

VHDL - DNF - 3 Input Var - TTL - 74xx - ...

David Vajda

29. Januar 2026

1 quine mc cluskey

wahrheitstabelle...

(C) David Vajda

Mon Jan 26 16:54:49 2026

3 Network - TTL - Disjunktive Normalform

	x2	x1	x0	y
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

	x2	x1	x0	y
0	0	0	0	1
5	1	0	1	1

verkuerzung nicht moeglich!!!

	x2	x1	x0	y
Gruppe 0:				
0	0	0	0	1
Gruppe 2:				
5	1	0	1	1

ok:

$y \leq (\text{not } x2 \text{ and not } x1 \text{ and not } x0) \text{ or } (x2 \text{ and not } x1 \text{ and } x0);$

dies war DNF

und nun KNF:

KNF nach muster:

- 1.) Vertausche AND durch OR
- 2.) OR durch AND
- 3.) Invertiere alle Eingangsvariablen

$y \leq (x_2 \text{ or } x_1 \text{ or } x_0) \text{ and}$
 $(\text{not } x_2 \text{ or } x_1 \text{ or } \text{not } x_0);$

Minterm: m

Maxterm: M

Minterm: m₁, m₂, m₃, ...

Maxterm: M₁, M₂, M₃, ...

Ein Minterm besteht aus konjunktiv verknuepften eingangsvariablen

Ein Maxterm ...

Eine DNF ist eine Disjunktive Verknuepfuug lauter Minterm

Ein Minterm besteht aus konjunktiv verknuepften eingangsvariablen

m₁ := (not x₂ and not x₁ and not x₀);

m₂ := (x₂ and not x₁ and x₀);

DNF := m₁ or m₂;

bei der syntax arithmetischer ausdruecke wiederum gilt:

expr ::= expr + term | term
term ::= term * factor | factor
factor ::= const | id | (expr)

fuer die abarbeitung selber ist die beseitigung der linksrekursion
durch ergaenzung der grammatikalischen beschreibung sprache (grammatik)
notwendig. dies ist allerdings die grammatikalische definition.
schon bereits beim rechnen lernt man das, doch wie kann man sich derartige
ausdruecke ernsthaft erklaren?

5, x, 10, a, b, ...: sind faktoren

5*x, 6*x³, ...: sind terme

doch wieso unterscheiden sie sich massgeblich von ausdruecken?

5*x = -7*x³

5*x + 7*x³ = 0

dies ist wiederum eine gleichung. aber

$5*x + 7*x^3$
ist ein ausdruck und unterscheidet sich von:
 $5*x + 7*x^3$
kann sein ich liege mit letzterem falsch.

das oben stimmt in jedem fall.

ok, das war nur ein einschub...

warum, weil ich die aufgabe nachher erweitere, aus welchem grund?

ganz einfach:

ich schreibe so einen parser-compiler-generator ... what ever
aber ohne lexer,
erzeugt ausdruecke, boolean...

nur: ein funktionsplotter ist schnell und einfach gemacht.
wenn sie ueber das vorstellungsvermoegen bei pixeln verfuegen...
sie finden php funktionsplotter - jetzt gzippt auf homepage
frueher zum anwenden...
es ist einfach einen plotter zu schreiben, wenn man die werte hat...
man nimmt die hoehe des bildes. dieses setzt man dem maximalwert gleich
dadurch rechnet man den aus. oder legt ihn vorher fest.

wie auch immer. da der maximalwert ... z.B. der eigentliche wert ist,
muessen sie den rest nur in bezug dazu sehen. von rec'htsschaendig, abszisse
x-achse, einfach punkt fuer punkt einsetzen.

wie erhalte ich werte bei komplizierten ausdruecken?

parsing. variablen merken und im nicht sichtbaeren speicher speichern.
entsprechend den wert einsetzen.
wenn passiert...

ergibt sich die gerade. punkt fuer punkt. wdh.

nur: bei ausdruecken boolean, waere es moeglich einen ausdruck zu erzeugen.
der parser erstellt eine Wahrheitstabelle, wie gerade eben.
er liest variablen ...

also, ein zweiter parser

ein compiler, hat 3 teile

- 1.) lexer - bzw. das, was die eingabe entgegennimmt token erzeugt.
- 2.) parser
- 3.) zielsprache

doch der parser wird durch token gesteuert.

dabei trifft er allerdings entscheidungen.
also, kann man an den stellen zufallsvariablen nehmen
anstatt die entscheidungen im zu lesenden text zu finden,
macht er sie so, erzeugt eine sprache, allerdings umgekehrt.

eine moeglichkeit ist naemlich:

- 1.) quellsprache => lexer - bzw. das, was die eingabe entgegennimmt token erzeugt.
- 2.) parser
- 3.) zielsprache => eigentlich identische kopie der quellsprache

die sprache die erzeugt wird ist die quellsprache, aber...

da ich den parser auch, mit zufallsvariablen gesteuert aendern koennte.

- 1.) parser aehnliches, steuern ueber zufall + abbruchkriterium
- 2.) zielsprache => eigentlich identische kopie der quellsprache

so.

nachdem ausdruecke nun da sind, die aber nicht DNF sind
mache ich mir nicht die muehe der umwandlung.
pech habe ich, wenn schaltwerke dargestellt werden.

wenn nicht: dann entsteht auf jeden fall eine DNF
daraus kann ich eine wahrheitstabelle machen. eigentlich auch
beim schaltwerk.

gut: wie kommt die wahrheitstbaelle zu stande. wie beim
funktionsplotter. ich setze einfach in alle variablen
boolsche werte ein. ist das geschehen ist sie da...

und jetzt habe ich eine normale wahrheitstabelle mit DNF

...

ok, die uebung kommt spaeter auch.
so: wie ich probiere spaeter eine DNF mit quine mc cluskey
aber verkuerzt zu:

4 Bit Parallel Addierer zu machen.

wie?

ich nehme ein Byte: das sind Operanden: (Summanden) A und B
ich nehme 4 Bit: das ist der Zieloperand: Die Summe C

ich habe eine Wahrheitstabelle mit 8 Eingangsvariablen, macht $2^8 = 256$ Zeilen, dazu aber 4 Bit Ausgangsvariablen
also muss sie: $4 \cdot 256 = 1024$ mal aufgestellt werden. das dauert sehr lange.
quine mc cluskey funktioniert Ausgangsvariable fuer Ausgangsvariable.

das liesse sich mit einem computerprogramm und algo aber einfach vollziehen.
man hofft zu einer sehr DNF und einfach artigen form - schlimmsten falls
der selben wie ripple carry chain adder, aber da sind XOR drin, zu kommen,
besten falls einfacher...

ok, jetzt weiter...

vhdl: sequenz:

```
-- (c) david vajda  
-- dnf - ghdl - 74/54/84xx will follow ...  
-- 01/27/2026
```

```
import ieee;  
use ieee.std_logic_1169.all;
```

```
entity min1_01272026 is  
port (  
  x2, x1, x0: in std_logic;  
  m1: out std_logic  
);  
end;
```

```
architecture behaviour of min1_01272026 is  
begin  
  m1 <= (x2 or x1 or x0);  
end;
```

```
import ieee;  
use ieee.std_logic_1169.all;
```

```
entity min2_01272026 is  
port (  
  x2, x1, x0: in std_logic;  
  m2: out std_logic  
);  
end;
```

```
architecture behaviour of min2_01272026 is  
begin  
  m2 <= (not x2 or x1 or not x0);  
end;
```

```
import ieee;
```

```

use ieee.std_logic_1169.all;

entity dnf01272026 is
port (
x2, x1, x0: in std_logic;
y: out std_logic
);
end;

architecture behaviour of dnf01272026 is
component min1_01272026 is
port (
x2, x1, x0: in std_logic;
m1: out_std_logic
);
end component;
component min2_01272026 is
port (
x2, x1, x0: in std_logic;
m2: out_std_logic
);
end component;
signal m1, m2: std_logic;
begin
min1: PORT MAP dnf01272026 (x2=>x2, x1=>x1, x0=>x0, m1=>m1);
min2: PORT MAP dnf01272026 (x2=>x2, x1=>x1, x0=>x0, m2=>m2);
y <= m1 and m2;
end;

import ieee;
use ieee.std_logic_1169.all;

entity quine01272026 is
port (
x2, x1, x0: in std_logic;
y: out std_logic
);
end;

architecture behaviour of quine01272026 is
component dnf01272026
port (
x2, x1, x0: in std_logic;
y: out std_logic
)
end component;
signal x2, x1, x0: std_logic;
begin
dnf: PORT MAP dnf01272026 (x2=>x2, x1=>x1, x0=>x0, y=>y);

```

```
-- (c) david vajda
-- dnf - ghdl - 74/54/84xx will follow ...
-- 01/27/2026
```

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity min1_01272026 is
port (
x2, x1, x0: in std_logic;
m1: out std_logic
);
end;
```

```
architecture behaviour of min1_01272026 is
begin
m1 <= (not x2 and not x1 and not x0);
end;
```

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity min2_01272026 is
port (
x2, x1, x0: in std_logic;
m2: out std_logic
);
end;
```

```
architecture behaviour of min2_01272026 is
begin
m2 <= (x2 and not x1 and x0);
end;
```

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity dnf01272026 is
port (
x2, x1, x0: in std_logic;
y: out std_logic
);
end;
```

```
architecture behaviour of dnf01272026 is
component min1_01272026 is
port (
x2, x1, x0: in std_logic;
m1: out std_logic
);
```

```

end component;
component min2_01272026 is
port (
x2, x1, x0: in std_logic;
m2: out std_logic
);
end component;
signal m1, m2: std_logic;
begin
min1: min1_01272026 PORT MAP (x2=>x2, x1=>x1, x0=>x0, m1=>m1);
min2: min2_01272026 PORT MAP (x2=>x2, x1=>x1, x0=>x0, m2=>m2);
y <= m1 or m2;
end;

library ieee;
use ieee.std_logic_1164.all;

entity quine01272026 is
port (
y: out std_logic
);
end;

architecture behaviour of quine01272026 is
component dnf01272026
port (
x2, x1, x0: in std_logic;
y: out std_logic
);
end component;
signal x2, x1, x0: std_logic;
begin
dnf: dnf01272026 PORT MAP (x2=>x2, x1=>x1, x0=>x0, y=>y);

```

1.1 Der Code zur generierung der Tabelle

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main (void) {
    int x0, x1, x2;
    time_t t;
    int i;
    struct tm tm = *localtime(&(time_t){time(NULL)});

```



```

m2: out std_logic
);
end;

architecture behaviour of min2_01272026 is
begin
m2 <= (x2 and not x1 and x0);
end;

library ieee;
use ieee.std_logic_1164.all;

entity dnf01272026 is
port (
x2, x1, x0: in std_logic;
y: out std_logic
);
end;

architecture behaviour of dnf01272026 is
component min1_01272026 is
port (
x2, x1, x0: in std_logic;
m1: out std_logic
);
end component;
component min2_01272026 is
port (
x2, x1, x0: in std_logic;
m2: out std_logic
);
end component;
signal m1, m2: std_logic;
begin
min1: min1_01272026 PORT MAP (x2=>x2, x1=>x1, x0=>x0, m1=>m1);
min2: min2_01272026 PORT MAP (x2=>x2, x1=>x1, x0=>x0, m2=>m2);
y <= m1 or m2;
end;

library ieee;
use ieee.std_logic_1164.all;

entity quine01272026 is
port (
y: out std_logic
);
end;

architecture behaviour of quine01272026 is

```



```

component dnf01272026
port (
x2, x1, x0: in std_logic;
y: out std_logic
);
end component;
signal x2, x1, x0: std_logic;
begin
dnf: dnf01272026 PORT MAP (x2=>x2, x1=>x1, x0=>x0, y=>y);
x0 <= '0' after 0 ns, '1' after 10 ns, '0' after 20 ns, '1' after 30 ns, '0' after 40 ns,

x1 <= '0' after 0 ns, '0' after 10 ns, '1' after 20 ns, '1' after 30 ns, '0' after 40 ns,

x2 <= '0' after 0 ns, '0' after 10 ns, '0' after 20 ns, '0' after 30 ns, '1' after 40 ns,
end;

```

2.1 Das hinzufuegen der eingangsvariablen,generieren,...

```

#include <stdio.h>

int main (void) {
    int i;
    int j;
    int k, l;
    int v = 0;
    char *a [] = {"x0", "x1", "x2"};

    for (k = 0, l = 1; k < 3; k++, l = l*2) {
        printf ("%s <= ", a [k]);
        for (j = 1, i = 0, v = 0; i < 80; i+=10, j++) {
            printf ("'%i' after %i ns, ", v, i);
            if ((j % l) == 0) {
                v = (v + 1) % 2;
            }
        }
        j = j%2;
        printf ("'%i' after %i ns;\n\n", v, i);
    }

    return 0;
}

```

übersetzen mit

```

user@host:~$ gcc generatetestbench3.c -o generatetestbench3
user@host:~$ ./generatetestbench3 >> quine01272026.vhdl
user@host:~$ echo "end;" >> quine01272026.vhdl

```

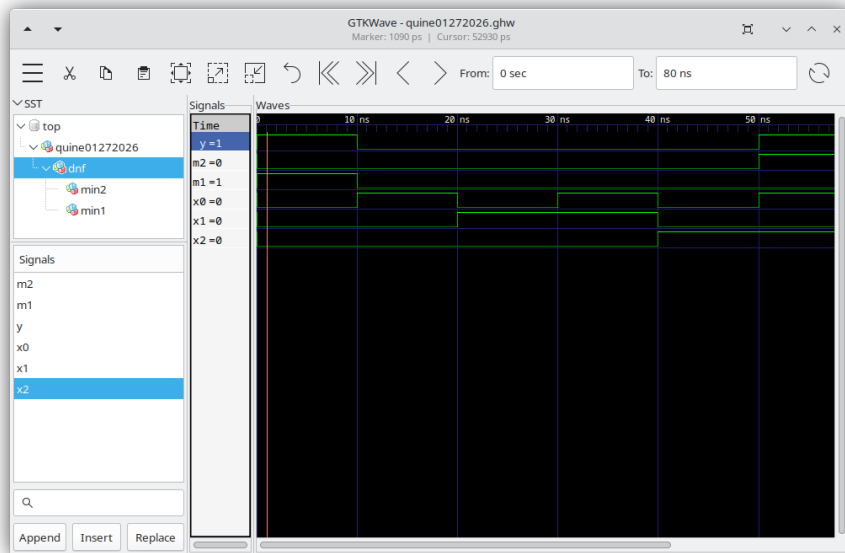
2.2 mit ghdl, erzeugtes Balkendiagramm

vhdl code - uebersetzen mit linux programm ghdl

```

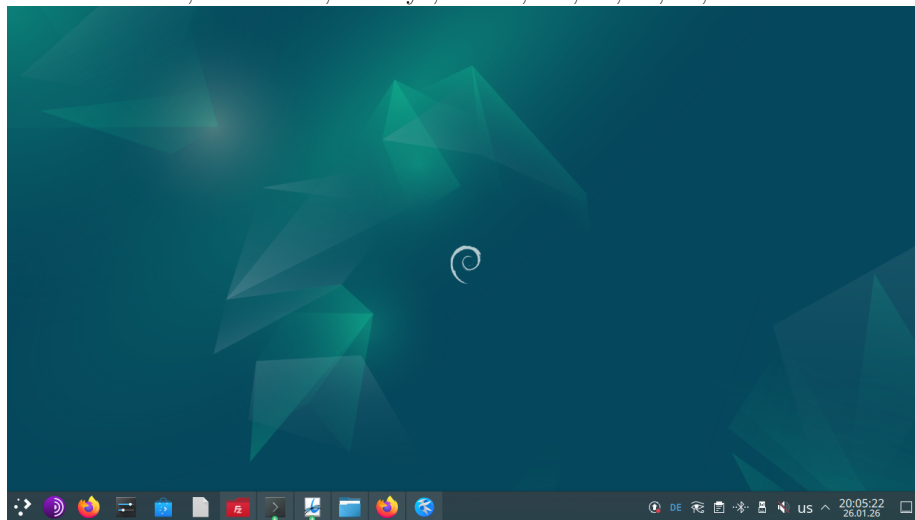
user@host:~$ ghdl -a quine01272026.vhdl
user@host:~$ ghdl -r quine01272026 --wave=wave01272026.ghw
user@host:~$ gtkwave wave01272026.ghw

```



2.2.1 installation der software

use debian 12.0, bookworm, bullseye, buster, . . . , 10, 11, 12, . . .

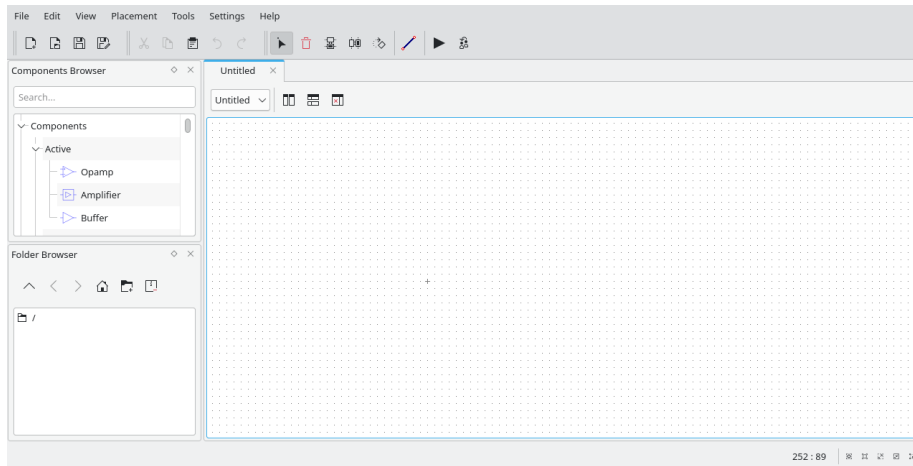


as root super user, run:

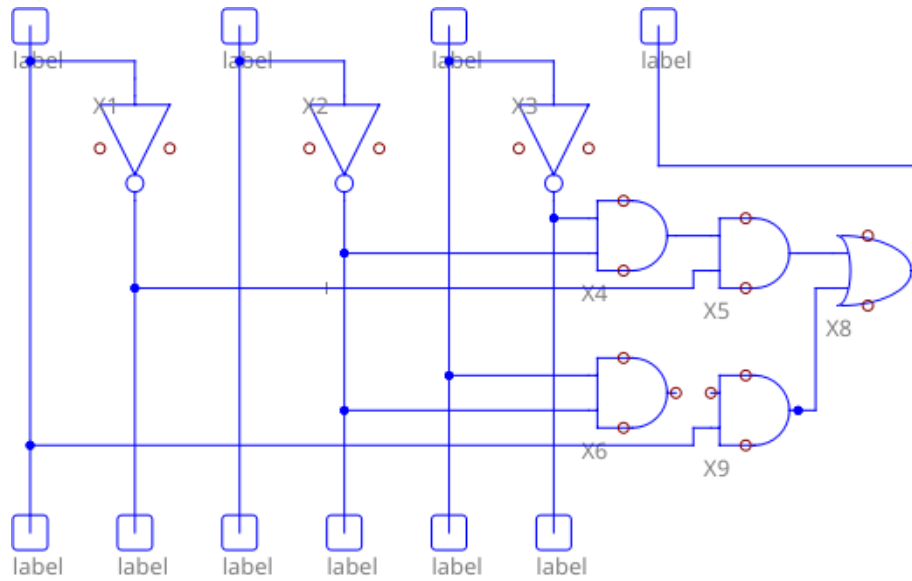
```
root@host:/home/root# apt-get install gtkwave ghdl
```

3 layout

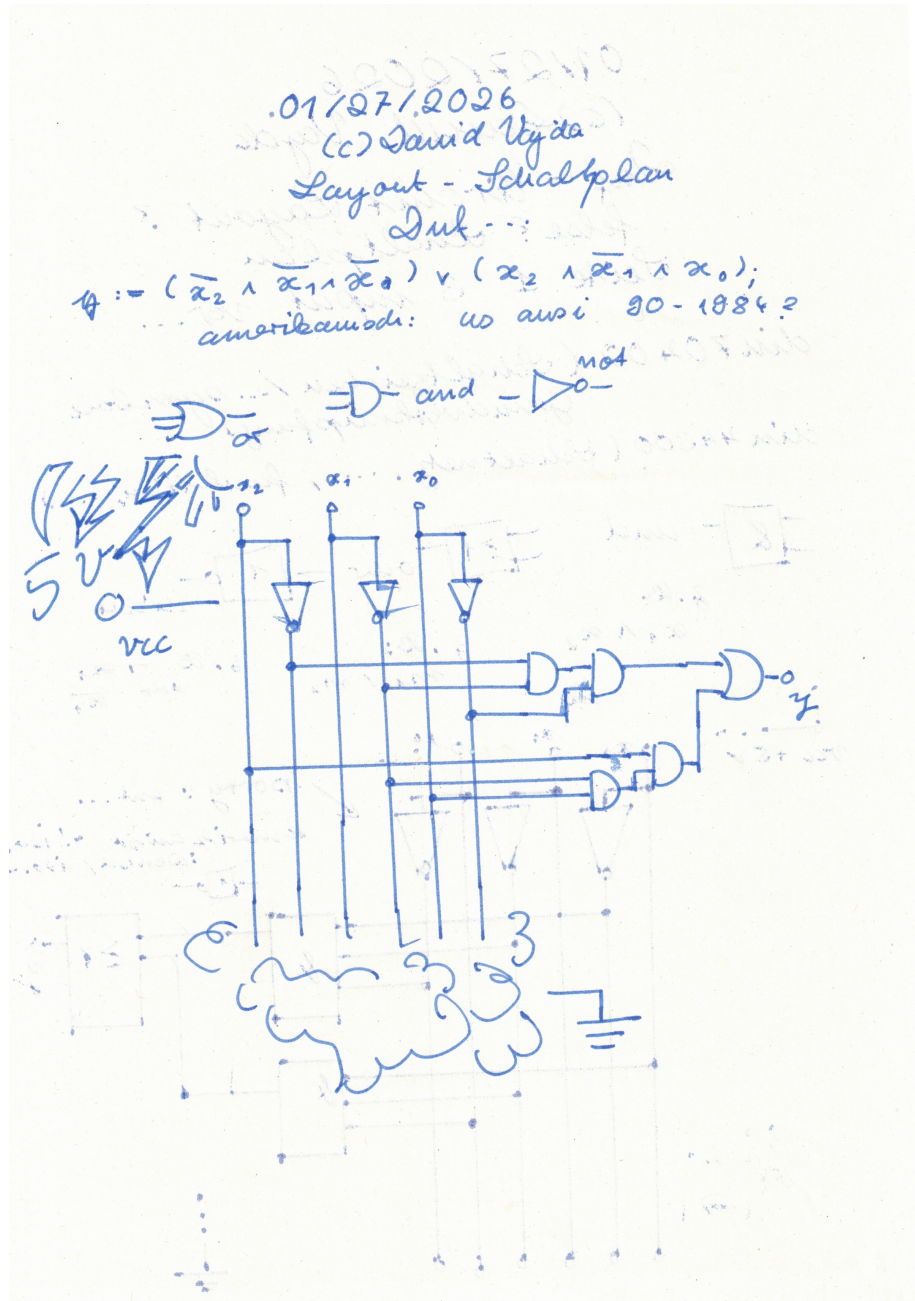
3.1 with linux caneda, draw the circuit



in this case like this for example



3.2 per hand



01/27/2006

(c) David Vajda
für alle ... infos...
Seite 3

Led:

- 1.) anode
- 2.) kathode

rot: 2.3 V z.B. Spannung

$$R = \frac{U}{I}$$

$$U = R \cdot I \text{ ohm'sches gesetz}$$

Digital: Betriebsspannung: $U_B: 5V...$

Diode 20 mA

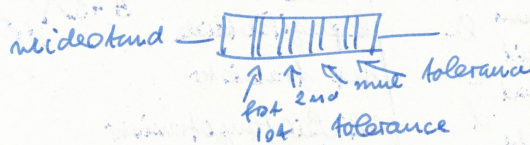
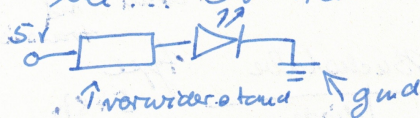
U in V (Volt) Ampere (A) für I und
 R_B für R

$\frac{U}{Z \cdot 1}$ $\frac{P}{u \cdot 1}$ oh... Led: Rot, Blau, weiß

weiß/Blau: 3V? Rot: 2V bzw 2,3V

10V Betriebsspannung: vorwiderstand: 1kΩ

bei 5V reichen bei mit 330 Ω... ok

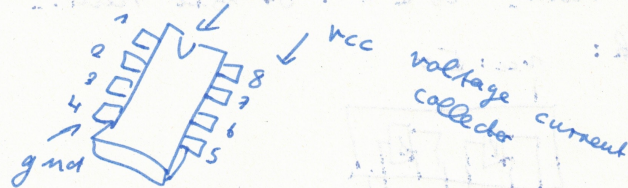


heißt: $22 \cdot 10^4 \text{ ohm}$
und $220 \text{ k}\Omega = 220 \text{ k}\Omega$

schwarz	0
braun	1
rot	2
orange	3
yellow	4
green	5
blue	6
purple	7
grey	8
white	9
gold	:
silver	:

01/07/2026 (K) David Vayda
Seite 5... Bauteile

dip: dual inline package



74xx 54xx 84xx

digital: TTL: bipolar:
Transistor-Transistor
Logic

cmos: hc

74xx: commercial, 84: industrial,
54xx: military...

a: advanced, L: Low power, Y: Schottky,
f: fast

E: Ecmos, H: High Speed, A: TTL compatible
[high speed] cmos

h:
hc
hct...

and x4
or x4

nor: x4

or, xor, not, 1, 2, 3 x

and: 7408
7409
7411
7421

or: 7432

not: inverters: 7404, 7405
7414

xor: =1 ⊕ : 7486
74186

not: 4x, nand: extreme

arith: 83, 85, 181, 182, 183 ... mux, demux,
decoder bcd-2-7 seg ff, monoflop, parallel...

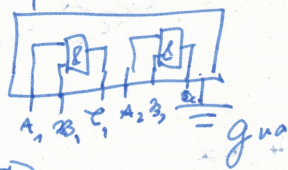
01/27/2026 (c) David Hogue

HEL Logic ... Part 6

parallel serial register

ram: 16 x 4 : 64 Bit ram...

oh: $v_{cc} = 5v$



and $\Rightarrow$

a	a	e
0	0	0
0	1	0
1	0	0
1	1	1

or:

a	a	c
0	0	0
0	1	0
1	0	0
1	1	1

not

a	e
0	1
1	0

oh...

01/27/2026

(C) Klausur Vorge
Inf und Co Ig... , Seite 4

gehäuse: die v. dip...
vorher

iso, iec, ieee, din

normen
ieee international ... electric ...
engineering?

Wie verein deutscher elektriker:

iso wie din in deutschland
din 4

Boolean, electric, schaltungs...
digital...

din 40200 din 40900 din 44300

iso 9600 iso 1164...

bauteil norm:

jedec: japan usa

jis: japan

? : europa →

halbleiter

aa... germanium diode

bc: silicium n-f transistor

1. Buchstabe	2. Buchstabe	type
a: germanium	a	diode
b: silicium silicium	b	kapazitätsdiode
c: galliumarsenid	c	n-f transistor
d: indiumarsenid	d	n-f Leistungs-transistor
e: verbleibendes	e	tunnelediode
f: :	f	h-f Leistungs-transistor
g: :	g	

548
549
30

01/27/2026

(c) David Vajda

Inf - Circuit Layout ?

felse? Schaltplan

Seite 1 3 input ver...

din 70400 (Schaltzeichen /... symbole
Grundverknüpfung)

din 44300 (Schaltnetz, ..., funktions...)

$\boxed{\&}$ und $\boxed{\geq 1}$ oder $\boxed{1}$ nicht

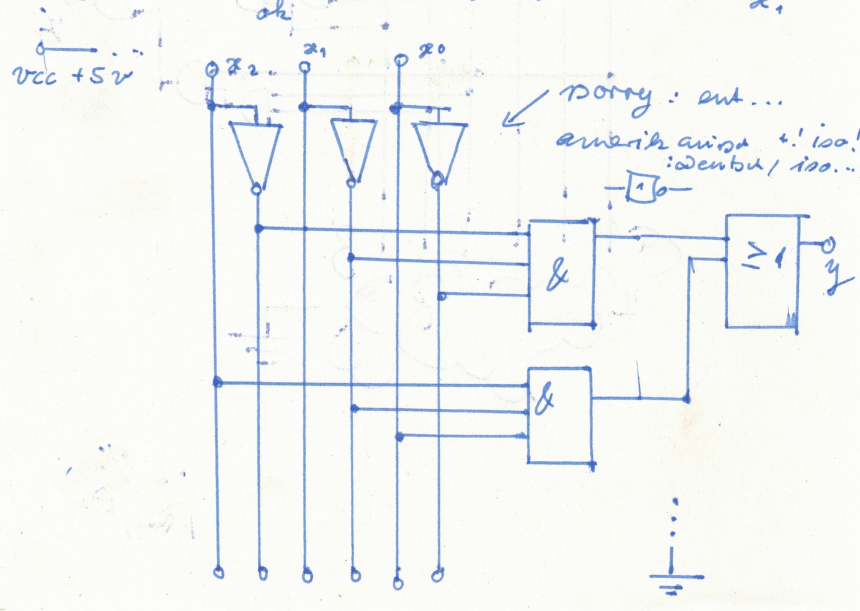
z.B.

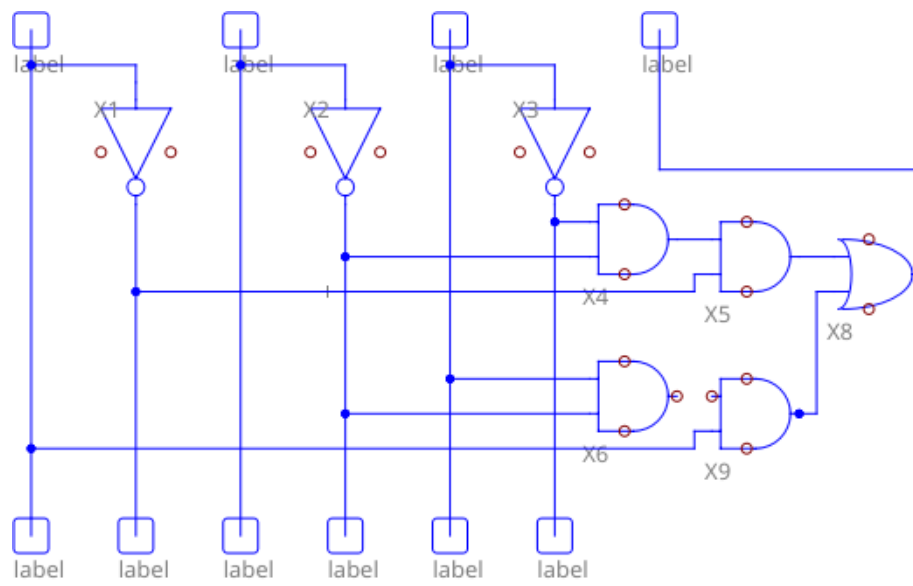
$x_1 \wedge x_2$

z.B.

$m_1 \vee m_2$

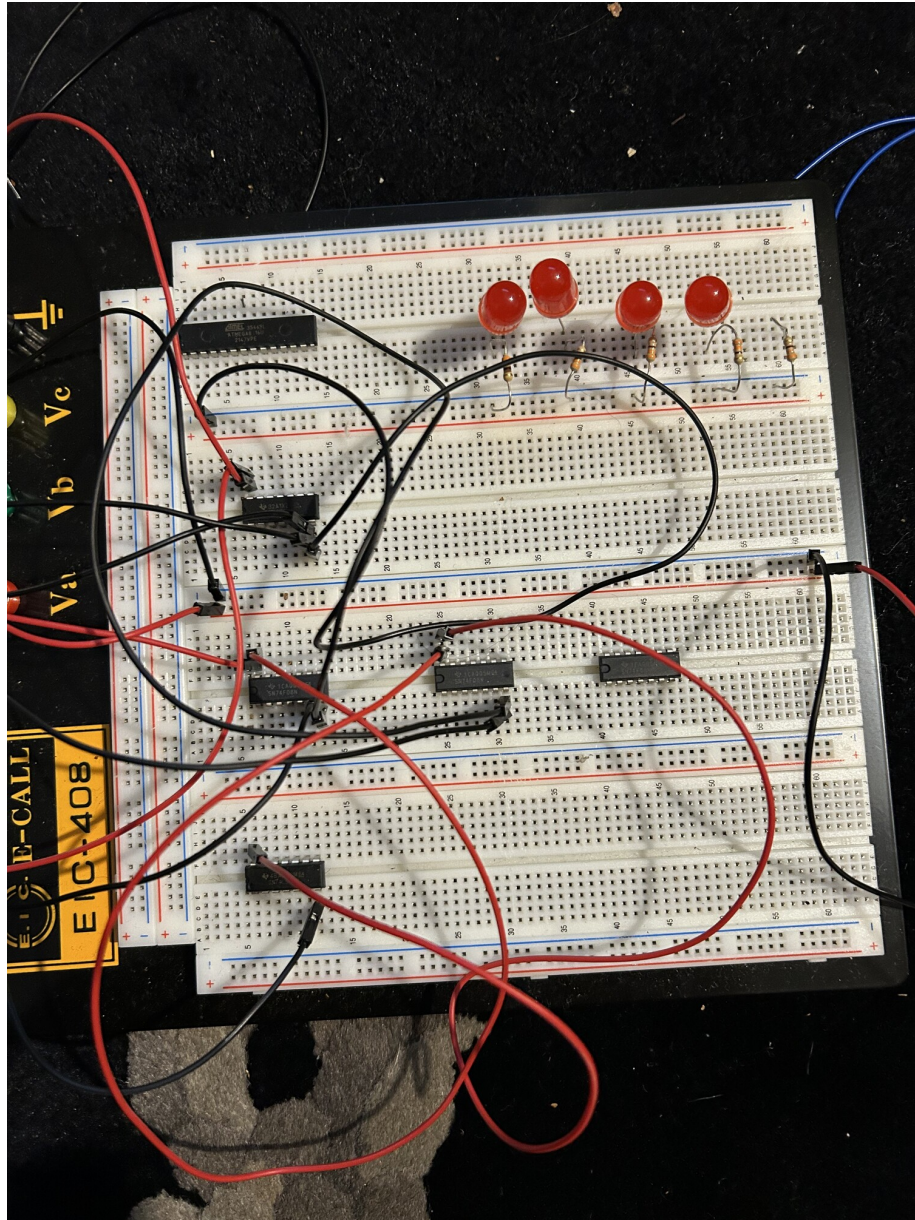
z.B. $\neg x_1$
oder $\overline{x_1}$



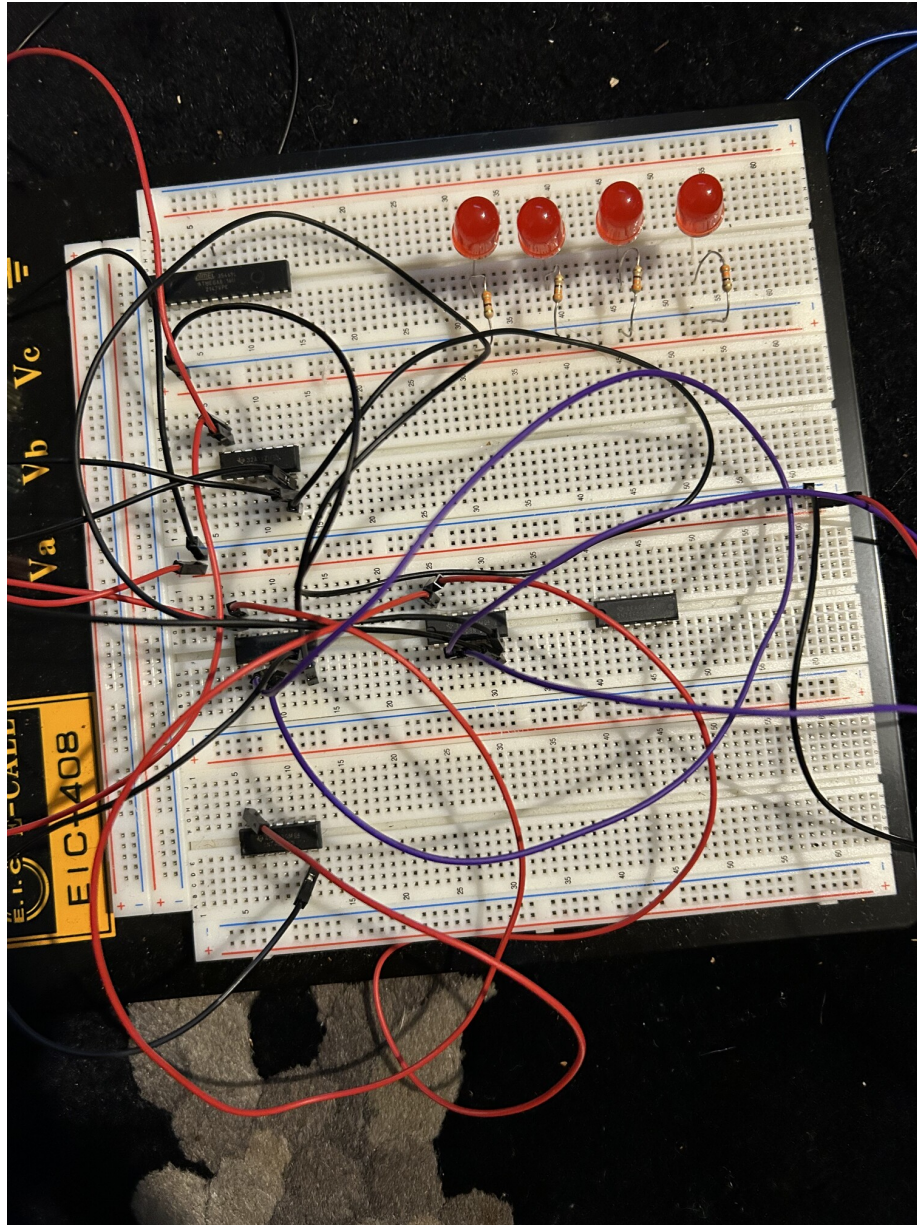


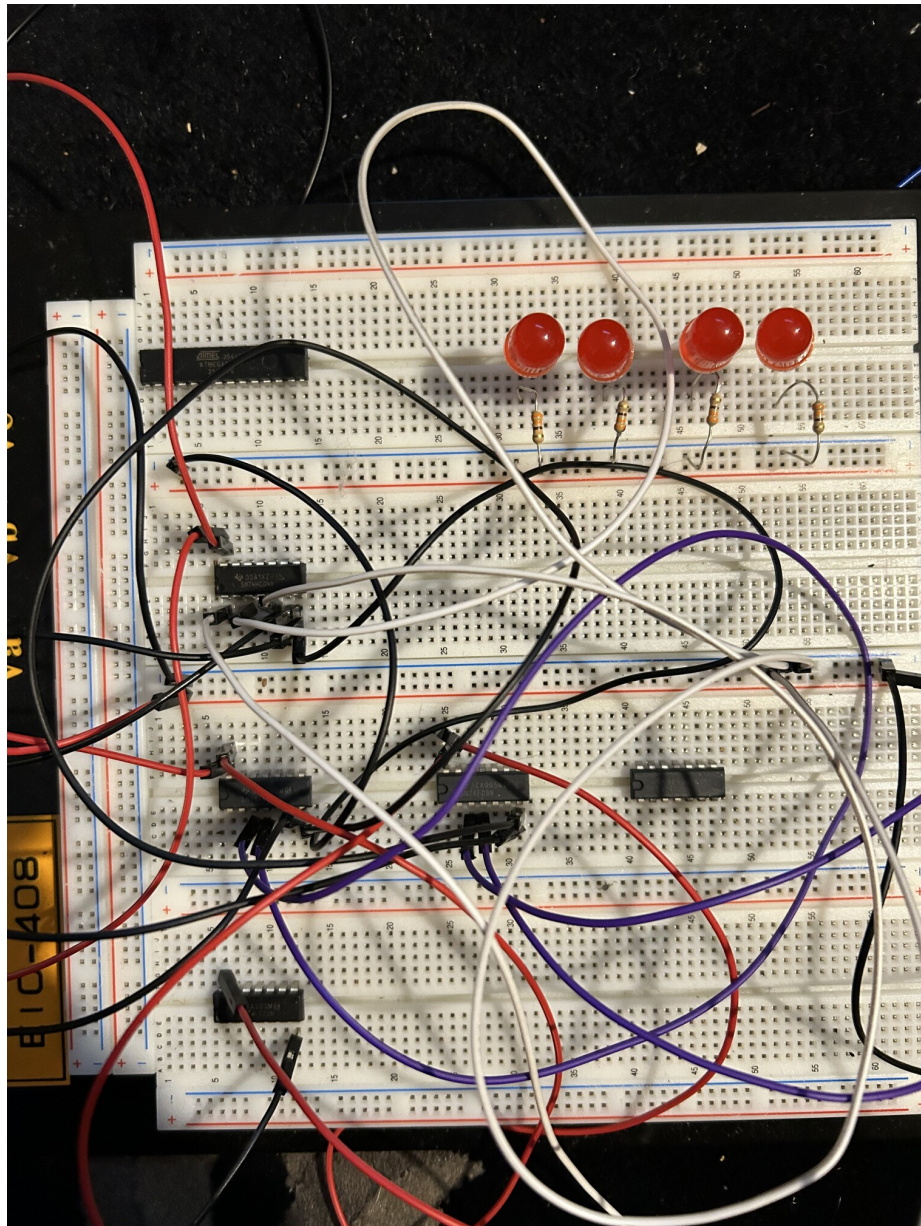
4 Bau auf dem steckboard - breadboard - physisch gatter

4.1 GND und VCC

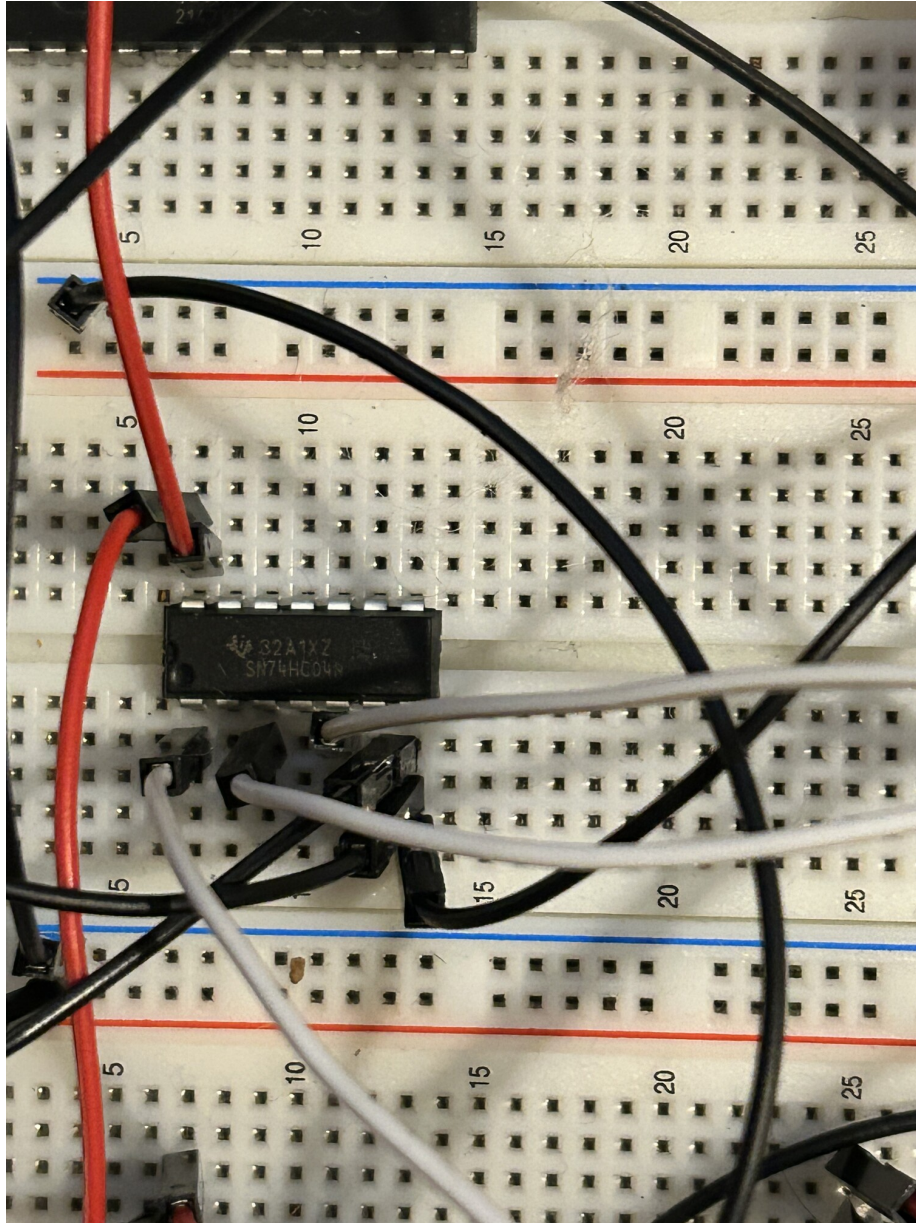


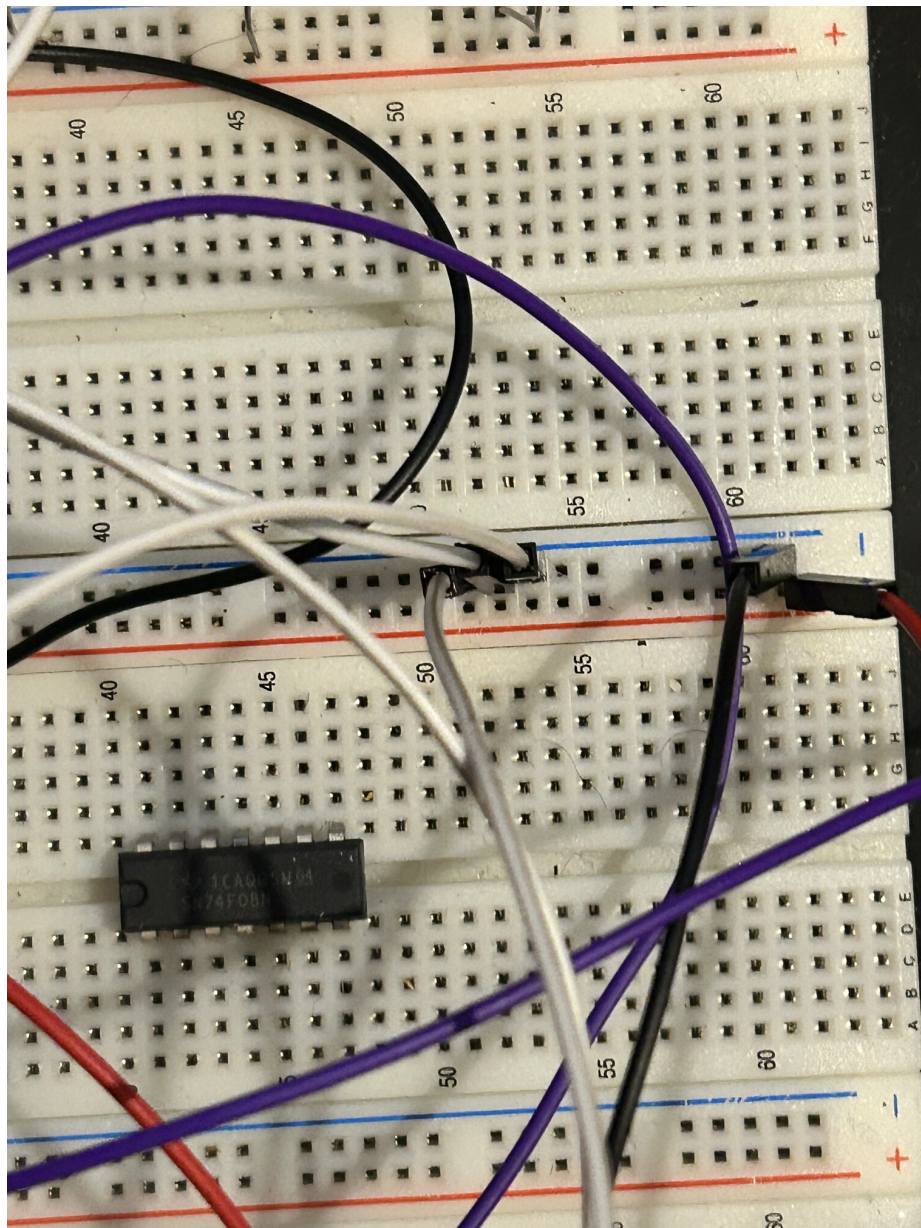
- 4.2 2x AND (7408) mit 3 eingangen, aus 2 AND mit jeweils zwei Eingangen = 4 und 2 Ausgaengen, jeweils, 1 Ausgang



