

Entwicklung eines Kennlinienschreibers

Mit dem Arduino lassen sich nur begrenzt alle Wünsche erfüllen. Er ist einfach zu langsam bei der Ansteuerung des Display. Dennoch ist er keine schlechte Wahl für den Einsatz eines Mikroprozessors. Denn er ist preiswert und man muss ja nicht den Komfort eines Gerätes von Tektronix haben. Die zeigen die Kennlinie korrekt an. Nämlich die tatsächlich vorhandene Kollektorspannung im Messaugenblick. Da sie einen Spannungsabfall am Kollektorwiderstand abfragen, sieht man die Verkürzung der Kurven mit steigendem Kollektorstrom. Das ist hier in diesem Projekt nicht so, da ich die Kurve in X-Richtung bis zum Ende des Bildschirmes ausfahre. Dennoch ist die Genauigkeit nicht geringer, denn bis zum Erreichen der max. Kollektorspannung sollte da kein Unterschied im Verhalten des Transistors zu vermuten sein. Das von mir gewählte Prinzip ist einfach leichter zu realisieren. Denn aufwendige Schaltungen verwenden einen DAC zur Gewinnung der Kollektorspannung. Und dann ist im Augenblick der Stromabfrage die Kollektorspannung absolut synchron. Ich habe das Konzept aber nicht weiter verfolgt. Eine Testschaltung mit zwei Synchronzählern SN74HC161 war der 8-Bit-Zähler. Der ADC war ein M140808 von Motorola. Der Aufwand war hoch, das Resultat kritisch zu handhaben.

Eigene Idee

Ich habe nach einigen Versuchen einen anderen Weg gefunden. Ich sagte mir, dass es doch möglich sein sollte, mit einer einfachen Schaltung einen Sägezahn für die

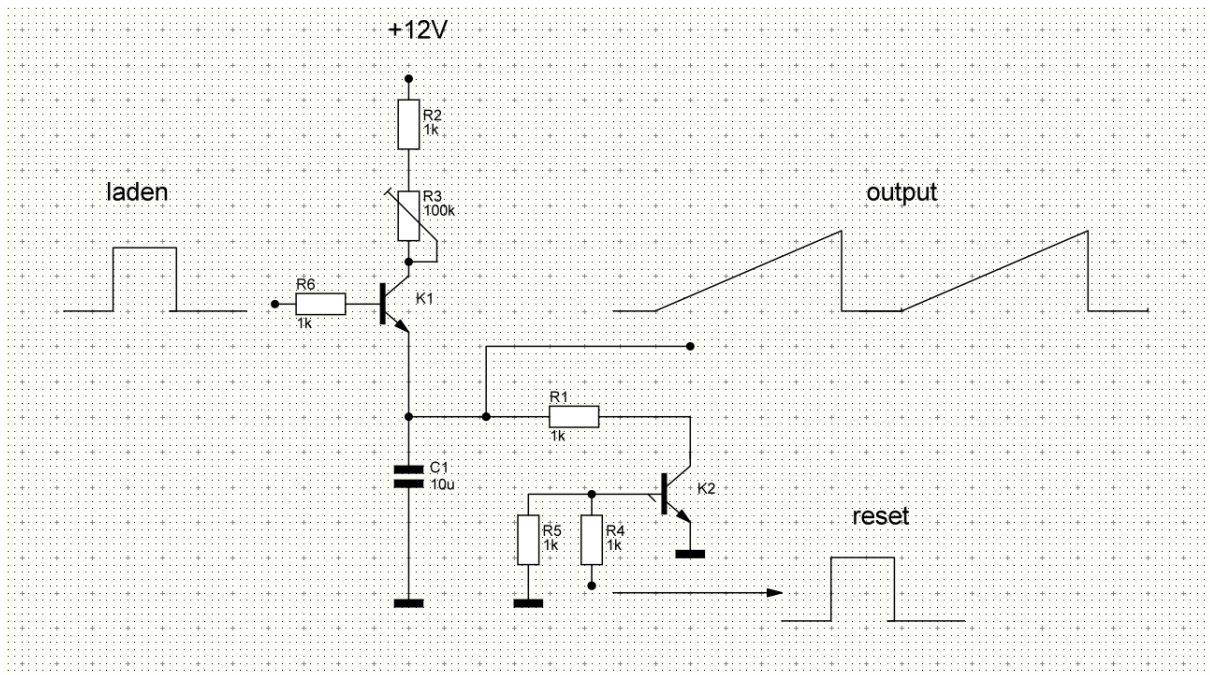
Kollektorspannung zu erzeugen. Mit dem portionsweisen Laden eines Kondensators wird dieser gefüllt. Wenn man einen großen Kondensator nimmt, wird er innerhalb der Zeit einer Kennlinie(volle X-Achse) nur im linearen Bereich der Ladekurve geladen. Die ist aber zu Beginn stets sehr gerade im Verlauf. Den "eingepprägten" Strom gewinnt man durch das Vorschalten eines großen Widerstands. Und tatsächlich war der Erfolg auf dem Oszillografen sofort sichtbar.



Das Schreiben einer Linie dauert ca. 700 ms. Wenn man 6 Kurven übereinander schreibt, vergehen max. 50 s. Nach dieser Zeit bleibt das Bild erhalten und wird erst neu geschrieben, wenn man den Messvorgang mit einer Taste neu startet. Das Bild bleibt so flimmerfrei und die Steuerung ist mit einem Arduino NANO sehr gut möglich.

Die Erzeugung der Kollektorspannung

Im Entwurf sieht man die einfache Schaltung. Sie funktioniert tatsächlich. Durch die Linearität ist eine synchrone Abfrage

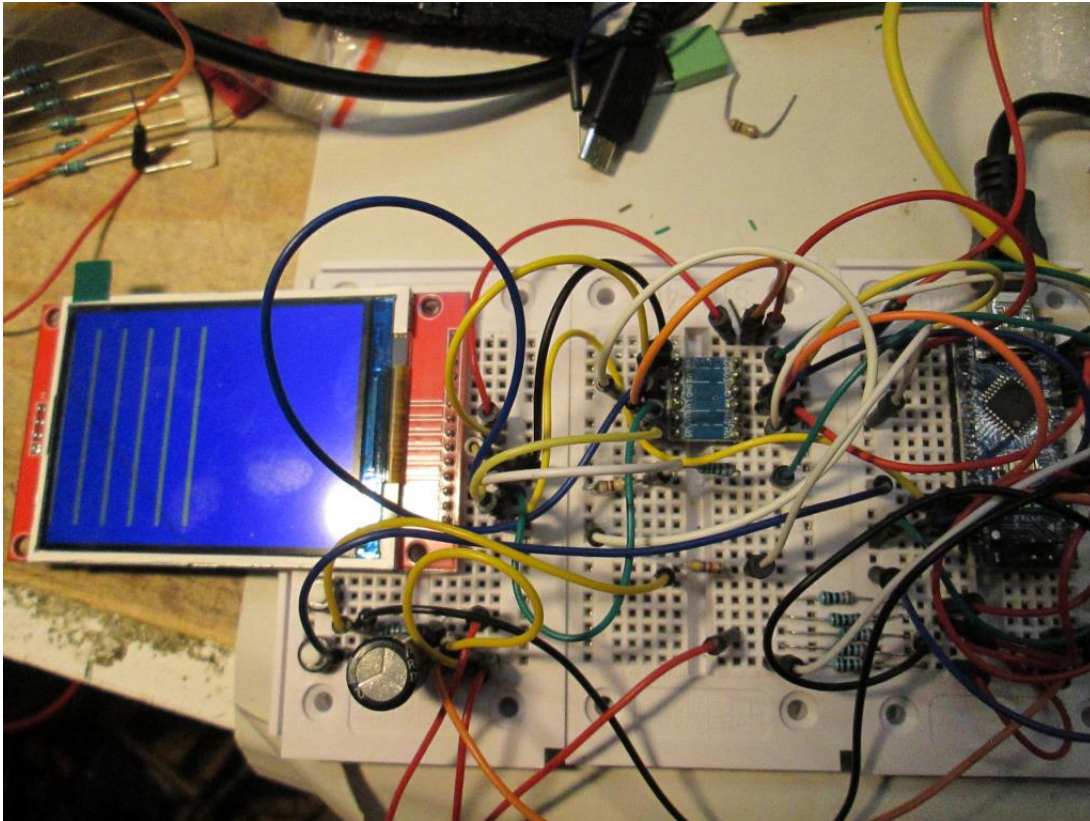


des Kollektorstroms gegeben. Am Beginn jeder Linie erfolgt die Synchronisierung. Zeitliche Abweichungen würden zum falschen Messpunkt führen. Die Fehler sind aber so gering, dass die Charakteristik des Transistors nicht verfälscht wird. Man sieht, was man sehen möchte. Der Aufwand ist gering. Diese X-Spannung wird mit zwei OPs polaritätsgerecht verstärkt und einer Transistorleistungsstufe(+15V/-15V) zugeführt. Zum Einsatz kommt der uA741C, der +/- 20V DC Versorgung verträgt. Nur mit mindestens +/- 15 kommt man auf einen Spitzenspannung des Sägezahns von 12 Vpp. Damit sind die üblichen OPs(LM358) bereits an der Grenze.

Kollektorstrom

Er wird synchron zur „Treppenstufe“ in X-Richtung 200 mal erfasst. Die Spannung am analogen Port A0 wird dazu abgefragt und in eine Y-Position umgerechnet. Dann erfolgt

das „Malen“ eines Punktes. Ich habe diesen Punkt etwas größer gestaltet, damit eine gut sichtbare Linie entsteht. Im Test zeigen die Linien sich so wie im folgenden Bild.

