Baumappe zur Entwicklerplatine "Freduino" auf Grundlage des Attiny13a

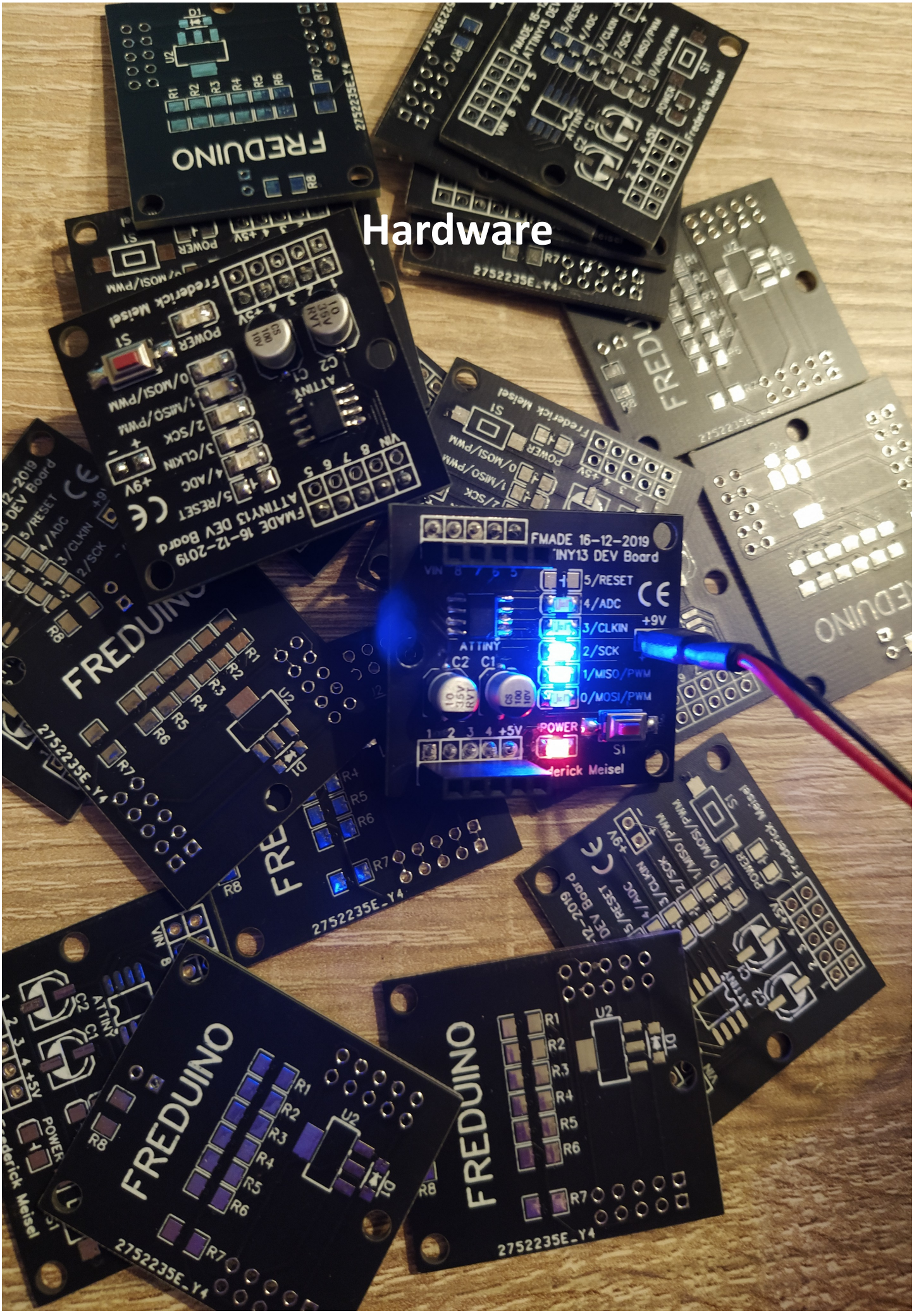
(Rev. 1.0)

