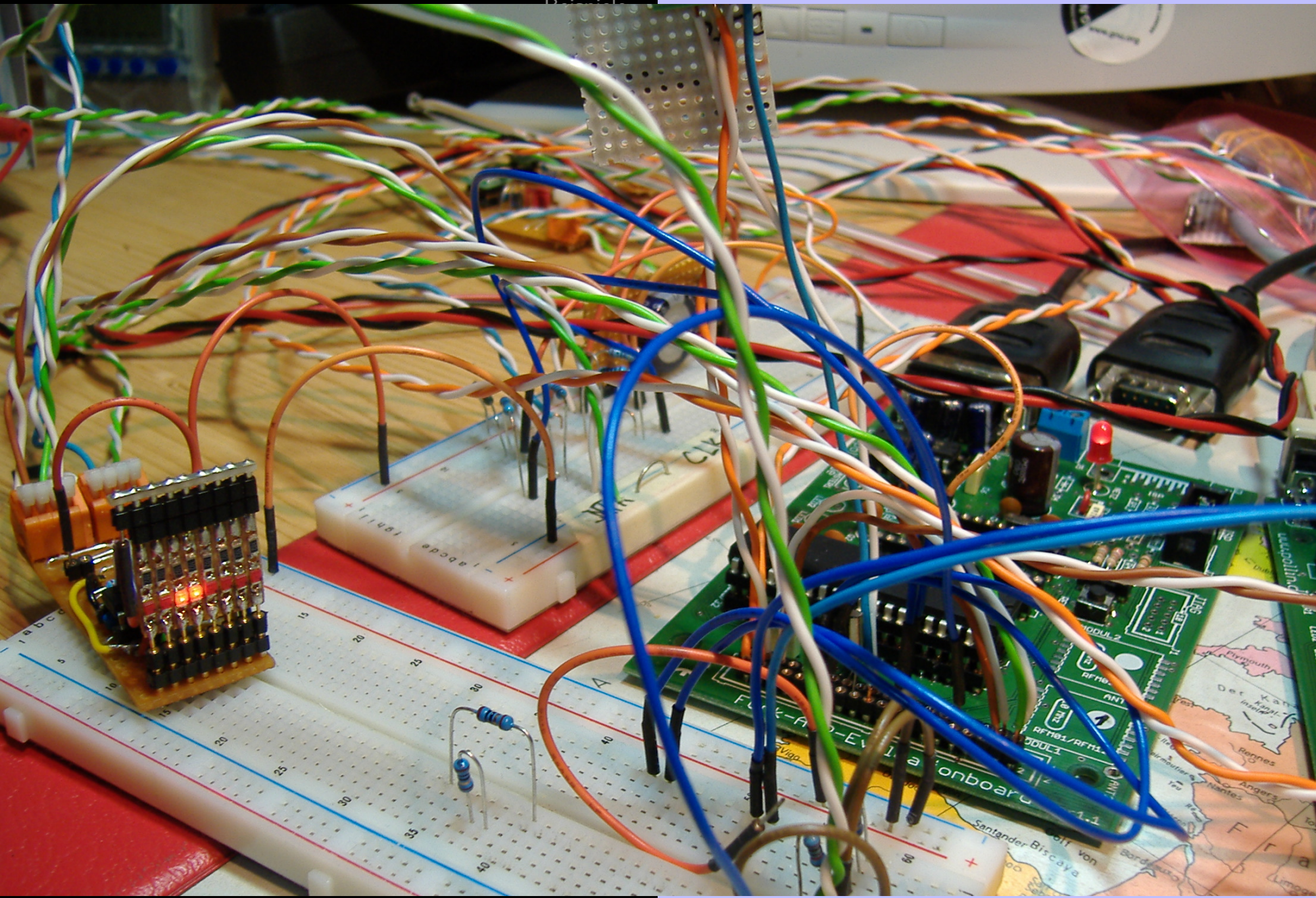
Dies und das zu atmega und amforth

Spaß mit Elektronikkruscht

Erich Wälde
ew.forth@nassur.net

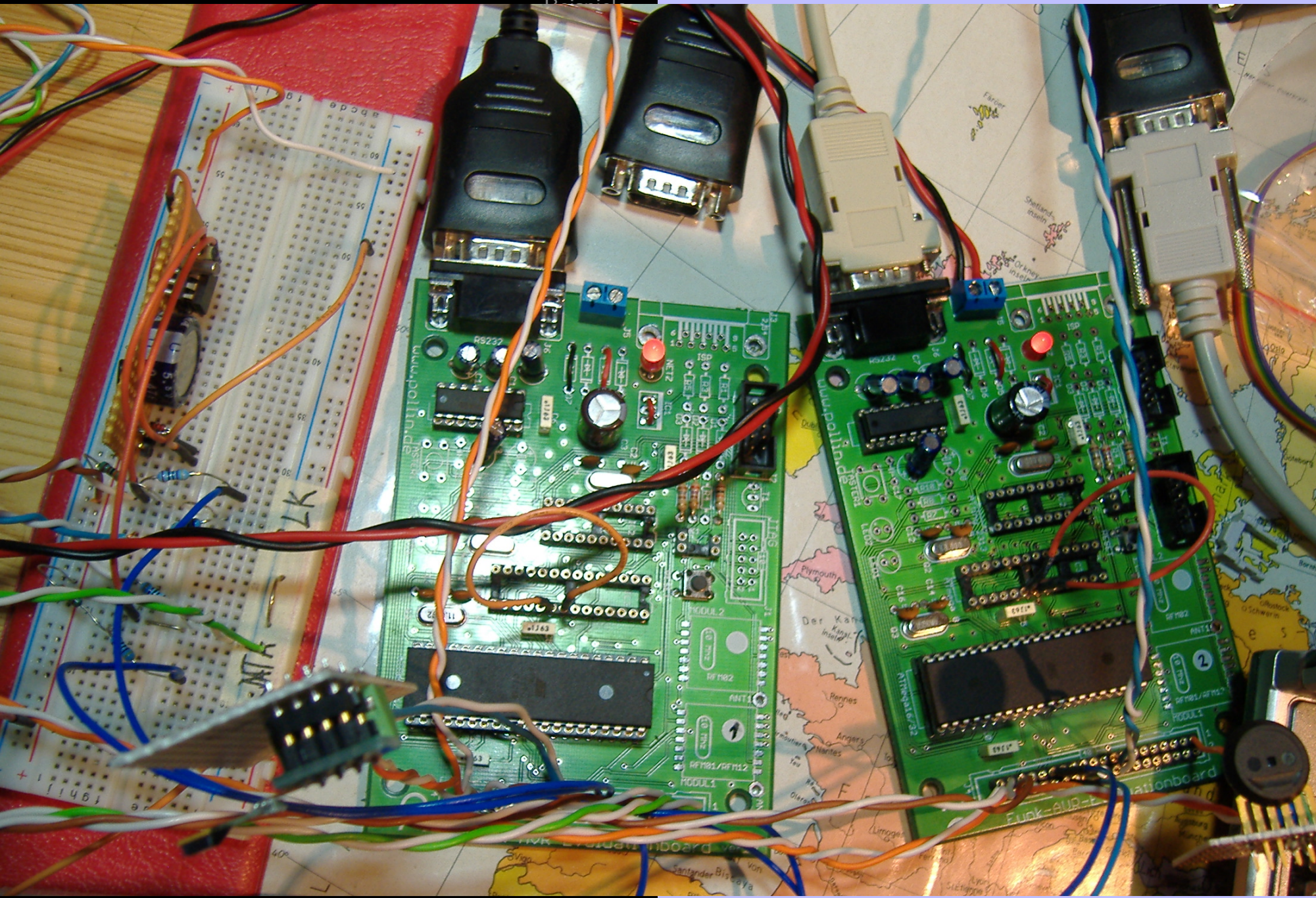
Linuxstammtisch Tübingen — März 2009



Wozu das Ganze?

Mensch kann damit

- Zeit versenken — das geht ganz hervorragend!
- LEDs blinken lassen — ist schnell langweilig
- das Klima messen (Temperatur, Feuchte, Luftdruck, . . .)
- die Zirkulationspumpe für's Warmwasser ein-/ausschalten
- den Füllstand der Regenwasserzisterne messen
- ein *fnordlicht* steuern
- einen Wasserhahn aufpeppen, so daß die Farbe der LED Beleuchtung zur Wassertemperatur passt — auch wenn das komplett nutzlos ist.
- einen Putz–Roboter steuern
- die Kollegen beeindrucken — funktioniert nicht immer.


RS232
ISP
JTAG
RFM02
RFM01/RFM12
MODUL1
MODUL2
www.pollin.de

Der Drahtverhau!