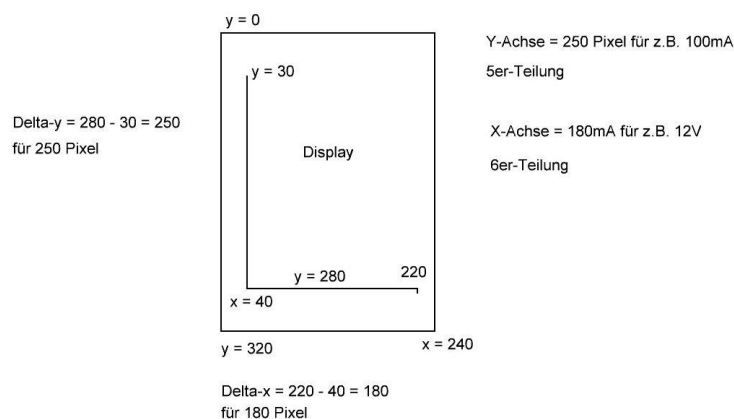
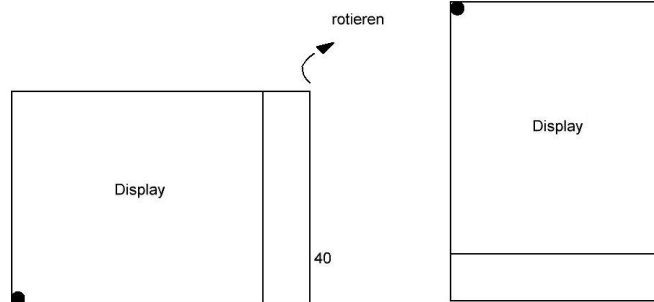
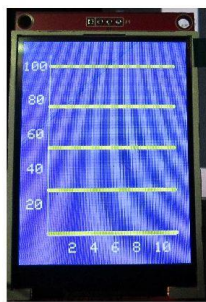


Das „Malen“ eines Punktes dauert 2ms. Somit für eine Linie $240 * 2 = 480$ ms. Hinzu kommen noch die Steuerimpulse, sodass nach etwa 700ms eine Kennlinie geschrieben wird. Stellt man nur zwei dar, um z.B. die Stromverstärkung zu bestimmen, verkürzt sich die Zeit für ein Frame auf ca. 1,5 s. Ich meine, dass diese Reaktionszeiten akzeptabel sind. Auf die Anzeige eines Widerstandswertes muss ich bei meinen Multimetern manchmal länger warten. Wem diese Zeiten zu lang sind, kann sie in der Software noch weiter verkürzen(`delay()`; ändern).

Über das Display

Elegant wäre ein Display mit Touchscreen. Gibt es und könnte man auch einsetzen. Man könnte Tasten einsparen. Ich lege aber Wert auf eine übliche Bedienung und brauche den Aufwand nicht. Tippe schon genug auf dem Handy rum, hi.

Das Display ist im Normalzustand so eingerichtet, dass der Ursprung für x und y unten links in der Ecke ist. Weil ich aber möglichst viele Kurven aufzeichnen möchte, klappte ich es in die senkrechte Lage. Der Projektionshintergrund rotiert nach rechts, aber die Ausrichtung der Schriftzeichen bleibt erhalten. Man kann also die beiden Koordinaten lesbar in der gewohnten Ausrichtung beschriften. Leider „wächst“ der Y-Wert (Pixel) nach unten! Aber der X-Wert verhält sich wie gewohnt und wächst nach rechts. Das folgende Bild zeigt den Zusammenhang.



Man muss sich an diesen Nachteil gewöhnen. Insbesondere ist ein Umdenken beim Programmieren erforderlich. Wer also unbefangen daran geht, sofort einen Code zu schreiben, muss event. anschließend mit viel Anstrengung alles wieder ändern. So ging es mir, weil ich mich nicht sonderlich mit dem Display befassen wollte. Doch dann musste ich es,hi.

DAC

Inzwischen entdeckte ich preiswerte DACs beim Reichelt. Sie werden mit dem SPI vom Arduino angesteuert und geben max. 4V heraus. Es ist der

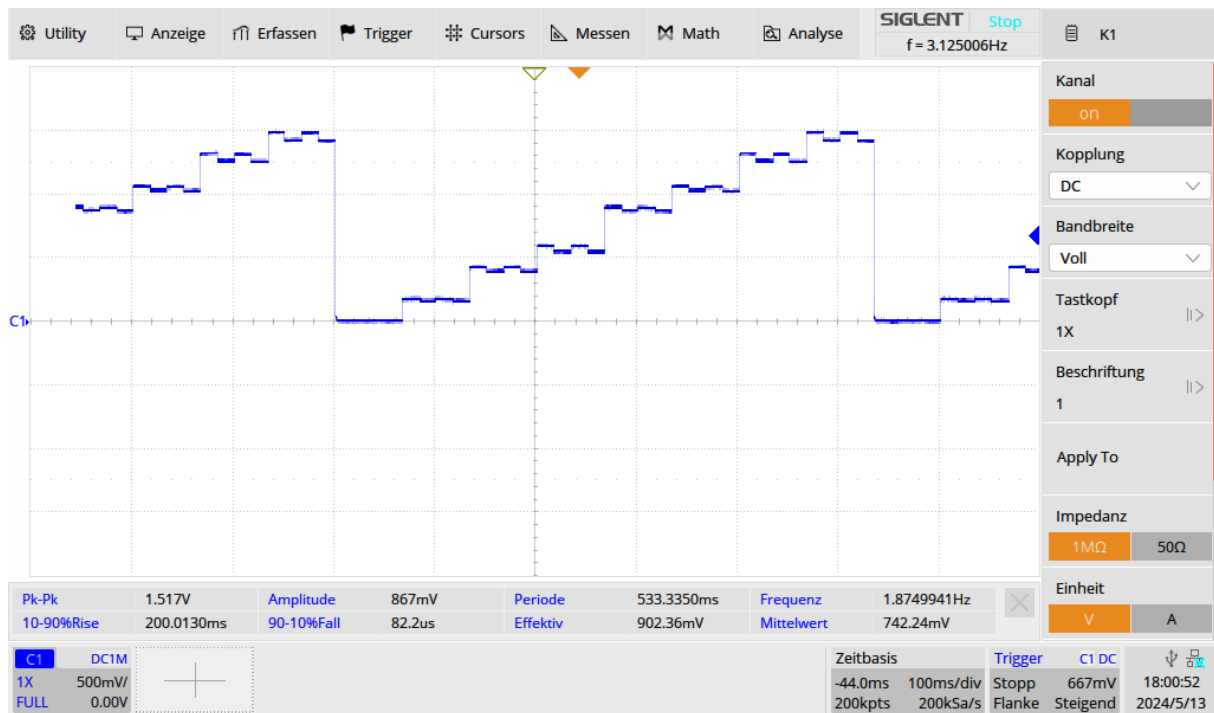
MCP 4802-E/P.

Kostet nur 2,30 EUR, leider mit 5,90 EUR Versand.

Den werde ich doch mal testen. Scheint das Projekt zu vereinfachen.

DAC mit R2R-Netzwerk

Hier mal ein Versuch mit einem präzisen R2R-Netzwerk von Bourns. Es wurde mit einem CD4029 angesteuert. Allerdings nur mit 4 Bits. Und leider war das Netzwerk sehr hochohmig mit 100k-Widerständen aufgebaut. Natürlich nur mit 1M „belastet“. Man sieht, dass die Stufenspannungen nicht sehr „glatt“ sind. Das könnte bei der Erfassung des Kollektorstroms zu kleinen Abweichungen führen, die sich als „unruhige“ Kennlinie zeigen. Eine Mittelwertbildung wäre deshalb unverzichtbar. Mir ist deshalb ein anderes Prinzip lieber.

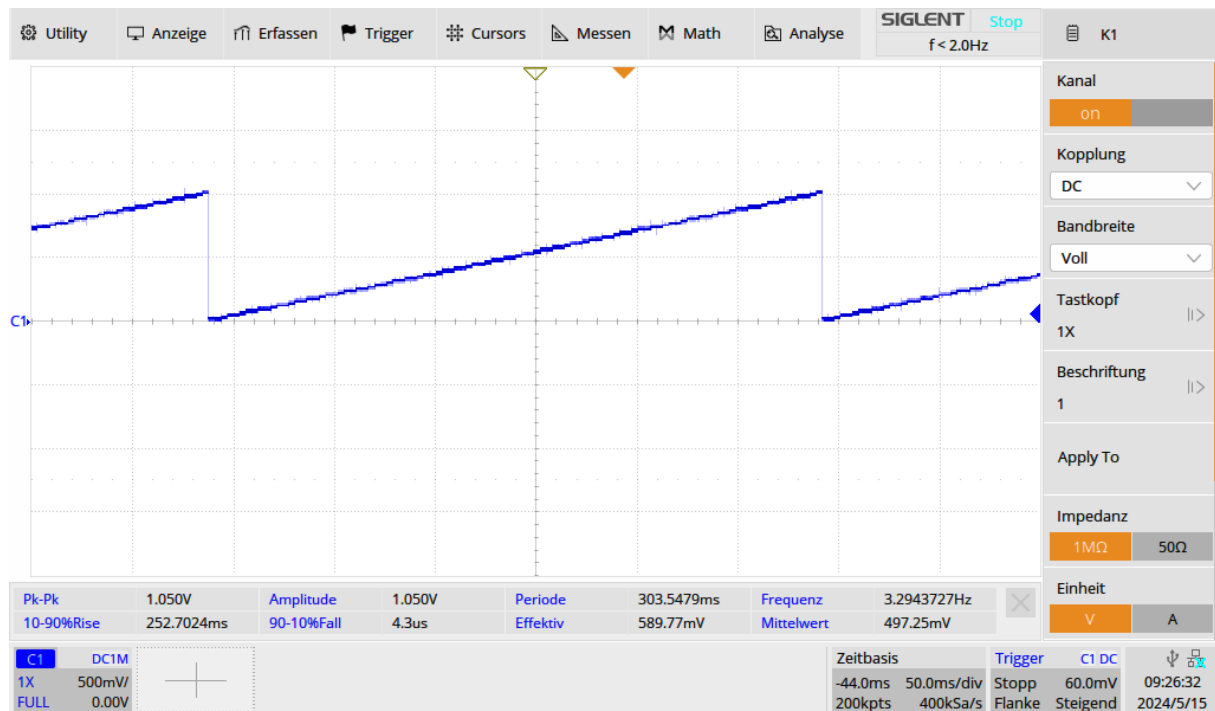


Die Stepwerte sind ungleich. Die „Belastungen“ der Ausgänge waren mit 1k und einer LED. Es könnte sein, dass hier unterschiedliche Spannungsabfälle im Spiel waren. Man erkennt, dass man solche Netzwerke besser mit dem hochohmigen Input eines OPs „belastet“ und vor dem Einsatz gründlich prüft. Acht Stufen sind für die Erhöhung des Basisstroms für die Anzahl der Kennlinien ausreichend. Ich werde solche Treppenspannungen mit dem OP und einigen Kondensatoren noch gründlich „sieben“, damit im Messpunkt eine beruhigte Gleichspannung vorliegt. Eine Spannungs-/Stromwandlung ist mit einem OP elegant zu machen.