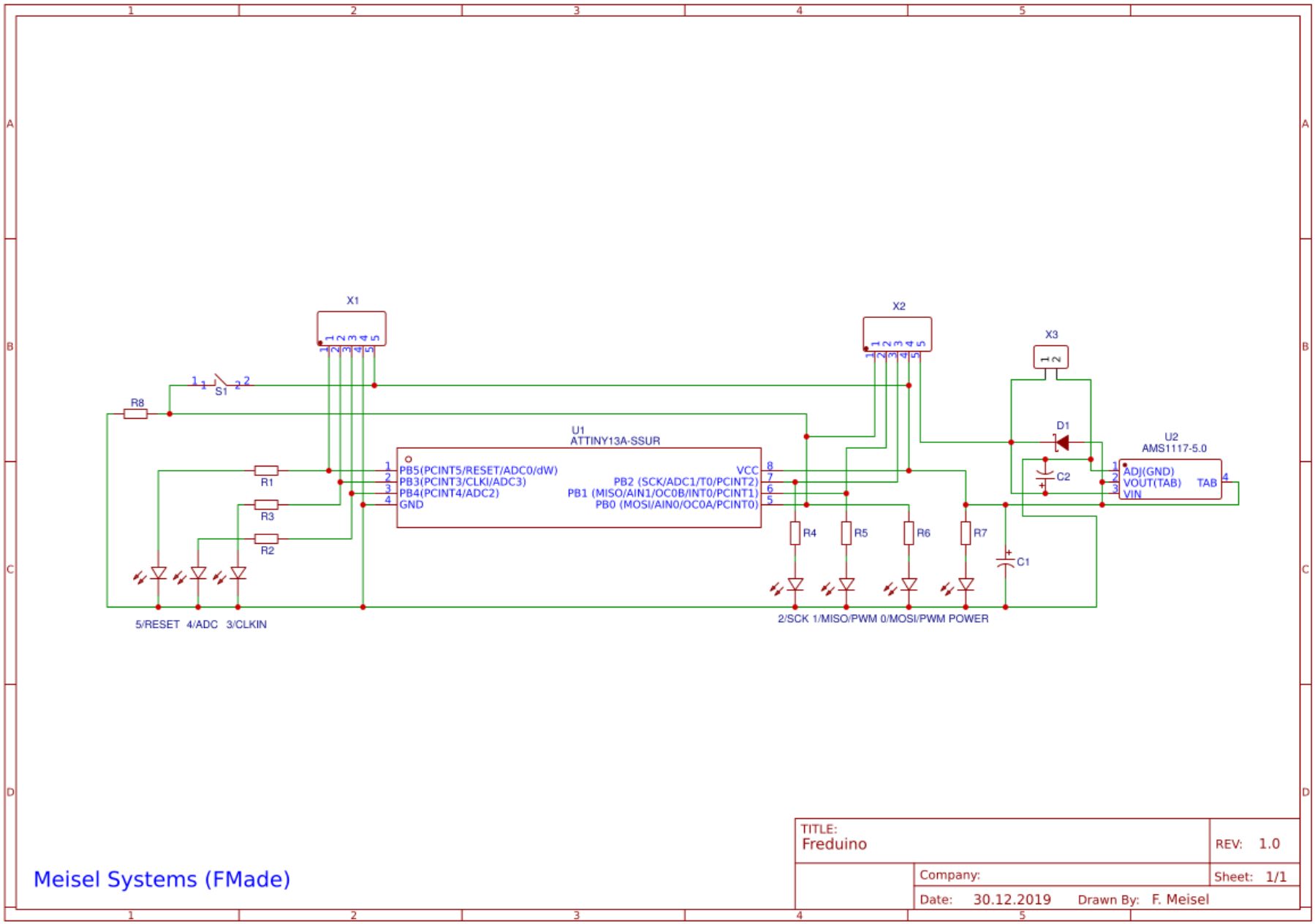
Inhalt

Da diese Baumappe wichtige Informationen zum Aufbau und Betrieb enthält, sollten Sie vor dem Aufbau und der Inbetriebnahme die gesamte Baumappe sorgfältig lesen.

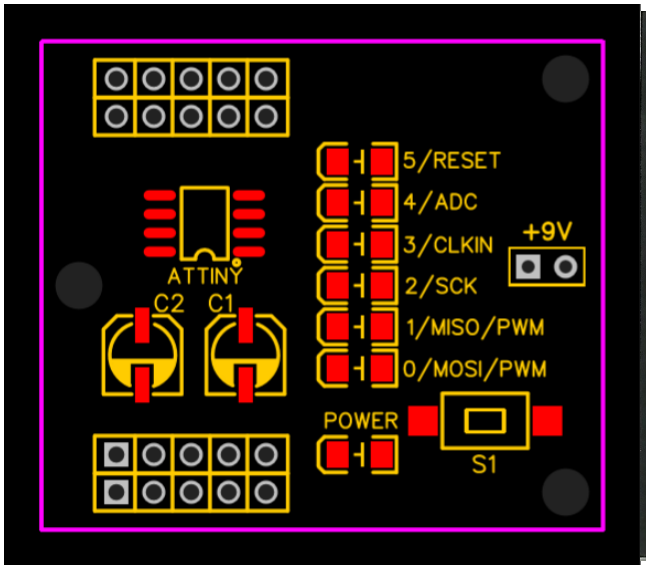
01.01.2020, Frederick Meisel

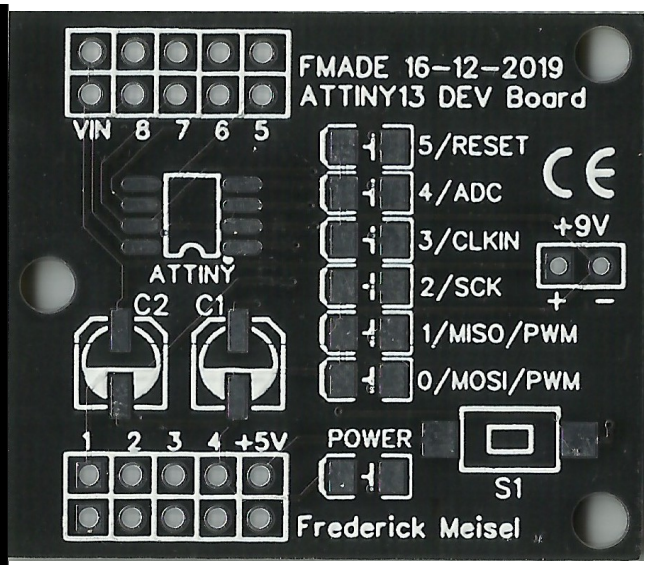
Hardware


FREDUINO
FMADE 16-12-2019
ATTINY13 DEV Board
5/RESET
4/ADC
3/CLKIN
2/SCK
1/MISO/PWM
0/MOSI/PWM
+9V
POWER
Frederick Meisel
2752235E_Y4


X1
X2
X3
S1
R8
R1
R3
R2
U1
ATTINY13A-SSUR
PB5(PCINT5/RESET/ADC0/dW)
PB3(PCINT3/CLKI/ADC3)
PB4(PCINT4/ADC2)
GND
VCC
PB2 (SCK/ADC1/T0/PCINT2)
PB1 (MISO/AIN1/OC0B/INT0/PCINT1)
PB0 (MOSI/AIN0/OC0A/PCINT0)
R4
R5
R6
R7
C1
D1
C2
U2
AMS1117-5.0
ADJ(GND)
VOUT(TAB)
VIN
TAB
5/RESET 4/ADC 3/CLKIN
2/SCK 1/MISO/PWM 0/MOSI/PWM POWER
TITLE:
Freduino
REV: 1.0
Company:
Sheet: 1/1
Date: 30.12.2019
Drawn By: F. Meisel
Meisel Systems (FMade)