Im luftleeren Raum geht gar nixx. Ein Kontroller braucht Leitungen, Strom, die richtige Ansprache und andere Streicheleinheiten, sonst geht's ihm nicht gut:

- Lötkolben, Lot, Werkzeug ...
- Stromversorgung (Steckernetzteil o.ä.)
- Steckbrett oder Platine
- Ein paar Bauteile: Quarz, LEDs und so.
- etwas Luxus: eine Prototypen-Platine
- Programmer

Ansprache pur

```
C:003a7f 01fb

C:003a80 9169
C:003a81 9179

C:003a82 9731
C:003a83 f01a
C:003a84 0f66
C:003a85 1f77
C:003a86 cffb

C:003a87 cd86
```

Ansprache mit Merkhilfen

```
          PFA_LSHIFT:
C:003a7f 01fb           movw zl, tosl
C:003a80   +            loadtos
C:003a80 9169           ld tosl, Y+
C:003a81 9179           ld tosh, Y+
          PFA_LSHIFT1:
C:003a82 9731           sbiw zl, 1
C:003a83 f01a           brmi PFA_LSHIFT2
C:003a84 0f66           lsl tosl
C:003a85 1f77           rol tosh
C:003a86 cffb           rjmp PFA_LSHIFT1
          PFA_LSHIFT2:
C:003a87 cd86           rjmp DO_NEXT
```

Ansprache mit Übersetzung

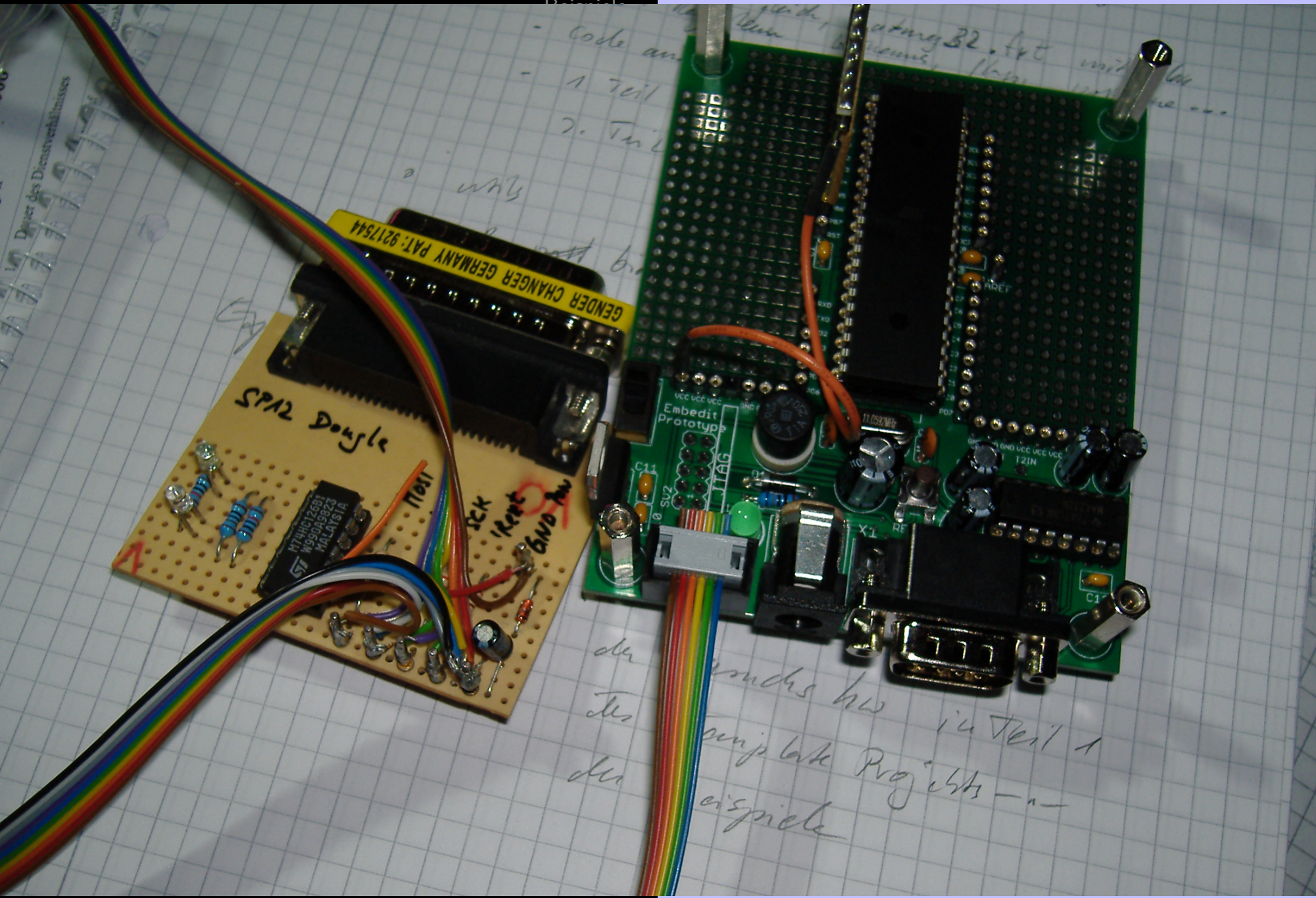
- Der Kontroller versteht nur eine Sprache: Maschinensprache.
- Der Programmierer versteht das normalerweise nicht fließend.
- Deswegen gibt es Merkhilfen (mnemonics) für die armen Programmierer
- Es gibt diverse besser lesbare Sprachen, z.B. C, Pascal, Bascom ...
- die werden normalerweise zuerst in Assembler übersetzt
- und dann in Maschinensprache.