DAC: MCP4802

Dieser DAC macht die Erzeugung eines Sägezahns sehr einfach. Ich habe auch eine Library entdeckt, die auf den Chip zugeschnitten ist. Sie funktioniert allerdings mit einem ähnlichen IC aus der Reihe. Macht aber nichts, denn jetzt ist

die Programmierung wesentlich einfacher. Ich werde die ZIP-Library für den Download beifügen, sodass sie auch in späteren Zeiten noch verfügbar ist. Hier ein Bild des Sägezahns für die X-Ablenkung(Kollektorspannung):



Die kürzeste Verweilzeit auf einer Stufe ist 5ms. Das reicht, um eine sichere Abtastung des Kollektorstroms zu machen. Der Durchlauf einer Kennlinie ist $6 \times 50\text{ms} = 300\text{ms}$. Also wird das Schreiben eines Frames noch kürzer als zuvor.

Der gleiche Baustein kann natürlich auch für die Erzeugung der Treppe für den Basisstrom verwendet werden.

Zwischenstand

Nun sind die wesentlichen Probleme eigentlich gelöst. Ich muss die Hardware basteln. Aber über die Möglichkeiten für PNP-Transistoren und Variationen der Spannungen und Ströme muss ich noch nachdenken. Es soll jedoch kein

Supergerät werden. Denn eigentlich habe ich selten einen Kennlinienschreiber benötigt. Er ist fast als Spielzeug einzuordnen. Dennoch ist es gelegentlich nützlich, zwei oder mehrere Transistoren zu vergleichen.

Fehlfunktion

Das Einfügen des Codes mit der Library

```
#include <MCP48xx.h>
```

funktioniert nicht gleichzeitig mit dem Aufruf des Treibers für das Display. Ja, ich habe den CS-Pin von 10 auf 7 geändert, weil ja 10 vom Display benutzt wird. Und ich habe den Aufruf der SPI-Library für das Display gesperrt, weil diese ja in der anderen Library inklusive aufgerufen wird. So wird ein doppelter Aufruf vermieden.

Alle Anstrengungen, das Programm zum Laufen zu bringen, scheiterten. Deshalb suchte ich nach einer Möglichkeit im Netz, den DAC ohne besondere Library zu bedienen. Und ohne die Konflikte. Leider findet man im Internet nur Schrott. Die Autoren machen die primitivsten Flüchtigkeitsfehler und meisten wollen sie mit besonderen Spitzfindigkeiten glänzen. Und also besann ich mich auf ChatGpt. Hatte sofort Erfolg, denn man stellte mir einen kompletten Code zur Verfügung. Doch auch hier wieder umständliche Eingabe mit der seriellen Schnittstelle in der DIE. Das musste ich alles mühsam entfernen, bis dann endlich nach der dritten Anfrage ein brauchbarer Code geliefert wurde. Zwischendurch waren dabei einige Beispiele, die überhaupt nicht liefen. ChatGpt nannte auch die vermutlichen Ursachen und machte

Vorschläge . dennoch war der Einsatz von KI hier wesentlich zielführender als die andauernde Suche im Netz. Kann ich nur jedem empfehlen, der Probleme mit der Codierung hat. Jetzt läuft das Programm mit dem Display und dem DAC MCP4802. Ich habe den gewählt, weil er im DIP-Gehäuse verfügbar ist. Besser und preiswerte ist aber der MCP49xx, den es leider nur im SO-Gehäuse(1,30 EUR) bei Reichelt kaufen gibt. Ein 12-Bit-Wandler, wir brauchen aber nur 8 Bit.

Hier mal ein Testprogramm mit dem MCP4802. Man kann die Analogspannung am PIN 6 (Output B) als Sägezahn am Oszilloskop beobachten:

```
#include <SPI.h>

// Chip Select Pin
const int CS_PIN = 7;

// Beispielwert
uint16_t valueA = 512; // Halber Wert für DAC A
uint16_t valueB = 2047; // Maximaler Wert für DAC ; kommen 220mV
heraus

int ux;

int wert = 0;

void setup() {
  // SPI initialisieren
  SPI.begin();
  // Chip Select Pin als Ausgang definieren
  pinMode(CS_PIN, OUTPUT);
  digitalWrite(CS_PIN, HIGH); // CS auf HIGH

  Serial.begin(9600);
}

void writeToDAC(uint16_t value, bool channelA) {
  // Wert in das richtige Format bringen
  // Bit 15: 0 = Write to DAC A, 1 = Write to DAC B
```

```

// Bit 14: 1 = Enable Buffer
// Bit 13: 1 = 1x (GAIN=1), 0 = 2x (GAIN=2)
// Bit 12: 1 = Active Mode
uint16_t command = (channelA ? 0x3000 : 0xB000) | (value & 0x0FFF);

// Chip Select auf LOW setzen
digitalWrite(CS_PIN, LOW);

// Daten senden (16-Bit Kommando)
SPI.transfer16(command);

// Chip Select auf HIGH setzen
digitalWrite(CS_PIN, HIGH);
}

void uce (){                                     //(uint16_t){

// Chip Select auf LOW setzen
digitalWrite(CS_PIN, LOW);
sendToDAC(valueA, false); // true = Wert an DAC A ausgeben
digitalWrite(CS_PIN, HIGH);
delay(2);

//Serial.println(analogRead(A0)); // read the value from analog pin A0 and
send it to serial

}

void zahn(){

do
{
delay(2);
valueA = valueA + 10;
uce();
ux++;

}while (ux < 200);
valueA = 0;
ux=0;
}

void loop() {

uint16_t valueA = 0;

```

```
zahn();//uce(valueA);  
//Serial.println(analogRead(A0)); // read the value from analog pin A0 and  
send it to serial  
  
//exit(0);  
  
delay(1000);  
}
```

Leider läuft diese Software nur im Standalone und nicht in der Anwendung mit dem Display. Es hängt sich bei den Befehlen:

```
mcp4802.write(value, channel);  
oder  
  
sendToDAC(valueA, true);
```

auf. Es wird also kein Sägezahn geschrieben und das Programm stoppt. Da könnte man noch tagelang experimentieren. Nur mit der Originaldatei als Example ist alles OK. Das eine SPI_Modul deaktiviert das andere, was ja nicht sein sollte. Ich weiß aber nicht, wie ich es dem Compiler beibringen kann. Nach der Erzeugung einer Instanz beim Aufruf des Konstruktors vom DAC wird das Display blockiert. Ich kann darauf nichts schreiben. Die Autoren haben vermutlich nicht genug getestet. Auch ChatGpt konnte nicht helfen.

Im Netz gibt es noch einen Programmiervorschlag mit einem Touchscreen. Die Software ist sehr aufwendig. Und es wird eine Library verwendet, die ich nicht finden kann. Ich habe kein Lust mehr, die verfügbaren Libraries zu untersuchen. Kehre vielleicht zurück zu meiner o.b. Lösung mit dem Ladekondensator. Die Stromwerte(Stufen) für die Basis werde

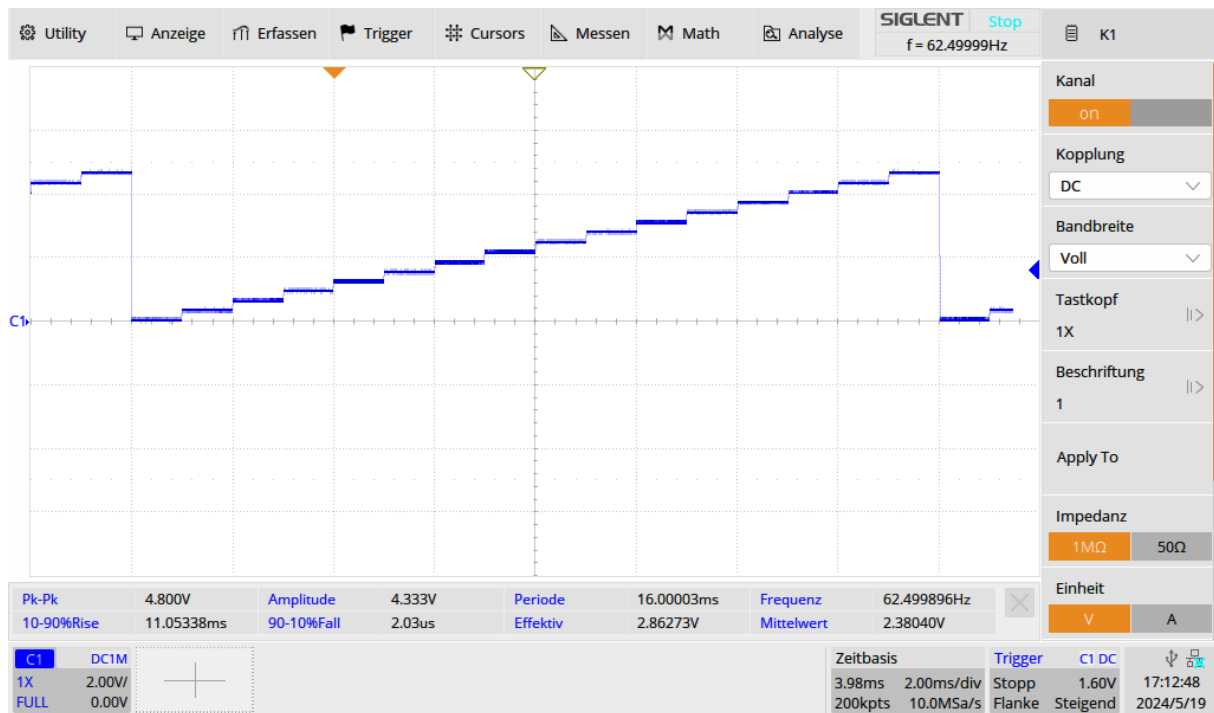
ich mit anderen DACs machen, nicht mit solchen, die umständlich über SPI bedient werden. Muss ja nicht sein.

