max and min values
    if (targetTempPb > maxTemp) targetTempPb = maxTemp;
  }
  if (digitalRead(pbDown) == LOW) {
    targetTempPb--;
    if (targetTempPb < 25) targetTempPb = 25;
  }
  if (digitalRead(pinStandby) == LOW) targetTemp = standbyTemp;
  else targetTemp = targetTempPb;
}

// function to update the changing values on the display
void LCD_update() {
  lcd.setCursor(5,0);
  lcd.print("000");
  if (targetTemp <100) lcd.setCursor(6,0);
  else lcd.setCursor(5,0);
  lcd.print(targetTemp);
  lcd.setCursor(5,1);
  lcd.print("000");
  if (actualTemp <100) lcd.setCursor(6,1);
  else lcd.setCursor(5,1);
  lcd.print(actualTemp);
}

// function with welcome text and initialization of the constant text
void LCD_ini() {
  lcd.begin(8,2);               // set up the LCD's number of columns and rows
  lcd.print("CREATIVE");        // print a message to the LCD.
  lcd.setCursor(0,1);           // cursor position to second row
  lcd.print("  LAB");           // ...
  delay(2000);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(" LOET-");
  lcd.setCursor(0,1);
  lcd.print("STATION");
```

```
  delay(2000);
  lcd.clear();
  lcd.print("Soll:");
  lcd.setCursor(0,1);
  lcd.print("Ist: ");
}
```

PART 3: 3D-Druck

1. Software zum Zeichnen des Objektes

Um das 3D Stück zu zeichnen stehen viele unterschiedliche Programme zur Verfügung. Wir bevorzugen freie Software wie Tinkercad, FreeCad oder Blender. FreeCad oder Blender benötigen eine recht lange Einarbeitungszeit, deshalb wird hier **Tinkercad** verwendet.

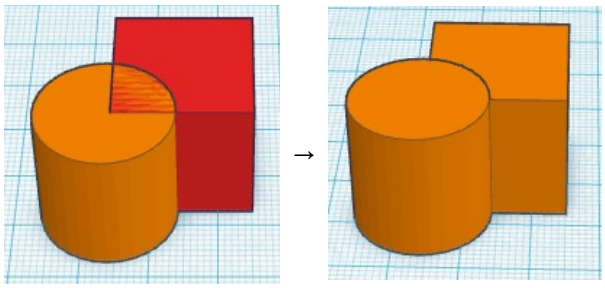
Alle Programme liefern als Ausgabe eine STL Datei.

2. Zeichnen mit Tinkercad

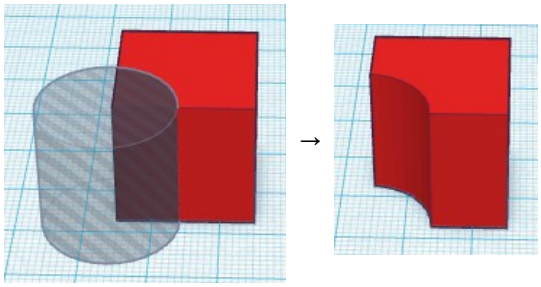
Tinkercad ist eine Internet Anwendung und läuft in jedem modernen Browser der WebGL unterstützt. Diese Anwendung ist recht einfach zu bedienen weil sie einerseits eine rech benutzerfreundliche Oberfläche hat und andererseits auf dem Prinzip des Zusammenführen von Formen basiert, mit welchem man quasi alle anderen Formen selbst „bauen“ kann.

2.1. Elemente gruppieren / verschmelzen

Hat man z.B. zwei feste Formen, so kann man diese mit der „Group“ Funktion (am Rand rechts) zu einem einzigen neuen Objekt zusammenfügen:

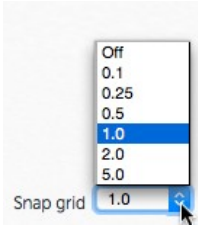


Die Farben diesen nur dazu, damit man die einzelnen Objekte besser von einander unterscheiden kann. Definiert man nun eines der Objekte als „Loch“, so kann mittels der Funktion „Group“ ein Stück aus einem anderen Objekt herausschneiden:



Natürlich lassen sich alle Verschmelzungen auch wieder mittels „Ungroup“ aufheben.

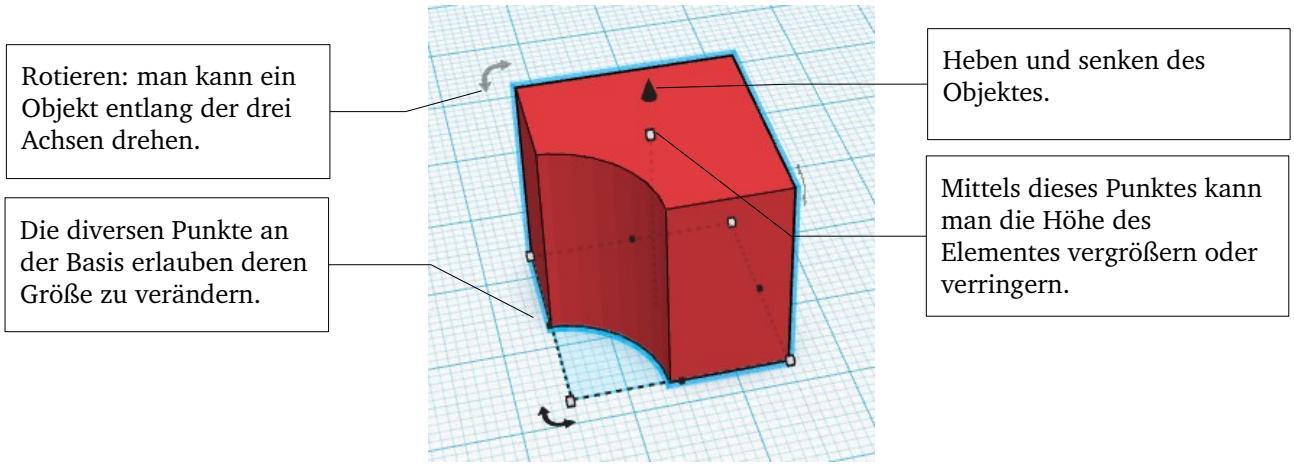
2.2. Raster einstellen



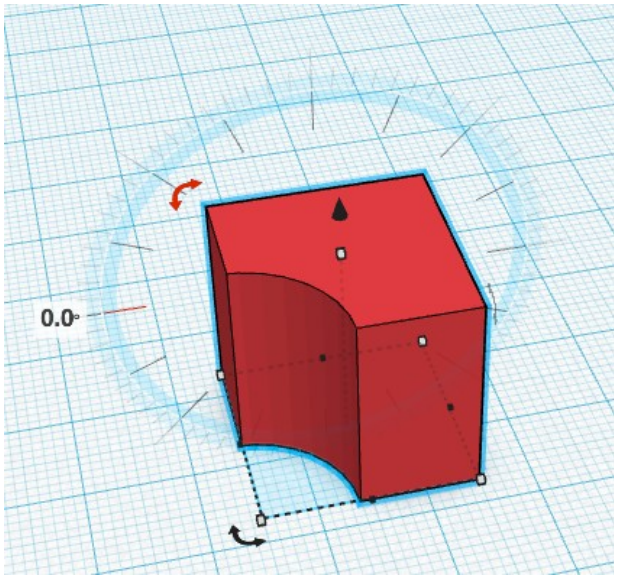
Wichtig zu wissen ist noch, dass man am unteren Bildschirmrand das „Snapping“ einstellen kann. Dies dient dazu damit die Bauteile sich automatisch dem Raster anpassen und hilft somit nicht nur bei der genauen Platzierung der Elemente sondern auch bei Einstellen deren Größe.