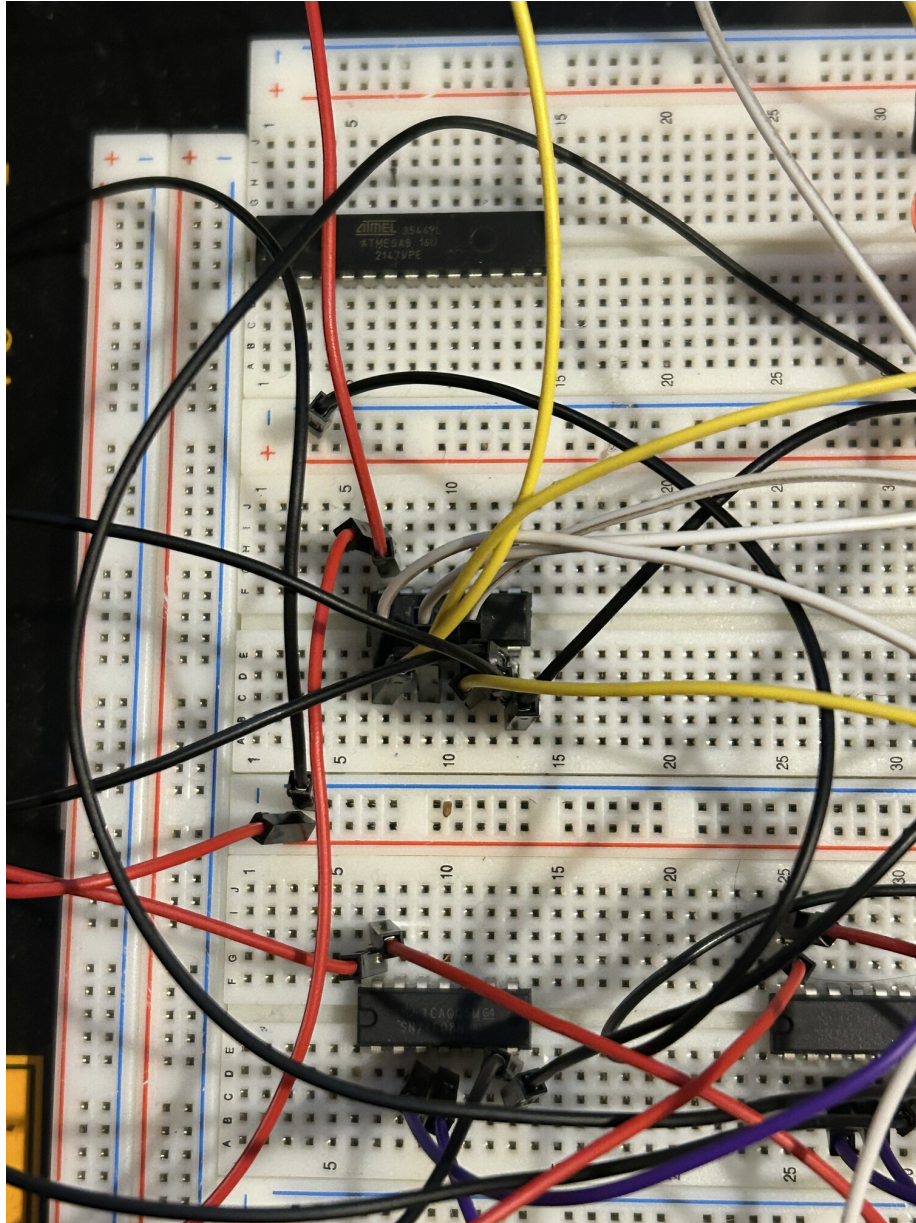


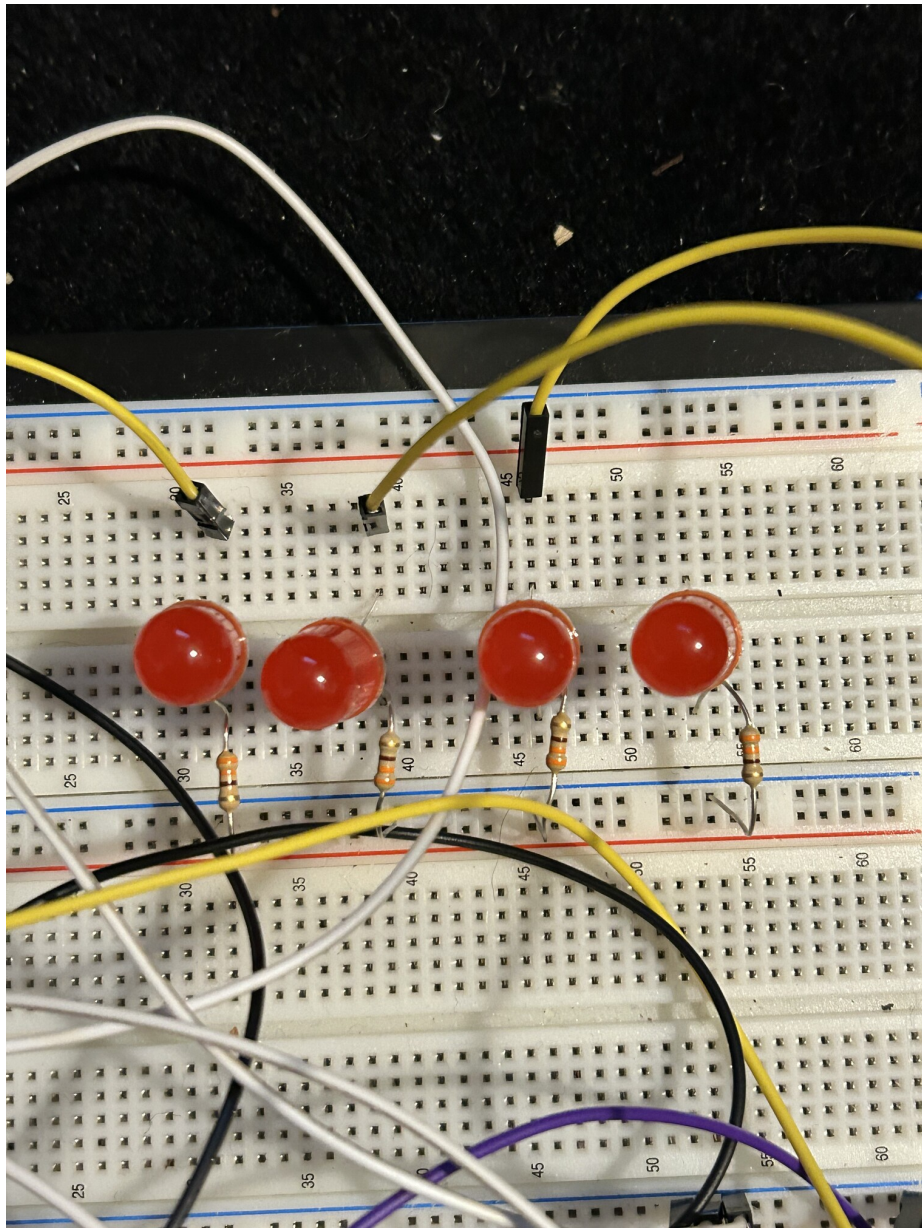
4.3 verbinden der eingangsvariablen $x(2,1,0)$



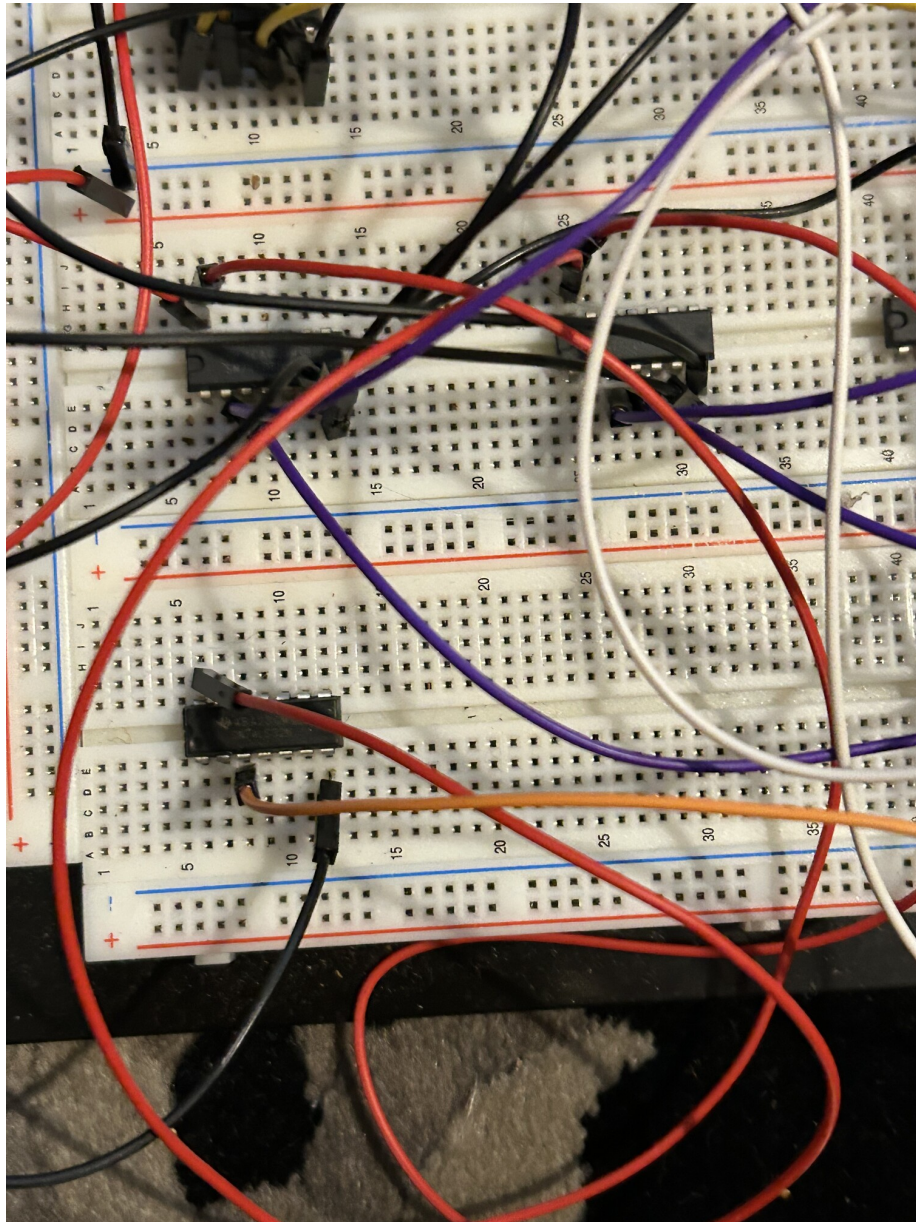


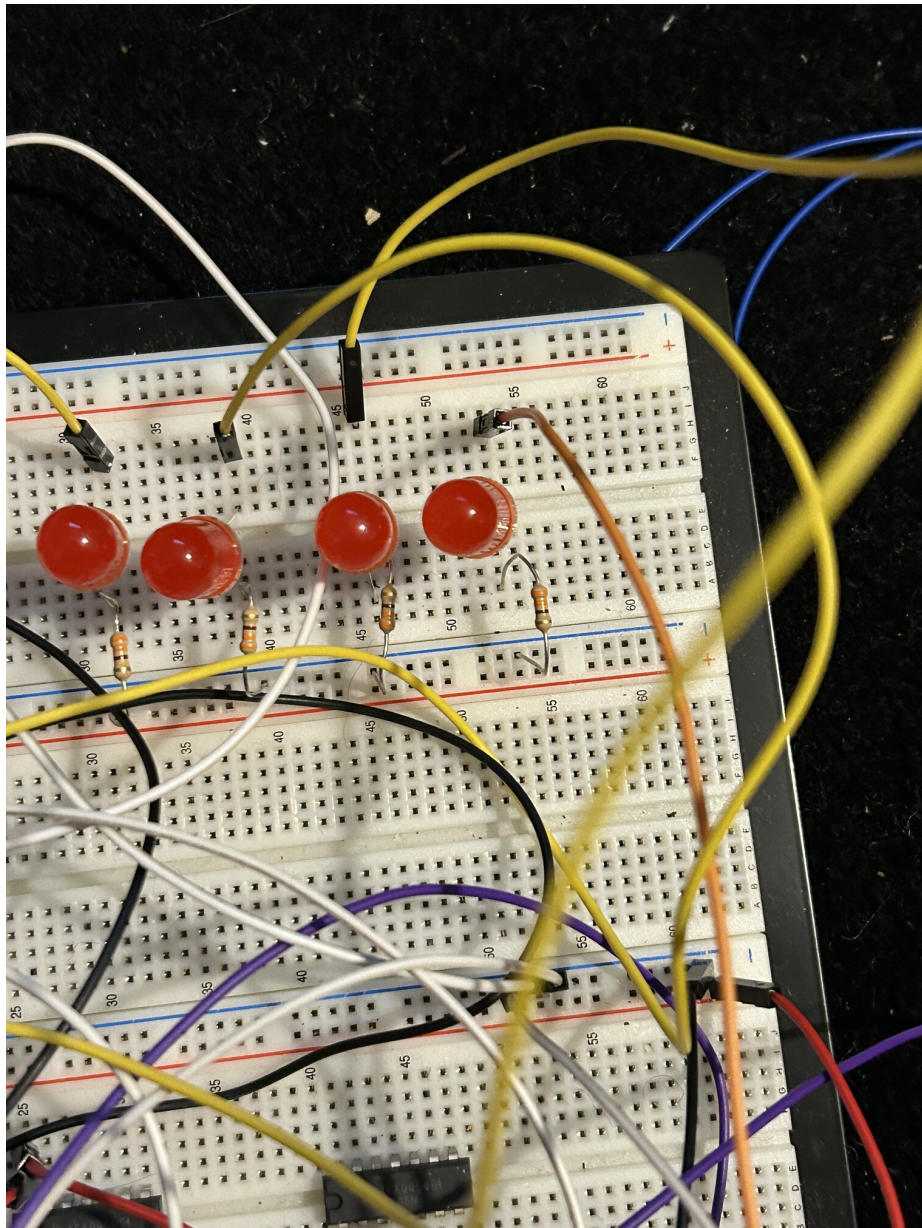
4.4 verbinden der eingangsvariablen x(2,1,0) mit den LED(2,1,0)



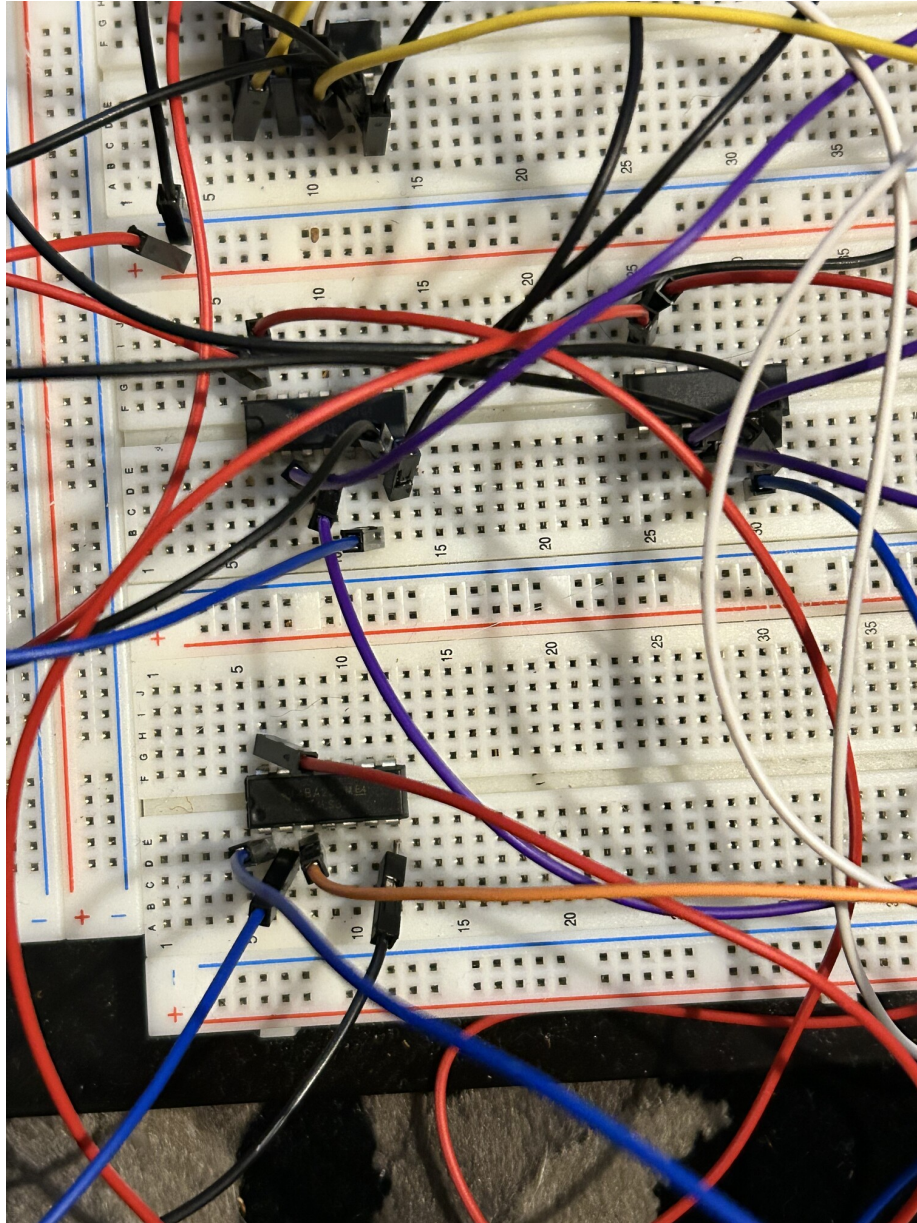


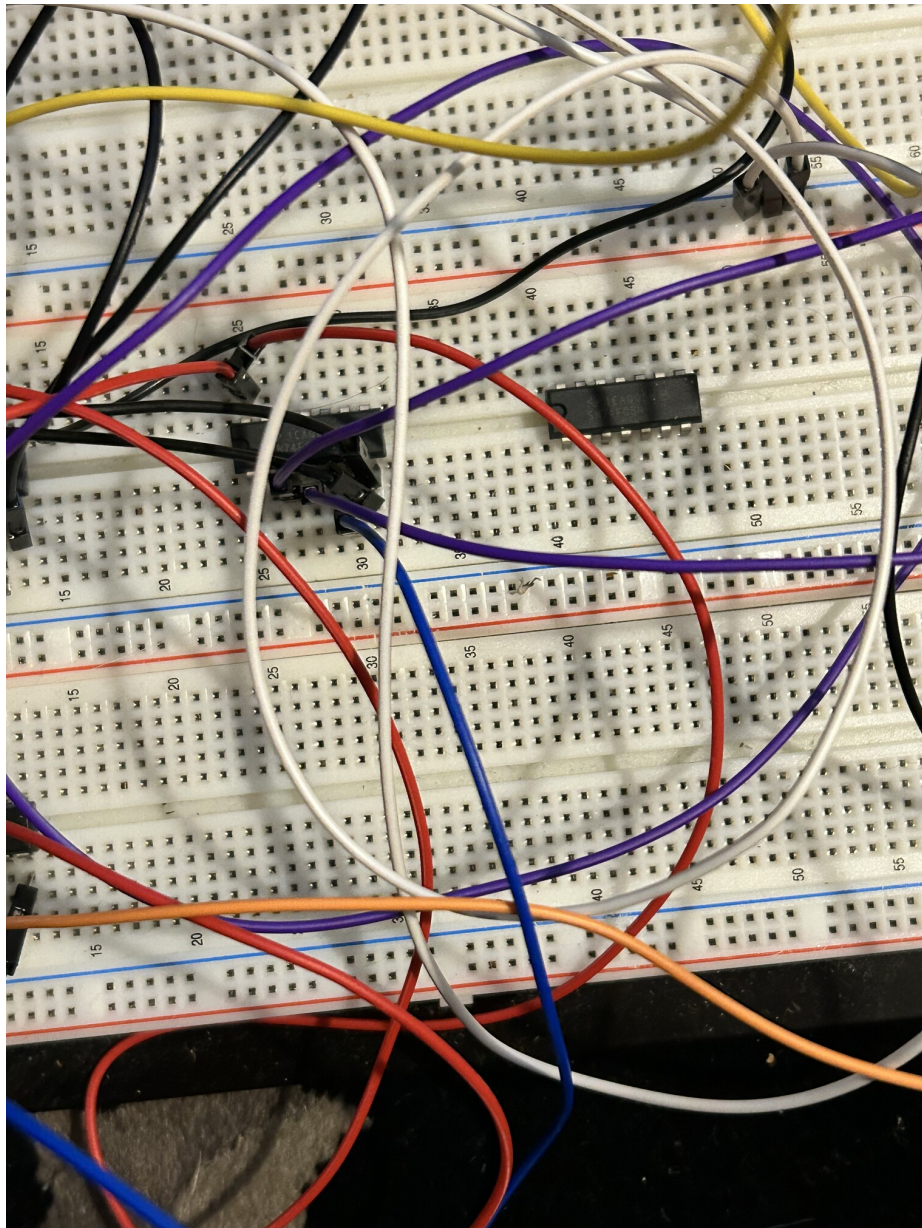
4.5 verbinden der ausgangvariable y mit der LED zur Anzeige

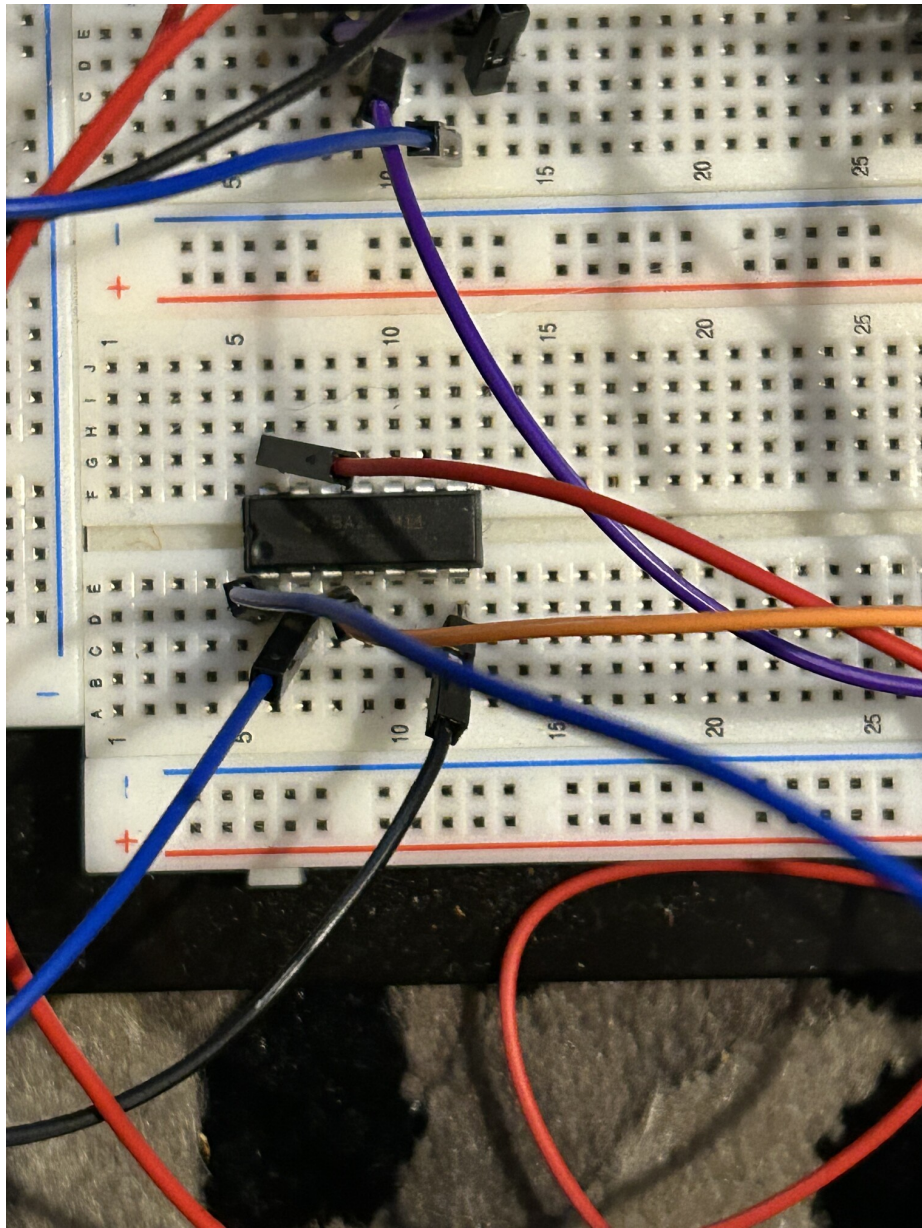




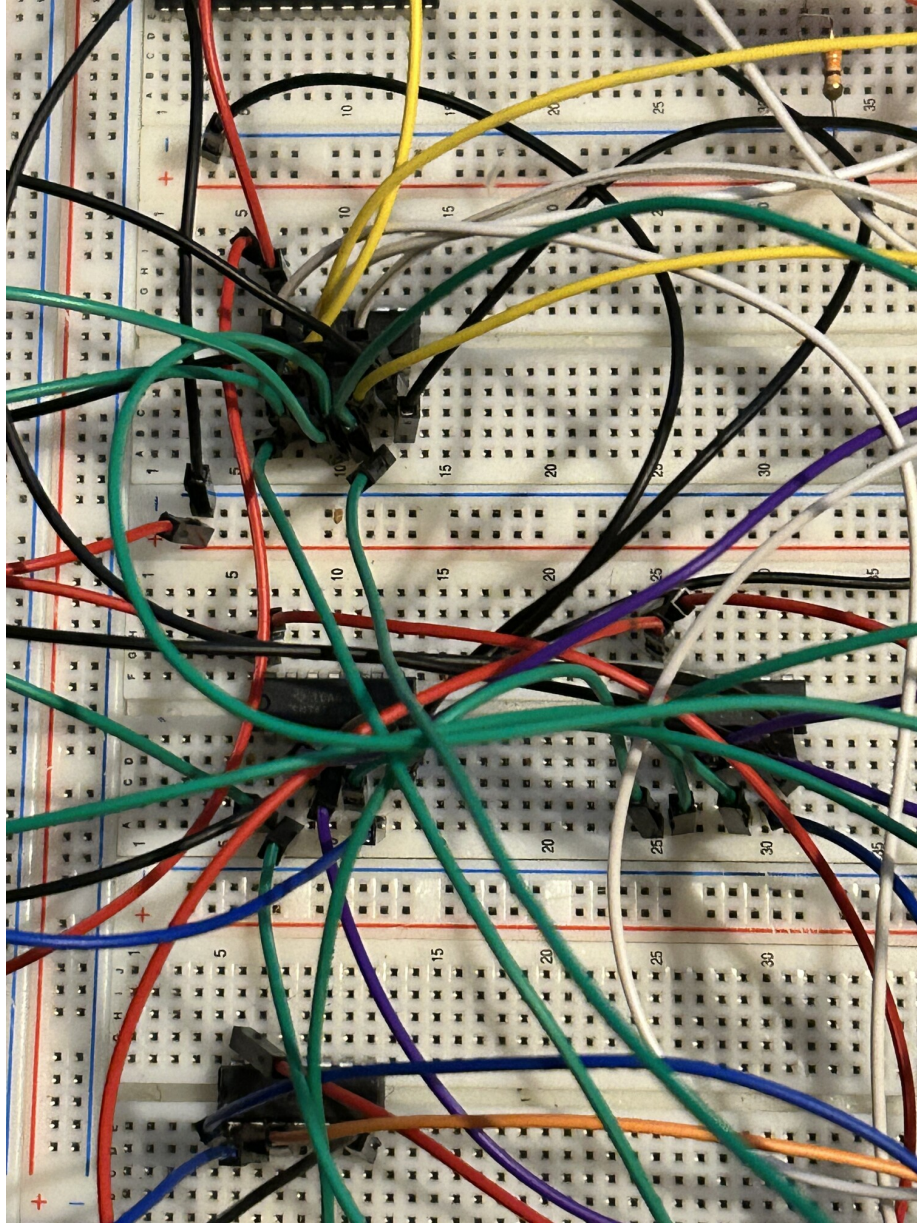
- 5 Verbinden der Ausgänge der Beiden UND -
für die DNF - zwei Minterme m_1 und m_2 wer-
den disjunktiv verknüpft

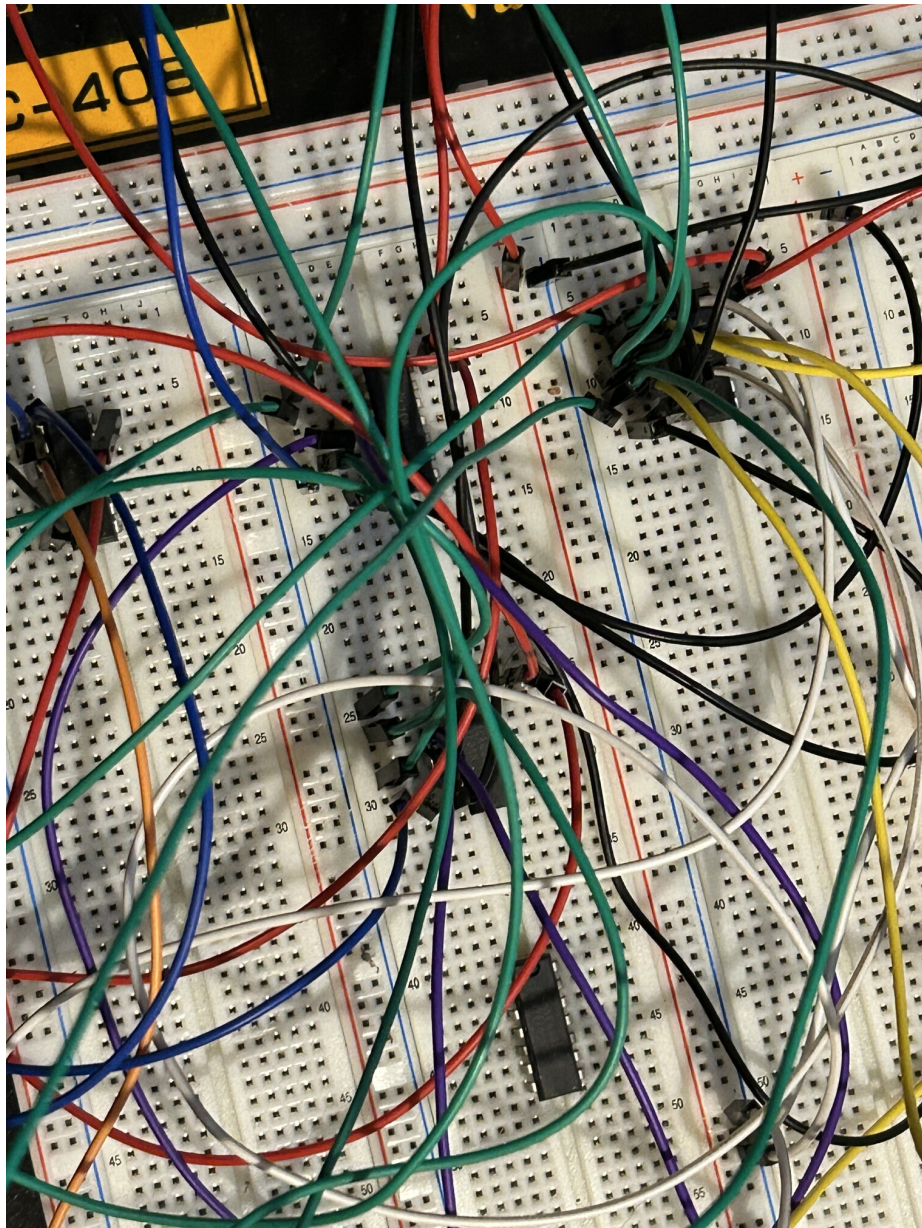




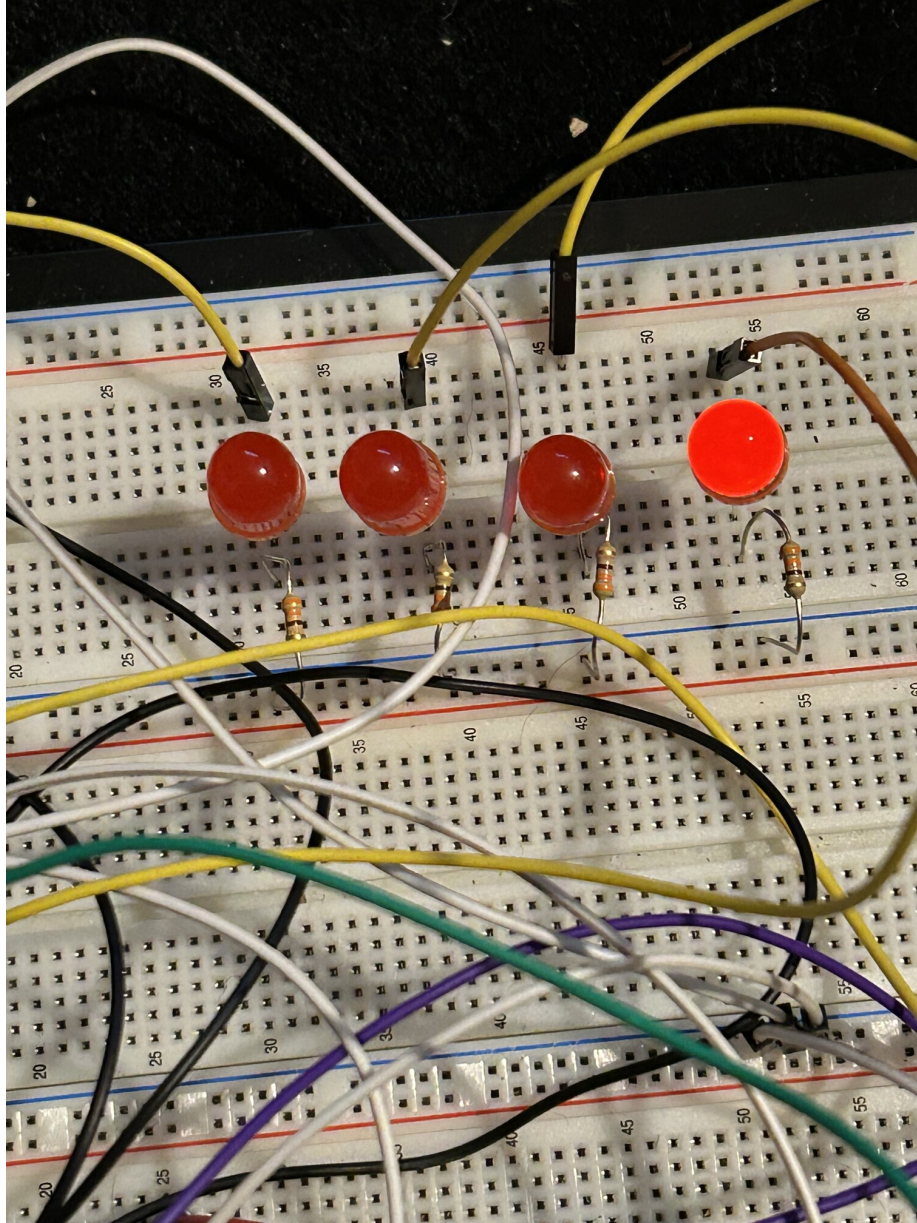


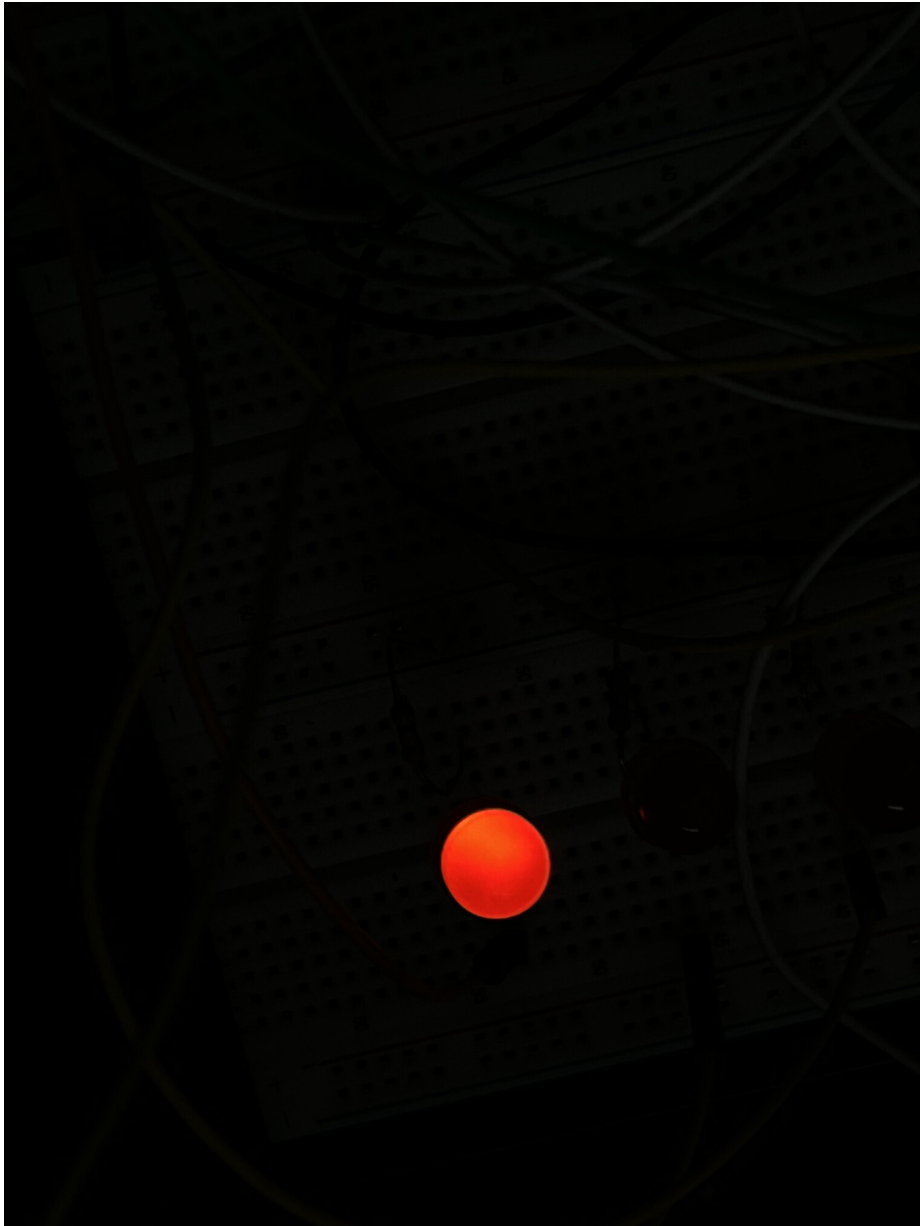
5.1 Die Eingangsvariablen negiert und nicht negiert werden entsprechend mit den Mintermen m_1 und m_2 verbunden, zwei Konjunktionen/UND/7408

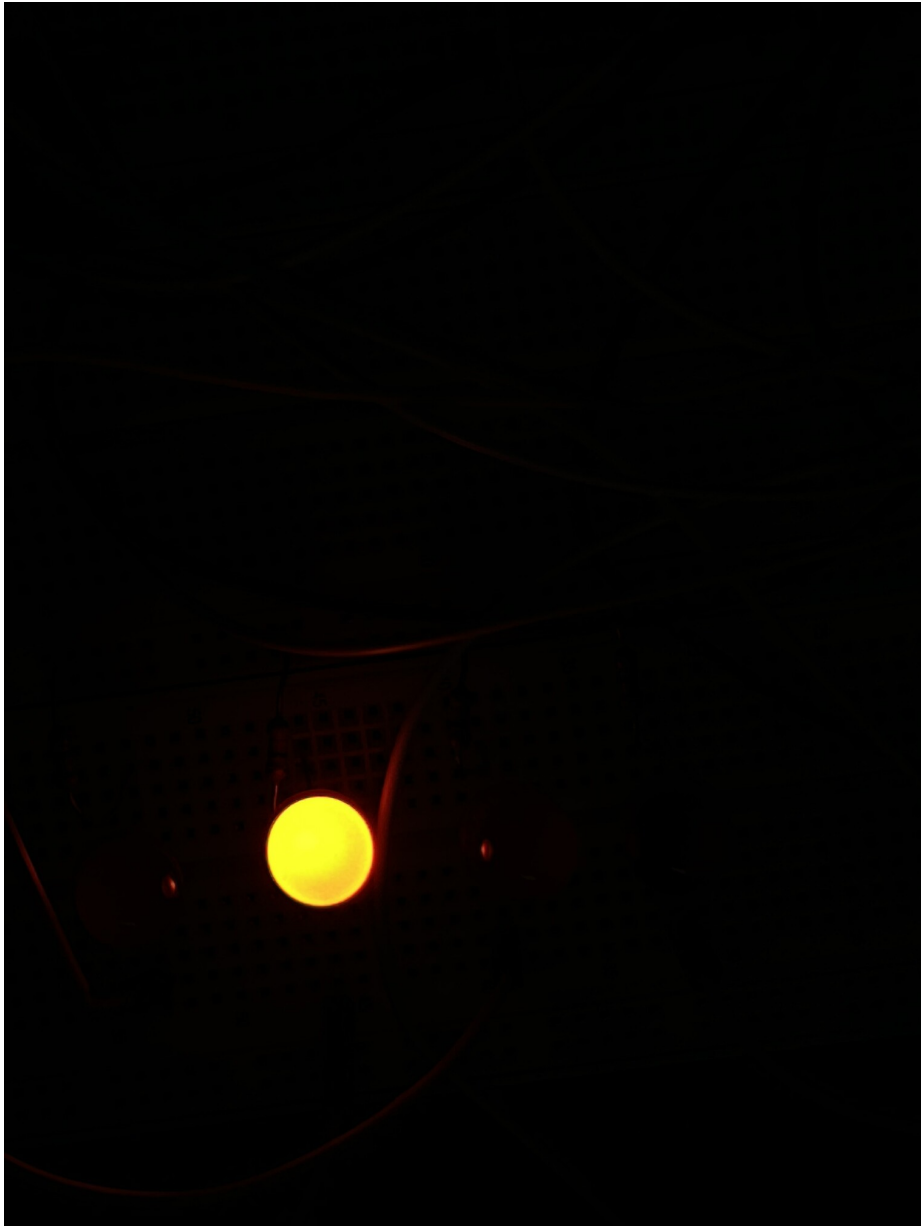


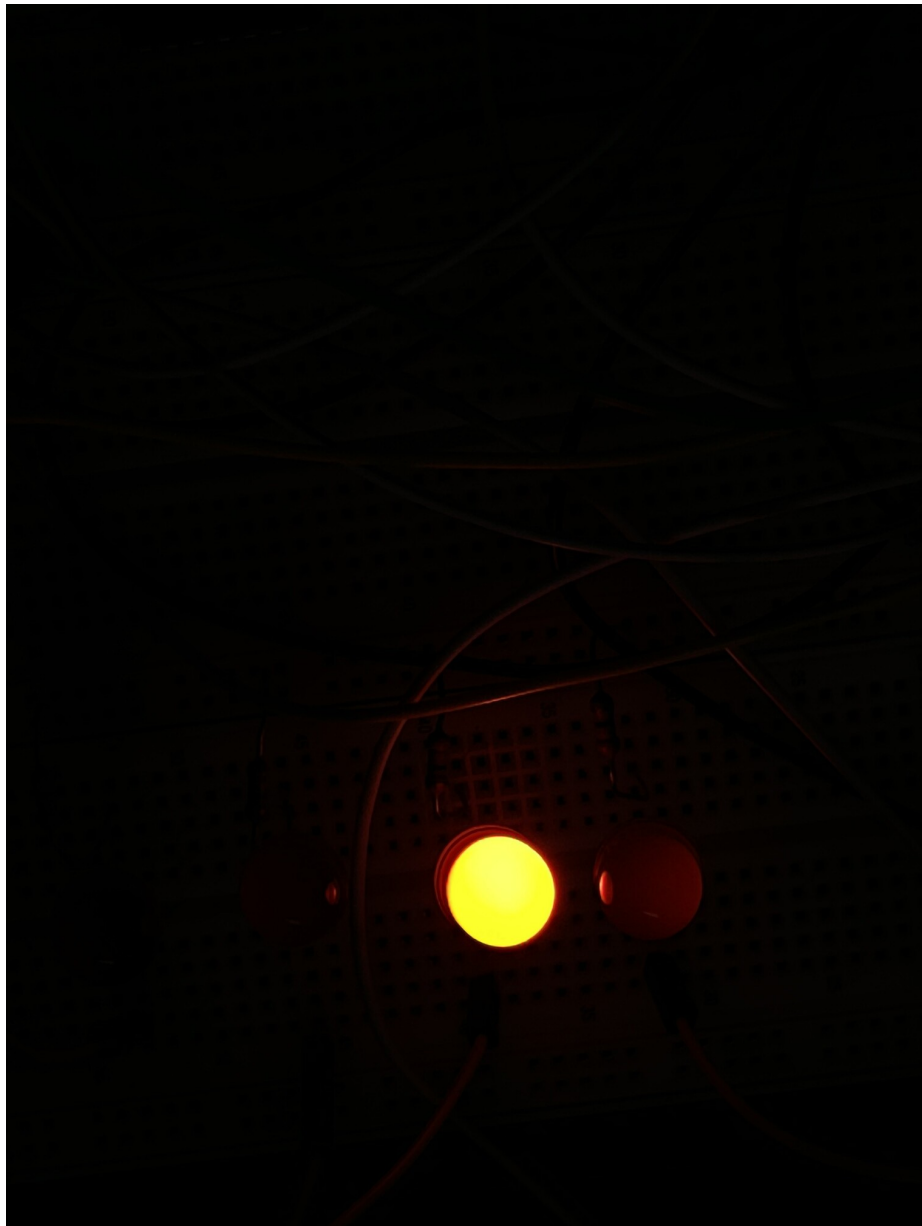


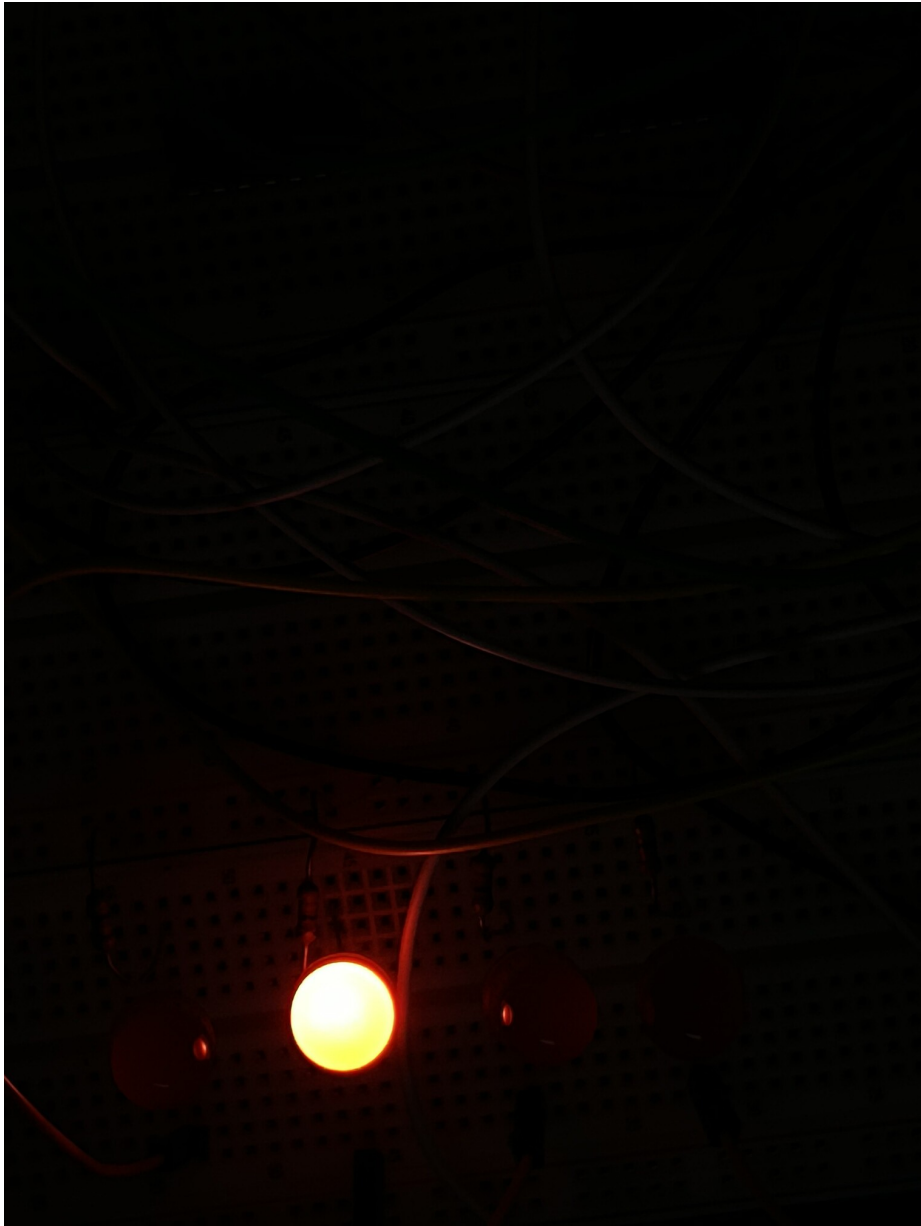
5.2 Der Physische test beginnt.