Hier der Sägezahn, der mit dem MCP4802 erzeugt wurde. Am Anfang ist er nicht linear. Die ersten vier Stufen sind unterschiedlich im Pegel. Also muss man da Korrekturen vornehmen. Das ist auch nicht ohne Aufwand zu erreichen. Ja, ICs liefern nicht immer ideale Ergebnisse. Da ist meine Lösung mit dem Ladekondensator einfacher.



Eine Treppenspannung mit CD4029

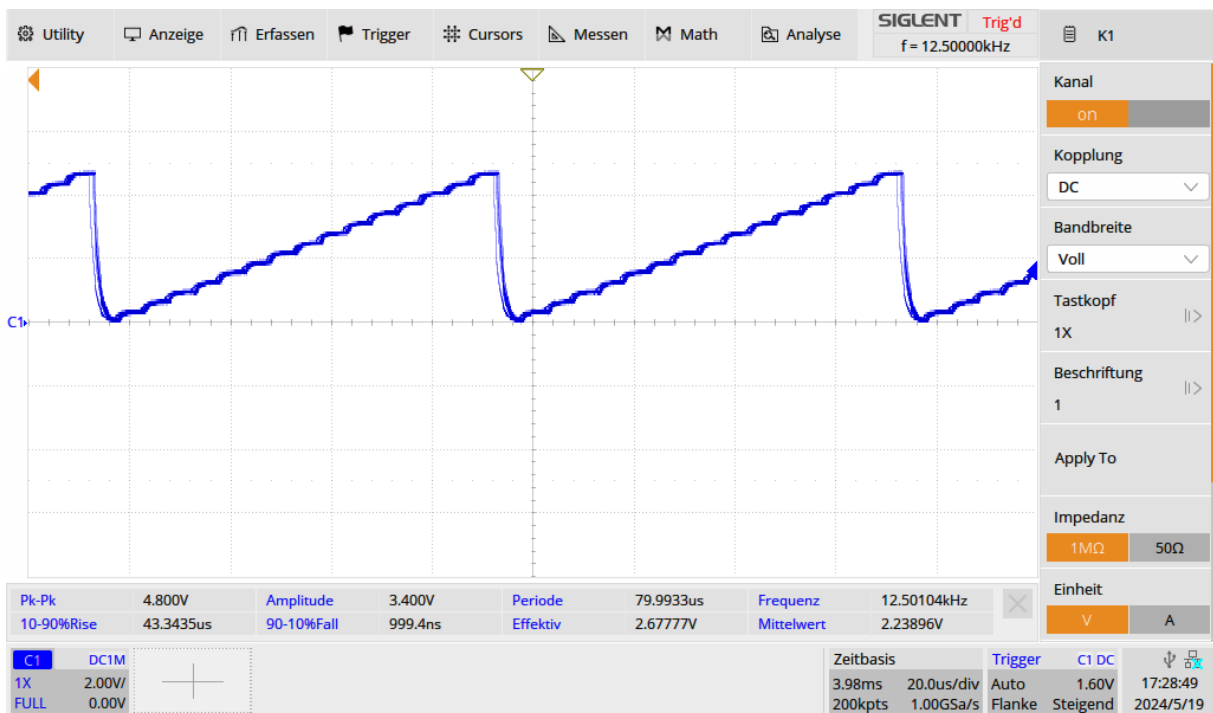
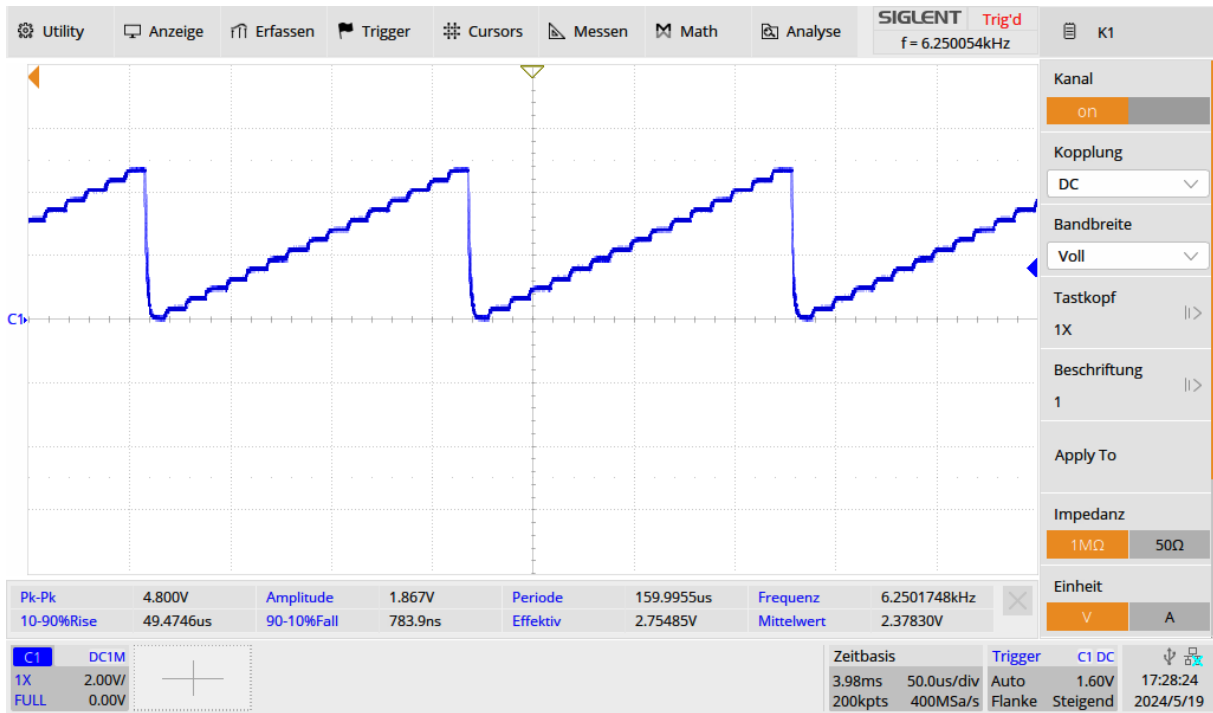
Habe mal mit einem CD4029(4-Bit-Zähler) ein 2R2-Netzwerk aufgebaut. Mit 1 kHz Takt zeigt sich folgendes Bild:



Es wurden 1%-Widerstände 22k/11k verwendet. Die Genauigkeit ist hervorragend. Alle 15 Stufen sind gleich. Der Aufwand ist akzeptabel. Ich werde diese Schaltung für die Basisstufen(...Ströme) für max. 8 Kennlinien nehmen. Ob sich das auch für die 200 Stufen für die Uce lohnt, werde ich ausprobieren. Habe hier noch monolithische 2R2 von BB rumliegen(100k).

Wie hoch darf der Takt werden?

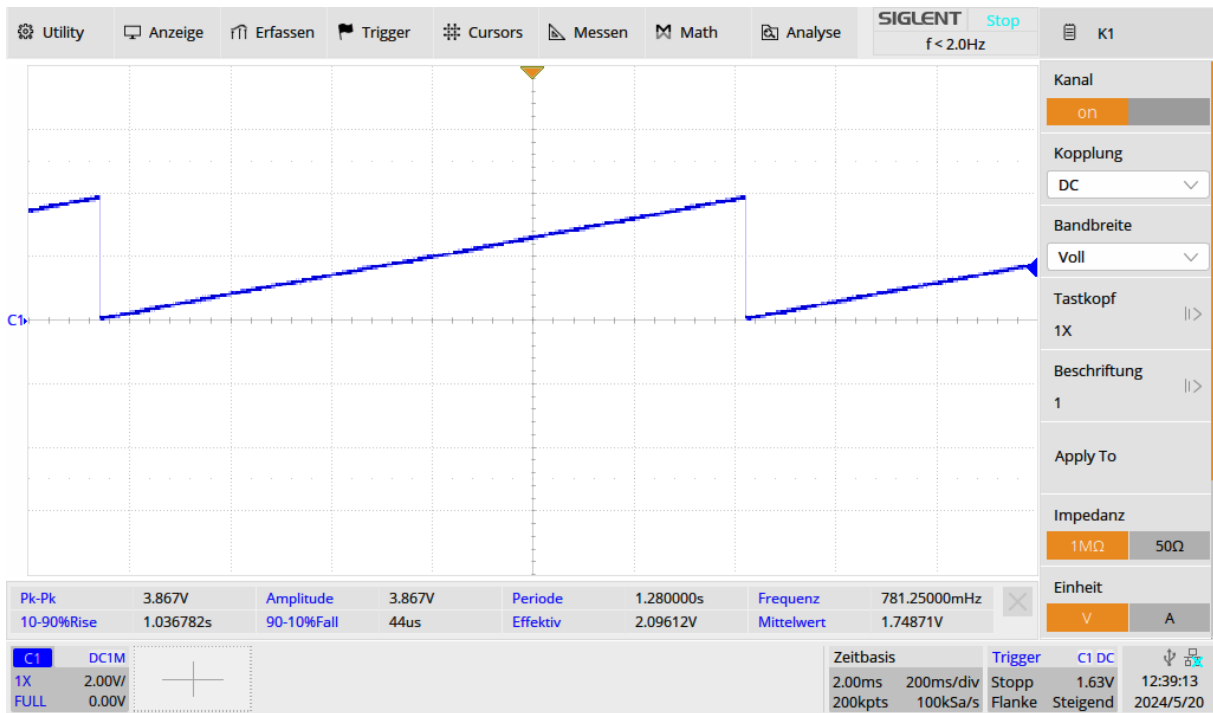
Im ersten Bild 100kHz, im zweiten 200 kHz. Die Stufen sind immer noch deutlich erkennbar, die Genauigkeit nicht zu beanstanden. Also könnte man die Schaltung auch für die Erzeugung von 200 Treppen einsetzen und damit eine Kennlinie schreiben.