2.3. Elemente rotieren

Wählt man ein Element an, so kann man dessen Größe und Position verändern:

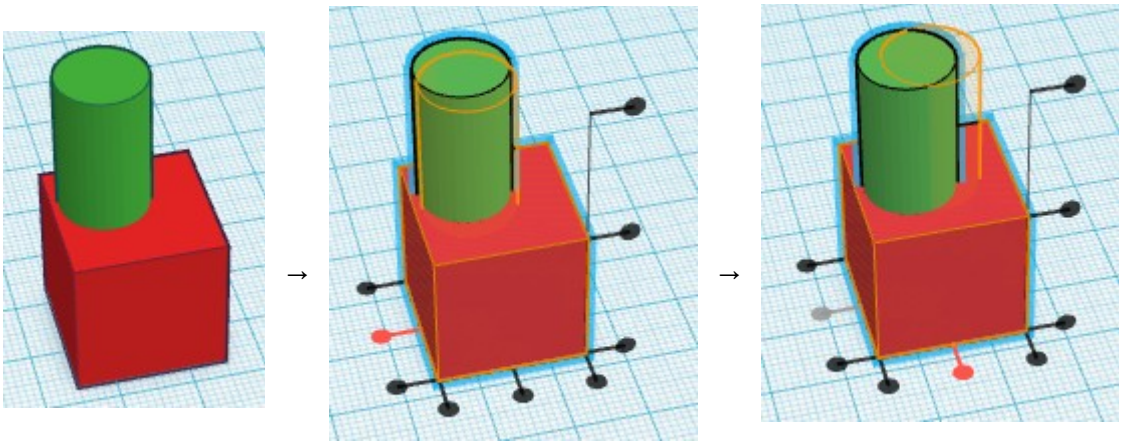


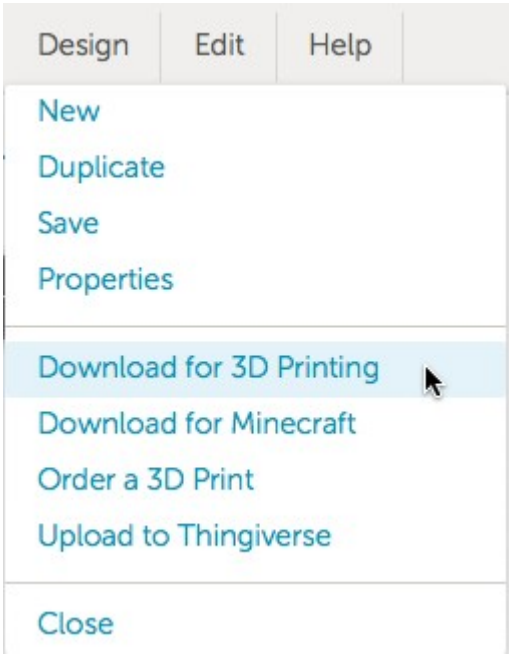
Bewegt man die Maus beim Rotieren eines Elementes im Innern des angezeigten Kreises, so wird das Objekt immer in Schritten von 22,5° gedreht. Befindet sich die Maus außerhalb des Kreises, so kann man Grad-genau einstellen wie weit man es drehen will.



2.4. Ausrichten von Objekten

Tinkercad erlaubt auch das Ausrichten von Objekten. Stellen wir uns beispielsweise vor, wir würden gerne einen Zylinder genau in die Mitte auf einen Würfel setzen. Um dies zu erreichen, wählt man beide Elemente aus und klickt dann im Menu oben rechts auf „Adjust“ > „Align“:





Fährt man dann mit der Maus über die einzelnen Ausrichtungspunkte, so sieht man in orange angegebener Farbe, die neuen Positionen der Elemente. Macht man dann einen Klick, so werden die Elemente an die neuen Positionen verschoben.

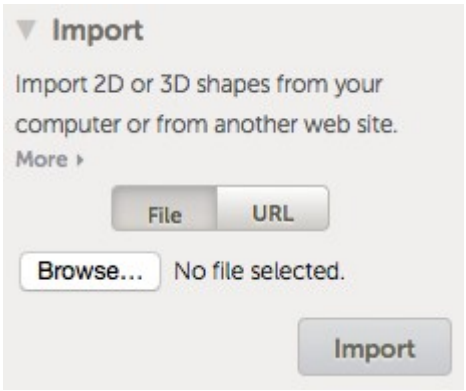
2.5. Zum 3D Druck herunterladen

Tinkercad speichert jeden Schritt automatisch ab, so dass man sich darum nicht selbst kümmern muss. Um schlussendlich das fertige 3D Modell als STL Datei herunter zu laden, klick man im Menu „Design“ auf „Download for 3D Printing“ und in dem sich dann öffnenden Fenster auf STL.

Da hat man dann auch die Möglichkeit nur ausgewählte Elemente als STL zu importieren. Dies macht Sinn, wenn man ein größeres Teil gebaut hat und erst nur einen Teil davon ausdrucken möchte.

2.6. Import von anderen STL Dateien

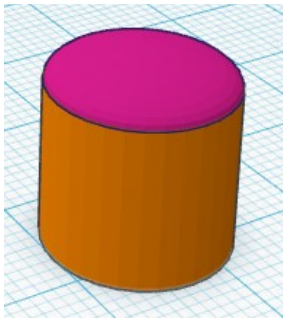
Es bleibt noch zu sagen, dass man auch STL Dateien importieren kann. Lädt man aber eine STL Datei herunter und importiert sie später wieder, so kann man die Verschmelzungen (Funktion „Group“) aber nicht mehr aufheben, das diese beim Export verloren gehen.



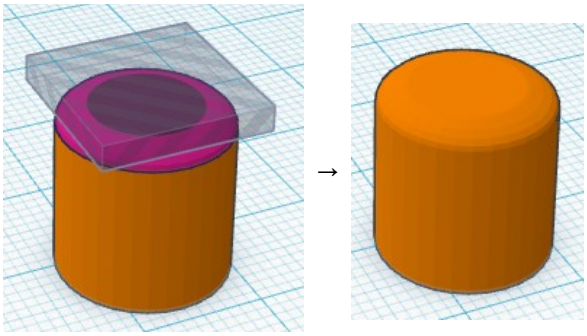
3. Erstellen eines Knopfes

Die Knöpfe der Lötstation mit Hilfe deren man die Temperatur einstellen kann, sind beide gleich, nur dass der eine um 90° gedreht ist. In diesem Kapitel befindet sich eine Schritt für Schritt Anleitung zum Erstellen des 3D Modells eines Knopfes.

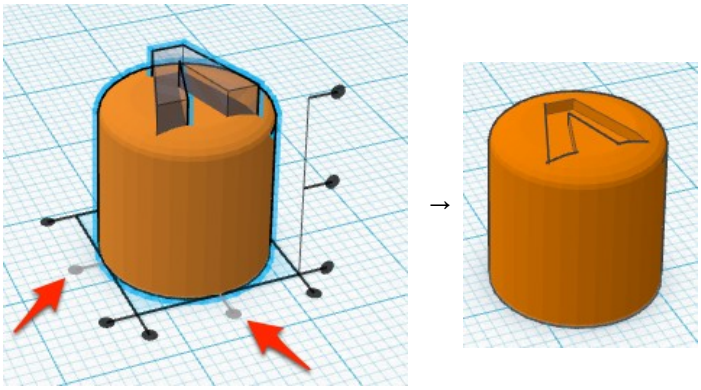
Die Basis eines solchen Knopfes bildet ein 13mm hoher Zylinder auf dem oben drauf eine 2mm hohe Halbkugel flach aufgedrückt liegt. Der Diameter des beiden Objekte beträgt 15mm.