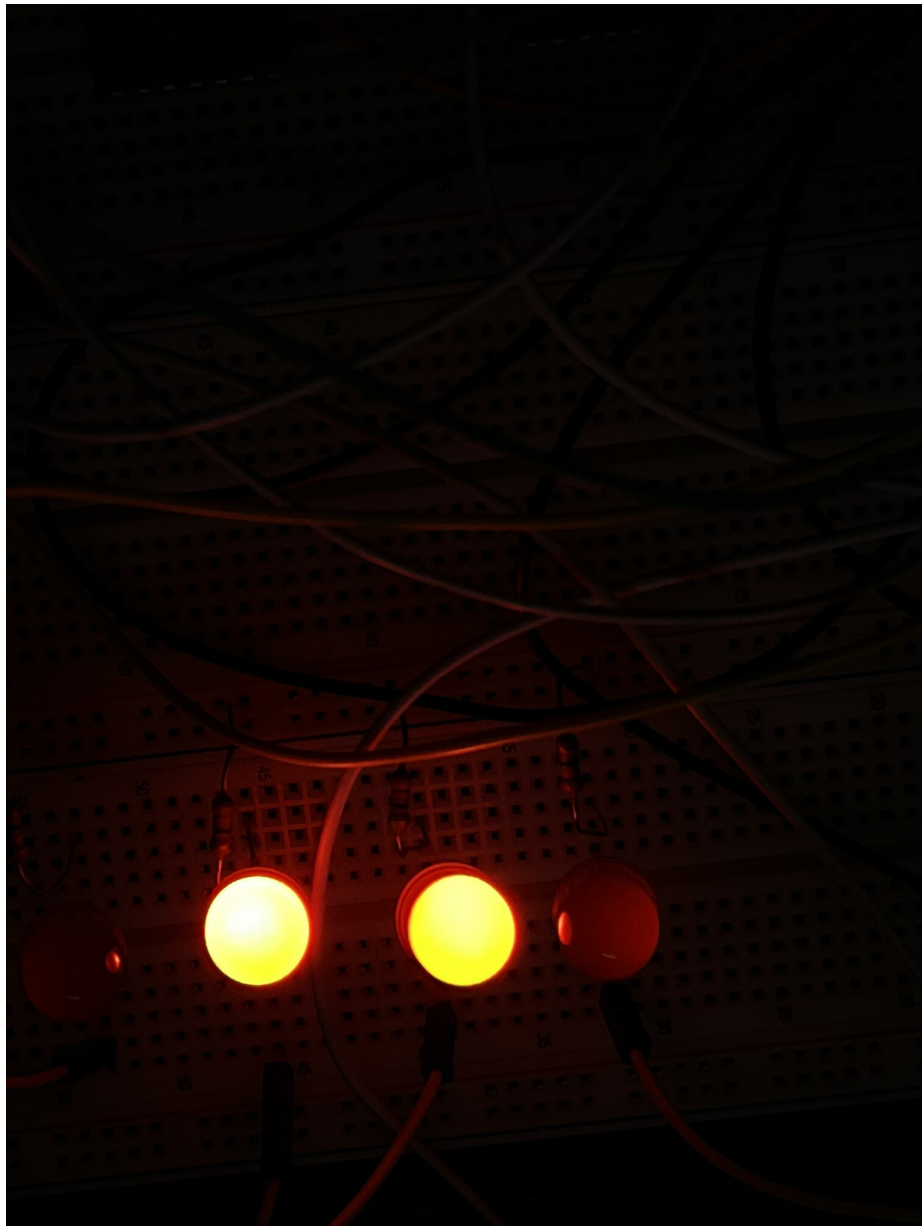


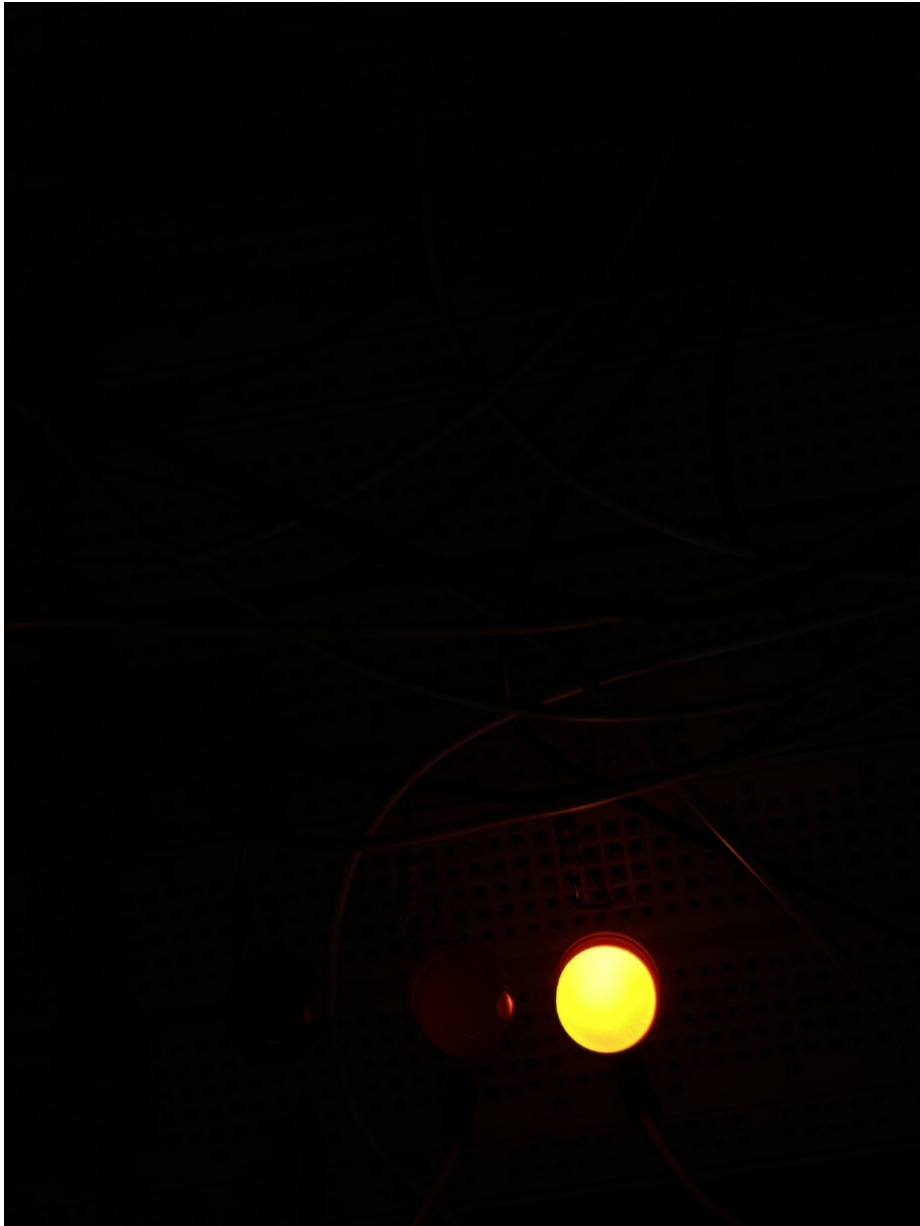




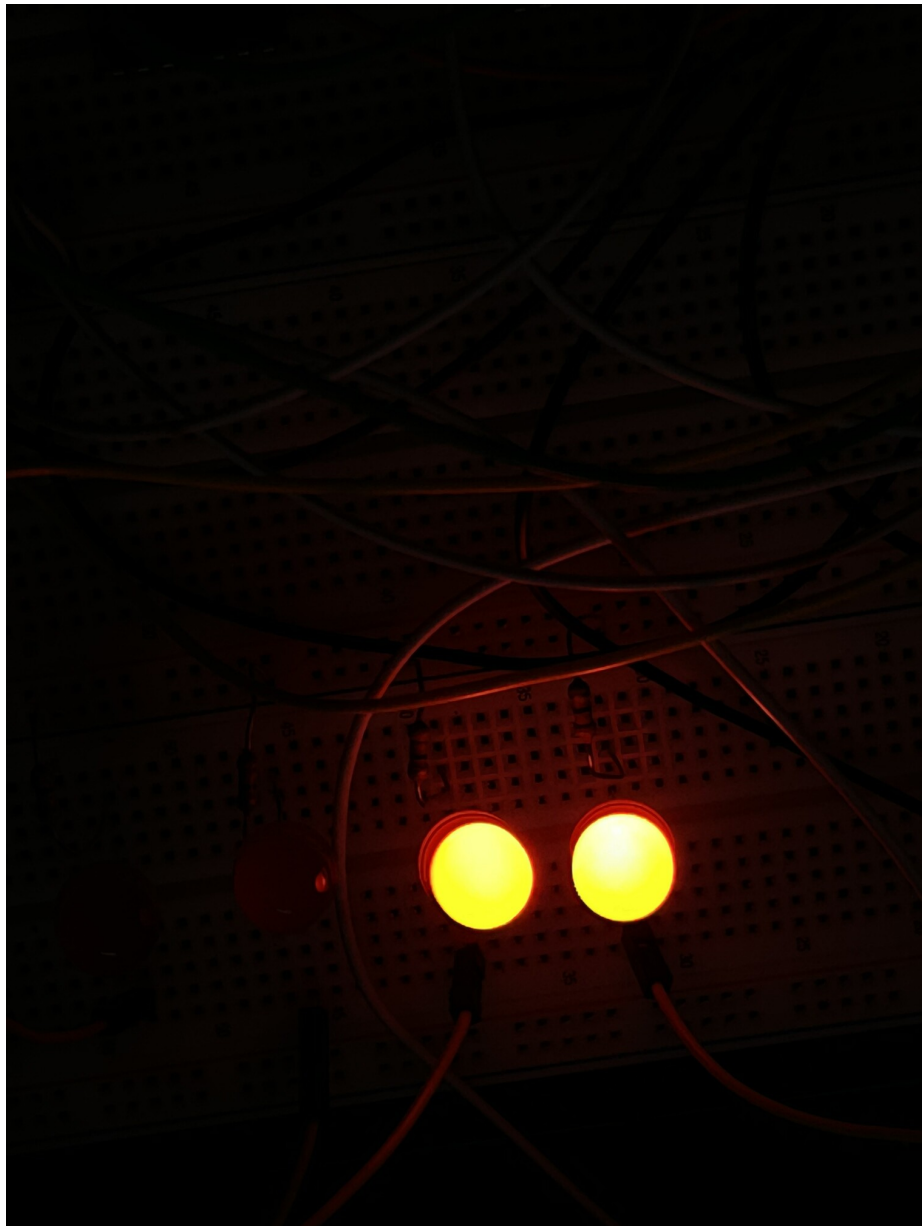


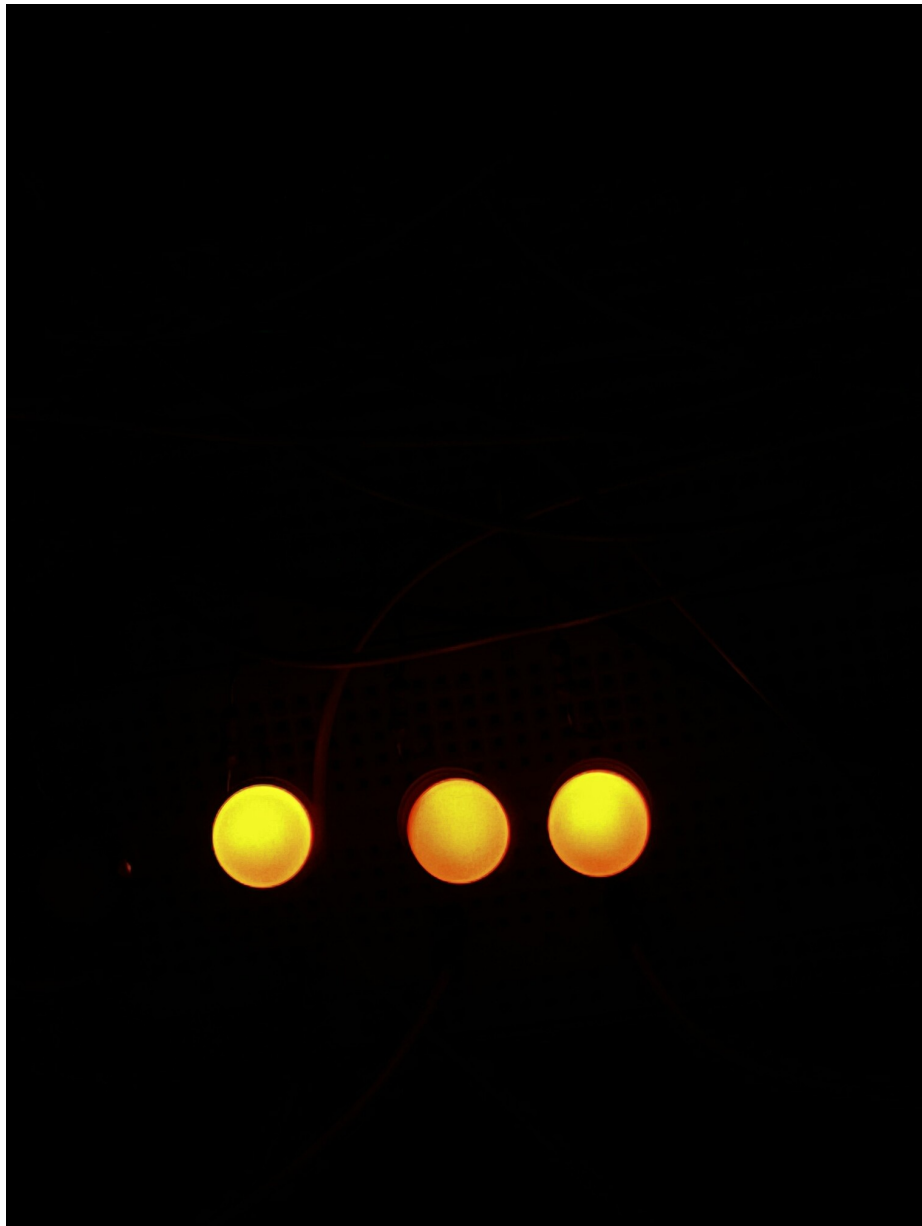


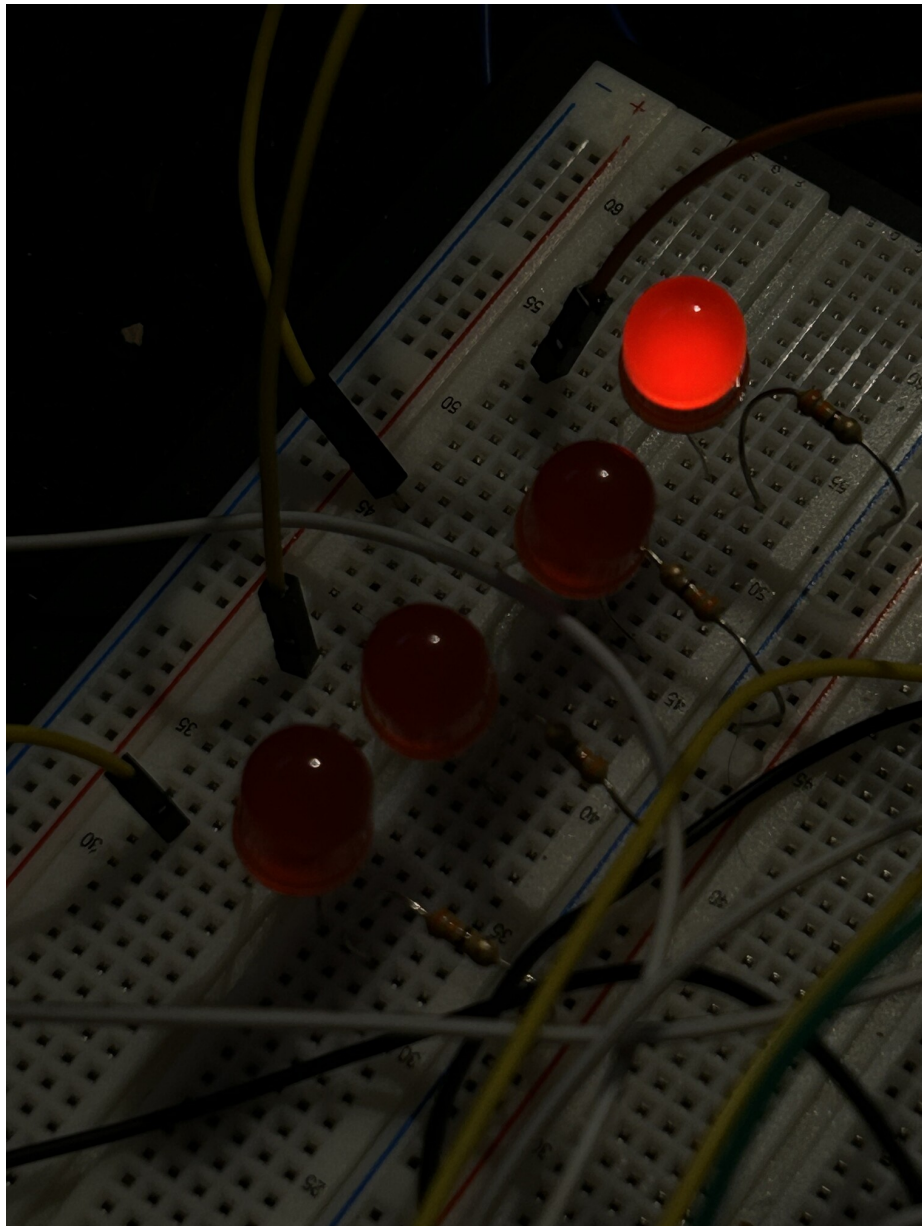


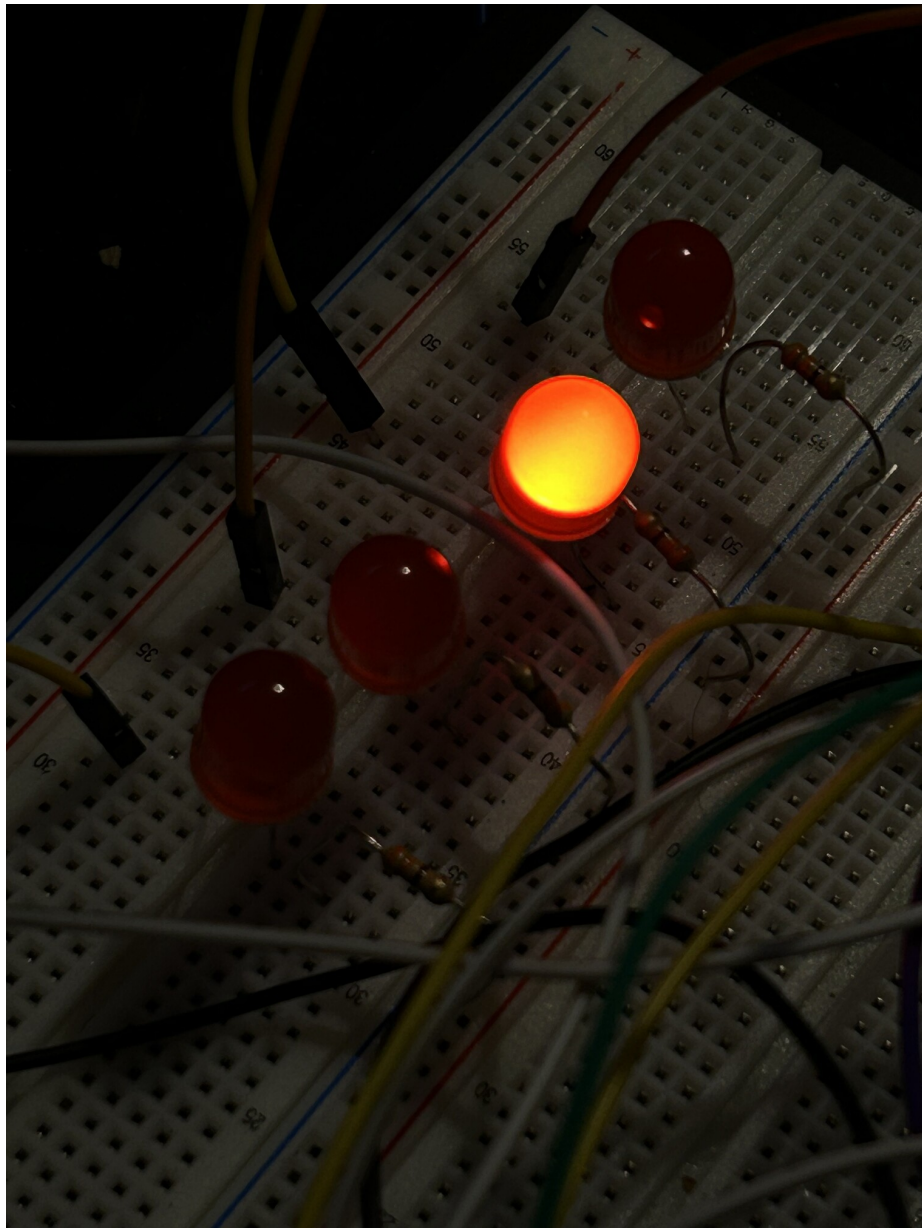


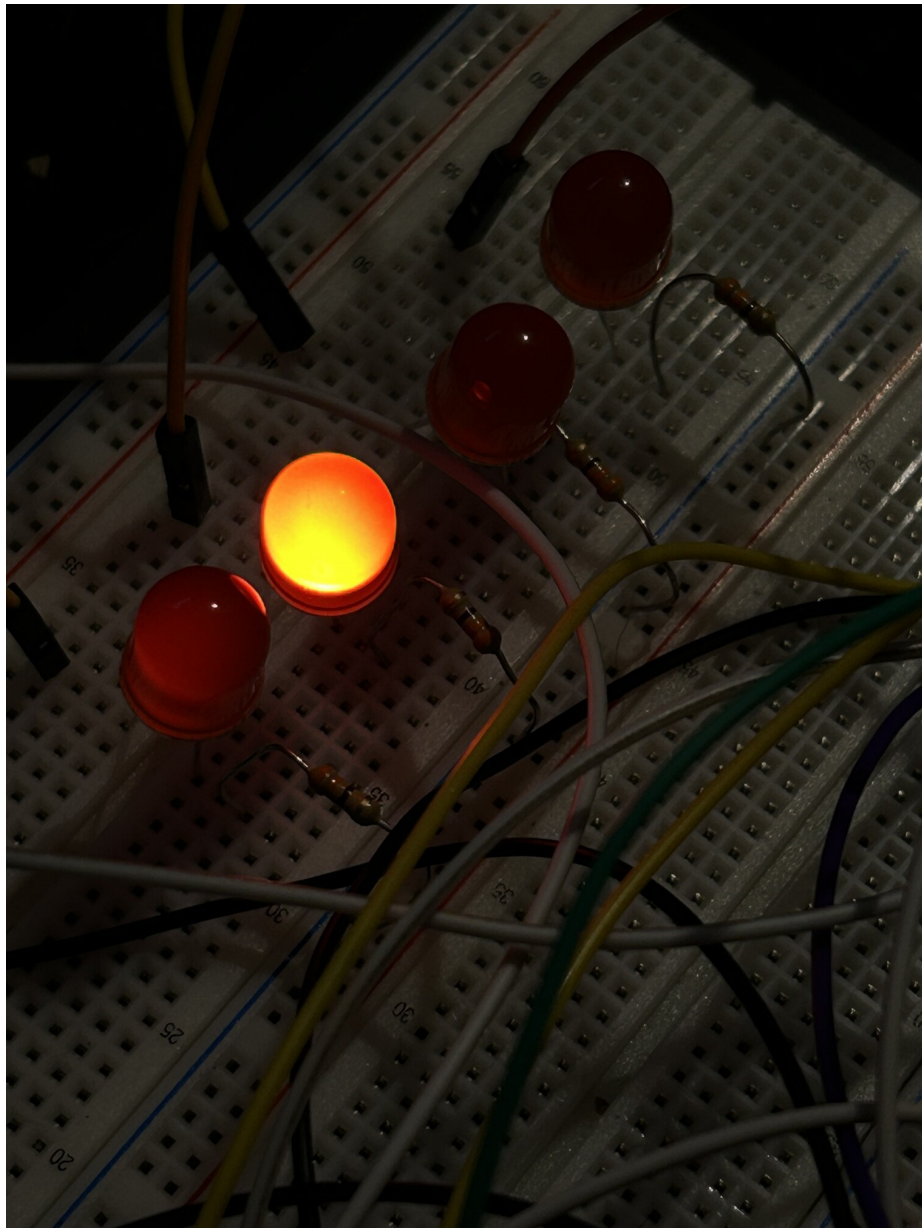


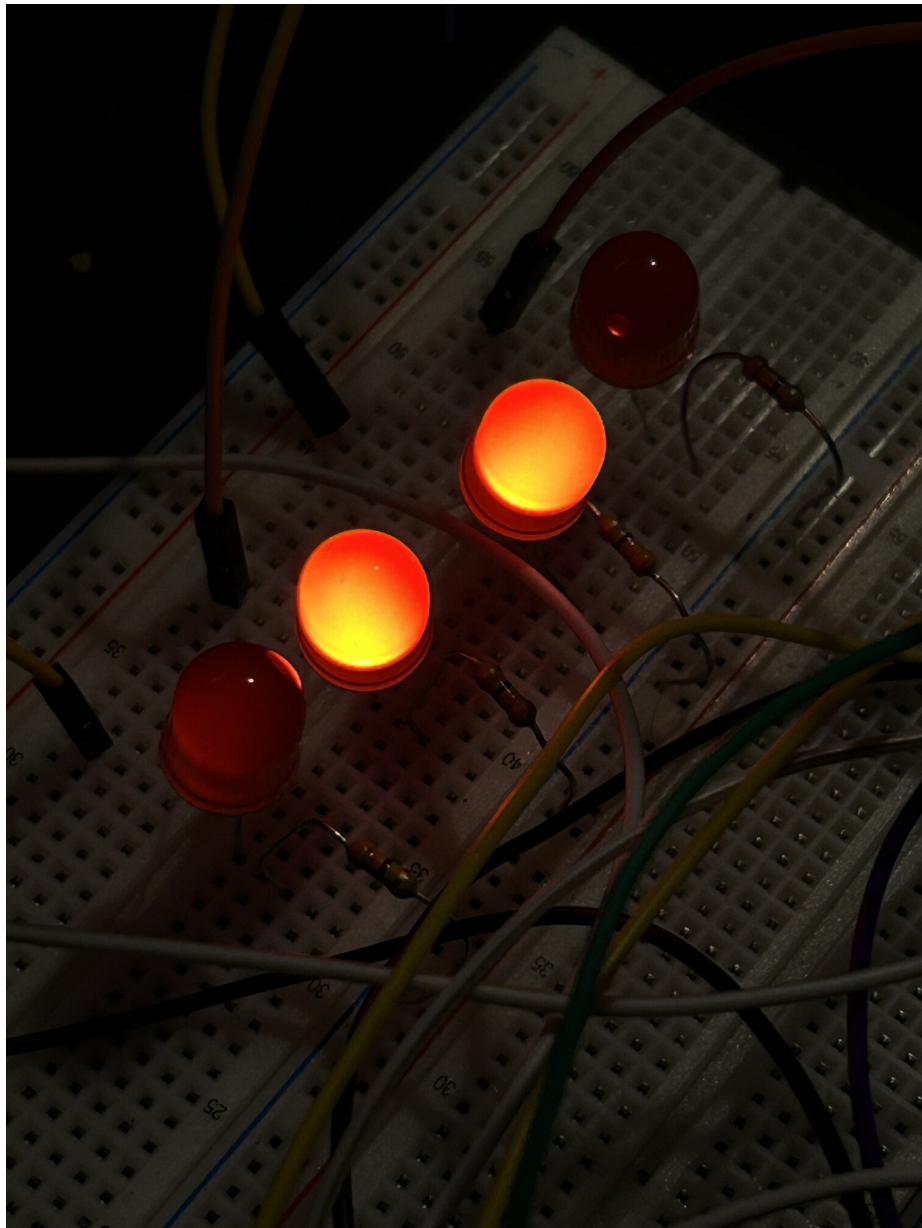


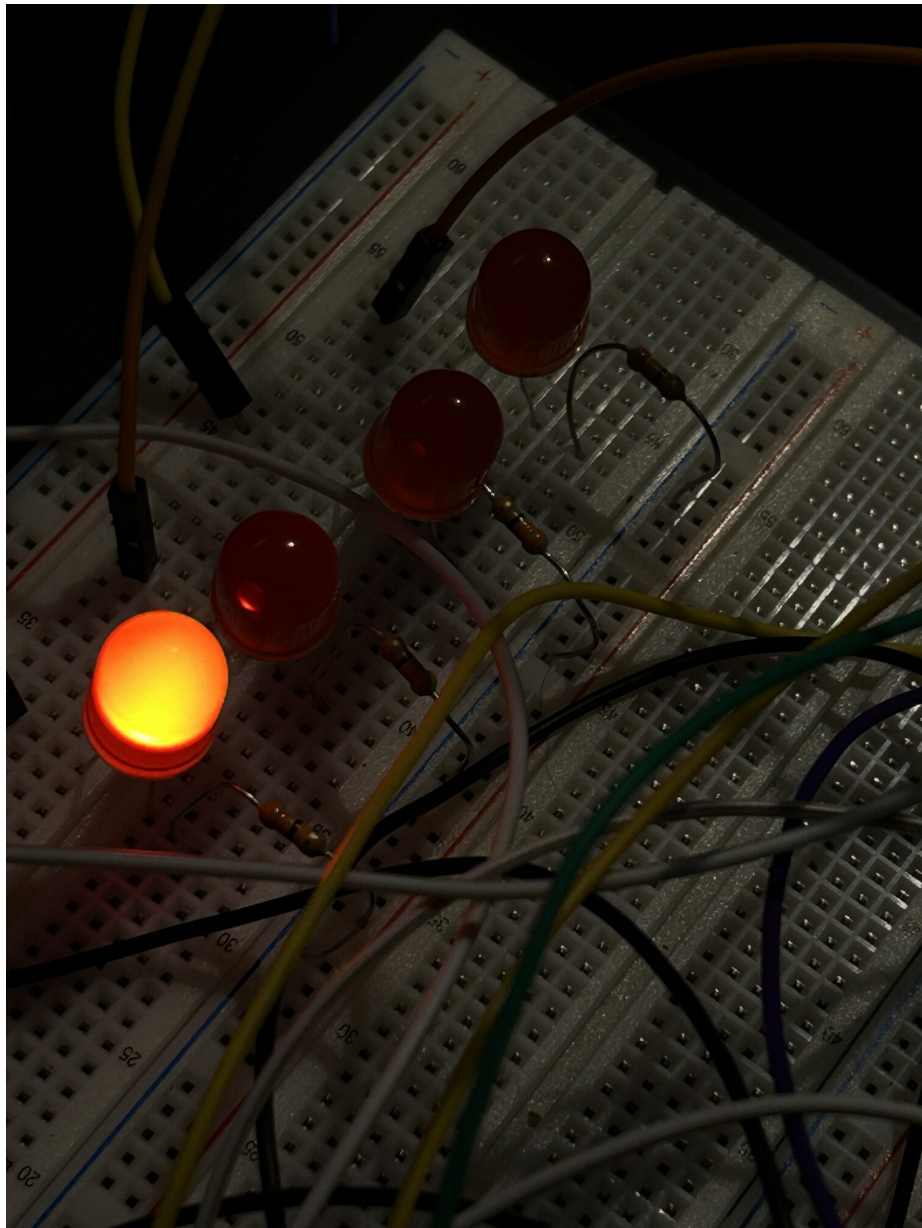


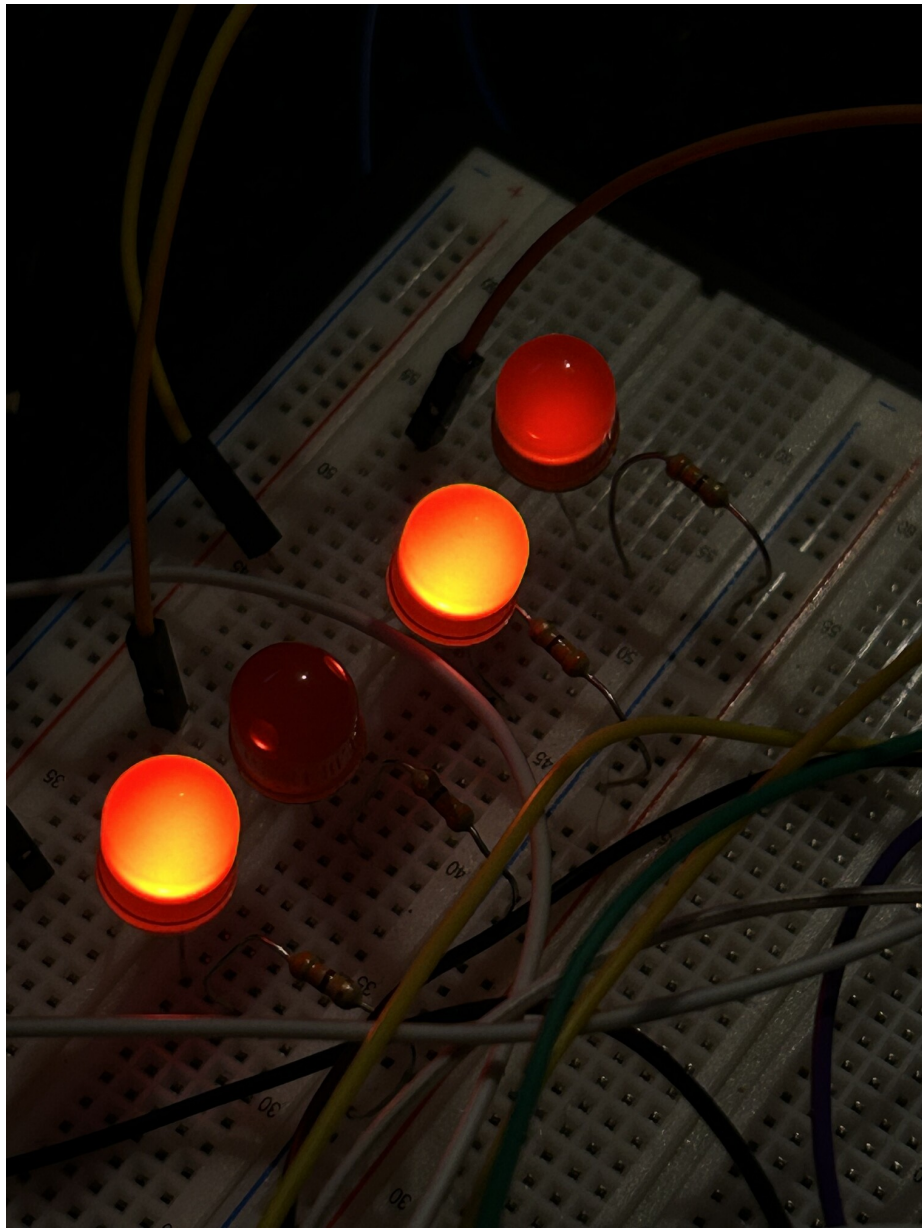


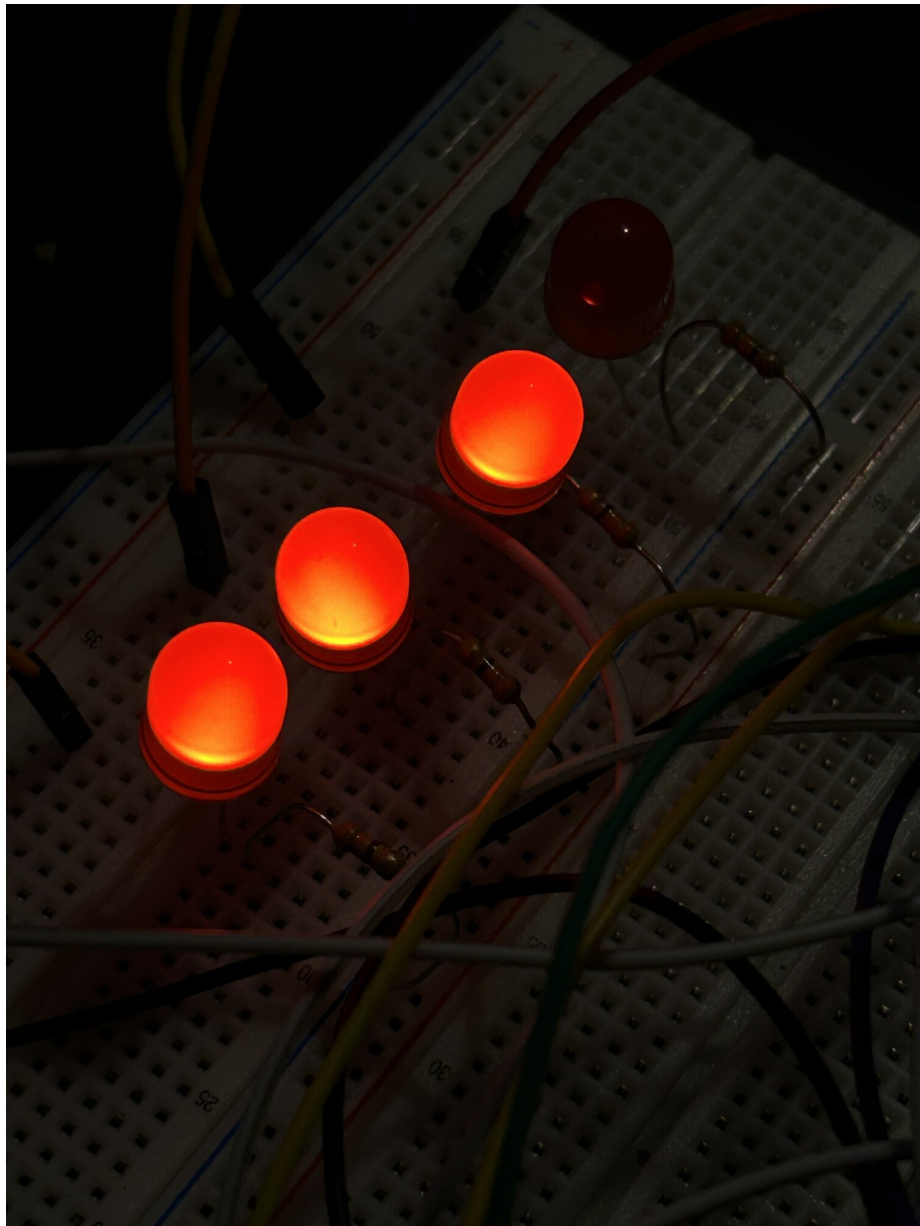












6 mit Mikrocontroller - atmega8 - atmel - bilder folgen gleich.

```
;; (c) david vajda  
;; https://www.vajda-breadboard.de  
;; 01/27/2026  
;; m8 led count 3 bit or more - ...  
;; ... human eye 1s visible clk ...
```

```

.include "m8def.inc"

ldi r16, HIGH (RAMEND)
out SPH, r16
ldi r16, LOW (RAMEND)
out SPL, r16
ldi r16, 0xff
out DDRB, r16

ldi r16, 0x00
l1:
out PORTB, r16
rcall f1
inc r16
rjmp l1

f1:
push r16
push r17
push r18
ldi r16, 0xff
l2:
ldi r17, 0xff
l3:
ldi r18, 0x04
l4:
dec r18
brne l4
dec r17
brne l3
dec r16
brne l2
pop r18
pop r17
pop r16
ret

```

6.1 Das übersetzen des assembler quelltextes

```

\begin{verbatim}
user@host:~$ avra ./m8count01272026.asm

```

damit entsteht ein Hex File - Binärcode wie auch immer.

```

./m8count01272026.hex

```

mit hexdump angeschaut sieht das file so aus: also, zum beispiel würde

```

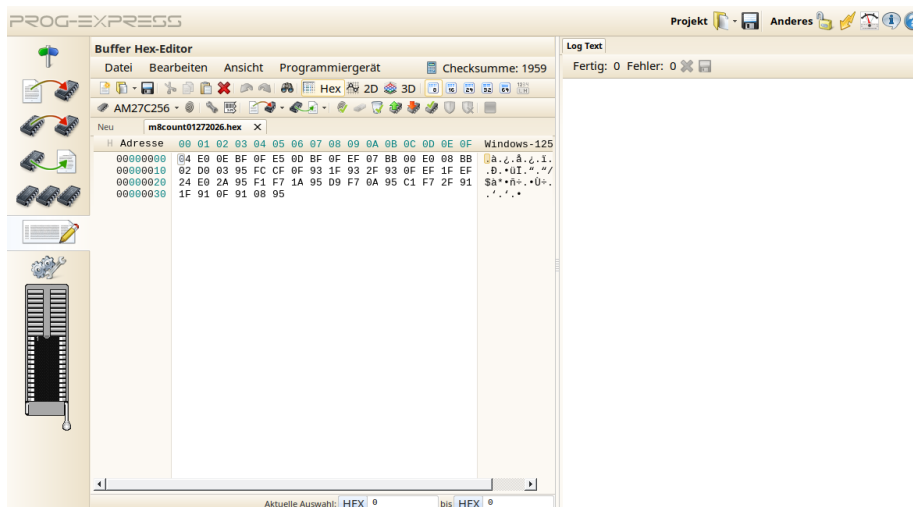
user@host:~/vhd101272026$ hexdump m8count01272026.hex
00000000 303a 3032 3030 3030 3032 3030 4630 0d43
00000010 3a0a 3031 3030 3030 3030 3430 3045 4530

```

```

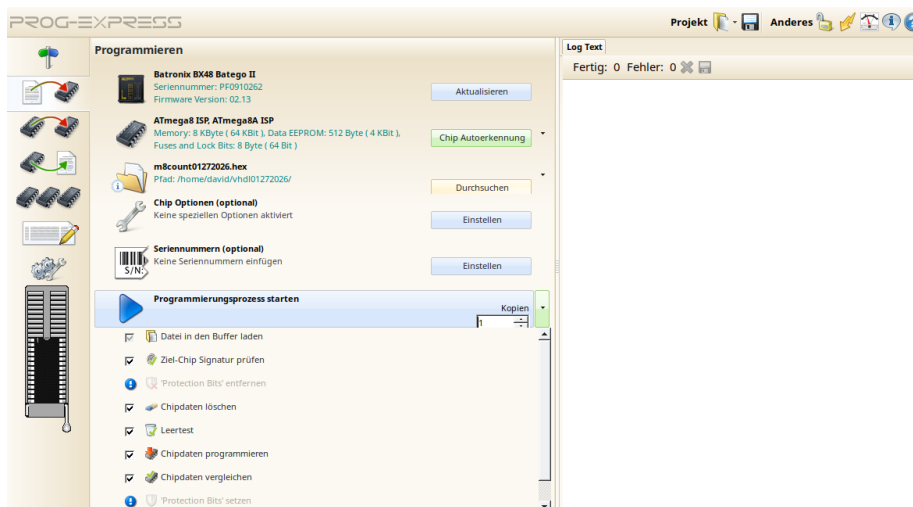
0000020 4642 4630 3545 4430 4642 4630 4645 3730
0000030 4242 3030 3045 3830 4242 4331 0a0d 313a
0000040 3030 3130 3030 3030 4432 3030 3933 4635
0000050 4343 3046 3946 3133 3946 3233 3946 3033
0000060 4546 3146 4546 3846 0d39 3a0a 3031 3030
0000070 3032 3030 3432 3045 4132 3539 3146 3746
0000080 4131 3539 3944 3746 4130 3539 3143 3746
0000090 4632 3139 4638 0a0d 303a 3036 3330 3030
00000a0 3130 3946 3031 3946 3031 3938 4435 0d44
00000b0 3a0a 3030 3030 3030 3130 4646 0a0d
00000be
user@host:~/vhd101272026$

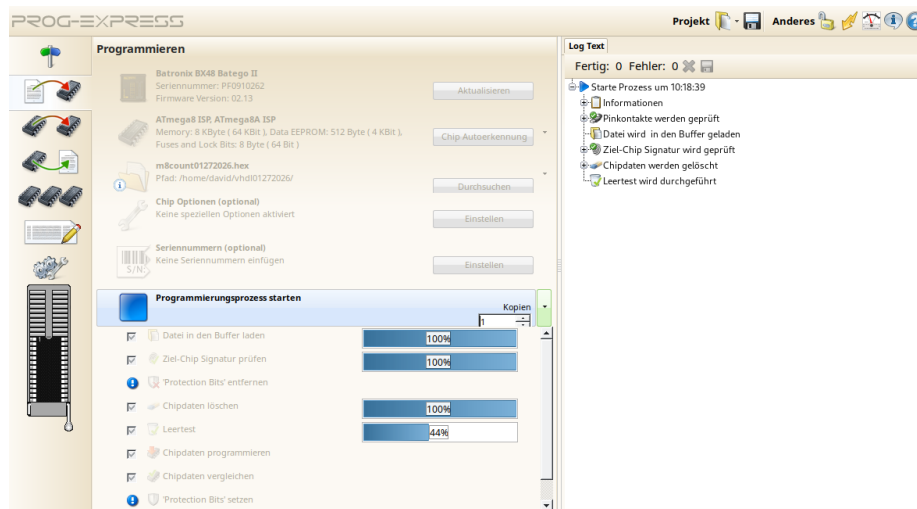
```



so, jetzt haben wir den hexfile. jetzt es drum damit den atmega8 zu programmieren.

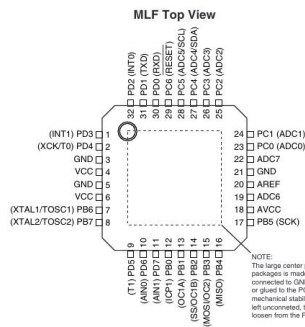
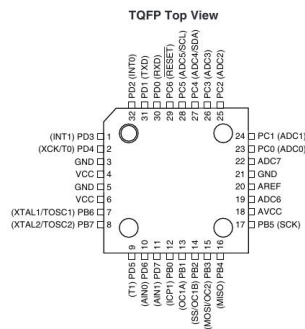
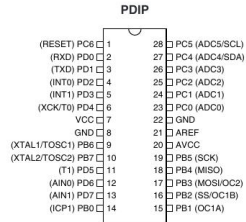
dafür gibt es verschiedene programmiergeräte, ich nehme das batronix selber. . .





so, jetzt haben wir das. jetzt geht es um das nächste... wenn den controller programmiert ist, muss er angeschlossen werden, statt schalter eingabe manuell muss der controller machen und jetzt muss man wissen, wo sind die pins, das steht im datasheet (datenblatt)

Pin Configurations



6.2 die funktion

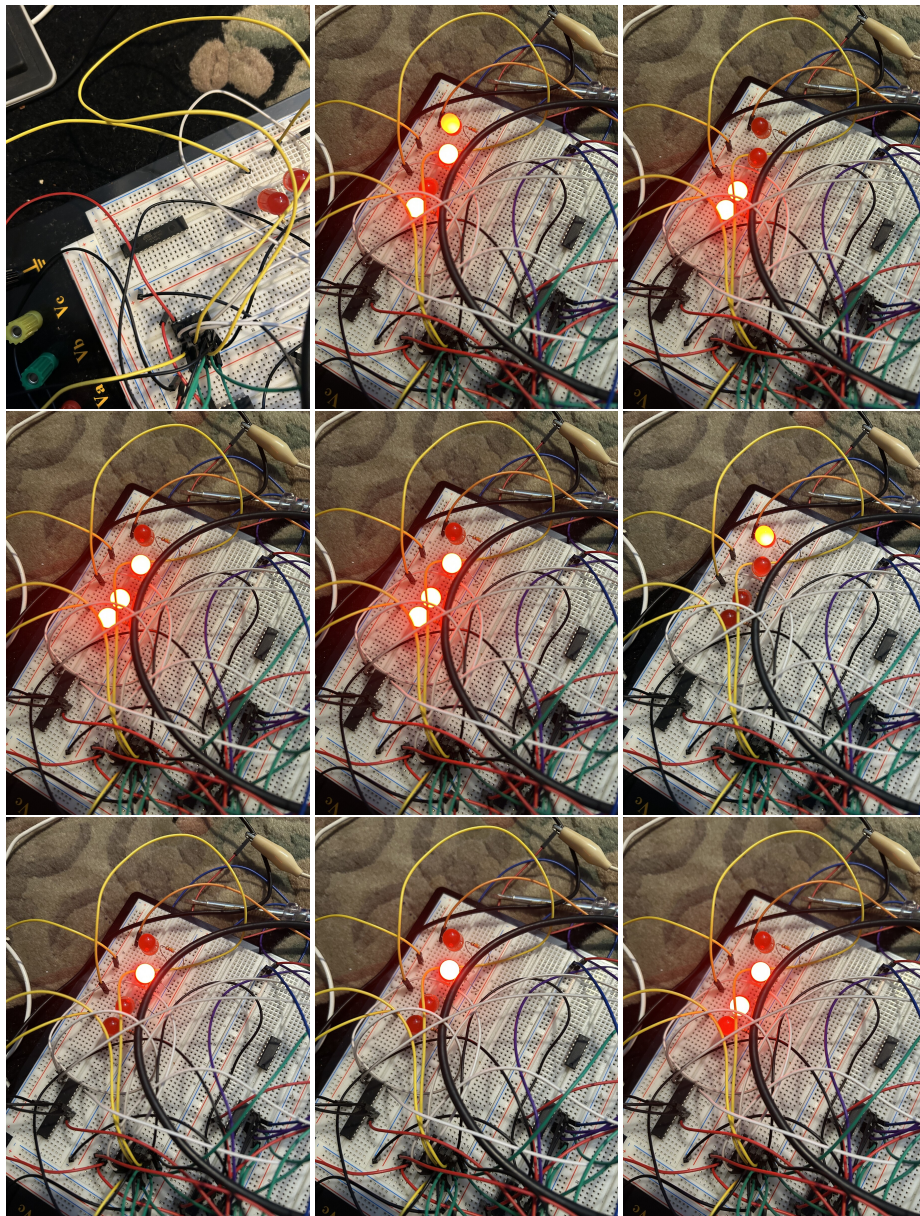
wenn das nun angeschlossen ist, ergibt das an und für sich keinen sinn, ohne film, meint man. zunächst müssen wir unterscheiden:

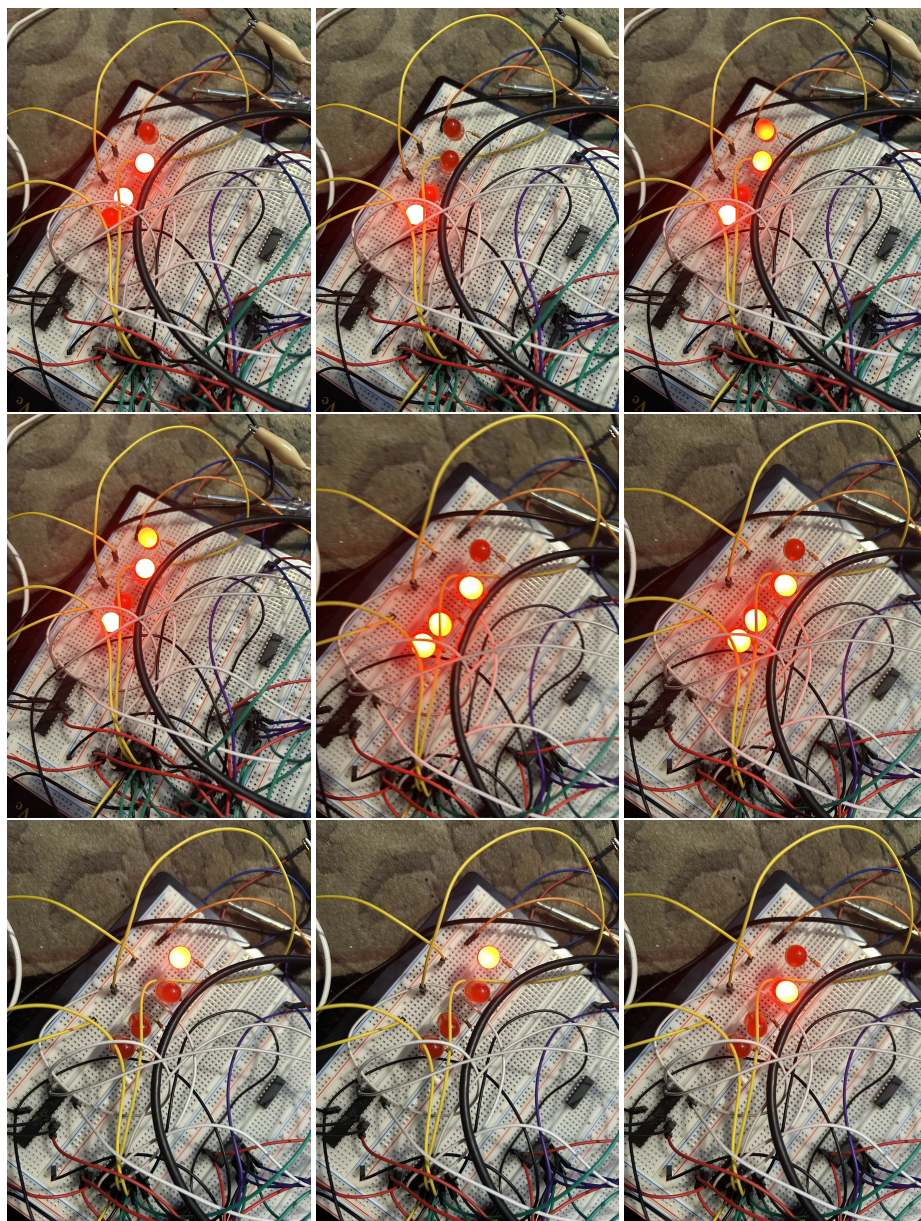
1. Taktfrequenz: für das menschliche Auge, LED - Lichterkette - dann brauchen wir entweder eine warteschleife oder ein timer intervall
2. oszilloskop: das entscheidende ist, dass sich das signal staendig wiederholt. wenn ein signal ein mal kommt, dann bleibt das nicht im oszilloskop. . .