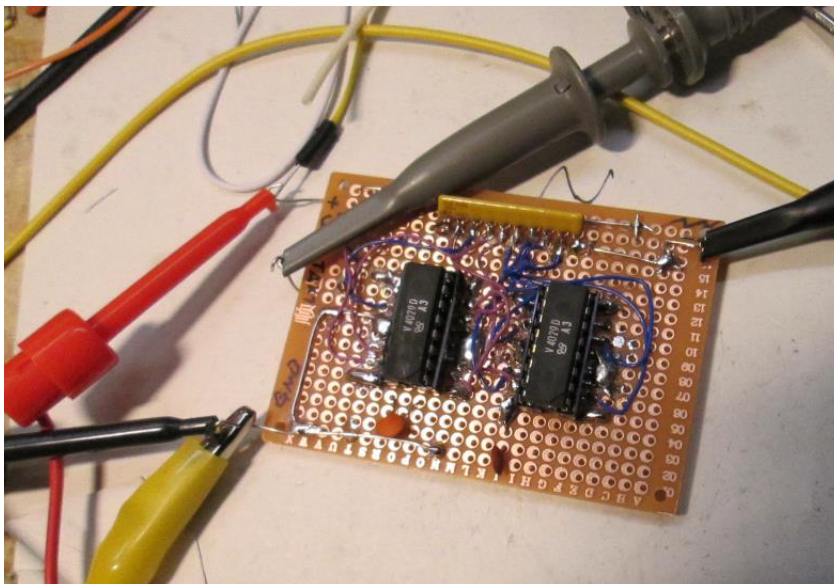
DAC für die Uce

Und nun die Schaltung für die Erzeugung der Kennlinie. Die X-Achse soll 200 Pixel haben. Es müssen also 200 Treppenstufen erzeugt werden. Das geht bei ca. 200 Hz

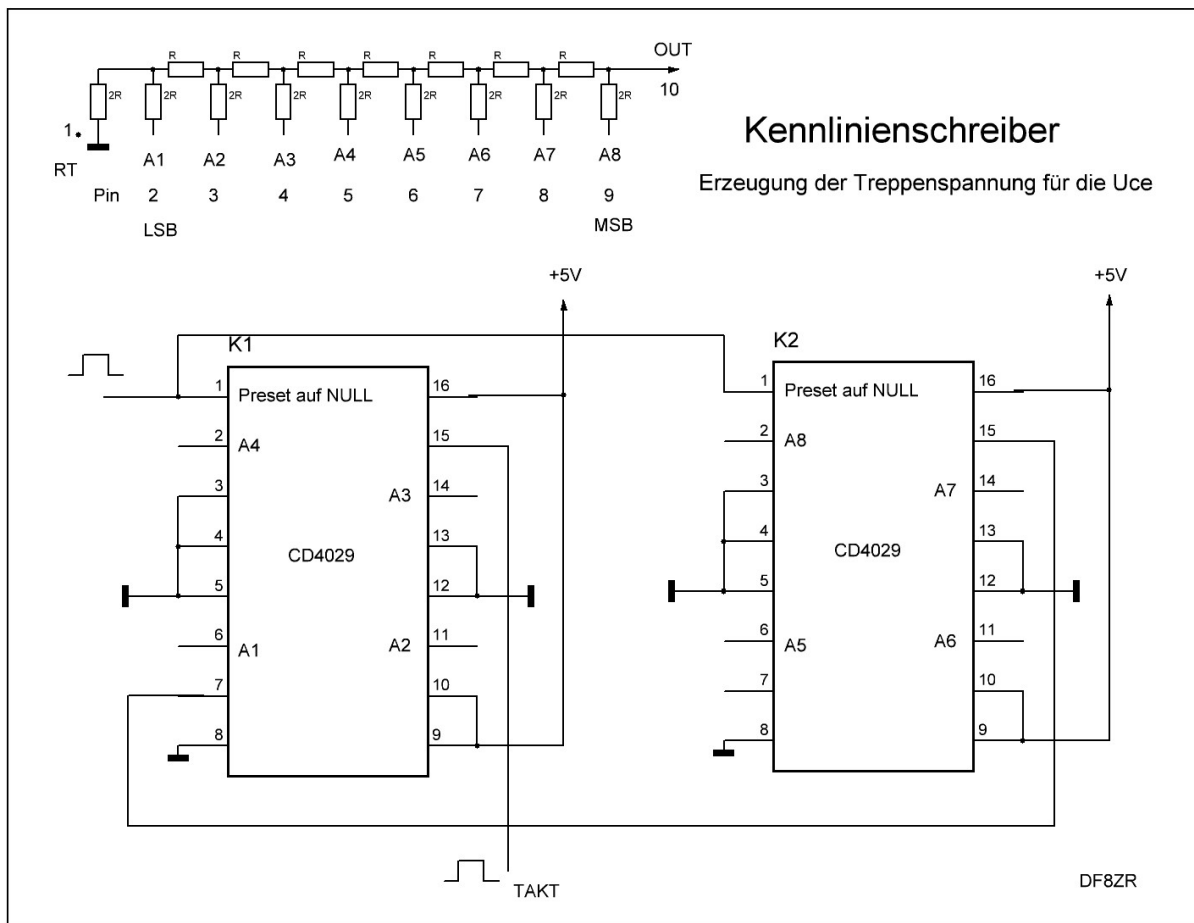
Taktfrequenz. Die Fotos zeigen den Verlauf am R2R-Netzwerk und der Ansteuerung mit zwei CD4029.



Es bleiben ca. 6ms für die Darstellung eines Messpunktes. Das sollte genügend Zeit sein für die Software. Mal sehen, wie sich das Display verhalten wird.



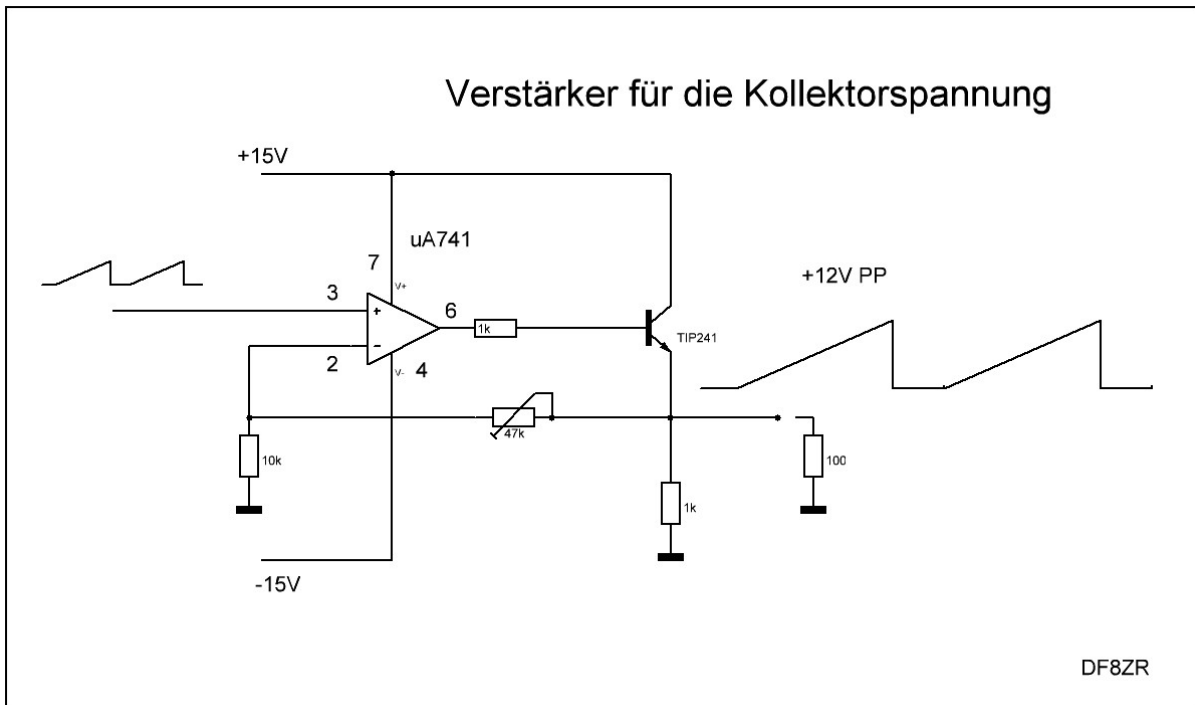
Und nachfolgend die Schaltung. Die Ausgangsspannung muss noch verstärkt werden.