Damit der Knopf oben drauf aber nicht rund ist, wird ein Rechteck, welches als „Loch“ markiert wurde, oben drauf gelegt. Nach dem Verschmelzen der 3 Objekte erhalten wir einen oben abgeflachten Zylinder von 14mm Höhe.



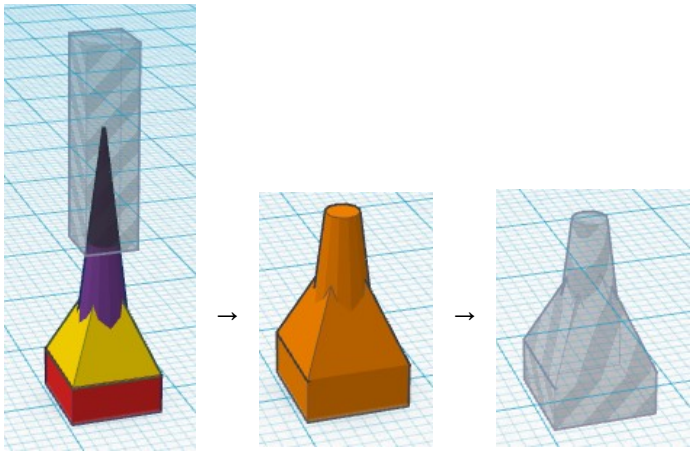
Als nächstes wird ein Pfeil-Symbol oben aus den Knopf herausgeschnitten. Hierzu verwendet man am besten den Buchstaben „V“, welcher auf einer Höhe von 13mm liegt und somit einen Schnitt oben auf dem Knopf verursacht. Um sicher zu stellen, dass beide Objekt schön zentriert sind, kann man die „Align“ Funktion (oben rechts als Punkt im Menü „Adjust“ zu finden) benutzen. Vorher muss man beide Objekte mir der Maus auswählen. Danach werden auch diese Objekte mir einander verschmolzen.



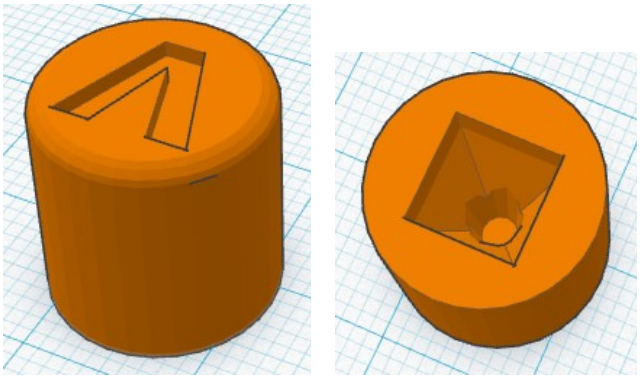
Der Knopf soll ja später genau auf das entsprechende Teil auf der Platine passen. Aus diesem Grund muss unten ein Teil herausgeschnitten werden. Das Modell des herauszuschneidenden Objektes sieht so aus:

- Ganz unten liegt ein Prisma mit einer Seitenlänge von 7mm und einer Höhe von 3mm.
- Darauf befindet sich eine 8mm hohe Pyramide mit gleicher Grundfläche
- Als nächstes wird ein Kegel mit 16mm Höhe und 4mm Basis-Durchmesser erzeugt. Dieser schwebt auf einer Höhe von 6mm über der Grundfläche und sitzt somit in der Pyramide drin.
- Die Gruppierung wird nun auf einer Höhe von 12mm oben abgeschnitten.

Alle Elemente werden mir Hilfe der „Align“ Funktion zentriert, damit sie mittig übereinander liegen. Danach werden sie gruppiert und schlussendlich als „Loch“ markiert.



Somit entsteht ein Knopf welcher perfekt auf dem sich auf der Platine befindenden Druckknopf aufsitzt. Man kann leicht drauf drücken ohne aber dass er sich drehen lässt.

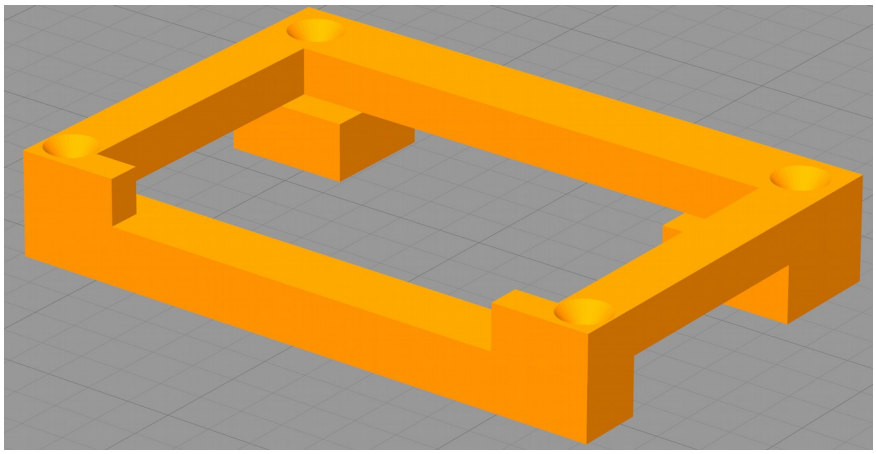


Ansicht oben Ansicht unten

Das Modell kann so wie es da ist ohne Stützen ausgedruckt werden. Die Druckdauer bei einer Schichthöhe von 0,25mm beträgt in etwa 15 Minuten.

4. Zeichnen des LCD-Halter

Das LCD-Display muss irgendwie an der Platine befestigt werden. Dazu soll ein LCD-Halter konstruiert werden.



Mit Hilfe einer Schieblehre werden am ausgedruckten Modell, am Display und an den Befestigungslöchern der Platine die Maße abgenommen (Außenmaße: 55x36,5x9).

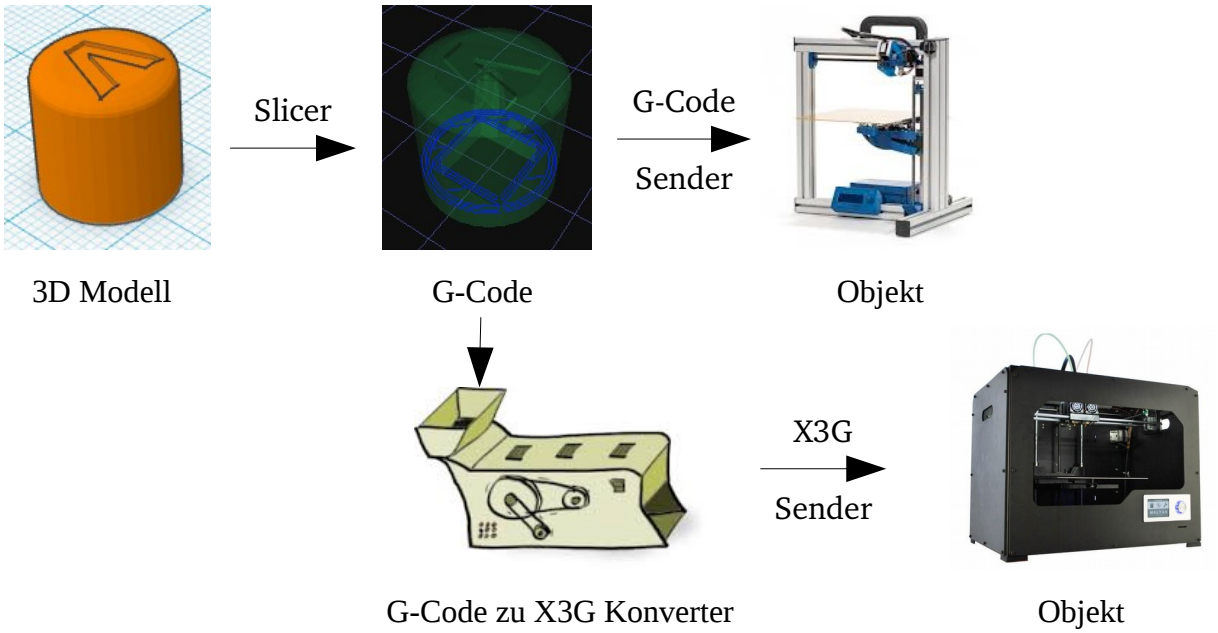
Mit Hilfe von Tinkercad wird der Halter nach gezeichnet. Es werden zum Befestigen M2 (2mm) Senkkopfschrauben verwendet. Die Schrauben sollen sauber versenkt werden.