Ansprache mit Übersetzung

```
void init_ports(void)
{
  DDRA = 0xff;                  // portA is output


  portA = 0x55;                 // LEDs startup pattern


  PORTA = ~portA;               // write out



  portA = 0x00;                 // clear shadow register

  // port B: 6: LED
  portB  = 0x00;                // clear shadow register
```

Ansprache mit Übersetzung

```
void init_ports(void)
{
  DDRA = 0xff;                    // portA is output
  8a:   8f ef           ldi       r24, 0xFF       ; 255
  8c:   8a bb           out       0x1a, r24       ; 26
  portA = 0x55;                   // LEDs startup pattern
  8e:   85 e5           ldi       r24, 0x55       ; 85
  90:   80 93 64 00     sts       0x0064, r24
  PORTA = ~portA;                 // write out
  94:   80 91 64 00     lds       r24, 0x0064
  98:   80 95           com       r24
  9a:   8b bb           out       0x1b, r24       ; 27
  portA = 0x00;                   // clear shadow register
  9c:   10 92 64 00     sts       0x0064, r1
  // port B: 6: LED
  portB  = 0x00;                  // clear shadow register
  a0:   10 92 63 00     sts       0x0063, r1
```


GENDER CHANGER GERMANY PAT: 9217544
SPI2 Dongle
Embedit
Prototype
JTAG

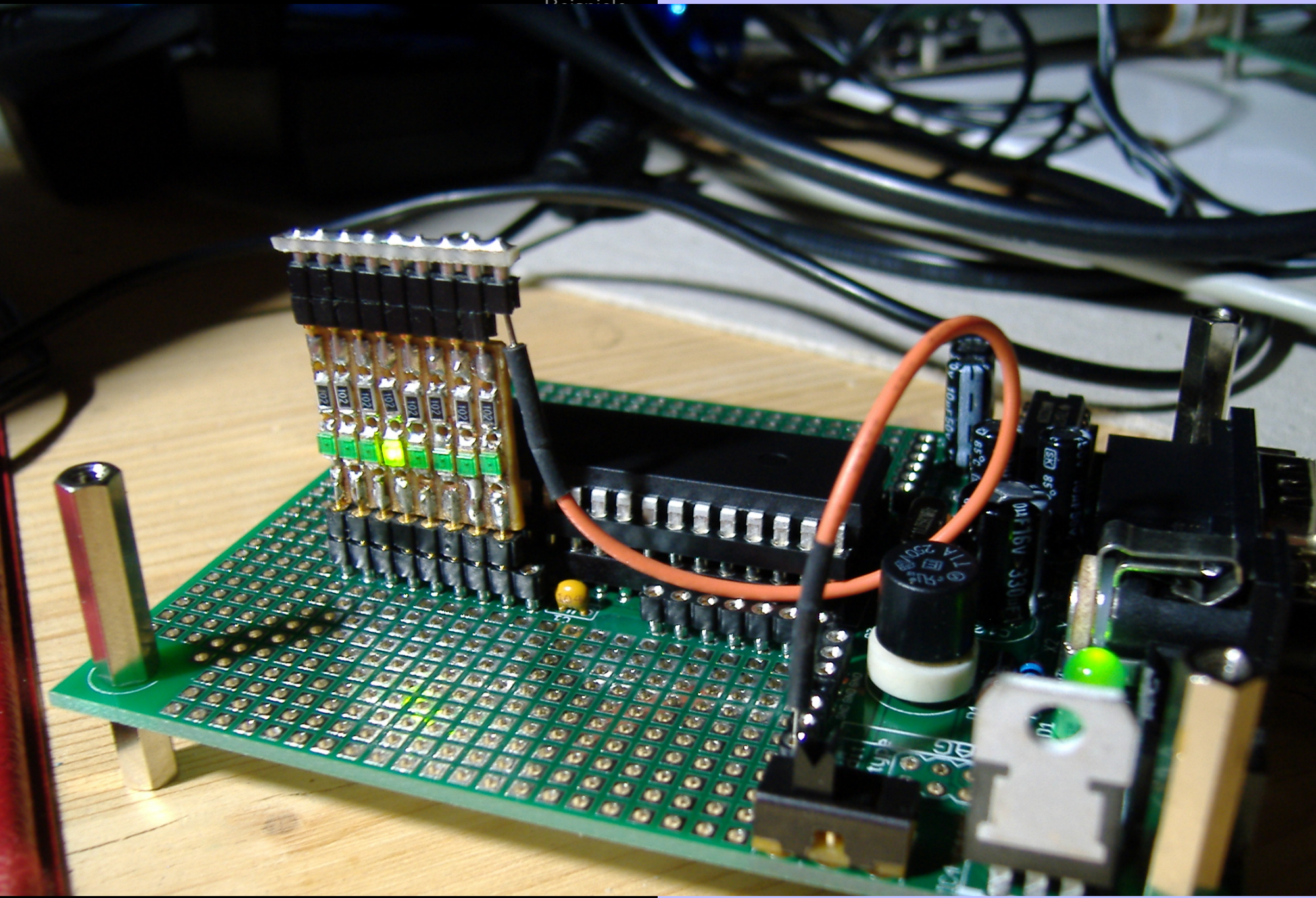
Wie kommt das Programm auf den Kontroller?

```
avrdude -c uisp  -P /dev/parport0 -p atmega32
  -U flash:r:template.hex:i -U eeprom:r:template.eep.hex:i
avrdude: AVR device initialized and ready to accept instructio
Reading | ################################################## |
avrdude: Device signature = 0x1e9502
avrdude: reading flash memory:
Reading | ################################################## |
avrdude: writing output file "template.hex"
avrdude: reading eeprom memory:
Reading | ################################################## |
avrdude: writing output file "template.eep.hex"
avrdude: safemode: Fuses OK
avrdude done.  Thank you.
```

(Sicht)Kontakt

Und wie sieht man jetzt, daß der Kontroller was tut?

- simpel: eine LED anschließen und vom Programm ein-/ausschalten.
- Luxus: ein LCDisplay mit z.B. 2x16 Zeichen
- Luxus: eine serielle oder USB Verbindung und das Programm redet direkt mit mir



Geduld!

Wenn das erste mal eine LED blinkt, hat man das "Hello, world!" Programm der Mikrokontrollerwelt geschafft. Dann braucht man

- **Geduld** und Englischkenntnisse
- **Geduld** und das Datenblatt (mehrere 100 Seiten!)
- **Geduld** und die Assemblerreferenz (nochmal 100 Seiten)
- **Geduld** und starke Nerven
- **Geduld** und ein Meßgerät oder Oszi (Luxus!)
- **Geduld** und eine ordentliche Programmiersprache
  meine Wahl: `amforth`

Forth???

Forth wurde ca. anno 1969 von einem gewissen Charles Moore entwickelt. Er sollte eine Steuerungssoftware für ein Teleskop schreiben. Und wahrscheinlich hatte er bald genug von Assembler! Forth ist anders:

- stack basiert (keine Argumentlisten)
- transparent (keine Kapselung oder Namensräume)
- umgekehrt polnish notiert (`2 3 * 4 +`)
- interaktiv (und das auf einem Mikrokontroller!)
- normalerweise ohne Gleitkommazahlen
- erweiterbar (der Programmierer kann den compiler erweitern)
- schnell, es läuft ja quasi direkt in Maschinencode
- in Assembler und Forth geschrieben
- Open Source!