Uce-Power

So, das wäre geschafft! Habe den Verstärker jetzt mit einem uA741 gebaut. Der verträgt max. +/- 20V DC. Die üblichen OPs sind da auf +/- 15V begrenzt. Nun haben wir etwas Reserve für einen sicheren Betrieb und könnten sogar +/- 18V wagen. Der Leistungstransistor wird gekühlt, denn er muss max. 1A fließen lassen. Die Ausgangsspannung verläuft jetzt von 0V bis +12V. Und da ja ein Frame bis max. 8 Kennlinien einmalig durchläuft, wird sich der Transistor wohl kaum bedenklich erwärmen. Jetzt muss ich nur noch den Zähler für die stufenweise Erhöhung basteln. Da er bis 8 zählen soll,

muss ich den CD4029 mit einem Gatter beschalten. Damit wird er bei > 8 durch einen positiven Reset wieder auf Null gesetzt. Mit hochohmigen Widerständen sollte sich ein „eingepprägter“ Basisstrom darstellen. Mit Trimmern wird der auf den jeweils gewünschten Festwert gestellt.



Die erste Kennlinie

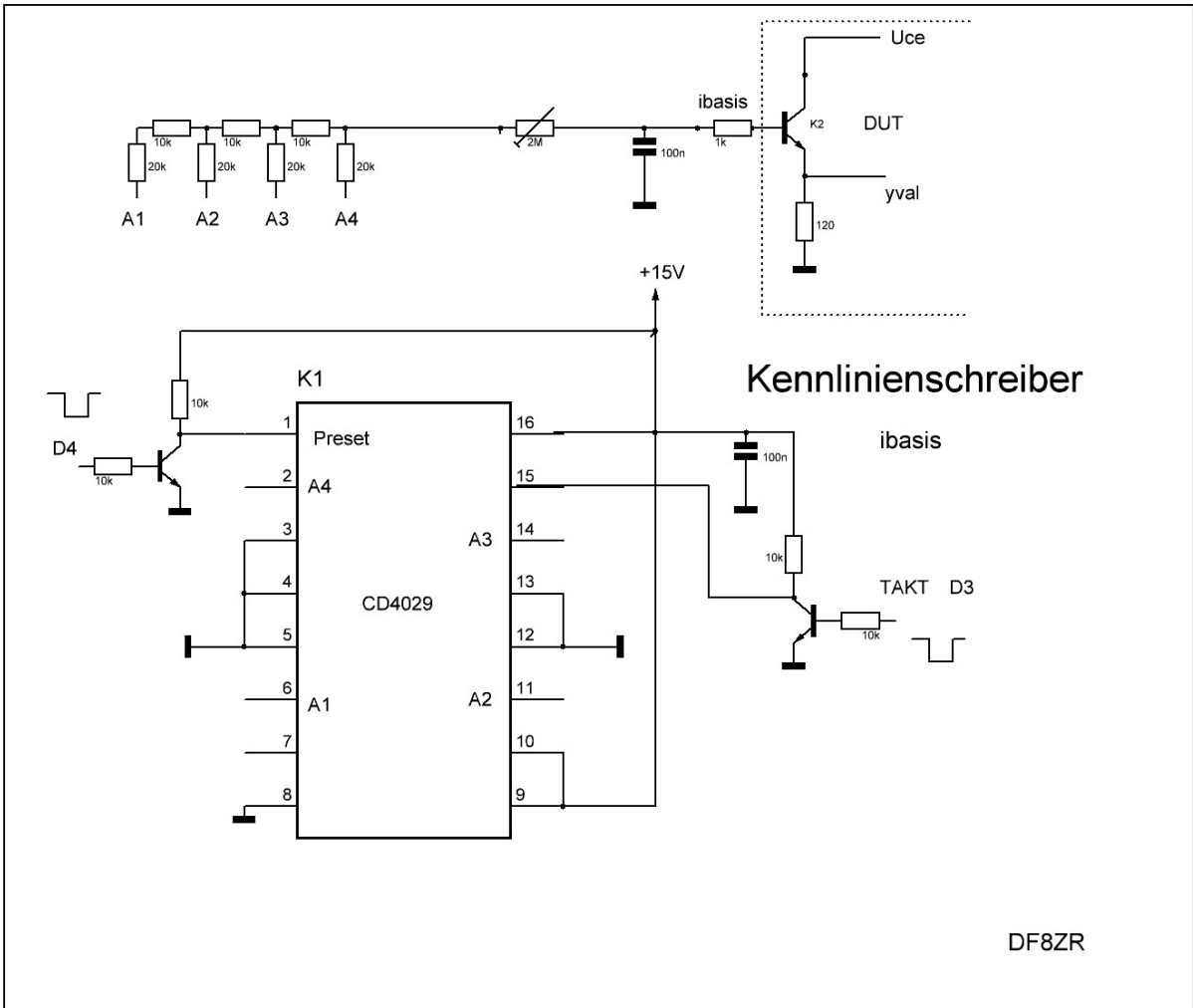
So, es war soweit:



Ein Schnappschuss. Und die Beschriftung muss noch korrigiert werden. Aber es werden die vollen +12V U_{ce} Spitzenspannung auf den Transistor geschaltet. Hier ein BC548C mit $\beta > 500$! Bin jetzt endlich mit der Darstellung zufrieden. Als Messwiderstand für den Kollektorstrom habe ich 120 Ohm eingesetzt. Dadurch wird der Ice mit 100mA begrenzt. Ich werde > 10 Stufen für den Basisstrom vorsehen. Dann können die oberen Stufen als „Spannungstreppen“ für die Gatespannungen an MOSFETs dienen. Und für die PNP werde ich einen OP zur Gewinnung von negativen Basisströmen verwenden und die Polarität am Messwiderstand ebenfalls mit einem OP umdrehen. Zum Schutz des Analogeingangs am Arduino werde ich eine Ableitdiode parallel schalten.

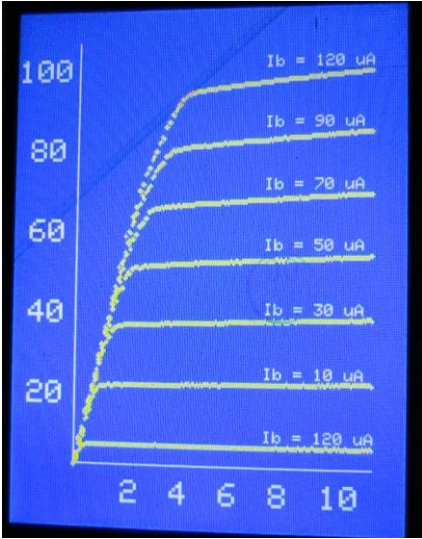
I-Basis erzeugen

Der mit 5V betriebene DAC gibt max. 2,5V heraus. Zu wenig für die max. Ansteuerung der Basis. Daher wurde der CD4029 jetzt mit +15V Betriebsspannung versorgt. Mit einem Trimpoti kann man im weiten Rahmen den richtigen Arbeitspunkt für die „Stromtreppe“ finden. Außerdem wird er die Gatespannungen in üblicher Höhe direkt abgeben können. Für PNP muss die Treppenspannung mit einem OP in der Polarität gedreht werden. Ebenso die abgefragte Spannung am Emitterwiderstand für die Stromwerte im Arduino. Alles etwas aufwendig, aber mit verständlicher Elektronik. Nachfolgend die Schaltung des DACs:



Es funktioniert

Hier das erste Frame:

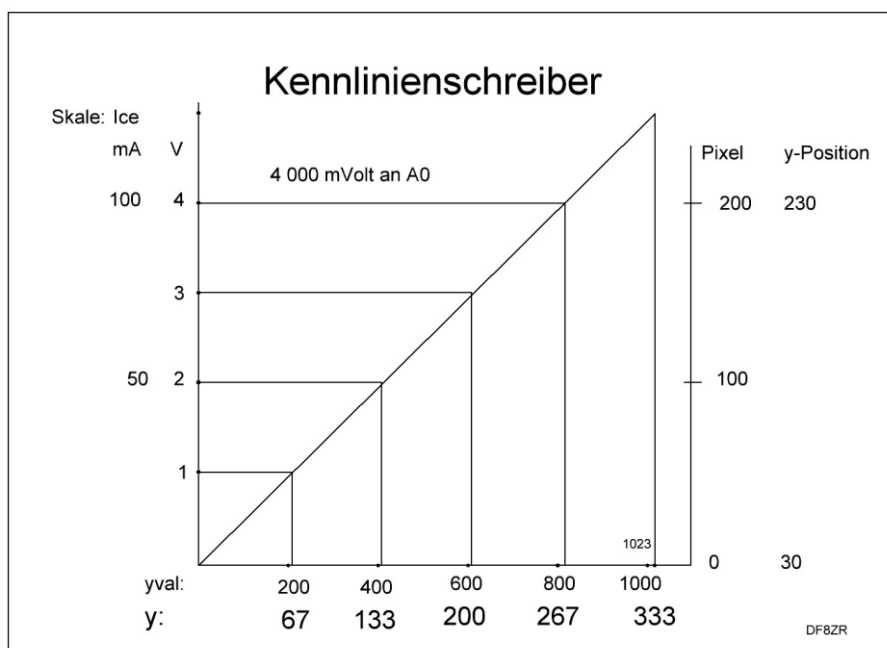


Wenn auch noch mit kleinen Fehlern. Der BC548C wurde gut angesteuert. Und da sind auch noch Reserven für einen Leistungstransistor. Jetzt kommt eigentlich nur noch Fleißarbeit auf mich zu, hi.