Platinenlayout/Bestückungsplan

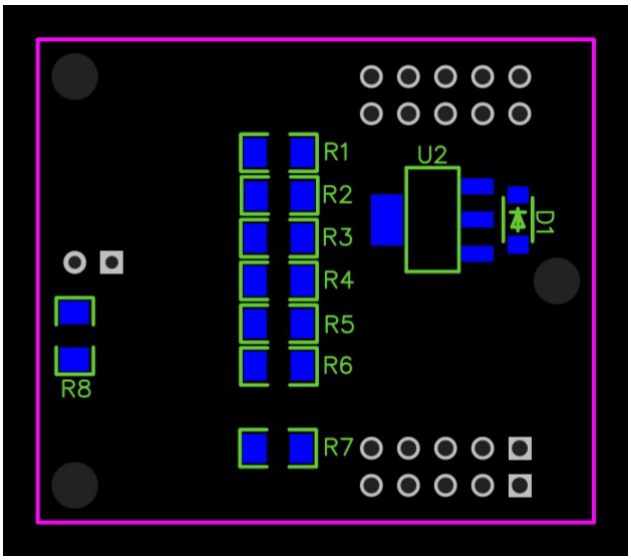


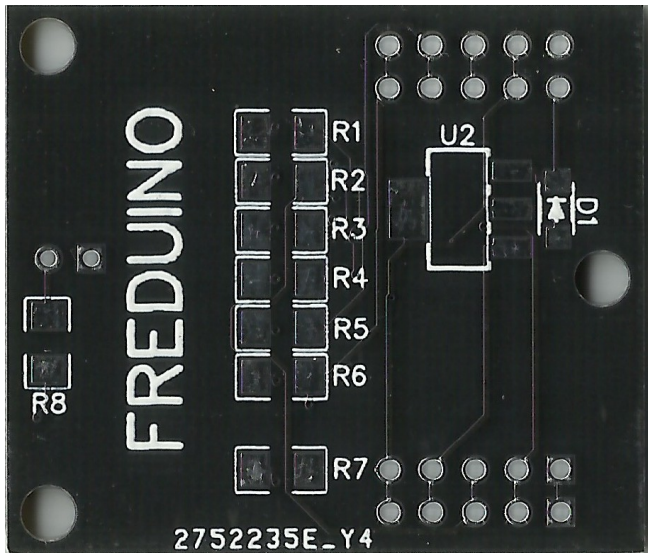


Bestückungsplan Oberseite

zu Beachten: Der Grüne Punkt an den SMD - LEDs zeigt die Kathode.

PB 5 / RESET wird nicht bestückt!

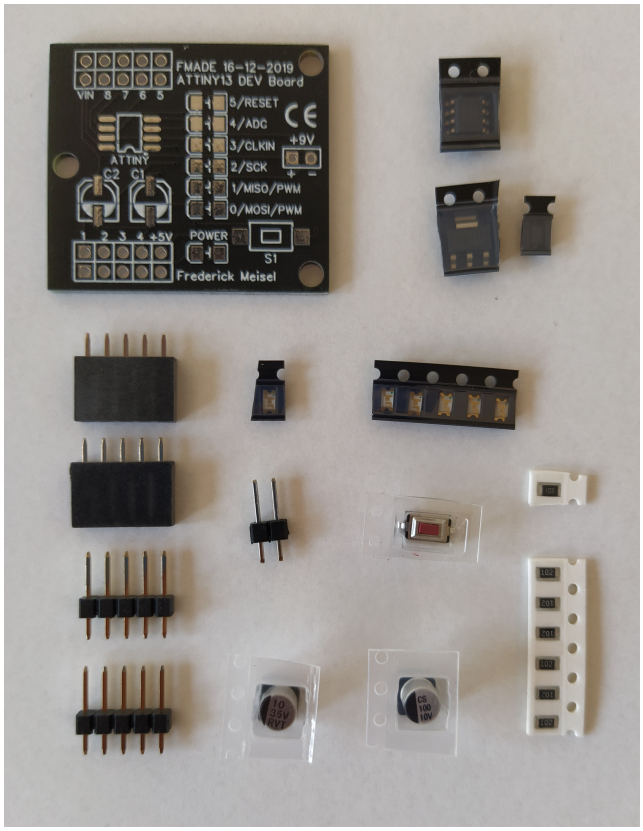




Bestückungsplan Unterseite

Stückliste

Nr.	Menge	Bezeichnung	Typ	Wert	Quelle	Art. Nr.	Stückpreis
1	1		Platine	Dev.	JLCPCB	--	0,22 €
2	1	ATTINY	Attiny	13A	LCSC	C40382	0,48 €
3	1	U2	Spannungsregler	5V	LCSC	C347223	0,03 €
4	1	D1	Shottky Diode	40V Ur	LCSC	C181206	0,01 €
5	1	LED1	Led	Rot	LCSC	C400035	0,01 €
6	5	LED2 – 6	Led	Blau	LCSC	C400036	0,01 €
7	2	X1, X2	Pin header (Female)	5-pin	LCSC	C350302	0,05 €
8	2	X1, X2	Pin header (Male)	5-pin	LCSC	C358687	0,02 €
9	1	C1	Elko SMD	10uF/35V	LCSC	C72486	0,03 €
10	1	C2	Elko SMD	100uF/10V	LCSC	C96182	0,02 €
11	1	S1	Taster	SMD	LCSC	C329185	0,03 €
12	1	R8	Widerstand	10k	LCSC	C380783	0,00 2€
13	6	R1 - R6	Widerstand	1k	LCSC	C115421	0,00 2€
						Gesamt:	1,03 €