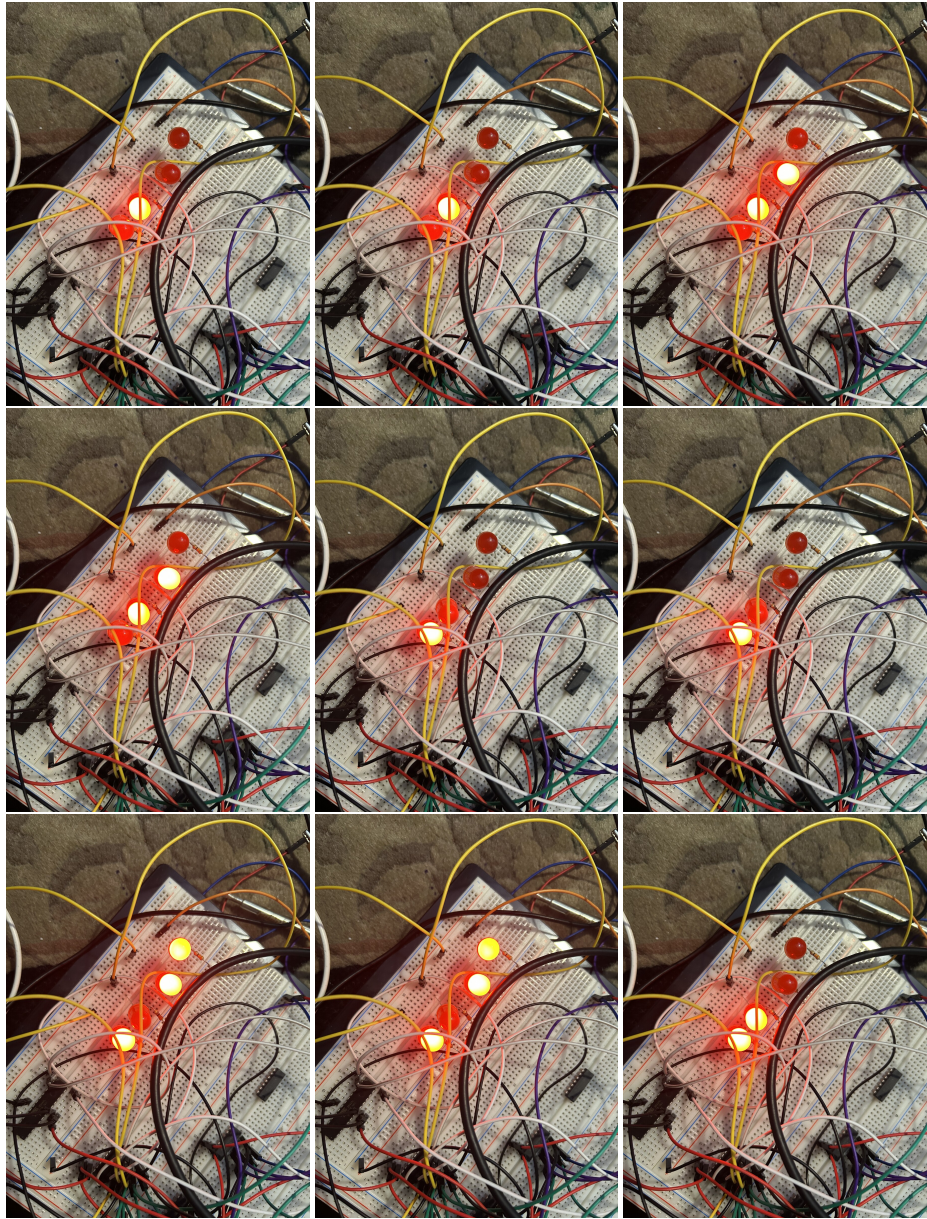
das ist allerdings hochfrequenz, ohne warteschleife.

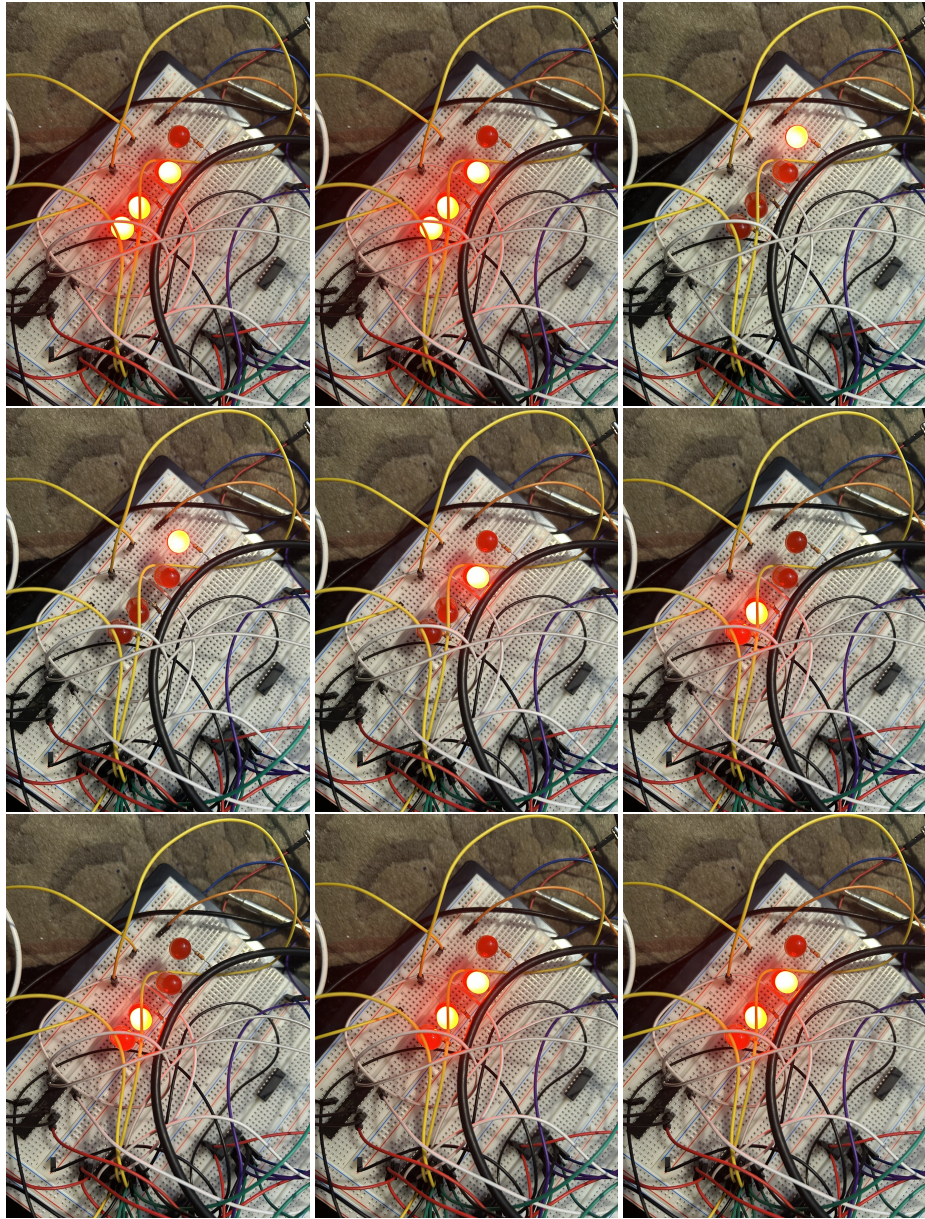
nun poste ich ein mal die photos die entstanden sind, als: die LED gelaufen sind.
und sage etwas dazu.

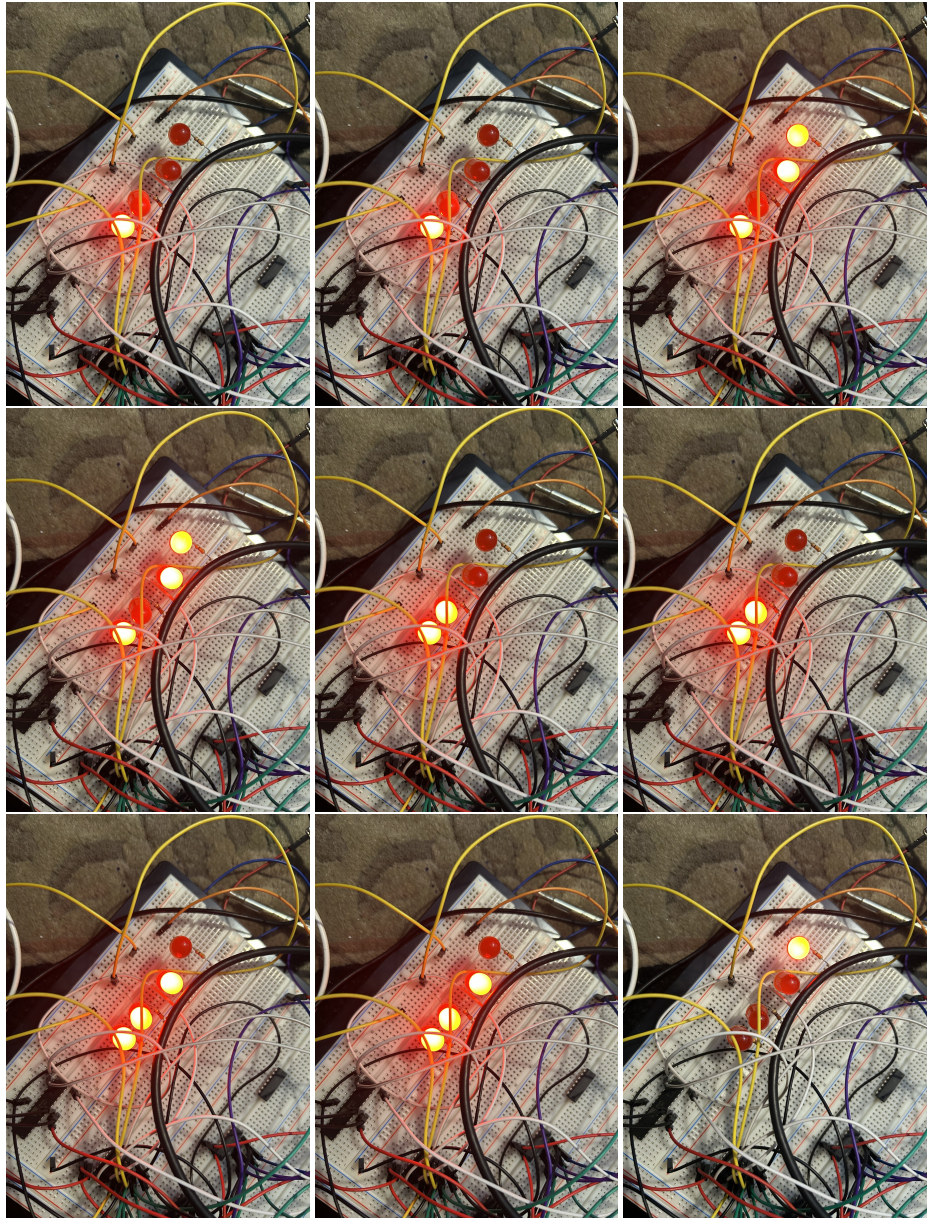
7 Bilder mit dem controller gezählt für das menschliche auge.

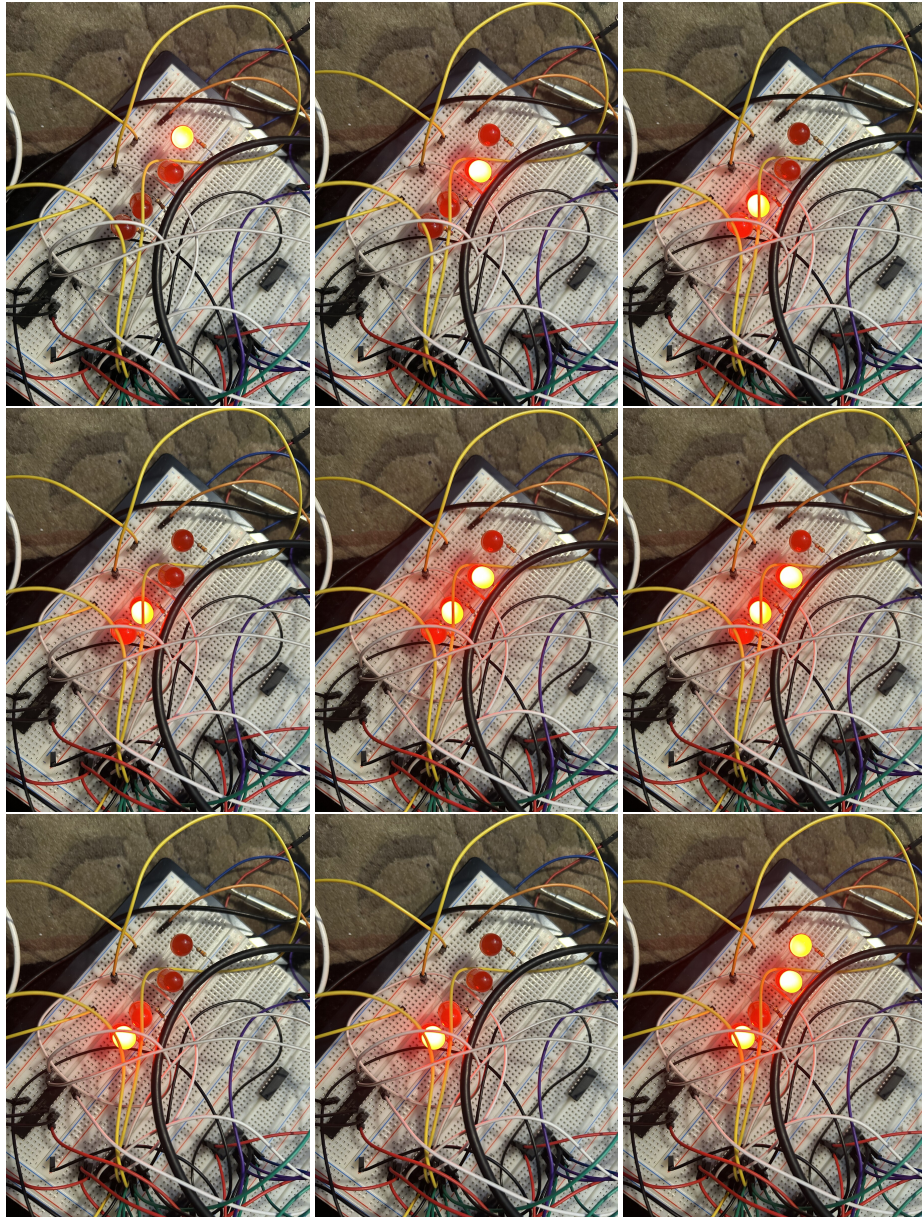


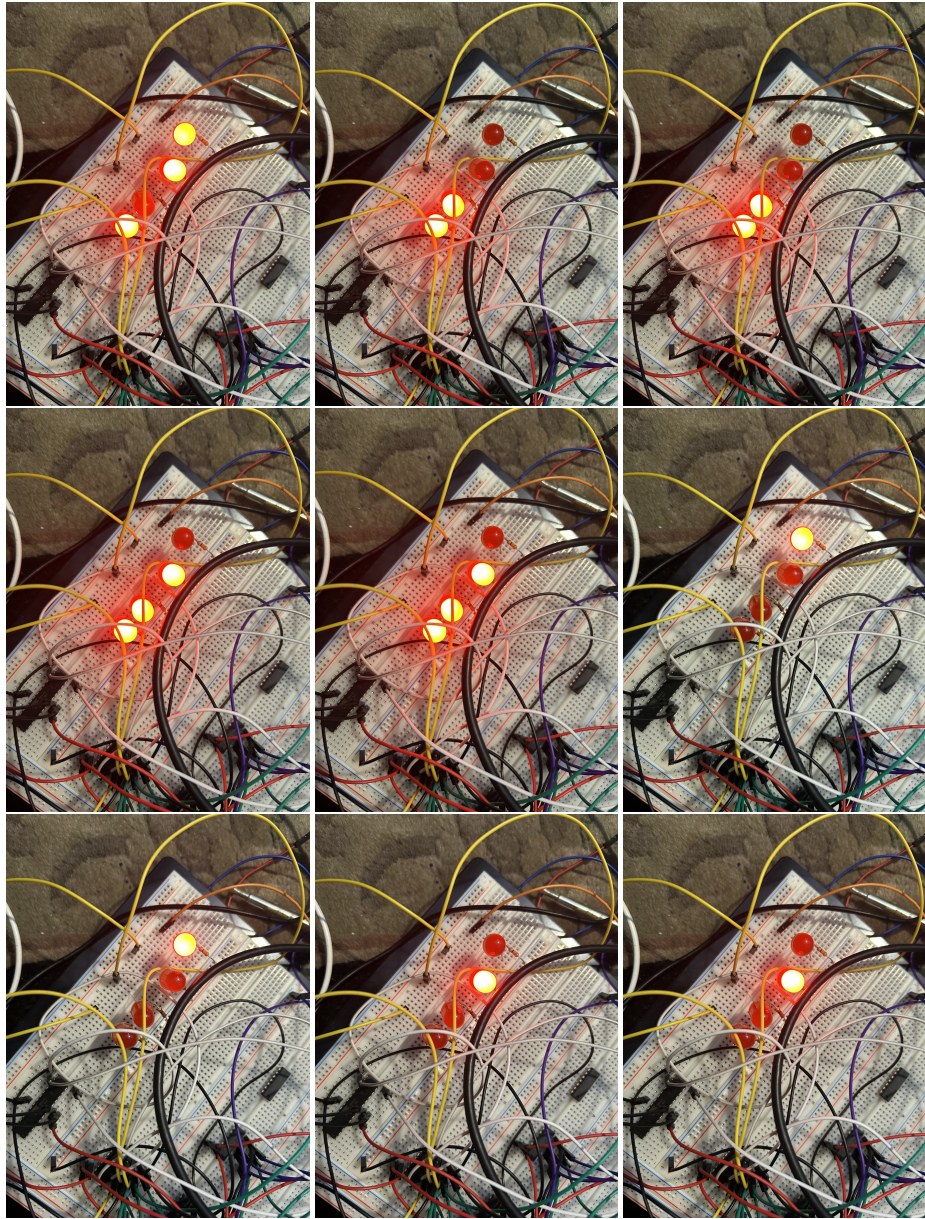


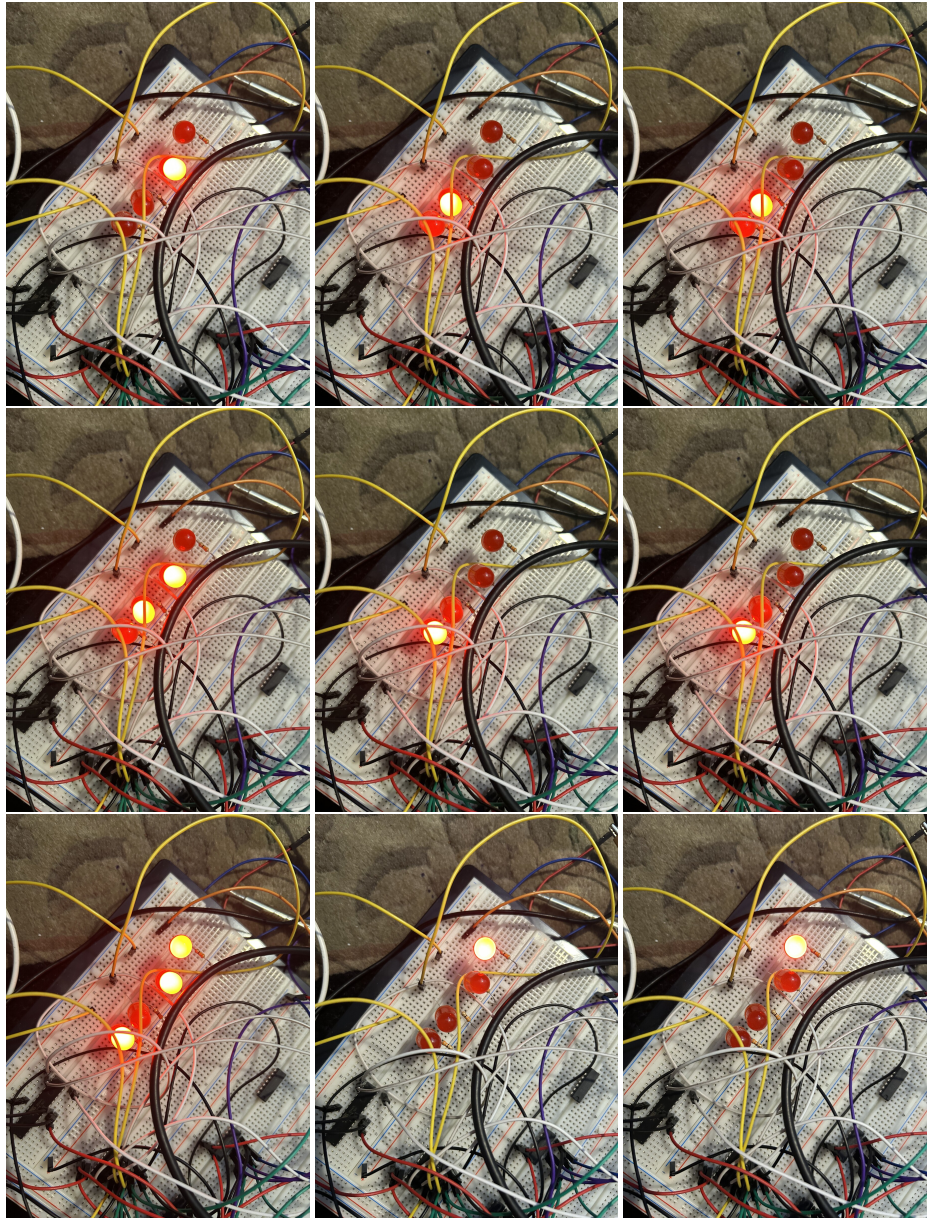


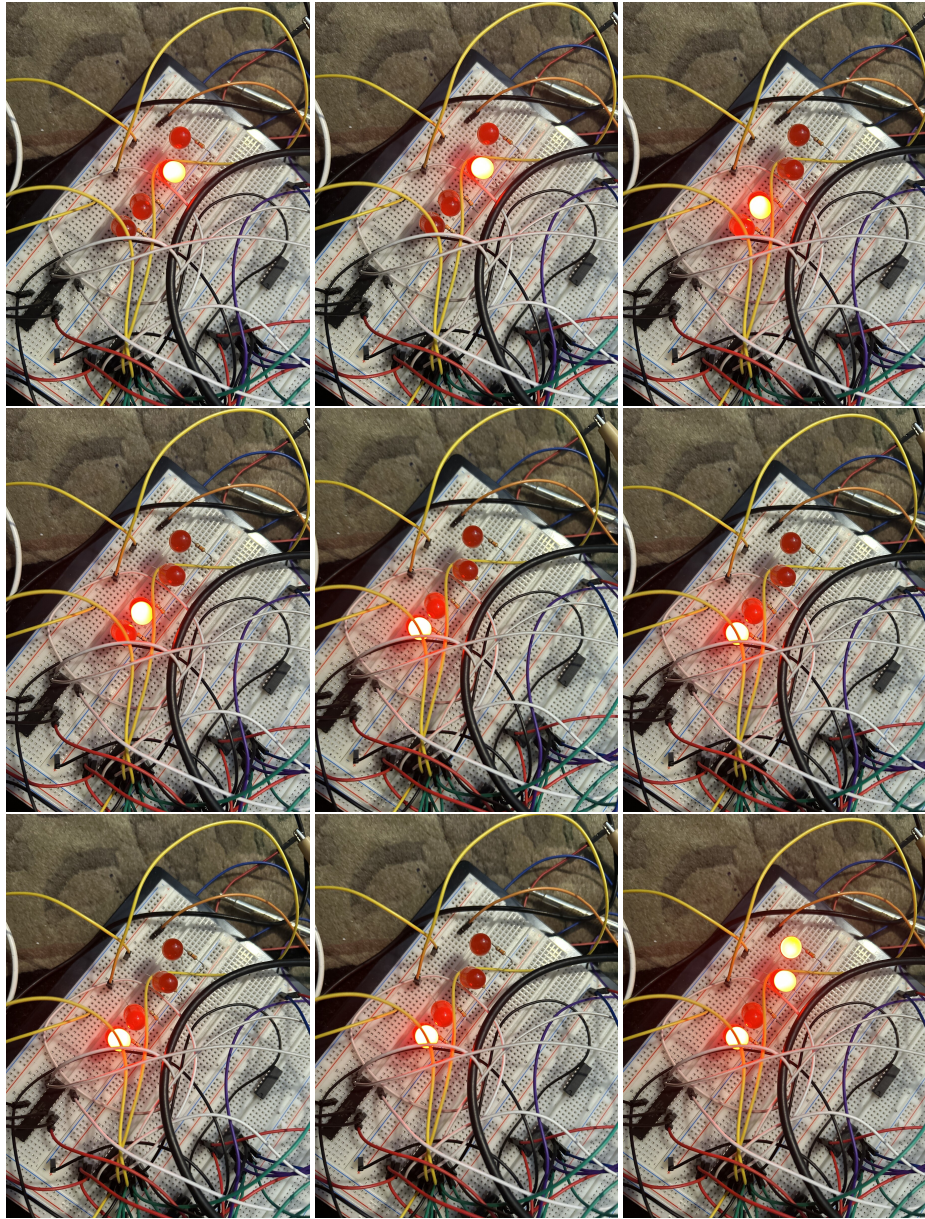


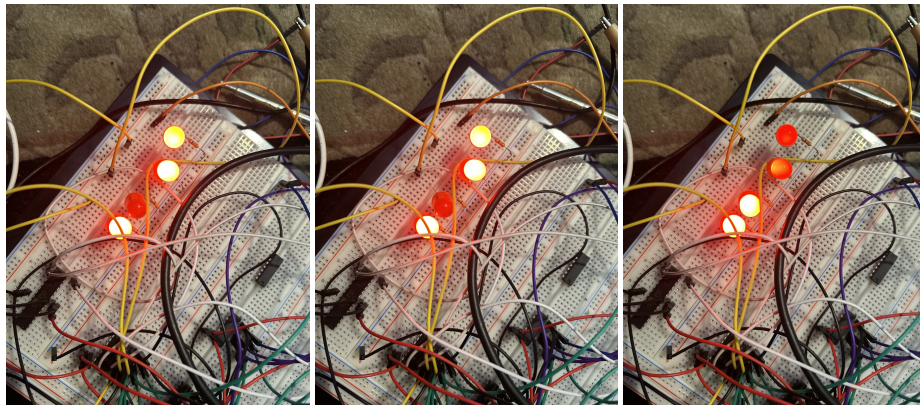












8 warum unsere LED so dunkel war...

Kriechströme?, fehlende pull-up, pull-down-widerstände, anything else? nein, ganz einfach:

ein kleines programmiergerät liefert gerne Strom und Spannung, sprich leistung für einen atmega8. in der schaltung, sind aber mindestens vier weitere TTL-Gatter-IC. die leistung steigt immens. was sich am ausgang also am ende der schaltung bemerkbar macht. anderes netzteil, macht gleich bessere led.

9 warum der abstand der photos nun mal keinen sinn macht

Zeitintervalle:

- eigentlich kommen graphen aus der mathematik. und sind seid euler eine wissenschaft für sich. die konsekutive nummerierung von zuständen lässt sich in der mathematik ausdrücken, aber hier ist das Thema Zeit nicht mehr frage nummer eins
- in der Physik rechnen wir mit Wellenlängen, $f = \frac{1}{T}$. zum beispiel, was den Zusammenhang zwischen Periode und Frequenz betrifft, aber eine andere Formel sagt das: $\lambda = \frac{c}{f}$, es gibt aber noch eine dritte, müsste man so sagen, weil die erste, definiert nicht, was ein herz ist.
- in der physik hat eine welle noch eine räumliche ausdehnung, in der graphentheorie, ist graph, graph und beim automaten, na ja, graph oder automat = schaltwerk, räumlich spielt hier keine rolle, normalerweise. ok.
- wenn man jetzt allerdings ganz klug ist so ein händchen für sachen hat, die nun mal einfach seltsam sind. schauen sie meinen drucker an. ein brother. der hat da hinten so eine hebel, darin landet das papier. was wenn nicht aufgeklappt, es liegt alles rum, beim durchzug. leider weiss ich jetzt nicht mehr, welche genau von welcher seite eingescant wurden. also, schon ein

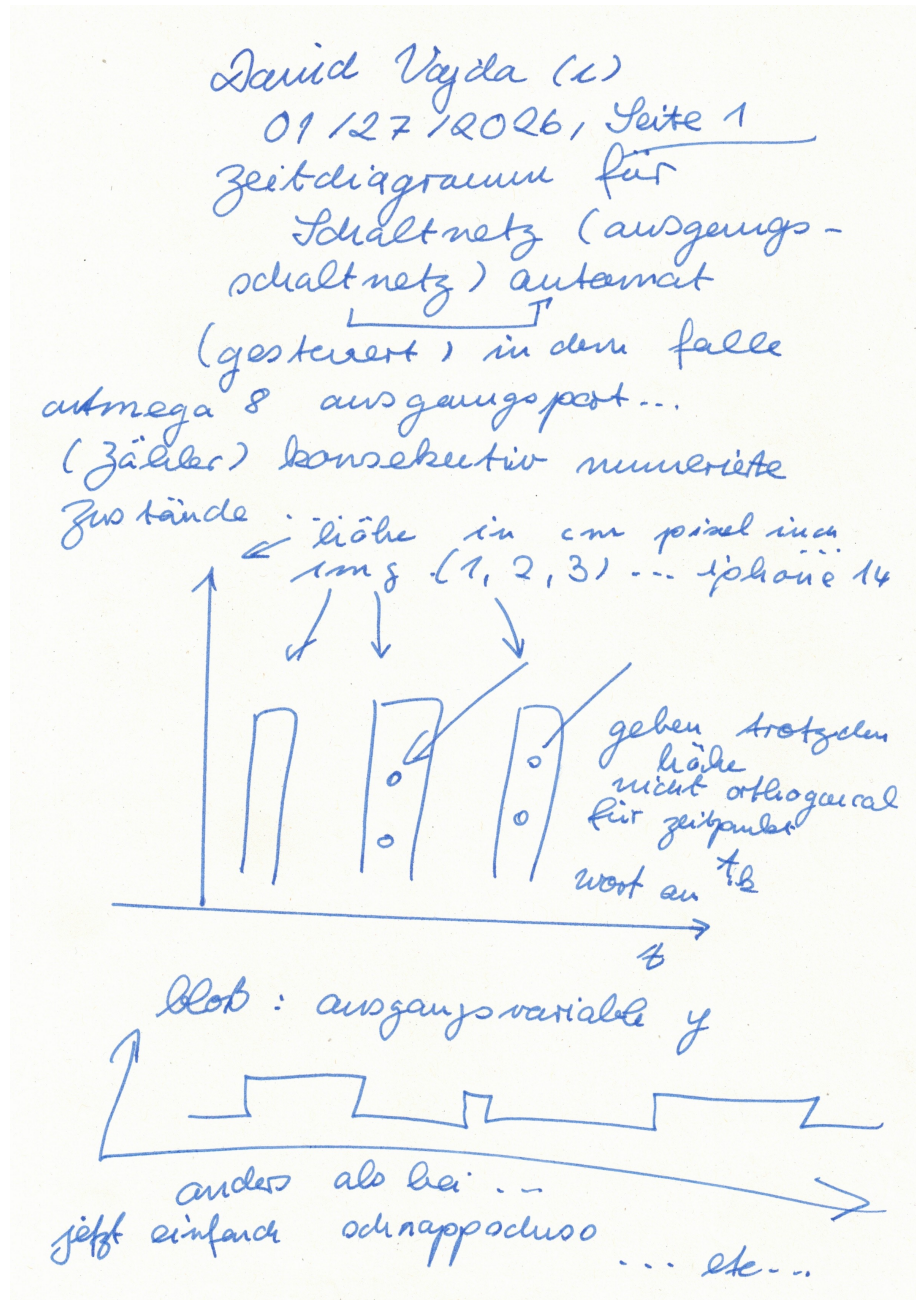
bisschen gucken, was der hersteller sagt. auch kopieren will gelernt sein.
ok. nächstes: wenn ich jetzt mein smartphone auf den sekundentakt abstelle, mache ich es wiederum falsch.

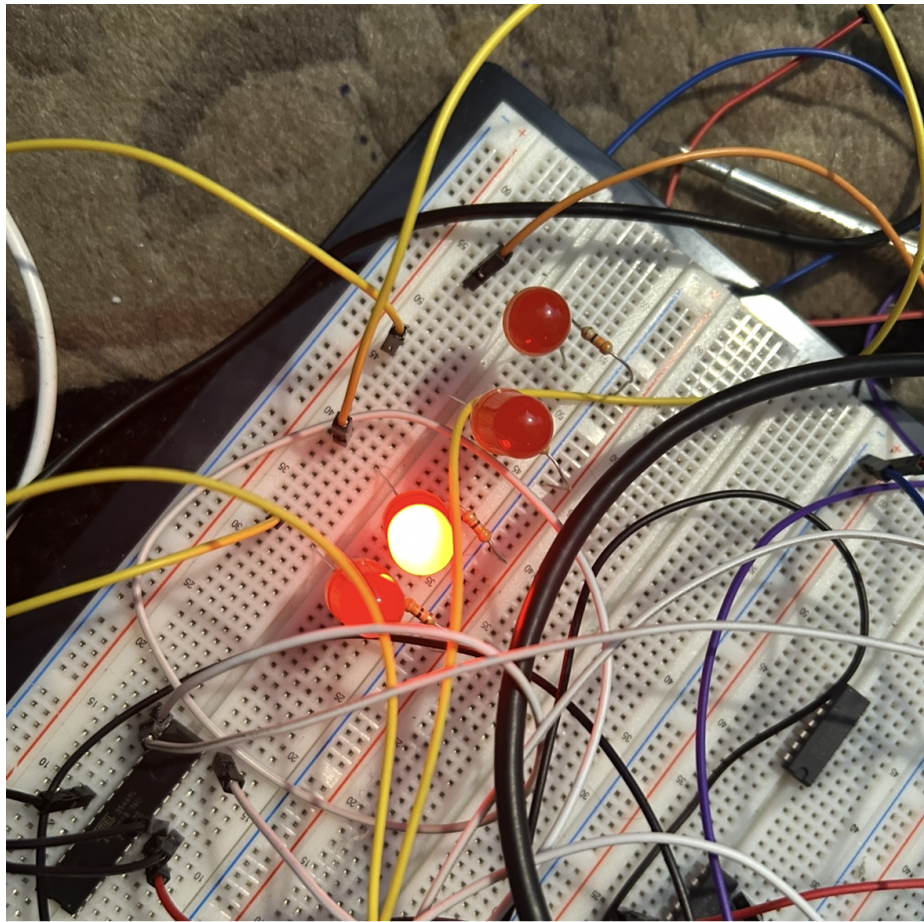
hier das naechste bild. schauen wir uns ein mal iPhone 14, photos an. hier steht neben Länge und Breitengrad, im irdischen sinne, auch noch der so ziemlich genaue zeitpunkt drin.

wenn man sich jetzt so ein schönes zeitdiagramm vorstellt (man könnte das mit dem stichprobenumfang beim pkw und lkw vorbeifahren vergleichen), in der wahrscheinlichkeitsrechnung, nur hier wird leider das papier viel zu breit. sortieren wir mit raffinierten, aber auch raffiniert einfachen bash programmen und linux, den genauen zeitpunkt raus, wir könnten nun ohne kenntnis, wann der atmega8 anfang, diese in eine physikalische zeitachse einordnen, zum beispiel mit automatisch generierten excel dateien oder so.

interessant ist, fast, wie bei Rene Descartes, verständnis für rechtshändige orthogonale Koordinatensystem nun zu stande käme. Abszisse und ordinate. passt in die wahrscheinlichkeitsrechnung, mit LED machbar. doch man könnte jetzt sogar über den namen nicht orthogonal nachdenken, indem man den binären 3-Bit Eingangsvektor nehme. wie auch immer.

9.1 hier die idee, bildhaft





Untertitel hinzufügen

Dienstag, 27. Januar 2026 um 11:57

[Anpassen](#)

IMG_8847

Apple iPhone 14

HEIF



Hauptkamera — 26 mm $f1.5$

12 MP • 3024 × 4032 • 1,6 MB

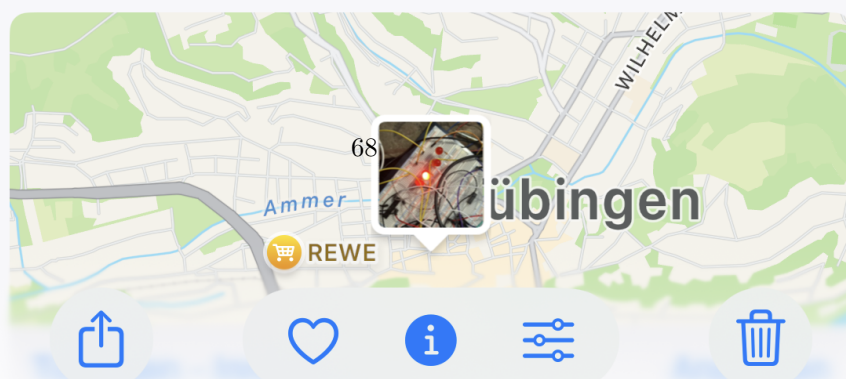
ISO 400

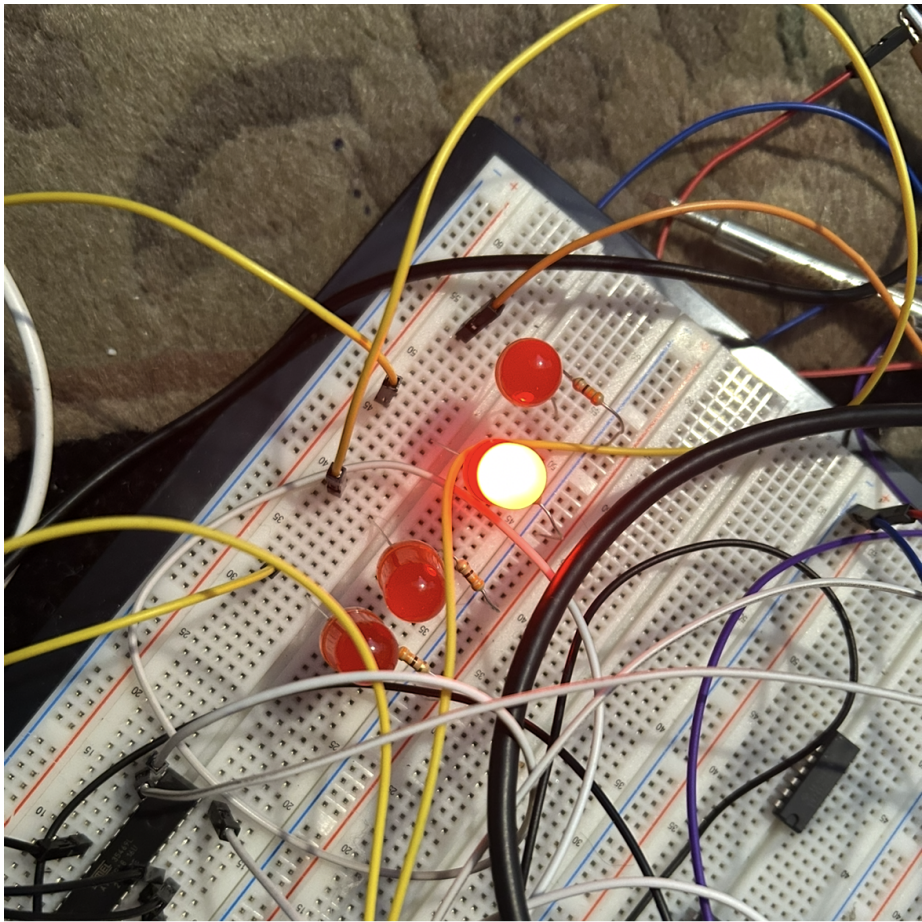
| 26 mm

| 0 ev

| $f1.5$

| 1/50 s





Untertitel hinzufügen

Dienstag, 27. Januar 2026 um 11:57

[Anpassen](#)

IMG_8846

Apple iPhone 14

HEIF



Hauptkamera — 26 mm $f1.5$

12 MP • 3024 × 4032 • 1,7 MB

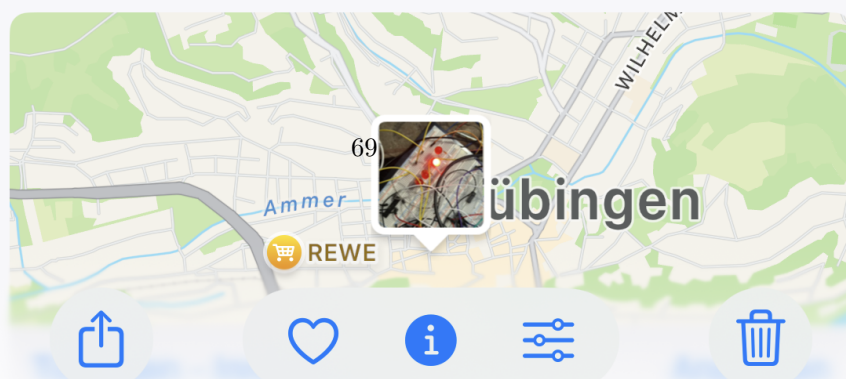
ISO 400

26 mm

0 ev

$f1.5$

1/50 s



10 nun mit oszilloskop

10.1 hier der quelltext

```
;; (c) david vajda
;; https://www.vajda-breadboard.de
;; 01/27/2026
;; m8 led count 3 bit or more - ...
;; ... human eye 1s visible clk ...

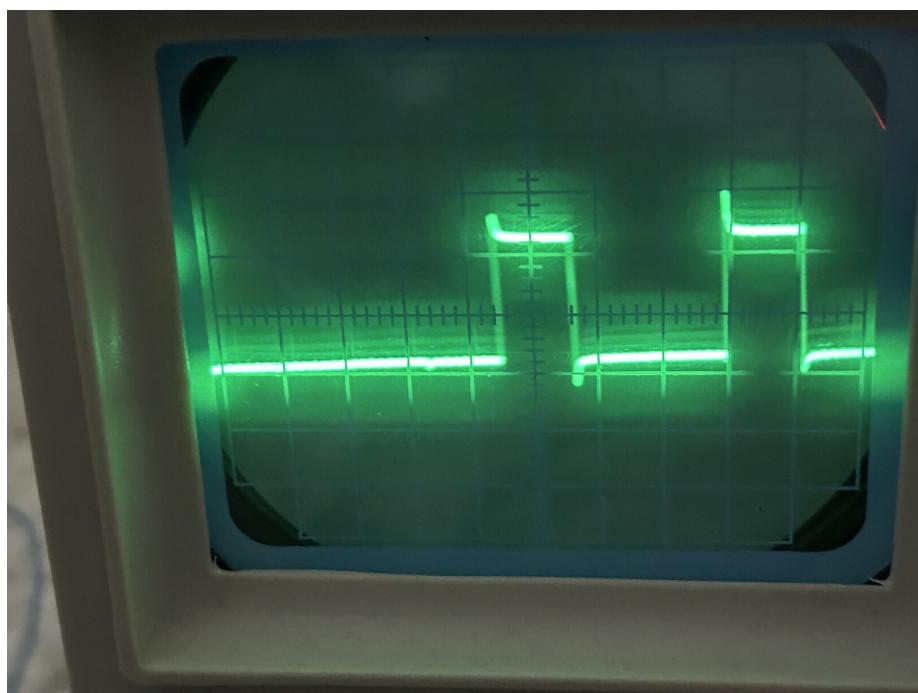
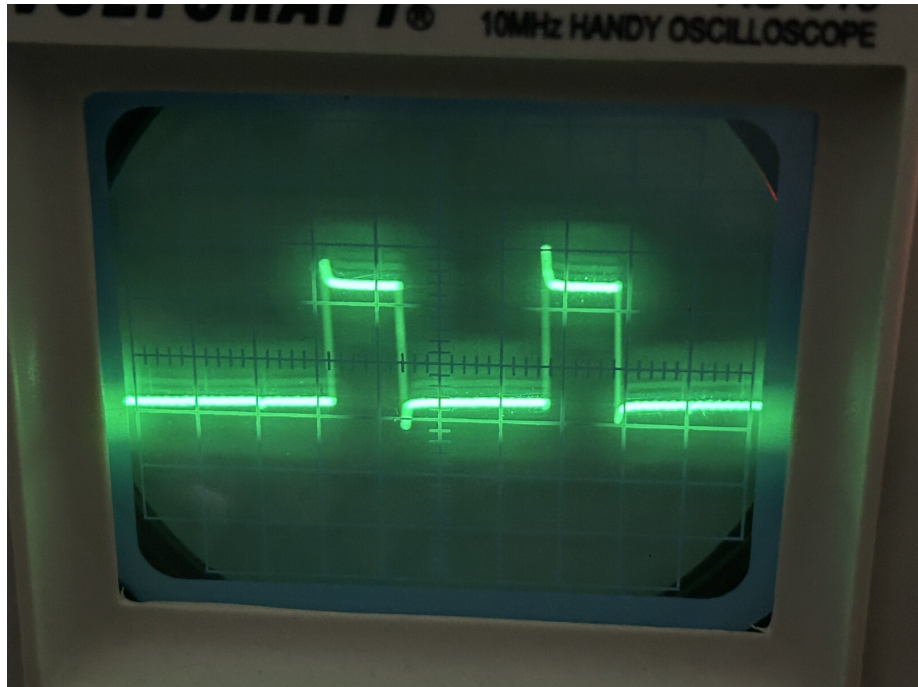
.include "m8def.inc"

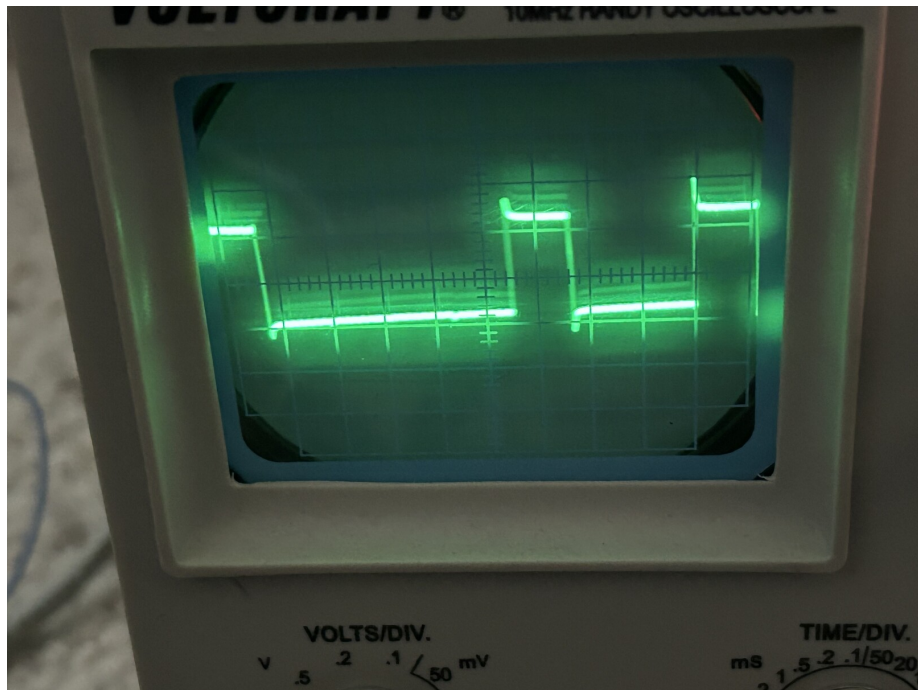
ldi r16, HIGH (RAMEND)
out SPH, r16
ldi r16, LOW (RAMEND)
out SPL, r16
ldi r16, 0xff
out DDRB, r16

ldi r16, 0x00
l1:
out PORTB, r16
;rcall f1
inc r16
rjmp l1

f1:
push r16
push r17
push r18
ldi r16, 0xff
l2:
ldi r17, 0xff
l3:
ldi r18, 0x04
l4:
dec r18
brne l4
dec r17
brne l3
dec r16
brne l2
pop r18
pop r17
pop r16
ret
```

10.2 das oszi





11 eeprom - 256k

```
:08000000FF00000000FF0000FA
:00000001FF
```

mit dem prom machen wir drei sachen:

1. DNF
2. Siebensegment Anzeige
3. Vierzehensegment Anzeige.

mit beschreibung:

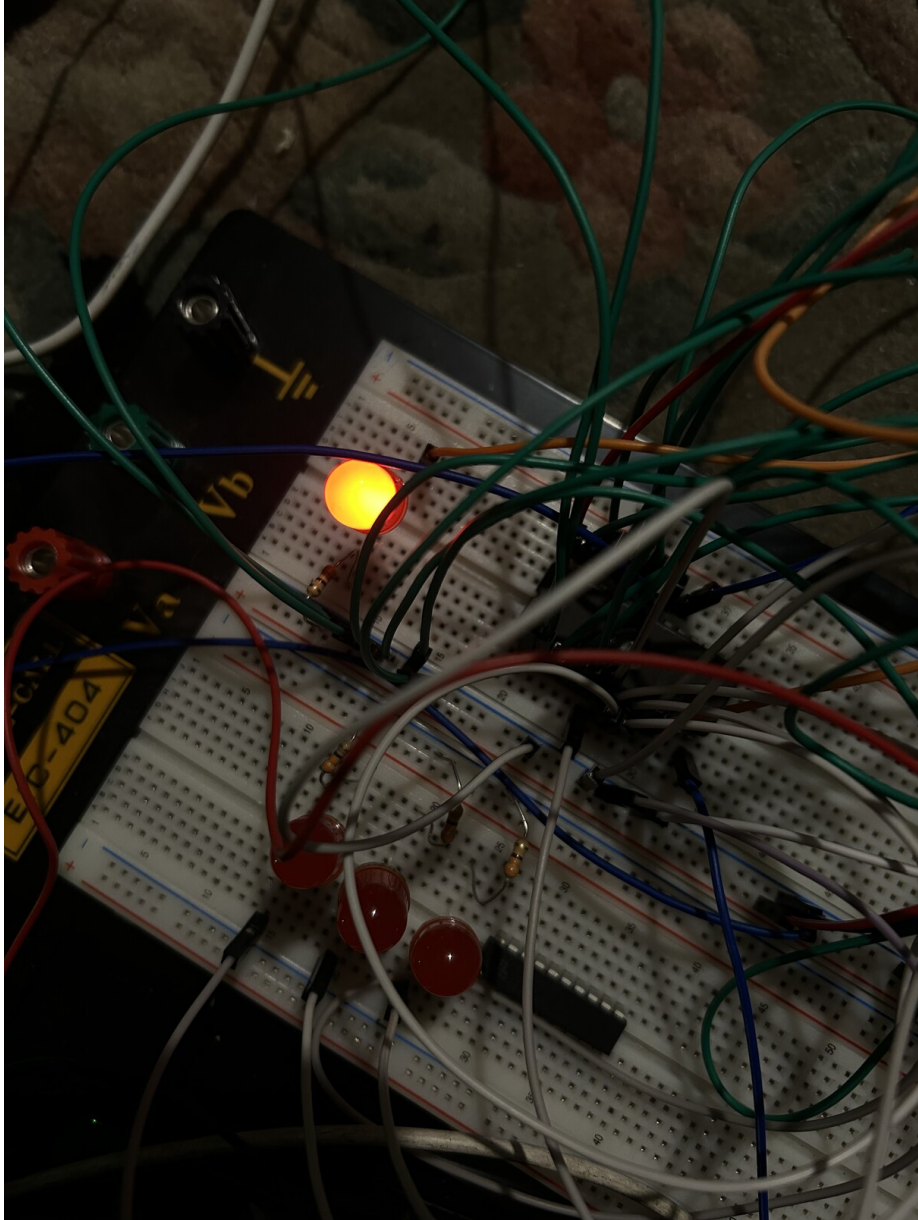
1. für eine DNF brauchen wir nicht automatisch einen GAL/PAL, ein PROM genügt. weil das ein schaltnetz ist, mit Eingängen (Eingangsvektor) $x_n, \dots, x_1, x_0$. wenn im speicher eine 0 gespeichert, dann ist die Ausgangsvariable y gleich null, wenn eine 1, dann gleich eins. man kann auch einen Ausgabevektor von zum Beispiel 8 Bit nehmen. Indem man verschiedene Worte speichert. Der Eingangsvektor ist an den Variablen entsprechend invertiert, über die Adresse 00000000, 00000001, 00000010,
2. Siebensegment Anzeige
3. Vierzehensegment Anzeige.