Messwerte an einem BC548C

$U_{ce} = +12V$; Emitterwiderstand = 120 Ohm

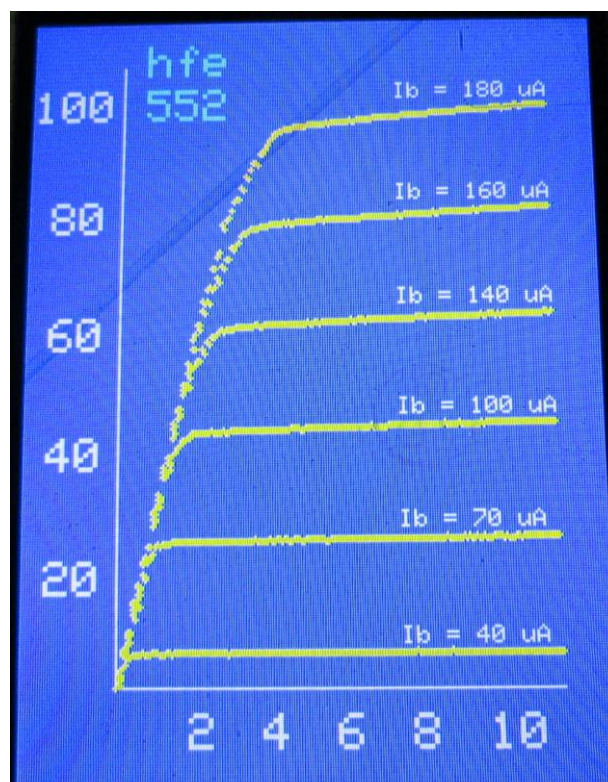
| Basis-Strom μA | U_e mV | an A0 mV | I_{ce} mA | yval |
|---------------------|----------|----------|-------------|------|
| 5 | 365 | 122 | 3,04 | 25 |
| 10 | 720 | 240 | 6,0 | 49 |
| 20 | 1420 | 473,3 | 11,83 | 97 |
| 30 | 2240 | 746,7 | 18,67 | 153 |
| 40 | 2670 | 890 | 22,25 | 182 |



Das Diagramm

ist ein Überblick über die Skalierung und der Programmierung der y-Achse. Diese beginnt bei $y = 30$ und endet bei $y = 230$. Sie ist also 200 Pixel lang für 100 mA. Unten sind die yval-Werte, die nach dem Auslesen der Spannung an A0 ermittelt werden. Die Proportionalitäten können zuweilen sehr unübersichtlich sein. Bei +12 V am Kollektor können 12V am Emitterwiderstand von 120 Ohm entstehen, wenn 100 mA fließen. Zum Schutz des Arduinos wird diese max. Spannung durch 3 geteilt. Dazu sind drei 1k in Reihe geschaltet. Es werden also max. +4V am Input anstehen. Die Auflösung mit 1023 Werten entspricht einem max. Inputpegel von + 5V. Wir nutzen nur den Bereich bis +4V. Diese Spannung entspricht einem Kollektorstrom von 100 mA.

Einen Schritt weiter:



So stelle ich mir die Anzeige vor. Der BC548C hatte tatsächlich diese hohe Verstärkung. Es wird bei der 3. Linie ($I_b = 100 \mu\text{A}$) der Ice gemessen. Bei Leistungstransistoren kann es sein, dass die oberen Linien nicht immer erreicht werden. Daher ist es besser, nicht zu hoch auszusteuern und mit niedrigeren Kollektorströmen den Verstärkungsfaktor zu bestimmen. Auf dem Display klebt noch die Schutzfolie.

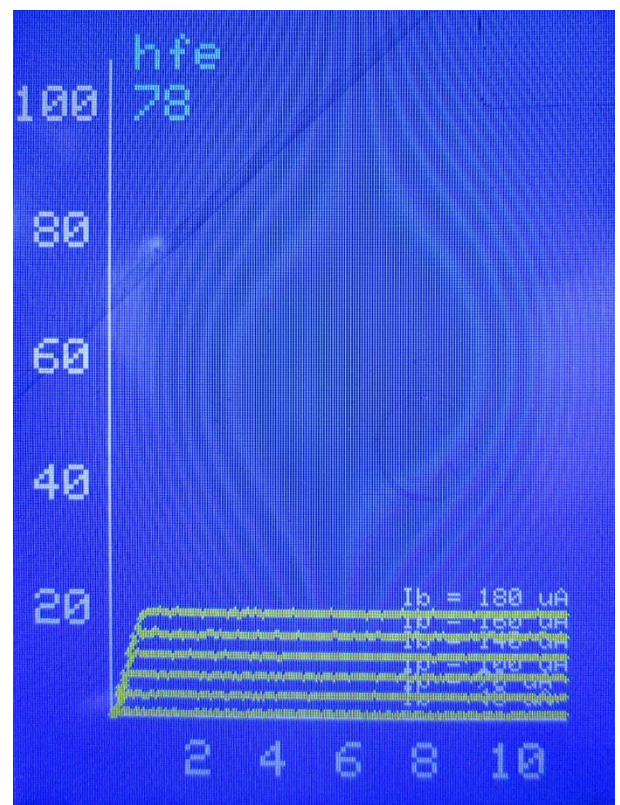
Und weil es Spaß macht

Hier noch einige Tests:





BSX 24C



BUV 48C

Weitere Planung

Für den Test eines NPN-Transistors habe ich eine Schaltung entwickelt. Damit kann man nur Kleinsignaltransistoren untersuchen. Für Leistungstransistoren ist sie ungeeignet, die grundsätzliche Funktion wird aber geprüft. Überhaupt brauchte ich in meinem langen Bastlerleben ganz selten mal einen Kennlinienschreiber. Es ist wie bei den Oszilloskopen. 90% aller Messungen habe ich mit einem Einkanal-Ozsi machen können. Selten brauchte ich einen Zweikanaler. Und fast nie einen Vierstrahler. Natürlich hängt es davon ab, auf welchem Gebiet der Elektronik die Schwerpunkte liegen. Der Aufwand für die vielfältigen Aufgaben bei der Untersuchung von Leistungstransistoren oder PNP-Transistoren bis hin zu

den MOSFETs ist enorm. Deshalb habe ich mich entschlossen, die vorliegende Schaltung weiter zu vereinfachen und nachbausicher zu machen. Wer mehr will, kann aus den folgenden Hinweisen Anregungen nehmen.

PNP-Transistoren

Sie brauchen negative Spannungen am Kollektor und an der Basis. Mit OPs lassen sich die Polaritäten wechseln. Und außerdem muss man dafür sorgen, dass der Arduino am A0 nur positive Spannungen erhält. Mit der LM358 wären die Anpassungen möglich. Der Schaltungsaufwand ist aber vergleichsweise enorm. Außerdem ist beim messen an Leistungstransistoren Vorsicht geboten. Es können Ströme von 1A fließen. Ohne Warnhinweis besteht die Gefahr, Kleinsignaltransistoren durch eine einzige Messung zu himmeln. Mit einer Warnleuchte(LED) kann man darauf aufmerksam machen.

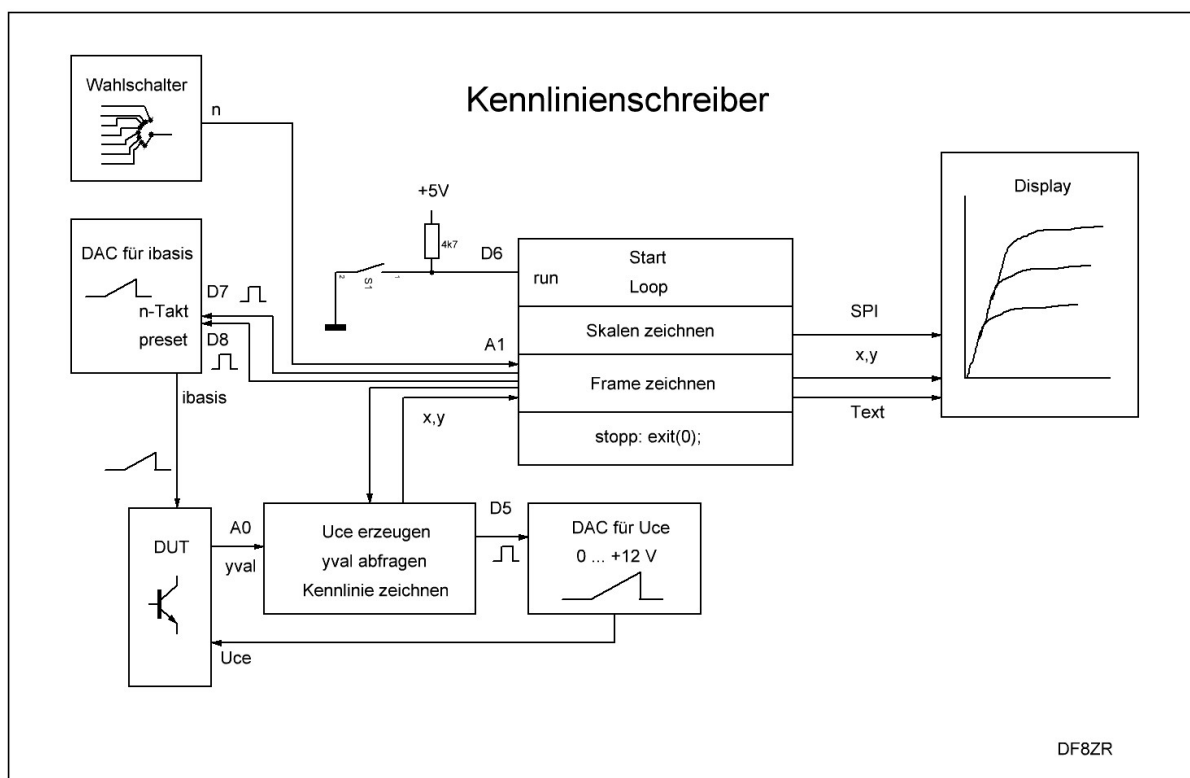
Schaltungsvorschläge

Ich habe mal die Schaltungen entworfen. Sie sind aber ungeprüft. Und man muss sicherlich Anpassungen vornehmen. Ich werde sie nicht realisieren. Wer ständig Leistungstransistoren aus dem HiFi-Bereich untersuchen oder vergleichen muss, ist auf Erweiterungen angewiesen. Dann lohnt sich auch der Aufwand. Z.B. könnte man zusätzlich den Kollektoranschluss schaltbar machen und über eine Buchse eine externe Spannung bis 60V anlegen, um den Durchbruch zu untersuchen. Das wurde schon beim ELV-Gerät so

Hat man keine Dreifach-Umschalter, kann man Drehschalter nehmen. Ein anderes Problem ist die Unterdrückung der Datenangabe bei Dioden. Hier gibt es kein hfe, wird aber falsch angezeigt. Man muss die Date ignorieren oder über einen Modenschalter abfragen, welcher Betrieb gewünscht ist. Das führt zu Problemen bei der begrenzten Verfügung von D-Eingängen am Arduino. Man muss dann Verfahren mit Analogspannungen an einem A-Port anwenden, um Variablen zu setzen. Z.B. Poti mit 30-Grad-Stellungen oder Stufenschalter. Alles viel Arbeit!