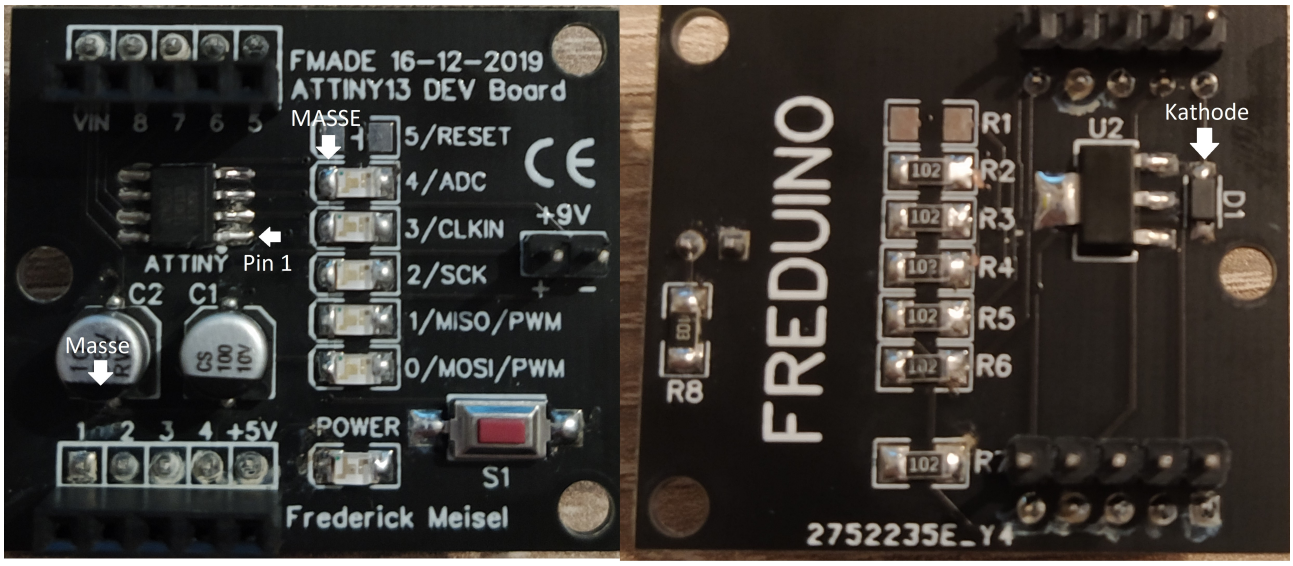
Widerstandswerte und LED Farben können verändert werden.

Überprüfungsprotokoll

Zum Testen der Schaltung nehmen sie sich ein Multimeter zur Hand.
Sollte einer der Schritte fehlschlagen, korrigieren sie den Fehler und beginnen den Fehlgeschlagenen Schritt erneut.

Achten sie zuerst auf die richtige Plazierung des Microcontrollers. Dieser sollte mit dem Punkt (Pin 1) in richtung Kerbe zeigend eingebaut werden.
Nun sollten alle LEDs auf die korrekte Einbaurichtung getestet werden, dazu nutzen sie den Dioden-Messbereich ihres Multimeters.
Wenn dies erfolgt ist muss noch die Einbaurichtung der Kondensatoren überprüft werden. Die Pins unterhalb der Schwarzen Markierung müssen hierbei auf Masse liegen.
Die Schottky-Diode parallel zum Spannungsregler sollte ebenfalls korrekt eingebaut sein. Die Kathode ist mit einer kaum erkennbaren, jedoch vorhandenen Linie gekennzeichnet.
Diese Linie sollte in Richtung der Eingangsspannung Zeigen.

Die Fertig bestückte Platine sollte folgendermaßen aussehen:



Technische Daten

Eingangsspannung:	Attiny13A direkt:	1,8V - 5V
	AMS1117:	6,3V - 12V
Stromabgabe:	PB0-5:	40mA (MAX!)
	+5V:	1A
	[Der Spannungsregler hat eine maximale Verlustleistung von ~1,4W]	
Stromaufnahme:		10mA – 1,8A
Betriebstemperatur:		-40°C - +85°C
Programmspeicher:		1kByte/Flash
EEPROM Speicher:		64Byte/EEPROM
Arbeitsspeicher:		64Byte/SDRAM
CPU frequenz:		0,128MHz - 9,6MHz
PWM frequenz:		1/256 CPU frequenz
Länge:		33mm
Breite:		38mm
Höhe(Voll bestückt):		18mm (+-1mm)

Pinbelegung

X1 Name	Bedeutung	X2 Name	Bedeutung
1	Attiny Pin 1	Vin	Vin (AMS1117)
2	Attiny Pin 2	8	Attiny Pin 8
3	Attiny Pin 3	7	Attiny Pin 7
4	Attiny Pin 4	6	Attiny Pin 6
+5V	Vout (AMS1117)	5	Attiny Pin 5

Datei Bearbeiten Sketch Werkzeuge Hilfe

L_ftersteuerung

```
  last_millis = millis();//set the start millis

}

void loop() {

  if(digitalRead(butt) == HIGH and active == false and millis(

    active = true;//activate
    digitalWrite(led, HIGH);//show that it is running

    last_millis = millis();//set the last_millis

  }

  delay(50);//slow it all down a bit

  if(digitalRead(butt) == HIGH and active == true and millis()

    active = false;//deactivate
    digitalWrite(led, LOW);//show that it is not running

    last_millis = millis();//set the last_millis

  }
```

Software

Kompilieren abgeschlossen.

```
Der Sketch verwendet 580 Bytes (56%) des Programmspeicherplatz
Globale Variablen verwenden 10 Bytes (15%) des dynamischen Spe
```