Schieblehre / Messschieber

5. Vom Modell zum Druck

Nun liegt das 3D Modell im STL Format vor. Dieses muss nun in G-Code umgewandelt werden. Diese Format enthält die Steuerbefehle für die Motoren des 3D Druckers. Da das Modell bei diesem Prozess in mehrere Schichten geschnitten wird, spricht man hier auch vom „Schichtenmodell“. Um dieses zu erzeugen benötigt man ein sogenanntes „Slicer Programm“. Danach wird dann der erzeugte G-Code an den Drucker übertragen.

Es gibt grundsätzlich aber zwei Arten von 3D-Drucker: jene die direkt G-Code verarbeiten können und jene die X3G-Code verstehen. Für die letzteren braucht man einen entsprechenden Konverter.. Man kann sich den Druck-Progress also wie folgt vorstellen:



Es gibt diverse „stand alone“ Slicer-Programme, deren einzige Aufgabe das Erstellen von G-Code an Hand eines 3D-Modells ist. Bekannte Namen sind „Cure“, „Slic3r“ und „KISSlicer“, welches hier vorgestellt wird. Der so erzeugte G-Code muss dann in einer nächsten Etappe mittels eines anderen Programms oder einer SD-Karte an den Drucker übertragen werden.

Es gibt allerdings auch „all-in-one“ Lösungen wie z.B. „Repetier Host“ oder „Simplify 3D“. Diese Programme erlauben es den gesamten Prozess vom 3D-Modell bis hin zum fertig gedruckten Objekt zu absolvieren.

6. KISSlicer

Bei jedem Slicer-Programm sollten folgende Einstellungen unbedingt überprüft werden:

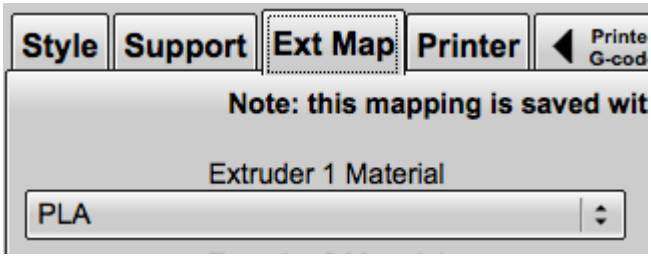
6.1. Material

Je nach verwendetem Material muss der Druckkopf eine andere Temperatur haben. Wir verwenden meistens PLA[3], welches bei 185-200 °C gedruckt wird. PETG hingeben braucht eher 220-240°C. Je nach Qualität des gedruckten Objektes, muss man die Temperatur eventuell noch anpassen.

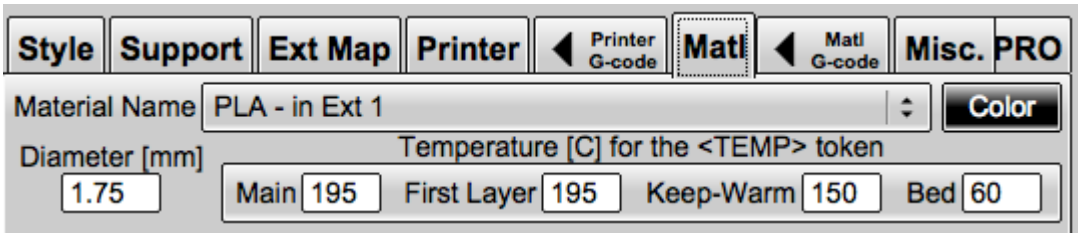
Details kann man hier nachlesen:

https://www.simplify3d.com/support/print-quality-troubleshooting/

In KISSlicer findet man diese Einstellung im Tab „Ext Map“:



Welche Temperaturen für das angegebene Material voreingestellt sind, kann man im Tab „Matl“ einsehen und diese auch da gegebenenfalls anpassen:



6.2. Layer thickness (Druckhöhe)

Die Druckhöhe ist der primäre Faktor welcher sowohl die Druckzeit als auch die Druckqualität beeinflusst. Die meisten Drucker erlauben einen Druck mit einer Schichthöhe von 50 bis 250 Mikrometer. Diese Einstellung bestimmt demnach in wie viele Schichten ein 3D Modell eingeteilt wird.

Nehmen wir das Beispiel eines Würfels mit einer Kantenlänge von 5mm. Mit einer Schichthöhe von 250µm, also 5 Schichten pro Millimeter, werden insgesamt 20 Schichten gefahren. Stellt man aber eine Druckhöhe von 50µm, so müssen 5 mal mehr Schichten Druckkopf abgefahren werden, nämlich 100. Die Druckzeit wird sich demnach auch verfünffachen.

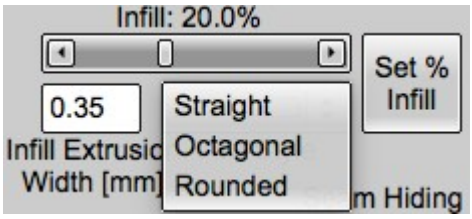
Für die meisten Sache ist die Einstellung „Draft“ (250µm) oder „Normal“ (200µm) durchaus OK.

3 https://de.wikipedia.org/wiki/Polylactide

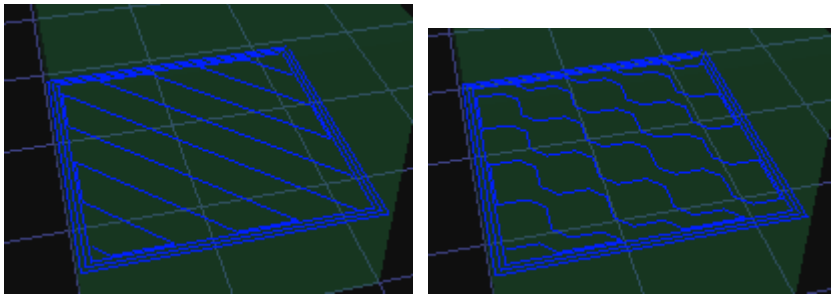
6.3. Infill (Füllmenge)

Die Füllmenge gibt an zu wie viel Prozent ein Hohlraum mit Plastik gefüllt werden soll. Diese Einstellung beeinträchtigt natürlich die Stabilität so wie die Dehnbarkeit eines Druckes.

In KISSlicer findet man diese Einstellungen im Tab „Style“:



Allerdings spielt auch die Form der Füllung eine gewisse Rolle. KISSlicer bietet hier drei Möglichkeiten an, wobei die zwei letzten sich dann doch sehr viel ähneln.



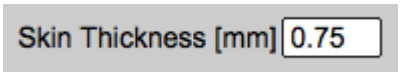
Straight Octogonal / Rounded

Normalerweise ist die Voreinstellung „Straight“ gut.