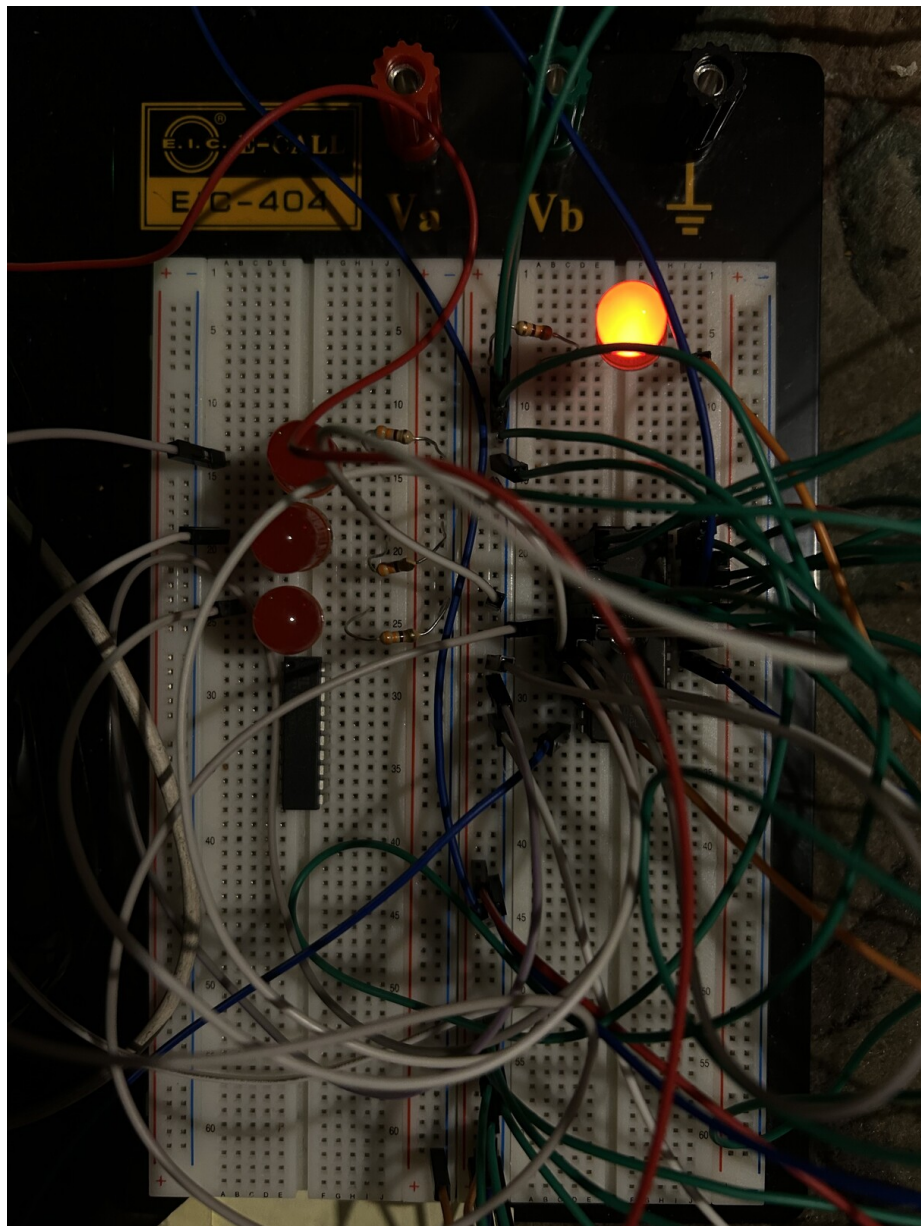
Im folgenden folgt auch noch ein GAL.

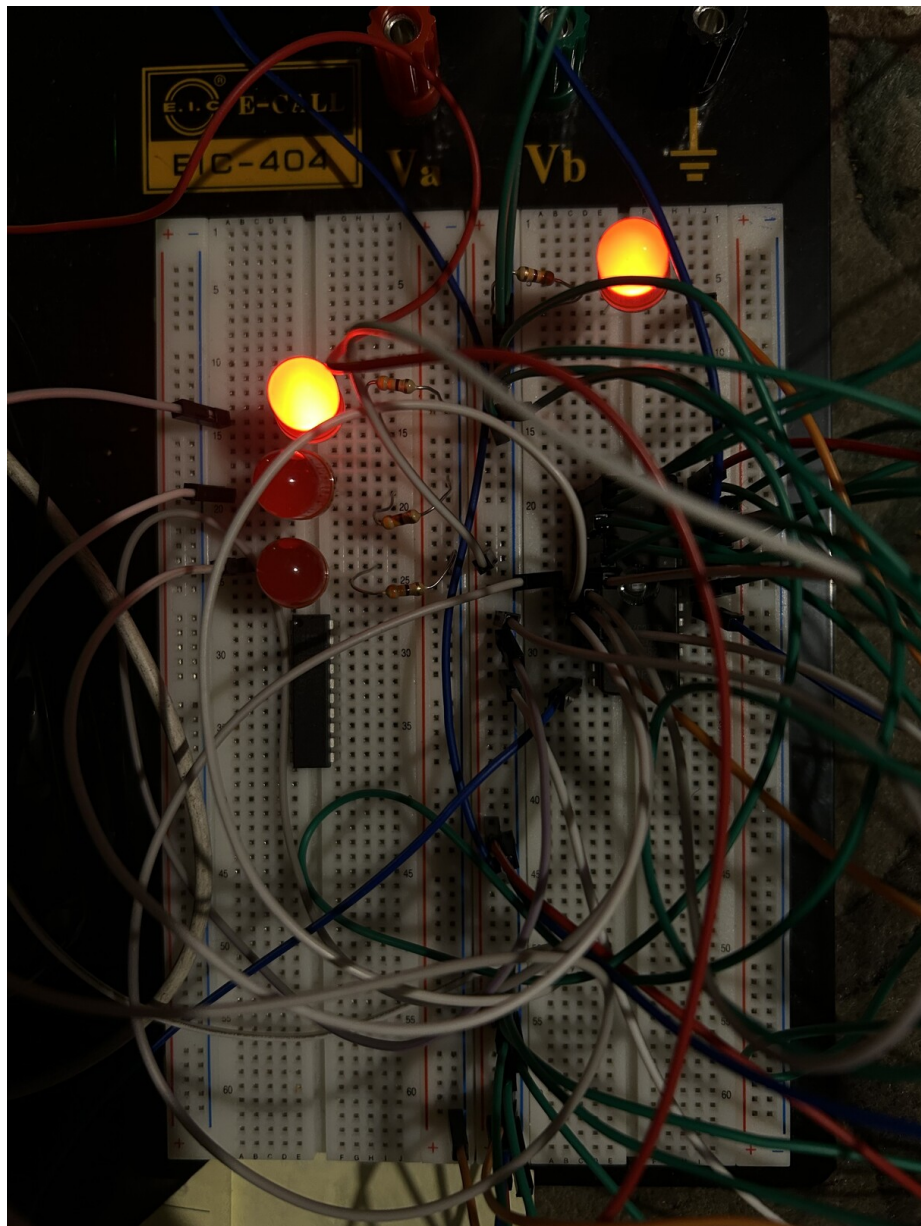
im ersten schritt habe ich ein funktionierendes Experiment durchgeführt. dieselbe Wahrheitstafel. dafür habe ich keinen binärcode erzeugt, sondern den Hex Code vom Batrox Prog Express genommen.

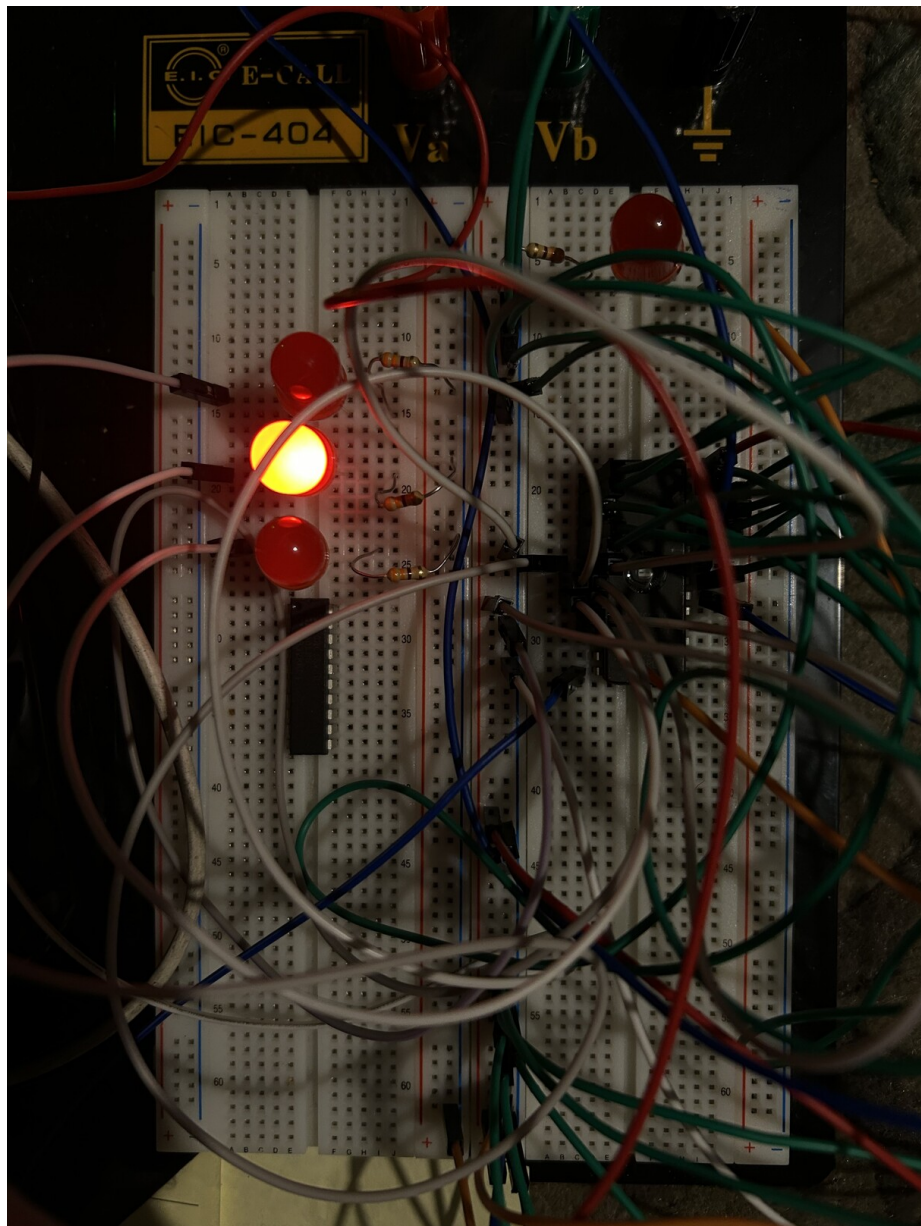
ich habe festgestellt die datei erzeugt keinen binärcode. ich habe es noch anders probiert. das problem am anfang war, dass bei 111 eingabevektor, trotzdem eine 1 bei y erschien, ich vermute das lag daran, das sich der code nicht wiederholt hat. was unlogisch ist. ich habe es noch ein mal verändert, und daraufhin veränderte sich das verhalten an anderer stelle.

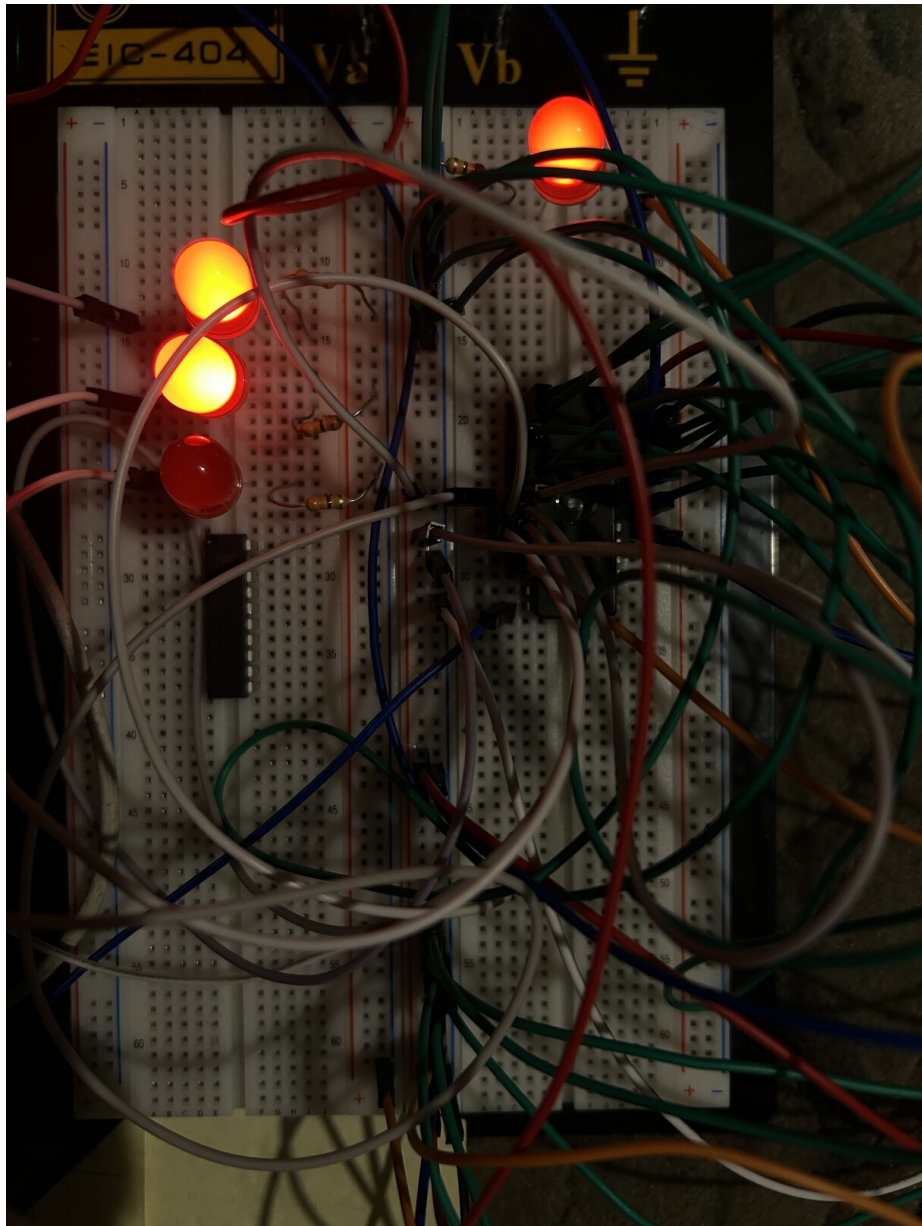
12 erster versuch

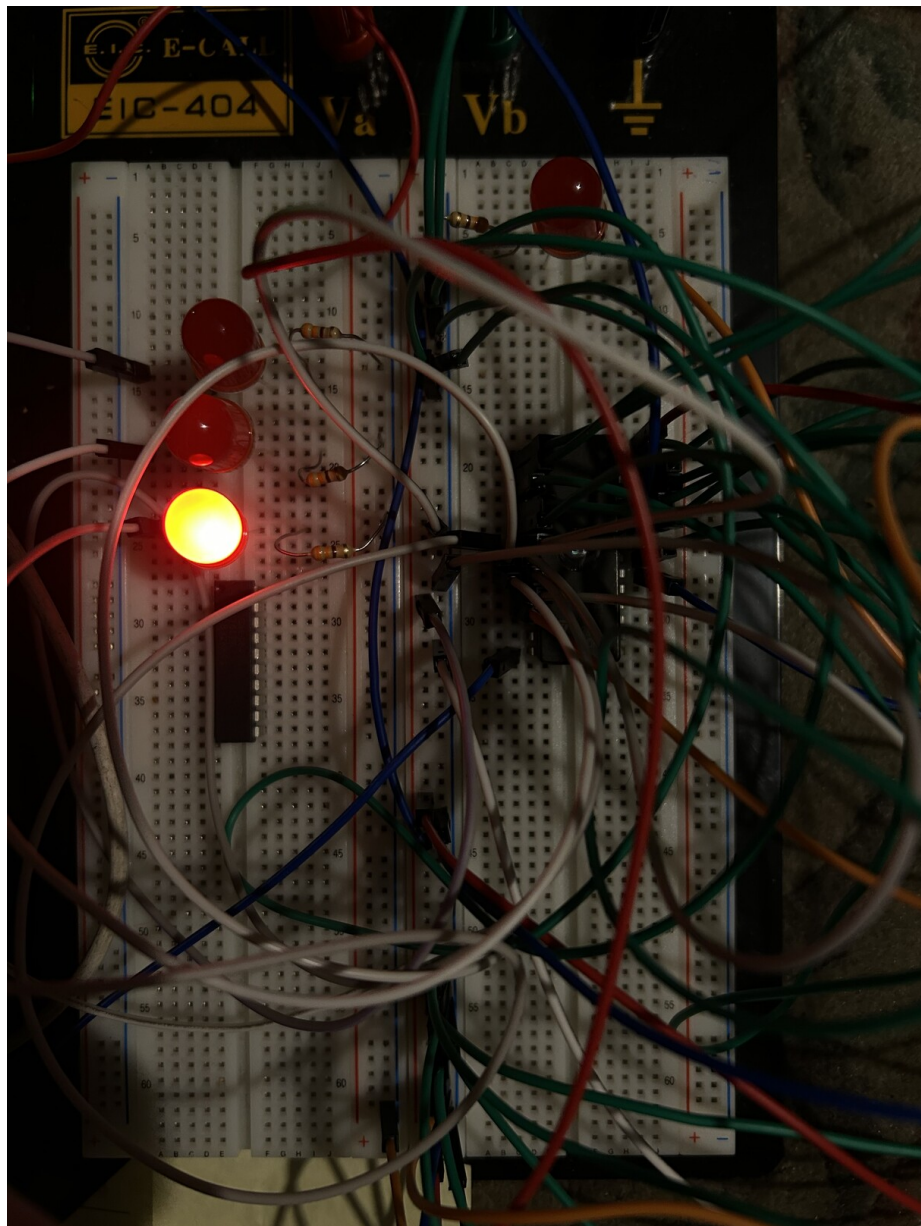


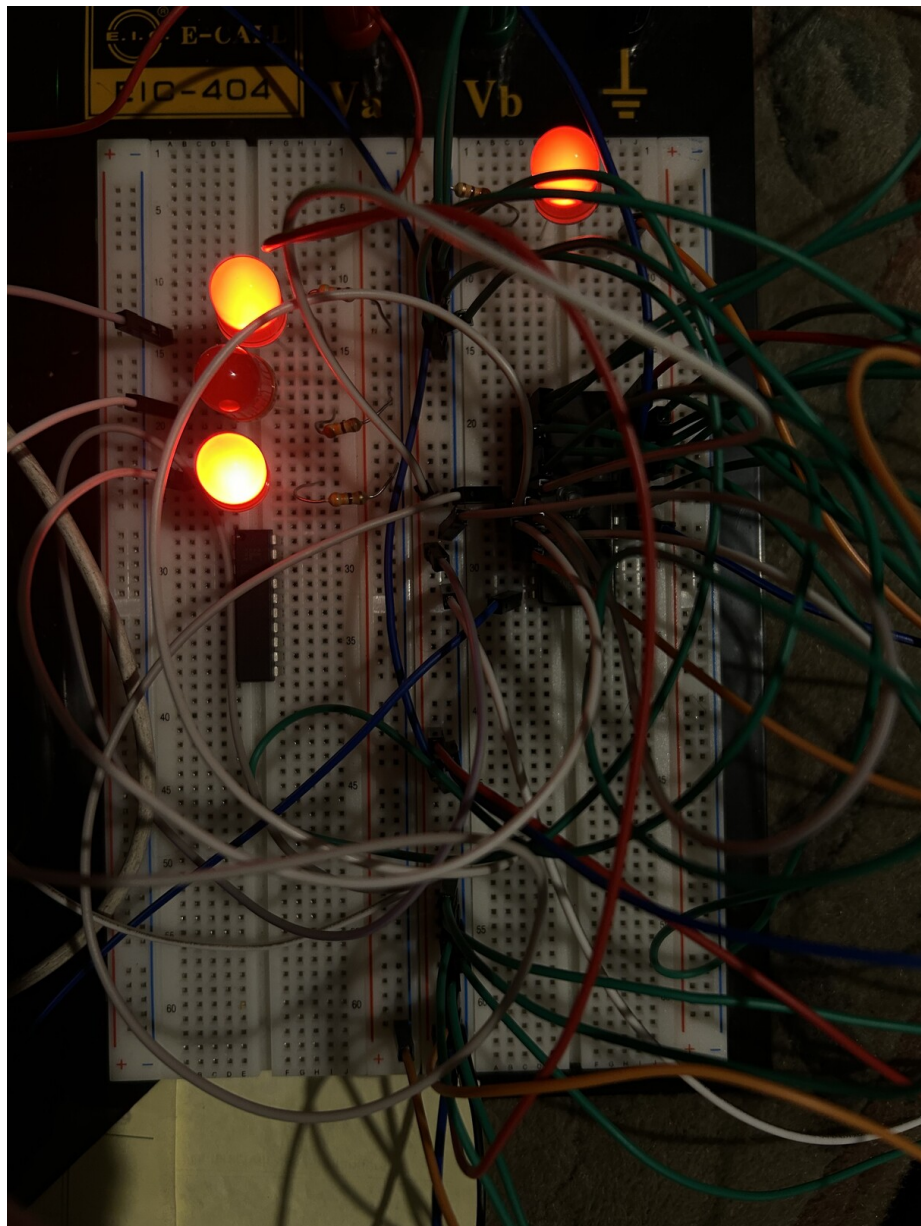


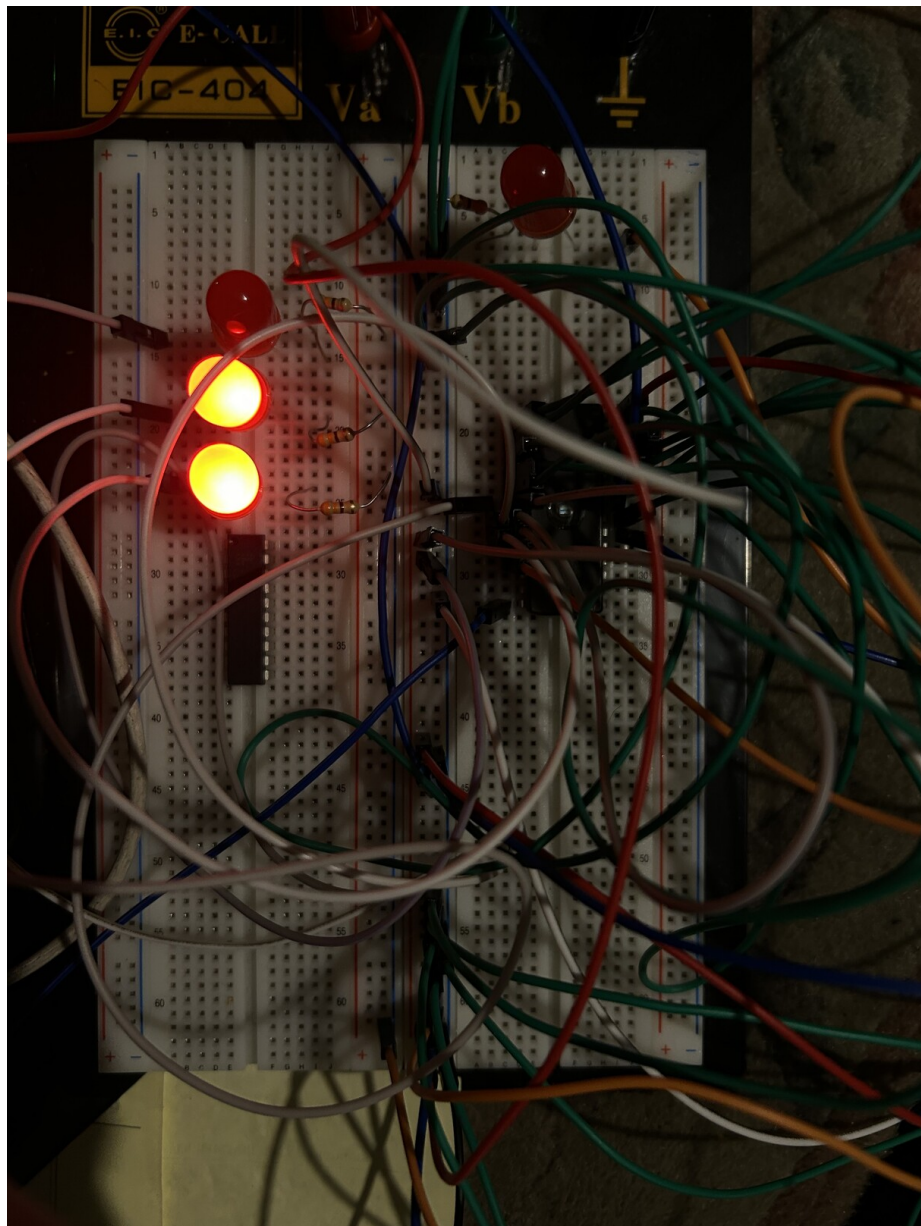


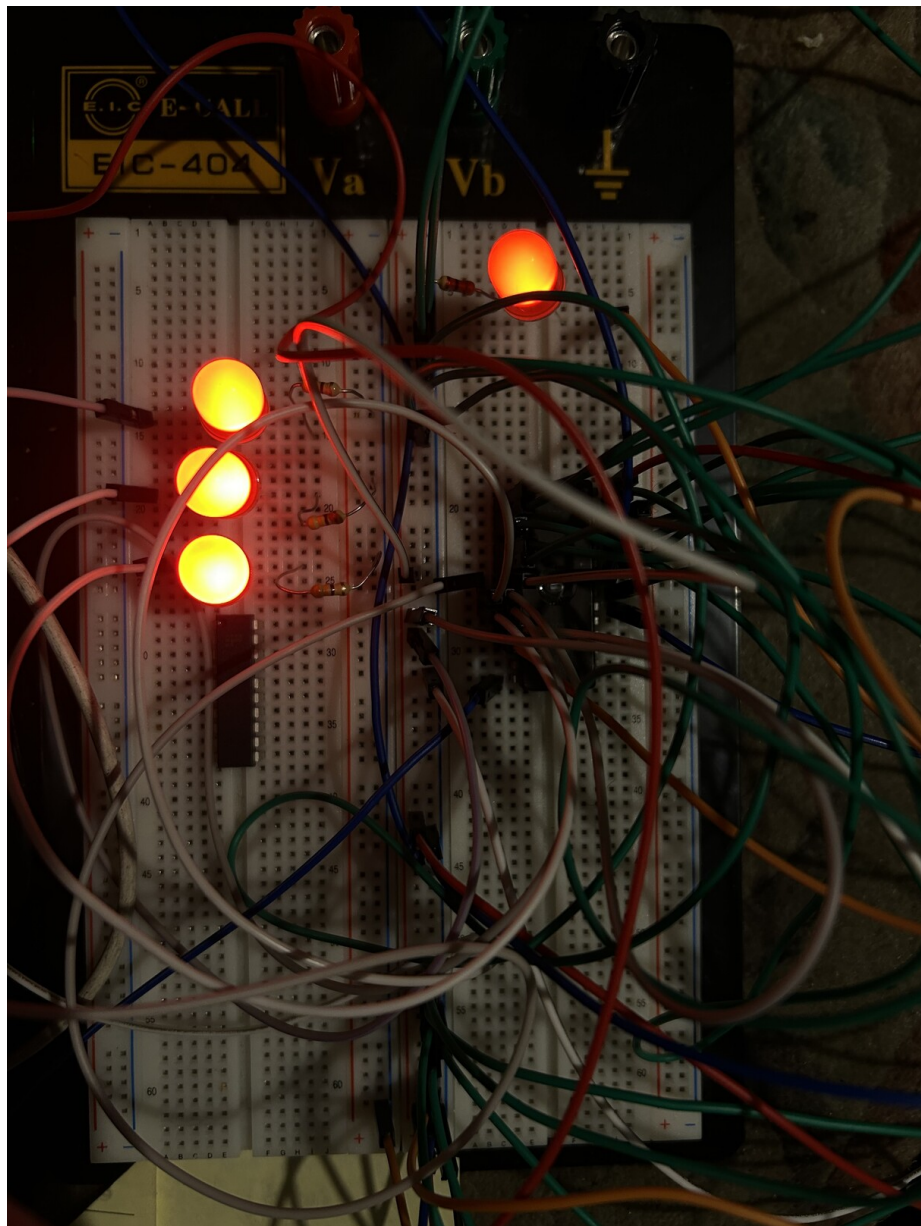












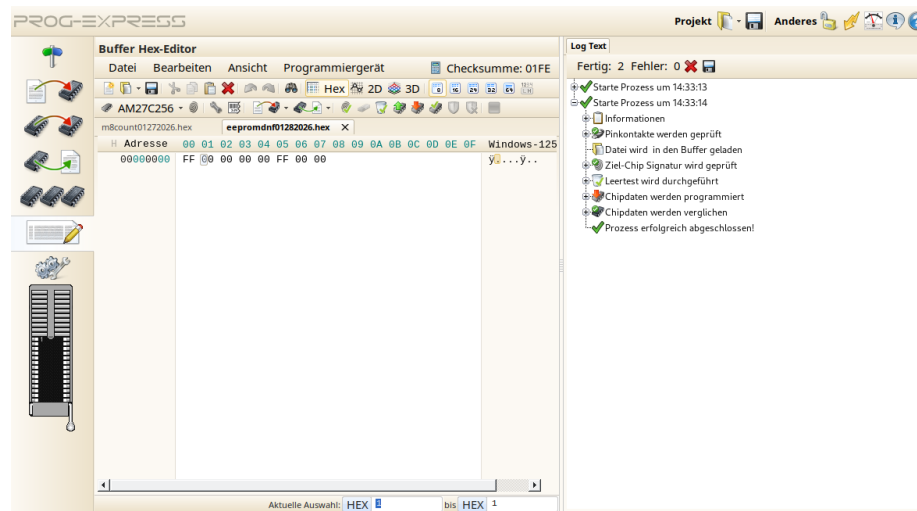
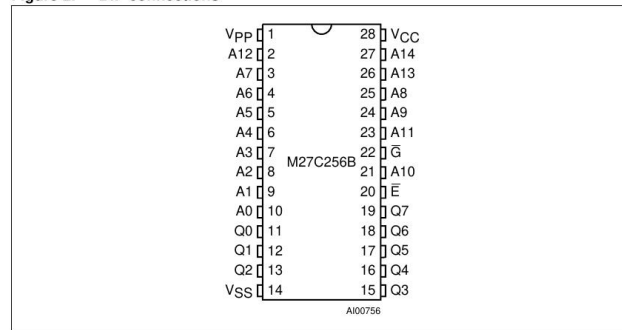


Table 1. Signal names

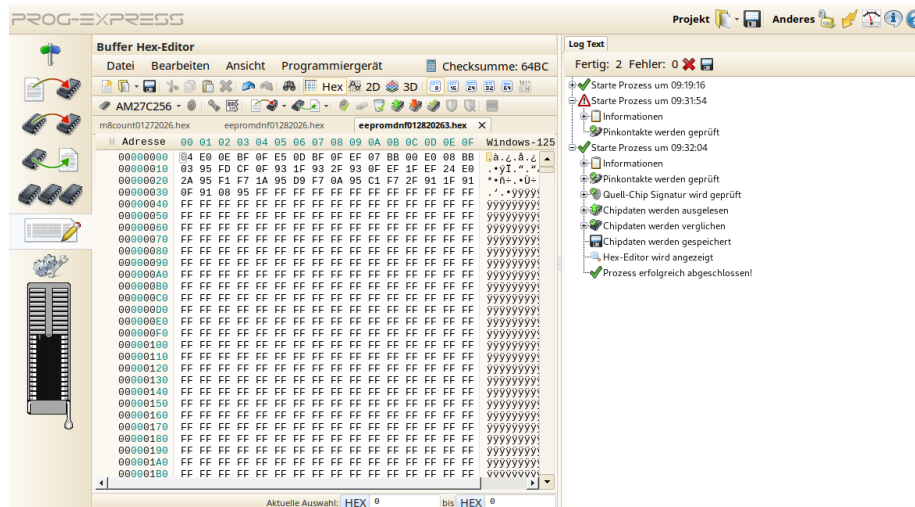
A0-A14	Address Inputs
Q0-Q7	Data Outputs
E	Chip Enable
$\bar{G}$	Output Enable
V _{PP}	Program Supply
V _{CC}	Supply Voltage
V _{SS}	Ground
NC	Not Connected Internally
DU	Don't Use

Figure 2. DIP connections



13 zweiter versuch

ich habe die anzahl der ausgaben vervielfacht. dabei ist mir aufgefallen, dass die hex dateien sich unterscheiden. das müssen wir wissen, von reinen binärfolgen.



```

Datei Bearbeiten Ansicht Lesezeichen Module Einstellungen Hilfe
Neues Unterfenster Ansicht teilen Kopieren Einfügen Suchen

0000070 3030 3030 4631 0a46 303a 3038 3030 3030
0000080 4630 3046 3030 3030 3030 4630 3046 3030
0000090 4630 0a41 303a 3030 3030 3030 4631 0a46
00000a0 303a 3038 3030 3030 4630 3046 3030 3030
00000b0 3030 4630 3046 3030 4630 0a41 303a 3030
00000c0 3030 3030 4631 0a46 303a 3038 3030 3030
00000d0 4630 3046 3030 3030 3030 4630 3046 3030
00000e0 4630 0a41 303a 3030 3030 3030 4631 0a46
00000f0 303a 3038 3030 3030 4630 3046 3030 3030
0000100 3030 4630 3046 3030 4630 0a41 303a 3030
0000110 3030 3030 4631 0a46 303a 3038 3030 3030
0000120 4630 3046 3030 3030 3030 4630 3046 3030
0000130 4630 0a41 303a 3030 3030 3030 4631 0a46
0000140
david@www001:~/vhd101272026$ hexdump eepromdnf01282026.hex
0000000 303a 3038 3030 3030 4630 3046 3030 3030
0000010 3030 4630 3046 3030 4630 0a41 303a 3030
0000020 3030 3030 4631 0a46
0000028
david@www001:~/vhd101272026$

```

```
#!/bin/bash
```

```
# (c) David Vajda
# 01/29/2026
# PROM DNF HEX CAT 8 Times
```

```
i=0
while [ $i -lt 8 ]
do
    cat eepromdnf01282026.hex >> eepromdnf012820262.hex
    i=$((i+1))
done
```

```
:08000000FF00000000FF0000FA
:00000001FF
```

```
:20000000FF00000000FF0000FF00000000FF0000FF00000000FF0000E8
```

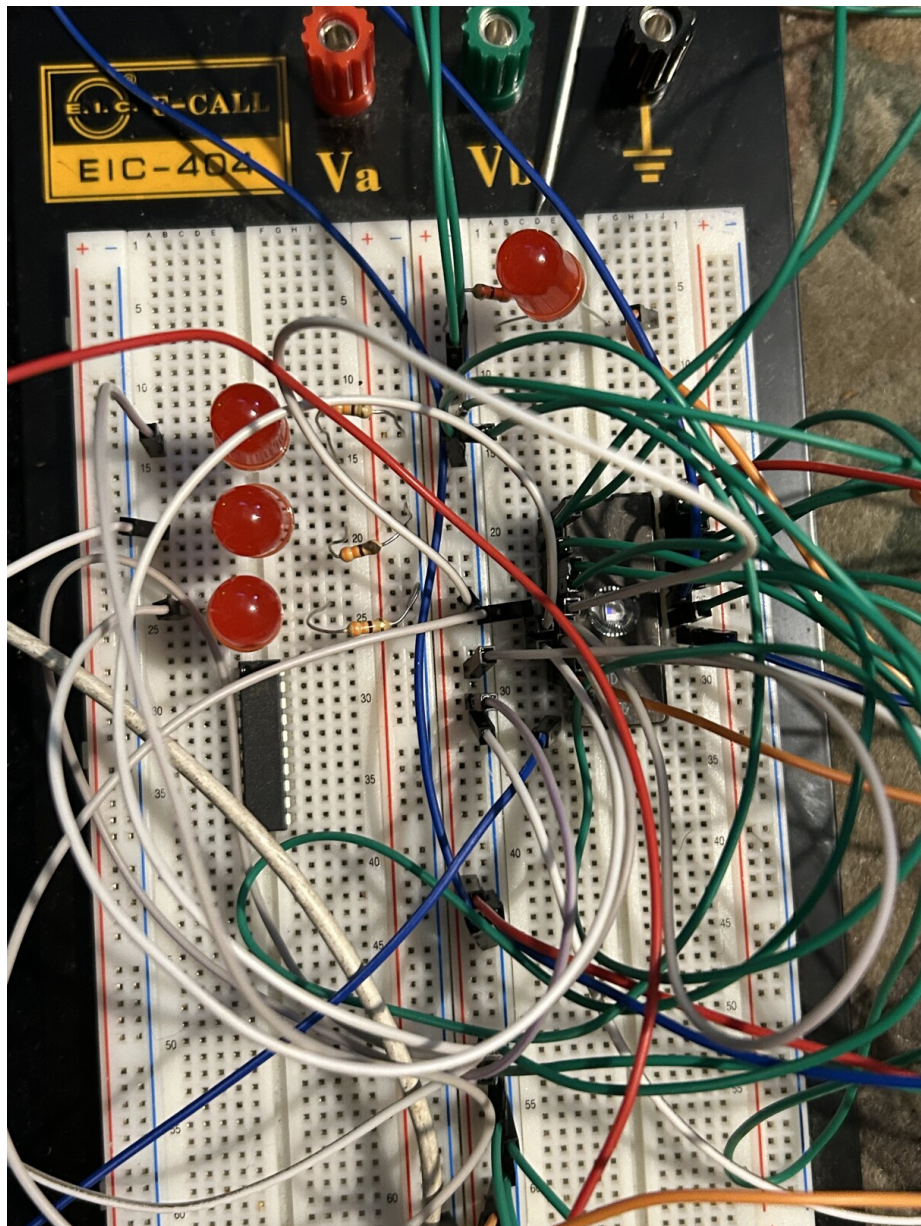
```

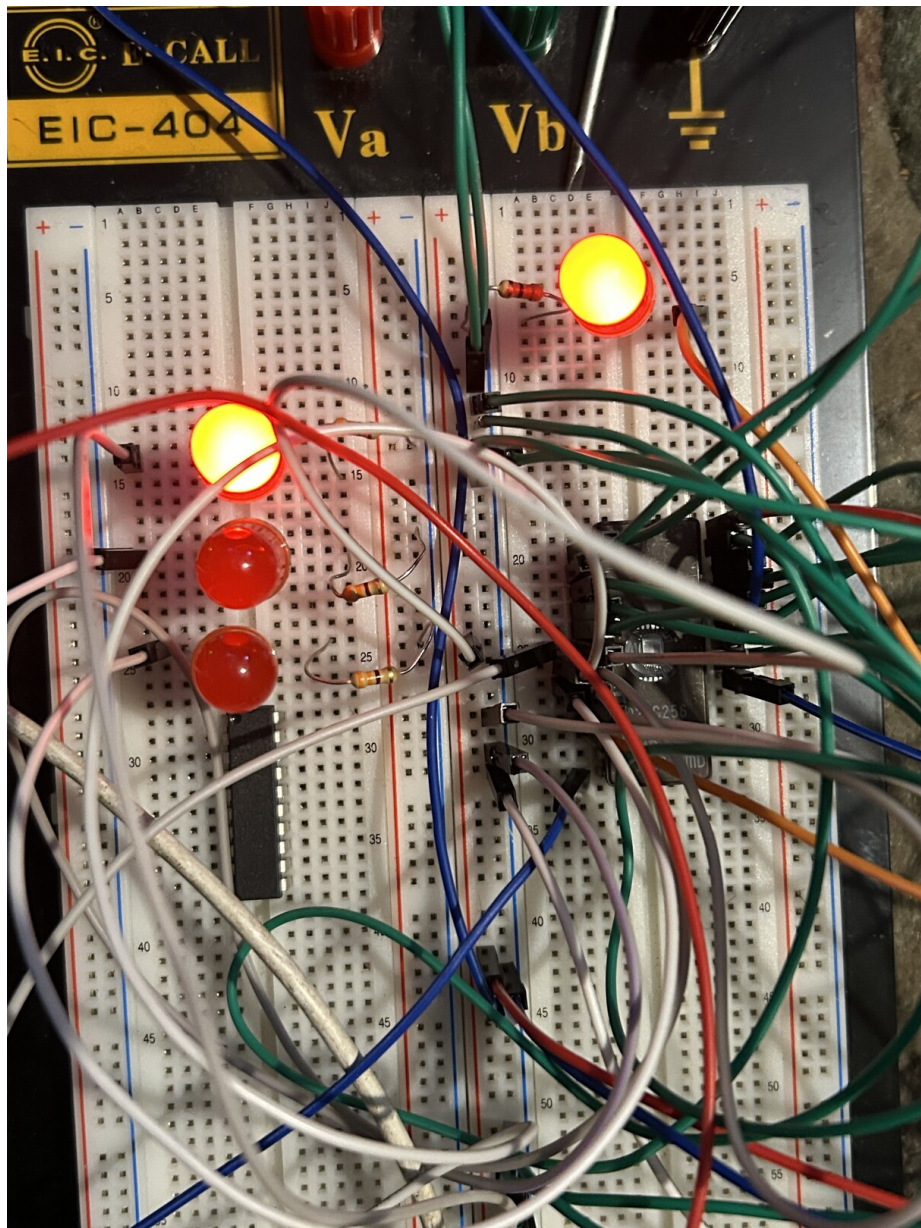
:20002000FF00000000FF0000FF00000000FF0000FF00000000FF0000FF00000000FF0000C8
:10004000FF00000000FF0000FF00000000FF0000B4
:00000001FF

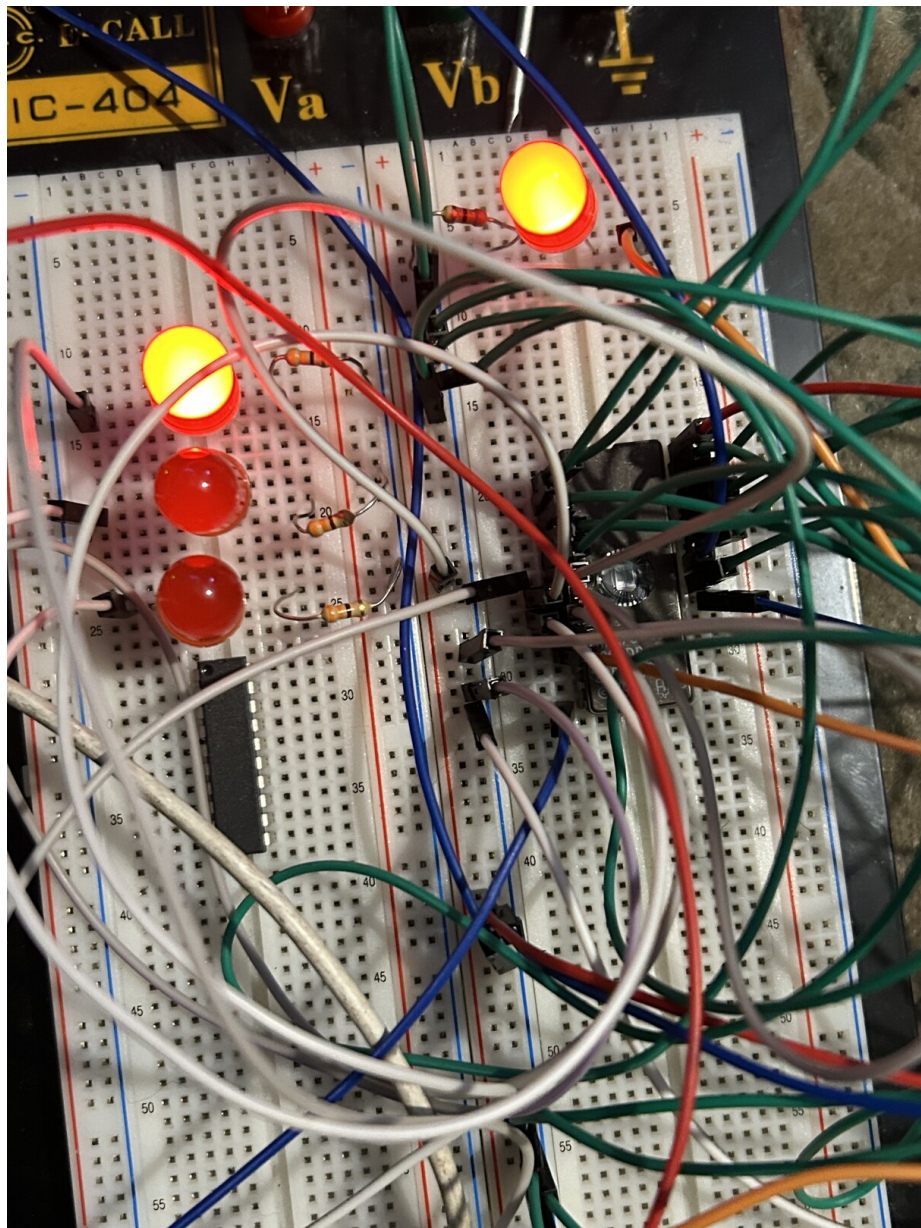
:2000000004E00EBF0FE50DBF0FEF07BB00E008BB0395FDCF0F931F932F930FEF1FEF24E082
:200020002A95F1F71A95D9F70A95C1F72F911F910F910895FFFFFFFFFFFFFFFFFFFFFFFFFA2
:20004000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC0
:20006000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFA0
:20008000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF80
:2000A000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF60
:2000C000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF40
:2000E000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF20
:20010000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
:20012000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFDF
:20014000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFDF
...
:207F8000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF01
:207FA000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE1
:207FC000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC1
:207FE000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFA1
:00000001FF

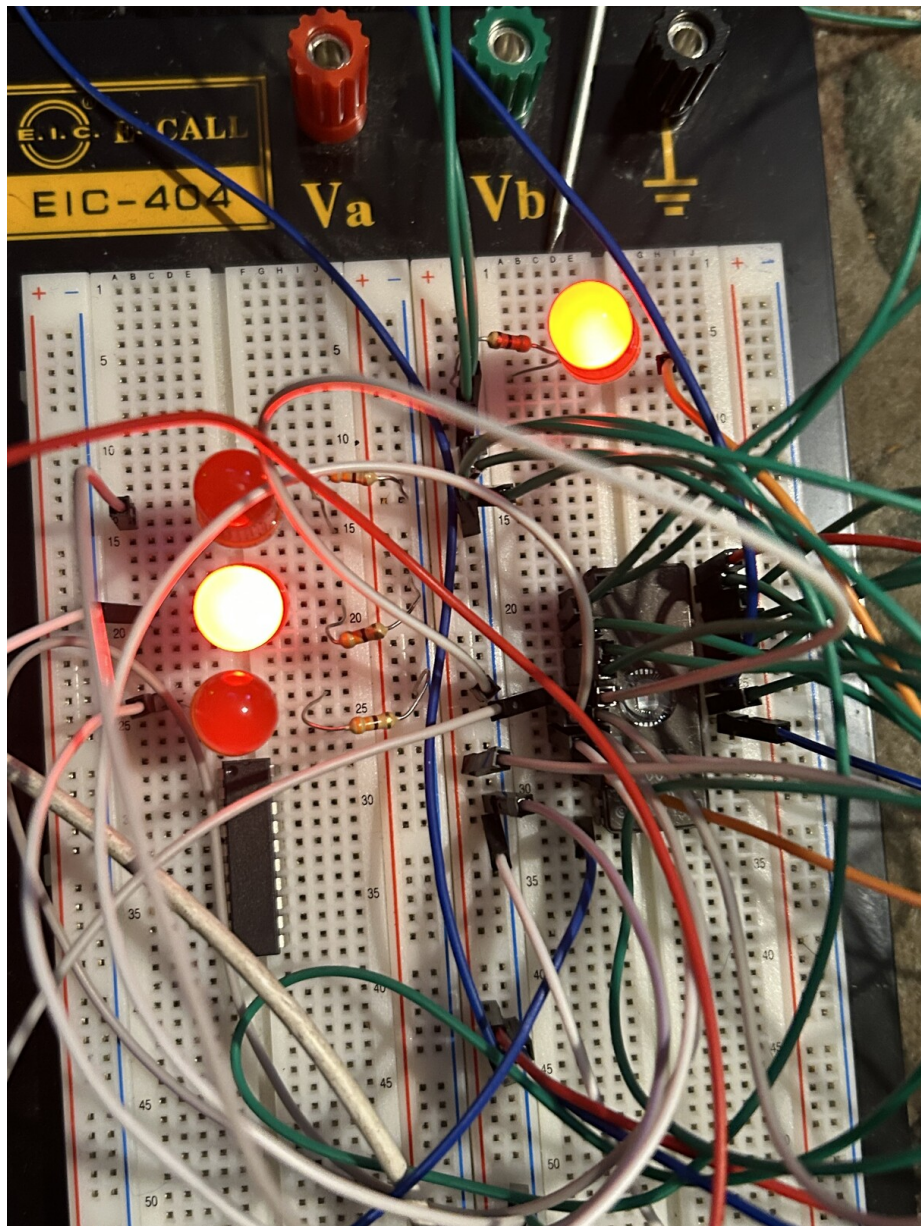
```

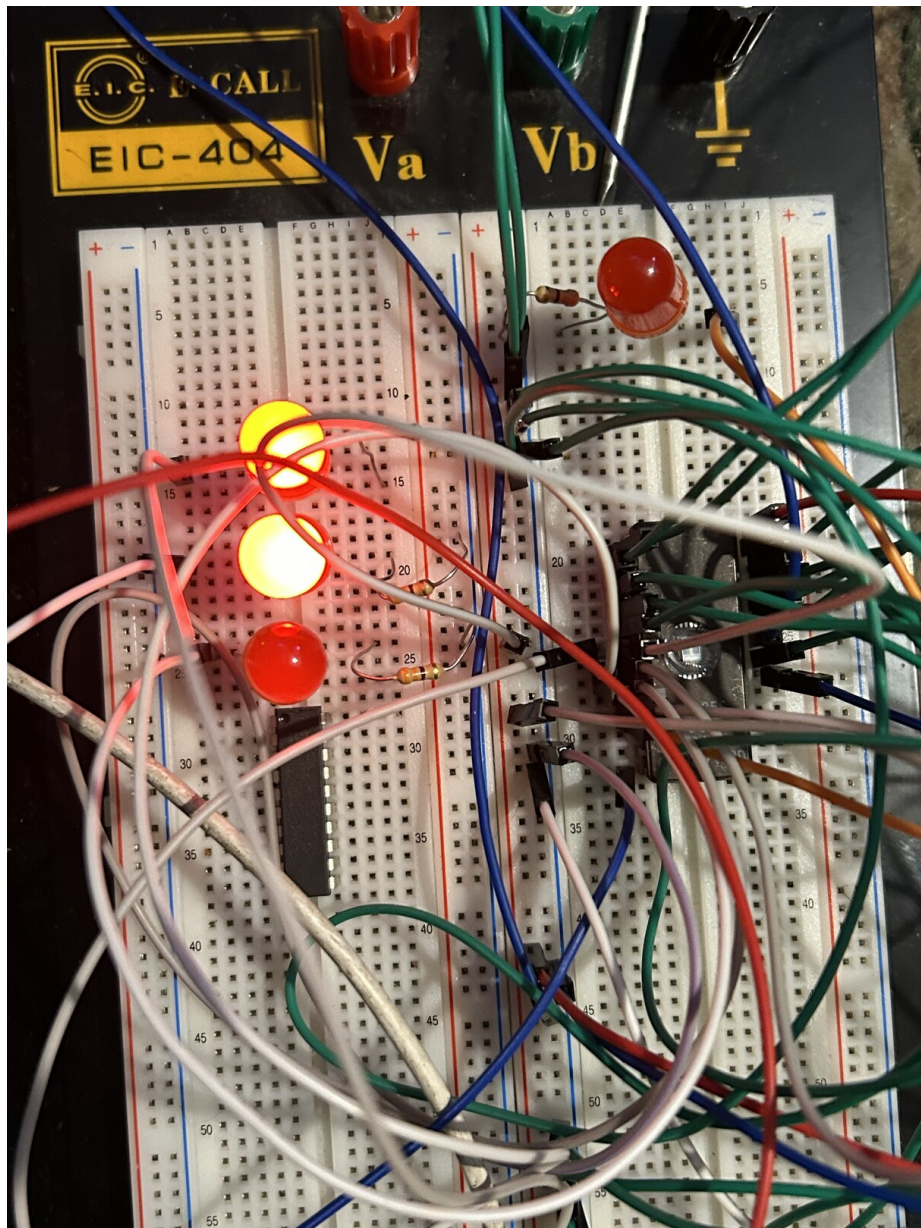
beim wiederholten male kam das heraus. ich betone ich verwende, eeprom und nicht eeprom. diese haben ein fenster, deswegen ist der löschprozess dargestellt.