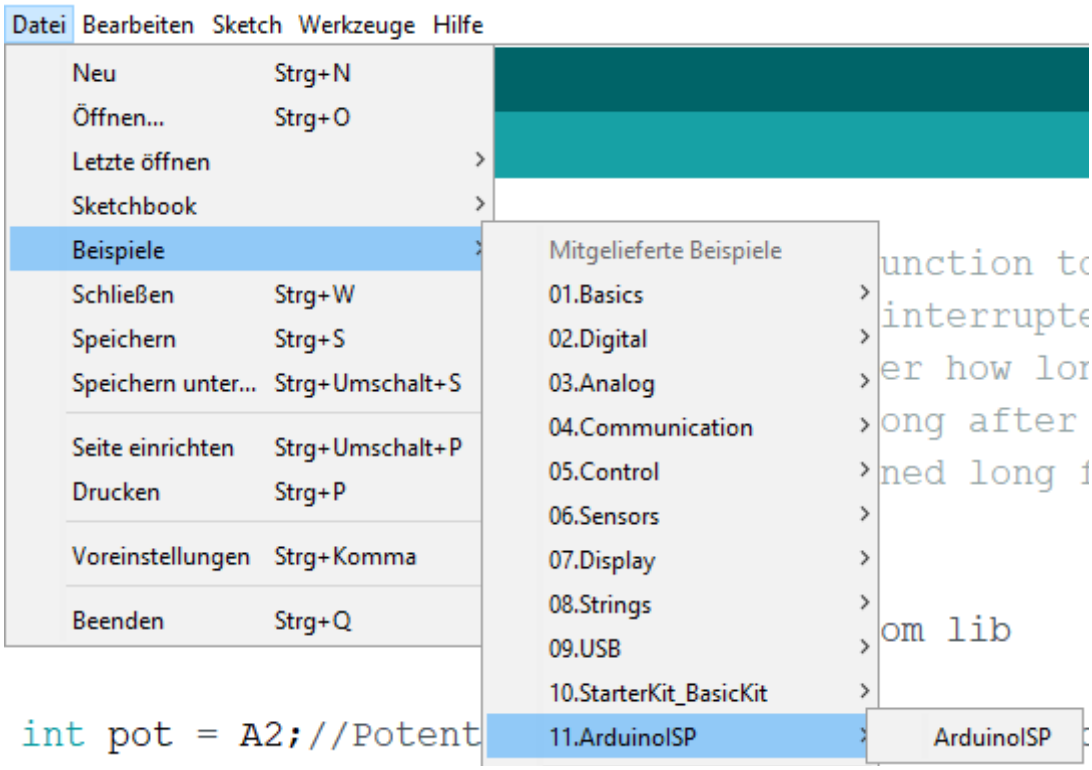
ATtiny13, 9.6 MHz internal osc., BOD 1.8V, LTO enabled (default) auf COM3

Setup des Programmiergeräts

Da der Attiny13 nicht über eine Uart- oder gar USB-Schnittstelle verfügt muss der sorgfältig programmierte Code per SPI programmer auf den Chip übertragen werden. Dies geschieht über den sogenannten ICSP (**i**n **c**ircuit **s**erial **p**rogrammer), oft auch ISP genannt.

Arduinos haben diese Schnittstelle standardmäsig verbaut, da in ihnen ebenfalls AVR-Controller verwendet werden.
Nun kann man diesen Vorteil nutzen, um den oft kostspieligen Kauf eines Programmiergeräts zu umgehen.

Der Arduino Uno oder Nano muss zuerst mit einem Programm bespielt werden, welches man in der Arduino IDE unter:
Datei -> Beispiele -> 11.ArduinoISP -> ArduinoISP findet.



Ist dieser Sketch nun auf dem Board "installiert" folgt der nächste Schritt.

Setup des Programmiergeräts

Der Freduino hat an den Pin-leisten X1 und X2 Beschriftungen, die auf die jeweiligen Funktionen deuten. (Siehe Tabelle Seite 7)

Die beiden Boards müssen nun folgendermaßen verbunden werden:

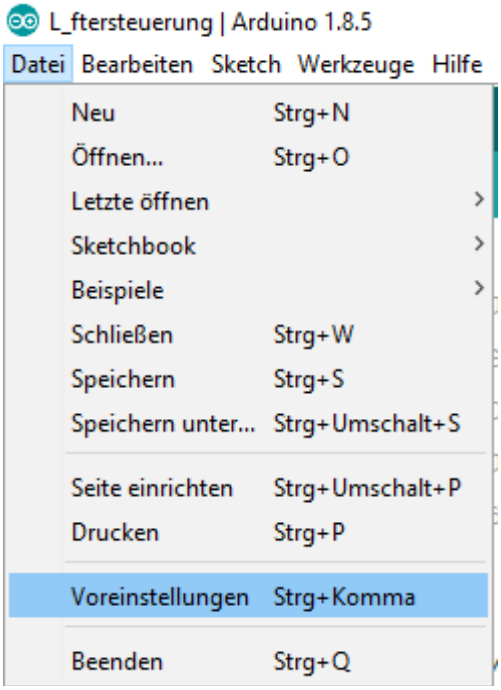
Arduino	Freduino	Funktion
+5V	+5V	+Ub
GND	GND	Masse
10	1	Reset
11	5	MOSI
12	6	MISO
13	7	SCK

Der Host-Controller (Arduino) ist nun per SPI ISP mit dem Slave-Controller (Freduino) verbunden. Das Programmieren läuft so ab:

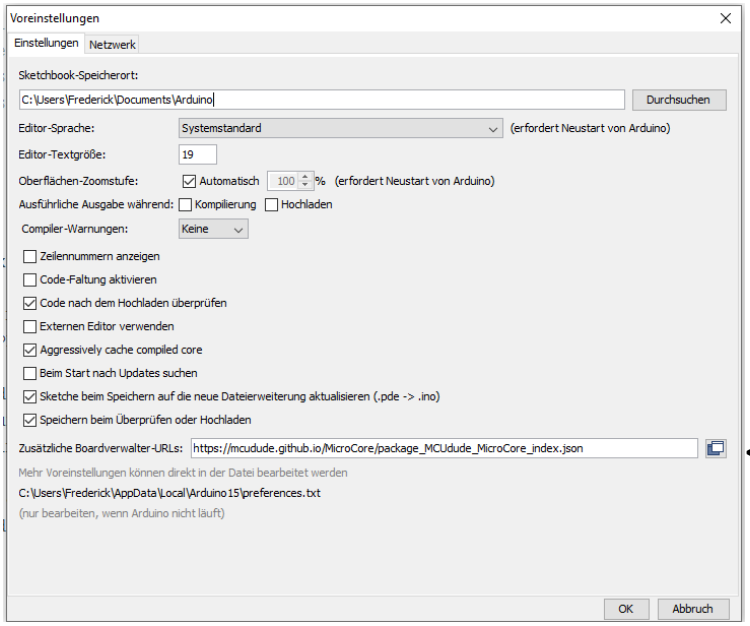
SCK (**S**erial **C**loc**k**) dient hierbei als Taktsignal (128kHz – 9,6MHz)
Reset wird Aktiviert und gehalten. (Active LOW)
Die Controller tauschen untereinander mittels MOSI und MISO Informationen, wie Prozessortakt, frei belegbarer Speicherplatz, Prozessortyp... etc. aus.
Wurde dieser "Handshake" von beiden Seiten bestätigt wird das Programm in Maschinensprache (Binär, Seriell) über den MOSI (**M**aster **o**ut **s**lave **i**n) Pin an den Attiny gesendet. Sobald der Attiny das Programm empfangen hat sendet er es zur Validation über den MISO (**M**aster **i**n **s**lave **o**ut) pin zurück an den Host. Wenn beide Partner zustimmen, dass das Programm richtig übertragen wurde ist der Prozess beendet und der Reset wird deaktiviert.