6.4. Skin thickness (Boden- und Dachhöhe)

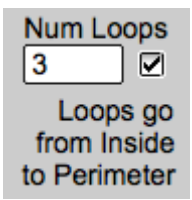
Die Boden- und Dachhöhe, in KISSlicer in Millimeter angegeben, besagt wie viele Schichten an Vollplastik oben und unter gedruckt werden. Zwischen diesen Schichten die aus Vollplastik bestehen, werden dann Schichten gedruckt welche, je nach eingestellter Füllmenge, Hohlräume enthalten.

Diese Einstellung beeinflusst demnach die Stabilität eines gedruckten Objektes.



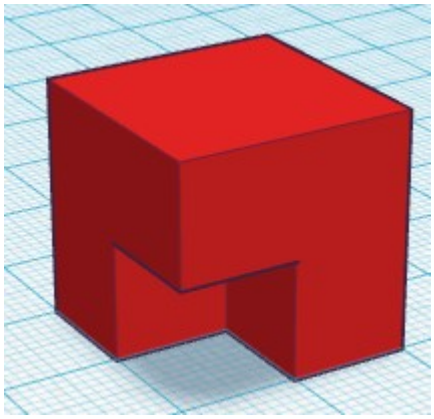
6.5. Loops (Wandstärke)

Genau wie die Boden- und Dachhöhe, beeinflusst auch die Wandstärke die Stabilität eines Objektes. Dies kann in KISSlicer auch auf dem Tab „Style“ eingestellt werden. Hier gibt man allerdings keine Längeneinheit ein, sondern gibt an wie oft der Druckkopf außer oder innen herumfahren soll. Man kann die maximale Breite aber berechnen indem man die „number of loops“ mit der daneben abgegebenen „extrusion width“ multipliziert. Letztere ist eine physikalische Angabe des Druckkopfes.

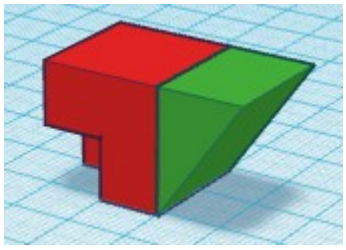


6.6. Support (Stützmaterial)

Druck man etwas mit einem Überhang, wie z.B. bei diesem Würfel bei dem unten ein Teil herausgeschnitten wurde, muss der Hohlraum mit einer Stütze versehen werden, da das obere Material sonst beim Druck einfach herunterfallen würde.

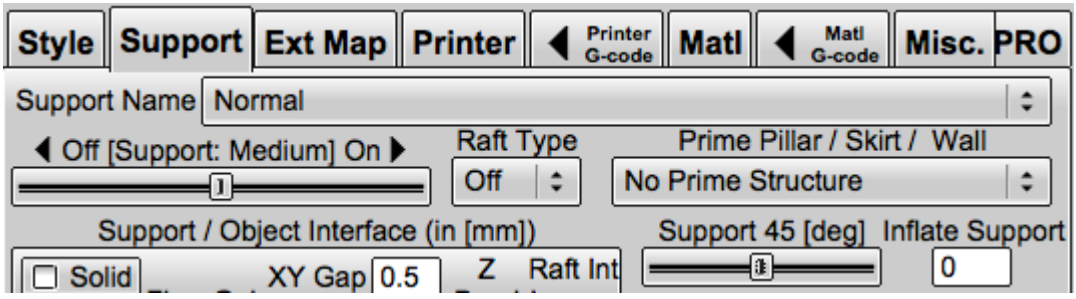


Für negative Winkel, wie zum Beispiel bei folgender Figur auf der rechten (grünen) Seite, braucht man kein Stützmaterial so lange der Winkel welcher die Wand mit dem Boden formt nicht kleiner als 45° ist.



Hohlraum welche komplett von Wänden umgeben sind, können normalerweise bis zu einer Breite von 1cm ohne Stützmaterial überbrückt werden. Dies hängt aber immer auch ein bisschen vom verwendeten Material, von der Temperatur der Düse und vor allem vom Drucker so wie von dessen Geschwindigkeit ab.

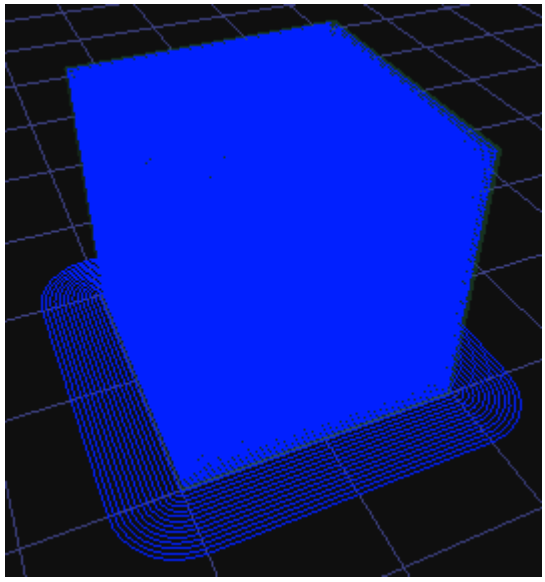
In KISSlicer findet man diese Einstellung im „Support“ Tab.



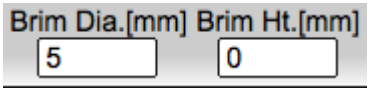
Hier gibt es auch die Möglichkeit die Stützen aus zu dehnen → Inflate Support. Dies bewirkt, dass das Stützmaterial aufgeblasen wird und etwas breiter als die zu stützende Fläche gedruckt wird.

6.7. Brim (Rand)

Grundsätzlich ist es ja so, dass die gedruckten Objekte auf dem Druckbett kleben müsse damit der Drucker die nächsten Schichten auftragen kann. Objekte mit einer sehr kleinen Auflagefläche können sich aber leicht lösen. Um diese trotzdem drucken zu können, kann man einen Rand einstellen. Hierbei werden die untersten Schichten (eine genügt meistens) ausgeweitet. Diese können nach Beenden de Druckes einfach abgeschnitten oder abgebrochen werden.



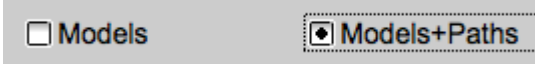
In KISSlicer findet sich diese Option auch auf dem Tab „Support“ wieder, und zwar unten rechts. Gibt man keine Höhe ein, so wird nur eine Schicht gedruckt, was meistens ausreichend ist. Gibt man eine Höhe ein, so hängt die Anzahl der Schichten von der eingestellten Schichthöhe ab.



6.8. Kontrolle!

Bevor man sein Objekt druckt, soll man es auf jeden Fall Schicht für Schicht kontrollieren. Dies spart Zeit und Rohmaterial, da man hier oft Fehler entdeckt und diese dann noch vor dem Druck beheben kann.

Hat man alle Einstellungen getätigt, klick man oben rechts auf die Schaltfläche „Slice“. Jetzt berechnet der Computer aus dem 3D Modell ein Schichtenmodell. Ist er damit fertig, kann man oben links einstellen, welche Ansicht man gerne hätte:



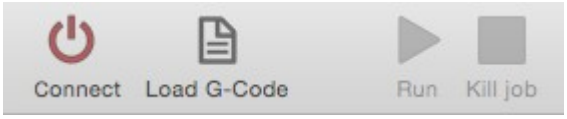
Die Einstellung „Models + Path“ erlaubt es nun mittels des vertikalen Sliders, welcher sich am rechten Rand des Fensters mit der 3D Ansicht befindet, durch die einzelnen Schichten zu scrollen. Je nach Version und Betriebssystem werden Wände, Füll- und Stützmaterial in anderen Farben dargestellt.