Funktionsübersicht

Hier nochmal eine Skizze zum Verständnis:



Und selbstverständlich die Software:

```
//tracer
#include <SPI.h>
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"
#include <Fonts/FreeMonoBoldOblique12pt7b.h>
#include <Fonts/FreeSerif9pt7b.h>

#define TFT_DC 9
#define TFT_CS 10 //10
#define TFT_RST 8
#define TFT_MISO 12 //wird nicht gebraucht, bleibt frei
#define TFT_MOSI 11
#define TFT_CLK 13

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK,
TFT_RST, TFT_MISO);

//Adafruit_ILI9341 tft(TFT_CS = 10);

// Beispielwert
uint16_t valueA = 512; // Halber Wert für DAC A
uint16_t valueB = 2047;//1023; // Maximaler Wert für DAC ; kommen 220mV
heraus

int i;

int ibeta;
int x1,x2,ibasis, nval;

int d, ic, y, x;
int n = 1; //Startbedingung für die Anzahl der Kennlinien in einem Frame
int xstep = 1;
long myTimer1 = 0;
long myTimer2 = 0;
long myTimeout = 1000;
long zeit = 0;
int yu;
long ymin, ymax,val,ywert, yval;
int icmax, ucmax, istep,ustep,ibstep;

void setup() {

  pinMode(6, INPUT_PULLUP); // Eingang mit Pullup-Widerstand
```

```

pinMode(2, INPUT_PULLUP);

pinMode(3, OUTPUT);//
pinMode(4, OUTPUT);// 7
pinMode(5, OUTPUT);// 8
pinMode(6, OUTPUT);// 9
pinMode(7, OUTPUT);// 10
pinMode(8, OUTPUT);// 11
pinMode(9, OUTPUT);//

//pinMode(A0, INPUT);//für y-wert
pinMode(A0, INPUT);
pinMode(A1, INPUT);//für
pinMode(A2, INPUT);//für
pinMode(A3, INPUT);//für

digitalWrite(6, LOW);

tft.begin();//Start des Displays

tft.fillScreen(ILI9341_BLACK);
tft.setCursor(24,80);
tft.setTextColor(ILI9341_YELLOW); tft.setTextSize(3);
tft.print("Halbleiter-");
tft.setCursor(58,110);
tft.print("Analyse");
tft.setTextColor(ILI9341_WHITE); tft.setTextSize(3);
tft.setTextSize(1);
tft.setCursor(88,151);
tft.print("(berndg42)");

Serial.begin(115200);

}

//*****
*****

/*
void laufzeit(){

zeit = myTimer2 -myTimer1;//bei myTimer1 war der Start des Programms
//Ermittlung der Laufzeit für eine Kennlinie

Serial.print("Zeit = ");
Serial.print(zeit);
Serial.println();

delay(1000);

```

```

//exit(0);
  //delay(100);//Dummy als Bearbeitungszeit fürs Schreiben auf das Display
}
*/

void skalen(void){

//tft.setRotation(0);
int x1=40; int y1 = 20; int x2 = 40;int y2 = 280;
tft.drawLine(x1, y1, x2, y2, ILI9341_WHITE);
delay(1);
x1=40; y1 = 280; x2 = 220; y2 = 280;
tft.drawLine(x1, y1, x2, y2, ILI9341_WHITE);
x = 10; y =300;
tft.setCursor(x,y);
tft.setTextColor(ILI9341_WHITE);  tft.setTextSize(2);

  tft.setTextSize(2);

// X-Achse beschriften; 180 Pixel entsprechen +12V, --> 30 = 2V
  tft.setCursor(70,290);
  tft.println("2");
  tft.setCursor(100,290);
  tft.println("4");
  tft.setCursor(130,290);
  tft.println("6");
  tft.setCursor(160,290);
  tft.println("8");
  tft.setCursor(190,290);
  tft.println("10");
  // tft.setCursor(210,290);
  // tft.println("12");
  /*
  tft.setCursor(180,290);
  tft.print("7");
  tft.setCursor(200,290);
  tft.print("8");
  tft.setCursor(220,290);
  tft.print("9");*/

//Y-Achse beschriften; 50 pixel = 20mA
  tft.setCursor(10,230);
  tft.println("20");
  tft.setCursor(10,180);
  tft.println("40");
  tft.setCursor(10,130);
  tft.println("60");
  tft.setCursor(10,80);

```

```

tft.println("80");
tft.setCursor(1,30);
tft.println("100");
}

//-----
//eine Kennlinie zeichnen

void trace(){

//an 120 Ohm wird gemessen; 100mA bei 12VUce erreicht ein
//Kleinleistungstransistor(BC548) seinen zulässigen Betriebsstrom;
//bei stärkeren Transistoren muss ein kleinerer Widerstand gewählt
werden(Umschalter)
//100mA > 240 pixel
//1mA > 2,4 pixel; y = (yval / 50) * 2.4; yval ist die Analogspannung an A0 in
mV
//tft.begin();//TEST nein!
// 445mV = 100mA
  x = 40; //Beginn im Ursprung; der liegt bei 40,40
//digitalWrite(5,LOW);//TEST

for ( i = 0; i <=180; i++) //Begrenzung in X-Richtung
{

//Kollektorspannung wird stufenweise erhöht; Treppenspannung bzw. wird hier
ein synchron laufender Sägezahn erzeugt

//Serial.println(x);

digitalWrite(5, HIGH);
delayMicroseconds(100);
digitalWrite(5,LOW);//TEST
delay(1);
yval = analogRead(A0);//Spannung Uce
//1023 bei +5V; max. stehen +4V am Spannungsteiler an ymax = 4000:5 = 800
// das entspricht einem Ice von 100mA; y wird mit 5mV aufgelöst
//es sind 280 Stufen auf der y-Achse; von 40 bis 320; 40 ist NULL = x-Achse

y = yval;

```

```

y =280 - y;//beginnt im Nullpunkt
tft.fillCircle(x,y,1, ILI9341_GREEN);//dauert 2s

x++;

} //Ende, der Zähler cd4029 muss wieder auf NULL gesetzt werden!!

digitalWrite(6, HIGH);//Preset auf 0000
delayMicroseconds(100);
digitalWrite(6, LOW);

delay(1);
beta();
//Serial.print(i); Serial.print(" ");
//Serial.println(yval);
}

//_____

void basisI(){

digitalWrite(3, HIGH);
digitalWrite(3,LOW);//einen Impuls herausgeben
delayMicroseconds(100);
digitalWrite(3, HIGH);

}

void beta(){

    int beta4;
    float y3, i3, faktor,ibasisf,hfe;

faktor = 100.0 / 268;

if (ibasis == 100) {y3 = yval;
y3 = y3 - 4;// Korrektur
ibasisf = (float) ibasis;

i3 = (float) (faktor * y3);
hfe = i3 / ibasisf;
hfe = 1000 * hfe;

//Serial.print(ibasis);Serial.print(" "); Serial.print(yval);
Serial.print(" ");Serial.println(hfe);
tft.setTextColor(ILI9341_CYAN);
tft.setTextSize(2);
tft.setCursor(50,10);
    tft.print("hfe");

```

```

    tft.setCursor(50,30);
    tft.print((int)hfe);
}

}

void frame(){
//es werden n Kennlinien in y-Richtung übereinander gezeichnet
digitalWrite(4, HIGH);
digitalWrite(4, LOW);//Preset des Basisstroms ; setzen auf NULL
delayMicroseconds(100);
digitalWrite(4, HIGH);

for(int i = 0; i <= (n-1); i++){

basisI(); //ibasis setzen; insgesamt n Impulse herausgeben; die Anzahl der
Impulse
//bestimmt den analogen Wert für die Ausgangsspannung des DAC und damit den
Basisstrom
//mit jedem Impuls steigt der Basisstrom
//vor dem Frame wird der DAC auf NULL gesetzt
//Serial.println("H1");
trace(); //die Kennlinie zeichnen

//Beschriftung der Kennlinie mit basis

    if(i == 5) ibasis = 180;
    if(i == 4) ibasis = 160;//145
    if(i == 3){ ibasis = 140; ibeta = i;}
    if(i == 2){ ibasis = 100; ibeta = i;}
    if(i == 1) ibasis = 70;
    if (i == 0)ibasis = 40;

x = x - 65;
y = y - 10;
tft.setCursor(x,y);
tft.setTextColor(ILI9341_WHITE); tft.setTextSize(1);
tft.print("Ib = ");

tft.print(ibasis);
tft.print(" uA");
    tft.setTextSize(2);

}

digitalWrite(4, LOW);//Reset des Basisstroms ; setzen auf NULL
delayMicroseconds(100);//ansonsten könnte der Transistor heiß werden

```

```

digitalWrite(4, HIGH);
}

void run(){

tft.fillScreen(ILI9341_BLACK);//Clear des Displays
  delay(100);
skalen();//nach dem Frame und dem Clear des Displays
y=30;
frame();//Anzahl n Kurven zeichnen = ein Frame
delay(100);

}

//*****
*****

void loop() {

n = 6;//Anzahl der Kurven

if(digitalRead(2) == LOW) {
delay(100);
if(digitalRead(2) == LOW) run();
}

//Serial.print("on");

}

```

Bis hier soll es nun genug sein. Ich werde die einfache Ausführung wahrscheinlich „NPN-Analyzer“ nennen und darüber einen Film bei YT einstellen und einen expliziten Nachbaubericht schreiben.

DF8ZR; 32.05.2024