Einbindung in die Arduino IDE

Um den Attiny13 nun auch Programmieren zu können werden zuerst einige Boardbibliotheken benötigt. Hierzu öffnet man in der Arduino IDE die Voreinstellungen:
Datei -> Voreinstellungen



Nachdem man das gemacht hat sollte Folgendes Fenster zu sehen sein:

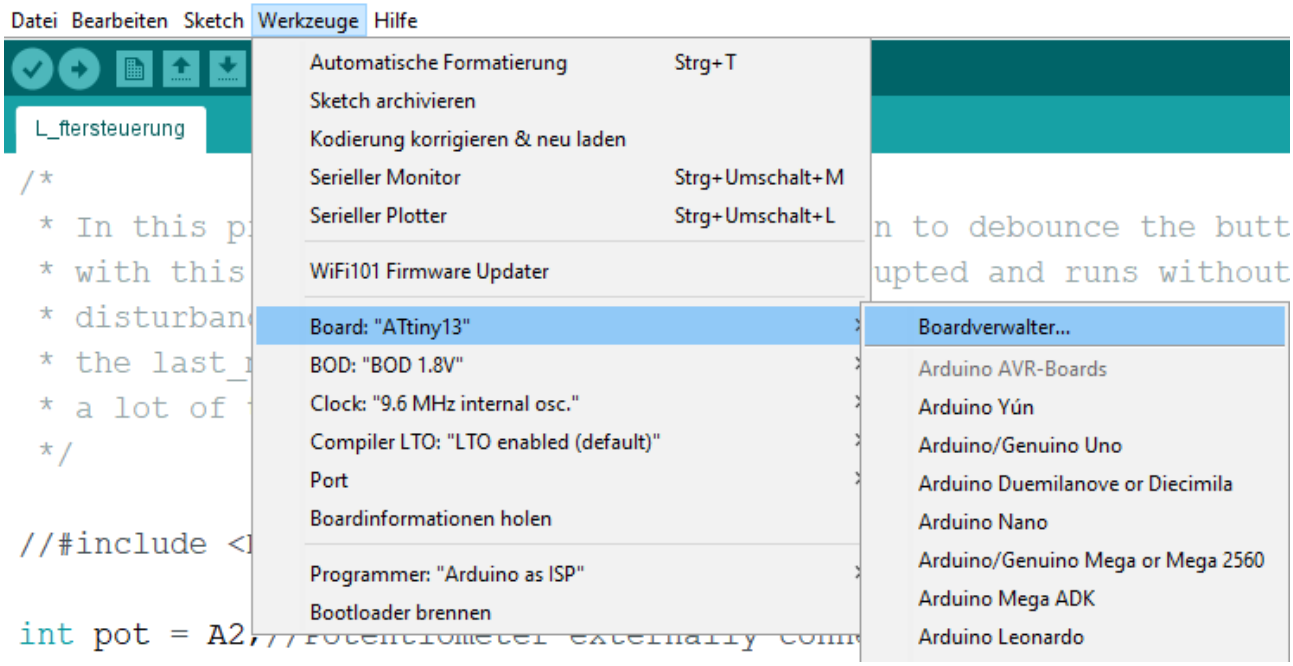


<- Nun fügt man Hier den folgenden Link ein.

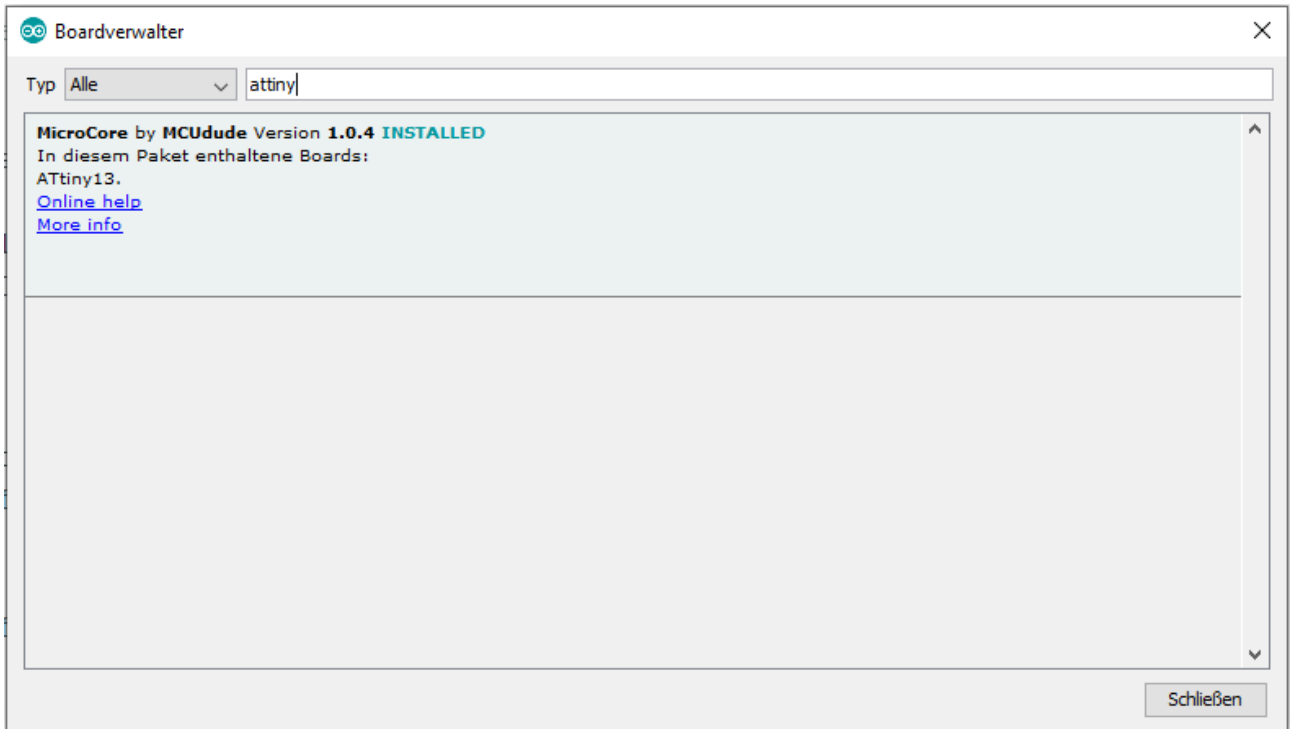
https://mcudude.github.io/MicroCore/package_MCUdude_MicroCore_index.json