Wenn alles korrekt ist, kann man oben rechts auf „Save“ drücken und den generierten G-Code abspeichern. Diese Datei wird dann in der nächsten Etappe auf den Drucker geschickt.

7. Repetier Host

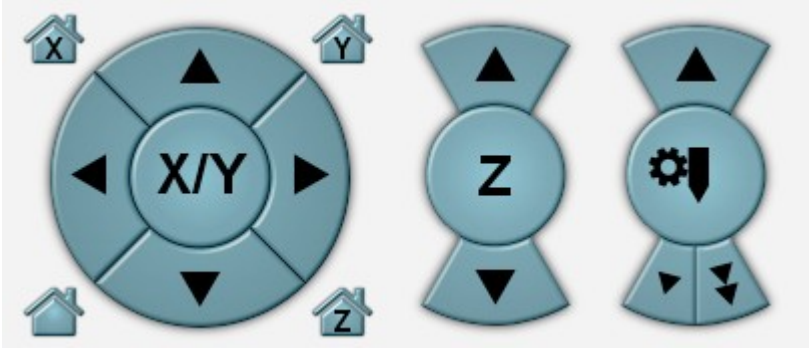
Es gibt diverse Programme um G-Code an einen Drucker zu senden. Wir verwenden hier die freie Software „Repetier Host“, da diese auch mir unserem Felix 3D Drucker mitgeliefert wurde und relativ einfach zu bedienen ist.

Die wichtigsten Schaltflächen befinden sich oben link in der Ecke:



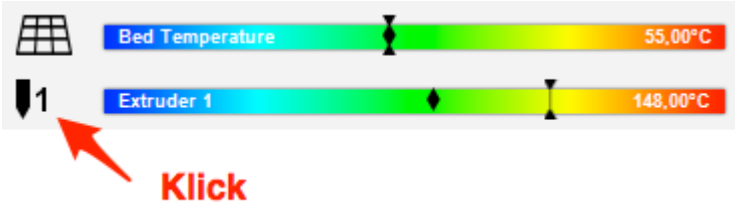
1. Verbindung mit dem Drucker herstellen
2. G-Code Datei Laden
3. Druckvorgang starten
4. Druckvorgang abbrechen

Auf der rechten Seite gibt es diverse Tab. Mit Hilfe des Tab‘s „Manual Control“, kann man den Drucker manuell steuern:



Der obige Teil erlaubt es die Achsen so wie den Extruder zu bedienen.

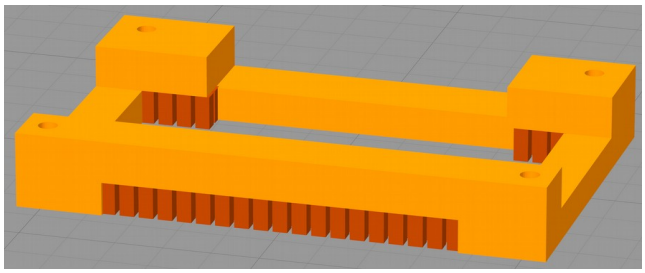
Etwas weiter unten kann man die Temperatur des Bettes so wie des Extruders einstellen und ihre Heizelemente mittels einem Klick auf die Ikonen aktivieren oder deaktivieren.



8. 3D-Druck des Gehäuses

8.1. LCD-Halter

Zum Drucken wird der Halter um 180° gedreht und mit Support versehen:

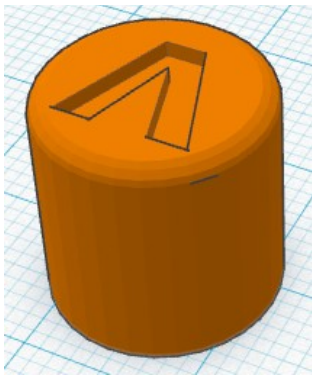


Druckzeit bei 0,2mm: 0,4 Stunden
PLA 1,6m: 4,8g (0,12€)

Da Löcher nach dem 3D-Druck öfter noch leicht verstopft sind, sind diese mit einem 2mm Bohrer aufzubohren.

8.2. Taster

Siehe Seite 29, Kapitel "Erstellen eines Knopfes"

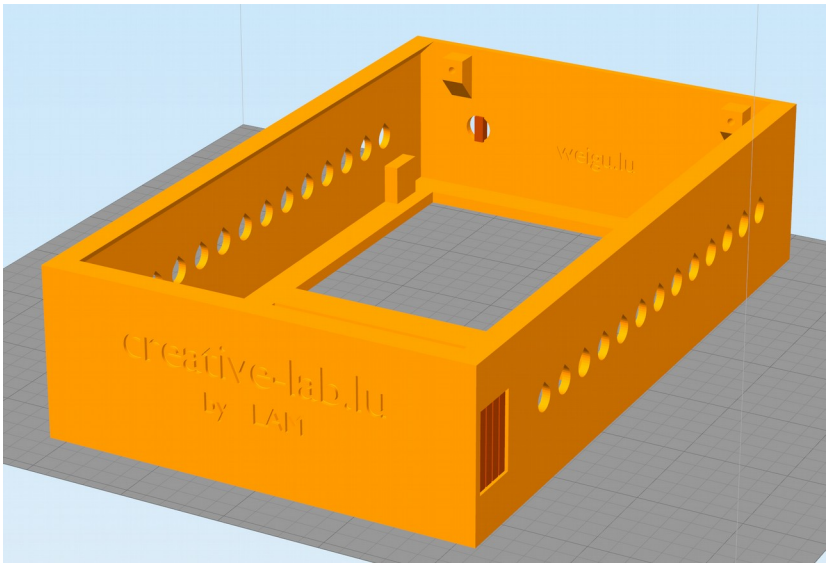


Der Knopf kann ohne Stützen geduckt werden

Druckzeit für 2 Taster bei 0,1mm: 0,4 Stunden
PLA 0,67m: 2g (0,05€)

8.3. Das untere Gehäuseteil

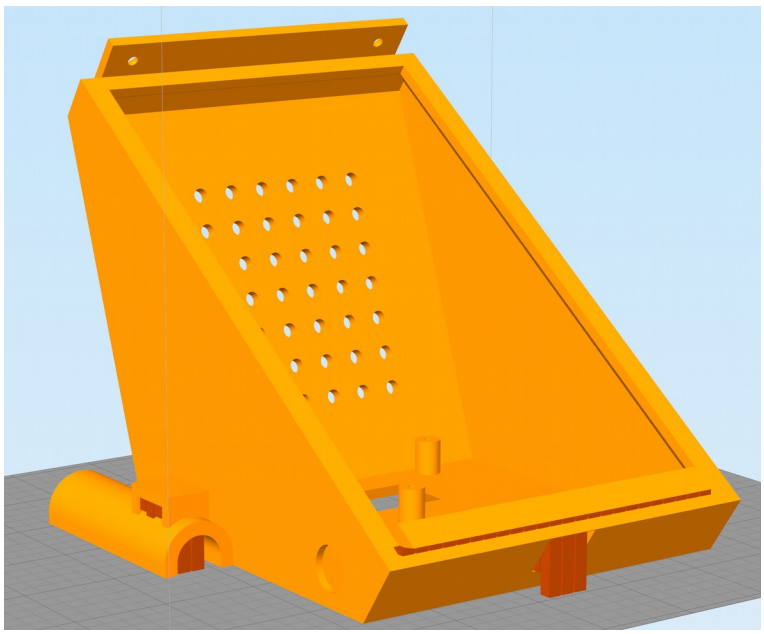
Das untere Gehäuse nimmt das Schaltnetzteil auf. Support ist nicht unbedingt nötig. Sinnvoll ist er nur bei der Schalteröffnung.