Der Stack

```
gforth
1 2 3 .s <3> 1 2 3  ok
+  ok
.s <2> 1 5  ok
+  ok
.s <1> 6  ok
. 6  ok
.s <0>  ok
  ok
1 2 3 .s <3> 1 2 3  ok
dup .s <4> 1 2 3 3  ok
+ .s <3> 1 2 6  ok
over .s <4> 1 2 6 2  ok
rot .s <4> 1 6 2 2  ok
* .s <3> 1 6 4  ok
+ + . 11  ok
.s <0>  ok
```

Demo!

1. ein amforth system laden
2. amforth ist interaktiv:
   - man kann Portpins direkt auf Ein–/Ausgabe schalten
   - man kann alle Register des Kontrollers lesen und beschreiben
   - man kann interaktiv neue Funktionen definieren
   - die überleben auch das Ausschalten!
3. Struktur eines Mikrokontrollerprogramms
   1. **alles** initialisieren: Variablen, Ein–/AusgabePins, Zähler, serielle Schnittstelle, Interrupts, Datenobjekte . . .
   2. dann eine endlose Schleife drehen, die die Arbeit macht

interaktiv!

Mit C geht das immer so:

1. Datei editieren und speichern
2. Programm übersetzen und assemblierern
3. Kontroller löschen und neu beschreiben
4. beten und einschalten

mit Forth kann man sich die Schritte 2 und 3 meistens sparen. Man kann die neuen Funktionen direkt in den Kontroller schreiben und sofort benutzen. Erst wenn sich's lohnt schreibt man das in eine Datei, die dann auf den Kontroller geladen wird — nur das Programm, nicht das Forth System selbst. Das muß man nur selten neu laden.

01_hello

- Pin 3 an PortB wird als `led1` deklariert
- der `led1` Pin wird auf Ausgabe gestellt
- der `led1` Pin wird auf `low` gestellt damit die LED leuchtet
- In der Schleife wird led1 200 ms lang eingeschalten und 800 ms aus.
- Die Schleife beendet sich, wenn man eine Taste drückt.

02_timer

Zusätzlich zu `01_hello` wird

- der `timer2` eingestellt
- er wird von einem zusätzlichen Uhrenquarz gesteuert
- `timer2` läuft mit $f_{clock}/256 = 128\,\mathrm{Hz}$
- In der Schleife wird bei jedem Durchgang der Zähler ausgelesen und mit dem Wert vom letzten Durchgang verglichen. Die Differenz wird ausgegeben und markiert, falls sie nicht 128 ist.
- die Schleife läuft etwas langsamer als geplant, weil die reine Wartezeit eine Sekunde beträgt.

03_adc

Zusätzlich zu `01_hello` wird

- der *Analog Digital Converter* eingestellt.
- in der Schleife wird der Spannungswert an Pin PortA.0 ausgelesen.
- Der Spannunswert kann mit einem Poti verändert werden.

04_i2c

01_hello wird nur als Gerüst verwendet.

- es wird die *Two Wire Interface*–Einheit initialisiert
- TWI und I2C ist ein Bussystem
- damit kann man andere Bausteine gezielt ansprechen
- zum Lesen z.B. Temperatursensor, Speicher, Uhr
- zum Schreiben, z.B. Speicher, Uhr, Schaltausgänge

05_timeup

Das ist ein vollständiges Programmgerüst mit

- Uhr und Kalender (verwendet `timer2`)
- Regelmäßigen Jobs pro tick, Sekunde, Minute, Stunde ...

Dieses Grundgerüst verwende ich bei meiner Wetterdatenerfassung.