Einbindung in die Arduino IDE

Ist die URL eingefügt sollte im Boardverwalter ein neues Board auftauchen. Hierzu Öffnet man:
Werkzeuge -> Board -> Boardverwalter



Im Boardverwalter Fenster sucht man, nachdem die Bibliotheken heruntergeladen wurden nach "Attiny"...



...und installiert das Paket *1.0.4*.
Fertig! Nun kann der Bootloader gebrannt werden.

Bootloader Brennen

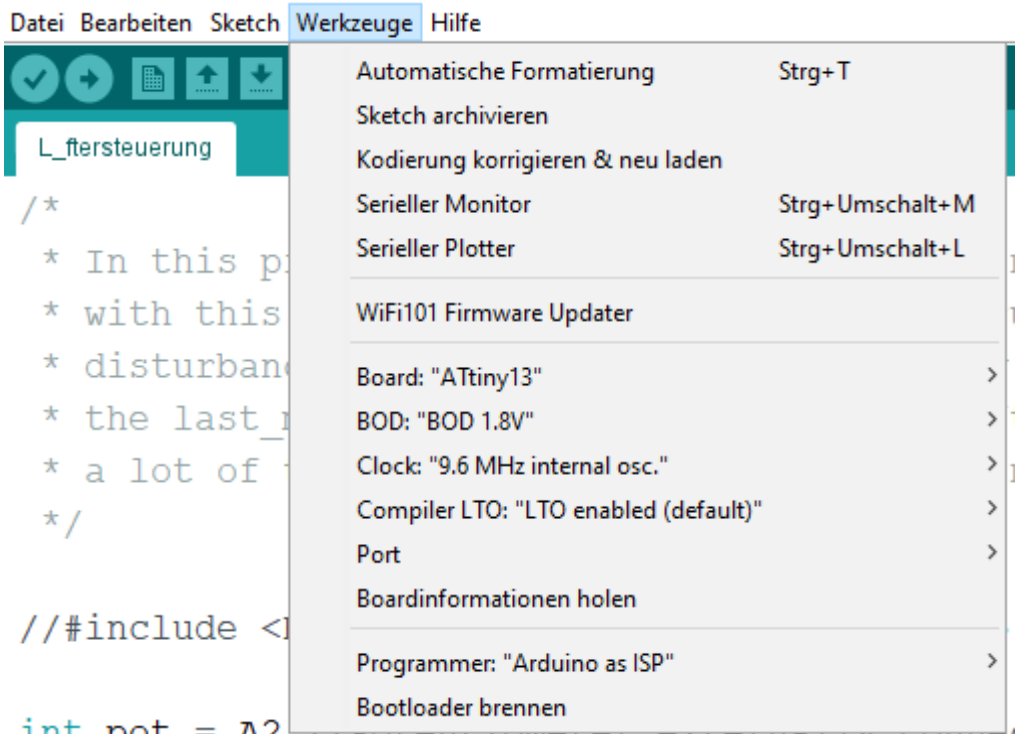
Für diesen Schritt muss der Freduino über das Arduino-board mit dem Pc verbunden werden.

Der Bootloader:
Als Bootloader verteht man das "BIOS" des Microcontrollers, er lädt das Hauptprogramm in das System.
Da der Attiny13 jedoch nur über 1kB Programmspeicher verfügt besteht der Bootloader aus sehr wenigen Befehlen und startet sehr schnell in das Hauptprogramm, im Gegensatz zum Bootloader des Atmega328 (Arduino Uno/Nano), der ungefähr eine Sekunde benötigt.
Im Bootloader werden Dinge wie Prozessortakt, Timer funktionen, Abschaltspannung und Programmierinformationen festgelegt.

Der letzte Schritt vor dem Programmieren besteht darin, den Bootloader zu brennen.
Hierzu öffnet man das Untermenü Werkzeuge und setzt Compilereinstellung, Prozessortakt und Abschaltspannung fest.

Nächste Seite ->

Bootloader Brennen



In diesem Menü hat man, wenn das Board "Attiny13" ausgewählt ist, die Optionen BOD (**B**rown **o**ut **d**etection [Abschaltspannung]), Clock (Prozessortakt) und Compiler LTO (**L**ink **t**ime **o**ptimisation [Verringert die Fehlerrate bei Datenübertragungen]) festzulegen. Diese Werte sollten mit denen auf dem Bild übereinstimmen.

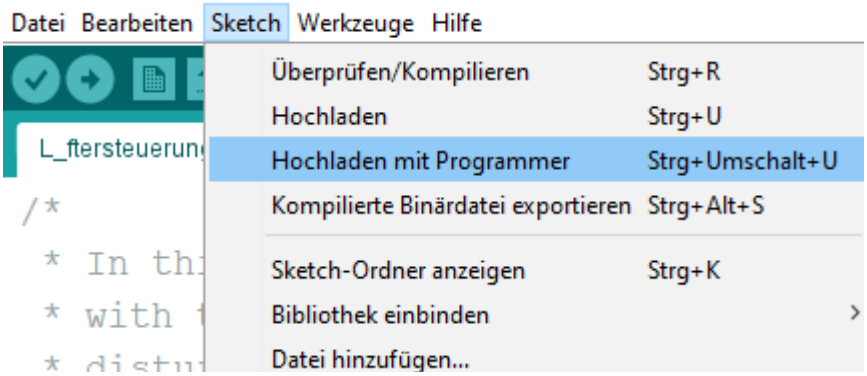
Board: Attiny13
BOD: 1,8V
Clock: 9,6MHz internal
LTO: enabled

[Vorsicht! Bei 128kHz kann man den Code nur mit einem AVR ISP auf den uC Programmieren. External Clock macht den Ic ebenfalls unbrauchbar ohne externen Takteingang]

Jetzt muss als Programmer noch "Arduino as ISP" ausgewählt werden, und man kann den Bootloader brennen.
(Klick auf Bootloader brennen.)
Der Bootloader wird nun installiert und kann beliebig oft geändert werden.