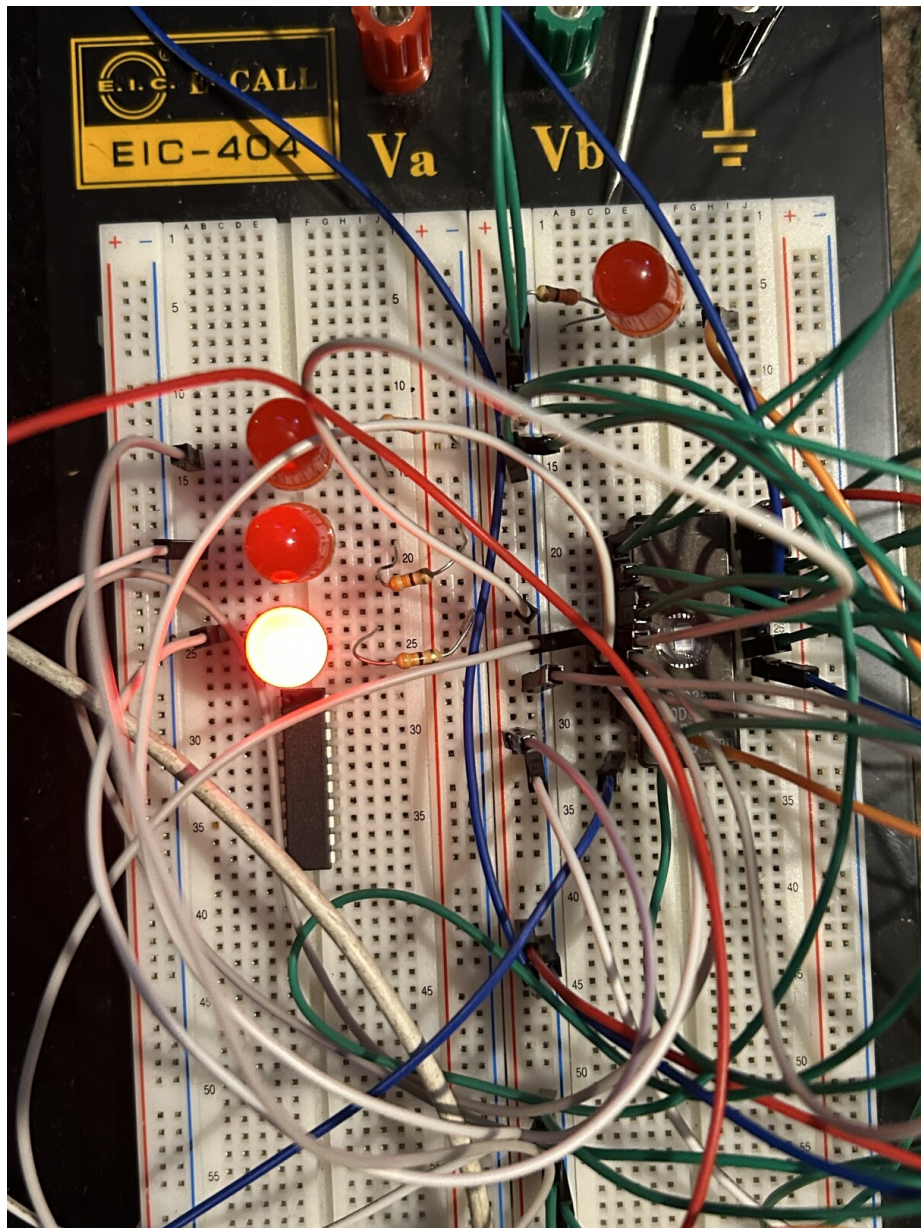


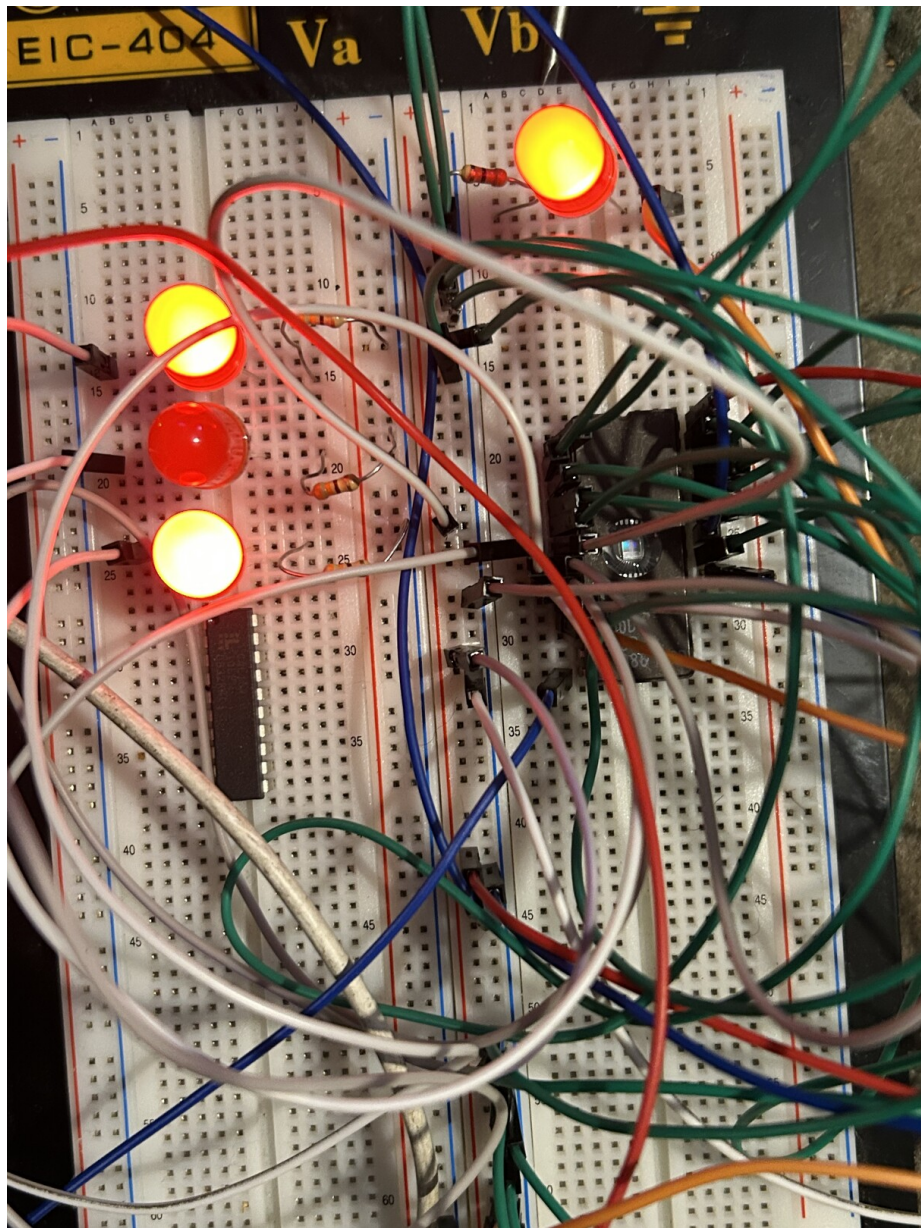


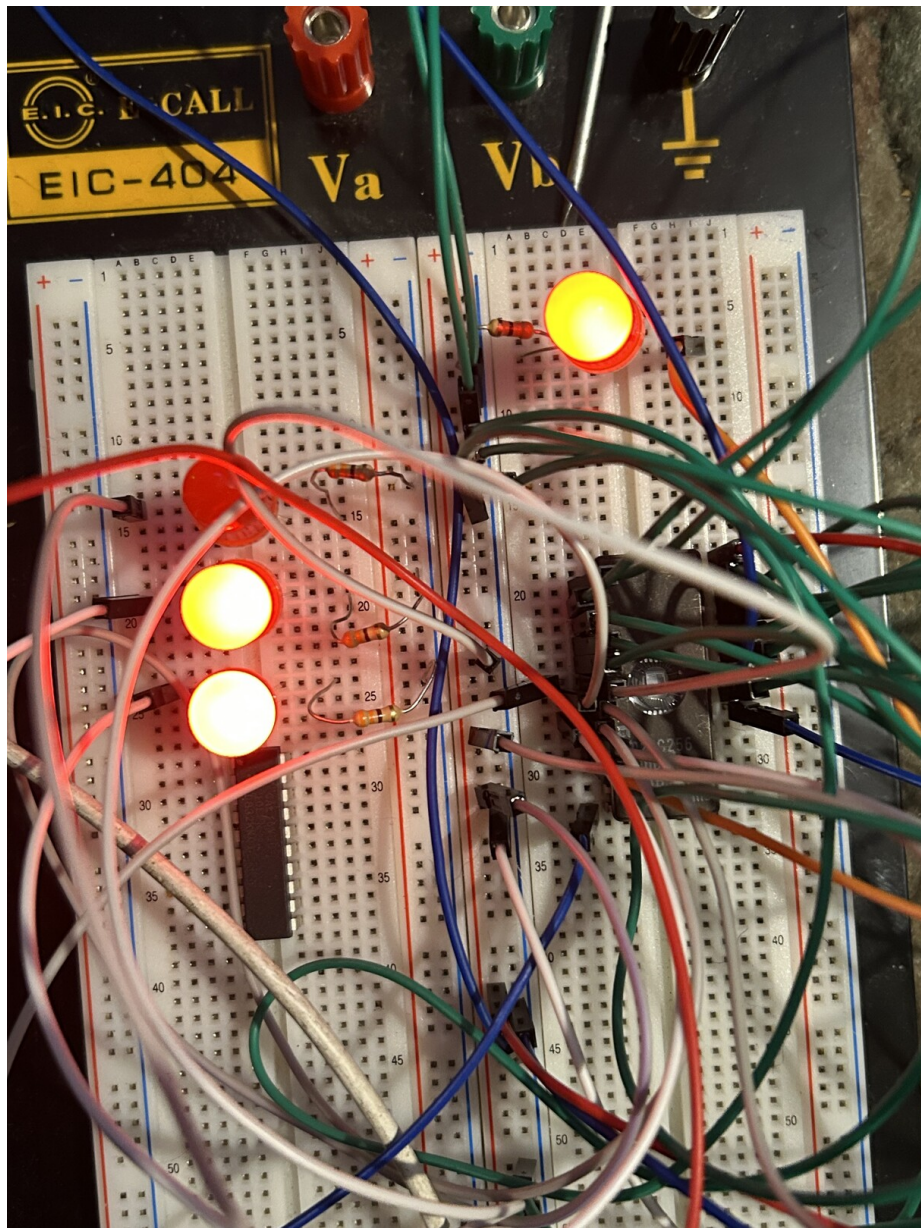


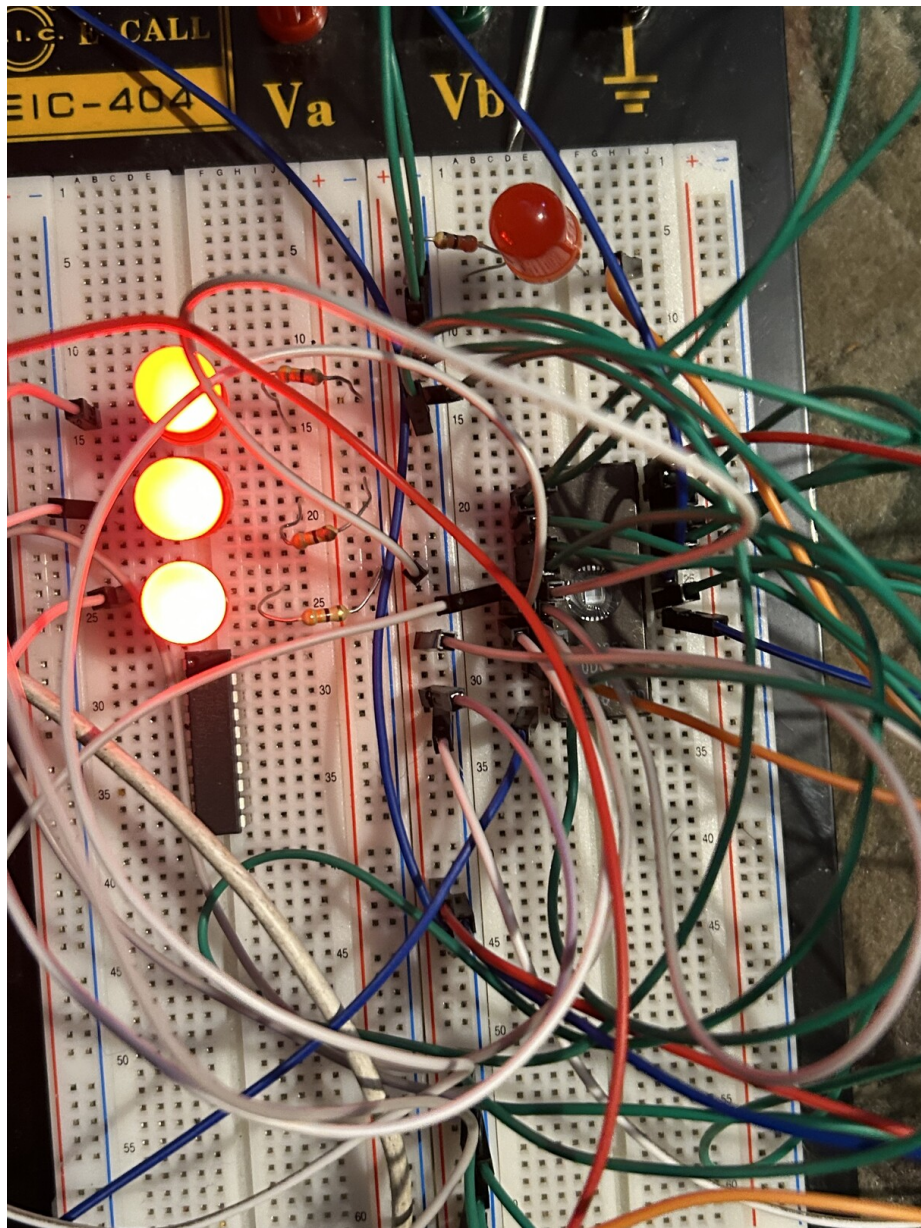


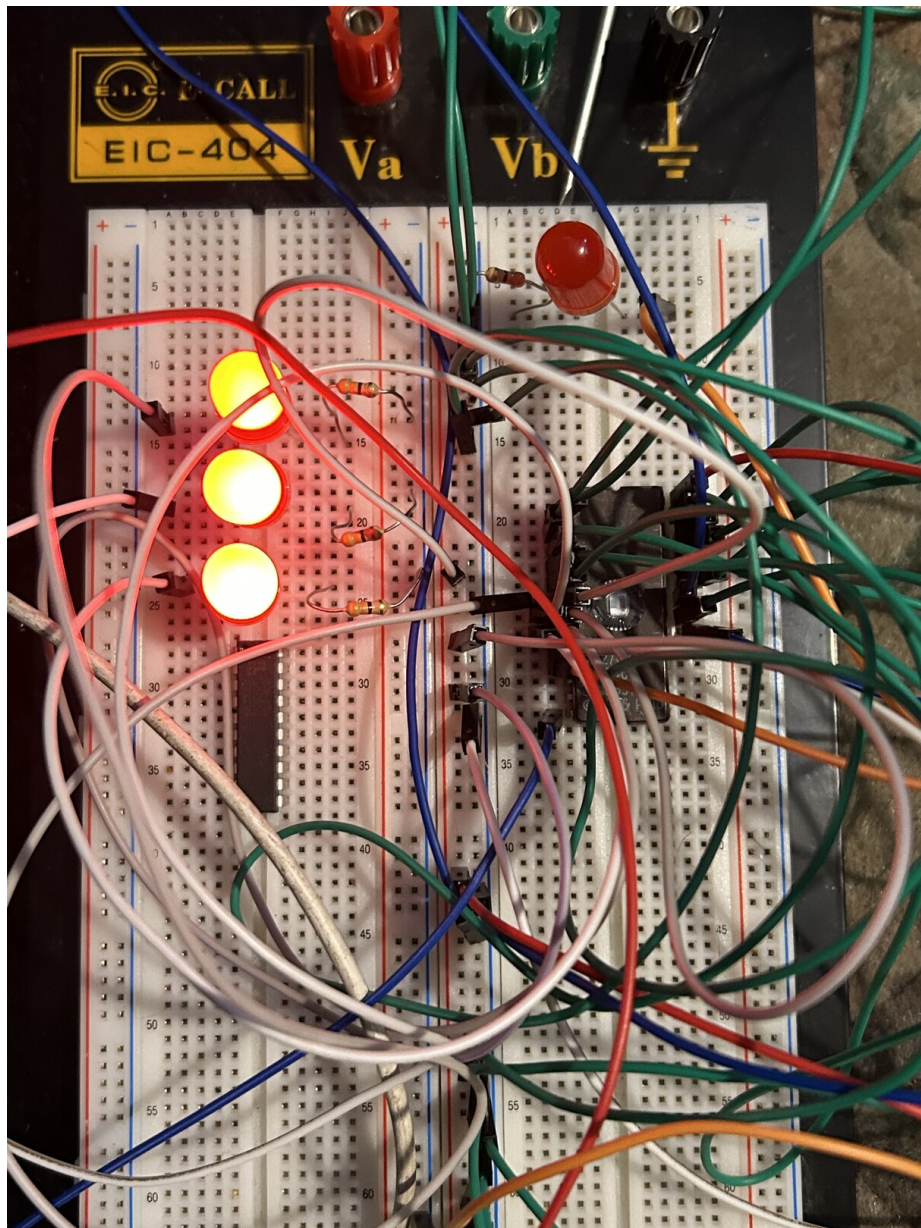


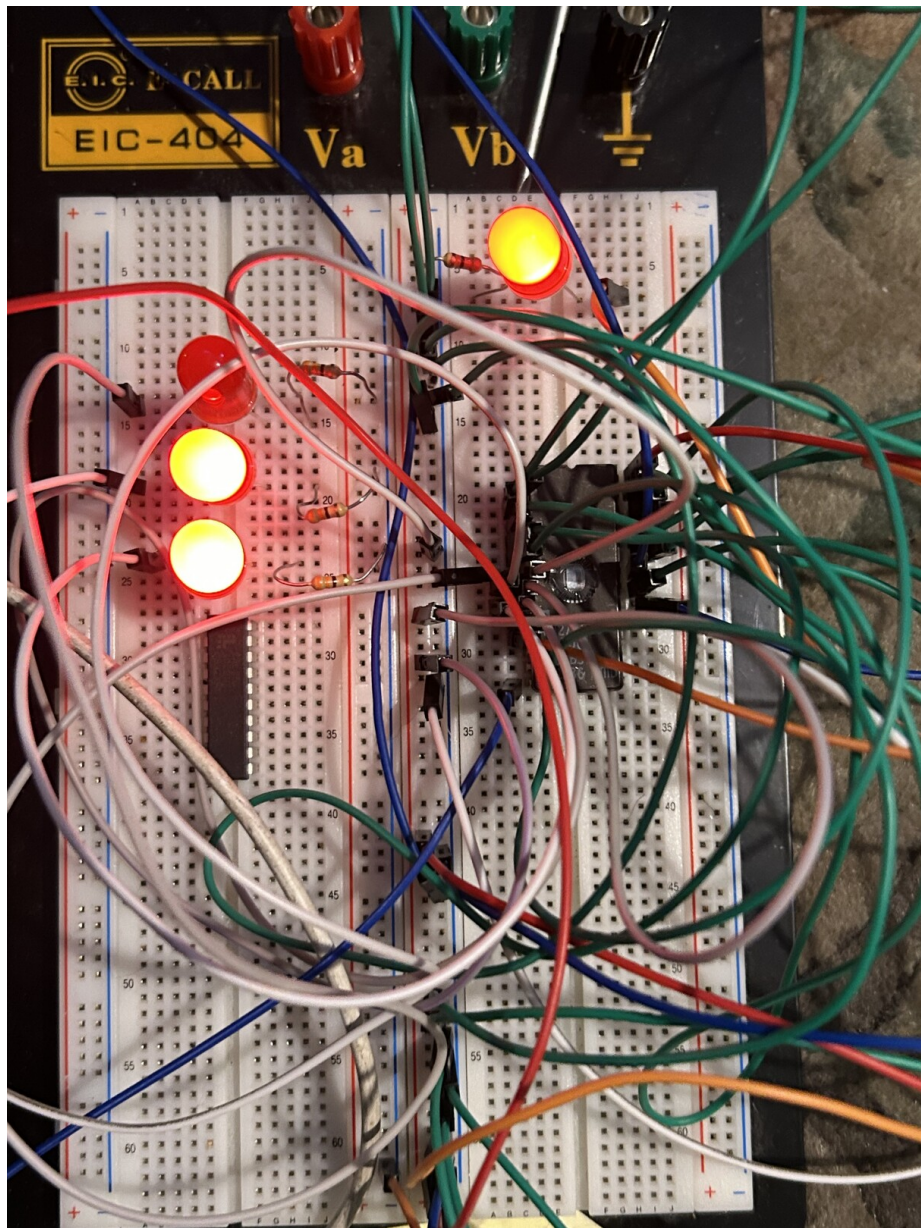


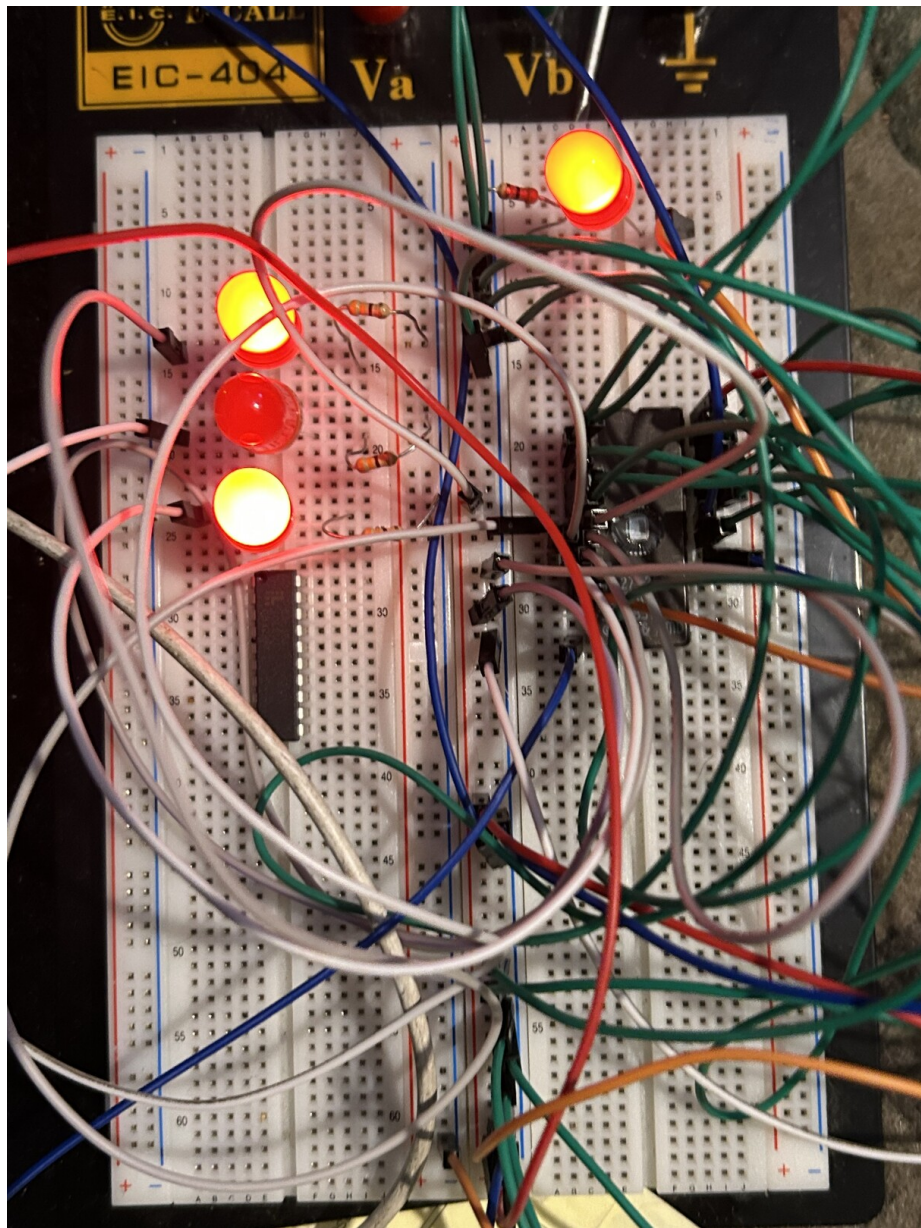


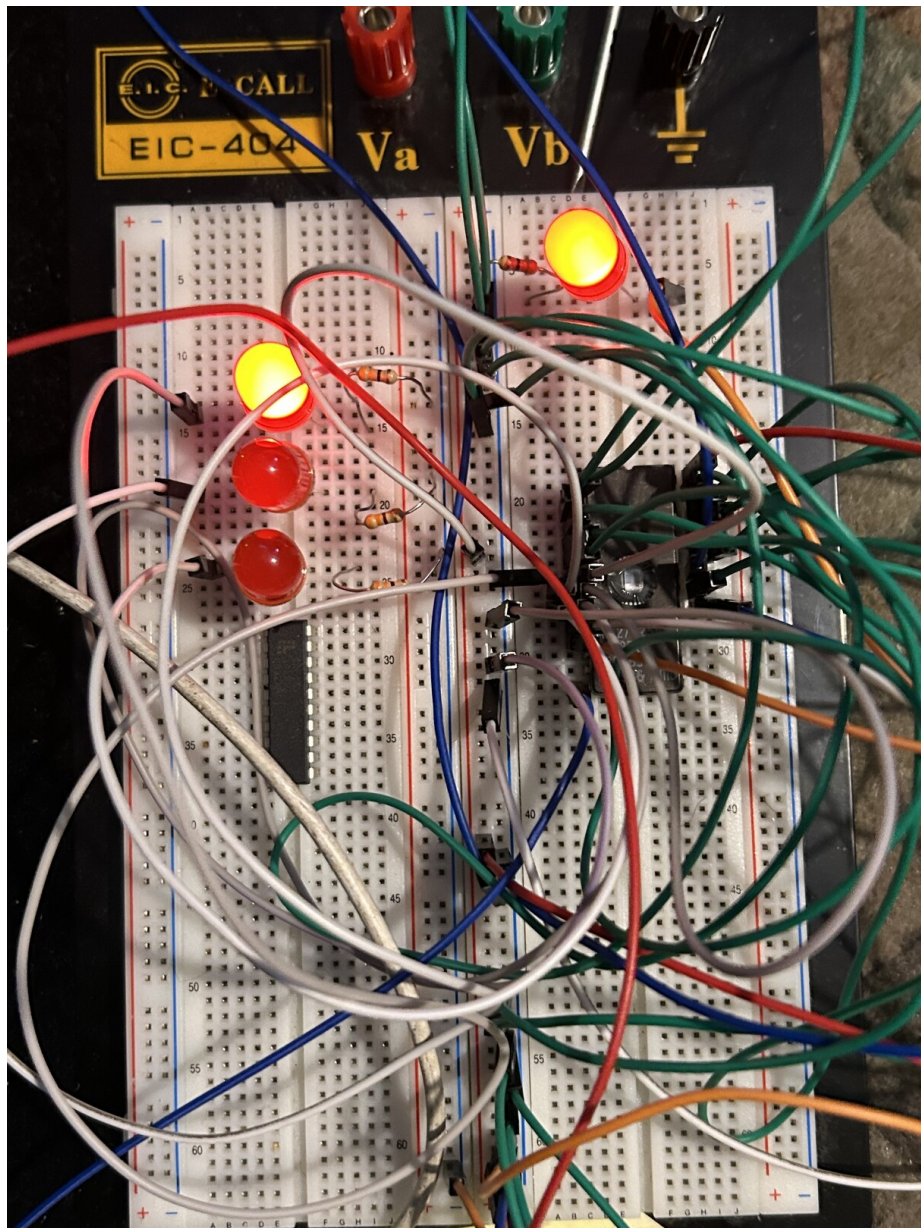


















14 ich habe den fehler gefunden

also, ich habe den fehler gefunden, und das problem liegt da. schaut man in den

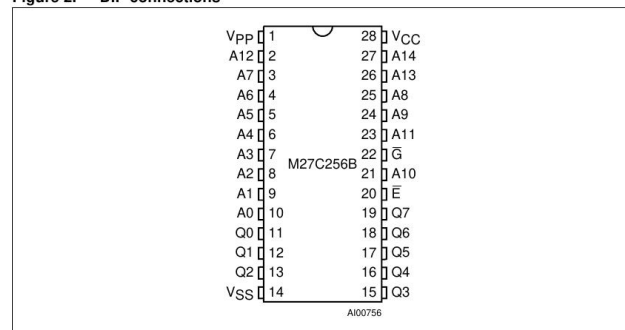
Summary description

M27C256B

Table 1. Signal names

A0-A14	Address Inputs
Q0-Q7	Data Outputs
E	Chip Enable
$\bar{G}$	Output Enable
V _{PP}	Program Supply
V _{CC}	Supply Voltage
V _{SS}	Ground
NC	Not Connected Internally
DU	Don't Use

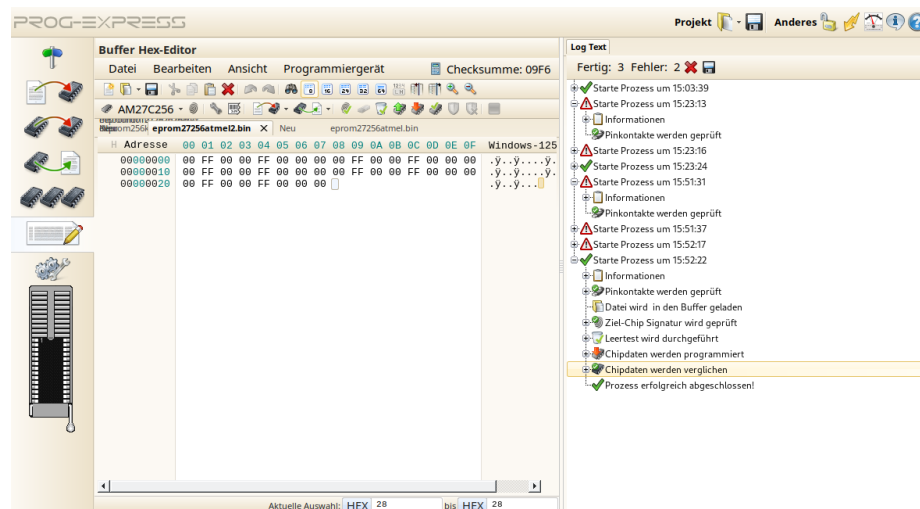
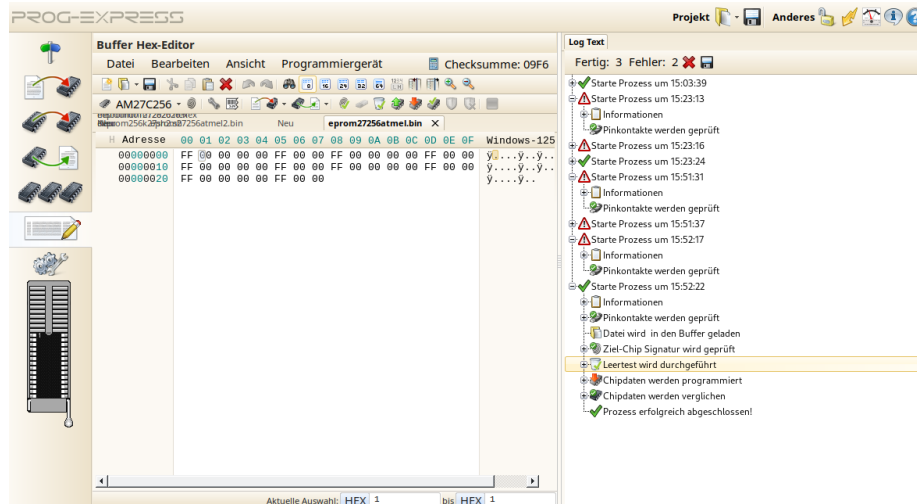
Figure 2. DIP connections



schaltplan.

so, der fehler war, dass ich nicht A0, A1, A2 genutzt habe, sondern A3, A2, A1 und das war nicht erkennbar das problem ist auch nicht little-endian oder big-endian, bei dem man angst haben müsste, es würde auf dem eeprom von der falschen seite gespeichert. es wird so gespeichert, auf dem eeprom wie auf der festplatte. das problem, waren hier nur die addressleitungen. . . .

Also, hier steht noch mal little und big endian, die reihenfolge der LED ist schon erkennbar, auf das abgebildet wie es da steht.



Hier wurde erzeugt ein mal big-endian (normal)

```
#include <stdio.h>

// (c) david vajda
// 01/29/2026
// eprom 27256, 256k

void main (void) {
```



```

char bytes [] = {0xff, 0x00, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00};
int i = 0;

fprintf (stderr, "(c) david vajda\n");
fprintf (stderr, "01/29/2026\n");
fprintf (stderr, "eprom 27256, 256k\n");
while (i < 8) {
    printf ("%c", bytes [i]);
    i = i + 1;
}

return;
}

little-endian

```

```

#include <stdio.h>

// (c) david vajda
// 01/29/2026
// eprom 27256, 256k

void main (void) {
    char bytes [] = {0xff, 0x00, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00};
    int i = 0;

    fprintf (stderr, "(c) david vajda\n");
    fprintf (stderr, "01/29/2026\n");
    fprintf (stderr, "eprom 27256, 256k\n");
    while (i < 8) {
        printf ("%c%c", bytes [i+1], bytes [i]);
        i = i + 2;
    }

    return;
}

```

und hier eine SH Datei zum wiederholten erzeugen little-endian

```

#!/bin/bash

# (c) david vajda
# 01/29/2026
# eprom 27256, 256k

i=0
while [ $i -lt 8 ]

```

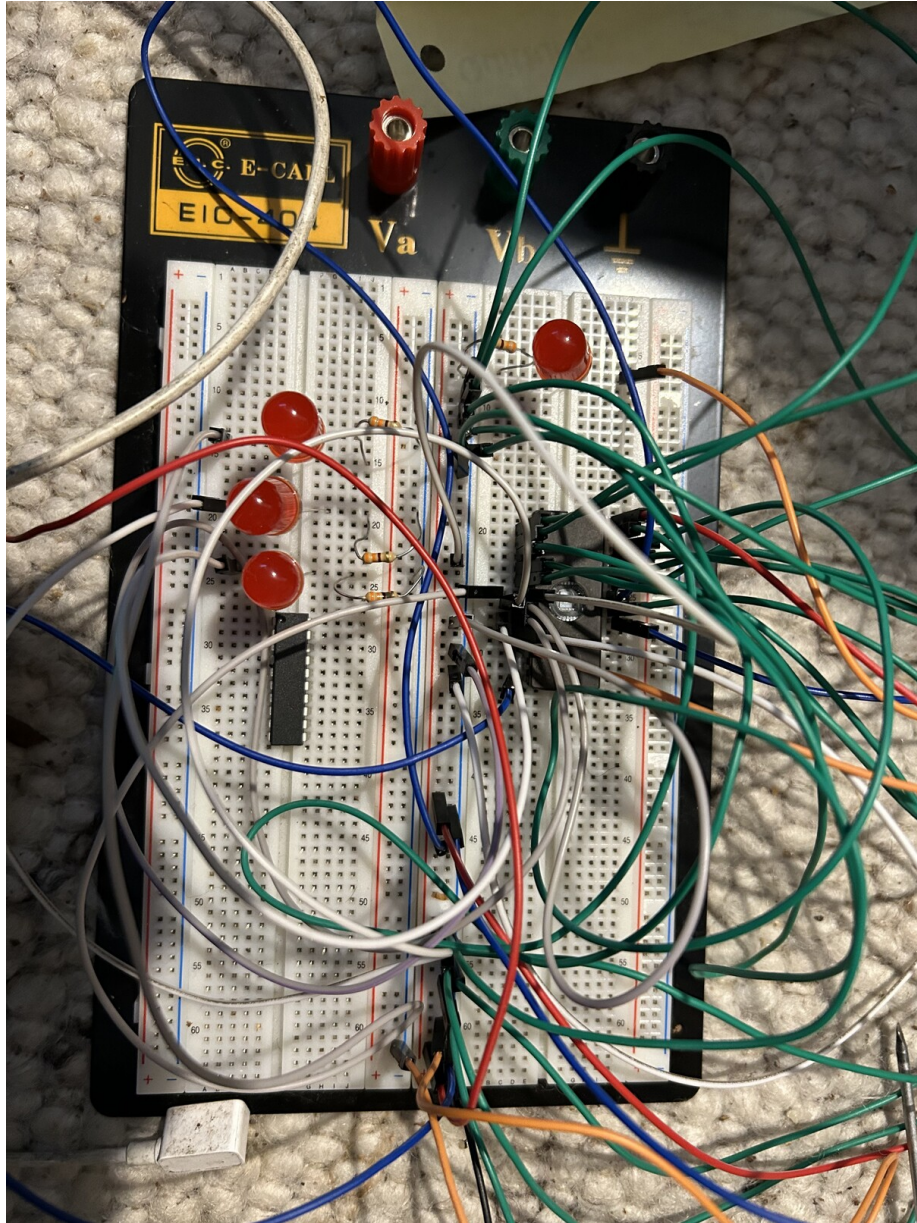
```
do
    ./eprom27256atmel >> eprom27256atmel2.bin
    i=$((i+1))
done
big-endian
```

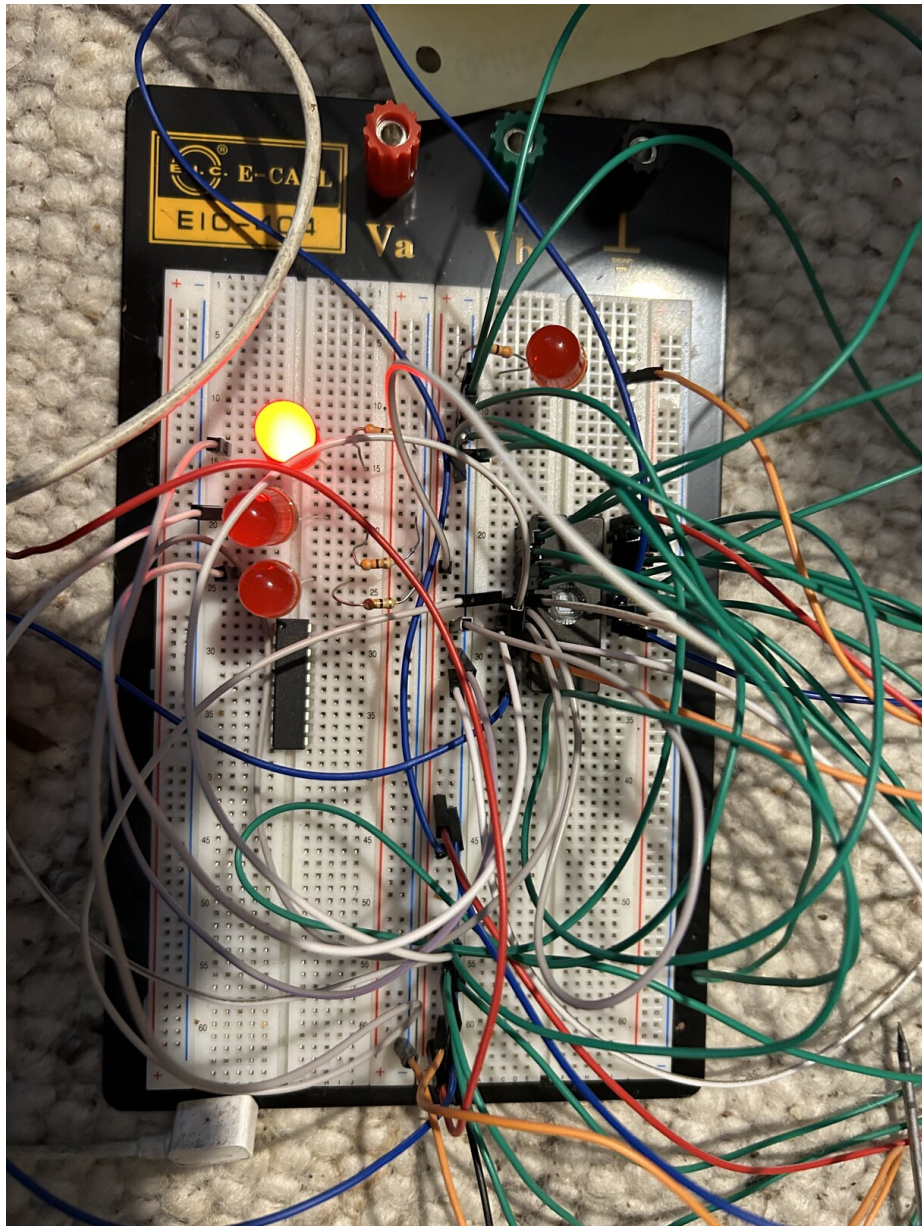
```
#!/bin/bash
```

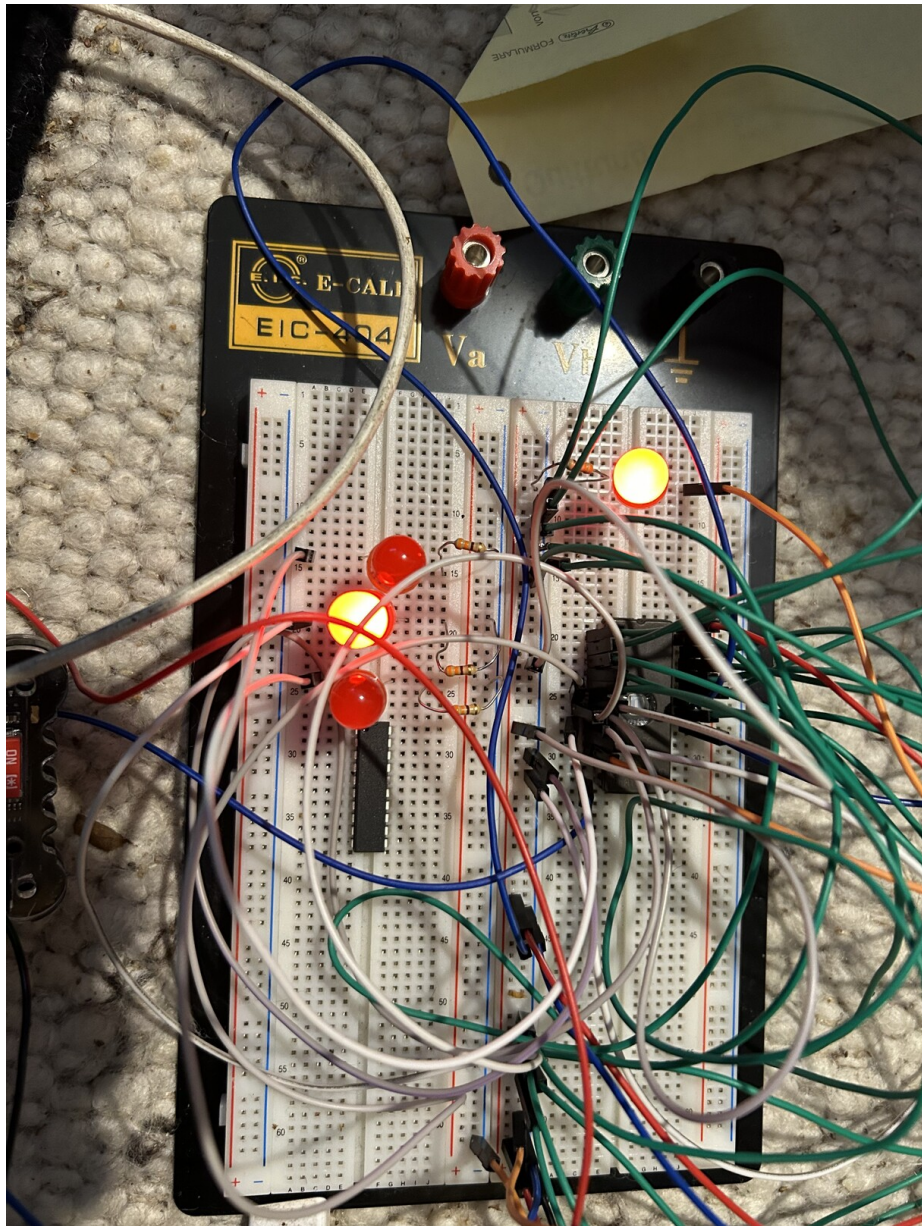
```
# (c) david vajda
# 01/29/2026
# eprom 27256, 256k
```

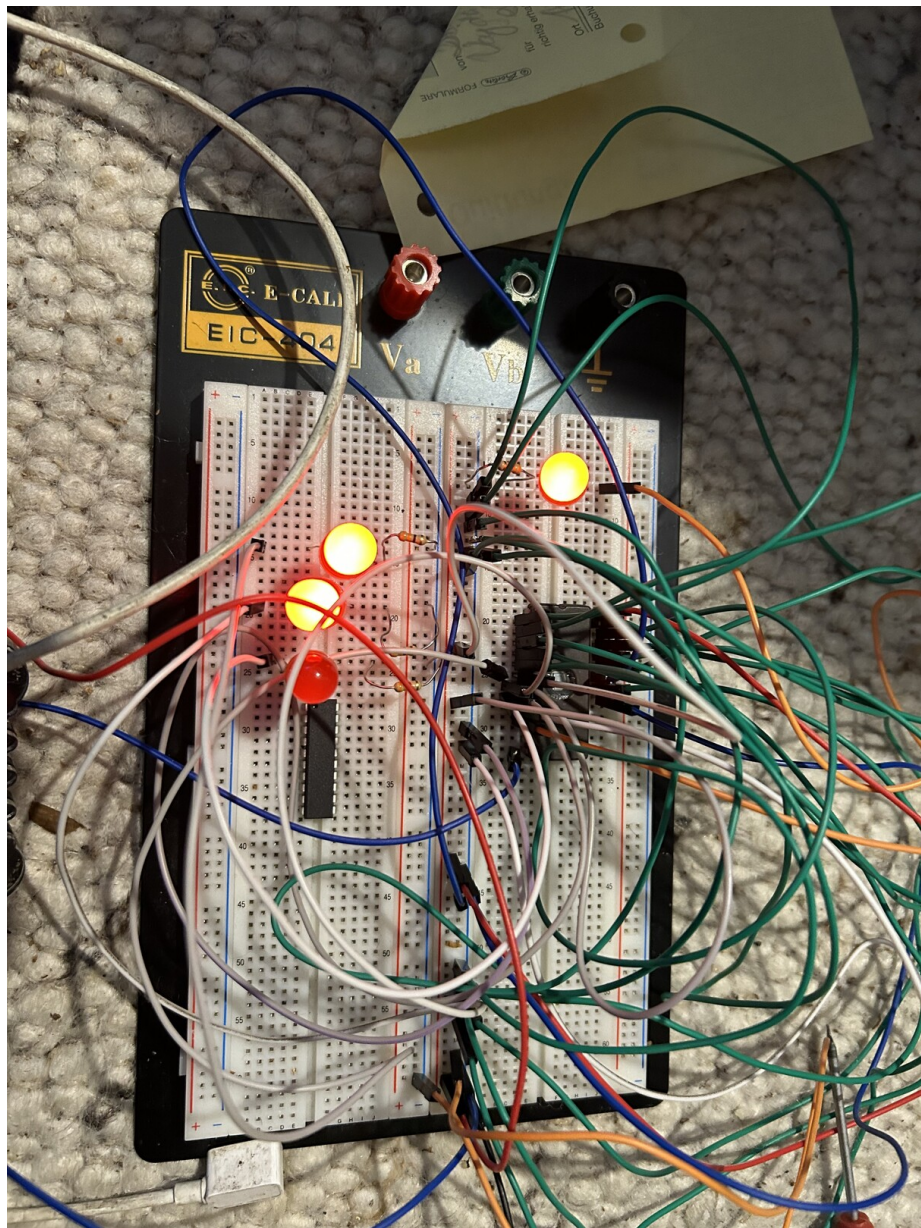
```
i=0
while [ $i -lt 8 ]
do
    ./eprom27256atmel2
    i=$((i+1))
done
```