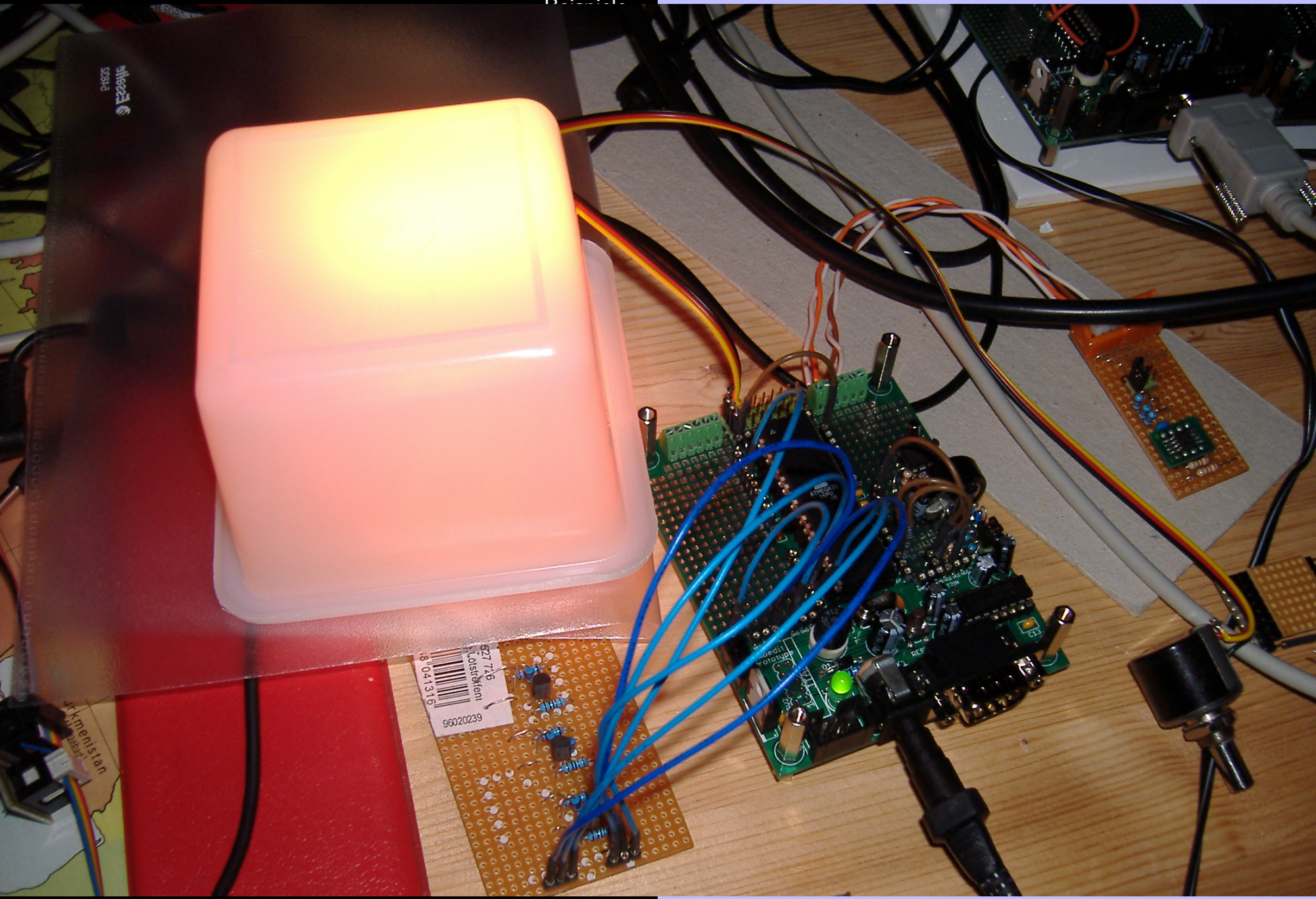
06_pwm

- es wird `timer0` als Signalgenerator verwendet
- das Signal ist eine Rechteckspannung mit einstellbarem Puls–/Pausen–verhältnis
- das Verhältnis wird langsam von 1 auf 100 % erhöht
- damit kann ich die Helligkeit einer LED steuern
- damit kann man auch eine Schaltstufe betreiben, die einen Motor steuert.

07_pwm

Abweichend von 06_pwm

- wird hier die Helligkeit mit dem Potentiometer eingestellt.