Aufspielen von Programmen

Jetzt, da der Bootloader installiert ist, ist es sehr Einfach Programme auf den IC zu spielen.
Hierzu muss lediglich Verbindung zum Computer hergestellt werden und der Sketch unter:
Sketch -> Hochladen mit Programmer



hochgeladen werden.

Hier ein Beispielsketch, der LED 4 blinken lässt:

LED-Blinker

```
/*
 * LED blinker sketch
 * Frederick Meisel
 * Freduino example
 * 07.01.2020
 */

void setup() {

  pinMode(4, OUTPUT);//initialize LED on PB4 as output

}

//Main loop
void loop() {

  digitalWrite(4, HIGH);   //turn on
  delay(1000);             //wait
  digitalWrite(4, LOW);    //turn off
  delay(1000);             //wait again

}
//end
```

Hardwaretestcode

Um sicherzustellen, dass alle LEDs, Widerstände und der Taster korrekt installiert wurden ist das Hardwaretestprogramm zu installieren. Es ist hier zu finden:
https://pastebin.com/MZFFMn4W

Falls Pastebin nicht zu erreichen ist, Hier ist der Code ausgeschrieben:

```
/*
 * Hardware Testcode
 * Frederick Meisel
 * Freduino programm
 * 07.01.2020
 *
 * In this program i use the millis() function to debounce the button,
 * with this procedure the code is not interrupted and runs without
 * disturbance.
 * The last_millis counter gets verry long after a bit of time, which caused
 * a lot of trouble, so I used an unsigned long for this type of data.
 */

int butt = 0;//on board Button on PB0

bool active = false;//stores THE data
unsigned long last_millis;//counter for debouncing
int debtime = 500;//debounce time

void setup() {

  DDRB = B011110;//Set PB1-4 as output, PB0 and 5 as input.<-Basically this is pinMode();

  last_millis = millis();//set the start millis

  for(int i=0; i<=2; i++){//Flash all leds for a short time

    PORTB = B011110;//Set PB1-4 HIGH. (Syntax: PORTB = B[PB5];[PB4];[PB3];[PB2];[PB1];[PB0];
    delay(100);
    PORTB = B000000;//Set PB1-4 LOW.<-Basically this is digitalWrite();
    delay(100);

  }

}
```

Hardwaretestcode

```
void loop() {
  if(digitalRead(butt) == HIGH and active == false and millis() - last_millis > debtime){//debounced
    active = true;//activate
    last_millis = millis();//set the last_millis
  }
  delay(50);//slow it all down a bit
  if(digitalRead(butt) == HIGH and active == true and millis() - last_millis > debtime){//debounced
    active = false;//deactivate
    last_millis = millis();//set the last_millis
  }

  if(active == true){
    PORTB = B000000;//Flash the LEDs Fast
    delay(100);
    PORTB = B111110;
    delay(100);
  }else{
    PORTB = B000000;//turn off when inactive
  }
}
//Viel Spaß beim Abschreiben :)
```

Anwendungsideen

Nun ist so ein Attiny13 Entwicklerboard schon etwas tolles, wenn man es selbst zusammengelötet hat, aber nach wenigen Blicken wirkt es für den normalverbraucher langweilig...
Deshalb habe ich hier einige Anwendungsbeispiele mit Beispielcode aufgelistet, um eventuell den Anreiz für Projekte zu schaffen.

LED Animationen:
https://pastebin.com/50eavzLz

Lüftersteuerung:
https://pastebin.com/3JfSh4mU

Spannungsregler:
https://pastebin.com/t9WN2dXh

Natürlich kann man auch selbst kreativ werden und eigenen Code schreiben, der Attiny13 akzeptiert mit dem MicroDude Core fast alle Befehle, die der Normale Arduino auch kennt und ist dabei so klein, dass man ihn in ein Batteriegehäuse quetschen könnte und fast alle batteriebetriebenen Dinge zum Infrarot-Empfänger verwandeln. Es ist ebenfalls möglich die *64byte* der internen EEPROM nutzen und die SPI schnittstelle zu verwenden. (Beispiele hierzu sind im Core enthalten)
Datei -> Beispiele -> Beispiele für Attiny13

Das Setup der Programmier und Entwicklerumgebung ist hiermit abgeschlossen.

Danksagung

Ohne Hilfe hätte ich dieses Projekt nie so weit führen können wie es jetzt ist. Ich habe viel gelernt und einiges auch wieder vergessen, aber alles in Allem war es eine erfolgreiche Reise.
Der Weg ist das Ziel.

Besten Dank an:

- ***-Łukasz Podkalicki*** als Lieferant für Detaillierte Informationen rund um Programmierung und Einbindung des Attiny
- **-Great Scott** als Lieferant für Informationen rund um Elektrotechnik, Microcontroller und Programmierung seit Beginn meiner "Karriere"
- **-Marian** als aktiver Befürworter des Projekts von Anfang an

Selbstverständlich waren die Produkte ettlicher User von Internetforen, Verfassern von Informationstexten und Programmierern teil meiner Internetrecherche, jedoch ist es unmöglich allen einzeln Danke zu sagen. Sie bleiben Helden im Herzen. :)

Ein Gehäuse sowie weitere "Shields" für die Basisplatine sind noch in Entwicklung, möglicherweise wird das Projekt öffentlich.

Im folgenden Teil befinden sich alle nötigen Datenblätter der verbauten Halbleiterelemente.

Baumappe und Dokumentation erstellt von:

_____

Frederick Meisel