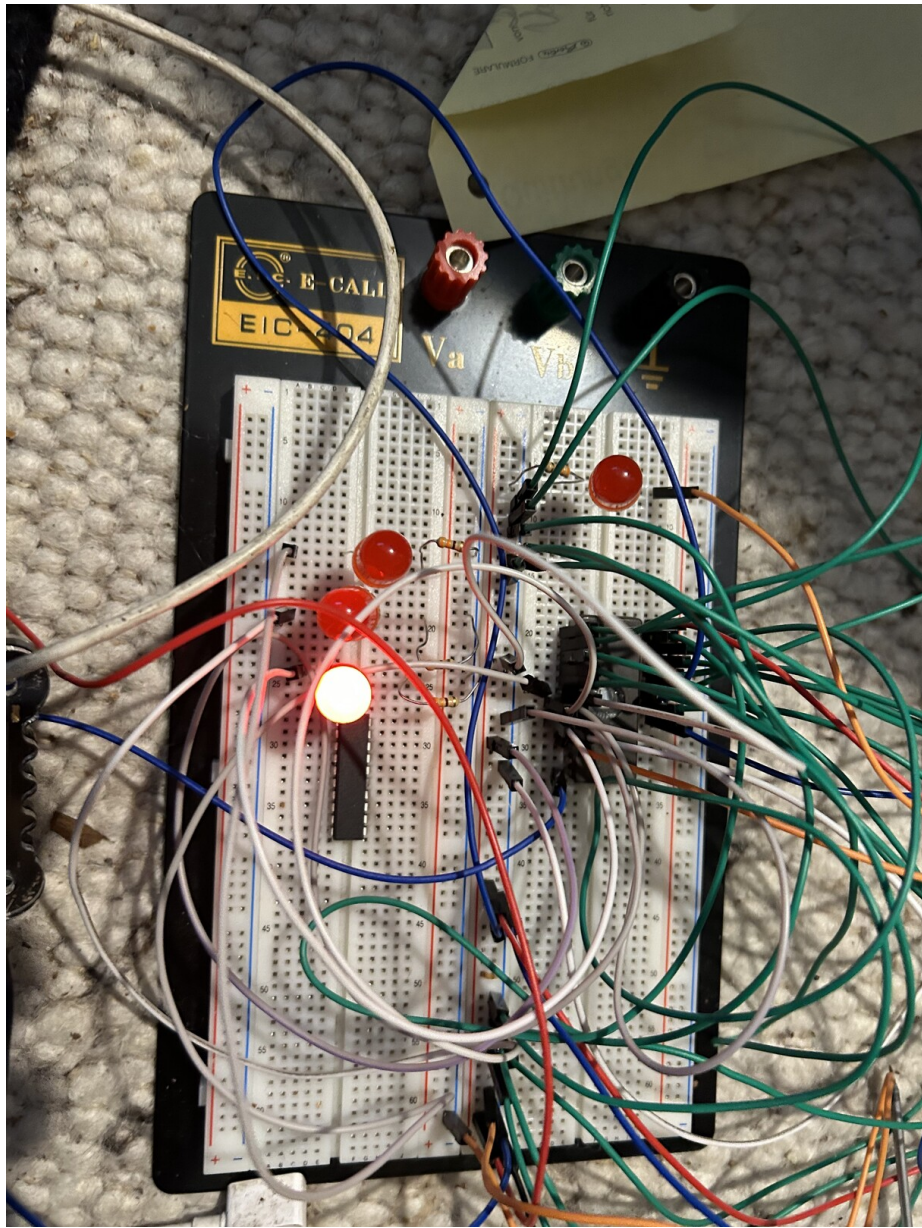
15 Ein Versuch

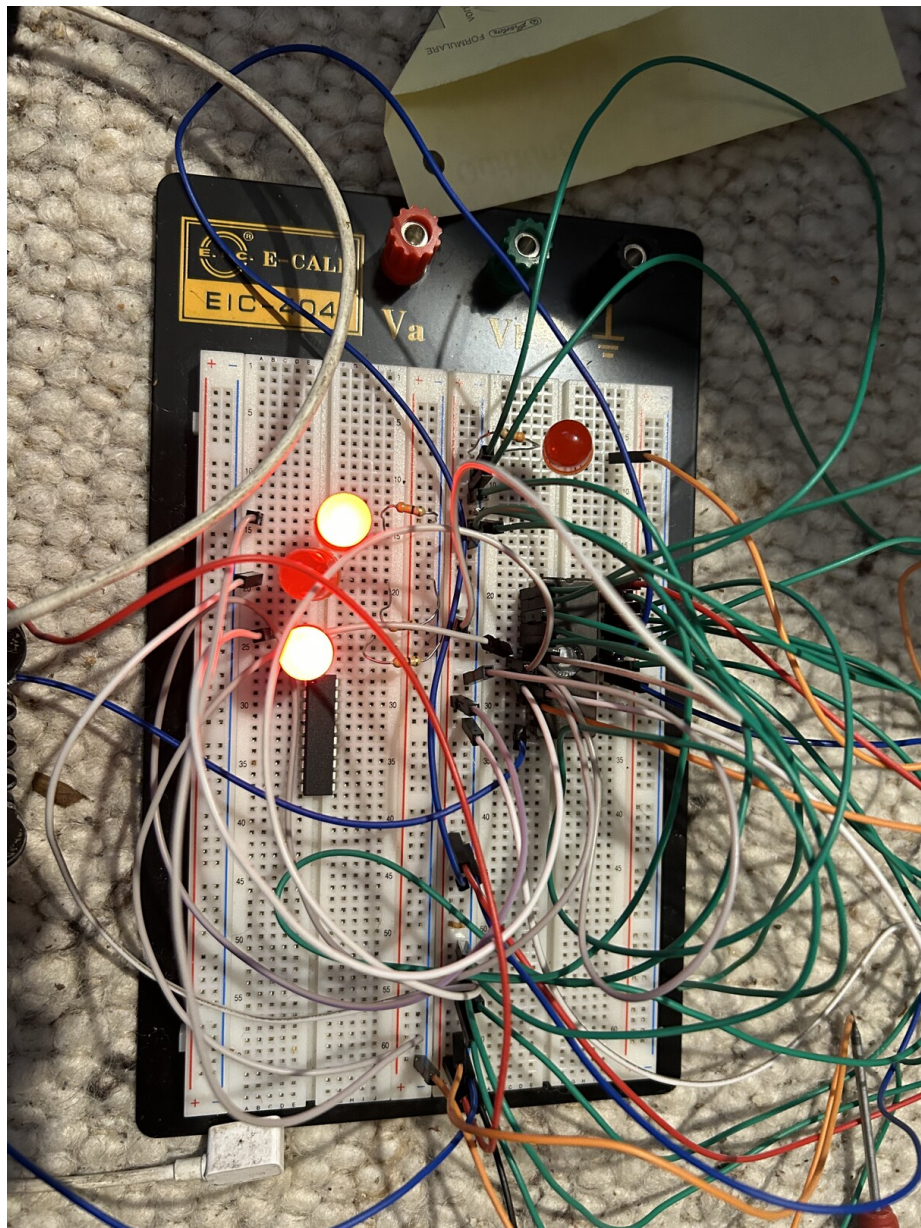


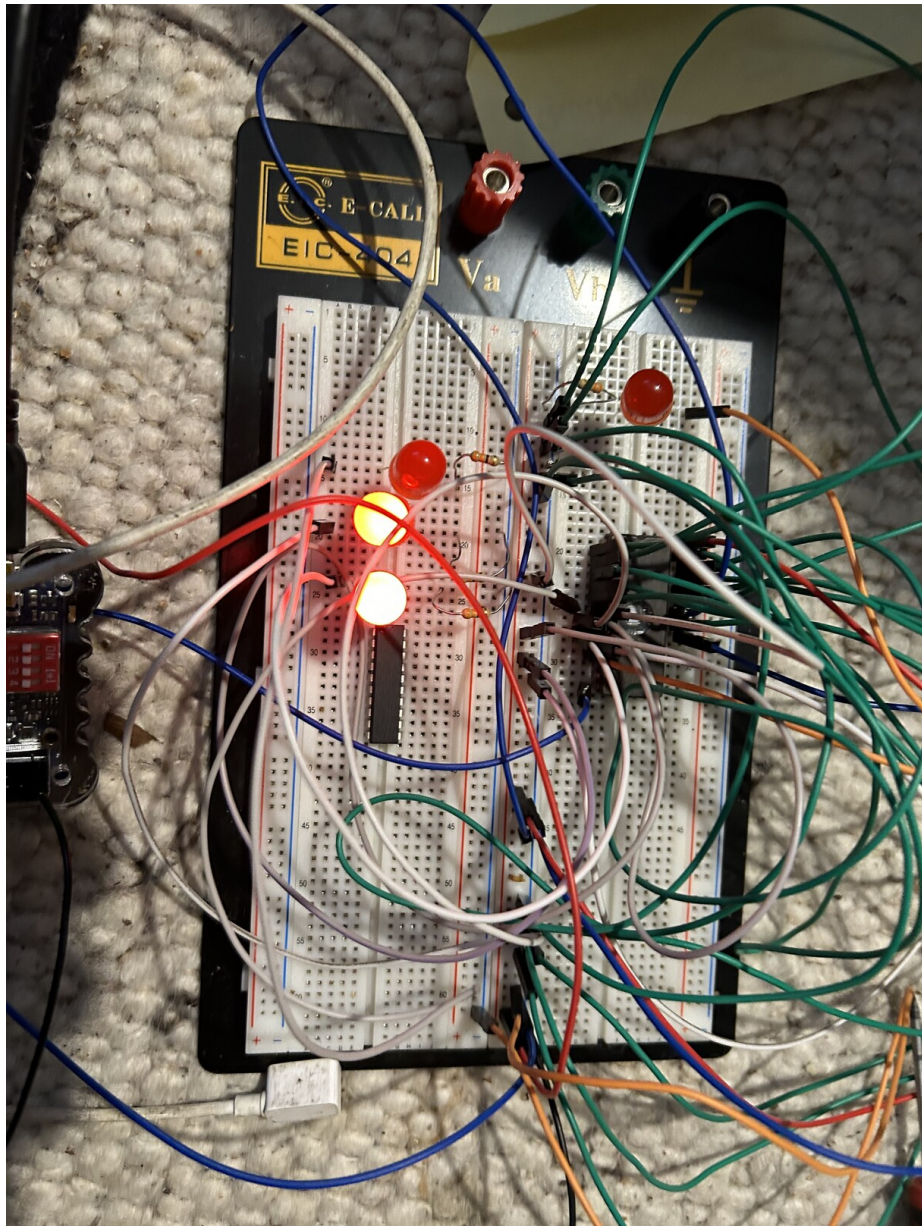


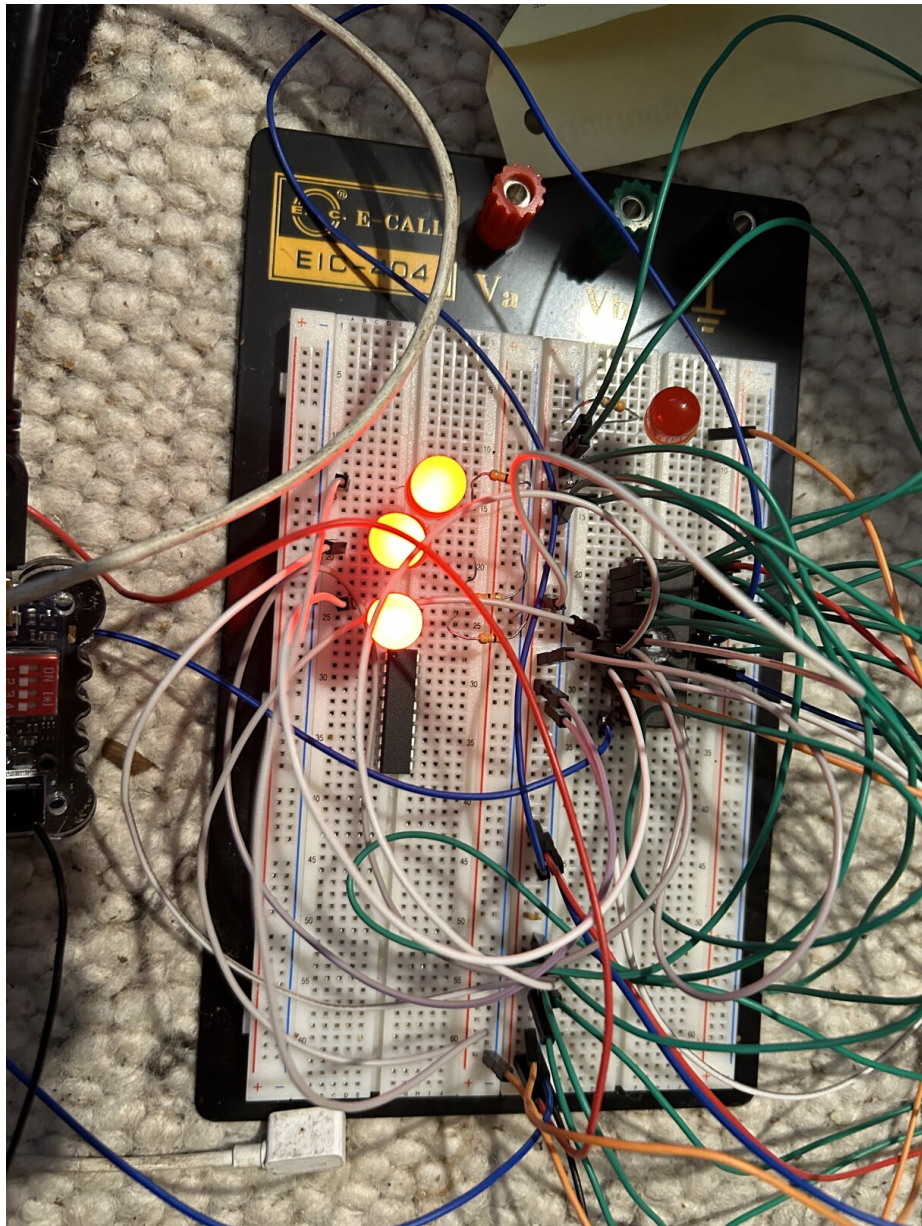




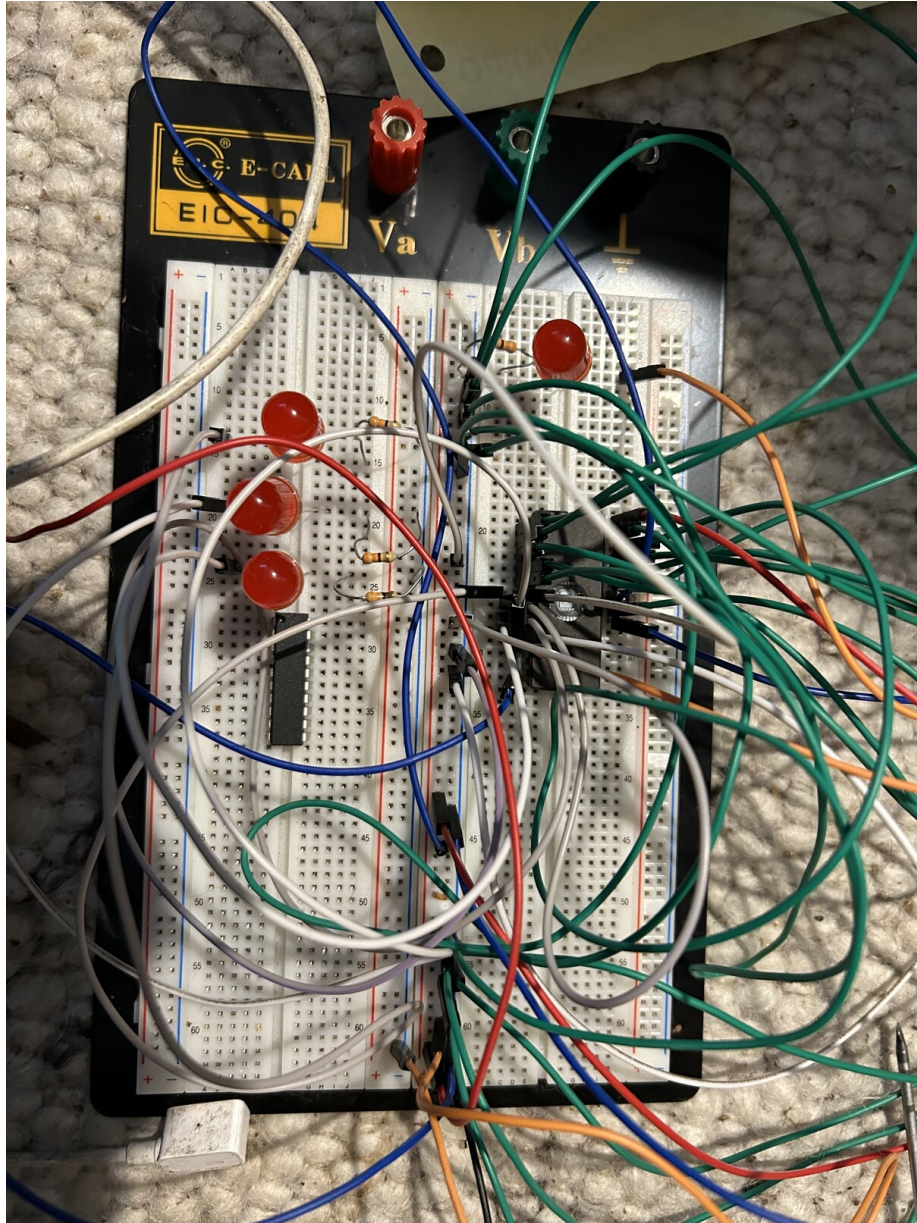


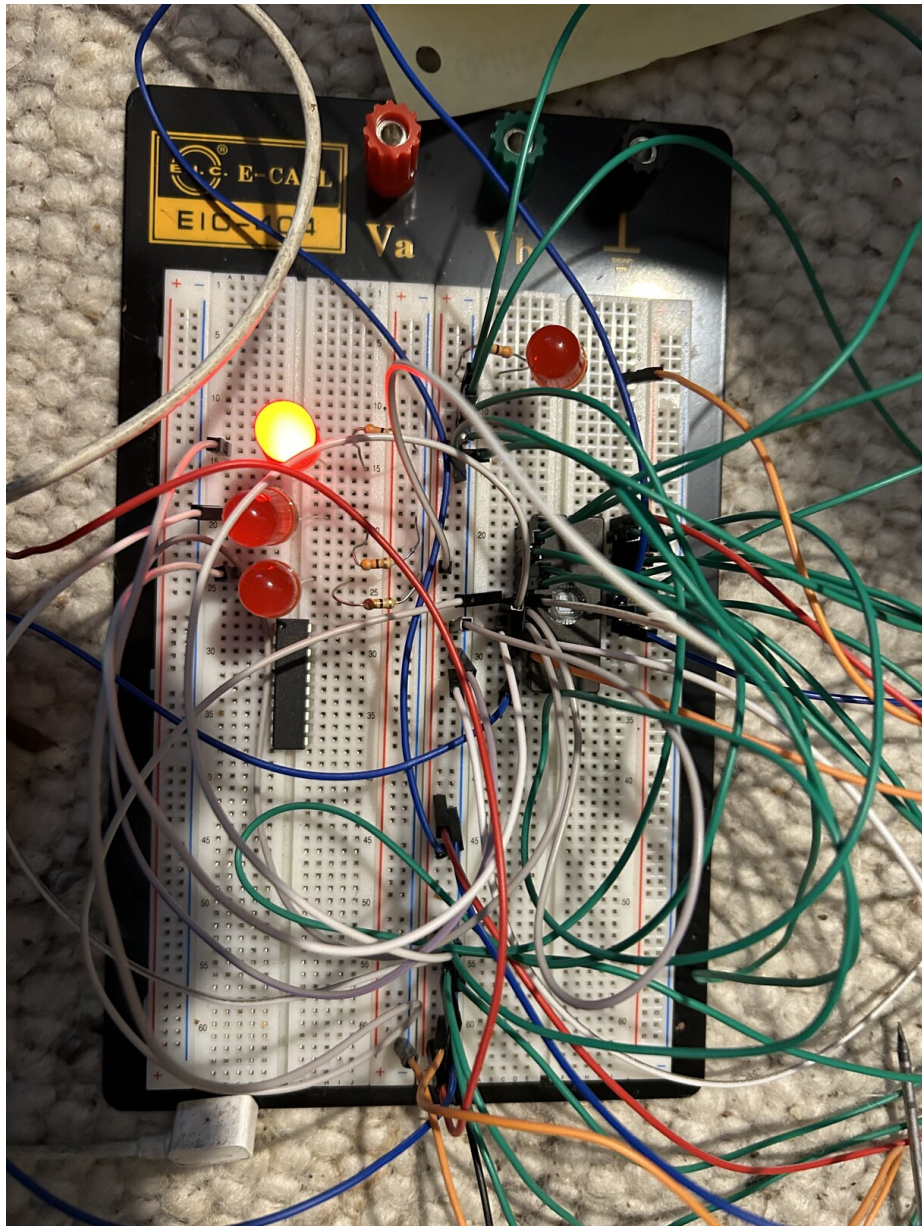


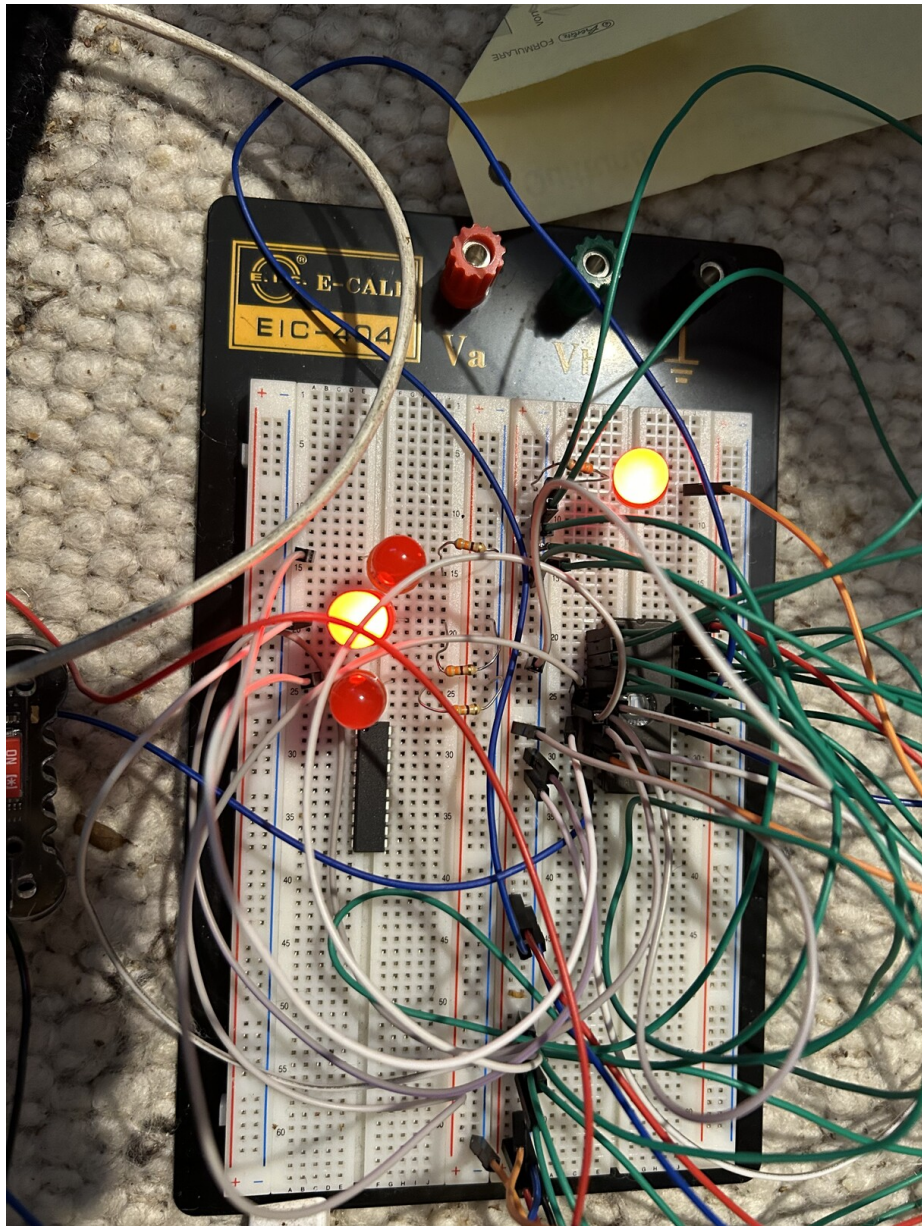


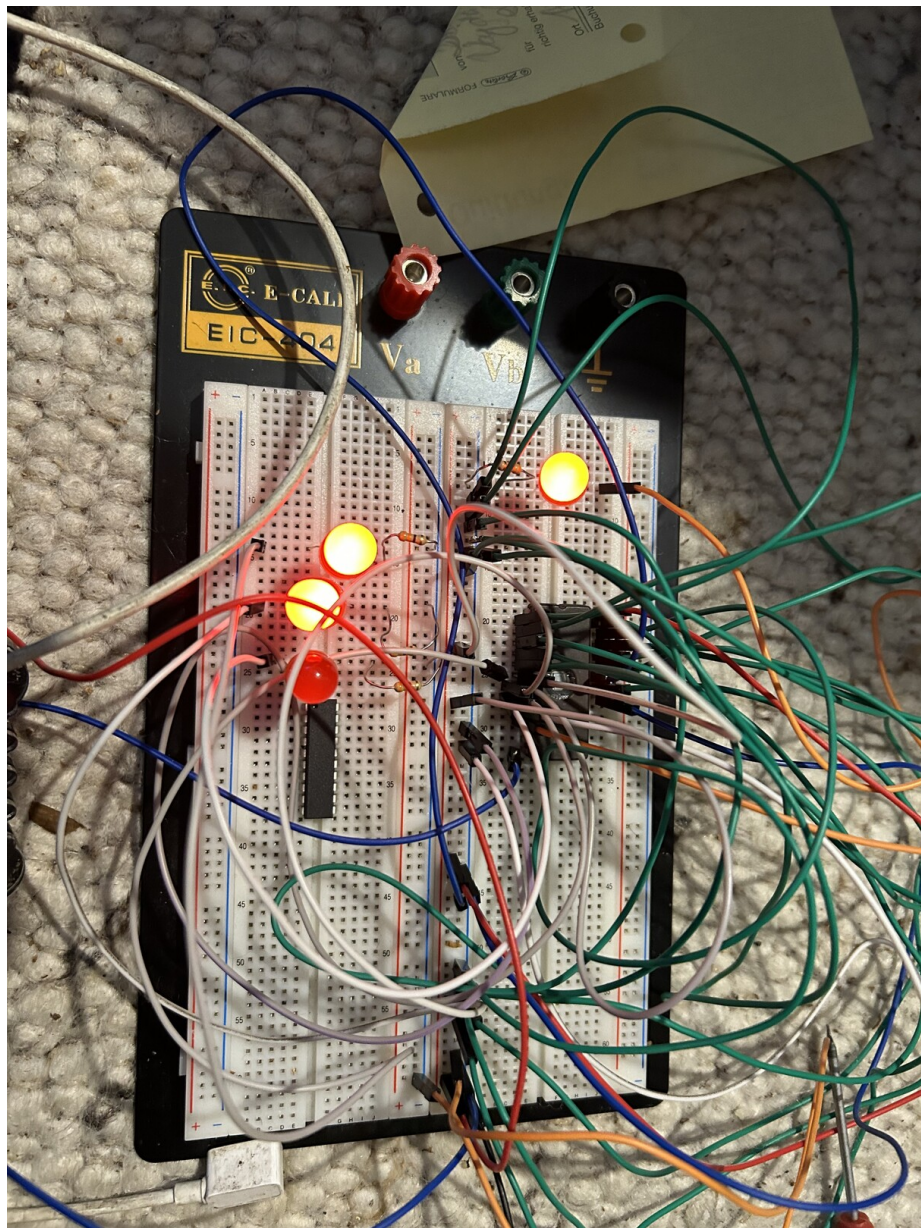


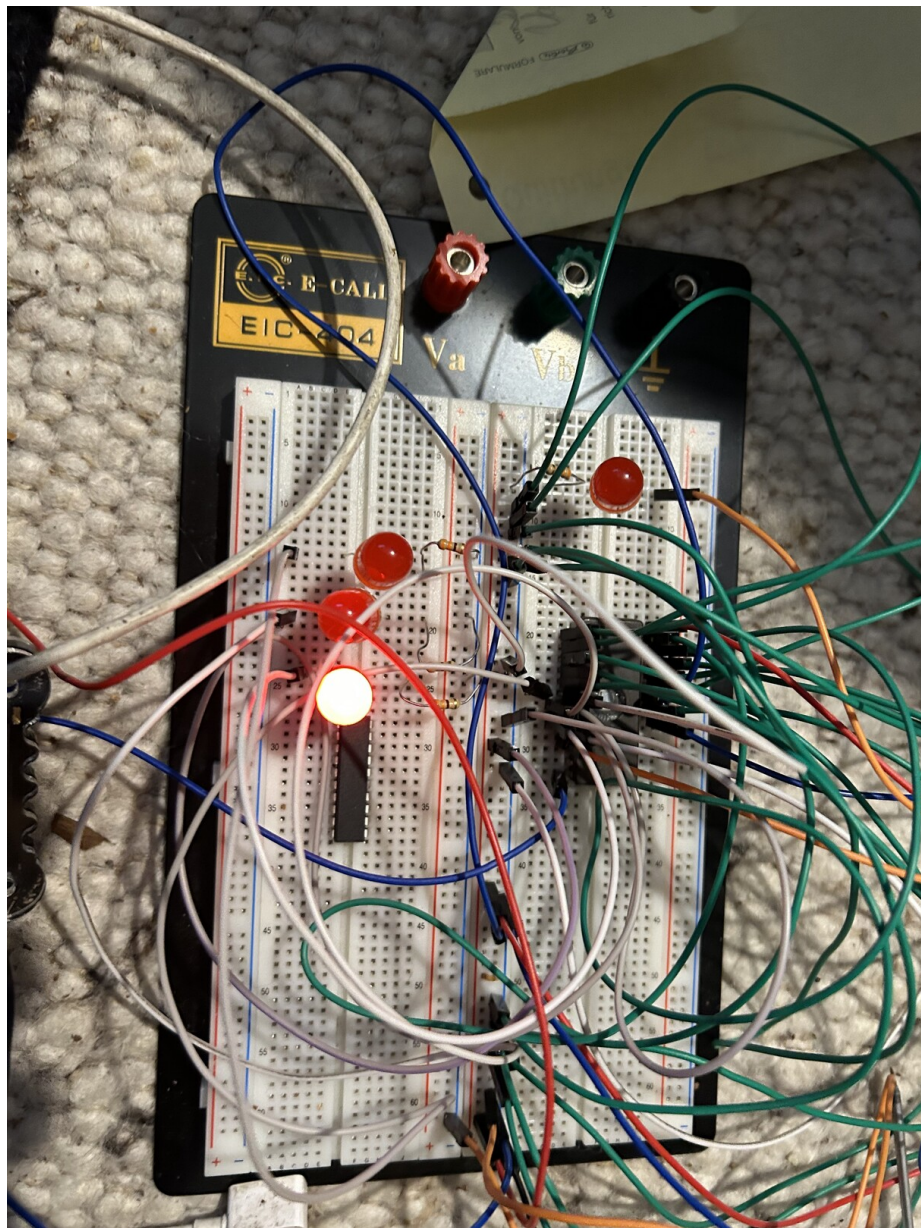
16 Zweiter Versuch

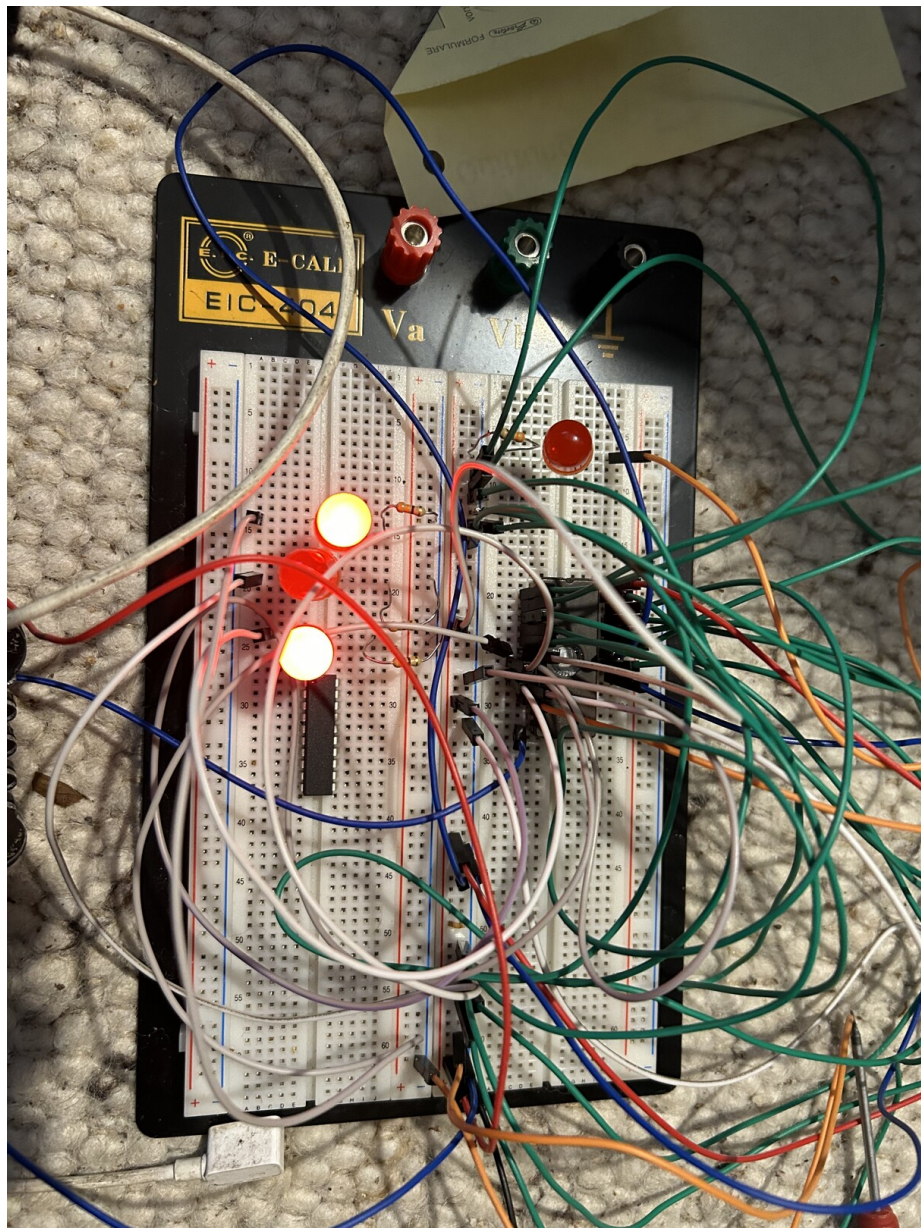


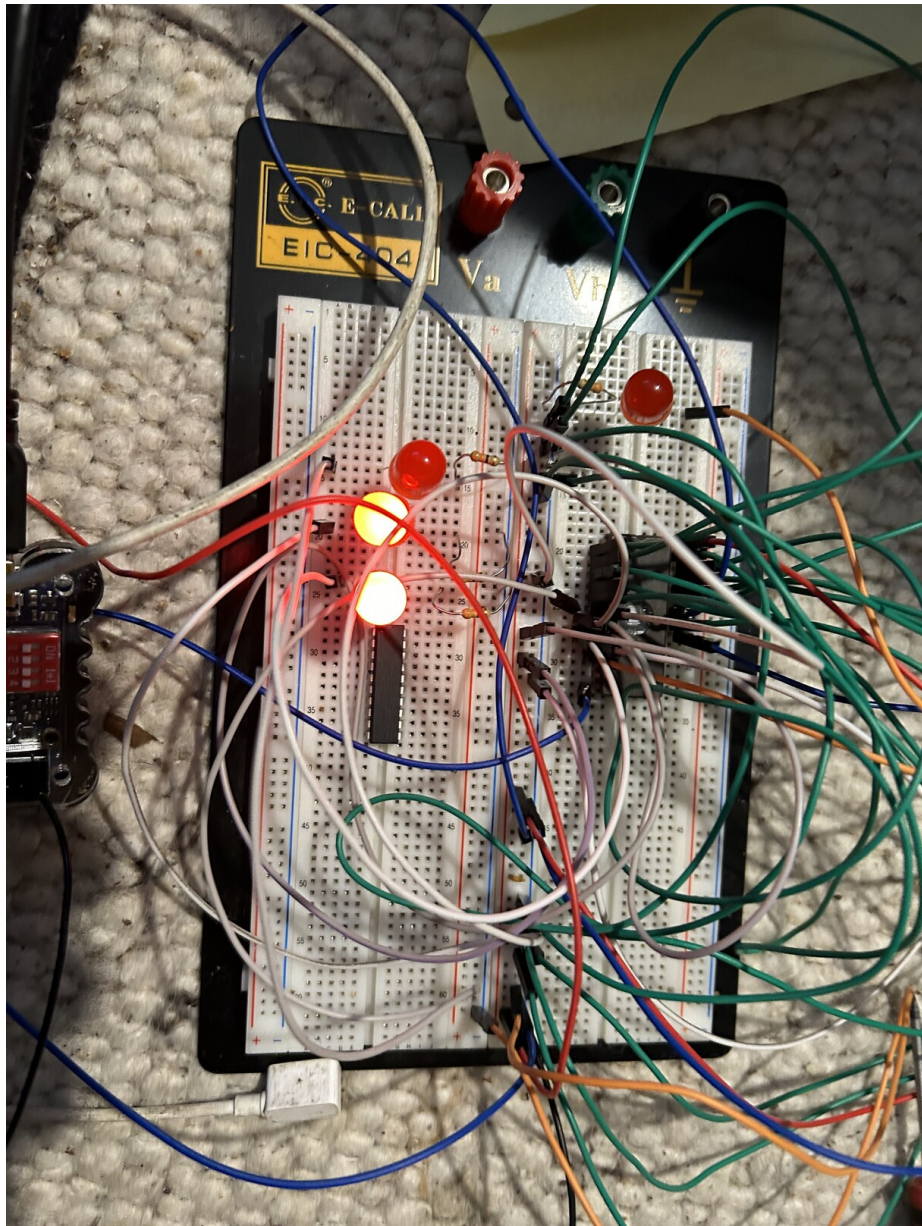


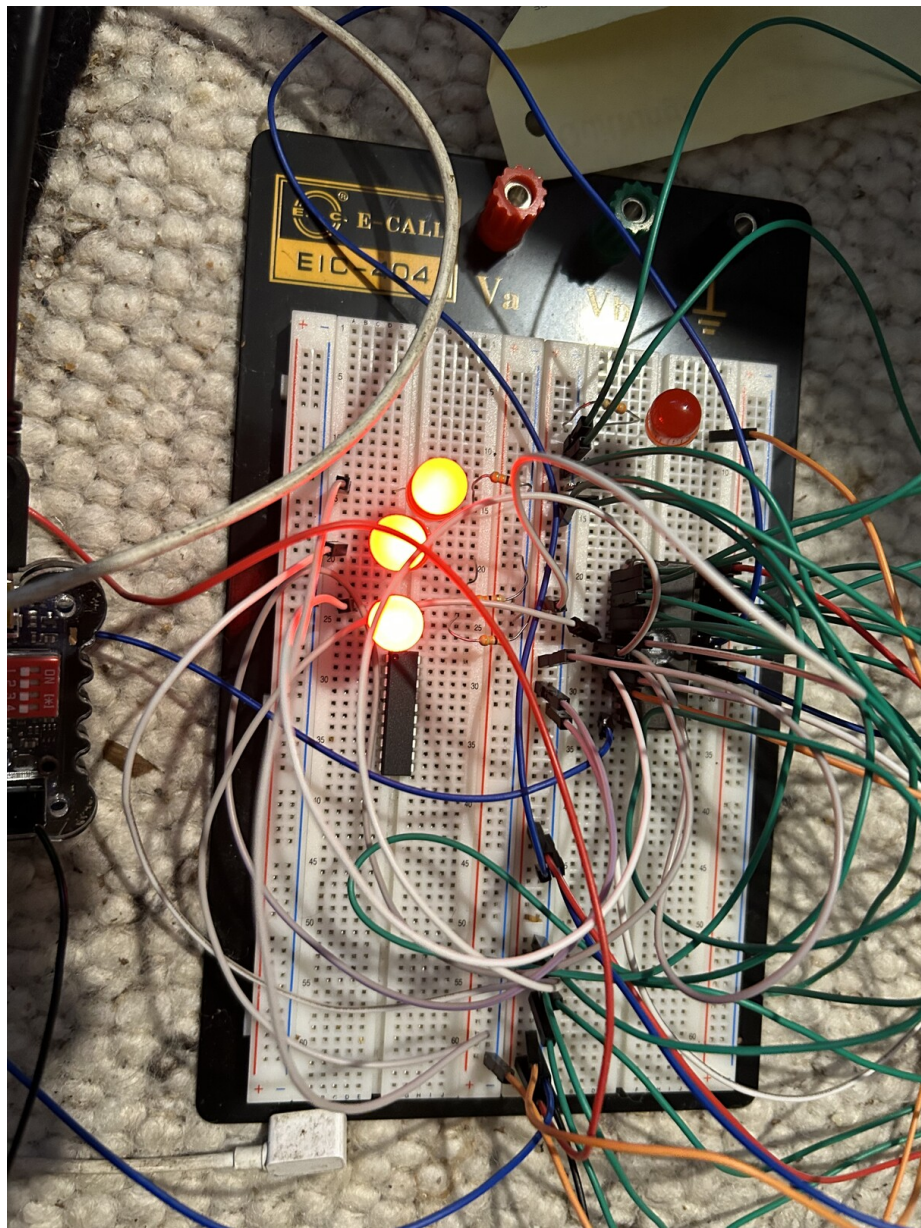


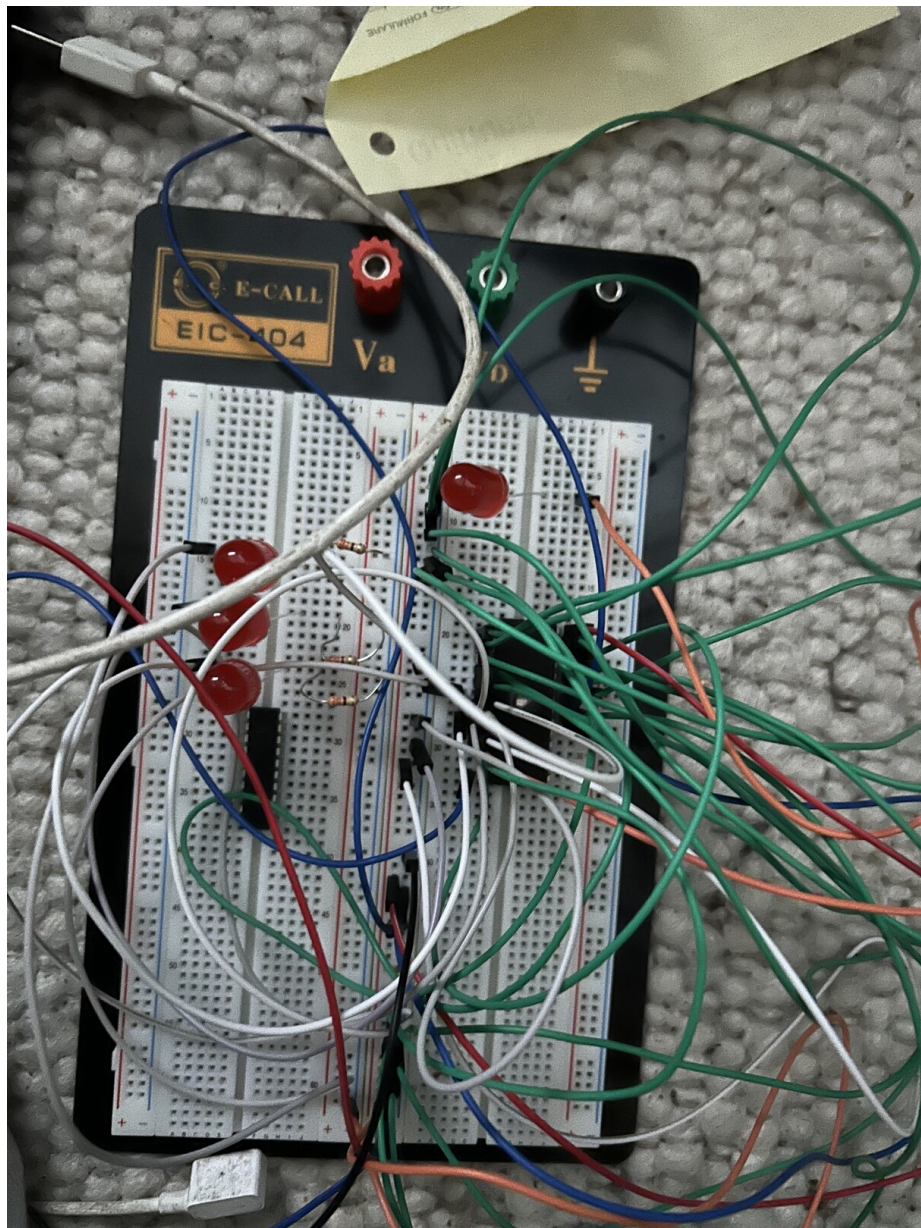


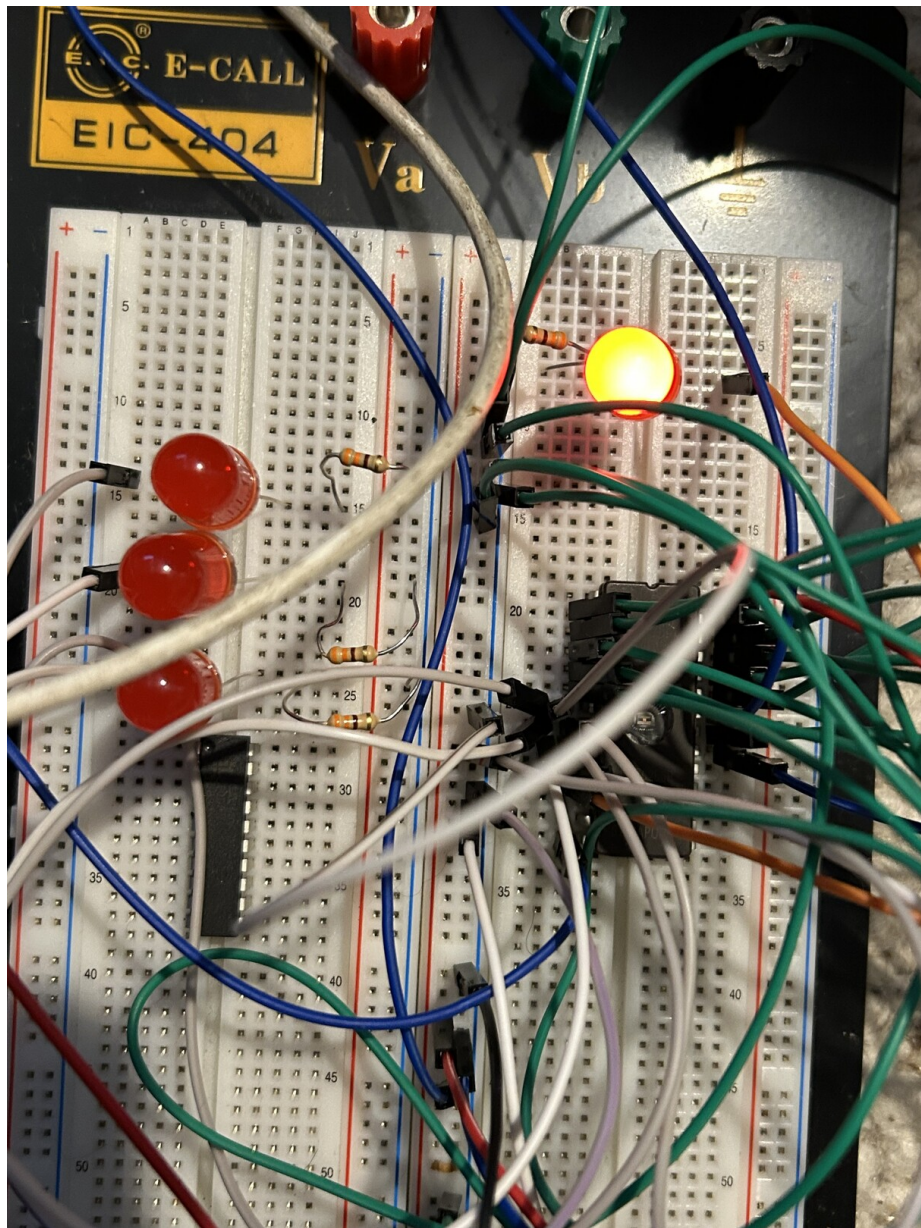


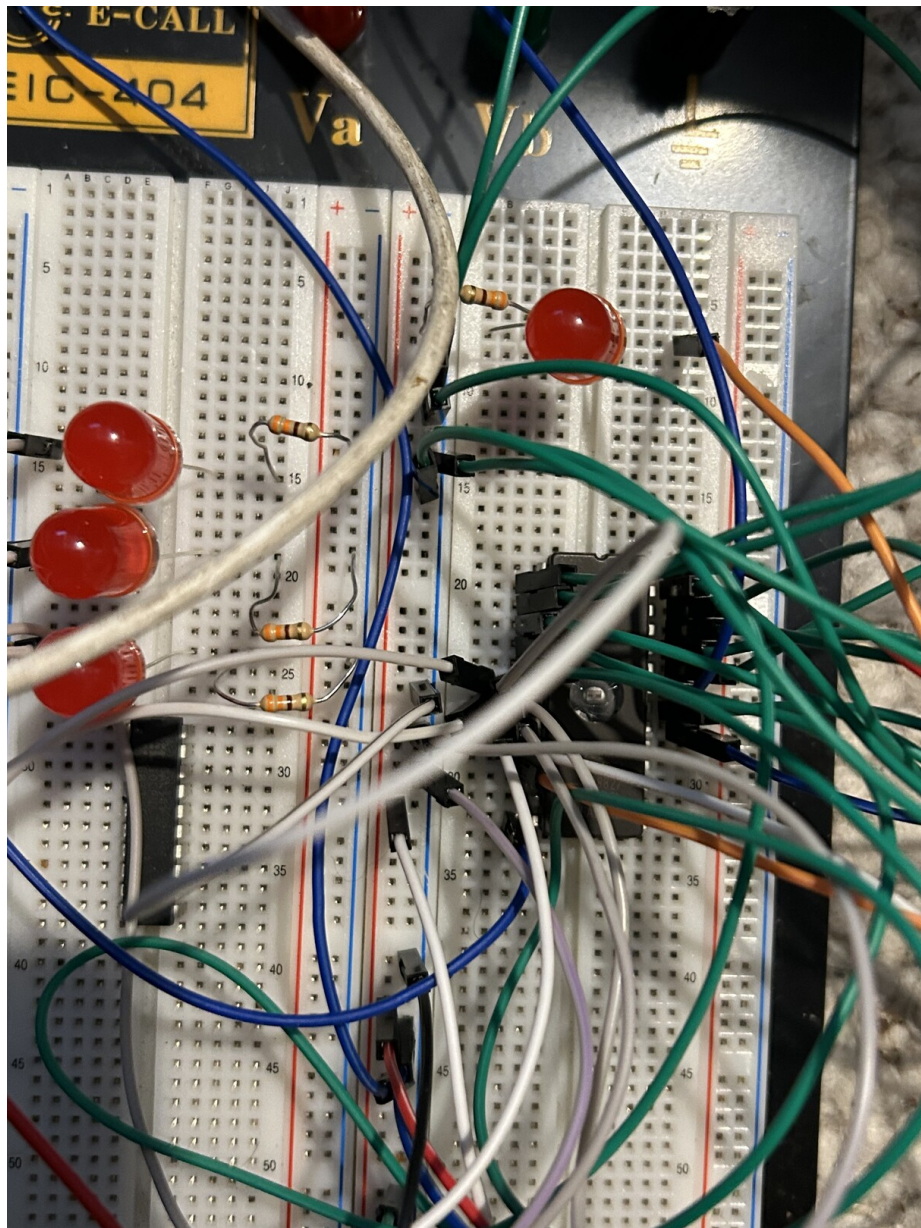


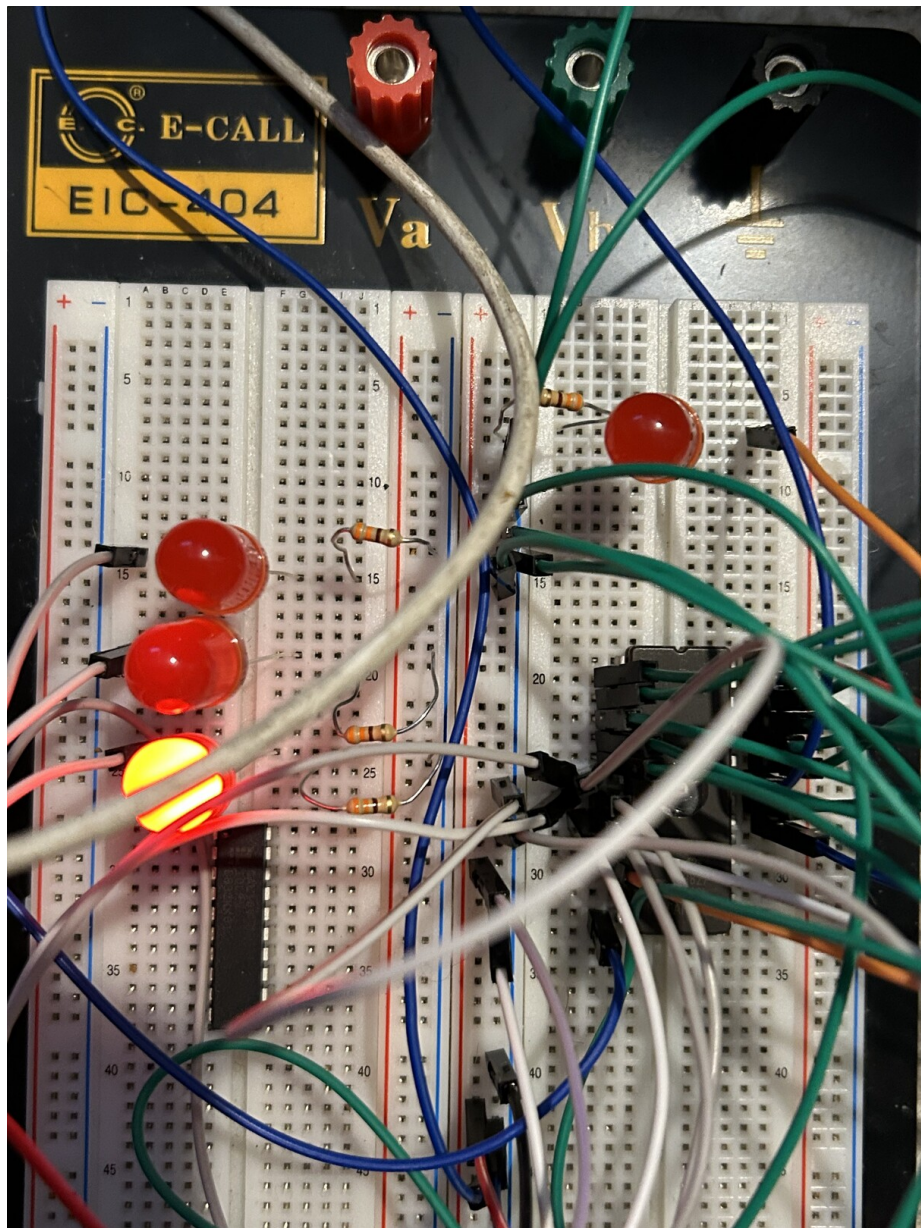


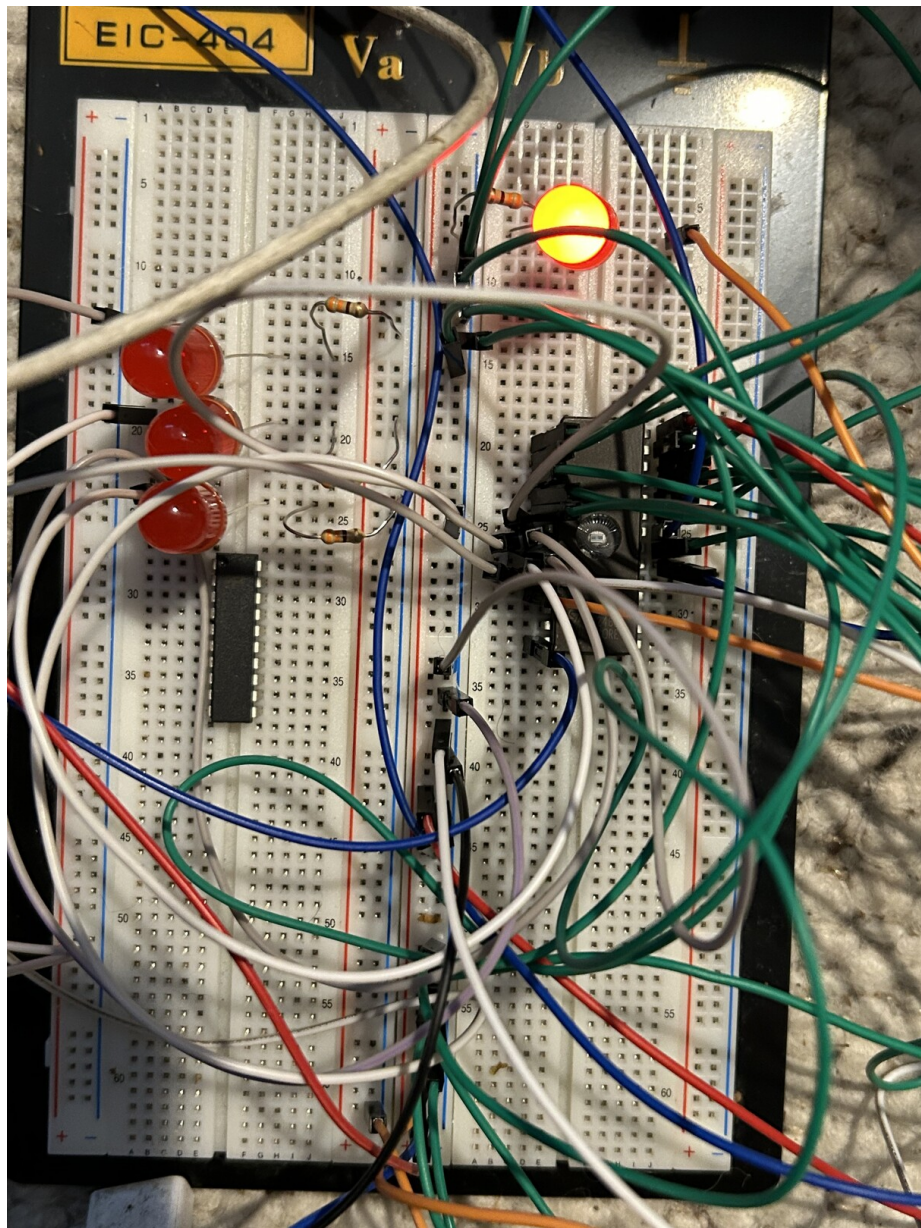


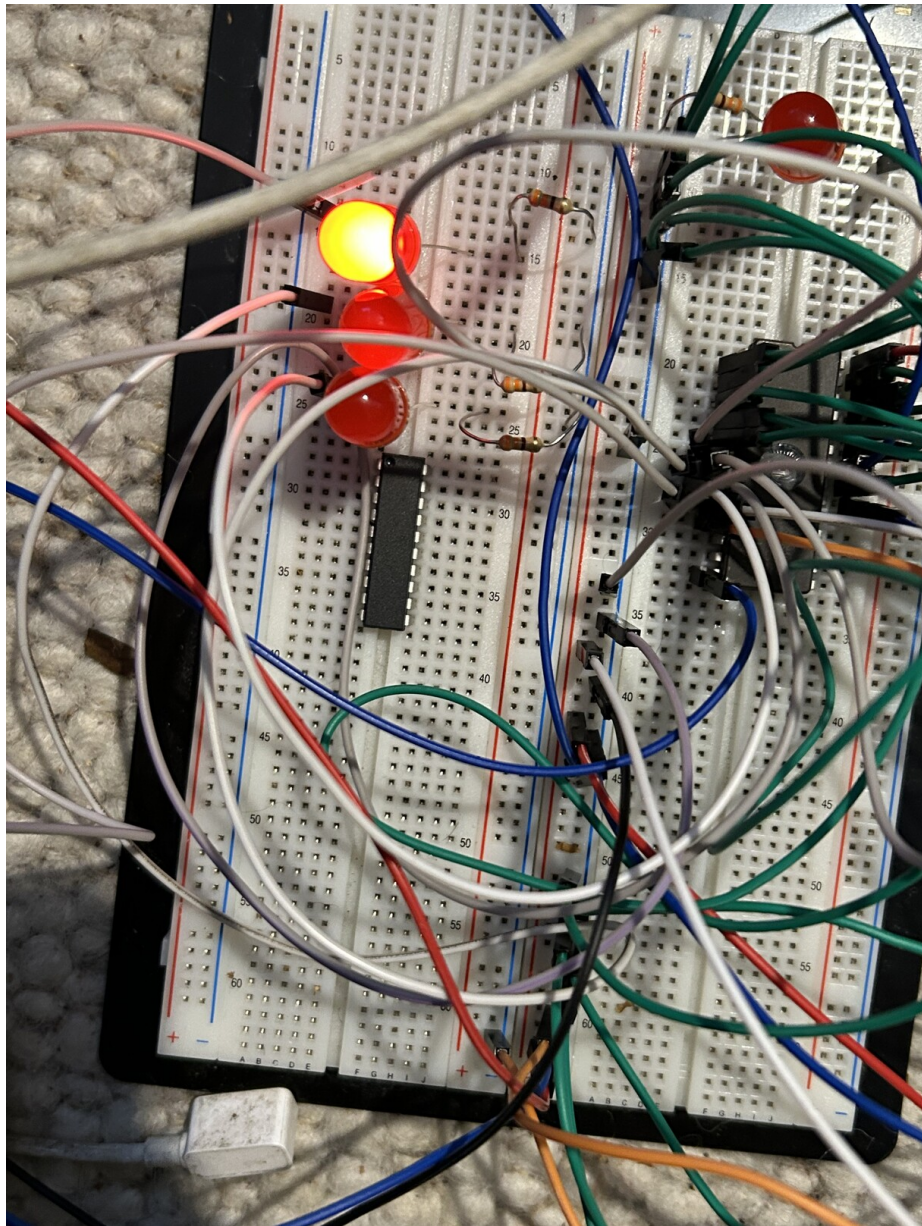