Lötstäfeli
96020239

08_fnordlicht

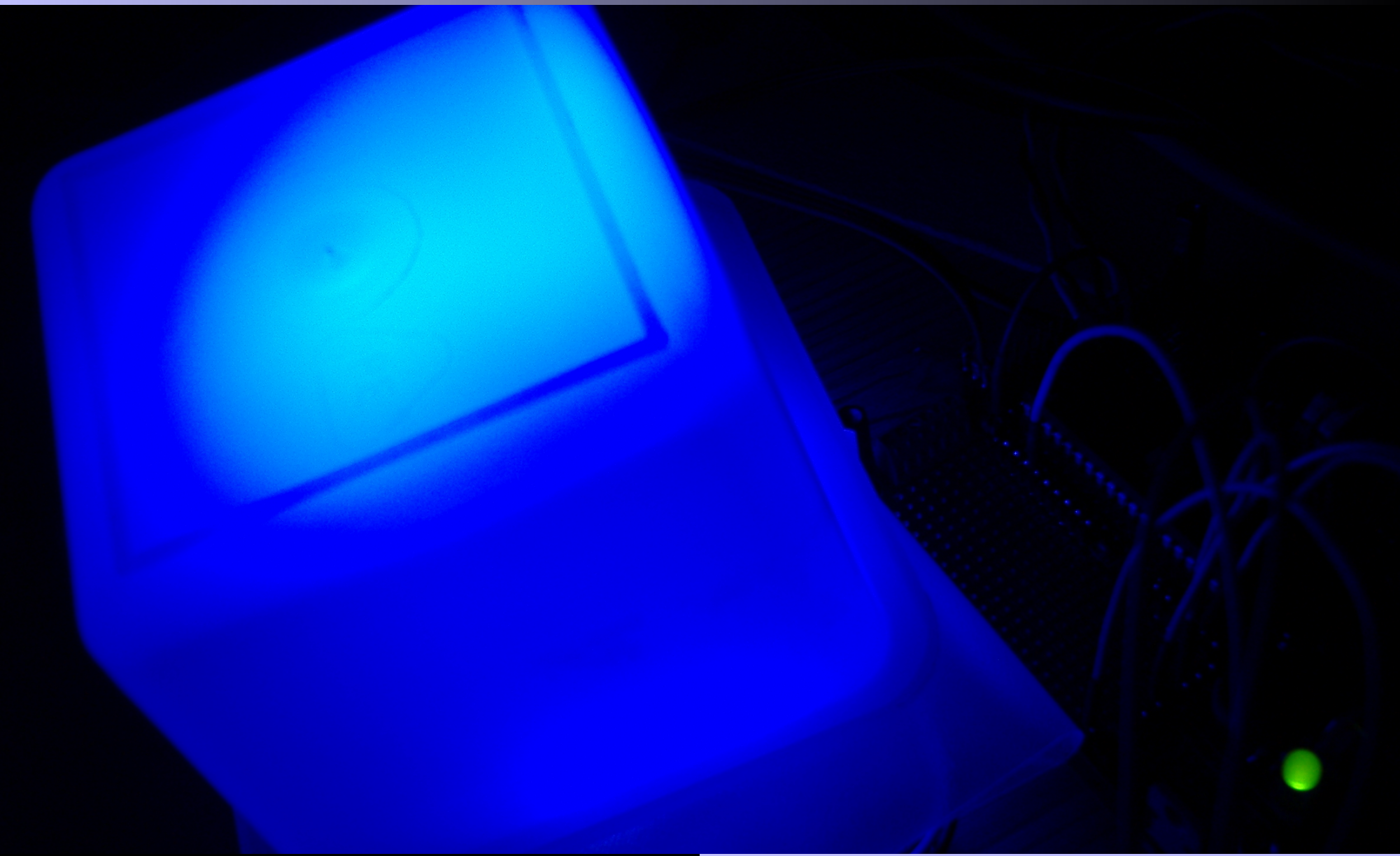
Das *Fnordlicht* ist eine Lampe mit roten, grünen und blauen LEDs. Damit kann man jede erdenkliche Farbe zusammenmischen.

- `timer0` steuert ein PWM–Signal für den blauen Kanal
- `timer1` steuert zwei unabhängige PWM–Signale für den roten und den grünen Kanal
- die Pulszeit kann in 256 Schritten eingestellt werden
- das Rechtecksignal läuft mit einer Frequenz von $f_{cpu}/(8 * 256) = 5400\,\mathrm{Hz}$
- Die Farbe wird jede Sekunde verändert, der Zyklus durchläuft alle Farben und wiederholt sich nach 256 Sekunden
- HSV Farbmodell, wird in RGB umgerechnet (mit $S = V = 255$)

Links

- de.wikipedia.org/wiki/Forth˙(Informatik)
- en.wikipedia.org/wiki/Forth˙(programming˙language)
- Leo Brodie — Starting Forth
  www.forth.com/starting-forth/index.html
- Leo Brodie — Thinking Forth
  thinking-forth.sourceforge.net
- Steven Pelc — Forth Handbook
- Forth Gesellschaft e.V. www.forth-ev.de
- www.atmel.com
- www.mikrocontroller.net
- www.avrfreaks.net
- www.roboternetz.de

Noch Fragen?



Danke!