Druckzeit bei 0,2mm: 6 Stunden
PLA 23m: 67g (2 €)

8.4. Das obere Gehäuseteil

Support ist nötig im Bereich der Lötspitzen-Halterung und der USB Buchse. Mit der 3D-Software Simplify3D kann dieser Support manuell eingestellt werden.



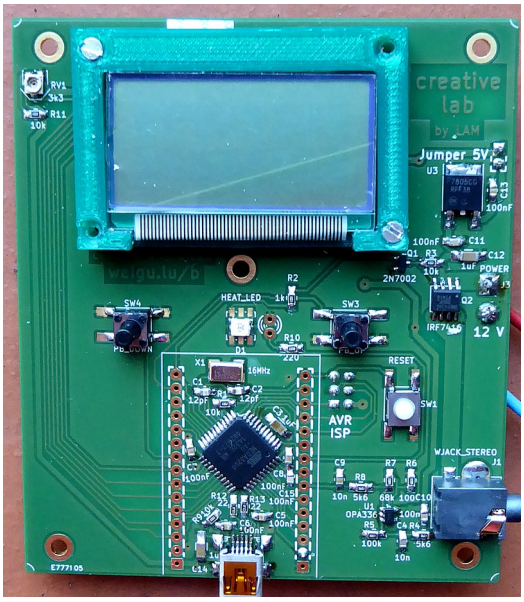
Druckzeit bei 0,2mm: 11 Stunden
PLA 42m: 123g (3 €)

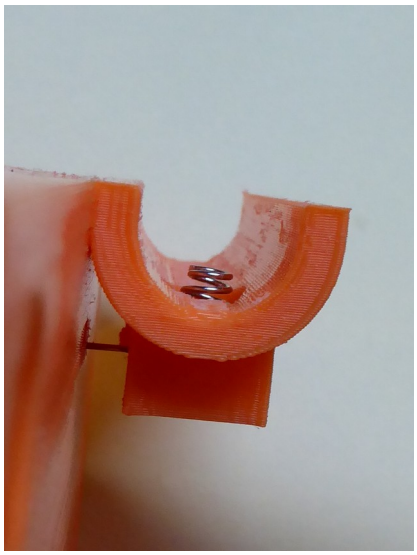
PART 4: Zusammenbau

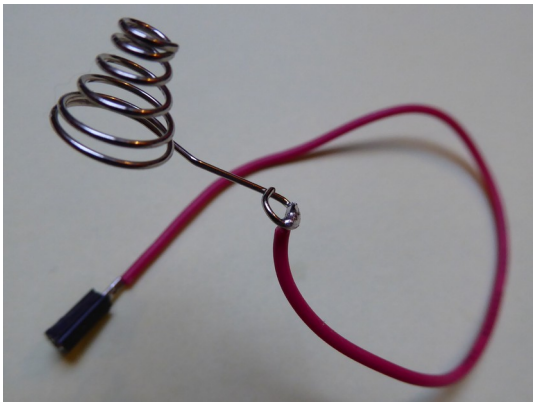
1. Platine einbauen & Federkontakt anbringen

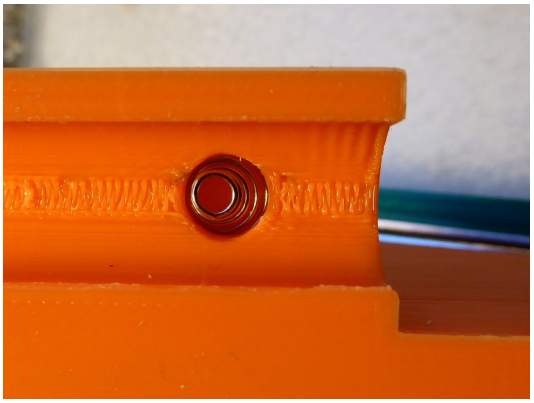
Jetzt muss das Gehäuse noch zusammen gebaut werden.

Die Löcher des LCD-Halter ev. mit einem 2mm Bohrer aufbohren. Das an der Platine schon befestigte LCD-Display wird durch den LCD-Halter durchgefädelt. Dieser wird dann mit 2-4 M2 Senkschrauben mit der Platine verbunden.









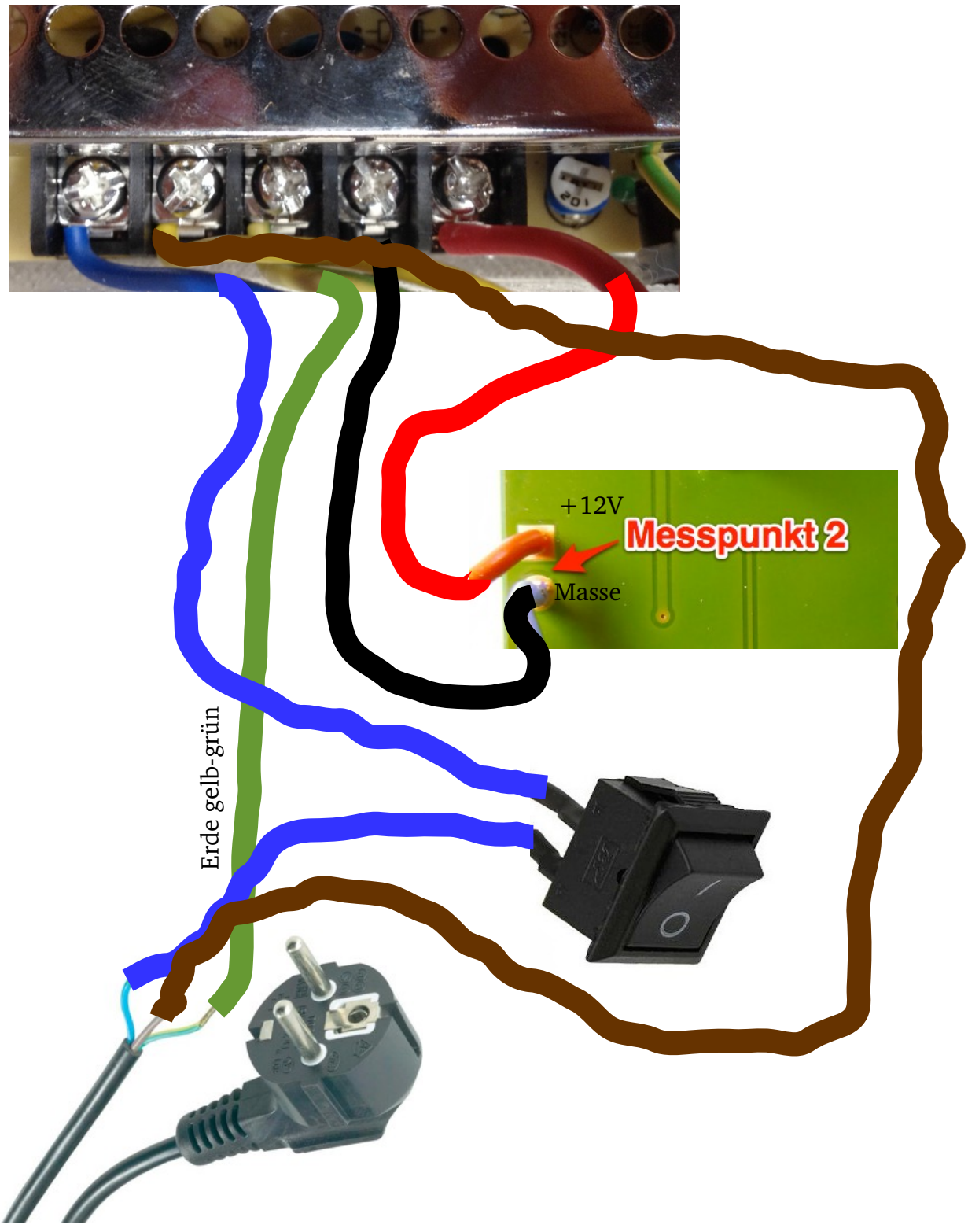
Danach wird eine Feder aus einem Batteriehalter recycled. Die Feder wird mit einem Stück flexiblen Draht (10cm) verlötet und mit einem Stecker (Buchsenleiste, Stiftleiste an der Platine) versehen. Dann wird die Feder zusammengedrückt und wie auf dem Bild in das Gehäuse eingefügt. Der Draht wird durch das Loch nach innen geführt.