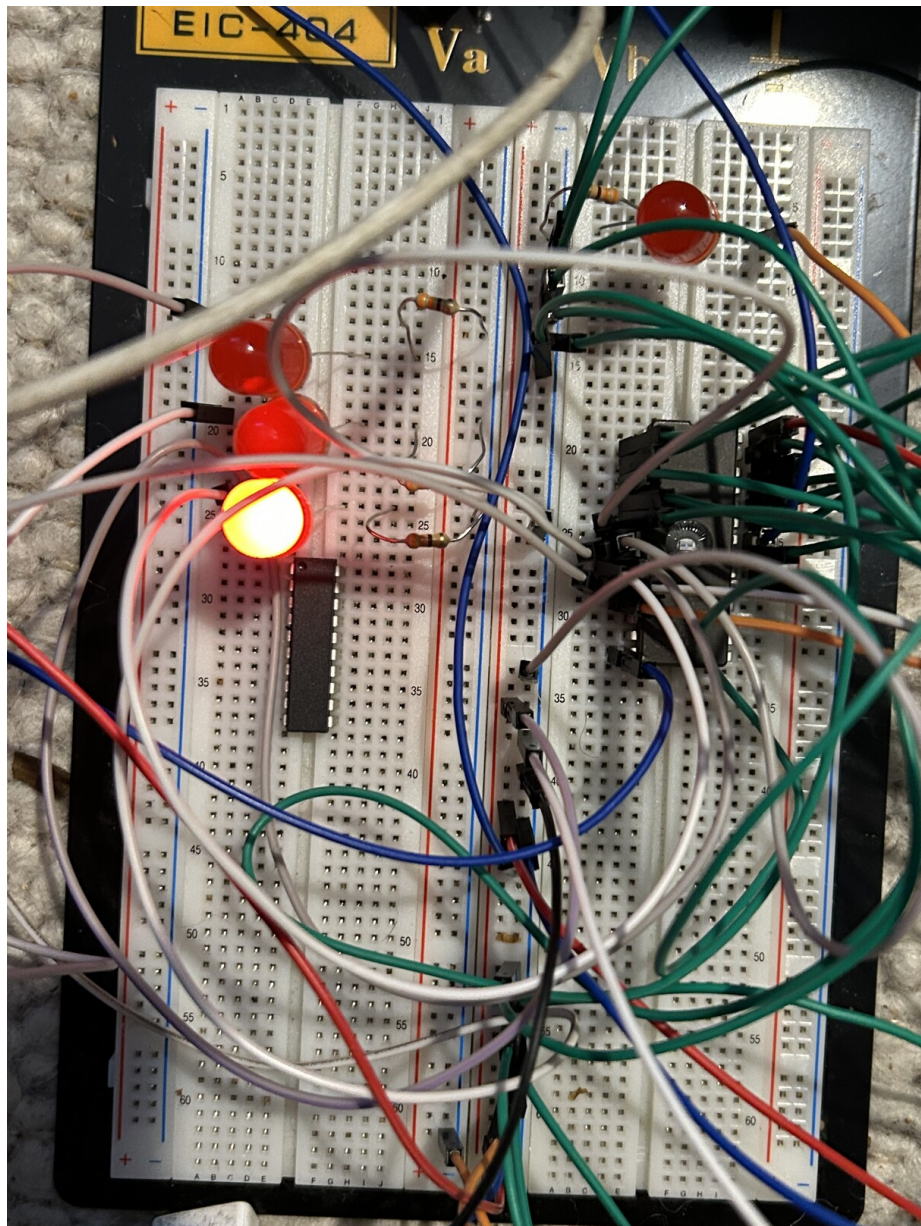


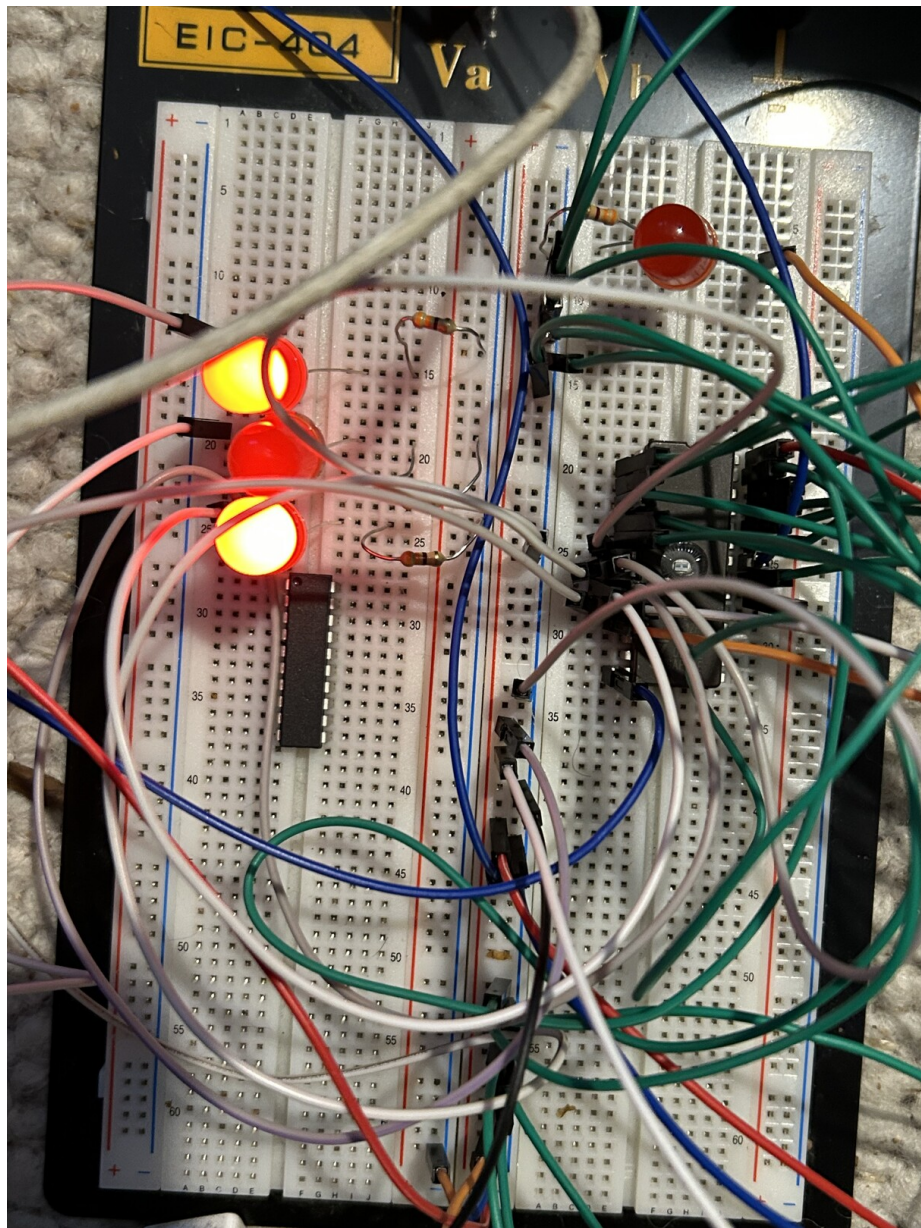


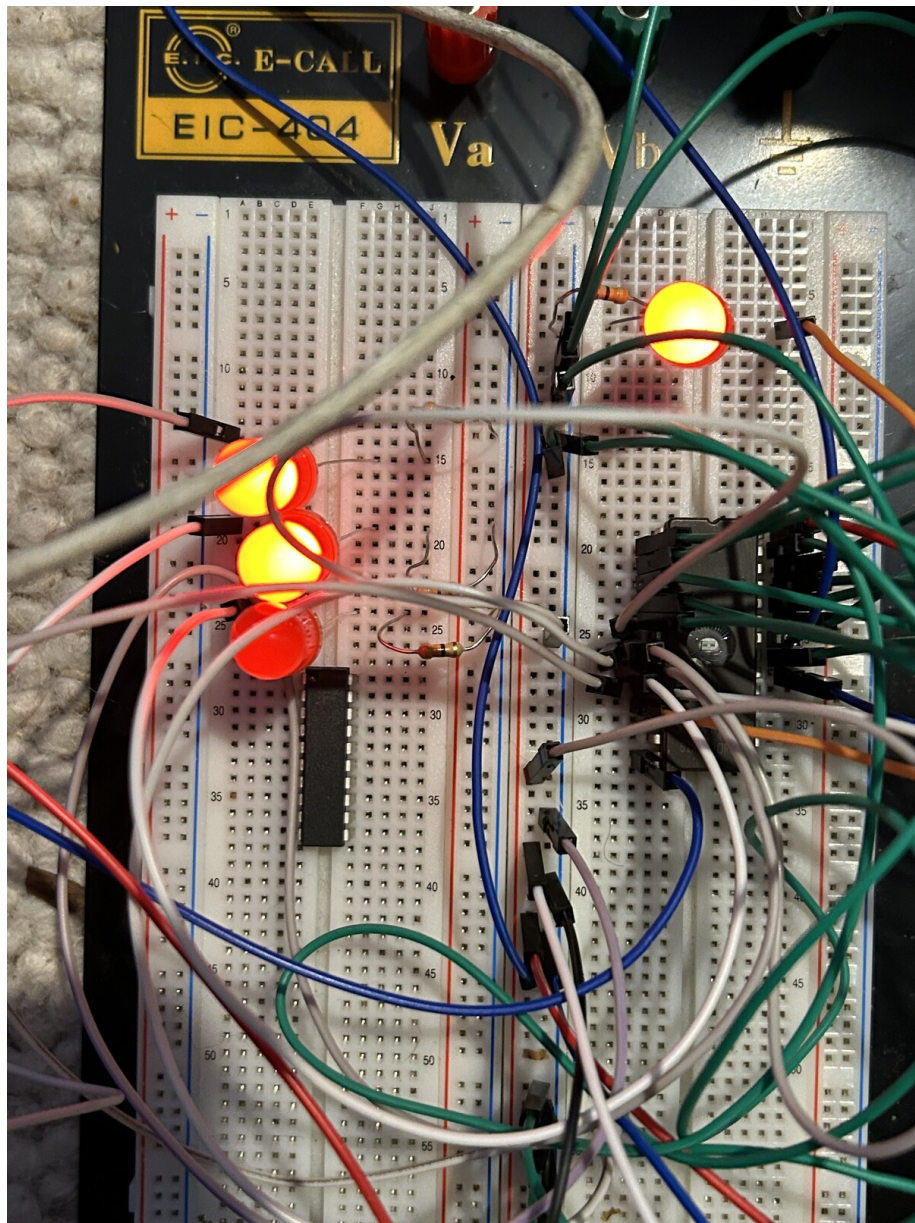








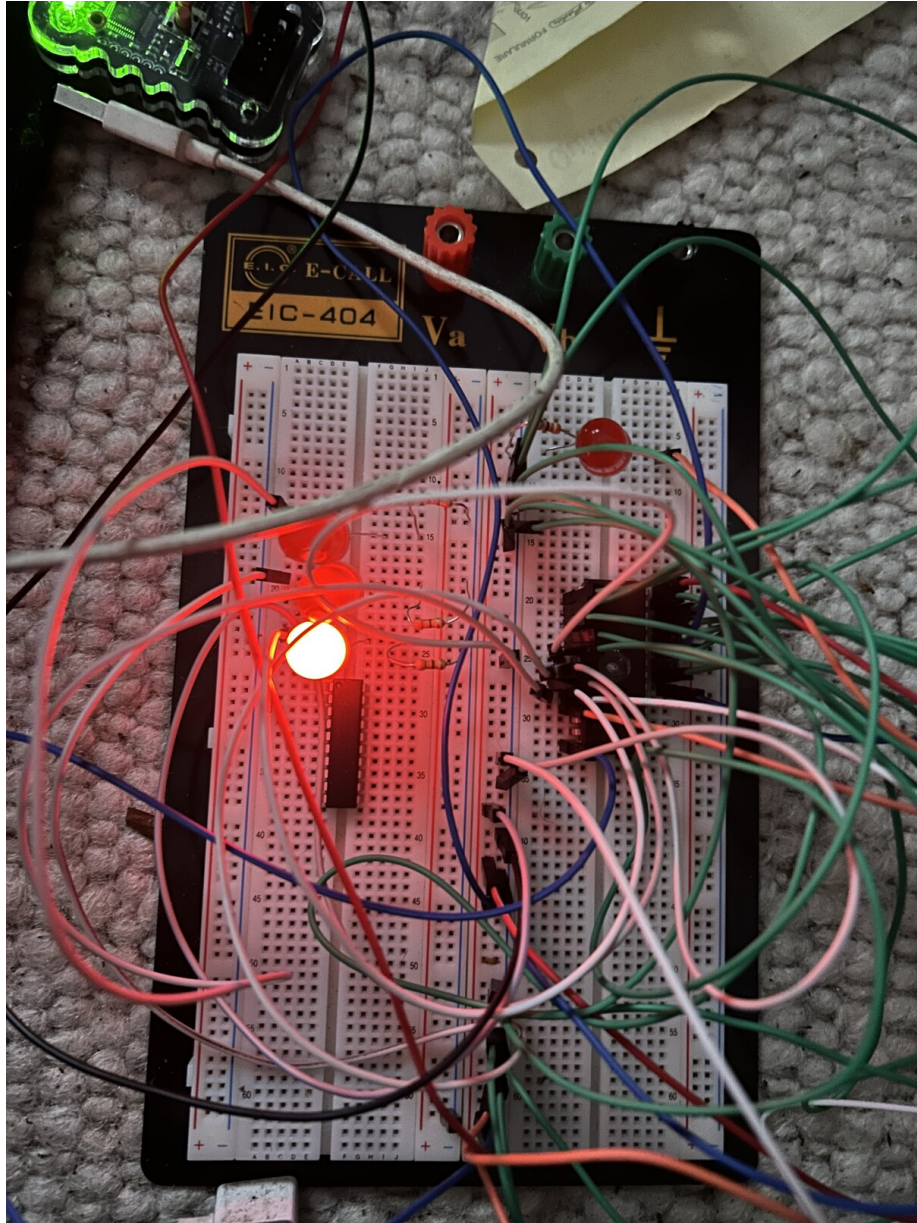


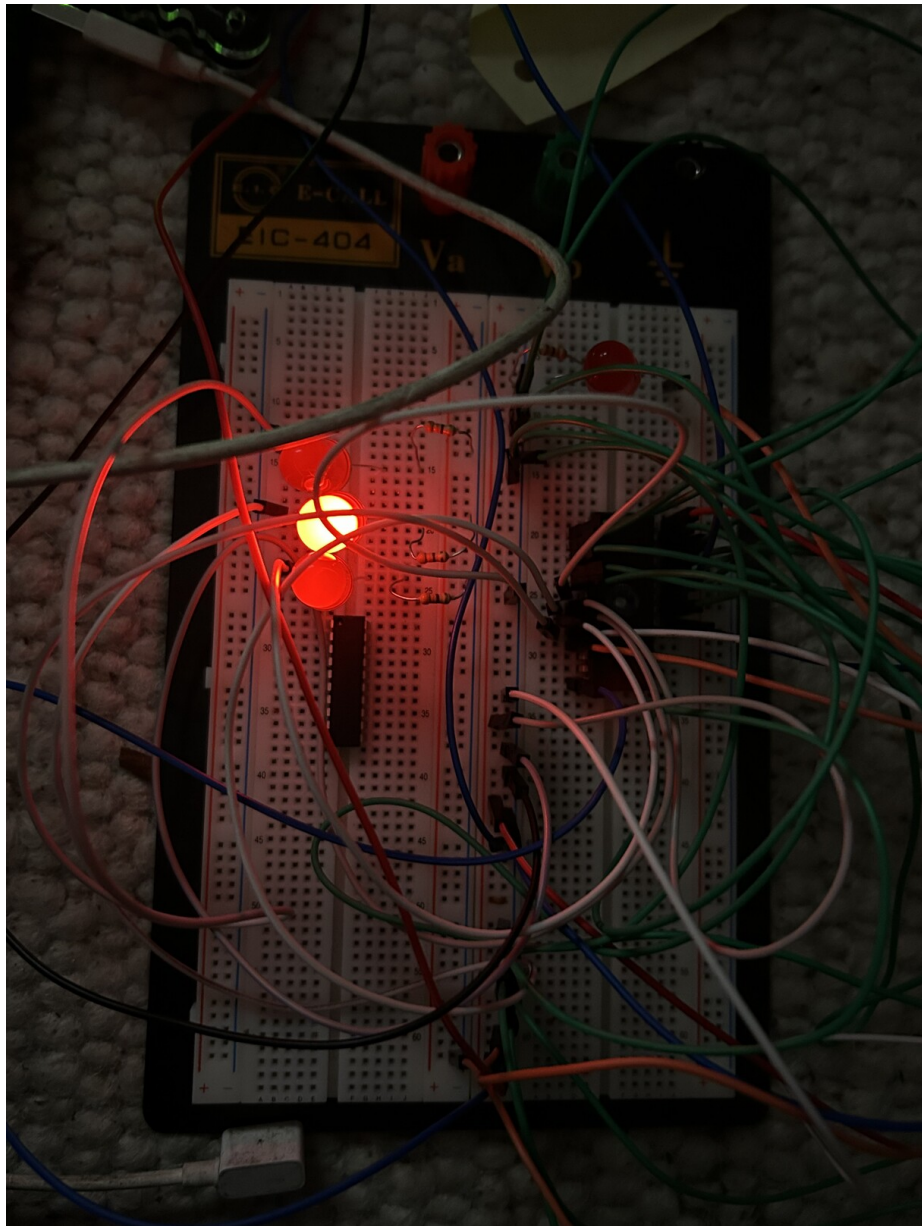


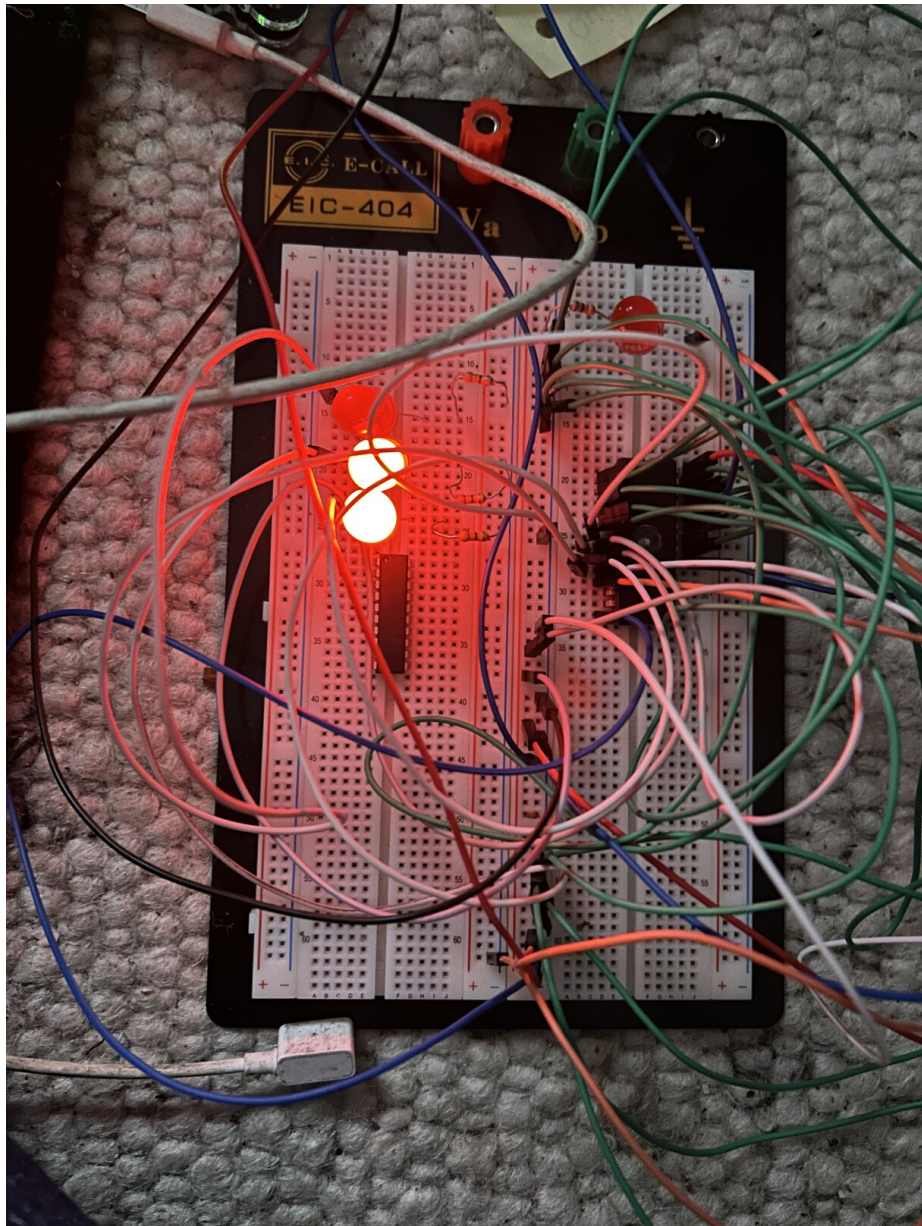
17 es wird folgen

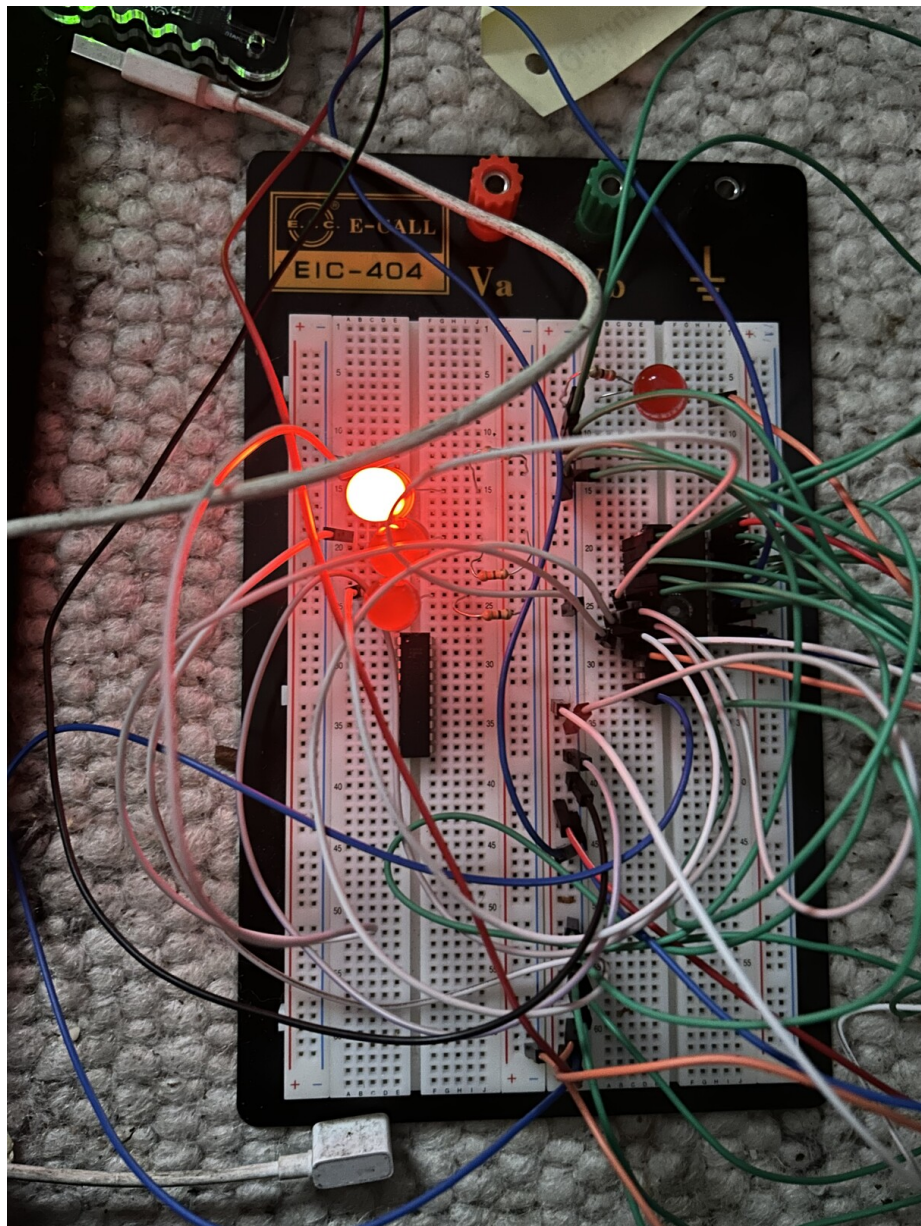
wenn das nun klar ist, kommt das Dekoder für 7-Segment und dann für 14 Segment. davor noch mal die DNF richtig. ist das geschehen folgt, die programmierung des GAL, so weit wie es tut. das ist weit gediehen, weil das AND Array funktioniert, muss gucken wo noch problem.

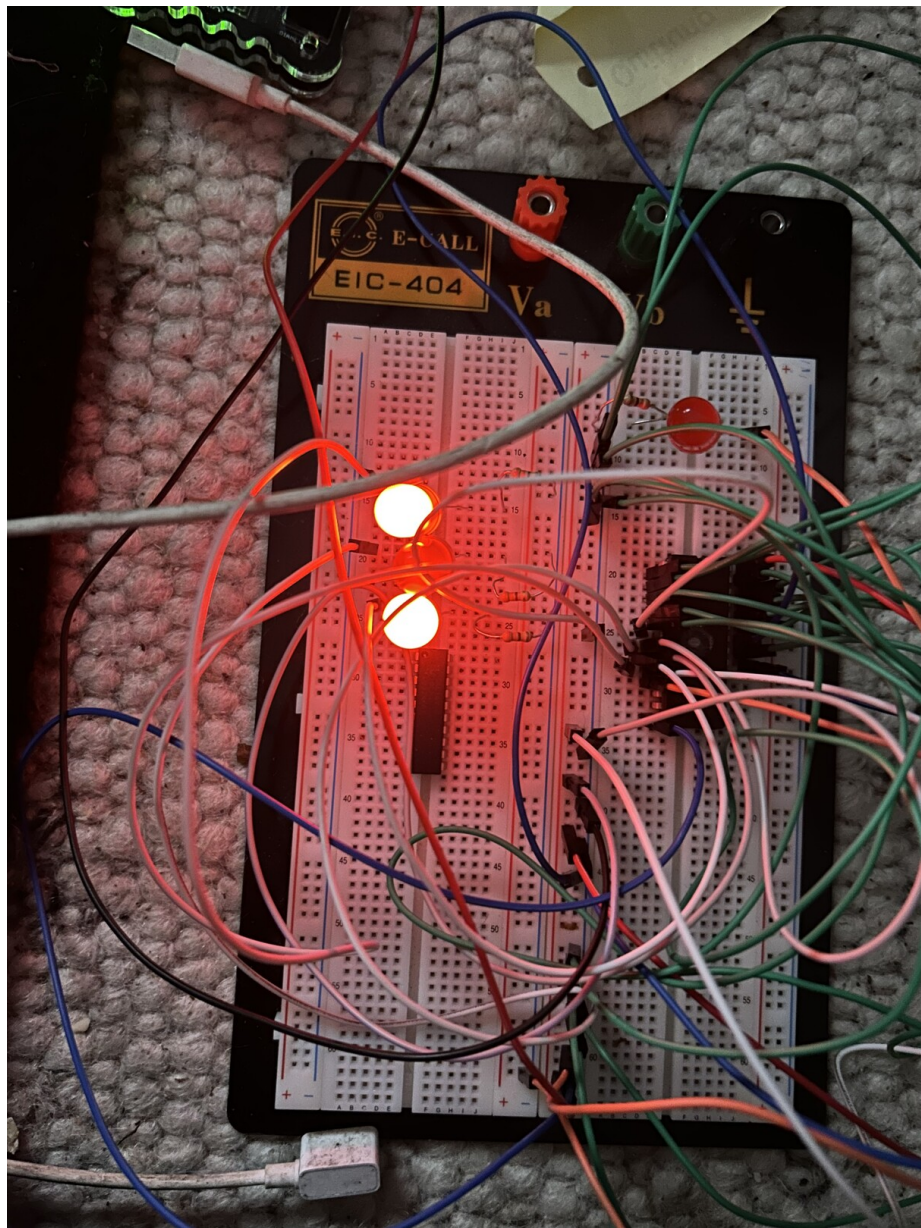
18 dritter Versuch

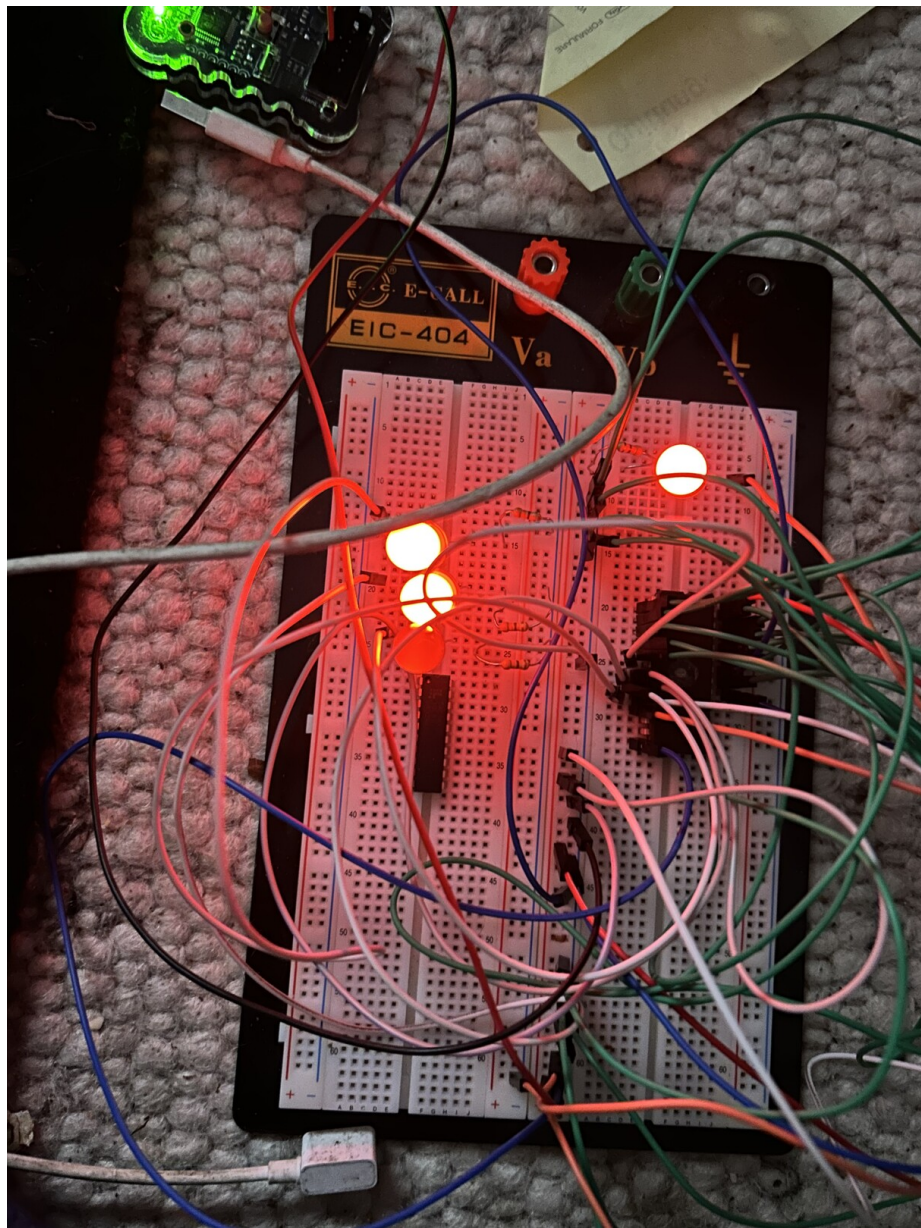


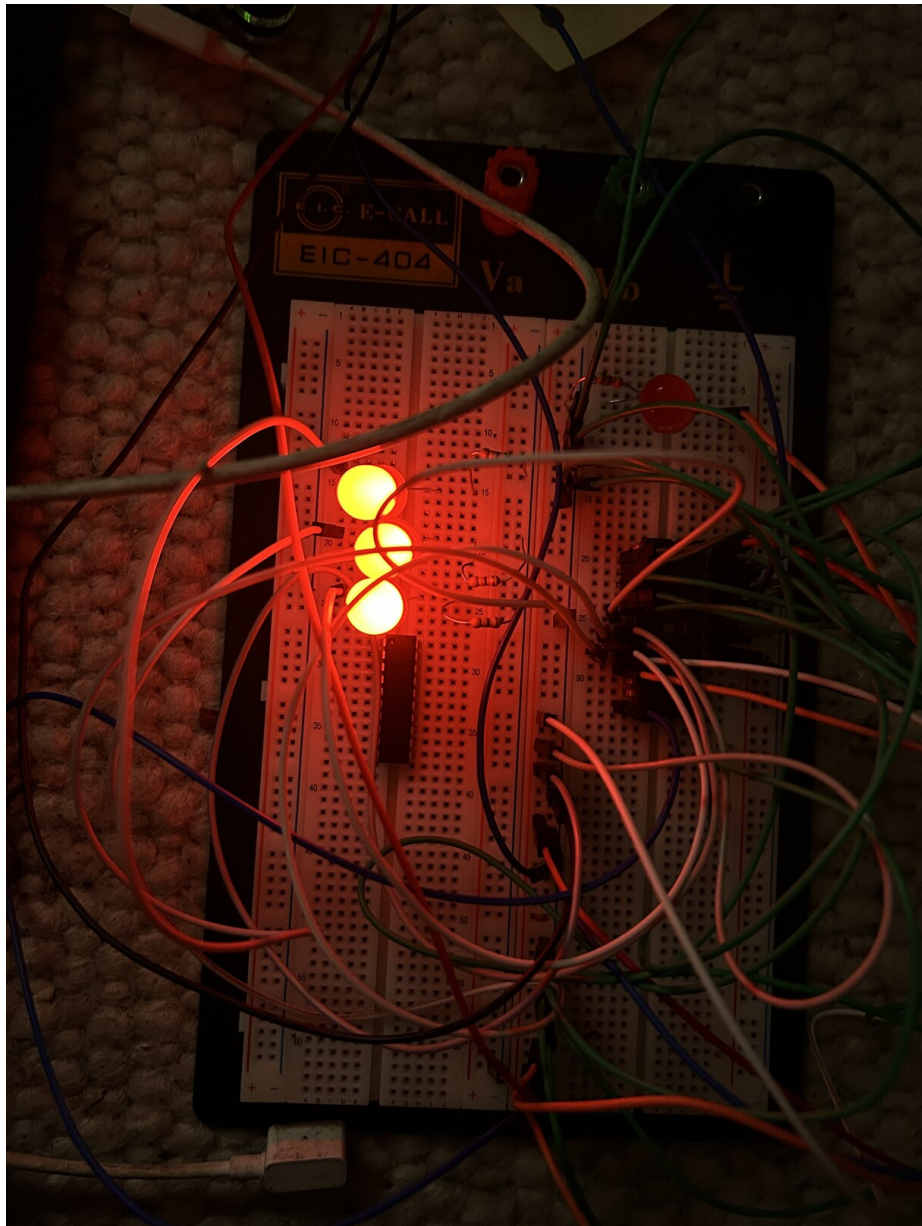


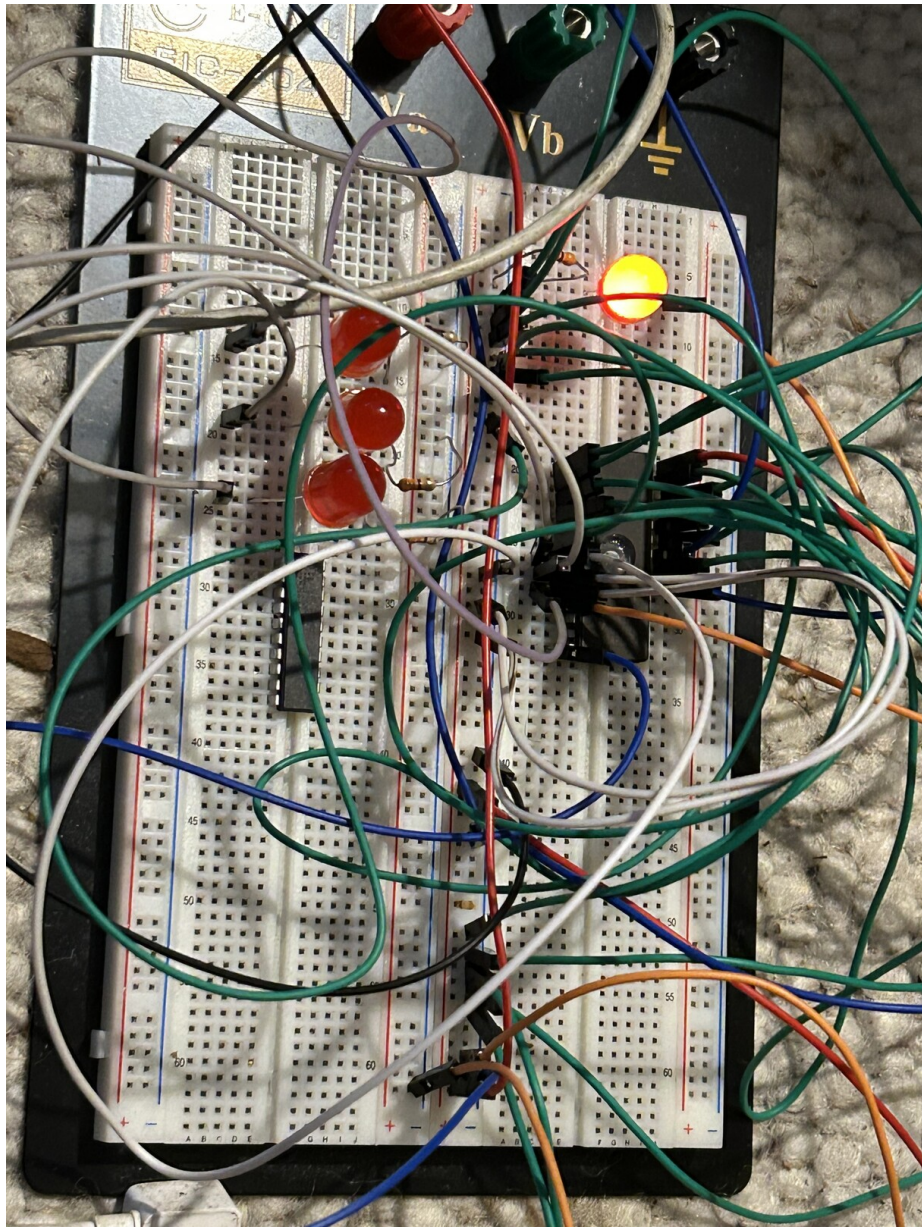












19 vierter Versuch