Jetzt kann die Platine mit 3-5 Schrauben am oberen Teil des Gehäuses befestigt werden (Bild übernächste Seite). Davor die Schutzfolie des Displays abziehen!

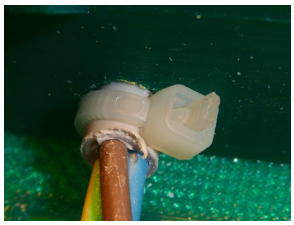
Dann wird die Feder noch mit Pin A0 verbunden und die 12V Versorgung kann angeschlossen werden.

2. Anschluss des Netzteiles

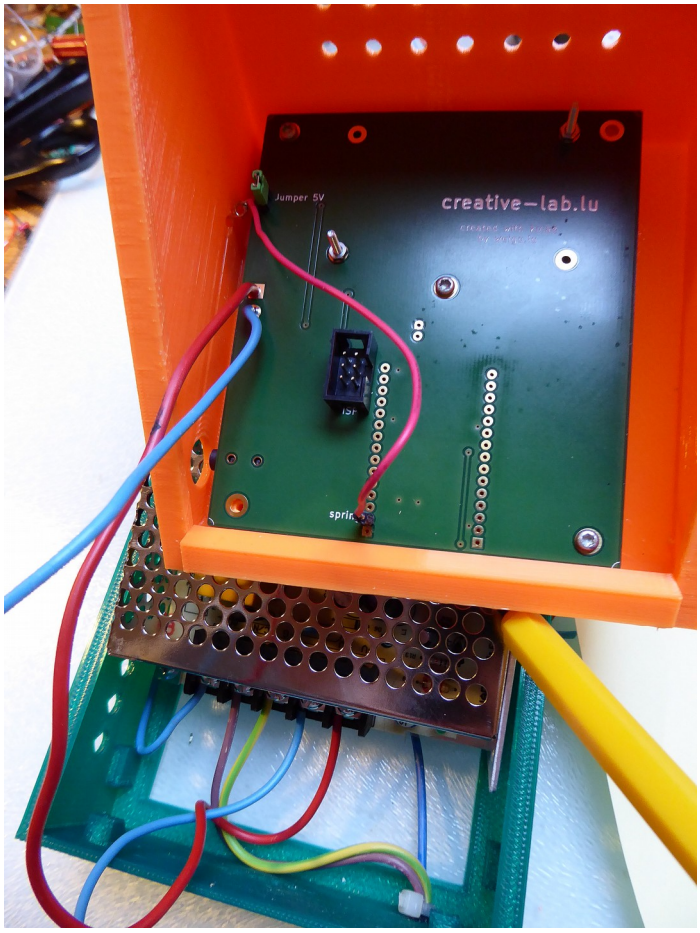
Das Netzteil muss wie folgt angeschlossen werden:



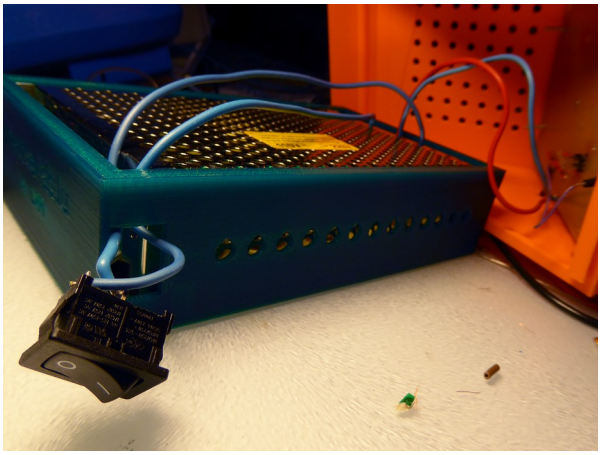
Hierzu recyceln wir ein altes Netzteilkabel. Dazu wird ein ev. vorhandene Kaltgerätbuchse abgeknipst und der Mantel auf 30cm entfernt.



Das Kabel wird durch das vorgesehene Loch im hinteren Teil des unteren Gehäuseteils durchgeführt und mit einem Kabelbinder zur Zugentlastung versehen.



Zwei der drei Adern (gelb-grün und braun) werden auf 10 cm verkürzt, einige Millimeter abisoliert und mit Aderendhülsen versehen. Die dritte Ader wird durch das Schalterloch (vorne) geführt und mit dem Schalter verlötet. Ein weiteres Stück Ader (30 cm, min. 0,75mm²) wird mit dem Schalter verlötet und am anderen Ende mit einer Aderendhülse versehen.

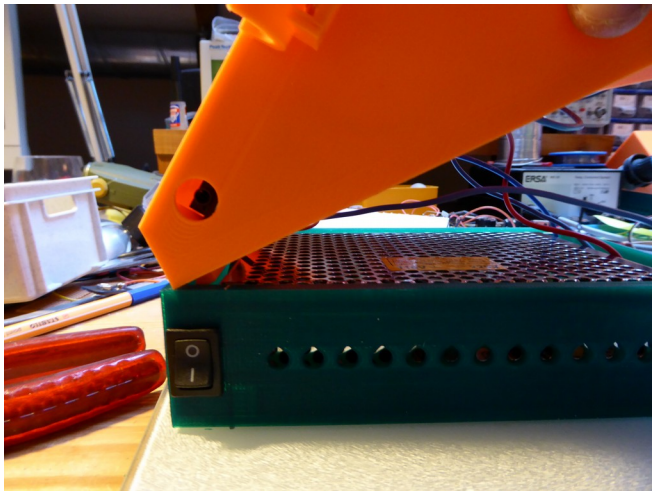


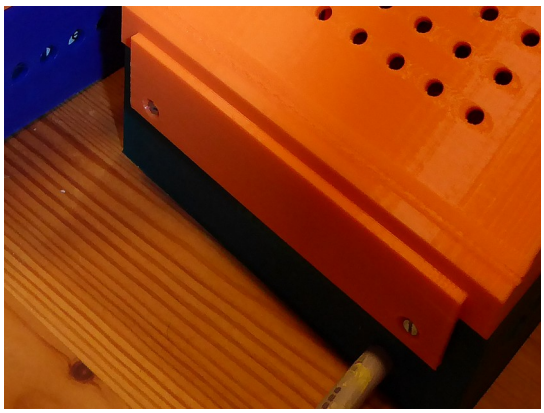
Das Netzteil wird angeschlossen (5 Adern festschrauben) bevor es ganz in das untere Gehäuseteil eingeschoben wird, da das Verschrauben sonst schwierig wird.

Dann wird das Netzteil nach hinten geschoben und der Schalter kann ins Gehäuse gedrückt werden.

3. Gehäuse zusammenbauen

Die beiden Gehäuseteile werden vorne einfach ineinander-geschoben und hinten mit zwei Senkopfschrauben verbunden.





Und dann haben wir

FERTIG!!


CREATIVE LAB
CREATIVE LAB
CREATIVE LAB
weigu.lu